Data Cleaning

Ihab F. Ilyas
Xu Chu

ASSOCIATION FOR COMPUTING MACHINERY

Data Cleaning

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

ACM Books

Editor in Chief

M. Tamer Özsu, *University of Waterloo*

ACM Books is a series of high-quality books for the computer science community, published by ACM and many in collaboration with Morgan & Claypool Publishers. ACM Books publications are widely distributed in both print and digital formats through booksellers and to libraries (and library consortia) and individual ACM members via the ACM Digital Library platform.

Data Cleaning

Ihab F. Ilyas, *University of Waterloo*
Xu Chu, *Georgia Institute of Technology*
2019

Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework

Robert J. Moore, *IBM Research–Almaden*
Raphael Arar, *IBM Research–Almaden*
2019

Heterogeneous Computing: Hardware and Software Perspectives

Mohamed Zahran, *New York University*
2019

Hardness of Approximation Between P and NP

Aviad Rubinstein, *Stanford University*
2019

Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker

Editor: Michael L. Brodie, *Massachusetts Institute of Technology*
2018

The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition

Editors: Sharon Oviatt, *Monash University*
Björn Schuller, *University of Augsburg and Imperial College London*
Philip R. Cohen, *Monash University*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Krüger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2018

RIGHTSLINK

Declarative Logic Programming: Theory, Systems, and Applications
Editors: Michael Kifer, *Stony Brook University*
Yanhong Annie Liu, *Stony Brook University*
2018

The Sparse Fourier Transform: Theory and Practice
Haitham Hassanieh, *University of Illinois at Urbana-Champaign*
2018

The Continuing Arms Race: Code-Reuse Attacks and Defenses
Editors: Per Larsen, *Immunant, Inc.*
Ahmad-Reza Sadeghi, *Technische Universität Darmstadt*
2018

Frontiers of Multimedia Research
Editor: Shih-Fu Chang, *Columbia University*
2018

Shared-Memory Parallelism Can Be Simple, Fast, and Scalable
Julian Shun, *University of California, Berkeley*
2017

Computational Prediction of Protein Complexes from Protein Interaction Networks
Sriganesh Srihari, *The University of Queensland Institute for Molecular Bioscience*
Chern Han Yong, *Duke-National University of Singapore Medical School*
Limsoon Wong, *National University of Singapore*
2017

The Handbook of Multimodal-Multisensor Interfaces, Volume 1: Foundations, User Modeling, and Common Modality Combinations
Editors: Sharon Oviatt, *Incaa Designs*
Björn Schuller, *University of Passau and Imperial College London*
Philip R. Cohen, *Voicebox Technologies*
Daniel Sonntag, *German Research Center for Artificial Intelligence (DFKI)*
Gerasimos Potamianos, *University of Thessaly*
Antonio Krüger, *Saarland University and German Research Center for Artificial Intelligence (DFKI)*
2017

Communities of Computing: Computer Science and Society in the ACM
Thomas J. Misa, Editor, *University of Minnesota*
2017

Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining
ChengXiang Zhai, *University of Illinois at Urbana–Champaign*
Sean Massung, *University of Illinois at Urbana–Champaign*
2016

RIGHTSLINK

An Architecture for Fast and General Data Processing on Large Clusters
Matei Zaharia, *Stanford University*
2016

Reactive Internet Programming: State Chart XML in Action
Franck Barbier, *University of Pau, France*
2016

Verified Functional Programming in Agda
Aaron Stump, *The University of Iowa*
2016

The VR Book: Human-Centered Design for Virtual Reality
Jason Jerald, *NextGen Interactions*
2016

Ada's Legacy: Cultures of Computing from the Victorian to the Digital Age
Robin Hammerman, *Stevens Institute of Technology*
Andrew L. Russell, *Stevens Institute of Technology*
2016

Edmund Berkeley and the Social Responsibility of Computer Professionals
Bernadette Longo, *New Jersey Institute of Technology*
2015

Candidate Multilinear Maps
Sanjam Garg, *University of California, Berkeley*
2015

Smarter Than Their Machines: Oral Histories of Pioneers in Interactive Computing
John Cullinane, *Northeastern University; Mossavar-Rahmani Center for Business and Government, John F. Kennedy School of Government, Harvard University*
2015

A Framework for Scientific Discovery through Video Games
Seth Cooper, *University of Washington*
2014

Trust Extension as a Mechanism for Secure Code Execution on Commodity Computers
Bryan Jeffrey Parno, *Microsoft Research*
2014

Embracing Interference in Wireless Systems
Shyamnath Gollakota, *University of Washington*
2014

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

Data Cleaning

Ihab F. Ilyas

University of Waterloo

Xu Chu

Georgia Institute of Technology

ACM Books #28

RIGHTSLINK

Copyright © 2019 by Association for Computing Machinery

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews—without the prior permission of the publisher.

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which the Association for Computing Machinery is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Data Cleaning
Ihab F. Ilyas
Xu Chu

books.acm.org
http://books.acm.org

ISBN: 978-1-4503-7152-0 hardcover
ISBN: 978-1-4503-7153-7 paperback
ISBN: 978-1-4503-7154-4 ePub
ISBN: 978-1-4503-7155-1 eBook

Series ISSN: 2374-6769 print 2374-6777 electronic

DOIs:

10.1145/3310205 Book
10.1145/3310205.3310206 Preface
10.1145/3310205.3310207 Chapter 1
10.1145/3310205.3310208 Chapter 2
10.1145/3310205.3310209 Chapter 3
10.1145/3310205.3310210 Chapter 4
10.1145/3310205.3310211 Chapter 5
10.1145/3310205.3310212 Chapter 6
10.1145/3310205.3310213 Chapter 7
10.1145/3310205.3310214 Chapter 8
10.1145/3310205.3310215 References/Index/Bios

A publication in the ACM Books series, #28
Editor in Chief: M. Tamer Özsu, *University of Waterloo*

This book was typeset in Arnhem Pro 10/14 and Flama using ZzTEX.
Cover photo: Jason Dorfman MIT / CSAIL

First Edition

10 9 8 7 6 5 4 3 2 1

RIGHTSLINK

To my family: Francis, Aida, Mirette, Andrew and Marina

To my wife Jianmei and my daughter Hannah

RIGHTSLINK
Copyright Clearance Center

RIGHTSLINK
Copyright Clearance Center

Contents

Preface xiii

Figure and Table Credits xv

Chapter 1 **Introduction** 1

1.1 Data Cleaning Workflow 3
1.2 Book Scope 4

Chapter 2 **Outlier Detection** 11

2.1 A Taxonomy of Outlier Detection Methods 12
2.2 Statistics-Based Outlier Detection 15
2.3 Distance-Based Outlier Detection 26
2.4 Model-Based Outlier Detection 30
2.5 Outlier Detection in High-Dimensional Data 32
2.6 Conclusion 44

Chapter 3 **Data Deduplication** 47

3.1 Similarity Metrics 49
3.2 Predicting Duplicate Pairs 54
3.3 Clustering 57
3.4 Blocking for Deduplication 60
3.5 Distributed Data Deduplication 66
3.6 Record Fusion and Entity Consolidation 73
3.7 Human-Involved Data Deduplication 81
3.8 Data Deduplication Tools 85
3.9 Conclusion 88

Chapter 4 **Data Transformation** 91

4.1 Syntactic Data Transformations 93

RIGHTSLINK

4.2 Semantic Data Transformations 107
4.3 ETL Tools 117
4.4 Conclusion 118

Chapter 5 **Data Quality Rule Definition and Discovery** 121
5.1 Functional Dependencies 124
5.2 Conditional Functional Dependencies 130
5.3 Denial Constraints 133
5.4 Other Types of Constraints 138
5.5 Conclusion 147

Chapter 6 **Rule-Based Data Cleaning** 149
6.1 Violation Detection 149
6.2 Error Repair 161
6.3 Conclusion 193

Chapter 7 **Machine Learning and Probabilistic Data Cleaning** 195
7.1 Machine Learning for Data Deduplication 196
7.2 Machine Learning for Data Repair 203
7.3 Data Cleaning for Analytics and Machine Learning 214

Chapter 8 **Conclusion and Future Thoughts** 223

References 227
Index 247
Author Biographies 259

RIGHTSLINK

Preface

Data quality is one of the most important problems in data management, since dirty data often leads to inaccurate data analytics results and incorrect business decisions. Poor data across businesses and the U.S. government are reported to cost trillions of dollars a year. Multiple surveys show that dirty data is the most common barrier faced by data scientists. Not surprisingly, developing effective and efficient data cleaning solutions is challenging and is rife with deep theoretical and engineering problems.

Data cleaning is used to refer to all kinds of tasks and activities to detect and repair errors in the data. Rather than focus on a particular data cleaning task, in this book, we give an overview of the end-to-end data cleaning process, describing various error detection and repair methods, and attempt to anchor these proposals with multiple taxonomies and views. Specifically, we cover four of the most common and important data cleaning tasks, namely, outlier detection, data transformation, error repair (including imputing missing values), and data deduplication. Furthermore, due to the increasing popularity and applicability of machine learning techniques, we include a chapter that specifically explores how machine learning techniques are used for data cleaning, and how data cleaning is used to improve machine learning models.

This book is intended to serve as a useful reference for researchers and practitioners who are interested in the area of data quality and data cleaning. It can also be used as a textbook for a graduate course. Although we aim at covering state-of-the-art algorithms and techniques, we recognize that data cleaning is still an active field of research and therefore provide future directions of research whenever appropriate.

Ihab Ilyas
Xu Chu
March 2019

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

Figure and Table Credits

Figures

Figure 2.3 Based On: Patrick Wessa. Free statistics software, office for research development and education, version 1.1. 23-r7. http://www.wessa.net, 2012

Figure 2.4 Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. *SIGMOD Rec.* 29, 2 (May 2000), 93–104. DOI: 10.1145/335191.335388.

Figure 2.5 Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages. DOI: 10.1145/1541880.1541882.

Figure 2.6 Charu C. Aggarwal. *Outlier Analysis*. Springer, 2013.

Figure 2.7 Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection. *IEEE Trans. Knowl. and Data Eng.,* 19(5), 2007.

Figure 3.3 Based On: Jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12).* ACM, New York, NY, USA, 85–96. DOI: 10.1145/2213836.2213847.

Figure 3.6 Jens Bleiholder and Felix Naumann. 2009. Data fusion. *ACM Comput. Surv.* 41, 1, Article 1 (January 2009), 41 pages. DOI: 10.1145/1456650.1456651.

Figure 3.7 Based On: George Beskales, Mohamed A. Soliman, Ihab F. Ilyas, and Shai Ben-David. Modeling and querying possible repairs in duplicate detection. *Proc. VLDB Endowment*, 2(1): 598–609, (August 2009), 598–609. DOI: 10.14778/1687627.1687695.

Figure 3.8 Based On: George Beskales, Mohamed A. Soliman, Ihab F. Ilyas, and Shai Ben-David. Modeling and querying possible repairs in duplicate detection. *Proc. VLDB Endowment*, 2(1): 598–609, (August 2009), 598–609. DOI: 10.14778/1687627.1687695.

Figure 3.11 Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proc. VLDB Endowment*, 5(11): 1483–1494, DOI: 10.14778/2350229.2350263.

Figure 3.12 Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proc. VLDB Endowment*, 5(11): 1483–1494, DOI: 10.14778/2350229.2350263.

RIGHTSLINK

Figure 3.13 Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14).* ACM, New York, NY, USA, 601–612. DOI: 10.1145/2588555.2588576.

Figure 3.14 Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. Magellan: Toward building entity matching management systems. *Proc. VLDB Endowment,* 9(12): 1197–1208, 2016.

Figure 3.15 Based on: Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, George Beskales, Mitch Cherniack, Stanley B. Zdonik, Alexander Pagan, and Shan Xu. Data curation at scale: The data tamer system. In *Proc. 6th Biennial Conf. on Innovative Data Systems Research, 2013.* http://cidrdb.org/

Figure 4.3 Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01),* Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard Thomas Snodgrass (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 381–390.

Figure 4.4 Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11).* ACM, New York, NY, USA, 65–74. DOI: 10.1145/2047196.2047205. and Jeffrey Heer, Joseph Hellerstein, and Sean Kandel. Predictive interaction for data transformation. In *Proc. 7th Biennial Conf. on Innovative Data Systems Research*, 2015. and Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11).* ACM, New York, NY, USA, 3363–3372. DOI: 10.1145/1978942.1979444.

Figure 4.5 Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/, (http://fsf.org/)

Figure 4.6 Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '11).* ACM, New York, NY, USA, 317–330. DOI: 10.1145/1926385.1926423.

Figure 4.7 Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11).* ACM, New York, NY, USA, 65–74. DOI: 10.1145/2047196.2047205.

Figure 4.8 Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti and M. Stonebraker, DataXFormer: A robust transformation discovery system, *2016 IEEE 32nd International Conference on Data Engineering (ICDE),* Helsinki, 2016, pp. 1134–1145. DOI: 10.1109/ICDE.2016.7498319.

Figure 4.9 Based On: Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti and M. Stonebraker, DataXFormer: A robust transformation discovery system, *2016 IEEE 32nd International Conference on Data Engineering (ICDE),* Helsinki, 2016, pp. 1134–1145. DOI: 10.1109/ICDE.2016.7498319.

RIGHTSLINK

Figure 4.10 Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti and M. Stonebraker, DataXFormer: A robust transformation discovery system, *2016 IEEE 32nd International Conference on Data Engineering (ICDE),* Helsinki, 2016, pp. 1134–1145. DOI: 10.1109/ICDE.2016.7498319.

Figure 4.11 Based On: Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti and M. Stonebraker, DataXFormer: A robust transformation discovery system, *2016 IEEE 32nd International Conference on Data Engineering (ICDE),* Helsinki, 2016, pp. 1134–1145. DOI: 10.1109/ICDE.2016.7498319.

Figure 5.3 Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16).* ACM, New York, NY, USA, 821–833. DOI: 10.1145/2882903.2915203.

Figure 5.5 Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with hydra. *Proc. VLDB Endow.* 11, 3 (November 2017), 311–323. DOI: 10.14778/3157794.3157800.

Figure 5.6 Grace Fan, Wenfei Fan, and Floris Geerts. Detecting errors in numeric attributes. In *Proc. 15th Int. Conf. on Web-Age Information Management*, pages 125–137. Springer, 2014a.

Figure 5.7 Jiannan Wang and Nan Tang. 2014. Towards dependable data repairing with fixing rules. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14).* ACM, New York, NY, USA, 457–468. DOI: 10.1145/2588555.2610494.

Figure 5.8 Matteo Interlandi and Nan Tang. Proof positive and negative in data cleaning. In *Proc. 31st Int. Conf. on Data Engineering*, 2015.

Figure 5.9 Matteo Interlandi and Nan Tang. Proof positive and negative in data cleaning. In *Proc. 31st Int. Conf. on Data Engineering*, 2015.

Figure 5.10 Matteo Interlandi and Nan Tang. Proof positive and negative in data cleaning. In *Proc. 31st Int. Conf. on Data Engineering*, 2015.

Figure 6.2 Based On: Anup Chalamalla, Ihab F. Ilyas, Mourad Ouzzani, and Paolo Papotti. Descriptive and prescriptive data cleaning. In *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pages 445–456, 2014. DOI: 10.1145/2588555.2610520.

Figure 6.3 Alexandra Meliou, Wolfgang Gatterbauer, Suman Nath, and Dan Suciu. 2011. Tracing data errors with view-conditioned causality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11).* ACM, New York, NY, USA, 505–516. DOI: 10.1145/1989323.1989376.

Figure 6.4 Eugene Wu and Samuel Madden. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment,* Vol. 6, No. 8. Copyright 2013 VLDB Endowment 2150-8097/13/06 553–564.

Figure 6.5 Anup Chalamalla, Ihab F. Ilyas, Mourad Ouzzani, and Paolo Papotti. Descriptive and prescriptive data cleaning. In *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pages 445–456, 2014. DOI: 10.1145/2588555.2610520.

Figure 6.6 Anup Chalamalla, Ihab F. Ilyas, Mourad Ouzzani, and Paolo Papotti. Descriptive and prescriptive data cleaning. In *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pages 445–456, 2014. DOI: 10.1145/2588555.2610520.

RIGHTSLINK

Figure 6.7 Based On: Anup Chalamalla, Ihab F. Ilyas, Mourad Ouzzani, and Paolo Papotti. Descriptive and prescriptive data cleaning. In *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pages 445–456, 2014. DOI: 10.1145/2588555.2610520.

Figure 6.9 Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. That's all folks! LLUNATIC goes open source. *Proceedings of the VLDB Endowment,* Vol. 7, No. 13. Copyright 2014 VLDB Endowment 2150-8097/14/08:1565–1568.

Figure 6.12 Maksims Volkovs, Fei Chiang, Jaroslaw Szlichta, and Rene'e J. Miller. Continuous data cleaning. In *Proc. 30th Int. Conf. on Data Engineering,* pages 244–255, 2014.

Figure 6.14 George Beskales, Ihab F. Ilyas, and Lukasz Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proc. VLDB Endowment*, 3(1–2): 197–207, DOI: 10.14778/1920841.1920870.

Figure 6.15 Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory (ICDT '09),* Ronald Fagin (Ed.). ACM, New York, NY, USA, 53–62. DOI: 10.1145/1514894.1514901.

Figure 6.16 Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. Guided data repair. *Proc. VLDB Endowment,* 4(5): 279–289, DOI: 10.14778/1952376.1952378.

Figure 6.17 Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. Guided data repair. *Proc. VLDB Endowment,* 4(5): 279–289, DOI: 10.14778/1952376.1952378.

Figure 6.18 Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management.* Synthesis Lectures on Data Management. 2012. © Morgan & Claypool.

Figure 6.19 Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15).* ACM, New York, NY, USA, 1247–1261. DOI: 10.1145/2723372.2749431.

Figure 6.20 Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15).* ACM, New York, NY, USA, 1247–1261. DOI: 10.1145/2723372.2749431.

Figure 6.23 George Beskales, Ihab F. Ilyas, and Lukasz Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proc. VLDB Endowment*, 3(1–2): 197–207, DOI: 10.14778/1920841.1920870.

Figure 7.1 Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '02).* ACM, New York, NY, USA, 269–278. DOI: 10.1145/775047.775087.

Figure 7.2 Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for

RIGHTSLINK

entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18).* ACM, New York, NY, USA, 19–34. DOI: 10.1145/3183713.3196926.

Figure 7.3 Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18).* ACM, New York, NY, USA, 19–34. DOI: 10.1145/3183713.3196926.

Figure 7.8 Jiannan Wang, Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pages 469–480, 2014. DOI: 10.1145/2588555.2610505.

Figure 7.9 Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endowment*, 9(12, August 2016): 948–959. DOI: 10.14778/2994509.2994514.

Tables

Table 3.2 Jens Bleiholder and Felix Naumann. 2009. Data fusion. ACM Comput. Surv. 41, 1, Article 1 (January 2009), 41 pages. DOI: 10.1145/1456650.1456651 and Xin Luna Dong and Felix Naumann. Data fusion: resolving data conflicts for integration. *Proc. VLDB Endowment*, 2(2): 1654–1655, 2009.

Table 4.1 Based On: Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11).* ACM, New York, NY, USA, 3363–3372. DOI: 10.1145/1978942.1979444.

Table 5.2 Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. On generating near-optimal tableaux for conditional functional dependencies. *Proc. VLDB Endowment,* 1(1): 376–390, DOI: 10.14778/1453856.1453900.

Table 6.1 Based On: Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *Proc. 29th Int. Conf. on Data Engineering,* pages 458–469, 2013b.

Table 6.3 Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2011. Interaction between record matching and data repairing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11).* ACM, New York, NY, USA, 469–480. DOI: 10.1145/1989323.1989373.

Table 7.1 Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: holistic data repairs with probabilistic inference. *Proc. VLDB Endow.* 10, 11 (August 2017), 1190–1201. DOI: 10.14778/3137628.3137631.

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

1 Introduction

Enterprises have been acquiring large amounts of data from a variety of sources in order to build large data repositories that power their applications, with the goal of enabling richer and more informed analytics. Data collection and acquisition often introduce errors in data, e.g., missing values, typos, mixed formats, replicated entries for the same real-world entity, and violations of business and data integrity rules. A survey about the state of data science and machine learning (ML) reveals that dirty data is the most common barrier faced by workers dealing with data.[1] With the popularity of data science, it has become increasingly evident that data curation, unification, preparation, and cleaning are key enablers in unleashing the value of data.[2] According to another survey of about 80 data scientists conducted by CrowdFlower and published in Forbes,[3] data scientists spend more than 60% of their time in cleaning and organizing data, and 57% of data scientists regard cleaning and organizing data as the least enjoyable part of their work. Not surprisingly, developing effective and efficient data cleaning solutions is challenging and is rife with deep theoretical and engineering problems.

Regardless of the type of data errors to be fixed, data cleaning activities usually consist of two phases: (1) error detection, where various errors and violations are identified and possibly validated by experts; and (2) error repair, where updates to the database are applied (or suggested to human experts) to bring the data to a cleaner state suitable for downstream applications and analytics. Error detection techniques can be either quantitative or qualitative. Specifically, quantitative error detection techniques often involve statistical methods to identify abnormal behaviors and errors [Hellerstein 2008] (e.g., "*a salary that is three standard deviations*

1. https://www.kaggle.com/surveys/2017

2. https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html

3. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/

RIGHTSLINK

away from the mean salary is an error"), and hence have been mostly studied in the context of outlier detection [Aggarwal 2013]. On the other hand, qualitative error detection techniques rely on descriptive approaches to specify patterns or constraints of a consistent data instance, and for that reason these techniques identify those data that violate such patterns or constraints as errors. For example, in a descriptive statement about a company HR database, "*for two employees working at the same branch of the company, the senior employee cannot earn less salary than the junior employee*," if we find two employees with a violation of the rule, it is likely that there is an error in at least one of them.

Various surveys and books detail specific aspects of data quality and data cleaning. For example, Rahm and Do [2000] classify different types of errors occurring in an Extract-Transform-Load (ETL) process, and survey the tools available for cleaning data in an ETL process. Some work focuses on the effect of incompleteness data on query answering [Grahne 1991] and the use of a Chase procedure [Maier et al. 1979] for dealing with incomplete data [Greco et al. 2012]. Hellerstein [2008] focuses on cleaning quantitative numerical data using mainly statistical techniques. Bertossi [2011] provides complexity results for repairing inconsistent data and performing consistent query answering on inconsistent data. Fan and Geerts [2012] discuss the use of data quality rules in data consistency, data currency, and data completeness, and their interactions. Dasu and Johnson [2003] summarize how techniques in exploratory data mining can be integrated with data quality management. Ganti and Sarma [2013] focus on an operator-centric approach for developing a data cleaning solution, involving the development of customizable operators that can be used as building blocks for developing common solutions. Ilyas and Chu [2015] provide taxonomies and example algorithms for qualitative error detection and repairing techniques. Multiple surveys and tutorials have been published to summarize different definitions of outliers and the algorithms for detecting them [Hodge and Austin 2004, Chandola et al. 2009, Aggarwal 2013]. Data deduplication, a long-standing problem that has been studied for decades [Fellegi and Sunter 1969], has also been extensively surveyed [Koudas et al. 2006, Elmagarmid et al. 2007, Herzog et al. 2007, Dong and Naumann 2009, Naumann and Herschel 2010, Getoor and Machanavajjhala 2012].

This book, however, focuses on the end-to-end data cleaning process, describing various error detection and repair methods, and attempts to anchor these proposals with multiple taxonomies and views. Our goals are (1) to allow researchers and general readers to understand the scope of current techniques and highlight gaps and possible new directions of research; and (2) to give practitioners and system implementers a variety of choices and solutions for their data cleaning activities.

RIGHTSLINK

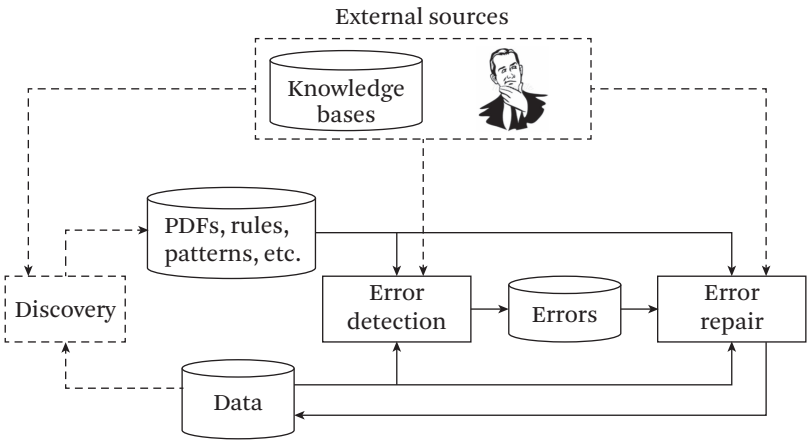


Figure 1.1 A typical data cleaning workflow with an optional discovery step, error detection step, and error repair step.

In what follows, we give a brief overview of the book's scope as well as a chapter outline.

1.1 Data Cleaning Workflow

Figure 1.1 shows a typical data cleaning workflow, consisting of an optional discovery and profiling step, an error detection step, and an error repair step. To clean a dirty dataset, we often need to model various aspects of this data, e.g., schema, patterns, probability distributions, and other metadata. One way to obtain such metadata is by consulting domain experts, typically a costly and time-consuming process. The discovery and profiling step is used to discover these metadata automatically. Given a dirty dataset and the associated metadata, the error detection step finds part of the data that does not conform to the metadata, and declares this subset to contain errors. The errors surfaced by the error detection step can be in various forms, such as outliers, violations, and duplicates. Finally, given the errors detected and the metadata that generate those errors, the error repair step produces data updates that are applied to the dirty dataset. Since there are many uncertainties in the data cleaning process, external sources such as knowledge bases and human experts are consulted whenever possible and feasible to ensure the accuracy of the cleaning workflow.

Example 1.1 Consider Table 1.1 containing employee records for a U.S. company. Every tuple specifies a person in a company with her id (GID), name (FN, LN), level (LVL),

RIGHTSLINK

Table 1.1 An example employee table

TID	FN	LN	LVL	ZIP	ST	SAL
t_1	Anne	Nash	5	10001	NM	110,000
t_2	Mark	White	6	87101	NM	80,000
t_3	Jane	Lee	4	10001	NY	75,000

zip code (ZIP), state (ST), and salary (SAL). Suppose a domain expert supplies two data quality rules for this table. The first rule states that if two employees have the same zip code, they must be in the same state. The second rule states that among employees working in the same state, a senior employee cannot earn a smaller salary than a junior employee.

Given these two data quality rules, the error detection step detects two violations. The first violation consists of four cells $\{t_1[ZIP], t_1[ST], t_3[ZIP], t_3[ST]\}$, which together violate the first data quality rule. The second violation consists of six cells $\{t_1[ROLE], t_1[ST], t_1[SAL], t_2[ROLE], t_2[ST], t_2[SAL]\}$, which together violate the second data quality rule. The data repair step takes the violations and produces an update that changes $t_1[ST]$ from "NM" to "NY", and the new data now has no violation with respect to the two rules.

1.2 Book Scope

The aforementioned data cleaning workflow describes a general purpose data cleaning process, but there are different data cleaning topics that address one or multiple steps in the workflow. We cover some of the most common and practical cleaning topics in this book: outlier detection, data deduplication, data transformation, rule-based data cleaning, ML guided cleaning, and human involved data cleaning. We briefly explain these topics in the following subsections; we also highlight the book structure in Section 1.2.7.

1.2.1 Outlier Detection

Outlier detection refers to detecting "outlying" values. While an exact definition of an outlier depends on the application, there are some commonly used definitions, such as "*an outlier is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism*" [Hawkins 1980] and "*an outlier observation is one that appears to deviate markedly from other members of the sample in which it occurs*" [Barnett and Lewis 1994]. For example,

RIGHTSLINK

for a company whose employees' salaries are around $100,000, an employee with a salary of $10,000 can be considered to be an outlier.

Applications of outlier detection include network intrusion detection, financial fraud detection, and abnormal medical condition detection. As a concrete example, imagine a company that is interested in improving road safety by making drivers more aware of their driving habits. To achieve this, data is collected from many hundreds of thousands of vehicles equipped with sensors connected with smartphones. The collected data includes phone model, trip length, battery drain, and so on. Outlier detection and explanation engines such as Macrobase [Bailis et al. 2017] can be used to analyze the collected data. For example, Macrobase can find that some trips showed abnormally short lengths, common in smartphones with Apple iOS 9.0 beta 1. This reason was reported to the engineers, who discovered that a buggy Bluetooth stack was introduced in OS 9.0 beta 1, preventing iOS devices from connecting to in-car sensors.

Outlier detection faces two main challenges. First, defining what is a normal data pattern and what is an outlier can be difficult as different data and applications differ in what is considered normal. Many different detection techniques have been proposed to define normal behavior. Second, many outlier detection techniques lose their effectiveness when the number of dimensions (attributes) of the dataset is large; this effect is commonly known as the curse of dimensionality.

1.2.2 Data Deduplication

Duplicate records occur for many reasons. Data deduplication, also known as duplicate detection, record linkage, record matching, or entity resolution, refers to the process of identifying tuples in one or more relations that refer to the same real-world entity. For example, a customer might be recorded multiple times in a customer database if the customer used different names when purchasing; a single item might be represented multiple times in an online shopping site; and duplicate records might appear after a data integration project because that record had different representations in original data sources. A data deduplication process usually involves many steps and choices, including designing similarity metrics to evaluate the similarity for a pair of records, training classifiers to determine whether a pair of records are duplicates, clustering all records to obtain clusters of records that represent the same real-world entity, consolidating clusters of records to unique representations, designing blocking or distributed strategies to scale up the deduplication process, and involving humans to decide whether a record pair are duplicates when machines are uncertain.

RIGHTSLINK

To address this, many open-source and commercial data deduplication tools have been built, such as Magellan [Konda et al. 2016] and Data Tamer [Stonebraker et al. 2013] (later commercialized as Tamr[4]). Fortune 500 companies use these tools to make sense of their large procurement data. Large companies often have multiple business units, and each unit buys many parts and products from many suppliers. These units might be buying the same part and product from the same supplier without each other's knowledge, and therefore cannot get the best pricing. Furthermore, the same part and product might be named differently across different units, which complicates the deduplication process. A great deal of money can saved by identifying multiple records from the same supplier.

Achieving good precision and recall at the same time is difficult in data deduplication—declaring all pairs are duplicates achieves perfect recall but poor precision while declaring no pairs as duplicates achieves perfect precision, but poor recall. The problem is especially challenging given the myriad of design choices in designing a data deduplication workflow. Furthermore, data deduplication is inherently a combinatorial task that has quadratic complexity. For example, when done naively, comparing all pairs of only 1000 records requires 499,500 comparisons. In addition, grouping records that refer to the same real-world entity can be even harder. For example, correlation clustering used for grouping tuples that represent the same real-world entity is an NP-hard problem [Elsner and Schudy 2009].

1.2.3 Data Transformation

Data transformation refers to the task of transforming data from one format to another, for example, transforming phone numbers to a standard format by adding "-" in between digits. Data transformations can be seen as error repair activities and are used at various stages of data analytics. For example, before running a data integration project, transformations are often used to standardize data formats, enforce standard patterns, or trim long strings. Transformations are also used at the end of the ETL process, for example, to merge clusters of duplicate records.

Transform-Data-by-Example is an Excel add-in that finds the desired transformation easily for a given transformation task.[5] Only a few examples of the desired output are needed, and Transform-Data-by-Example automatically finds relevant data transformation functions from a large collection that it has already indexed. This collection is acquired from a variety of sources, including Github source codes, Stackoverflow code snippets, and .Net libraries. Users can also extend the collec-

4. https://www.tamr.com

5. https://www.microsoft.com/en-us/research/project/transform-data-by-example/

RIGHTSLINK

tion to cover domain-specific transformation capabilities by providing their own transformation functions.

Often, no ground truth is available to train or create accurate transformation solutions, so there might be an infinite number of applicable transformation programs. Although the desired transformation might be clear to a human expert, running all of these possibilities is expensive. Thus, practical data transformation tools must effectively prune the space interactively to minimize cost.

1.2.4 Rule-Based Data Cleaning

Another common way to detect and rectify errors in databases is through the enforcement of data quality rules, often expressed as integrity constraints (ICs) [Chu et al. 2013b, Kolahi and Lakshmanan 2009, Fan and Geerts 2012]. An example data quality is "*For two employees working at the same branch of a company, the senior employee cannot have less vacation time than the junior employee.*" Given a dataset, data quality rules can either be manually designed by domain experts who understand the semantics of the data, or be automatically mined from the data [Chu et al. 2013a, Chiang and Miller 2008]. In this context, (qualitative) error detection is the process of identifying *violations of ICs*, namely, subsets of records that cannot co-exist, and error repair is the exercise of modifying the database, such that the violations are resolved and the new data conforms to the given ICs.

Rule-based data cleaning techniques using denial constraints [Chu et al. 2013a, Chu et al. 2013b] have been proven to be extremely valuable in detecting and repairing errors in real data in consultation with animal scientists at UC Berkeley studying the effects of firewood cutting on small terrestrial vertebrates (birds, amphibians, reptiles, and small mammals) [Tietje et al. 2018]. Each record in the dataset contains information about capturing an animal, including the time and the location of the capture, the tag ID of the animal captured, and the properties of the animal at the time of the capture, such as weight, species type, gender, and age group. The dataset was collected over the past 20 years. Since every capture was first recorded in a log book and then later transcribed into Excel sheets, there are missing values, typos, and transcribing errors in the dataset. A dozen discovered data quality rules, expressed in the form of denial constraints [Baudinet et al. 1999], are used to detect and repair data errors. This helped the animal data scientists to correct hundreds of errors that would otherwise be very difficult to spot by humans.

Deriving a comprehensive set of integrity constraints that accurately reflects an organization's policies and domain semantics is a difficult task. Data is often scattered across silos; for example, different departments may keep their personnel records separate and may have different policies to determine an employee's salary.

RIGHTSLINK

Furthermore, data quality rules of varying expressiveness can be found in an organization, ranging from ad-hoc program scripts to simple sanity checks. Rather than consulting domain experts, techniques to automatically discover ICs can be used, which need to balance the trade-off between the expressiveness of the ICs and the efficiency of discovery. Given a set of defined ICs and their violations in a dirty data set, the number of possible ways to update the data to solve the violations is often too large for humans to examine exhaustively. Thus, data cleaning algorithms must be able to search through the huge space of possible updates efficiently to suggest the most plausible updates.

1.2.5 ML-Guided Cleaning

One can easily recognize, from the discussion so far, that ML is a natural tool to use in several tasks: classifying duplicate pairs in deduplication, estimating the most likely value for a missing value, predicting a transformation, or classifying values as normal or outliers. Indeed, several of the proposals that we discuss in this book use ML as a component in the proposed cleaning pipeline. We describe these in the corresponding chapters, and in Chapter 7 detail the use of ML for cleaning. However, with the rapid advancement in scaling inference and deploying large-scale ML models, new opportunities arise, namely, treating data cleaning as an ML task (mainly statistical inference), and dealing with all the aforementioned problems holistically. HoloClean [Rekatsinas et al. 2017a] is one of the first ML-based holistic cleaning frameworks, built on a probabilistic model of unclean databases [De Sa et al. 2019]. We discuss this direction in detail in Chapter 7.

1.2.6 Human-Involved Cleaning

It is usually impossible for machines to clean data errors with 100% confidence; therefore, human experts are often consulted in a data cleaning workflow to resolve various kinds of uncertainties whenever possible and feasible. For example, the metadata (e.g., keys and integrity constraints) generated by the discovery and profiling step need to be verified by humans before they can be used for detecting and repairing errors, since the metadata is discovered based on one instance of the data set and may not necessarily hold on any data instance of the same schema (e.g., future data). Another example of "judicious" human involvement is in the error detection step: to build a high-quality classifier to determine whether a pair of records are duplicates, we need enough training data, namely, labeled records (as duplicates or as non-duplicates). However, there are far more non-duplicate record pairs than duplicate record pairs in a data set; asking humans to label a random set of record pairs would lead to a set of unbalanced training examples. Therefore,

RIGHTSLINK

humans need to consulted in an intelligent way to solicit training examples that are most beneficial to the classifier.

Humans are also needed in the error repair step: data updates generated by automatic data repair techniques are mostly based on heuristics and/or statistics, such as minimal repair distance or maximum likelihood estimation, and thus are not necessarily correct repairs. Therefore, they must be verified by humans before they can be applied to a data set. We highlight when and how humans need to be involved when we discuss various cleaning proposals in this book.

1.2.7 Structure of the Book

We present the details of current proposals addressing the topics discussed earlier in multiple chapters organized along two main axes, as shown in Figure 1.2: (1) task: detection vs. repair; and (2) approach: qualitative vs. quantitative. For example, outlier detection is a quantitative error detection task (Chapter 2); data deduplication is a qualitative data cleaning task (Chapter 3), which has both detection of duplicates and repair (also known as record consolidation); data transformation is considered a repair task (Chapter 4) that can be used to fix many types of errors, including

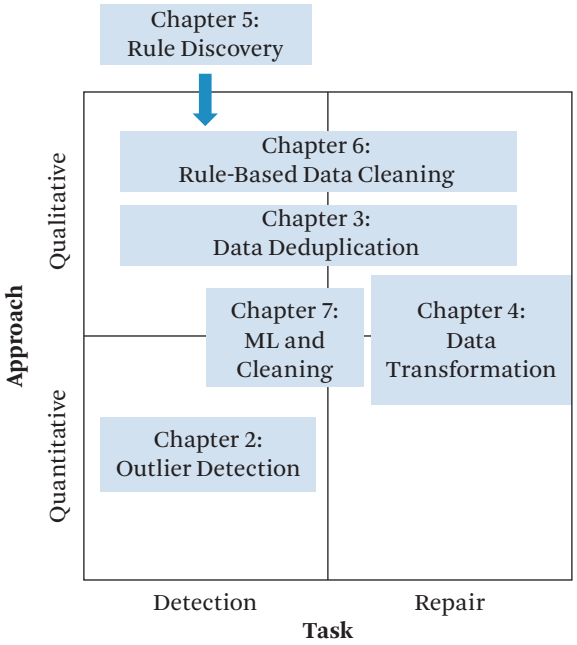


Figure 1.2 Structure of the book organized along two dimensions: task and approach.

RIGHTSLINK

outliers and duplicates; and rule-based data cleaning is a qualitative cleaning task (Chapter 6). In addition, a discovery step is usually necessary to obtain data quality rules necessary for rule-based cleaning (Chapter 5). Finally, we include a chapter that discusses the recent advances in ML and data cleaning (Chapter 7) spanning both dimensions.

Due to the diverse challenges of data cleaning tasks, different cleaning solutions draw principles and tools from many fields, including statistics, formal logic systems such as first-order logic, distributed data processing, data mining, and ML. Although a basic familiarity with these areas should facilitate understanding of this book, we try to include necessary background and references whenever appropriate.

RIGHTSLINK

2 Outlier Detection

Quantitative error detection often targets data anomalies with respect to some definition of "normal" data values. While an exact definition of an outlier depends on the application, there are some commonly used definitions, such as "*an outlier is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism*" [Hawkins 1980] and "*an outlier observation is one that appears to deviate markedly from other members of the sample in which it occurs*" [Barnett and Lewis 1994]. Different outlier detection methods will generate different "candidate" outliers, possibly along with some confidence scores.

Many applications of outlier detection exist. In the context of computer networks, different kinds of data, such as operating system calls and network traffic, are collected in large volumes. Outlier detection can help with detecting possible intrusions and malicious activities in the collected data. In the context of credit card fraud, unauthorized users often exhibit unusual spending patterns, such as a buying spree from a distant location. Fraud detection refers to the detection of criminal activities in such financial transactions. Last, in the case of medical data records, such as MRI scans, PET scans, and ECG time series, automatic identification of abnormal patterns or records in these data often signals the presence of a disease and can help with early diagnosis.

There are many challenges in detecting outliers. First, defining normal data patterns for a normative standard is challenging. For example, when data is generated from wearable devices, spikes in recorded readings might be considered normal if they are within a specific range dictated by the sensors used. On the other hand, spikes in salary values are probably interesting outliers when analyzing employee data. Therefore, understanding the assumptions and limitations of each outlier detection method is essential for choosing the right tool for a given application domain. Second, many outlier detection techniques lose their effectiveness when the number of dimensions (attributes) of the dataset is large; this effect is commonly known as the "curse of dimensionality." As the number of dimensions increases, it

RIGHTSLINK

becomes increasingly difficult to accurately estimate the multidimensional distribution of the data points [Scott 2008], and the distances between points approach zero and become meaningless [Beyer et al. 1999]. We give some concrete examples and more details on these challenges in the next section.

In Section 2.1, we present a taxonomy of outlier detection techniques. We discuss in detail each of these categories in Sections 2.2, 2.3, and 2.4, respectively. In Section 2.5, we discuss outlier detection techniques for high-dimensional data that address the "curse of dimensionality."

2.1 A Taxonomy of Outlier Detection Methods

Outlier detection techniques mainly differ in how they define normal behavior. Figure 2.1 depicts the taxonomy we adopt to classify outlier detection techniques, which can be divided into three main categories: statistics-based outlier detection techniques, distance-based outlier detection techniques, and model-based outlier detection techniques [Aggarwal 2013, Chandola et al. 2009, Hodge and Austin 2004]. In this section, we give an overview of each category and their pros and cons, which we discuss in detail.

Statistics-Based Outlier Detection Methods. Statistics-based outlier detection techniques assume that the normal data points would appear in high probability regions of a stochastic model, while outliers would occur in the low probability regions of a stochastic model [Chandola et al. 2009]. There are two commonly used categories of approaches for statistics-based outlier detection. The first category is

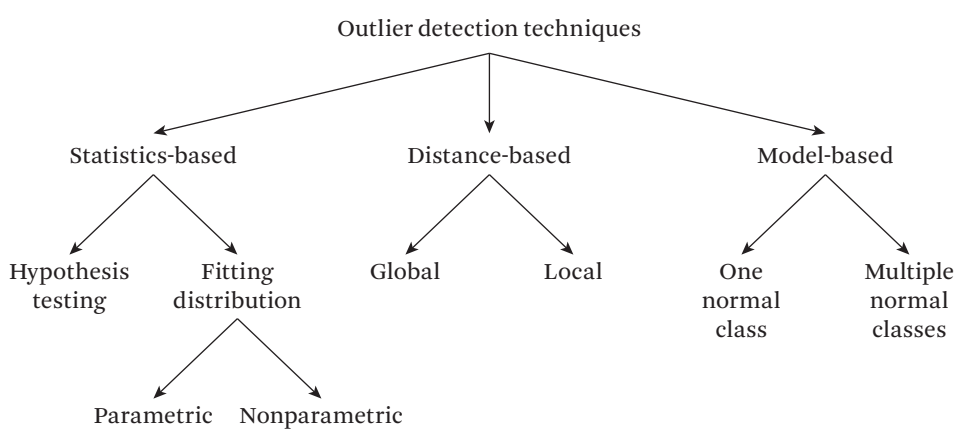


Figure 2.1 A taxonomy of outlier detection techniques.

RIGHTSLINK

based on *hypothesis testing methods*, such as the Grubbs Test [Grubbs 1969] and the Tietjen-Moore Test [Tietjen and Moore 1972]; they usually calculate a test statistic, based on observed data points, which is used to determine whether the null hypothesis (there is no outlier in the dataset) should be rejected. The second category of statistics-based outlier detection techniques aims at *fitting a distribution or inferring a probability density function (pdf) based on the observed data*. Data points that have low probability according to the pdf are declared to be outliers. Techniques for fitting a distribution can be further divided into parametric approaches and nonparametric approaches. Parametric approaches for fitting a distribution assume that the data follows an underlying distribution and aim at finding the parameters of the distribution from the observed data. For example, assuming the data follows a normal distribution, parametric approaches would need to learn the mean and variance for the normal distribution. In contrast, nonparametric approaches make no assumption about the distribution that generates the data; instead, they infer the distribution from the data itself.

There are advantages of statistics-based techniques.

1. If the underlying data follows a specific distribution, then the statistical outlier detection techniques can provide a statistical interpretation for discovered outliers.
2. Statistical techniques usually provide a score or a confidence interval for every data point, rather than making a binary decision. The score can be used as additional information while making a decision for a test data point.
3. Statistical techniques usually operate in an unsupervised fashion without any need for labeled training data.

There are also some disadvantages of statistics-based techniques.

1. Statistical techniques usually rely on the assumption that the data is generated from a particular distribution. This assumption often does not hold true, especially for high-dimensional real datasets.
2. Even when the statistical assumption can be reasonably justified, there are several hypothesis test statistics that can be applied to detect anomalies; choosing the best statistic is often not a straightforward task. In particular, constructing hypothesis tests for complex distributions that are required to fit high-dimensional datasets is nontrivial.

Distance-Based Outlier Detection Methods. Distance-based outlier detection techniques often define a distance between data points that is used for defining a

RIGHTSLINK

normal behavior. For example, a normal data point should be close to many other data points, and data points that deviate from such normal behavior are declared outliers [Knorr and Ng 1998, 1999, Breunig et al. 2000]. Distance-based outlier detection methods can be further divided into global or local methods depending on the reference population used when determining whether a point is an outlier. A *global distance-based outlier detection method* determines whether a point is an outlier based on the distance between that data point and all other data points in the dataset. On the other hand, a *local method* considers the distance between a point and its neighborhood points when determining outliers.

There are advantages of distance-based techniques.

1. A major advantage of distance-based techniques is that they are unsupervised in nature and do not make any assumptions regarding the generative distribution for the data. Instead, they are purely data driven.
2. Adapting distance-based techniques to different data types is straightforward, and primarily requires defining an appropriate distance measure for the given data.

There are disadvantages of distance-based techniques.

1. If the data has normal instances that do not have enough close neighbors, or if the data has anomalies that have enough close data points, distance-based techniques will fail to label them correctly.
2. The computational complexity of the testing phase is also a significant challenge since it involves computing the distance of every pair of data points.
3. Performance of a nearest neighbor-based technique greatly relies on a distance measure, defined between a pair of data instances, which can effectively distinguish between normal and anomalous instances. Defining distance measures between instances can be challenging when the data is complex, for example, graphs, sequences, and so on.

Model-Based Outlier Detection Methods. Model-based outlier detection techniques first learn a classifier model from a set of labeled data points, and then apply the trained classifier to a test data point to determine whether it is an outlier. Model-based approaches assume that a classifier can be trained to distinguish between the normal data points and the anomalous data points using the given feature space. They label data points as outliers if none of the learned models classify them as normal points. Based on the labels available to train the classifier,

RIGHTSLINK

model-based approaches can be further divided into two subcategories: *multi-class* model-based techniques and *one-class* model-based techniques. Multi-class model-based techniques assume that the training data points contain labeled instances belonging to multiple normal classes. On the other hand, one-class model-based techniques assume that all the training data points belong to one normal class.

The advantages of model-based techniques include:

1. Model-based techniques, especially the multi-class techniques, can make use of powerful algorithms that can distinguish between instances belonging to different classes.
2. The testing phase of model-based techniques is fast, since each test instance needs to be compared against the precomputed model.

The major disadvantages of model-based techniques is that they must rely on the availability of accurate labels for various normal classes, which is often not possible.

2.2 Statistics-Based Outlier Detection

In this section, we discuss statistics-based outlier detection methods. First, we review some basics about data distributions in Section 2.2.1. Section 2.2.2 then shows how hypothesis testings, such as Grubbs' Test, are used for detecting outliers. In Section 2.2.3, we present parametric approaches for fitting a distribution to the data, and also introduce the notion of *robust statistics*, which can capture important properties of the underlying distribution in the presence of outliers in the data. In Section 2.2.4, we introduce nonparametric approaches for fitting a distribution to the data by using histograms and kernel density estimations (KDEs).

2.2.1 A Note on Data Distribution

Database practitioners usually think of data as a collection of records. They are interested in descriptive statistics about the data (e.g., "what is the min, max, and average salary for all the employees?"), and they usually do not care much about how the data is generated. On the other hand, statisticians usually treat data as a sample of some data-generating process. In many cases, they try to find a model or a distribution of that process that best describes the available sample data.

The most commonly used distribution is the Gaussian or normal distribution, which is characterized by a mean μ and a standard variance σ [Barnett and Lewis 1994]. While the normal distribution might not be the exact distribution of a given

RIGHTSLINK

dataset, it is the cornerstone of modern statistics, and it accurately models the sum of many independently identically distributed variables according to the central limit theorem. The following equation shows the pdf of the normal distribution $N(\mu, \sigma^2)$:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}.$$

Multivariate Gaussian distribution or multivariate normal distribution is a generalization of the univariate normal distribution to higher dimensions. A random vector consisting of d random variables X_i, where $i \in [1, d]$, is said to be d-variate normally distributed if every linear combination of its d components has a univariate normal distribution. The mean of the d variables is a d-dimensional vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_d)^T$, where μ_i is the mean of the random variable X_i. The covariance matrix Σ of these d random variables (also known as dispersion matrix or variance–covariance matrix) is a matrix, where $\Sigma_{i,j}$ (Row i and Column j of Σ) is the covariance between the two random variables X_i and X_j, namely, $\Sigma_{i,j} = cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$. Intuitively, the covariance matrix generalizes the notion of variance of a single random variable or the covariance of two random variables to d dimensions.

2.2.2 Hypothesis Testing for Outlier Detection

A statistical hypothesis is an assumption about a population that may or may not be true. Hypothesis testing refers to the formal procedures used by statisticians to accept or reject statistical hypotheses, which usually consists of the following steps [Lehmann and Romano 2006]: (1) formulate the *null and the alternative hypothesis*; (2) decide which test is appropriate and state the relevant *test statistic* T; (3) choose a *significance level* α, namely, a probability threshold below which the null hypothesis will be rejected; (4) determine the *critical region* for the test statistic T, namely, the range of values of the test statistic for which the null hypothesis is rejected; (5) compute from the current data the observed value t_{obs} of the test statistic T; and (6) reject the null hypothesis if t_{obs} falls under the critical region. Different known test statistics exist for testing different properties of a population. For example, the *chi-square test for independence* is used to determine whether there is a significant relationship between two categorical variables, the *one-sample t-test* is commonly used to determine whether the hypothesized mean differs significantly from the observed sample mean, and the *two-sample t-test* is used to determine whether the difference between two means found in two samples is significantly different from the hypothesized difference between two means.

RIGHTSLINK

A number of formal hypothesis tests for detecting outliers have been proposed in the literature. These tests usually differ according to the following characteristics.

- What is the underlying distribution model for the data?
- Is the test designed for testing a single outlier or multiple outliers?
- If the test is for detecting multiple outliers, does the number of outliers need to be specified exactly in the test?

For example, the Grubbs Test [Grubbs 1969] is used to detect a single outlier in a univariate dataset that follows an approximately normal distribution. The Tietjen-Moore Test [Tietjen and Moore 1972] is a generalization of the Grubbs test to the case of more than one outlier; however, it requires the number of outliers to be specified exactly. The Generalized Extreme Studentized Deviate (ESD) Test [Rosner 1983] requires only an upper bound on the suspected number of outliers and is often used when the exact number of outliers is unknown.

We use Grubbs' test as an example to show how hypothesis testing is used for detecting outliers, as it is a standard test when testing for a single outlier. This outlier is expunged from the dataset and the test is repeated until no outliers are detected. However, multiple iterations change the probabilities of detection, and the test is not to be used for a small sample size. Grubbs' test is defined for the following hypothesis, where H_0 is the null hypothesis and H_a is the alternative hypothesis:

H_0 : there are no outliers in the dataset

H_a : there is exactly one outlier in the dataset.

Grubbs' test statistic is defined as $G = \frac{\max_{i=1,\ldots,N} |Y_i - \overline{Y}|}{s}$, with $\overline{Y}$ and s denoting the sample mean and standard deviation, respectively. Grubbs' test statistic is the largest absolute deviation from the sample mean in units of the sample standard deviation. The above test statistic is the *two-sided* version of the test. Grubbs' test can also be used as a *one-sided* test for determining whether the minimum value is an outlier or the maximum value is an outlier. To test whether the minimum value Y_{min} is an outlier, the test statistic is defined as $G = \frac{\overline{Y} - Y_{min}}{s}$; similarly, to test whether the maximum value Y_{max} is an outlier, the test statistic is defined as $G = \frac{Y_{max} - \overline{Y}}{s}$. For the two-sided test, the null hypothesis of no outliers is rejected if $G > \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\alpha/(2N),N-2}}{N-2+t^2_{\alpha/(2N),N-2}}}$ with $t^2_{\alpha/(2N),N-2}$ denoting the upper critical value of the t-distribution with $N-2$ degrees of freedom and a significance level of $\alpha/2N$. For a one-sided test, replace $\alpha/2N$ with α/N.

RIGHTSLINK

Table 2.1 **Employee records, including name, age, income, and tax**

	Name	Age	Income	Tax
t_1	Vivian Baskette	1	70	7
t_2	Jamison Marney	25	110	11
t_3	Marie Mulero	27	80	8
t_4	Trudi Kimmell	30	130	13
t_5	Stephanie Lindemann	32	120	7
t_6	Dia Werley	35	80	8
t_7	Abbie Lama	40	90	9
t_8	Misti Luce	41	100	10
t_9	Wilda Byerly	1000	120	12

Example 2.1 Consider Table 2.1, including the name, age, income, and tax of employees. Domain knowledge suggests that the first value $t_1[age]$ and the last value $t_9[age]$ are outlying age values. We use Grubbs' test with a significance level $\alpha = 0.05$ to identify outliers in the age column.

The mean of the 9 age values is 136.78, and the standard deviation of the 9 age values is 323.92. Grubbs' test statistic $G = \frac{\max_{i=1,\ldots,9} |t_i[age]-136.78|}{323.92} = 2.66$, which is obtained at t_9. With a significance level $\alpha = 0.05$, the critical value is computed as $G_{\text{critical}} = 2.21$. Since $G > G_{\text{critical}}$, the null hypothesis is rejected. Therefore, $t_9[age]$ is reported as an outlier.

Removing $t_9[age]$, we are left with 8 age values. The mean of the 8 age values is 28.88, and the standard deviation of the 8 age values is 12.62. Grubbs' test statistic is $G = \frac{\max_{i=1,\ldots,8} |t_i[age]-28.88|}{12.62} = 2.21$, which is obtained at t_1. With a significance level $\alpha = 0.05$, the critical value is computed to $G_{\text{critical}} = 2.12$. Since $G > G_{\text{critical}}$, the null hypothesis is rejected. Therefore, $t_1[age]$ is reported as an outlier.

Removing $t_1[age]$, we are left with 7 age values. The mean of the 7 age values is 32.86, and the standard deviation of the 7 age values is 6.15. Grubbs' test statistic $G = \frac{\max_{i=2,\ldots,8} |t_i[age]-32.86|}{6.15} = 1.32$, which is obtained at t_8. With a significance level $\alpha = 0.05$, the critical value is computed as $G_{\text{critical}} = 2.01$. Since $G < G_{\text{critical}}$, the null hypothesis is accepted.

Previous discussion assumes that the data follows an approximately normal distribution. To assess this, several graphical techniques can be used, including the Normal Probability Plot, the Run Sequence Plot, the Histogram, the Box Plot, and the Lag Plot.

RIGHTSLINK

Iglewicz and Hoaglin provide an extensive discussion of the outlier tests previously given [Iglewicz and Hoaglin 1993]. Barnett and Lewis [1994] provide a book length treatment of the subject. They provide additional tests when data is not normally distributed.

2.2.3 Fitting Distribution: Parametric Approaches

The other type of statistics-based approach first fits a statistical distribution to describe the normal behavior of the given data points, and then applies a statistical inference procedure to determine if a certain data point belongs to the learned model. Data points that have a low probability according to the learned statistical model are declared as anomalous outliers. In this section, we discuss parametric approaches for fitting a distribution to the data.

Univariate

We first consider univariate outlier detection, for example, for a set of values $x_1, x_2, \ldots, x_n$ that appear in one column of a relational table. Assuming the data follows a normal distribution, fitting the values under a normal distribution essentially means computing the mean μ and the standard deviation σ from the current data points $x_1, x_2, \ldots, x_n$. Given μ and σ, a simple way to identify outliers is to compute a *z-score* for every x_i, which is defined as the number of standard deviations away x_i is from the mean, namely $z\text{-}score(x_i) = \frac{|x_i - \mu|}{\sigma}$. Data values that have a *z-score* greater than a threshold, for example, of three, are declared to be outliers.

Since there might be outliers among $x_1, x_2, \ldots x_n$, the estimated μ and σ might be far off from their actual values, resulting in missing outliers in the data, as we show in Example 2.2.

Example 2.2 Consider again the age column in Table 2.1. The mean of the 9 age values is 136.78, and the standard deviation of the 9 age values is 323.92. The procedure that identifies values that are more than 2 standard deviations away from the mean as outliers would mark values that are not in the range of $[136.78 - 2 * 323.92, 136.78 + 2 * 323.92] = [-511.06, 784.62]$. The last value $t_9[age]$ is not in the range, and thus is correctly marked as an outlier. The first value $t_1[age]$, however, is in the range and is thus missed.

This effect is called *masking* [Hellerstein 2008]; that is, a single data point has severely shifted the mean and standard deviation so much as to mask other outliers. To mitigate the effect of masking, *robust statistics* are often employed, which can correctly capture important properties of the underlying distribution even in the face of many outliers in the data values. Intuitively, the *breakdown point* of an

RIGHTSLINK

estimator is the proportion of incorrect data values (e.g., arbitrarily large or small values) an estimator can tolerate before giving an incorrect estimate. The mean and standard deviation have the lowest breakdown point: a single bad value can distort the mean completely.

Robust Univariate Statistics. We now introduce two robust statistics: the median and the median absolute deviation (MAD) that can replace mean and standard deviation, respectively. The median of a set of n data points is the data point for which half of the data points are smaller, and half are larger; in the case of an even number of data points, the median is the average of the middle two data points. The median, also known as the 50th percentile, is of critical importance in robust statistics with a breakdown point of 50%; as long as no more than half the data are outliers, the median will not give an arbitrarily bad result. The median absolute deviation (MAD) is defined as the median of the absolute deviations from the data's median, namely, $MAD = median_i(|x_i - median_j(x_j)|)$. Similar to the median, MAD is a more robust statistic than the standard deviation. In the calculation of the standard deviation, the distances from x_i to the mean are squared, so large deviations, which often are caused by outliers, are weighted heavily, while in the calculation of MAD, the deviations of a small number of outliers are irrelevant because MAD is using the median of the absolute deviations.

The median and MAD lead to a robust outlier detection technique known as *Hampel X84* [Hampel et al. 2011] that is quite reliable in the face of outliers since it has a breakdown point of 50%. Hampel X84 marks outliers as those data points that are more than 1.4826θ MADs away from the median, where θ is the number of standard deviations away from the mean one would have used if there were no outliers in the dataset. The constant 1.4826 is derived under a normal distribution, where one standard deviation away from the mean is about 1.4826 MADs.

Example 2.3 For detecting outliers in the ages column in Table 2.1, we used 2 standard deviations away from the mean in Example 2.2. Therefore, we would like to flag points that are $1.4826 * 2 = 2.9652$ away from the median as outliers.

The median of the set of values in Example 2.2 is 32 and the MAD is 7. The normal value range would be $[32 - 7 * 2.9652, 32 + 7 * 2.9652] = [11.2436, 52.7564]$, which is a much more reasonable range than $[-511.06, 784.62]$ derived using mean and standard deviation. Under the new normal range, the first value and the last value are now correctly flagged as outliers.

RIGHTSLINK

Multivariate

So far, we have considered detecting univariate outliers in one dimension by fitting the data to a normal distribution. However, some outliers can only be revealed when considering multiple dimensions; we call these *multivariate outliers*.

Example 2.4 Consider the income and tax columns in Table 2.1. In general, income varies for different persons, and so do tax rates. However, one would expect a positive correlation between income and tax rate, namely, a higher income is usually associated with a higher tax rate. Therefore, a tax record that has a high income but low tax rate should be flagged as an outlier, even though the tax income or the tax rate in that record might not be an outlier when considered separately in their respective columns. Consider t_5 in Table 2.1. $t_5[income]$ is not an outlier compared with the values in the income column, and $t_5[tax]$ is also not an outlier compared with the values in the tax column. However, taking the income and tax columns into account simultaneously, t_5 is an outlier, since it has a relatively low tax for a relatively high income value.

Suppose we have a table that has n rows and d columns; we model each column as a random variable X_i, $\forall i \in [1, d]$. Let the mean of the d variables be a d-dimensional vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_d)^T$ and the covariance matrix be Σ, where $\Sigma_{i,j}$is the covariance between the two random variables X_i and X_j. Given the mean and the covariance matrix, the questions are how to measure the distance between a particular data point to the mean using the covariance matrix, and how much distance should qualify a point as an outlier. The *Mahalanobis distance* [Mahalanobis 1936] is the multidimensional generalization of measuring how many standard deviations away a point is from the mean of the population. Specifically, the Mahalanobis distance of a point x to the mean vector μ is defined as $Mahalanobis(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$. Notice that if Σ is the identity matrix, then the Mahalanobis distance reduces to the standard Euclidean distance between x and μ. If every dimension is a normal distribution, then the square of the Mahalanobis distance follows a χ^2_d distribution with d degrees of freedom. Using the probability cumulative distribution function of χ^2_d, we define the threshold of the Mahalanobis distance to be a number where most of the Mahalanobis distance (e.g., 95%) is smaller than that threshold distance.

Robust Multivariate Statistics. Similar to the univariate mean and variance, the multivariate mean and covariance matrix are not robust, i.e., a single bad data

RIGHTSLINK

Algorithm 2.1 **FASTMCD**

Input: One relational instance I, schema R
Output: Mean μ and the covariance matrix Σ
$H_1 \leftarrow$ a random subset of I, of size h
$H_0 \leftarrow \emptyset$
while $H_0 \neq H_1$ **do**
  $H_0 \leftarrow H_1$
  compute μ and Σ of H_0
  calculate the Mahalanobis distance for all points in I using μ and Σ
  $H_1 \leftarrow$ choose h points with the smallest distance
end while
repeat the above procedure many times (e.g., 500),
and choose the H whose Σ has the smallest determinant

point might severely distort the mean and the covariance matrix, and thus robust estimators must be used. The most famous method for the robustification of multivariate mean and covariance matrix is called the Minimum Covariance Determinant (MCD) by Rousseeuw and Driessen [1999]. MCD chooses a subset of h points that minimizes the determinant of the covariance matrix over all possible subsets of size h (h is usually chosen to reflect the belief about how many points in the dataset are outliers). For example, assume 10% of n points are outliers, then h is chosen to be $0.9n$. The multivariate mean and covariance matrix of the h points are used to compute the Mahalanobis distance for all n points. A brute force way to find the covariance matrix with the smallest determinant is to enumerate all possible subsets of size h from the n data points, which is obviously very expensive as one has to evaluate all the possible subsets. A greedy algorithm called FASTMCD [Rousseeuw and Driessen 1999] tries to solve the efficiency problem. Algorithm 2.1 describes FASTMCD. It starts by randomly selecting a subset of h data points, and the mean μ and the covariance matrix Σ are computed using the random sample. Then, the Mahalanobis distances for all points in I are computed using the μ and Σ. The random sample is updated using h points with the smallest Mahalanobis distances. The sample is updated in iterations until it remains unchanged between two iterations. The above process is repeated by using a different random sample as the initial seed, and the best result is used.

The correctness of the algorithm can be proven by showing that $det(H_1) \leq det(H_0)$ in the iterative portion. The mean and the covariance matrix computed

RIGHTSLINK

using FASTMCD can be shown to have a 50% breakdown point [Rousseeuw and Driessen 1999].

Most of our discussions of parametric approaches have been based on normal distributions. In practice, not all datasets are normally distributed. Two other distributions are frequently observed: multimodal distributions and Zipfian distributions. In some cases, data appears to have many "peaks," such distributions are typically referred as being *multimodal*. Oftentimes, these multimodal distributions can be seen as a superimposed normal distributions, known as a *mixture of Gaussians*. In some other cases, a large fraction of the data is condensed into a small fraction of values, while the remainder of the data values are spread across a long tail of rare values. The Zipfian distribution exhibits such a phenomenon. It is important to choose the right distribution to detect outliers using parametric approaches.

2.2.4 Fitting Distribution: Nonparametric Approaches

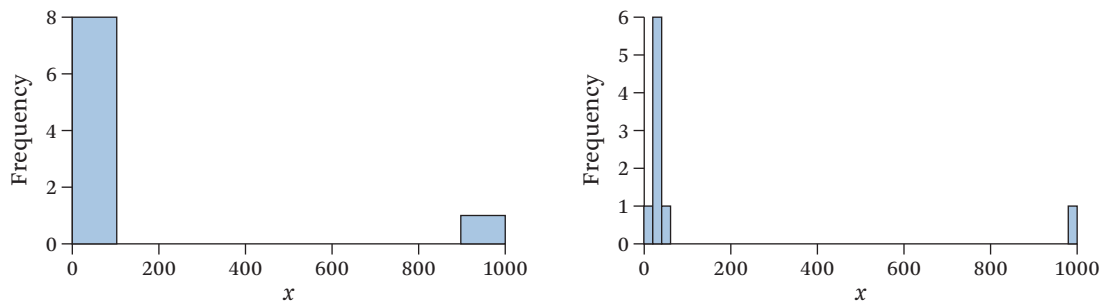
An obvious drawback of using parametric approaches for fitting a distribution is that they assume the data to follow an underlying distribution. In contrast, nonparametric approaches make no assumption about the distribution that generates the data; instead, they infer the distribution from the data itself. There are mainly two types of techniques in this category: histogram-based techniques and kernel density-based techniques.

Histograms

A histogram [Pearson 1894] is a graphical representation of the distribution of numerical data values, and is often used to estimate the probability distribution of a continuous random variable. The first step toward creating a histogram is to discretize or bin the range of values by dividing the range of the random variable into a series of intervals, which are usually consecutive and non-overlapping. The second step is to count how many values fall under each bin. If the bins are of equal size, a bin is created with height proportional to the frequency of each bin, i.e., the number of values a bin contains; this type of histogram is called an *equi-width* histogram. In general, however, bins can be of varying width. If the width of bins are chosen such that every bin has the same frequency, the histogram is called an *equi-depth* histogram.

Equi-width histograms can be used to detect outliers; data points that belong to bins that have very low frequency are reported as outliers. A major challenge with histogram-based outlier detection approaches is that it is often very difficult to come up with the "right" width of the interval. If the width of the bins is too

RIGHTSLINK



(a) 10 bins (b) 50 bins

Figure 2.2 Equi-width histograms for outlier detection.

narrow, the frequency of some bins might be low, and normal data points belong to those bins are reported as outliers, and if the width of the bins is too wide, outlier data points might get absorbed in bins that have normal data points, and thus fail to report the real outliers. The problem is further exacerbated when adapting histograms to multivariate data, since the frequency of every division diminishes as the number of dimensions increases, as we mentioned earlier as a curse of dimensionality.

Example 2.5 Figure 2.2 shows two equi-width histograms we built for the age column in Table 2.1 using free statistics software [Wessa 2012]. We report data points residing in bins that contain less than three data points as outliers.

In Figure 2.2(a), the age values are split into 10 equal-width bins, namely (0, 100], (100, 200], . . . , (900, 1000]. Only the first bin and the last bin contain data points. Specifically, bin (0, 100] contains 8 points and bin (900, 1000] contains 1 point. Therefore, we report $t_9[age] \in (900, 1000]$ as an outlier.

In Figure 2.2(b), the age values are split into 50 equal-width bins, namely (0, 20], (20, 40], . . . , (980, 1000]. Only the first three bins and the last bin contain data points. Specifically, bin (0, 20] contains 1 point, bin (20, 40] contains 6 points, bin (40, 60] contains 6 points, and bin (980, 1000] contains 1 point. Therefore, we report $t_1[age] \in (0, 20]$, $t_8[age] \in (40, 60]$, and $t_9[age] \in (980, 1000]$ as outliers.

There are usually two ways to generalize a histogram to deal with multivariate data: (1) computing an outlier score for each dimension using the one-dimensional histogram, and then combining the score to obtain an overall outlier score for every data point; and (2) binning every dimension to divide multiple dimensions together; the number of data points belonging to each division is counted, and the

RIGHTSLINK

data points that belong to divisions with very low frequency counts are reported as outliers. The performance of histogram methods usually deteriorates with increasing number of dimensions due to the sparsity of the grid structure with increasing dimensionality [Aggarwal 2013]. For example, given a 10-dimensional space with each dimension being split into 10 bins, the grid will contain 10^{10} grid cells. It would require 10^{10} available data points to allow every grid cell to have one data point in expectation.

Kernel Density Estimation

KDE [Rosenblatt 1956, Parzen 1962, Gan and Bailis 2017] is a nonparametric way to estimate the pdf $f(x)$ of a random variable X based on a sample of n data points, which is the same as histograms, but can be endowed with properties such as *smoothness* or *continuity* by using a suitable kernel.

Let $x_1, x_2, \ldots, x_n$ be an independent and identically distributed sample drawn from some distribution with unknown pdf $f(x)$. KDE estimates $f(x)$ using $\hat{f}(x)$ defined as follows:

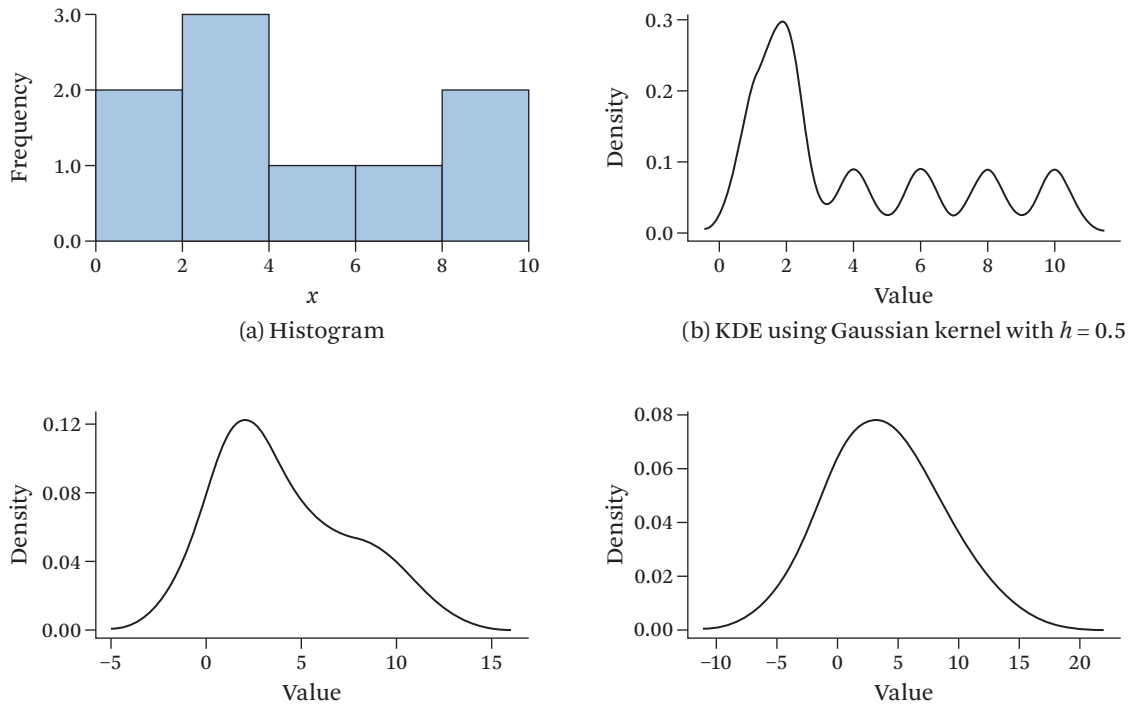
$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - x_i}{h}),$$

where $K(\bullet)$ is the kernel, and h is a smoothing parameter called the *bandwidth*. A kernel has to be a non-negative function that integrates to one and has mean zero, namely, $\int_{-\infty}^{\infty} K(x)dx = 1$ and $K(x) = K(-x)$. A kernel with a subscript h is a scaled kernel that satisfies $K_h(x) = 1/hK(x/h)$. Some commonly used kernel functions include the normal, uniform, triangular, biweight, triweight, and Epanechnikov kernels. Just as the choice of the width of the bin influences the result of histograms, the bandwidth of the kernel is a free parameter strongly influencing the KDE as well.

Example 2.6 To compare KDEs with histograms, consider 9 data points $x_1 = 1, x_2 = 1, x_3 = 2, x_4 = 2, x_5 = 2, x_6 = 4, x_7 = 6, x_8 = 9$, and $x_9 = 10$. Figure 2.3 compares the pdf derived using both the equi-width histogram with a bin width equal to 2 and three KDEs using different kernel functions and bandwidths. As we can see, KDEs in general produce much smoother estimates than the histogram. Comparing Figures 2.3(b), 2.3(c), and 2.3(d), we can see that bandwidth clearly controls the smoothness of the pdf produced. Given the estimated pdf, any points that have low probability are flagged as outliers.

The generalization of multivariate KDE from univariate KDE is similar to the generalization of multivariate histograms from univariate histograms, and is discussed in detail in Simonoff [2012].

RIGHTSLINK



(a) Histogram

(b) KDE using Gaussian kernel with $h = 0.5$

(c) KDE using Gaussian kernel with $h = 2$

(b) KDE using Gaussian kernel with $h = 4$

Figure 2.3 Compare histograms with KDEs. Graphs were produced using Wessa [2012].

2.3 Distance-Based Outlier Detection

In contrast to statistics-based outlier detection techniques, distance-based outlier techniques do not assume an underlying model that generates the data and are often nonparametric. These approaches often define a distance between data points, which is used for defining a normal behavior, such as normal data points should be close to many other data points, and any data point that deviates from such normal behavior is declared an outlier [Knorr and Ng 1998, 1999, Breunig et al. 2000].

Distance-based outlier detection methods can be further divided into global or local methods depending on the reference population used when determining whether a point is an outlier. A global distance-based outlier detection method determines whether a point is an outlier based on the distance between that data point and all other data points in the dataset. On the other hand, a local method consid-

RIGHTSLINK

ers the distance between a point and its neighborhood points when determining outliers.

2.3.1 Global Distance-Based Outlier Detection

Distance-based methods were first introduced by Knorr and Ng [Knorr and Ng 1998, 1999]. An object O in a dataset I is a distance-based outlier, i.e., $DB(p, D)$ outlier, if at least fraction p of the objects in I lies greater than distance D from O. $DB(p, D)$ can be seen as a generalization of some existing statistics-based outlier detection definitions. For example, let I be observations from an univariate normal distribution $N(\mu, \delta)$ and O be a point from I, then the z-score of O is greater than 3 if and only if O is a $DB(0.9988, 0.13\delta)$ outlier [Knorr and Ng 1998]. Knorr and Ng proved that statistics-based outlier detection definitions based on other underlying distributions, such as Student t-distribution and Poisson distribution, are also equivalent to $DB(p, D)$ outliers with suitable choice of p and D.

The definition of $DB(p, D)$ does not provide a ranking or a score of outliers and requires specifying a distance parameter D, which could be difficult to determine and may involve a trial and error process. There are other distance-based outlier definitions that look at the k nearest neighborhood of a data point. Kollios et al. [2003] defines an object O as outlier in a dataset I if at most k objects in I lie at distance at most D from O. Ramaswamy et al. [2000] defines outliers as the top few data elements whose distance to the kth nearest neighbor is greatest.

The simplest category of algorithms [Knorr and Ng 1998, 1999, Ramaswamy et al. 2000] for finding distance-based outliers are those using nested loops to find the distance between every point and every other point, which has a quadratic complexity. Another category of algorithms uses spatial index structures such as KD-trees, R-trees, or X-trees to find the nearest neighbors of each point [Knorr and Ng 1998, 1999, Ramaswamy et al. 2000]. This category of algorithms works well for low-dimensional datasets, and has a complexity of $O(n \log n)$ if the index tree can allow a $O(\log n)$ lookup for finding a point's nearest neighbor. However, the index structures become increasingly complex as the dimensionality increases. For example, Breunig et al. [2000] used an X-tree variant to do a nearest neighbor search; they found that the index performed well for fewer than 5 dimensions, and the performance dropped dramatically for just 10–20 dimensions. A third category of algorithms partitions the space into regions to enable a faster search of nearest neighbors. For each region, certain summary statistics such as the minimum bounding rectangle are stored. During the search procedure for finding nearest neighbors of a point, the point is compared with the summary statistics of a region to determine if it is possible to skip all the points in that region. Knorr and

RIGHTSLINK

Ng [1998] found out that the partitioning-based approach has a complexity linear in $O(n)$, but exponential in the number of dimensions.

An efficient algorithm for finding distance-based outliers was devised by Bay and Schwabacher [2003]; it is an adaptation of the simple nested loop algorithm and has an expected running time of $O(n)$. Algorithm 2.2 describes the procedure. The distance function between two points x and y is denoted as $distance(x, y)$, such as Euclidean distance for numerical values and Hamming distance for categorical values. The function $score(x, Y)$ denotes the nearest neighbor distances between x and its neighbors Y and is monotonically decreasing.

Algorithm 2.2 partitions all the points I into a set of blocks $I_1, \ldots, I_b$, and compares each point in every block to every point in I. It keeps track of the top m outliers and the weakest outlier of the m outliers, namely, the point with the smallest score. The main idea in the nested loop is to keep track of the closest k neighbors found so far for each point x. When a point x achieves a score lower than the cutoff c, we can safely remove the point x since x can no longer be an outlier. As the loop continues, the algorithm finds more outliers, and the cutoff threshold c increases along with the pruning power. In the worst case, the algorithm still has a complexity of $O(n^2)$; however, the average case analysis reveals the algorithm has an expected complexity of $O(n)$ [Bay and Schwabacher 2003].

2.3.2 Local Distance-Based Outlier Detection

Distance-based outlier detection methods such as $DB(p, D)$ and its variants may miss certain kinds of outliers because every point is compared with every other point in the dataset.

Example 2.7 Consider a two-dimensional dataset with data points shown in Figure 2.4. There are two clusters of points, C_1 and C_2, and points in C_1 are sparse, whereas points in C_2 are quite dense. Obviously, points o_1 and o_2 should be reported as outliers. However, distance-based methods would not flag o_2 as an outlier before flagging any points in C_1, since the distances between o_2 are much closer to the majority of the data points, i.e., C_2, than those points in C_1.

The local outlier factor (LOF) method [Breunig et al. 2000] scores data points based on the density of their neighboring data points, and can capture outlier points such as o_2 in Figure 2.4.

Given a positive integer k, the k-distance of an object p, denoted as $k\text{-}distance(p)$, is defined as the distance $distance(p, o)$ between p and another object o, such that: (1) there exist at least k objects other than p in the dataset whose distance to p is less than or equal to $distance(p, o)$; and (2) there exist at most $k - 1$ objects other

RIGHTSLINK

Algorithm 2.2 **Randomized algorithm for finding distance-based outliers**

Input: One relational instance I, schema R
Output: $ALLO$, namely, a list of top m outliers
Let $distance(x, y)$ return the distance between x and y
Let $score(x, Y)$ be any function returning the outlier score of x given its neighbor Y
Let $maxDist(x, Y)$ return the maximum distance between x and points in Y
Let $closest(x, Y, k)$ return the k closest points in Y to x
$c \leftarrow 0$ ▷ set the cutoff distance to be 0
$ALLO \leftarrow \emptyset$
partition I into several blocks $I_1, \ldots I_b$
for all $I_i \in \{I_1, \ldots I_b\}$ **do**
  $Neighbors(x) \leftarrow \emptyset$ for all x in I_i
  for all $y \in I$ **do**
    if $|Neighbors(x)| < k$ or $distance(y, x) < maxDist(x, Neighbors(x))$ **then**
      $Neighbors(x) \leftarrow closest(x, Neighbors(x) \cup y, k)$
      if $score(Neighbors(x), x) < c$ **then**
        remove x from I_i
      end if
    end if
  end for
  $ALLO \leftarrow$ the top m outliers from $B \cup ALLO$ based on $score(Neighbors(x), x)$
  $c \leftarrow$ the minimum score for $O \in ALLO$
end for

than p in the dataset whose distance to p is strictly less than $distance(p, o)$. Given $k\text{-}distance(p)$, the k-distance neighborhood of p, denoted as $N_k(p)$, contains all objects other than p whose distance to p is less than or equal to $k\text{-}distance(p)$. The *reachability distance* of an object p with respect to object o is defined as $reach\text{-}dist_k(p, o) = max\{k\text{-}distance(o), d(p, o)\}$. Intuitively, if an object p is too far away from o, then $reach\text{-}dist_k(p, o)$ is the actual distance between p and o. On the other hand, if p and o are sufficiently close, then $reach\text{-}dist_k(p, o)$ is replaced by the k-distance of o. In this way, the distance fluctuations of all objects close to o are reduced; the higher the k, the greater the smoothing effect.

Definition 2.1 The local reachability density of p is defined as:

$$lrd_k(p) = 1/\left(\frac{\sum_{o \in N_k(p)} reach\text{-}dist_k(p, o)}{|N_k(p)|}\right).$$

RIGHTSLINK

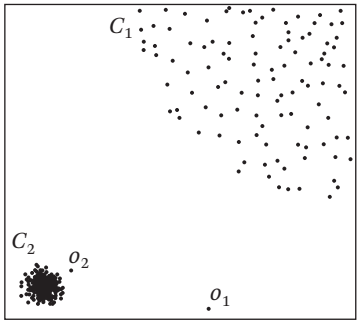


Figure 2.4 An example showing distance-based outlier detection failure [Breunig et al. 2000].

The local outlier factor of p is defined as:

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|}.$$

Intuitively, the local reachability density of an object p is the inverse of the average reachability distance based on the k nearest neighbors of p. The local outlier factor of p is the average of the ratio of the local reachability density of p and those points in the k-distance neighborhood of p. It has been shown [Breunig et al. 2000] that the LOF definition has many desirable properties; for example, $LOF_k(p)$ is approximately equal to 1 in a cluster (a region with homogeneous density around the point and its neighbors), $LOF_k(p) \gg 1$ if p is an outlier.

There are many proposed variants of LOF. Jin et al. [2001] only mine for the top outliers, and thus do not need to compute the LOF scores for all data points by deriving bounds for the LOF scores. Tang et al. [2002] consider connectivity based outlier factors (COF) since LOF may make it hard to detect outliers in low density regions. Jin et al. [2006] take symmetric neighborhood into account by considering both the k-nearest neighborhood of p and its reverse k-nearest neighborhood. Papadimitriou et al. [2003] aim at getting rid of choosing the parameter k by considering all the possible ks.

2.4 Model-Based Outlier Detection

Model-based outlier detection techniques first learn a classifier model from a set of labeled data points, and then apply the trained classifier to a test data point to determine whether it is an outlier. Model-based approaches assume that a classifier

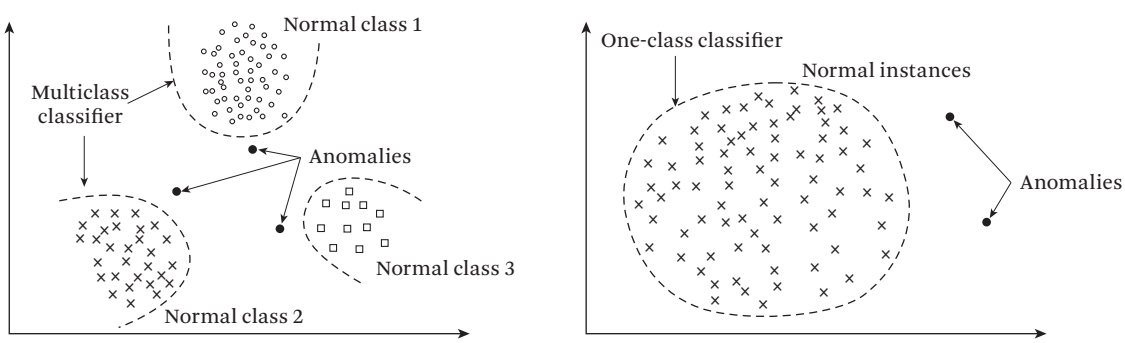
RIGHTSLINK

can be trained to distinguish between the normal data points and the anomalous data points using the given feature space.

Based on the labels available to train the classifier, model-based approaches can be further divided into two subcategories: *multi-class* model-based techniques and *one-class* model-based techniques. Multi-class model-based techniques assume that the training data points contain labeled instances belonging to multiple normal classes [De Stefano et al. 2000]. Multiple classifiers are trained, where each classifier is supposed to distinguish between one of the normal classes and the rest of the classes. A test data point is declared as an outlier if it is not classified as belonging to any of the normal classes by any of the classifiers. In contrast, one-class model-based techniques assume that all the training data points belong to one normal class. Such techniques learn a discriminative boundary to distinguish between all the normal data points and the abnormal data points using a one-class classification algorithm, such as one-class support vector machines (SVMs) [Schölkopf et al. 2001, Ratsch et al. 2002]. Any test data point that does not fall within the learned boundary is declared an outlier.

Example 2.8 Figure 2.5(a) shows an example of a multi-class anomaly detection case where there are three normal classes. Three data points that do not belong to any of the three normal classes are flagged as outliers.

Figure 2.5(b) shows an example case of one-class anomaly detection. The classifier learns a boundary for the normal data points. Data points that lie outside the boundary are flagged as outliers.



(a) Multiclass anomaly detection (b) One-class anomaly detection

Figure 2.5 Model-based outlier detection techniques [Chandola et al. 2009].

RIGHTSLINK

Given that there are many different classification algorithms, such as neural networks, Bayesian networks, SVMs, and decision trees, a variety of model-based outlier detection techniques have been developed. Basic neural networks have been used as a multi-class model-based outlier detection method [Taylor and Addison 2000, De Stefano et al. 2000]. Variants of the basic neural networks have been proposed that use different types of neural networks, such as replicator neural networks [Hawkins et al. 2002]. SVMs have been used as one-class model-based outlier detection techniques. Such techniques learn a region that contains the training data points [Ratsch et al. 2002]. Different kernels, such as Gaussian kernels and radial basis function kernels, can be used to learn complex regions. A test data point is declared as an outlier if it falls outside the boundary. Different variants of the basic SVMs have been proposed for outlier detection in audio signal data [Davy and Godsill 2002] and in temporal data [Ma and Perkins 2003]. Robust support vector machines have also been proposed to detect the boundary in the presence of outliers in the training data [Song et al. 2002]. Rule-based models such as decision trees have also been used for outlier detection [Joshi et al. 2001, Fan et al. 2004]. Each learned rule has an associated confidence value that is based on the number of training normal data points correctly classified by the rule and the total number of training normal data points covered by the rule. If a test point is not captured by any of the learned rules, it is declared an outlier.

2.5 Outlier Detection in High-Dimensional Data

Real datasets can be high dimensional; some may contain hundreds or even thousands of attributes (e.g., the large number of sensor readings in an airplane). Many outlier detection techniques lose their effectiveness when the number of dimensions (attributes) of the dataset is large; this effect is commonly known as the "curse of dimensionality." For statistics-based outlier detection approaches, it becomes increasingly difficult to accurately estimate the multidimensional distribution of the data points [Scott 2008] as the number of dimensions increases. For distance-based approaches, the distances between data points approach zero and become meaningless [Beyer et al. 1999] with increasing dimensionality.

One popular category of techniques to deal with high-dimensional data is by using *dimensionality reduction*, which refers to the process of finding a low-dimensional representation of a high-dimensional dataset that preserves certain properties of the dataset, such as distances or similarities between data points. This topic has been well studied and surveyed in statistics, ML, and data mining [Cunningham and Ghahramani 2015, Fodor 2002, Ding et al. 2008]. Consider a set

RIGHTSLINK

of m real-valued d-dimensional data points represented as an $m \times d$ matrix X, where each row corresponds to a data point $x_i \in \mathbb{R}^d$. The goal of dimensionality reduction is to devise a transformation function $T : \mathbb{R}^d \rightarrow \mathbb{R}^k$ that maps each data point x_i to a new data point in a k-dimensional space ($k < d$), such that the transformed data preserves some properties of interest. We denote by $\tilde{x}_i$ the transformed data point of x_i and by $\tilde{X}$ the transformed dataset. One of the simplest methods to perform dimensionality reduction is through *random projections* [Achlioptas 2001]. To form a new data point $\tilde{x}_i$ from x_i, k random d-dimensional vectors are generated to form a matrix $T \in \mathbb{R}^{k \times d}$. The new dataset $\tilde{X}$ can be computed as $\tilde{X} = XT^T$. It is shown that random projections preserve pairwise distances between vectors [Achlioptas 2001].

Principal Component Analysis (PCA) [Pearson 1901, Hotelling 1933] is perhaps the most popular statistical procedure to perform dimensionality reduction. PCA uses orthogonal transformation to convert a set of data points of possibly correlated dimensions into a set of data points of linearly uncorrelated dimensions, called principal components. The transformation is done in such a way that the first component accounts for the largest possible variance in the data, and each succeeding component has the largest variance possible given that it is orthogonal to all preceding components. PCA can be done by eigenvalue decomposition of the covariance matrix or singular value decomposition (SVD) [Shlens 2014]. To reduce the complexity of computing PCA via SVD, progressive sampling strategies are used [Suri and Bailis 2017].

Dimensionality reduction techniques use all of the available dimensions of the original dataset and aim to find new datasets that preserve certain properties. In what follows, we focus on two additional categories of approaches that discover outliers using a subset of dimensions from the original dataset: subspace outlier detection techniques and contextual outlier detection techniques. Subspace outlier detection techniques select one or more subsets of attributes from all attributes and find outliers with respect to every selected subset of attributes, namely, a subspace [Aggarwal and Yu 2001, Zhang et al. 2004, Muller et al. 2008, Kriegel et al. 2009]. Contextual outlier detection techniques select from all attributes one or more pairs of subsets of attributes, where each pair of subsets consists of a subset of *environmental attributes* (also referred to as *contextual attributes*) and a subset of *behavioral attributes* (also referred to as *indicator attributes or metric attributes*). The environmental attributes are used to select subsets of tuples from all tuples, where each subset of tuples is referred to as a *context*. Outliers are detected within each context with respect to the behavioral attributes [Wei et al. 2003, Zhu et al. 2004, Angiulli et al. 2009, Tang et al. 2013, Liang and Parthasarathy 2016].

RIGHTSLINK

Example 2.9 Consider again the employee records in Table 2.1. We know from Example 2.4 that t_5 is a multivariate outlier with respect to the *income* and *tax* attributes. Example 2.4 assumes that the *income* and *tax* attributes are given as input. However, in reality, we are often only given a table with many attributes, and we have no knowledge of which subsets of attributes would reveal interesting outliers.

Given Table 2.1, a subspace outlier detection algorithm would first identify the *income* and the *tax* attributes as a subspace, and then discover t_5 as an outlier in that subspace.

A contextual outlier detection algorithm would identify *income* to be the environmental attributes and *tax* to be the behavioral attributes. t_5 is then reported as a contextual outlier with respect to the *tax* attribute within the context $income > 100$; in other words, among the tuples with $income > 1000$, t_5 has an abnormal *tax*.

In Example 2.9, the same outlier could be detected by both subspace outlier detection techniques and contextual outlier detection techniques. Subspace outlier detection techniques usually need to enumerate all potentially interesting subspaces. Contextual outlier detection techniques not only need to enumerate possible environmental attributes and behavioral attributes, but also need to enumerate contexts based on the environmental attributes. Contextual outliers are generally more interpretable than subspace outliers (e.g., t_5 in Example 2.9).

We first lay out two commonly found use cases for high-dimensional outlier detections in Section 2.5.1. We then discuss in detail subspace outlier detection techniques in Section 2.5.2 and contextual outlier detection techniques in Section 2.5.3.

2.5.1 Two Use Cases for High-Dimensional Outlier Detection

Techniques for detecting outliers in high-dimensional data often find two general use cases, regardless of whether they are looking for subspace outliers or contextual outliers.

Use Case 1: High-dimensional outlier detection for data exploration: One of the main characteristics of big data exploration tasks, in contrast to querying, is the fact that analysts do not necessarily know what they are looking for. Traditional outlier detection techniques are limited since they require users to specify the interesting attributes. Consider an analyst performing market research who wishes to determine which companies are unusually profitable. Since companies from different sectors may not be comparable in profits, the analyst might answer this query by running traditional outlier detection queries multiple times, each time on a subset of companies belonging to a specific sector (e.g., technology and media) to

RIGHTSLINK

identify the most profitable companies within each sector. There are potentially a large number of interesting subgroups of which technology and media companies are only two possible subgroups. In addition, other grouping criteria (e.g., based on location) might also reveal outliers. Instead of relying on analysts to come up with subspaces and contexts, analysts need tools that can discover subspaces and contexts which contain outliers. In the previous example, while Company X might have "normal" reported profit compared to all technology companies—with normal defined, for example, as being within two standard deviations from the mean—it may have very unusual (outlying) profit when compared to companies with less than 10,000 employees. In this case, if a tool could produce [Business = "technology" $\wedge$ EmployeeCount $<$ 1000], this may be an automatically discovered context of interest to the analyst.

Use Case 2: High-dimensional outlier detection for targeted explanation and diagnosis: Analysts often would like to answer the question "*What is so special about this entity or record?*," a key question in explaining analytics results or diagnosing reported errors or anomalies. For example, an analyst performing troubleshooting at a call center may wish to understand why an important customer in industrial manufacturing is calling the troubleshooting hotline. Using conventional tools, the analyst might check whether a few of the customer's key performance metrics (e.g., quality of service) are abnormal. However, this approach is brittle; for example, high-value clients may naturally exhibit deviations from the overall population. Instead, high-dimensional outlier analysis can take into account other dimensions with the performance metric dimensions to reveal the subspaces or the contexts in which the client is meaningfully outlying. For example, a tool might discover that the client is experiencing an unusually high rate of poor-quality service compared to users in his location using his hardware make and model.

Problem formulations for high-dimensional outlier detection generally fall into one of the two use cases, as we show in the following when we discuss in detail subspace outlier detection techniques and contextual outlier detection techniques.

2.5.2 Subspace Outliers

To better appreciate the challenges in detecting subspace outliers, consider the four different two-dimensional views of a hypothetical dataset in Figure 2.6 as an example [Aggarwal 2013]. We see that Point A is considered an outlier in View 1 and Point B is considered an outlier in View 4, whereas neither point is considered an outlier in View 2 and View 3. The example shows that different data points may be considered outliers in different subspaces. For datasets with high dimensionality, it

RIGHTSLINK

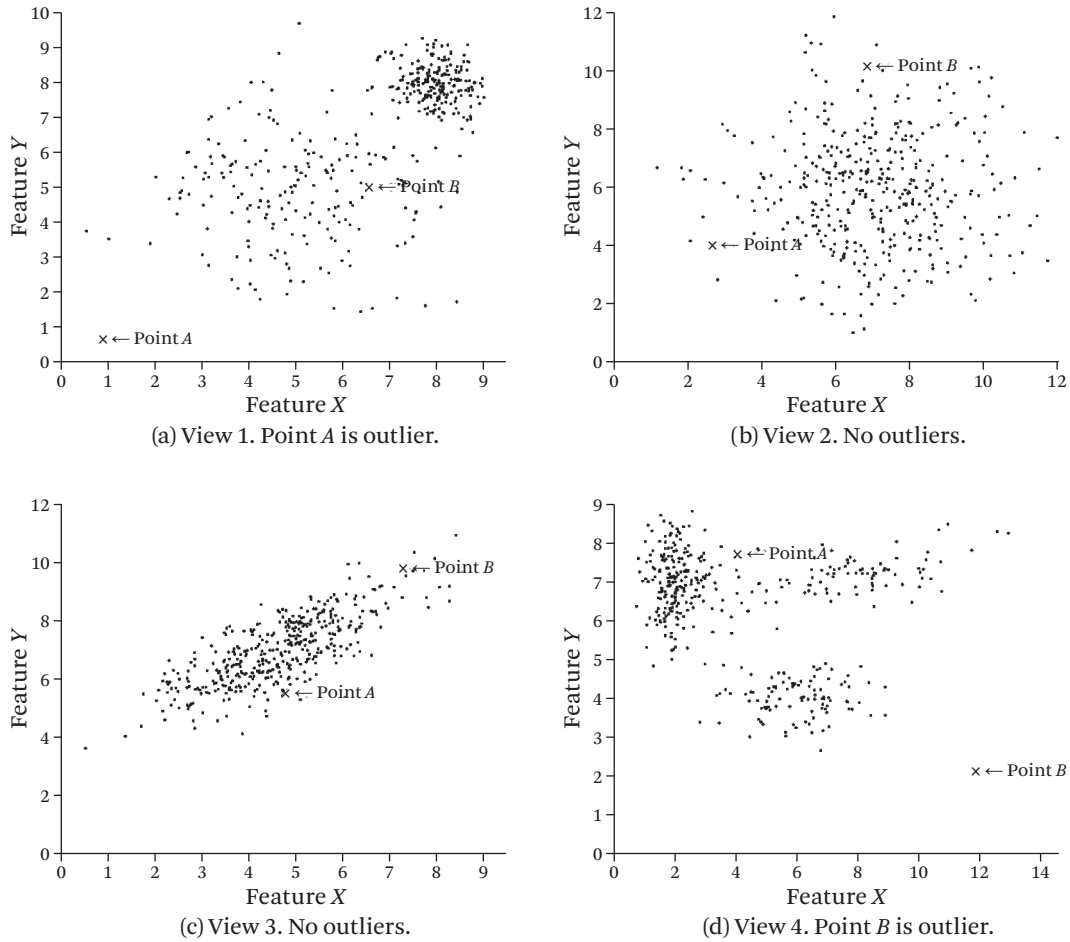


Figure 2.6 The outliers may be present or be lost in different subspaces of the full dimensions [Aggarwal 2013].

is very likely that only a small fraction of all possible subspaces contains interesting outliers.

Detecting outliers in subspaces is a challenging problem mainly due to the fact that the number of possible subspaces is exponential with respect to the number of dimensions in the data. It is worth noting that there have been many proposals for finding clusters in subspaces [Parsons et al. 2004], and many techniques exploit the downward closure properties (a cluster is dense is subspace AB only if it is dense in subspace A and subspace B). Finding outliers in subspaces is even more challenging

RIGHTSLINK

than finding clusters in subspaces. This is because outliers, by definition, are rare events in contrast to clusters, which are dense regions; hence, the downward closure property is usually not applicable to detecting outliers in subspaces. An effective outlier detection method needs to search the subspaces in such a way that the outliers are revealed as quickly as possible. In what follows, we present in detail a first algorithm [Aggarwal and Yu 2001] that achieves this goal, and we briefly discuss other proposals [Zhang et al. 2004, Muller et al. 2008, Kriegel et al. 2009].

Aggarwal and Yu [2001] discover outliers in subspaces by finding *localized regions* of the data in subspaces with abnormally low density. Data points contained in low density regions are identified as outliers. To determine abnormally low density regions in subspaces, a grid-based approach is used to perform a discretization of the data. Each dimension of the data is divided into ϕ ranges of equi-depth, and hence each range contains $f = \frac{1}{\phi}$ of the records. These ranges from one dimension form the localized regions in subspaces. Consider a k-dimensional region that is created by picking grid ranges from k different dimensions. The expected fraction of records in that region is f^k, assuming that the attributes are statistically independent of each other. Of course, the attributes are far from statistically independent of each other, and thus the distribution of points in that region would differ significantly from the expected behavior. It is precisely those regions with abnormally low density that are useful for identifying outliers.

Formally, under the independence assumption, the presence of any data point in a k-dimensional region is a Bernoulli random variable with probability f^k. Therefore, the expected fraction and standard deviation of data points in the k-dimensional region are $N \cdot f^k$ and $\sqrt{N \cdot f^k \cdot (1 - f^k)}$, respectively, where N is the total number of data points. Let $n(D)$ be the actual number of data points in a k-dimensional region D. The *sparsity coefficient* $S(D)$ of D is defined as follows:

$$S(D) = \frac{n(D) - N \cdot f^k}{\sqrt{N \cdot f^k \cdot (1 - f^k)}}.$$

Assuming $n(D)$ fits a normal distribution, the normal distribution tables can be used to quantify the probabilistic level of significance of its deviation. Aggarwal and Yu [2001] aim to find regions with low $S(D)$.

To avoid searching the exponentially large number of regions and computing $S(D)$ for each region in a brute-force manner, evolutionary algorithms are used [Aggarwal and Yu 2001] to select regions with low $S(D)$. In evolutionary methods, every solution or candidate to an optimization is treated as an individual in an evolutionary system. The "fitness" of an individual is exactly the objective function value of the corresponding solution. An evolutionary algorithm uses mechanisms

RIGHTSLINK

Algorithm 2.3 **Evolutionary algorithm for finding outliers in high-dimensional data**

Input: One relational instance I, schema R of d dimensions,
k denoting the number of dimensions of interest in a subspace
Output: *ALLO*, namely, a list of outliers
$S \leftarrow$ initial seed population of p strings
BestSet $\leftarrow$ *null*
while termination condition not met **do**
  $S \leftarrow Selection(S)$
  $S \leftarrow CrossOver(S)$
  $S \leftarrow Mutation(S)$
  update *BestSet* to be the m solutions in *BestSet* $\cup S$ with most negative sparsity coefficients
end while
ALLO $\leftarrow$ set of data points represented by *BestSet*
return *ALLO*

inspired by biological evolution, including mutation, recombination, and selection. Accordingly, the candidate solutions to an optimization problem are mutated, recombined, and selected according to the objective function until a convergence criterion is met. Every candidate solution is usually encoded as a string. Algorithm 2.3 describes the procedure for detecting outliers in a high-dimensional dataset using the evolutionary method. For a d-dimensional dataset, the encoding string will be of length d and contain k specified positions, where k is a user provided input specifying the number of dimensions of interest. The fitness for the corresponding solution is computed using the sparsity coefficient $S(D)$. The evolutionary search procedure starts with a population of p randomly selected solutions and iteratively uses the process of selection, crossover, and mutation until a convergence criterion is met (e.g., after a maximum number of iterations). The selection process works by ranking current solutions according to the $S(D)$, and select higher ranked solution with a higher probability; the crossover process uses a two-point crossover procedure. At the end of the evolutionary algorithm, all data points contained in the final solutions are reported as outliers.

Example 2.10 Consider a dataset with $d = 5$ dimensions in total and $\phi = 10$ ranges for each dimension, and we are interested in subspaces with $k = 3$ dimensions. Every candidate solution will be encoded using a string of length 5, and every position in a string represents which range is selected for every dimension. For instance, a candidate solution, encoded as 3*2*1, represents a region with three dimensions, where the

RIGHTSLINK

first dimension takes the third range, the third dimension takes the second range, and the fifth dimension takes the first range.

Performing a two-point crossover between two solutions 3*2*1 and 1*33* after the third position will result in two new solutions 3*23* and 1*3*1; and the mutation process randomly flips a position to a number between 1 to ϕ with a predefined mutation probability.

Algorithm 2.3 is an example of the first use case (cf. Section 2.5.1). OutRank [Muller et al. 2008] is another example of the first use case, which relies on subspace clustering approaches to find dense clusters in subspaces [Assent et al. 2007, Assent et al. 2008] and treats data points that are not in those dense clusters as outliers. Clusters are frequent objects (as opposed to outliers), and hence are amenable to the downward closure property. The outlyingness of an object is calculated based on how often the object is part of dense clusters, the dimensionality of the dense clusters, and the size of the dense clusters. In OutRank, outliers are side-products of a subspace clustering algorithm, which may not be satisfactory. For example, in a dataset where no subspace clusters are present, all objects may be reported as subspace outliers.

Instead of searching for relevant subspaces first and then finding outliers within those subspaces, HOS-Miner [Zhang et al. 2004] and SOD [Kriegel et al. 2009] are designed to find subspaces in which a given data point is an outlier. For a data point x, HOS-Miner aims at finding all subspaces such that the sum of its k-nearest neighbor distances in that subspace is at least δ. This approach does not normalize the distances with the number of dimensions, and hence a subspace with more dimensions is more likely to be in the output. One advantage of HOS-Miner is that the outlier definition it adopts exhibits the downward closure property—any subspace of a non-outlying subspace is also not outlying and every superset of an outlying subspace is also outlying. Only *minimal* subspaces, in which the given data point x is an outlier, are interesting. HOS-Miner users an X-Tree to perform k-nearest neighbor searches efficiently. HOS-Miner also uses the fixed threshold δ to discern outliers in all subspaces, which could be problematic since these scores are rather incomparable in different subspaces. SOD [Kriegel et al. 2009] also aims at finding subspaces in which a given data point x is an outlier. Given the data point x, a set of reference data points $S(x)$ are determined, which represent the proximity of the current data point x. Based on $S(x)$, the *relevant subspace* for $S(x)$ is determined as the set of dimensions in which the variance is small among points in $S(x)$. If the query point x deviates considerably from the $S(x)$ in the relevant subspace, then x is said to be an outlier in that subspace.

RIGHTSLINK

While subspace outlier detection seems a viable approach for detecting outliers in high-dimensional data, it still faces many challenges: (1) In spite of recent advances, the combinatorial nature of the problem requires more efficient search procedures of possible subspaces. (2) Almost all the current techniques adopt their own definitions of outliers in subspaces mainly for the purpose of effective enumeration or search of potential subspaces. For example, the sparsity coefficient is used as the fitness function in the evolutionary algorithm [Aggarwal and Yu 2001], and HOS-Miner [Zhang et al. 2004] is able to exploit the downward closure property based on its unique definition of subspace outliers. How to efficiently search possible subspaces under a general definition of multivariate outliers remains an open question. (3) A point might be considered an outlier in different subspaces. Therefore, one may combine results from these (incomparable) subspaces and rank subspace outliers effectively. This can also be seen as an opportunity since results from multiple subspaces may indicate more robust outliers.

2.5.3 Contextual Outlier Detection

The notion of contextual outliers was first studied in the context of time-series data [Weigend et al. 1995, Salvador et al. 2004] and spatial data [Shekhar et al. 2001, Kou et al. 2006], where the contextual attribute is the time dimension or the spatial dimension. For example, a certain temperature might not be considered high through the year, but is considered high for the winter months. Recently, many proposals have been made on discovering contextual outliers with general environmental attributes; some target the data exploration use case [Wei et al. 2003, Song et al. 2007, Tang et al. 2013, Liang and Parthasarathy 2016] (cf. Section 2.5.1), whereas some target the explanation and diagnosis use case [Angiulli et al. 2009, Zhu et al. 2004]. For proposals targeting the data exploration case, some assume that the environmental attributes and the behavioral attributes are given as input [Song et al. 2007, Liang and Parthasarathy 2016], while others explore the space of possible environmental and behavioral attributes to identify contextual outliers [Wei et al. 2003, Tang et al. 2013]. The goal of the proposals targeting the explanation and diagnosis use case is to discover the environmental attributes and the behavioral attributes in which a given object is an outlier. In what follows, we give an example algorithm for when the environmental attributes and the behavioral attributes are given [Song et al. 2007], and an example algorithm for when they are not [Wei et al. 2003].

Song et al. [2007] learn a correlation between the provided environmental attributes and the provided behavioral attributes, and define contextual outliers as those data points that deviate from the learned correlation. The correlation is

RIGHTSLINK

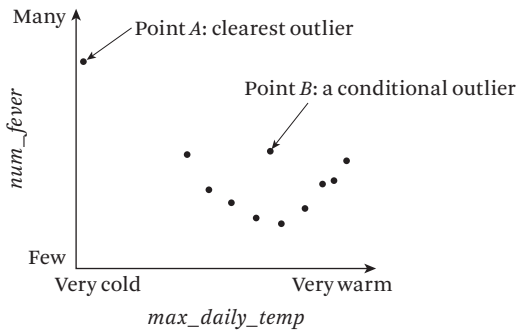


Figure 2.7 Conditional anomaly detection given the environmental attribute *max_daily_temp* and the behavioral attribute *num_fever* [Song et al. 2007].

modeled using a Gaussian mixture model, and the parameters of the model are learned using expectation-maximization methods. Example 2.11 shows an application where the correlation between the environmental attributes and the behavioral attributes can be used to detect interesting contextual outliers.

Example 2.11 Consider the application of detecting a disease outbreak using a dataset with two dimensions: *max_daily_temp*, which denotes the maximum outside temperature on a given day, and *num_fever*, which denotes how many people are admitted to an emergency room due to high fever on a given day. Clearly, *max_daily_temp* is not a direct indicator of disease outbreak; however, it should be taken into account as an environmental attribute when detecting outliers, as *max_daily_temp* directly affects *num_fever*.

A dataset is shown in Figure 2.7. Point *A* is considered as an outlier by most outlier detection methods if we consider *num_fever* alone, since it has an abnormally high number for *num_fever*. However, Point *A* is not interesting for monitoring disease outbreaks, since it is expected that the *num_fever* will be high on a cold day, as is Point *A*.

Point *B* will not usually be considered as an outlier if we consider *num_fever* alone. However, it is an interesting outlier for monitoring disease outbreak, since it has abnormally high *num_fever* for the *max_daily_temp* it has.

The HOT algorithm proposed by Wei et al. [2003] and the COD algorithm proposed by Tang et al. [2013] have explored contextual outliers on categorical datasets, where the environmental attributes and the behavioral attributes are not given. Both approaches consist of two steps: context enumeration and detecting contextual outliers within each enumerated context. Contexts in both approaches

RIGHTSLINK

Algorithm 2.4 **The HOT algorithm for discovering contextual outliers**

Input: One relational instance I, schema R of d dimensions,
k denoting the number of dimensions of interest in a subspace
Output: A list of $\langle t, C\rangle$, denoting t is an outlier in context C.
ResultSet $\leftarrow$ *emptyset*
enumerate all attribute-value pairs from I and R
FrequentContexts $\leftarrow$ frequent contexts using a frequent itemset mining approach
for all Context C in *FrequentContexts* **do**
  behavioralAttributes $\leftarrow$ all attributes in R, but not in C
  for all Each attribute A in *behavioralAttributes* **do**
    compute the histogram of frequencies associated with each value attribute A takes
    for all each tuple t in the context C **do**
      add $\langle t, C\rangle$ to *ResultSet* if $t[A]$ has an abnormally low frequency according to a user defined threshold
    end for
  end for
end for
return *ResultSet*

are essentially conjunctions of predicates, namely, attribute-value pairs, and they are enumerated using a lattice data structure. An example candidate context is $A = a_1 \wedge B = b_1$, which contains all the tuples that have value a_1 for attribute A and have value b_1 for attribute B. The differences between these two approaches lie in the space of contexts they are interested in and how outliers are defined in a certain context. Algorithm 2.4 gives the details of the HOT algorithm as a concrete example. First, all contexts are enumerated and frequent contexts are mined using the Apriori algorithm [Agrawal and Srikant 1994]. For each frequent context C, the histogram of frequencies associated with each attribute A not in C is stored, which is then used to compute the deviation of each value taken by A in the database. Finally, the objects assuming a value on the attribute A whose deviation is smaller than a user-provided threshold are returned as outliers. Example 2.12 shows an example contextual outlier detected by Algorithm 2.4.

Example 2.12 Consider Table 2.2 with four attributes, *Name*, *AgeRange*, *CarType*, and *SalaryLevel*, and ten tuples. An example frequent context is $C : AgeRange = \text{'}Young\text{'}$ with five tuples, namely, t_3, t_5, t_6, t_8, t_{10}. The attribute *CarAttribute* is not in C, and is therefore a potential behavioral attribute. There are two values for *CarAttribute* in the five tu-

RIGHTSLINK

Table 2.2 A example table for contextual outlier detection with categorical attributes

	Name	AgeRange	CarType	SalaryLevel
t_1	Mike	Middle	Sedan	Low
t_2	Jack	Middle	Sedan	High
t_3	Mary	Yong	Sedan	High
t_4	Alice	Middle	Sedan	Low
t_5	Frank	Yong	Sports	High
t_6	Linda	Yong	Sports	Low
t_7	Bob	Middle	Sedan	High
t_8	Sam	Yong	Sports	Low
t_9	Helen	Middle	Sedan	High
t_{10}	Gary	Young	Sports	Low

ples; the value "Sedan" has a frequency of 1, and the value "Sports" has a frequency of 4. Therefore, the value "Sedan" in tuple t_3 is considered as an outlying value in context $C : AgeRange = 'Young'$.

Despite recent advances of contextual outlier detection, it still faces many challenges. (1) Most current contextual outlier detection techniques assume the attributes in the data to be either entirely categorical [Wei et al. 2003, Angiulli et al. 2009, Tang et al. 2013] or entirely numerical [Song et al. 2007, Liang and Parthasarathy 2016]. It remains an open question how to handle mixed types of attributes for contextual outlier detection. (2) Context enumeration is usually expensive to process, and is exponential with respect to the number of environmental attributes. Current techniques treat context enumeration and detecting outliers in every context as two separate processes with little interaction [Wei et al. 2003, Tang et al. 2013]. One future direction is to interleave the two processes to avoid enumerating contexts that do not have outliers without actually running an outlier detection method in them. (3) A data point might be considered to be a contextual outlier by treating different subsets of attributes as environmental attributes or behavioral attributes. In most scenarios, users must investigate outliers reported by an automatic technique. Therefore, we need techniques to filter, rank, and report different contextual outliers discovered so most interesting outliers will be examined first.

RIGHTSLINK

2.6 Conclusion

Outlier detection is a rich topic that has been studied within several communities and application domains, and there exist several surveys and books on this topic [Barnett and Lewis 1994, Hellerstein 2008, Chandola et al. 2009, Aggarwal 2013]. This chapter is complementary to those surveys. Our goal is: (1) to provide a classification of major types of outlier detection techniques, namely, statistics-based methods, distance-based methods, and model-based methods; and (2) to discuss outlier detection in high dimensional data.

Statistics-based outlier detection techniques assume that normal data points would appear in high probability regions of a stochastic model, while outliers would occur in low probability regions of a stochastic model. We distinguish between two different types of statistics-based approaches: the hypothesis testing approach and the distribution fitting approach. The hypothesis testing approach usually calculates a test statistic based on observed data points to determine whether the null hypothesis (there is no outlier in the dataset) should be rejected. The distribution fitting approach, as the name suggests, fits a pdf to the observed data and marks as outliers those points with low probability according to the pdf. Distance-based outlier detection techniques are based on a definition of distances between data points. In general, normal points should be close to each other, and outlying points should be distant from normal points. Distance-based methods can be further divided into global methods and local methods depending on the reference population used when determining whether a point is an outlier. Model-based approaches follow the classic ML paradigm to learn one or more classifiers to distinguish between normal points and outliers. Model-based approaches are dependent on the availability of labeled training data. We further discuss techniques for handling high dimensional data, including dimensionality reduction, subspace outlier detection, and contextual outlier detection.

As discussed, outlier detection techniques define "normal" differently and also make different assumptions about the underlying data set. It is important for practitioners to choose the outlier detection methods that best match their use case. If probabilistic interpretations of results are needed, then statistics-based techniques provide a formal framework for statistical reasoning. If the underlying data follows a known distribution, such as normal distribution, then techniques such as Grubbs' test or *z-score* are applicable, as shown in Section 2.2.2; otherwise, KDE, as shown in Section 2.2.4, provides a nonparametric way for estimating the probability distribution, which can then be used for outlier detection. If probabilistic interpretations of results are not needed or hard to know, then distance-based

RIGHTSLINK

(cf. Section 2.3) and model-based techniques (cf. Section 2.4) provide alternative approaches. Distance-based approaches usually require the user to specify a distance parameter or threshold, which may involve a trial and error process. Model-based approaches usually require labeled training data to train supervised ML models. Unfortunately, there is no gold standard or dominant outlier detection method. Our taxonomy provides a good reference for navigating the vast literature of outlier detection.

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

3 Data Deduplication

In this chapter, we discuss a specific data cleaning task, namely, data deduplication. Duplicate records can occur for many reasons. For example, a customer might be recorded multiple times in a customer database if the customer used different names at the time of purchase; a single item might be represented multiple times in an online shopping site; or a record might appear multiple times after a data integration project because that record had different representations in the original data sources. Data deduplication, also known as duplicate detection, record linkage, record matching, or entity resolution, refers to the process of identifying tuples in one or more relations that refer to the same real-world entity. It is often followed by an entity consolidation or fusion process to find the unified representation for duplicate records that best represents the real-world entity.

Example 3.1 Figure 3.1 illustrates a simple example of data deduplication. The similarities between pairs of records are computed, and are shown in the similarity graph (upper-right graph in Figure 3.1). The missing edges between any two records indicate that they are non-duplicates. An edge between two entities indicate a similarity measure, e.g., the similarity score between P_1 and P_2 in the figure is 0.9. We assume in this example that the score is between 0 and 1, and the higher the score between two entities, the more similar those two entities are.

Records are then clustered together based on the similarity graph. Suppose the user sets the threshold to be 0.5, i.e., any record pairs having similarity greater than 0.5 are considered duplicates. Although P_1 and P_5 have a similarity score less than 0.5, they are clustered together due to transitivity; that is, they are both considered duplicates to P_2.

Finally, all records in the same cluster are consolidated into one record in the final clean relation. Many different strategies can be used to consolidate multiple records. For instance, in consolidating P_1, P_2, and P_5 into one record C_1, majority voting is used to obtain the name and the ZIP attributes, and numerical averaging is used to obtain the income attribute.

RIGHTSLINK

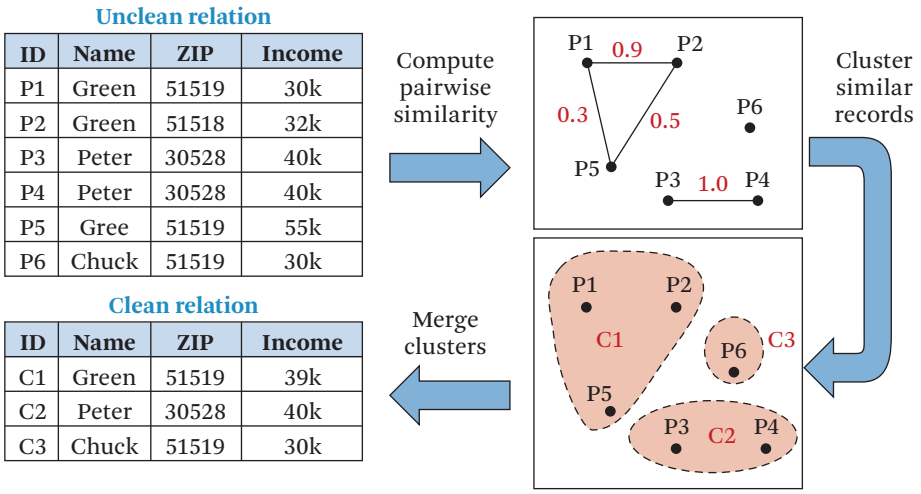


Figure 3.1 A typical deduplication task.

The above simple example uses one similarity function to give a score. Oftentimes, multiple similarity functions are used to produce a similarity score between a pair of entities. Machine learning classifiers or other aggregation methods are used to combine these similarity functions into one score, as shown in Section 3.2.

The topic has been extensively covered in many surveys [Koudas et al. 2006, Elmagarmid et al. 2007, Herzog et al. 2007, Dong and Naumann 2009, Naumann and Herschel 2010, Getoor and Machanavajjhala 2012]: some of the surveys provide an extensive overview of all the steps involved in data deduplication [Elmagarmid et al. 2007, Herzog et al. 2007, Getoor and Machanavajjhala 2012]; some focus on the design of similarity metrics [Koudas et al. 2006, Naumann and Herschel 2010]; some discuss the efficiency aspect of data deduplication [Naumann and Herschel 2010]; and some focus on how to consolidate multiple records [Dong and Naumann 2009].

We cover the various aspects of designing a data deduplication workflow and the different choices available in each aspect. Section 3.1 describes multiple similarity metrics and classifies them into three categories: character-based similarity metrics, token-based similarity metrics, and phonetics-based similarity metrics. Section 3.2 discusses different classifiers used to predict whether a pair of entities are duplicates, including Bayes classifiers and active learning approaches. Section 3.3 surveys different clustering algorithms used to group identified pairs into clusters that represent the same real-world entities as shown in the previous ex-

RIGHTSLINK

ample. Section 3.4 discusses different optimization strategies to reduce the number of comparisons by avoiding comparing record pairs that are unlikely to be matches. Section 3.5 presents different distribution strategies to scale out the data deduplication process in a distributed shared-nothing environment. Section 3.6 provides a classification of different fusion strategies to consolidate multiple records that refer to the same entity into one representation. Section 3.7 explores how humans can be involved in the deduplication system/workflow. Finally, Section 3.8 highlights some open-source and commercial tools for data deduplication.

3.1 Similarity Metrics

Measuring the similarity between two values in the same column is an essential component in determining whether two records are duplicates, and a variety of similarity metrics have been proposed to handle different types of errors and different data types (e.g., numerical data and string data). Obviously, the similarity between two numerical values should be directly dependent on the distance between those two values; for example, one way to define the similarity between two numerical values v_1 and v_2 in attribute A can be defined as the following: $NumSim(v_1, v_2) = 1.0 - \frac{|v_1 - v_2|}{max(A) - min(A)}$, where $max(A)$ and $min(A)$ denote the maximum value and the minimum value in A, respectively. In contrast to numerical values, there are many choices in defining similarities between two string values. As follows, we describe three categories of similarity metrics that deal with three different common errors, namely, character-based similarity metrics, token-based similarity metrics, and phonetics-based similarity metrics.

3.1.1 Character-based

Typographical errors are common and multiple character-based similarity metrics can be used to handle them. The edit distance between two strings s_1 and s_2 is defined as the minimum cost of a sequence of edit operations needed to transform s_1 to s_2. Three types of edit operations are usually considered: inserting a character, deleting a character, and replacing one character with another. Generally, the term "edit distance" is used to refer to the Levenshtein distance [Levenshtein 1966], where the cost of each operation is assumed to be a unit cost.

Definition 3.1 The three types of basic edit operations allowed by edit distance are formally defined as follows:

Insertion. Inserting a character x into string uv generates a new string uxv, where u and v denotes two substrings.

RIGHTSLINK

Deletion. Deleting a character x from string uxv generates a new string uv.

Substitution. Substituting a character x with y in string uxv generates a new string uyv.

Given two string s_1 and s_2, the Levenshtein distance for them is defined as the smallest number of operations that can transform s_1 to s_2.

The edit distance between s_1 and s_2 can be normalized to give a similarity value by $1.0 - \frac{EditDistance(s_1,s_2)}{\max(|s_1|,|s_2|)}$. The most common algorithm to compute the Levenshtein distance between two strings uses dynamic programming, assuming that s_1 has m characters and s_2 has n characters. We use $s_1[i]$ $(1 \leq i \leq m)$ to denote the ith character of s_1 and $s_2[j]$ $(1 \leq j \leq n)$ to denote the jth character of s_2. Let $d(i,j)$ denote the distance between $s_1[1 \ldots i]$ and $s_2[1 \ldots j]$. The recursive structure of the dynamic programming is as follows:

$$d(i,j) = \min \begin{cases} d(i-1,j-1) + \delta(s_1[i], s_2[j]) \\ d(i-1,j) + 1 \\ d(i,j-1) + 1 \end{cases}$$

$$\text{where } \delta(s_1[i], s_2[j]) = \begin{cases} 0 & \text{if } s_1[i] = s_2[j] \\ 1 & \text{otherwise.} \end{cases}$$

Example 3.2 The edit distance between two strings "iPhone 6s"and "iPhone 6" is 1 since it only requires one deletion operation to transform "iPhone 6s" to "iPhone 6", indicating that these two strings are similar.

The edit distance between two strings "Kim" and "Jack" is 4 since it requires three substitutions and one insertion to transform "Kim" to "Jack".

There are many different variants of the basic edit distances. For example, the Hamming distance can be seen as a form of edit distance where only the substitution operation is allowed; hence, it can only be defined for two strings of the same length. Specifically, the Hamming distance for two strings of the same length is defined as the number of positions at which the corresponding characters are different. The Damerau–Levenshtein distance [Bard 2007] adds an edit operation to the basic three types of operations, namely, the transposition operation that swaps two adjacent characters in a string. The Damerau–Levenshtein distance of two strings is defined as the minimum number of operations (insertions, deletions, substitutions, and transpositions) required to transform one string to another. The affine gap distance [Waterman et al. 1976] allows the three basic types of operations, but differentiates between two types of insertions and deletions: opening a

RIGHTSLINK

new gap or extending an existing gap. The cost of opening a new gap is bigger than the cost of extending an existing gap. The Jaro distance [Jaro 1976] and its extension, the Jaro–Winkler distance [Winkler 1990], are also variants of edit distances where only transpositions are allowed; they are specifically designed to match personal names. In the following, we use the affine gap distance and the Jaro distance as two example extensions of the basic edit distance to describe in more detail.

The edit distance falls short when comparing strings that have been truncated or expanded. For example, the two names "Chris R Lang" and "Christopher Richard Lang" should be deemed similar. However, according to the Levenshtein distance, they have a large distance of 126 insertions (or deletions) for aligning "*Chris*" with "*Christopher*" and 6 insertions (or deletions) for aligning "*R*" with "*Richard*". The affine gap distance [Waterman et al. 1976] addresses this problem by introducing two additional edit operations: opening a gap and extending a gap. The cost of opening a gap is usually larger than the cost of extending a gap, which results in a smaller cost for gap mismatches than the cost under the edit distance.

Example 3.3 While the two strings "*Chris R Lang*" and "*Christopher Richard Lang*" have a large distance of 12 using Levenshtein distance, they have a much smaller distance under affine gap distance, and thus are deemed much more similar.

Assume that the affine gap distance has a gap opening cost 1 and gap extending cost 0.5. The two strings would have an affine gap distance of 7. This is because aligning "*Chris*" with "*Christopher*" requires opening a gap and extending that gap by 5 characters, which has a cost of $1 + 5 * 0.5 = 3.5$. Similarly, aligning "*R*" with "*Richard*" also requires opening a gap and extending that gap by 5 characters, which also has a cost of 3.5.

The Jaro distance [Jaro 1976] is usually used for comparing personal names. There are three steps in computing Jaro distance between two strings s_1 and s_2.

1. Compute the length of two strings, denoted as $|s_1|$ and $|s_2|$, respectively.
2. Find the number of matching characters m in s_1 and s_2, where two characters $s_1[i]$ and $s_2[j]$ from s_1 and s_2, respectively, are considered matching if $s_1[i]$ is the same as $s_2[j]$, and they are no more than half the length of the longer string in distance, namely, $|i - j| \leq \frac{1}{2} \max\{|s_1|, |s_2|\} - 1$.
3. Find the number of transpositions t as follows: the lth matching character in s_1 is compared with the lth matching character in s_2. If they are not the same, the number of transpositions is increased by one. The number of transpositions t is then divided by 2 to get the actual number of transpositions.

RIGHTSLINK

Definition 3.2 Given two strings s_1 and s_2, the number of matching characters m, and the number of transpositions t, the Jaro distance of s_1 and s_2 is computed as $Jaro(s_1, s_2) = \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m})$.

Example 3.4 Consider two strings $s_1 =$ "Paul" and $s_2 =$ "Pual." The number of matching characters m is equal to 4. Obviously, $s_1[1]$ is matching $s_2[1]$, and $s_1[4]$ is matching $s_2[4]$. $s_1[2]$ is considered to match with $s_2[3]$ since they are one position apart, which is no more than $frac12 \max\{|s_1|, |s_2|\} - 1 = 1$ apart; similarly, $s_1[3]$ is also considered to match $s_2[2]$. The number of transpositions is $2/2 = 1$. Therefore $Jaro(s_1, s_2) = \frac{1}{3}(\frac{4}{4} + \frac{4}{4} + \frac{4-1}{4}) \approx 0.92$.

The Jaro–Winkler distance [Winkler 1990] uses a prefix scale p to favor two strings that share a common prefix of length l since matching prefixes are generally more important for matching personal family names.

Definition 3.3 The Jaro–Winkler distance of s_1 and s_2 is computed as $Jaro_Winkler(s_1, s_2) = Jaro(s_1, s_2) + (lp(1 - Jaro(s_1, s_2)))$, where l is the length of the prefix up to a maximum of four characters and p is a constant scaling factor that does not exceed 0.25. A standard value for p is 0.1.

3.1.2 Token-based

While character-based similarity metrics are suitable for handling typographical errors, they often fail to capture the similarity between two strings that use the same set of tokens, but with different ordering (e.g., "James Smith" vs. "Smith James"). Multiple token-based similarity metrics can be used to handle such errors, including the overlap coefficient, the Jaccard coefficient, and Dice's coefficient.

Given a string s, we use $tok(s)$ to denote the tokens of s. Different tokenization strategies exist. One type forms tokens from s by separating s using some separators, such as space and hyphen. This type of strategy is insensitive to the location of words, and thus allows for natural word swaps, such as swapping first names with last names. Unfortunately, it does not tolerate spelling errors. For example, "James Smith" and "Smyth Jamas" would have zero similarity.

Another type of popular tokenization strategy using q-grams, that is, splitting a string into shorter substrings of length q using a sliding window, can be used to solve this problem, since spelling errors minimally affect the set of common q-grams of two strings. When $q = 2$, q-grams is commonly referred to as bigrams; when $q = 3$, q-grams is commonly referred to as trigrams. For example, the trigram of the string "Apple" is {"app", "ppl", "ple"}. Given two strings s_1 and s_2 and their to-

RIGHTSLINK

kens $tok(s_1)$ and $tok(s_2)$, three different token-based similarity metrics are formally defined as follows.

Definition 3.4 The overlap coefficient is defined as:

$$sim_{overlap}(s_1, s_2) = \frac{|tok(s_1) \cap tok(s_2)|}{\min(|tok(s_1)|, |tok(s_2)|)}.$$

The Jaccard coefficient is defined as:

$$sim_{jaccard}(s_1, s_2) = \frac{|tok(s_1) \cap tok(s_2)|}{|tok(s_1) \cup tok(s_2)|}.$$

Dice's coefficient is defined as:

$$sim_{dice}(s_1, s_2) = \frac{2 \times |tok(s_1) \cap tok(s_2)|}{|tok(s_1)| + |tok(s_2)|}.$$

Example 3.5 Consider two strings $s_1 =$ "iPhone 6s" and $s_2 =$ "iPhone 6s Plus". Using space as a separator, we have $tok(s_1) = \{$"iPhone", "6s"$\}$, and $tok(s_2) = \{$"iPhone", "6s", "Plus"$\}$. We can calculate the three different similarities:

$$sim_{overlap}(s_1, s_2) = \frac{2}{2} = 1.0$$

$$sim_{jaccard}(s_1, s_2) = \frac{2}{3} = 0.67$$

$$sim_{dice}(s_1, s_2) = \frac{4}{5} = 0.8.$$

3.1.3 Phonetics-based

Phonetics-based similarity metrics are used to detect the similarity of two strings that are phonetically similar, even if they are not similar according to character-based or token-based similarity metrics (e.g., "Clair" versus "Clare"). Soundex [Russell 1918] is the best known phonetic encoding algorithm and was developed to encode surnames in English for use in censuses. It operates with the following steps: (1) it keeps the first letter in a string and converts the rest of the letters into numbers according to Table 3.1; (2) all zeros (the first line in Table 3.1) are then removed and sequences of the same number are reduced to one number only (e.g., "222" is replaced with '2'); and (3) the final encoding string is the original first letter and the subsequent three numbers. If the sequence of the number is longer than three, it is cut off; if it is less than three, it is padded with zeros.

Example 3.6 "Peter" is encoded using Soundex as follows: (1) letter "P" is kept, and the rest of the letters are converted to numbers "0306" according to Table 3.1; (2) removing 0s in

RIGHTSLINK

Table 3.1 Soundex conversion table

a,e,h,i,o,u,w,y	0
b,f,p,v	1
c,g,j,k,q,s,x,z	2
d,t	3
l	4
m,n	5
r	6

"0360" gets us "36"; and (3) we obtain the final encoding "P360" by concatenating the first letter *P* with "36" and padding an additional 0.

According to Soundex, both "Paul" and "Pual" have the same Soundex encoding "P400," and thus are considered to be the same name.

There are several variants of the Soundex encoding scheme. Phonix [Gadd 1990] is a variant of the Soundex that tries to improve the quality of the encoding by preprocessing names according to more than 100 rules; for example, "kn" is replaced with "n" and "wr" is replaced with "r." The transformed string is then encoded using the Soundex encoding procedure. Daitch-Mokotoff Soundex (D-M Soundex) [Steuart and Staff 1994] is a refinement of Soundex to better match surnames of Slavic and Germanic origin.

The New York State Identification and Intelligence System Phonetic Code, commonly known as NYSIIS [Taft 1970], differs from Soundex in that it does not use numerical digits to replace letters; rather, it replaces consonants with other phonetically similar letters.

Metaphone [Philips 1990] uses 16 consonant sounds that can describe a large number of sounds used in many English and non-English words. Double-Metaphone [Philips 2000] recognizes that one string may have multiple different pronunciations, and thus allows for multiple encodings for one string.

3.2 Predicting Duplicate Pairs

In Section 3.1, we described different similarity metrics that can be used to match individual attributes of two records. In real settings, most data records consist of multiple attributes, complicating the data deduplication process. In this section, we discuss how to predict whether or not a pair of tuples are duplicates based on one or multiple similarity scores. The result of the prediction can either be a binary

RIGHTSLINK

variable, which indicates whether a tuple pair is a duplicate or not, or a real number between 0 and 1, which gives a likelihood of a tuple pair being duplicates.

Techniques for predicting duplicate pairs can be classified as unsupervised techniques and supervised techniques; we discuss the latter here. We further discuss how active learning techniques can be beneficial in reducing the number of required labeled examples for data deduplication in Section 7.1.1 when we summarize the use of machine learning techniques in data cleaning.

Unsupervised techniques decide on matching pairs without the need for a training dataset. Examples include using a pre-specified threshold on a distance function [Monge and Elkan 1996, Chaudhuri et al. 2005] and employing domain specific rules [Hernández and Stolfo 1998, Doan et al. 2003, Weis et al. 2008], such as: *If the first name and last name of two persons are similar, then the two records are duplicates.* A rule for records in Table 6.1 could be "*if* $Jaro(t_1[FN], t_2[FN]) > 0.8$ *and* $Jaccard(t_1[CT], t_2[CT]) > 0.5$, *then* t_1 *and* t_2 *are duplicates.*"

Supervised learning techniques rely on a training dataset in the form of record pairs labeled as matching or non-matching. The similarity scores between record pairs serve as features for training a classifier to be applied to the rest of the data. Examples of classifier models include Naïve Bayes [Winkler 1999], decision tree [Chaudhuri et al. 2007], and support vector machine (SVM) [Bilenko and Mooney 2003]. In the following, we describe the first supervised probabilistic approach for data deduplication. Newcombe et al. [1959] first recognized detecting duplicate record as a Bayesian inference problem. Fellegi and Sunter [1969] later formalized the intuition.

Let A and B denote the tables we would like to match (A and B could be the same table). We would like to assign every record pair $\langle \alpha, \beta \rangle (\alpha \in A, \beta \in B)$ into one of two classes M and U, where the class M contains record pairs that represent same real-world entities ("matching") and the class U contains record pairs that represents different real-world entities ("unmatch"). A comparison vector $\gamma = [\gamma_1, \gamma_2, \ldots, \gamma_k]$ that represents the similarity between a record pair $\langle \alpha, \beta \rangle$ in different dimensions is computed; the decision whether $\langle \alpha, \beta \rangle$ is matched or unmatched is made based on the comparison vector. The most straightforward decision rule based on simple probability is as follows:

$$\langle \alpha, \beta \rangle = \begin{cases} M & \text{if } p(M|\gamma) \geq p(U|\gamma) \\ U & \text{otherwise.} \end{cases}$$

This decision rules states that, given the comparison vector γ for a record pair $\langle \alpha, \beta \rangle$, if the probability of the class M is larger or equal to the probability of the class U, then classify $\langle \alpha, \beta \rangle$ as a matched pair. Using Bayes' theorem, the above

RIGHTSLINK

decision rules can be rewritten as:

$$\langle \alpha, \beta \rangle = \begin{cases} M & \text{if } \frac{p(\gamma|M)}{p(\gamma|U)} \geq \frac{p(U)}{p(M)} \\ U & \text{otherwise.} \end{cases}$$

The ratio $\frac{p(\gamma|M)}{p(\gamma|U)}$ is called the likelihood ratio, and the ratio $\frac{p(U)}{p(M)}$ is the threshold value of the likelihood ratio for the decision rule. The above decision rule can be shown to result in the minimum error if the distributions of $p(\gamma|M)$ and $p(\gamma|U)$ and the prior $p(U)$ and $p(M)$ are known [Friedman et al. 2001], which is rarely the case in practice.

One commonly used approach to simplify the computation of the two distributions $p(\gamma|M)$ and $p(\gamma|U)$ is to make a conditional independence assumption between the dimensions in the comparison vector; namely, the probabilities $p(\gamma_i|M)$ and $p(\gamma_j|M)$ are independent if $i \neq j$, and, similarly, the probabilities $p(\gamma_i|U)$ and $p(\gamma_j|U)$ are independent if $i \neq j$. This approach is usually called Naive Bayes. Based on the Naive Bayes assumption, we have $p(\gamma|M) = \Pi_{i=1}^{k} p(\gamma_i|M)$ and $p(\gamma|U) = \Pi_{i=1}^{k} p(\gamma_i|U)$.

If there are training data available, namely, record pairs labeled M or U, we can estimate $p(U)$, $p(M)$, $p(\gamma_i|M)$, and $p(\gamma_i|U)$ easily. We can estimate $p(U)$ (resp. $p(M)$) by counting the percentage of record pairs in the training data with U (resp. M) labels. We can estimate $p(\gamma_i|M)$ and $p(\gamma_i|U)$ by assuming a Gaussian distribution. For example, let μ_M and δ_M^2 be the mean and the variance of the values in dimension i of the comparison vectors of record pairs in the training data with M Then $p(\gamma_i|M)$ can be computed by plugging γ_i into the equation for a Gaussian distribution parameterized by μ_M and δ_M^2. That is, $p(\gamma_i|M) = \frac{1}{\sqrt{2\pi\delta_M^2}} e^{-\frac{(\gamma_i-\mu_M)^2}{2\delta_M^2}}$. If there are small amounts of training data or no training data available, expectation maximization algorithms are usually used to estimate the parameters of the Bayes model [Dempster et al. 1977, Winkler 1993].

The Bayes decision rules lead to optimal results only when the distribution parameters are known. However, in practice, the estimated parameters are often not ideal, and thus classification errors are prevalent, especially when $\frac{p(\gamma|M)}{p(\gamma|U)}$ is close to $\frac{p(U)}{p(M)}$. Fellegi and Sunter [1969] suggested adding another reject class J in addition to classes M and U. The J class contains record pairs for which it is not possible to make a definite inference according to the model and a clerical review is necessary. Record pairs in J are therefore examined by experts to decide whether they are M or U. By setting thresholds for the allowed errors on M and U, a rejection region can be defined.

RIGHTSLINK

3.3 Clustering

So far, we have discussed methods for predicting whether a pair of tuples are duplicates. However, detecting duplicate pairs is not the goal of data deduplication. Indeed, data deduplication needs to find groups or clusters of records that refer to the same real-world entity. In this section, we discuss clustering techniques that achieve this goal.

The result of pairwise comparison process, which takes $O(n^2)$ comparisons for a database of n records, can be represented as a graph, where nodes represent records and edges between nodes exist if they are considered duplicates by the classifiers. Each edge may also have a weight, reflecting the confidence of the two nodes connected by the edge being duplicates; it could be the similarity between the two records, or it could be the probability returned by the classifier. The graph is thus referred to as *similarity graph*. The goal of clustering is to partition all records into disjoint clusters of records, where each cluster corresponds to one real-world entity and records in a cluster are different representations of the same entity [Verykios et al. 2000, Gruenheid et al. 2014]. The clustering algorithms used for data deduplication usually do not require the number of clusters as input, as the number of unique records in a dataset is often unknown. There is no one single perfect clustering algorithm for every dataset. Hassanzadeh et al. [2009] provides a framework for evaluating different clustering algorithms in data deduplication, in terms of scalability, ability to find the correct number of clusters, and robustness of the clustering algorithm. Their findings suggest that none of the clustering algorithms produce perfect results; quantitative information needs to be compared for a specific deduplication task. In the following, we give details about three most commonly used clustering algorithms for data deduplication, namely, transitivity-based clustering, hierarchical clustering, and correlation clustering.

One simple way to obtain such clustering of records is to leverage the transitivity of duplicate records: that is, if Record A is a duplicate of Record B, and Record B is a duplicate of Record C, then Record A is a duplicate of Record C. Then the clustering problem becomes the problem of finding all connected components in the similarity graph. Each connected component is one cluster that represents one real-world entity [Hernández and Stolfo 1995]. It is straightforward to compute the connected components of a graph in linear time (in terms of the numbers of the vertices and edges of the graph) using either breadth-first search or depth-first search. One major drawback of such an approach is that it may mistakenly consider two records as duplicates because they are in the same connected component, even though they are very dissimilar. An example of such a clustering algorithm was shown in Example 3.1.

RIGHTSLINK

Hierarchical clustering is a type of clustering algorithm that seeks to build a hierarchy of clusters. Hierarchical clustering algorithms are either top-down or bottom-up. Bottom-up algorithms treat each object as a singleton cluster at the beginning, and then successively merge pairs of clusters until all clusters have been merged into a single cluster that contains all objects, or a predefined stopping criterion is met. Bottom-up hierarchical clustering is therefore also called hierarchical agglomerative clustering or HAC. Top-down clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual objects are placed into singleton clusters, or a predefined stopping criteria is met. Since HAC is more frequently used than top-down clustering, we give the outline of the HAC in Algorithm 3.1. HAC takes as input a distance function $dist(c_1, c_2)$ that measures the distance between two clusters c_1 and c_2. At every iteration, it chooses clusters that have the smallest distance as determined by the $dist(c_1, c_2)$ function, and merges those two clusters into one cluster. If c_1 and c_2 are singleton clusters, the $dist(c_1, c_2)$ simply uses the similarity score as the distance between those two clusters. If c_1 and c_2 are non-singleton clusters, there are multiple ways to define $dist(c_1, c_2)$. (1) The distance $dist(c_1, c_2)$ is defined as the shortest distance between any two points in each cluster. In this case, the HAC algorithm is called the single linkage clustering algorithm. (2) The distance $dist(c_1, c_2)$ is defined as the longest distance between any two points in each cluster. In this case, the HAC algorithm is called the complete linkage clustering algorithm. (3) The distance $dist(c_1, c_2)$ is defined as the average distance between any two points in each cluster. In this case, the HAC algorithm is called the average linkage clustering algorithm. In its most general form, the HAC algorithm has a time complexity of $O(n^3)$. The time complexity can be reduced to $O(n^2 \log n)$ by using a heap data structure to quickly identify the next two clusters to merge.

Example 3.7 Figure 3.2 shows a scenario of using HAC to cluster five records with different stopping criteria. In the first iteration, clusters $\{r_4\}$ and $\{r_5\}$ have the smallest distance, and thus are merged first. In the second iteration, clusters $\{r_1\}$ and $\{r_2\}$ have the smallest distance, and thus are merged next. In the third iteration, clusters $\{r_3\}$ and $\{r_4, r_5\}$ have the smallest distance, and thus are merged next. Choosing different thresholds τ will lead to different clustering results, as shown in Figure 3.2.

Correlation clustering provides a method for clustering all records into the optimum number of clusters without specifying that number in advance [Elsner and Schudy 2009]. The objective of correlation clustering is to minimize the sum of distances/cost between nodes within the same cluster, and the distances/cost between nodes in different clusters. Correlation clustering can be viewed as an

RIGHTSLINK

Algorithm 3.1 **Hierarchical agglomerative clustering or HAC**

Input: A set of records $r_1, r_2, \ldots r_n$, the similarity graph G, a distance function between two clusters $dist(c_1, c_2)$, distance threshold τ.
Output: A set of clusters C
for all i from 1 to n **do**
  create a new cluster c_i with record r_i
end for
$C = \{c_1, c_2, \ldots, c_n\}$
$dist(c_i, c_j) \leftarrow$ the distance in the similarity graph G
$min_{dist} \leftarrow \infty$
while $C.size > 1$ or $min_{dist} > \tau$ **do**
  $min_{dist} \leftarrow$ minimum distance between any two clusters c_{min1} and c_{min2} in C
  remove c_{min1} and c_{min2} from C
  $c_{merge} = c_{min1} \cup c_{min2}$
  add c_{merge} to C
end while
return C

integer linear programming (ILP) problem. Let $e_{xy} \in \{0, 1\}$ denote if two records x and y are in the same cluster; then $w^+_{xy} \in [0, 1]$ denote the cost of clustering x and y together, and $w^-_{xy} \in [0, 1]$ denote the cost of placing x and y in two different clusters. Thus, correlation clustering can be formalized as follows:

$$\text{minimize} \sum_{xy} (e_{xy} w^+_{xy} + (1 - e_{xy}) w^-_{xy}) \qquad \text{subject to } \forall x, y, z, e_{xy} + e_{xz} + e_{yz} \neq 2.$$

The objective function in the above formulation is exactly the sum of the cost between every pair of records in the dataset, whether they are in the same cluster (w^+_{xy}) or in different clusters (w^-_{xy}). As an example cost function, w^+_{xy} can be defined as the similarity between x and y, and w^-_{xy} can be defined as 1.0 minus the similarity. The constraint $\forall x, y, z, e_{xy} + e_{xz} + e_{yz} \neq 2$ states that for every three records, it cannot be the case that two record pairs end up in the same cluster while the third record pair are in a different cluster; this ensures that the transitivity property holds in the final result of the clustering output.

Since solving the ILP problem is NP-hard [Ailon et al. 2008], a number of greedy approaches have been proposed [Ailon et al. 2008, Elsner and Schudy 2009] that generally work in the following steps.

1. All records are randomly permuted.

RIGHTSLINK

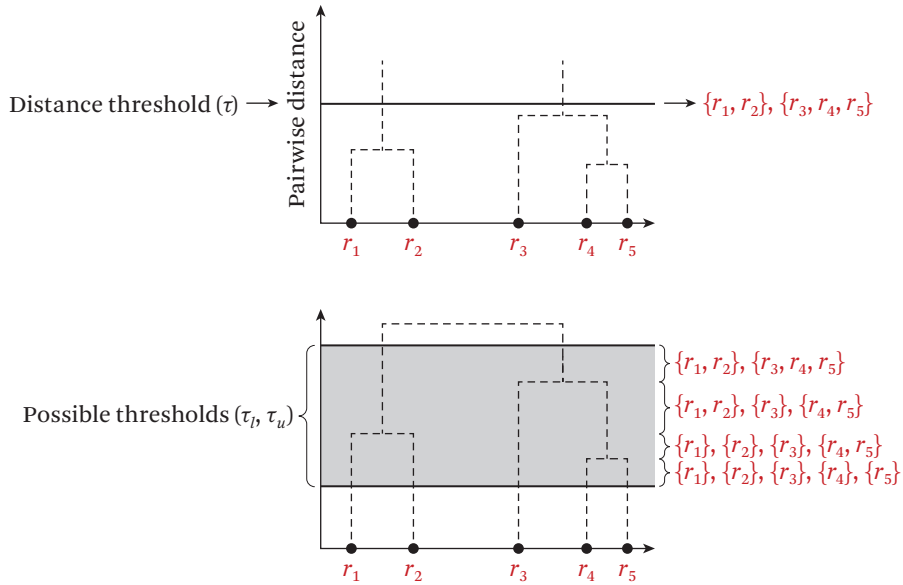


Figure 3.2 An illustration of hierarchical agglomerative clustering or HAC with different stopping criteria.

2. Each record x is either assigned to an existing cluster or a new cluster, according to a certain criterion, such as assigning x to a cluster that contains the closest match to x [Ng and Cardie 2002], assigning x to a cluster that contains the most recent record y with $w^+_{xy} > 0$ [Soon et al. 2001], and assigning x to a cluster that minimizes the objective function [Elsner and Charniak 2008].
3. Run the greedy approach for multiple times, and choose the run that results in the best objective value.

Both hierarchical clustering and correlation clustering do not require the number of clusters to be specified in advance, which is a big advantage. Compared with hierarchical clustering, correlation clustering techniques usually produce relatively high accuracy results, but at the cost of scalability.

3.4 Blocking for Deduplication

The number of tuple pairwise comparisons required for a dataset of n records is $O(n^2)$, which is expensive for large n. Blocking techniques or blocking functions aim at reducing the number of comparisons by avoiding comparing tuple pairs that are unlikely to be matches. Given an input dataset instance I, blocking techniques

RIGHTSLINK

or blocking functions produce a set of *blocks* $\{C_1, C_2, \ldots, C_k\}$, where $C_i \subset I$ and $\cup_{i=1}^{k} C_i = I$. Tuple pairs within the same block get compared, while tuple pairs across different blocks are skipped. Depending on the blocking techniques, the set of blocks may be disjoint or overlapping. Sometimes, after a set of blocks is produced, they are further consolidated via a *meta-blocking* process to further reduce the number of comparisons [Papadakis et al. 2014], which retains tuple pairs that appear in multiple blocks and discards tuple pairs that appear in few blocks.

Example 3.8 Consider a dataset that has 1 million records, where each record stores the name of a restaurant and the city of the restaurant. Assume there are 1,000 unique cities and each city has 1,000 restaurants; hence, there are 1 million records in total. This dataset would require about $\frac{10^6 \times 10^6}{2}$ tuple pair comparisons. Assuming each comparison takes 1 μs, it would take about 5.78 days.

Since domain knowledge suggests that restaurants from different cities are unlikely to be matches, the records can be partitioned into 1000 blocks, where each block contains 1000 restaurants from one city. The number of comparisons required after blocking is $1000 \times \frac{10^3 \times 10^3}{2}$, which would only take 20.8 min.

Although blocking techniques can significantly reduce the number of tuple pair comparisons, it might also miss true matches if they never appear together in a block. Therefore, there is a natural trade-off between the number of reduced comparisons and the number of missed matches when designing blocking techniques. Multiple metrics can be used to quantify this trade-off. The *reduction ratio* of a blocking function is defined as the number of tuple pair comparisons induced by the set of blocks divided by the total number of tuple pair comparisons before blocking. The *recall* of a blocking function is defined as the number of true matches compared in the set of blocks divided by the total number of true matches in the dataset. The *precision* of a blocking function is defined as the number of true matches compared divided by the total number of matches compared. Since computing the precision and the recall of a blocking function requires the ground truth of true matches in a dataset, which is often not available, they are computed by involving humans to verify matches in a sample of the dataset.

There are multiple surveys and studies summarizing existing blocking technique [Christen 2012, Papadakis et al. 2016], which are mainly built using two types: hash-based and similarity-based. Hash-based techniques involve judicious use of hash functions to place records into blocks, where each block is associated with a unique hash key; they usually produce disjoint blocks. Similarity-based techniques cluster nearby records together according to a similarity metric; they usually produce overlapping clusters. We discuss them in detail as follows.

RIGHTSLINK

3.4.1 Hash-Based Blocking

We discuss two types of different hash-based blocking techniques. The first is based on the equality of hash values, and the second is based on locality sensitive hashing, which hashes input records such that similar records are mapped into the same block with high probability.

Equality-based

A simple way to perform blocking is to scan all the records and compute the hash value of a hash function for each record based on some attributes, commonly referred to as *blocking keys*. Those records with the same hash value are placed within the same block. Examples of blocking keys are the first three characters of the last name, and the concatenation of city, state, and zip attribute. Although blocking can substantially increase the comparison efficiency, it can result in many false negatives when two duplicate records do not agree on the blocking key, and thus reside in two different blocks. One way to alleviate such a problem is to perform the deduplication algorithm in multiple passes, using a different blocking criterion for each pass; another approach is to build a complex blocking function that is a combination of multiple blocking criteria [Michelson and Knoblock 2006, Sarma et al. 2012].

Locality Sensitive Hashing

Given a specific similarity function $Sim : U \times U \rightarrow [0, 1]$, a locality sensitive hashing (LSH) scheme is a probability distribution over a set $\mathcal{H}$ of hash functions such that $Pr_{h \in \mathcal{H}}[h(A) = h(B)] = Sim(A, B)$ for two objects $A, B \in U$ [Charikar 2002]. LSHs represent similarities between records using probability distributions over hash functions. For a particular similarity function, there may or may not exist an LSH scheme.

One of the most popular LSH schemes is the MinHash scheme (or min-wise independent LSH scheme) [Broder 1997], which is a technique for estimating how similar two sets A and B are in terms of their Jaccard similarity $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. Let h be a hash function that maps the members of A and B to distinct integers, and for any set S, let $h_{min}(S)$ be the member x of S with the minimum value of $h(x)$. Applying h_{min} to both A and B, and assuming no hash collisions, we will get the same value exactly when the element with the minimum hash value in $A \cup B$ is also in the intersection $A \cap B$, namely $Pr[h_{min}(A) = h_{min}(B)] = J(A, B)$. If we use k independent hash functions, and let y be the number of hash functions for which $h_{min}(A) = h_{min}(B)$, then we can use $\frac{y}{k}$ as an unbiased estimator for $J(A, B)$. In what follows, we introduce how to design blocking strategies based on MinHash [Leskovec et al. 2014].

RIGHTSLINK

Step 1. Since Jaccard similarity is based on sets, the first step is to map every record into a set of elements, a process also known as "shingling." There are two main ways to shingle a string record into a set: word-level shingling and letter-level shingling. For example, a string "John Computer Science PhD" is mapped into a set {John Computer, Computer Science, Science PhD} using word-level shingling with shingle size (i.e., number of words in an element) 2; and a string "science" is mapped into a set {sci, cie, ien, enc, nce} with shingle size 3.

Step 2. The second step is to create a *MinHash signature* for every record t using k independent hash functions $h^1, h^2, \ldots, h^k$. The MinHash signature for a record t is a vector $\langle h^1_{min}(t), h^2_{min}(t), \ldots h^k_{min}(t)\rangle$, where $h^i_{min}(t)$ denotes the minimum hashing value any element in t has applying the hashing function h_i. If the final goal is to compute the Jaccard similarity for every record pair, we need to count how many elements two MinHash signatures have in common for every record pair (although parallel computation can reduce computation time). However, often, we only would like to compare most similar records, or records whose similarities are above a certain threshold.

Step 3. The final step is to create blocks such that similar records are more likely to be placed in the same block based on the MinHash signatures. A straightforward approach is to directly use the MinHash signatures to create blocks: two records are in the same block if there is at least one common element in their MinHash signatures. This approach can create many false positives, many non-similar records will end up in the same block. A better approach is to put two records in the same block if they have multiple elements in common in their MinHash signatures. To achieve this, the k-length signature is divided into b bands, where each band contains $\frac{k}{b}$ elements [Leskovec et al. 2014]. Two records are placed in the same block if there is at least one common band of elements. Given b bands for a signature and two records with a Jaccard similarity s, we can calculate the probability that these two records end up in the same block as follows.

1. The probability that two signatures agree in all elements of one band is $s^{\frac{k}{b}}$.
2. The probability that two signatures disagree in at least one element of a particular band is $1 - s^{\frac{k}{b}}$.
3. The probability that two signatures disagree in at least one element of all bands is $(1 - s^{\frac{k}{b}})^b$, which is the probability two records will not be placed in the same block.
4. The probability that two records end up in the same block is $1 - (1 - s^{\frac{k}{b}})^b$.

With a reasonable choice of the parameters s, k, and b, users can place similar records into the same block with high probability. For example, with $s = 0.8$, $k =$

RIGHTSLINK

100, and $b = 20$, the probability that two records end up in the same block is as high as 0.9996.

3.4.2 Similarity-based Blocking

Similarity-based blocking techniques place records into different blocks based on the desired notion of similarity between two records. We discuss two approaches in this category: windowing methods and prefix-filtering methods.

Windowing Methods

Windowing methods, also known as sorted neighborhood approaches [Hernández and Stolfo 1995, Hernández and Stolfo 1998], aim at grouping similar records together according to some *sorting key*, where a sorting key for a record is extracted from fields or portions of fields of that record. The records in a dataset are then sorted according to the sorting keys. A fixed size window is moved through the sorted list of records, and every window constitutes a cluster. Given a window size of w records, then every new record that enters that window is compared with the previous $w - 1$ records. For a dataset of $n(> w)$ records, there will be $n - w + 1$ clusters and there will be $(w - 1) \times (n - w + 1)$ total number of comparisons. The windowing methods rely on the assumption that duplicate records are close to each other in the sorted list according to the sorting key. It can be seen that the effectiveness of the windowing methods is highly dependent on the sorting keys. Often, a single key is not sufficient to place all duplicate record pairs in the same window. Thus, similar to blocking methods, multiple passes based on different ordering keys can be employed.

Prefix Filtering

Prefix-filtering methods [Chaudhuri et al. 2006, Bayardo et al. 2007, Wang et al. 2012b] use an efficient prefix-filter to filter out record pairs that cannot be similar given a similarity metric and a threshold, and verify the rest of the record pairs by computing their real similarities. They consist of three steps:.

Step 1. Each record is treated as or transformed into a set of elements, and the given similarity threshold is mapped to how many elements two sets need to have in common, called the *overlap similarity*. Given two records r, s, a similarity function sim, and a threshold θ, if $sim(r, s) \geq \theta$, then the overlap similarity must be at least τ. For Jaccard similarity, given $\frac{|r \cap s|}{|r|+|s|-|r \cap s|} \geq \theta$, we have $|r \cap s| \geq \theta(|r| + |s| - |r \cap s|)$. Since $|s| \geq |r \cap s|$, we have $|r \cap s| \geq \theta|r|$. Hence, $\tau = \theta|r|$. Note that the prefix filtering techniques work only when all the records have the same number of elements, i.e., $|r|$ is a constant value.

RIGHTSLINK

Step 2. The elements of every record are sorted according to a fixed global ordering of all elements from all records, and a prefix set of elements is selected for every record based on the overlap similarity so that two records are similar only if their prefixes overlap. Let $Prefix(r)$ be the prefix set of $|r| - \tau + 1$ elements in the ordered set of elements of r.

Theorem 3.1 If $|r \cap s| \geq \tau$, then their prefixes must overlap: $Prefix\ (r) \cap Prefix\ (s) \neq \emptyset$.

Proof This can be proven by contradiction. Let $Rest(r)$ be all the elements in r excluding the $Prefix(r)$, and let $Rest(s)$ be defined similarly; we have $|Rest(r)| = |Rest(s)| = \tau - 1$. Assume that $Prefix(r) \cap Prefix(s) = \emptyset$; then the overlap can only happen either between $Prefix(r)$ and $Rest(s)$, between $Prefix(s)$ and $Prefix(r)$, or between $Rest(r)$ and $Rest(s)$. We analyze two cases.

Case 1. If $Prefix(r) \cap Rest(s) = \emptyset$, then $Prefix(r)$ does not have any overlap with elements in s. Since $Rest(r) = \tau - 1$, then there is at most $\tau - 1$ overlap between elements in r and s, which contradicts $|r \cap s| \geq \tau$.

Case 2. If $Prefix(r) \cap Rest(s) \neq \emptyset$, let the largest element in $Prefix(r) \cap Rest(s)$ be x; then $Rest(r) \cap Prefix(s)$ must be empty, since elements in $Rest(r)$ are greater than x and elements in $Prefix(s)$ are less than x. Now we know that $Rest(r) \cap Prefix(s)$ and $Prefix(r) \cap Prefix(s)$, then $Prefix(s)$ does not have any overlap with elements in r. Thus, all overlap involving elements of s must be in $Rest(s)$. Since $Rest(s) = \tau - 1$, then there is at most $\tau - 1$ overlap between elements in s and r, which contradicts $|r \cap s| \geq \tau$. ■

Step 3. Inverted indexes are built to quickly filter out those record pairs whose prefix sets of elements do not overlap, instead of enumerating all record pairs. An inverted index maps an element to a list of records containing that element. The inverted indexes only need to be built for all elements appearing in prefix sets of elements. The inverted indexes can then be used to directly emit record pairs whose prefix sets overlap as follows. If the task is to perform data deduplication on one collection of records R, then we can impose an ordering of records in R; only records ranked higher need to be compared with record ranked lower to avoid enumerating every record pair twice. For each $r \in R$, to obtain other objects $s \in R$ such that $Prefix\ (r) \cap Prefix\ (s) \neq \emptyset$, we only need to merge the records in the inverted indexes of elements in $Prefix(r)$, and then filter out those records whose rank is lower than r. If the task is to perform record linkage between two collections of records R and S, then the inverted indexes are built for records in S only. For every $r \in R$, to obtain other objects $s \in S$ such that $Prefix\ (r) \cap Prefix\ (s) \neq \emptyset$, we only need to merge the records in the inverted indexes of elements in $Prefix\ (r)$.

RIGHTSLINK

r_1	e_5, e_6, e_2, e_3, e_9
r_2	e_3, e_6, e_7, e_8, e_9
r_3	e_7, e_8, e_9, e_2, e_4
r_4	e_1, e_5, e_4, e_8, e_9
r_5	e_5, e_6, e_7, e_8, e_1

(a) Input R

Prefix(r_1)	e_2, e_3
Prefix(r_2)	e_3, e_6
Prefix(r_3)	e_2, e_4
Prefix(r_4)	e_1, e_4
Prefix(r_5)	e_1, e_5

(b) Prefixes

$e_1 \rightarrow r_4, r_5$
$e_2 \rightarrow r_1, r_3$
$e_3 \rightarrow r_1, r_2$
$e_4 \rightarrow r_3, r_4$
$e_5 \rightarrow r_5$
$e_6 \rightarrow r_2$

(c) Inverted indexes

Figure 3.3 A prefix filtering example [Wang et al. 2012b].

Example 3.9 Consider a data deduplication task on the input table R in Figure 3.3(a). Each record in R has a set of five elements. Suppose we consider a record pair as duplicates if their Jaccard similarity is greater than $\theta = 0.8$.

In **Step 1**, we obtain the threshold $\tau = \theta \times 5 = 4$. In other words, 2 records need to have at least 4 elements in common for their Jaccard similarity to exceed 0.8.

In **Step 2**, we build prefixes for every record in R. We impose a global ordering of all elements; in this case, we choose the ordering $e_1 < e_2 < e_3 < e_4 < e_5 < e_6$. For every record in R, we sort all elements according to the ordering, and we select a prefix of $5 - 4 + 1 = 2$ elements. The prefixes for all records are shown in Figure 3.3(b).

In **Step 3**, we build inverted indexes for all elements appearing in prefixes, as shown in Figure 3.3(c). For every record r_i, to obtain records r_j such that $j > i$ and $Prefix(r_i) \cap Prefix(r_j) \neq \emptyset$, we merge all records in the inverted indexes of elements in $Prefix(r_i)$. For r_1, we merge the inverted indexes of e_2 and e_3, which gives $\{r_1, r_2, r_3\}$. Since we only need to consider record r_j where $j > 1$, r_1 only needs to be compared with r_2 and r_3. In the end, we obtain all the record pairs that need to be compared using this approach: $\langle r_1, r_2\rangle$, $\langle r_1, r_3\rangle$, $\langle r_3, r_4\rangle$, $\langle r_4, r_5\rangle$. This gives only four record pairs for which we need to calculate the Jaccard similarity, which is significantly less than the original ten record pairs.

3.5 Distributed Data Deduplication

Despite the use of blocking techniques, data deduplication remains a costly process that can take hours to days to finish for real-world datasets on a single machine [Köpcke et al. 2010]. Most of the work on data deduplication is situated in a centralized setting [Ananthakrishna et al. 2002, Bilenko et al. 2006, Christen 2012] and does not leverage the capabilities of a distributed environment to scale out computation; hence, it does not scale to large distributed data.

RIGHTSLINK

This section describes distributed techniques that parallelize data deduplication. Multiple challenges need to be addressed to achieve this goal. First, unlike centralized settings, where the dominating cost is almost always computing the similarity scores of all tuple pairs, multiple factors contribute to the elapsed time in a distributed computing environment, including network transfer time, local disk I/O time, and CPU time for pairwise comparisons. These costs also vary across different deployments. Second, as it is typical in a distributed setting, any algorithm has to be aware of data skew, and achieve load-balancing [Beame et al. 2014, DeWitt et al. 1992]. Every machine must perform a roughly equal amount of work in order to avoid situations where some machines take much longer than others to finish, a scenario that greatly affects the overall running time. Third, the distribution strategy must be able to handle effectively multiple blocking functions; the use of multiple blocking functions impacts the number of times each tuple is sent across nodes, and also induces redundant comparisons when a tuple pair belongs to the same block according to multiple blocking functions.

DEDOOP [Kolb et al. 2012a, Kolb et al. 2012b] uses MapReduce [Dean and Ghemawat 2008] for data deduplication. It optimizes for computation cost and requires a large memory footprint to keep the necessary statistics for its distribution strategy, thus limiting its performance and applicability. DISDEDUP [Chu et al. 2016] considers both the communication cost and the computation cost, and aims at minimizing elapsed time by minimizing the maximum cost across all machines. We describe in detail the DISDEDUP strategy.

3.5.1 Parallel Computation Model

Since all workers or machines are running in parallel, to minimize the overall elapsed time, DISDEDUP focuses on minimizing the largest cost across all workers. For worker i, let X_i be the communication cost and Y_i be the computation cost. Assume that there are k workers available. DISDEDUP defines X (resp. Y) to be the maximum X_i (resp. Y_i) at any worker:

$$X = \max_{i \in [1,k]} X_i \qquad Y = \max_{i \in [1,k]} Y_i. \tag{3.1}$$

Example 3.10 Consider a scenario where a single blocking function produces few large blocks and many smaller blocks. To keep the example simple, suppose that a blocking function partitions a relation of $n = 100$ tuples into 5 blocks of size 10 and 25 blocks of size 2. The total number of comparisons W in this case is $W = 5 \cdot \binom{10}{2} + 25 \cdot \binom{2}{2} = 250$ comparisons.

RIGHTSLINK

Assume $k = 10$ workers. Consider first a strategy that sends all tuples to every worker. In this case, $X_i = 100$ for every worker i, which results in $X = 100$ according to Equation (3.1). We then assign $Y_i = \frac{W}{k} = 25$ comparisons to worker i (for example by assigning to worker i tuple pairs numbered $[(i-1)\frac{W}{k}, i\frac{W}{k}]$). Therefore, $Y = 25$ according to Equation (3.1). This strategy achieves the optimal Y, since W is evenly distributed to all workers. However, it has a poor X, since every tuple is replicated 10 times.

Consider a second strategy that assigns one block entirely to one worker. For example, we could assign each of the 5 blocks of size 10 to the first 5 workers, and to each of the remaining 5 workers we assign 5 blocks of size 2. In this case, $X = 10$, since each worker receives exactly the same number of tuples; moreover, each tuple is replicated exactly once. However, even though the input is evenly distributed across workers, the number of comparisons is not. Indeed, the first 5 workers perform $\binom{10}{2} = 45$ comparisons, while the last 5 workers perform only $5 \cdot \binom{2}{2} = 5$ comparisons. Therefore, $Y = 45$ according to Equation (3.1).

The above example demonstrates that the distribution strategy has significant impact on both X and Y, even in the case of a single blocking function. Consider a blocking function h that produces m blocks $B_1, B_2, \ldots, B_m$. A distribution strategy would have to assign, for every block B_i, a subset of the k workers of size $k_i \leq k$ to handle B_i.

A straightforward strategy assigns one block entirely to one worker, i.e., $k_i = 1, \forall i \in [1, m]$; hence, parallelism happens only across blocks. Another straightforward strategy uses all the available workers to handle every block, i.e., $k_i = k, \forall i \in [1, m]$, so that parallelism is maximized for every block. Any existing parallel join algorithm [Afrati and Ullman 2010, Okcan and Riedewald 2011] can be used to handle every block. However, both strategies are not optimal.

In light of these two straightforward strategies, DisDedup first studies how to distribute the workload of one block B_i to k_i workers to minimize X and Y (Section 3.5.2). Given the distribution strategy for a single block, DisDedup then shows how to assign workers to blocks $B_1, \ldots, B_m$ generated by a single blocking function h, so as to minimize both X and Y across all blocks (Section 3.5.3). Given the distribution strategy for a single blocking function h, DisDedup finally presents how to assign workers given multiple blocking functions $h_1, \ldots, h_s$, so that the overall X and Y are minimized (Section 3.5.4).

3.5.2 Self-Join

For any distribution strategy that performs data deduplication on a block of size n using k reducers, DisDedup derives the lower bounds for the maximum input

RIGHTSLINK

$X > X_{low} = \frac{n}{\sqrt{k}}$ and the lower bound for the maximum number of comparisons $Y \geq Y_{low} = \frac{n(n-1)}{2k}$. DisDedup adopts a distribution strategy, called *triangle distribution*, which guarantees with high probability a small constant-factor approximation of the lower bounds.

The name of the distribution strategy comes from the fact that DisDedup arranges the k reducers in a triangle whose two sides have size l (thus $k = l(l+1)/2$ for some integer l). To explain why DisDedup organizes the reducers in such a fashion, consider the scenario studied in Afrati and Ullman [2010], Beame et al. [2014] that computes the Cartesian product $R \times S$ of two relations of size n: in this case, the reducers are organized in a $\sqrt{k} \times \sqrt{k}$ square, as shown in Figure 3.4(a) for $k = 36$. Each tuple from R is sent to the reducers of a random row, and each tuple from S is sent to all the reducers of a random column; the reducer function then computes all pairs it receives. However, if we apply this idea directly to a self-join (where $R = S$), the comparison of each pair would be repeated twice, since if a tuple pair ends up together in the reducer (i, j), it will also be in the reducer (j, i). For example, in Figure 3.4(b) tuple t_1 is sent to all reducers in row 2 and column 2, and tuple t_2 is sent to all reducers in row 4 and column 4. Therefore, the joining of t_1 and t_2 is duplicated at reducers $(2, 4)$ and $(4, 2)$. Because of symmetry, the lower left half of the reducers in the square are doing redundant work. Arranging the reducers in a triangle circumvents this problem and allows us to use all available reducers. Figure 3.4(b) gives an example of such an arrangement for $k = 21$ reducers with $l = 6$. Every reducer is identified by a two-dimensional index (p, q), where p is the row index and q is the column index, and $1 \leq p \leq q \leq l$. Each reducer (p, q) has a unique reducer ID, which is calculated as $(2l - p + 2)(p - 1)/2 + (q - p + 1)$. For example, Reducer $(2, 4)$ marked purple in Figure 3.4(b) is Reducer 9.

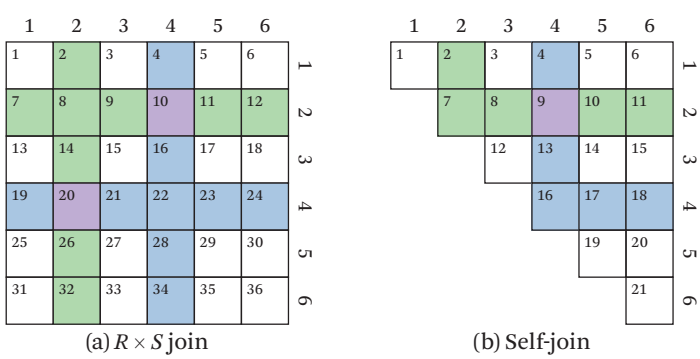


Figure 3.4 Reducer arrangement. (The number in the upper left corner of each cell is the reducer ID.)

RIGHTSLINK

For any tuple t, the mapper randomly chooses an integer a, called an *anchor*, between $[1, l]$, and distributes t to all reducers whose row or column index $= a$. By replicating each tuple l times, for every tuple pair, there must exist at least one reducer that receives both tuples. In fact, if two tuples have different anchor points, there is *exactly one* reducer that receives both tuples, while if two tuples have the same anchor point a, both tuples will be replicated on the same set of reducers, but DisDedup only compares the tuple pair on the reducer (a, a). The key of the key-value pair of the mapper output is the reducer ID, and the value of the key-value pair of the mapper output is the tuple augmented with a flag L, S, or R to avoid comparing tuple pairs that have the same anchor points a in reducers other than (a, a). Within each reducer, tuples with flag L are compared with tuples with flag R, and tuples with flag S are compared only with each other.

Example 3.11 Figure 3.5 gives an example for three tuples t_1, t_2, t_3 given the arrangement of the reducers in Figure 3.4(b). Suppose that tuple t_1 has anchor point $a = 2$, and tuples t_2, t_3 have the same anchor point $a = 4$. The mapping function takes t_1 and gener-

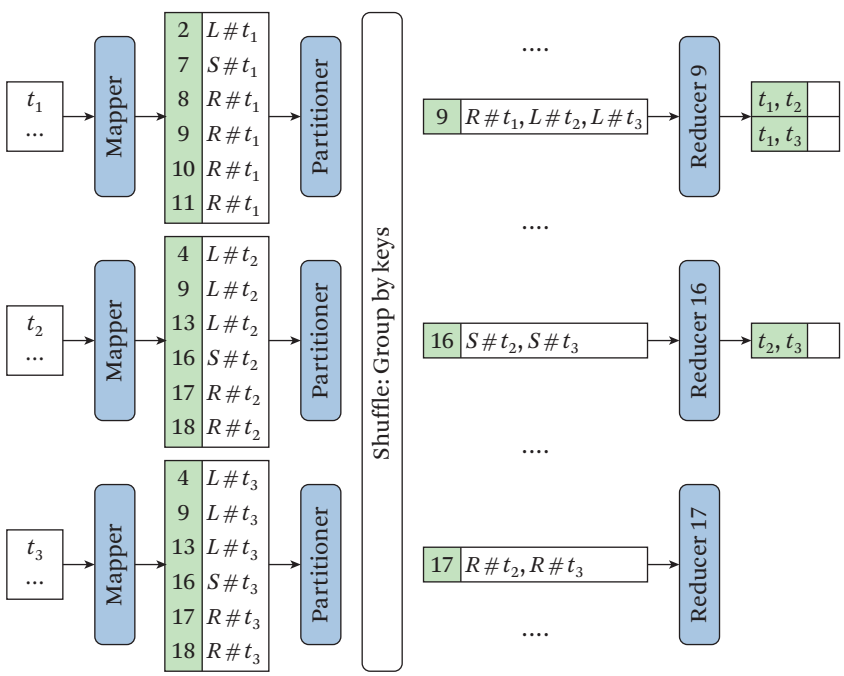


Figure 3.5 Single block distribution example using three tuples, given reducers in Figure 3.4(b).

RIGHTSLINK

ates the key-value pairs $(2, L\#t_1)$, $(7, S\#t_1)$,$(8, R\#t_1)$,$(9, R\#t_1)$,$(10, R\#t_1)$,$(11, R\#t_1)$. Note the different tags L, S, R for different key-value pairs. Reducer 9 receives a list of values $R\#t_1, L\#t_2, L\#t_3$ associated with key 9, and compares tuples marked with R with tuples marked with L, but not tuples marked with the same tag. Reducer 16 receives a list of values $S\#t_2$, $S\#t_3$, and performs comparisons among all tuples marked with S.

The triangle distribution strategy achieves with high probability[1] maximum input $X \leq (1 + o(1))\sqrt{2}X_{low}$ and maximum number of comparisons $Y \leq (1 + o(1))Y_{low}$.

3.5.3 Handling a Single Blocking Function

This section describes the distribution strategy to handle a set of disjoint blocks $\{B_1, \ldots, B_m\}$ produced by a single blocking function h. Let m denote the number of blocks, and for each block B_i, where $i \in [1, m]$, let $W_i = \binom{|B_i|}{2}$ denote the number of comparisons needed. Thus, the total number of comparisons across all blocks is $W = \sum_{i=1}^{m} W_i$. For any distribution strategy that performs data deduplication for n tuples and W total comparisons resulting from a set of disjoint blocks using k reducers, DisDedup derives that $X \geq X_{low} \geq \max(\frac{n}{k}, \frac{\sqrt{2W}}{\sqrt{k}})$ and $Y \geq Y_{low} = \frac{W}{k}$ [Chu et al. 2016]. DisDedup adopts a distribution strategy which guarantees that both X and Y are always within a constant factor from X_{low} and Y_{low}, by assigning reducers to blocks in proportion to the workload of every block.

Intuitively, since the goal is to balance computation, a block of a larger size needs more reducers than a block of a smaller size. Since the blocks are independent, DisDedup allocates the reducers to blocks in proportion to their workload, namely, block B_i will be assigned to $k_i = \frac{W_i}{W}k$ reducers. However, k_i might not be an integer, and it is meaningless to allocate a fraction of reducers. Thus, k_i needs to be rounded to an integer. If $k_i > 1$, DisDedup assigns $\lfloor k_i \rfloor \geq 1$ reducers to B_i. If $k_i \leq 1$, which means $\lfloor k_i \rfloor = 0$, DisDedup must still assign at least one reducer to B_i. The total number of reducers after rounding might be greater than k, in which case reducers have to be responsible for more than one block. Therefore, there needs to be an effective way of assigning reducers to blocks such that both X and Y are minimized.

If $k_i \leq 1$, DisDedup calls B_i a *single-reducer block*; otherwise, B_i is a *multi-reducer block*. Let $\mathcal{B}_s$ and $\mathcal{B}_l$ be the set of single-reducer blocks and multi-reducer blocks, respectively. Next, we show how DisDedup handles single-reducer blocks and

1. The term "with high probability" means that the probability of success is of the form $1 - 1/f(n)$, where $f(n)$ is some polynomial function of the size of the dataset n.

RIGHTSLINK

multi-reducer blocks separately, such that X and Y are bounded by a constant factor:

$$\mathcal{B}_s = \{B_i \mid W_i \leq \frac{W}{k}\}, \qquad \mathcal{B}_l = \{B_i \mid W_i > \frac{W}{k}\}.$$

Every block $B_i \in \mathcal{B}_l$ has $k_i \geq 1$ reducers assigned to it, and DisDedup will use k_i reducers to distribute B_i via the triangle distribution strategy in Section 3.5.2. If k_i is fractional, such as $k_i = 3.1$, DisDedup will simply use $\lfloor k_i \rfloor$ reducers to handle B_i. Since $\sum_{i=c+1}^{m} k_i \leq k$, every reducer will exclusively handle at most one multi-reducer block.

For every block $B_i \in \mathcal{B}_s$, since DisDedup assigns $k_i \leq 1$ reducers to it, DisDedup can use one reducer to handle every single-reducer block. However, DisDedup must assign single-reducer blocks to reducers to ensure that every reducer has about the same amount of workload.

Given the strategies to handle the single-reducer blocks and the multi-reducer blocks, respectively, DisDedup achieves with high probability $X \leq c_x X_{low}$, and $Y \leq c_y Y_{low}$, where $c_x = 5 + o(1)$ and $c_y = 5 + o(1)$ [Chu et al. 2016].

3.5.4 Handling Multiple Blocking Functions

Since a single blocking function might result in false negatives by failing to assign duplicate tuples to the same block, multiple blocking functions are often used to decrease the likelihood of a false negative. In this section, we show how DisDedup distributes the blocks produced by s different blocking functions $h_1, h_2, \ldots, h_s$.

A straightforward strategy to handle s blocking functions would be to apply the same strategy for handling a single blocking function s times (possibly simultaneously). However, this straightforward strategy has two problems: (1) it fails to leverage the independence of the blocking functions, and the tuples from blocks generated by different blocking functions might be sent to same reducer, where they will not be compared; and (2) a tuple pair might be compared multiple times if that tuple pair belongs to multiple blocks, each from a different blocking function. DisDedup addresses these two problems by two principles: (1) allocating reducers to blocking functions proportional to their workload; and (2) imposing an ordering of the blocking functions.

Reducer Allocation. DisDedup allocates one or more reducers to a blocking function in proportion to the workload of that blocking function. Let m_j denote the number of blocks generated by a blocking function h_j, B_i^j denote the ith block generated by h_j, and $W_i^j = \binom{|B_i^j|}{2}$ denote the workload of B_i^j. Let $W^j = \sum_{i=1}^{m_j} W_i^j$ be the

RIGHTSLINK

total workload generated by h_j, and $W = \sum_{j=1}^{t} W^j$ be the total workload generated by all t blocking functions. Therefore, the number of reducers B_i^j gets assigned is $\frac{W_i^j}{W^j} \cdot \frac{W^j}{W} k$, where $\frac{W^j}{W} k$ is the number of reducers for handling the blocking function h_j. Thus, the number of reducers assigned to B_i^j is $\frac{W_i^j}{W}$, regardless of which blocking function it originates from.

This means that DisDedup views the blocks from multiple blocking functions as a set of (possibly overlapping) blocks produced by one blocking function, and apply DisDedup as is. For s blocking functions, it achieves $X < (5s + o(1))X_{low}$, and $Y = (5 + o(1))Y_{low}$.

The above analysis tells us that the number of comparisons will be optimal, but the input may have to be replicated as many as s times. The reason for this increase is that the blocks may overlap, in which case tuples that belong in multiple blocks may be replicated.

Blocking Function Ordering. Since a tuple pair can have the same blocking key values according to multiple blocking functions, a tuple pair can occur in multiple blocks. To avoid producing the same tuple pair more than once, DisDedup imposes an ordering of the blocking functions, from h_1 to h_s. Every reducer has knowledge of all s blocking functions and their fixed ordering. Inside every reducer, before a tuple pair t_1, t_2 is compared according to the jth blocking function h_j, it applies all the lower-numbered blocking function h_z, $\forall z \in [1, j-1]$, to see if $h_z(t_1) = h_z(t_2)$. If such h_z exists, then the tuple pair comparison according to h_j is skipped in that reducer, since there must exist a reducer that can see the same tuple pair according to h_z. In this way, every tuple pair is only compared according to the *lowest numbered blocking function* that puts them in the same block. The blocking function ordering technique assumes that applying blocking functions is much cheaper than applying the comparison function. If this is not true (e.g., the pairs comparison is cheap) the ordering benefit will not be obvious.

Example 3.12 Suppose two tuples t_1 and t_2 are in the same block according to the blocking functions h_2, h_3, h_5, namely, $h_2(t_1) = h_2(t_2)$, $h_3(t_1) = h_3(t_2)$ and $h_5(t_1) = h_5(t_2)$. In this case, DisDedup only compares t_1 and t_2 in the block generated by h_2, and omit the comparison of those two tuples in the blocks generated by h_3 and h_5.

3.6 Record Fusion and Entity Consolidation

Record fusion refers to the process of consolidating multiple records representing the same real world entity into a single representation. Records referring to the

RIGHTSLINK

same entity could come from a single data source or multiple data sources, and there are usually conflicts in different representations. There are two types of data conflicts: *uncertainty* and *contradiction* [Dong and Naumann 2009]. Uncertainty is a conflict between a non-null value and one or more null values that are used to describe the same property of a real world entity. Uncertainty is caused by missing information, usually represented by null values in a source. Contradiction is a conflict between two or more different non-null values that represent different values of the same property of a real world entity. Contradiction is caused by different sources providing different values for the same attribute.

The key issue in record fusion is thus to find the best value among conflicting values caused by the aforementioned two types of conflicts. In Section 3.6.1, we present a classification of different conflict resolution strategies. In Section 3.6.2, we discuss a unique probabilistic fusion approach, which models possible resolutions in a probabilistic manner. In Section 3.6.3, we show more advanced fusion approaches, which take into account the accuracies of sources, freshness of sources, and dependencies between sources when making resolution decisions.

3.6.1 A Classification of Conflict Resolution Strategies

Figure 3.6 shows the classification of different record fusing strategies [Bleiholder and Naumann 2008], and Table 3.2 gives some example strategies according to the classification. *Conflict ignorance* strategies ignore the conflicts between mul-

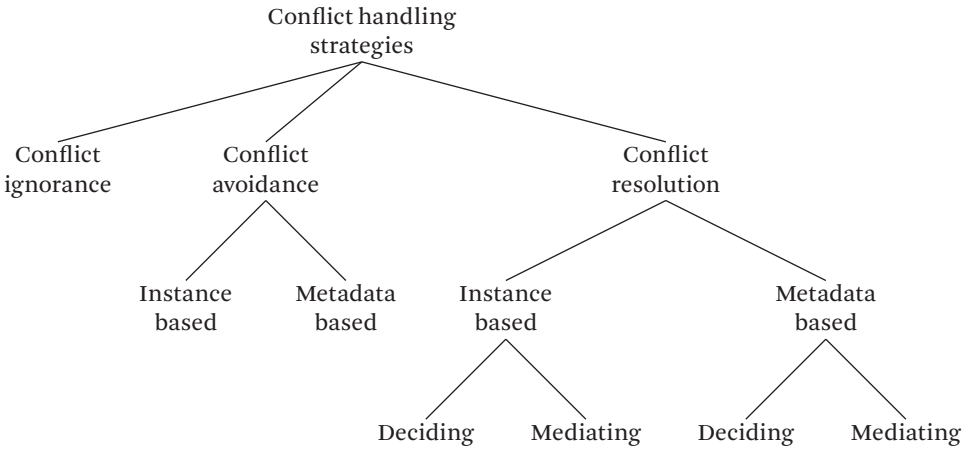


Figure 3.6 Classification of strategies to fuse records [Bleiholder and Naumann 2008].

RIGHTSLINK

Table 3.2 Example of conflict resolution strategies [Bleiholder and Naumann 2008, Dong and Naumann 2009]

Strategy	Classification	Short Description
pass it on	ignoring	escalates conflicts to user or application
consider all possibilities	ignoring	creates all possible value combinations
take the information	avoiding, instance based	prefers values over nulls
no gossiping	avoiding, instance based	returns only "consistent" tuples
trust your friends	avoiding, metadata based	takes the value of a preferred source
cry with the wolves	resolution, instance based, deciding	takes the most often occurring value
roll the dice	resolution, instance based, deciding	takes a random value
meet in the middle	resolution, instance based, mediating	takes an average value
keep up to date	resolution, metadata based, deciding	takes the most recent value

tiple records that refer to the same entity, and pass the conflicts to the users or applications. There are two major options in this category: one is to escalate the conflicts to user or application, while the other is to consider all possible resolution strategies. We discuss probabilistic deduplication that assigns a probability to each possible resolution in Section 3.6.2. *Conflict avoidance* strategies acknowledge the existence of conflicting records, and apply a simple rule to take a unique decision based on either the data instance or the metadata. An example of instance-based conflict avoidance strategy is to prefer non-null values over null values. An example of metadata-based conflict avoidance strategy is to prefer values from one relation over values from another. *Conflict resolution* strategies resolve the conflicts, by picking a value from the already present values (deciding) or by choosing a value that does not necessarily exist among present values (mediating). An example of instance-based, deciding conflict resolution strategy is to take the most frequent value. An example of instance-based, mediating conflict resolution strategy is to take the average of all present values. For instance, to resolve the SAL attribute of

RIGHTSLINK

duplicate tuples t_4 and t_9 in Table 5.1, the average value of t_4[SAL] and t_9[SAL] can be taken.

These basic conflict resolution strategies mostly rely on participating values to resolve conflicts. More advanced strategies leverage external information and dependencies, such as accuracies of sources, freshness of sources, and dependencies between sources, which we discuss in Section 3.6.3.

3.6.2 Probabilistic Resolution

Rather than coming up with a "golden" record for multiple records referring to the same entity, Beskales et al. [2009] study the problem of modeling and querying possible repairs in the context of duplicate detection. Figure 3.7 shows an input relation representing sample census data that possibly contains duplicate records. Duplicate detection algorithms generate a clustering of records (represented as sets of record IDs in Figure 3.7), where each cluster is a set of duplicates that are eventually merged into one representative record per cluster. A one-shot duplicate detection approach identifies records as either duplicates or non-duplicates based on the given cleaning specifications (e.g., a single threshold on record similarity). Hence, the result is a single clustering (repair) of the input relation (e.g., any of the three possible repairs shown in Figure 3.7). However, in the probabilistic duplicate detection approach, this restriction is relaxed to allow for uncertainty in deciding on the true duplicates (e.g., based on multiple similarity thresholds). The result is a set of multiple possible clusterings (repairs), as shown in Figure 3.7.

Beskales et al. [2009] constrain the space of all possible repairs to repairs generated by parameterized hierarchical clustering algorithms for two reasons: (1) the size of the space of possible repairs is linear in the number of records in the unclean relation; and (2) a probability distribution on the space of possible repairs

Person

ID	Name	ZIP	Income
P1	Green	51519	30k
P2	Green	51518	32k
P3	Peter	30528	40k
P4	Peter	30528	40k
P5	Gree	51519	55k
P6	Chuck	51519	30k

Uncertain clustering

Possible repairs

X_1	X_2	X_3
{P1}	{P1,P2}	{P1,P2, P5}
{P2}	{P3,P4}	{P3,P4}
{P3,P4}	{P5}	{P6}
{P5}	{P6}	
{P6}		

Figure 3.7 Probabilistic duplicate detection [Beskales et al. 2009].

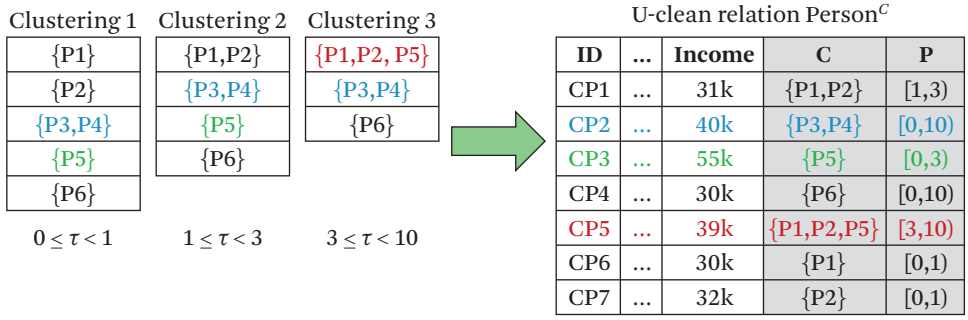
RIGHTSLINK

can be induced based on the probability distribution on the values of the parameters of the algorithm. Specifically, let τ represent possible parameter values of a duplicate detection algorithm $\mathcal{A}$ (e.g., τ could be the threshold value of deciding whether two clusters should be merged in a hierarchical clustering algorithm), let $[\tau^l, \tau^u]$ represent the possible values of τ, and let f_τ represent the probability density function of τ defined over $[\tau^l, \tau^u]$. The set of possible repairs $\mathcal{X}$ is defined as $\{\mathcal{A}(R, t) : t \in [\tau^l, \tau^u]\}$. The set $\mathcal{X}$ defines a probability space created by drawing random parameter values from $[\tau^l, \tau^u]$, based on the density function f_τ, and using the algorithm $\mathcal{A}$ to generate the possible repairs corresponding to these values. The probability of a specific repair $X \in \mathcal{X}$, denoted $\Pr(X)$, is equal to the probability of the parameter range that generates such repair.

Uncertain clean relation (*U-clean relation* for short) is used to encode the possible repairs $\mathcal{X}$ of an unclean relation R generated by a parameterized clustering algorithm $\mathcal{A}$. A U-clean relation, denoted R^c, is a set of c-records where each c-record is a representative record of a cluster of records in R. Attributes of R^c are all attributes of Relation R, in addition to two special attributes: C and P. Attribute C of a c-record is the set of record identifiers in R that are clustered together to form this c-record. Attribute P of a c-record represents the parameter settings of the clustering algorithm $\mathcal{A}$ that lead to generating the cluster represented by this c-record. Figure 3.8 illustrates the model of possible repairs for the unclean relation `Person`. U-clean relation $\texttt{Person}^c$ is created by clustering algorithms $\mathcal{A}$ using parameters τ that are defined on the real interval $[0, 10]$ with uniform distributions. Relation $\texttt{Person}^c$ captures all repairs of the base relations corresponding to possible parameter values. For example, if $\tau \in [1, 3]$, the resulting repair of Relation `Person` is equal to $\{\{\texttt{P1}, \texttt{P2}\}, \{\texttt{P3}, \texttt{P4}\}, \{\texttt{P5}\}, \{\texttt{P6}\}\}$, which is obtained using c-records in $\texttt{Person}^c$ whose parameter settings contain the interval $[1, 3]$. Moreover, the U-clean relation allows identifying the parameter settings of the clustering algorithm that lead to generating a specific cluster of records. For example, the cluster $\{\texttt{P1}, \texttt{P2}, \texttt{P5}\}$ is generated by algorithm $\mathcal{A}$ if the value of parameter τ belongs to the range $[3, 10)$.

Relational queries over U-clean relations are defined using the concept of *possible worlds semantics*, as shown in Figure 3.9. More specifically, queries are semantically answered against individual clean instances of the dirty database that are encoded in input U-clean relations, and the resulting answers are weighted by the probabilities of their originating repairs. For example, consider a selection query that reports persons with `Income` greater than 35k, considering all repairs encoded by Relation $\texttt{Person}^c$ in Figure 3.8. One qualified record is CP3. However, such a record is valid only for repairs generated at the parameter settings $\tau \in [0, 3)$.

RIGHTSLINK



ID	...	Income	C	P
CP1	...	31k	{P1,P2}	[1,3)
CP2	...	40k	{P3,P4}	[0,10)
CP3	...	55k	{P5}	[0,3)
CP4	...	30k	{P6}	[0,10)
CP5	...	39k	{P1,P2,P5}	[3,10)
CP6	...	30k	{P1}	[0,1)
CP7	...	32k	{P2}	[0,1)

Figure 3.8 An example of U-clean relation [Beskales et al. 2009].

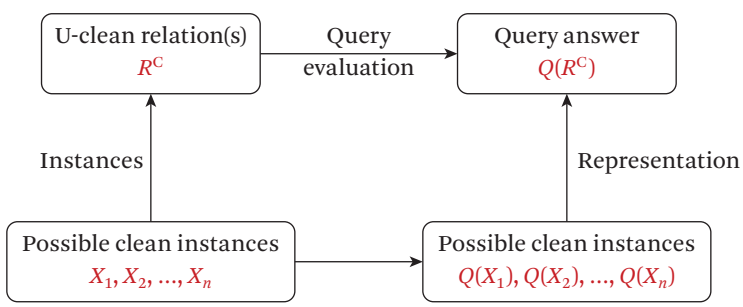


Figure 3.9 U-clean relation query model.

Therefore, the probability that record CP3 belongs to the query result is equivalent to the probability that τ is within [0, 3), which is 0.3.

3.6.3 Advanced Techniques for Conflict Resolution

The basic resolution strategies discussed in Section 3.6.1 mostly use conflicting values for resolution, and they often fall short in some or in all of the following three aspects. First, data sources have different qualities; data values provided by more accurate data sources are usually more accurate. However, more accurate data sources can also provide incorrect values, and therefore an advanced resolution strategy is often needed to take source quality into consideration when predicting the correct value. Second, data sources can copy from each other, and ignoring these kinds of dependencies between data sources can cause wrong resolution decisions. For example, the majority vote strategy to resolve conflicts would be affected if some data items in a source are copied. Third, the correct value for a data

RIGHTSLINK

item may evolve over time as well (e.g., a person's affiliation); hence, it is crucial to distinguish between incorrect value and *outdated* value when evaluating source accuracies and making resolution decisions.

The building block of advanced data fusion strategies is to evaluate the trustworthiness or quality of a source. In this section, we discuss how the accuracy of a data source is modeled by Dong et al. [2009a], and we mention how that model is extended to handle source dependencies [Dong et al. 2009a] and source freshness [Dong et al. 2009b]. For a more comprehensive treatment on the subject of advanced data fusion, we refer readers to the tutorial of Dong and Naumann [2009] and the book of Dong and Srivastava [2015].

Source Accuracy

Dong et al. [2009a] measure the accuracy of a source as the fraction of true values provided by a source. The accuracy of a source S is denoted by $A(S)$, which can be considered as the probability that a value provided by S is the true value. Let $V(S)$ denote the values provided by S. For each $v \in V(S)$, let $Pr(v)$ denote the probability that v is the true value. Then $A(S)$ is computed as follows:

$$A(S) = Avg_{v \in V(S)} Pr(v).$$

Consider a data item D. Let $Dom(D)$ be the domain of D, including one true value and n false values. Let S_D be the set of sources that provide a value for D, and let $S_D(v) \subseteq S_D$ be the set of sources that provide the value v for D. Let $\Phi(D)$ denote the observation of which value each $S \in S_D$ provides for D. The probability $Pr(v)$ can be computed as follows:

$$Pr(v) = Pr(v \text{ is true value} | \Phi(D)) \propto Pr(\Phi(D) | v \text{ is true value}).$$

Assume that sources are independent and that the n false values are equally likely to happen; $Pr(\Phi(D)|v$ is true value) can be computed as follows:

$$Pr(\Phi(D)|v \text{ is true value}) = \prod_{S \in S_D(v)} A(S) \prod_{S \in S_D \setminus S_D(v)} \frac{1 - A(S)}{n}$$

which can be rewritten as

$$Pr(\Phi(D)|v \text{ is true value}) = \prod_{S \in S_D(v)} \frac{nA(S)}{1 - A(S)} \prod_{S \in S_D} \frac{1 - A(S)}{n}.$$

Since $\prod_{S \in S_D} \frac{1-A(S)}{n}$ is the same for all values, we have

$$Pr(\Phi(D)|v \text{ is true value}) \propto \prod_{S \in S_D(v)} \frac{nA(S)}{1 - A(S)}.$$

RIGHTSLINK

Accordingly, the *vote count* of a data source S is defined as:

$$C(S) = \ln \frac{nA(S)}{1 - A(S)}.$$

The *vote count* of a value v is defined as:

$$C(v) = \sum_{S\ inS_D(v)} C(S).$$

Intuitively, a source with a higher vote count is more accurate and a value with a higher vote count is more likely to be true. Combining the above analysis, the probability of each value v can be computed as follows:

$$Pr(v) = \frac{\exp(C(v))}{\sum_{v_0\ inDom(v)} \exp(C(v_0))}.$$

Obviously, for a data item D, the value $v \in Dom(D)$ with the highest probability $Pr(v)$ would be selected as the true value. As we can see, the computation of the source accuracy $A(S)$ depends on the probability $Pr(v)$, and the computation of the probability $Pr(v)$ depends on the source accuracy $A(S)$. Dong et al. [Dong et al. 2009a] proposes an algorithm that starts with the same accuracy for every source and the same probability for every value, and then iteratively computes probabilities for all sources and probabilities for all values until convergence. The convergence criterion is set to be when there is no change in source accuracies and no oscillation in decided true values.

Source Dependency

The above computation for source accuracy assumes that sources are independent. In reality, sources copy from each other, which creates dependencies. Dong et al. [2009a] rely on two intuitions for copy detection between sources. First, for a particular data item, there is only one true value, but there are usually multiple false values. Two sources sharing the same true value does not necessarily imply dependency; however, two sources sharing the same false value is typically a rare event, and thus would more likely imply source dependency. Second, a random subset of values provided by a data source would typically have accuracies similar to the full set of values provided by the data source. However, for a copier data source, the subset of values it copies may have different accuracies than the rest of the values it provides independently. Thus, between two dependent sources where one copies another, the source whose own data values' accuracies differ significantly from the values shared with the other source is more likely to be the copier. Based on these intuitions, Dong et al. [2009a] then use a Bayesian model to compute the

RIGHTSLINK

probability of copying between two sources S_1 and S_2 given the observations Φ on all data items; this probability is then used to adjust the computation of the vote count for a value $C(v)$ to account for source dependencies.

Source Freshness

We have so far assumed that data fusion is done on a static snapshot of the data. However, in reality, data evolves over time and the true value for an item might change as well. For example, the scheduled departure time for a flight might change in different months; a person's affiliation might change over time; and the CEO of a company could also change. To capture such changes, data sources will need to update their data. In this dynamic setting, data errors occur for these several reasons: (1) the sources may provide wrong values, similar to the static setting; (2) the sources may fail to update their data at all; and (3) some sources may not update their data in time. Data fusion, in this context, aims at finding all correct values and their valid periods in the history, when the true values evolve over time. While the source quality can be captured by accuracy in the static case, the metrics for evaluating source quality are more complicated in the dynamic setting—a high quality source should provide a new value for a data item *if and only if* and *right after* the value becomes the true value. Dong et al. [2009b] use three metrics to capture this intuition: the *coverage* of a source measures the transitions of different data items that it captures; the *exactness* measures the percentage of transitions a source mis-captures (by providing a wrong value); and the *freshness* measures how quickly a value change is captured by a source. As before, Dong et al. [2009b] rely on Bayesian analysis to decide both the time and the value of each transition for a data item.

3.7 Human-Involved Data Deduplication

Automatic data deduplication techniques sometimes may not be able to recognize certain duplicate records that may be easily identified by humans. In this section, we study how humans are involved in the data deduplication process to improve the accuracy.

3.7.1 CrowdER

CrowdER [Wang et al. 2012a] uses crowd workers to aid in the data deduplication task. The motivation for CrowdER is that while automatic techniques for data deduplication have been improving, the quality remains far from perfect; meanwhile, crowdsourcing platforms offer a more accurate, if expensive (and slow), way

RIGHTSLINK

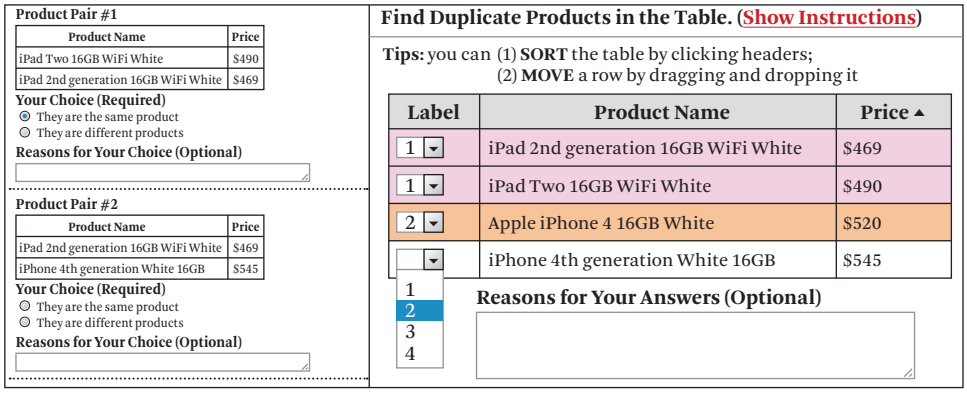


Figure 3.10 HITs for data deduplication.

to bring human insight into the process. Crowdsourcing platforms, such as Amazon Mechanic Turk, support crowdsourced execution of "microtasks" or Human Intelligence Tasks (HITs), where people do simple jobs requiring little or no domain expertise, and are paid per job. Figure 3.10 shows two types for HITs used by CrowdER. The pair-based HIT in Figure 3.10(a) asks a human to check each pair of records individually; the cluster-based HIT in Figure 3.10(b) asks a human to cluster multiple records at the same time.

CrowdER proposes a human-machine workflow as shown in Figure 3.11. The workflow first uses machine-based techniques to compute, for each pair, the likelihood that they refer to the same entity. For example, the likelihood could be the similarity value given by a similarity-based technique. Then, only those pairs whose likelihood exceeds a specified threshold are sent to the crowd. It is shown that by specifying a relatively low threshold, the number of pairs that need to be verified can be dramatically reduced with only a minor loss of quality. Given the set of pairs to be sent to the crowd, the next step is to generate HITs so that people can check them for matches. HIT generation is a key component of the workflow. Finally, generated HITs are sent to the crowd for processing and the answers are collected.

Example 3.13 Figure 3.12 shows a workflow of CrowdER for deduplicating a table consisting of nine records $r_1, \ldots, r_9$ by using pair-based HIT. Instead of asking humans to check all pairs of records, that is $\frac{9*8}{2} = 36$ pairs, CrowdER first employs a machine based approach to calculate the similarities between pairs of records. Those record pairs whose similarities are lower than a threshold are pruned, such as (r_3, r_6). The

RIGHTSLINK

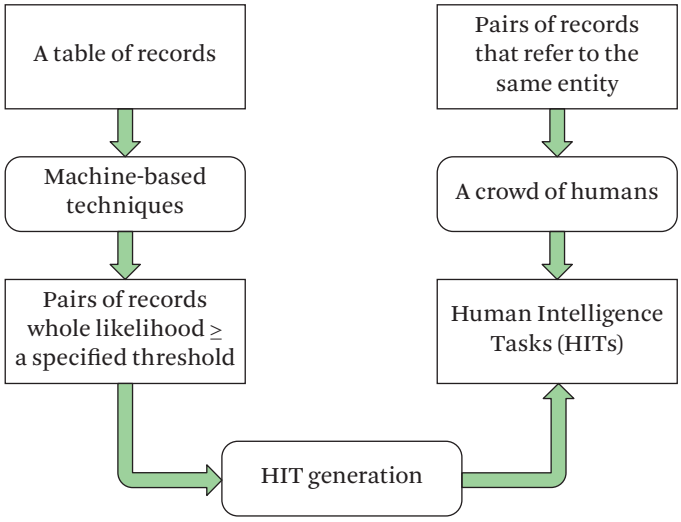


Figure 3.11 Hybrid human-machine workflow [Wang et al. 2012a].

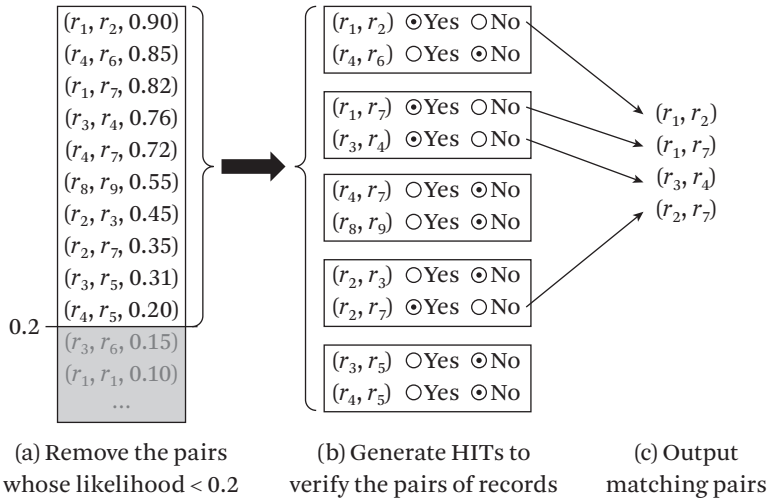


Figure 3.12 CrowdER: an example of using the hybrid human-machine workflow [Wang et al. 2012a].

RIGHTSLINK

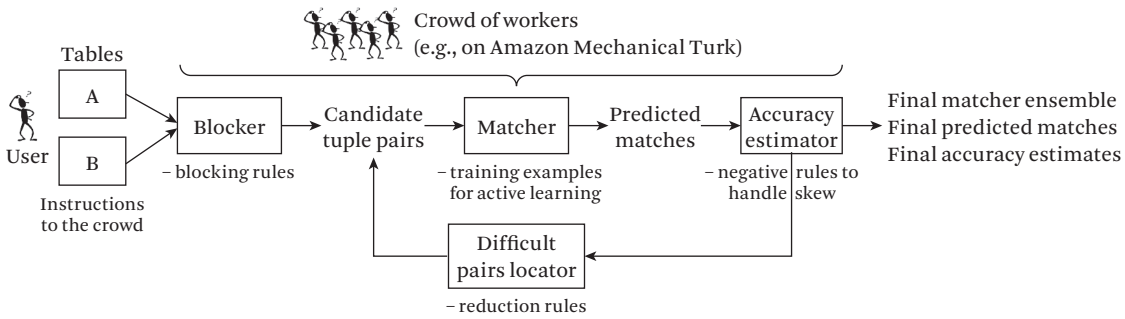


Figure 3.13 The Corleone architecture [Gokhale et al. 2014].

remaining ten pairs can fit into five pair-based HITS, where each HIT contains two questions. The final matching pairs are collected based on user answers.

3.7.2 Corleone

In contrast to CrowdER, which is a hybrid human-machine approach for data deduplication, Corleone [Gokhale et al. 2014] is a data deduplication system that is completely crowdsourced, i.e., no developers need to be involved. Corleone is desirable because enterprises routinely need to solve tens to hundreds of data deduplication tasks. To involve a developer to write the blocking rules and matching function for every deduplication task is time consuming and costly. Figure 3.13 shows the Corleone architecture, which consists of four main components: Blocker, Matcher, Accuracy Estimator, and Difficult Pairs Locator. Blocking often uses heuristic rules, e.g., "if the prices of two products differ by more than $100, then they do not match," to reduce the number of pairs of records to be matched. Corleone takes a small sample from all pairs of records and asks the crowd to label a small set of informative pairs to learn a random forest, from which potential blocking rules are extracted. The crowd is involved again to validate the quality of obtained blocking rules. After blocking, the next step is to build and apply a matcher to match surviving pairs of records. Corleone employs active learning to minimize crowdsourcing costs, taking into account possible noisy crowd input. The next step, i.e., estimating the matching accuracy, is vital for real world data deduplication tasks. To do this, Corleone considers constructing a minimal labeled set, given a maximum allowable error bound. The difficult pairs locator finds pairs that the current matcher has matched incorrectly according to the accuracy estimator. The entire process is iterated until the estimated matching accuracy no longer improves.

RIGHTSLINK

3.8 Data Deduplication Tools

In this section, we discuss some representative data deduplication tools, including two academic open-source data deduplication tools, AJAX [Galhardas et al. 2001] and Febrl [Christen 2008], as well as a commercial tool, Data Tamer [Stonebraker et al. 2013].

3.8.1 AJAX

AJAX [Galhardas et al. 2001] is a data deduplication framework that separates the logic and physical levels of data cleaning. The logic level supports the design of the data flow graph that models the data operations needed to clean the data, while the physical level supports the implementations and optimizations of the data operations. AJAX models the data deduplication workflow using four types of operations: *mapping*, which standardizes data formats (e.g., dates) whenever possible, *matching*, which applies similarity functions to one or more attributes to produce a similarity value for every pair of compared records, *clustering*, which groups records together based on the similarity values, and *merging*, which produces a canonical record for every cluster of records.

AJAX provides a declarative language, which is SQL enriched with a set of specific primitives, to express the four data operations. AJAX raises an exception, implemented via the Java exception mechanism, whenever the process fails, and the users are expected to manually examine and resolve the exceptions. AJAX was the first open-source tool to be explicit about the work flow of data deduplication by mapping it into a sequence of four operations of mapping, matching, clustering, and merging. AJAX was later extended with the notion of quality constraints [Galhardas et al. 2011] imposed on relations within the directed graph of data transformation, and the violations of the quality constraints can be inspected by manual data repairs.

3.8.2 Febrl

A data deduplication project has many components, such as blocking and classification, and significant advances have been made in each of these components. However, many new techniques are difficult for researchers and practitioners to experiment with and compare because they are either implemented as research prototypes or hidden inside expensive commercial software. The Freely Extensible Biomedical Record Linkage system (Febrl) [Christen 2008] aims at bridging this gap by providing a graphical user interface that allows users to easily select which techniques to use for a component. In addition, since Febrl is written in Python and open-sourced, it is also easy to include new techniques in it. Therefore, Febrl can

RIGHTSLINK

be used by researchers and practitioners to compare existing data deduplication techniques, to include their own techniques, and to ultimately compose a suitable data deduplication workflow for their own projects.

The Febrl interface contains many tabs, including Data, Explore, Index, Compare, Classify, and Log. Inside every tab, users are given many options to choose from. The Data tab allows users to choose the input data source, and whether the task is to perform data deduplication on one dataset or to perform record linkage between two datasets; the Explore tab allows users to get a better understanding of the dataset by showing, for every attribute, the number of unique values, the smallest and the largest value, the quantile distribution of values, the number of missing values, and an inferred type of the attribute; the Index tab allows users to select one or more blocking functions, such as the ones discussed in Section 3.4; and the Compare tab allows users to select which similarity functions to use for which attributes. Febrl contains 26 similarity functions as well as functions for specialized data types such as dates and times; the Classify tab allows users to select classifiers to use for deciding whether a tuple pair are duplicates, including both supervised and unsupervised techniques; and the Log tab contains the Febrl Python code generated through the project, allowing users to verify the correctness of the code and to export the code to run outside of Febrl.

3.8.3 Magellan

Magellan [Konda et al. 2016] is an entity matching management system that focuses on enabling users to build entity matching workflows effectively and efficiently. Magellan has some important features. It provides how-to guides to tell users what to do in each scenario step by step. It provides tools to help users to perform these steps. These tools cover all the steps in an entire workflow. The tools are built on top of known Python data processing libraries, which allows Magellan to borrow a rich set of capabilities found in Python libraries, such as data cleaning and visualization. It also provides a powerful scripting environment to facilitate interactive experimentation and allows users to quickly "patch" the workflow if needed.

Figure 3.14 shows the architecture of Magellan. The system includes a set of entity matching scenarios. For each scenario, it provides a how-to guide, which solves the scenario in two stages: development and production. In the development stage, the guide leads the user step by step. For each step such as blocking and matching, the user can choose a set of supporting tools, each of which is a set of Python commands. The development stage is typically done on a data sample for fast experimentation. In the production stage, the guide also tells the user how to

RIGHTSLINK

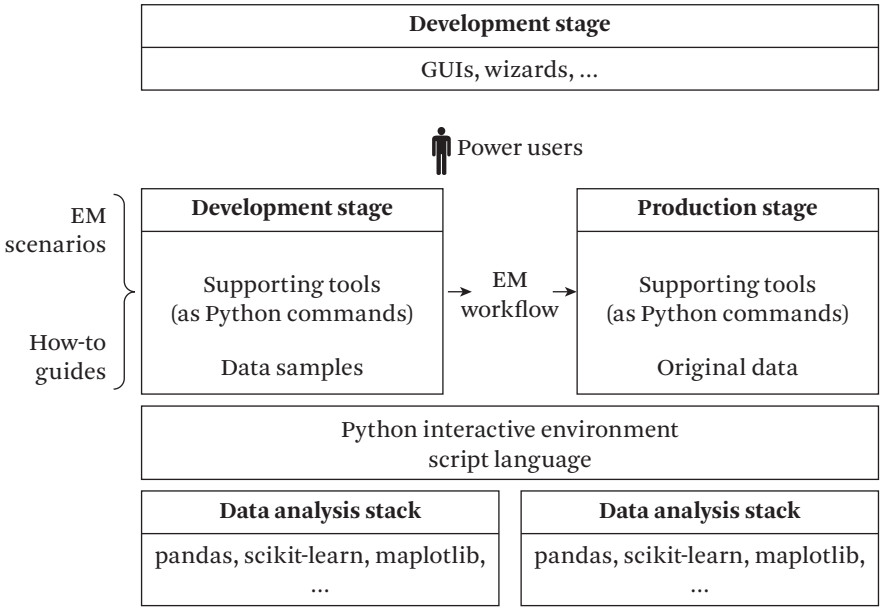


Figure 3.14 The Magellan architecture [Konda et al. 2016].

implement and execute the workflow on the entire dataset, using a set of supporting tools. Both stages rely on the Python and its interactive environment (e.g., iPython). The tools used by Magellan are built on top of known Python data processing libraries, which come with many useful functionalities. Magellan targets power users who can program, but intends to develop a GUI on top to allow lay users to build entity matching workflows easily.

3.8.4 Data Tamer

Data Tamer [Stonebraker et al. 2013] is a data curation system that cleans and transforms large scale data sources at the enterprise level. Data Tamer integrates the schema, and instances of these sources, through a series of mapping, deduplication, and linking exercises. The core technological innovation of Data Tamer is the automation of the data curation process while involving various roles of data experts, including data owners, data stewards, data scientists, and data curators. This closely coupled design of machine and human enables practical, end-to-end curation at the scale of hundreds to thousands of disparate datasets. Human experts and owners are involved in multiple tasks, including: (1) answering pairwise comparison questions for training machine learning models (e.g., classifiers); (2) validating

RIGHTSLINK

the machine decision on matching attributes or records from data sources; and (3) providing explicit business rules for deduplication or cleaning data sources. A task-expertise matching system is used to dispatch human tasks to solve curation tasks at different granularities (e.g., comparing pairs of columns, comparing pairs of values, or validating a cluster of related entities). Data Tamer is an example of involving users in the cleaning process at multiple levels in the curation stack and at multiple granularities.

3.9 Conclusion

Duplicate records appear in many different datasets, such as duplicated customer records, duplicated publication records, and duplicated Wikipedia entries. While the problem definition of data deduplication—finding and merging records that represent the same real world entity—is concise and clear, designing an end-to-end solution presents many challenges. In this chapter, we covered various aspects of designing a data deduplication workflow.

To determine whether a pair of records are duplicates, we introduced several similarity metrics and various classifier-based approaches based on those similarity metrics. We classified string similarity metrics into character-based metrics, token-based metrics, and phonetics-based metrics. We discussed both unsupervised techniques and supervised techniques to predict duplicates based on similarity metrics.

Given the computed similarity graph, where each node represents a record and each edge represents the likelihood of two records being duplicates, a clustering step is used to partition all records into disjoint clusters of records, where each cluster corresponds to one real-world entity. We discussed several commonly used clustering algorithms, including clustering using connected-component, hierarchical agglomerative clustering, and correlation clustering algorithms. We also presented different choices available to find a canonical representation for records in the same cluster.

To improve the running time of data deduplication for large datasets, we discussed two approaches: blocking and distributed computation. Blocking techniques or blocking functions aim at reducing the number of comparisons by avoiding comparing tuple pairs that are unlikely to be matches. We classified blocking techniques into two types: hash-based blocking and similarity-based blocking. Hash-based techniques, such as locality sensitive hashing, use one or multiple hash functions to place records into blocks, where each block is associated with a unique

RIGHTSLINK

hash key. Similarity-based techniques, such as prefix filtering, group nearby records according to a similarity metric. Distributed data deduplication techniques have also been proposed to parallelize the task with the main objective of minimizing computation and communication costs in a shared-nothing parallel computing environment.

We also discussed how humans can be involved in the data deduplication process to determine record pairs that machines cannot predict confidently. The main optimization problem in human-involved data deduplication is to minimize human involvement as much as possible since humans are usually much more expensive than machines.

After years of research and development that have produced open-source tools as well as commercial products, data deduplication remains an active field of research. Designing better similarity metrics, classifiers, and clustering algorithms to improve data deduplication accuracy is needed. Another possible research direction is performing data deduplication in streaming scenarios. As new data comes in and new evidence becomes available, previously declared non-duplicates records may suddenly become duplicates and previously declared duplicates may need to be revoked. Deduplicate detection in other data formats, such as texts, videos, and images, also deserves more attention.

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

4 Data Transformation

Both outlier detection (Chapter 2) and duplicate detection (Chapter 3) expect the input data to be in the "right" format. For example, a list of temperatures in different units (e.g., Celsius or Fahrenheit) needs to be converted into the same unit for outlier detection, and attributes of different records need to be aligned properly for detecting duplicates. *Data transformation* refers to the data preparation activity of running user-defined programs, rules, or scripts to convert data from one format or structure into another format or structure. Besides outlier detection and data deduplication, transformations are used in a variety of tasks, and at different stages of the ETL life cycle. For example, before running a data integration project, transformations are often used to standardize data formats, to enforce standard patterns, or to trim long strings. Transformations are also used at the end of the ETL process, for example, to merge clusters of duplicate records, to find a unique representation for a cluster of records (also known as golden record), or to prepare data to be consumed by analytics tools. Data transformation can also be seen as a tool for data repair (Section 6.2), since it can be used to "transform" erroneous data.

Data transformation tasks can be classified into two types: *syntactic data transformations* [Raman and Hellerstein 2001, Kandel et al. 2011, Gulwani 2011, Jin et al. 2017] and *semantic transformations* [Singh and Gulwani 2012, Abedjan et al. 2016b]. Syntactic transformations aim at transforming a table from one syntactic format to another, often without requiring external knowledge or reference data. Example syntactic transformations include transforming phone numbers to a standard format, concatenating two columns containing first name and last name into a name column, or altering the layout of a table to make it easier to read (cf. Figure 4.1(a)). Semantic transformations usually involve understanding the meaning/semantics, or the typical use of the data; hence, they usually require referencing external data sources. For example, transforming conference name abbreviations to full conference names (e.g., "SIGMOD" to "Special Interest Group on Management of Data") would require an external data source that contains both the abbreviations and the full names of conferences.

RIGHTSLINK

Name	
John	Tel: 7188751243
	Fax:7188751200
Mike	Tel: 7186359762
	Fax:7186359700
Frank	Tel: 5198780763
	Fax:5198780700
Julie	Tel: 5176543809
	Fax:5176543800

Name	Tel	Fax
John	718-875-1243	718-875-1200
Mike	718-635-9762	718-635-9700
Frank	519-878-0763	519-878-0700
Julie	517-654-3809	517-654-3800

(a) Syntactic transformations

Country code
US
CA
CN
DE

Country
United States
Canada
China
Germany

(b) Semantic transformations

Figure 4.1 Example transformations.

Example 4.1 Figure 4.1(a) shows a syntactic transformation, where tabular data containing person names, telephone numbers, and fax numbers, originally arranged in a tabular format with phone and fax numbers stored in the same column as a complex data type, are flattened into a normal relational table representation for efficiently querying of this data. In addition, the telephone numbers and the fax numbers are also transformed by adding two dashes between the nine digits. Both cases do not require external knowledge to perform these transformations.

Figure 4.1(b) shows a semantic transformation, where country codes are transformed into country names, requiring external knowledge, such as an external knowledge base that contains both full and abbreviated country names.

In this chapter, we discuss techniques for performing both syntactic and semantic data transformations in Sections 4.1 and 4.2, respectively. Since data transformation is usually part of commercial Extract-Transform-Load (ETL) tools, we give an overview of some current main ETL tools and their data transformation and cleaning capabilities in Section 4.3. We limit our discussion to data transformations that accept as input a tabular data form and output another tabular data form.

RIGHTSLINK

Other types of data manipulation tasks, such as *information extraction*, which aims at extracting structured data from unstructured or semi-structured data [Sarawagi 2008], are outside the scope of this text.

4.1 Syntactic Data Transformations

There are three major components/dimensions of a syntactic data transformation system: *language*, *authoring*, and *execution*, as shown in Figure 4.2. Since there are many ways to transform a dataset, syntactic data transformation solutions usually adopt a *transformation language* that limits the space of possible transformations. The language could consist of a finite set of operations allowed on a table [Raman and Hellerstein 2001, Kandel et al. 2011], such as splitting a column and merging two columns, or it could consist of a set of functions for string manipulations [Gulwani 2011], such as extracting a substring and concatenating two substrings. A language needs to be expressive enough so that it captures many real-world transformation tasks, and at the same time, restricted enough to allow for effective and easy authoring of the transformations. For a specific transformation task, syntactic transformation tools have different interaction models with the users to *author a*

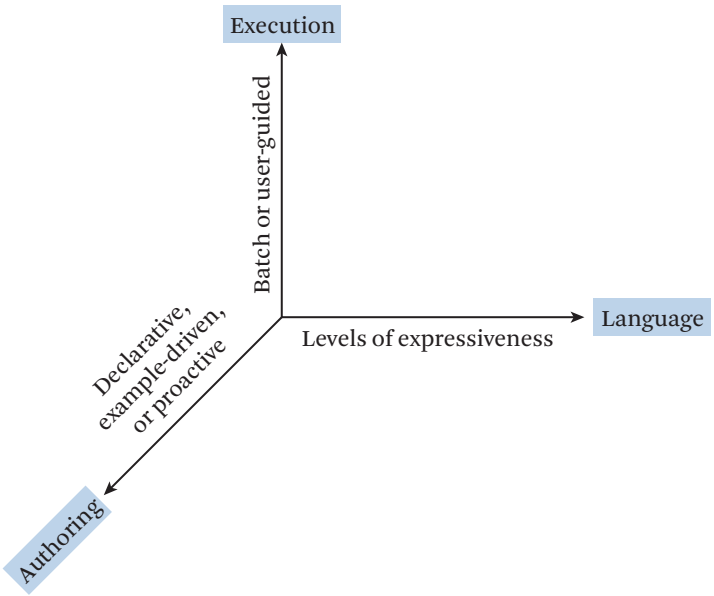


Figure 4.2 The three components/dimensions of a syntactic transformation system.

RIGHTSLINK

transformation program using the adopted language. The interaction models can be generally classified as declarative, example-driven, or proactive. In the declarative model, users specify the transformations directly, usually through a visual interface. In the example-driven model, users are required to give a few input-output examples, based on which the tool automatically infers likely transformations that match the examples. In the proactive interaction model, the transformation tool hints at which data needs transformation, and sometimes even suggests possible transformations. Finally, *transformation execution* applies the specified transformations to the dataset. Since there might be multiple specified transformations from the previous step, transformation tools usually offer assistance in selecting the desired transformation, for example, by displaying the effect of the specified transformation immediately on sample data, or by providing interpretations of the specified transformations.

Different transformation tools offer different capabilities along these three dimensions. In this section, we use example syntactic transformation systems to illustrate these three components in detail: Data Wrangler [Kandel et al. 2011, Guo et al. 2011, Heer et al. 2015] is based on Potter's Wheel [Raman and Hellerstein 2001] and is designed for transforming tabular data; QuickCode [Gulwani 2011] is designed for string manipulations; and KNIME[1] is an open source workflow authoring tool that can compose data transformation programs declaratively.

4.1.1 Transformation Language

A transformation language defines the set of allowed operators a syntactic transformation tool allows. In what follows, we discuss the languages adopted by Data Wrangler and QuickCode.

Data Wrangler [Kandel et al. 2011, Guo et al. 2011, Heer et al. 2015] adopts a language that consists of a sequence of individual operators to support common data manipulation tasks, such as splitting or extracting values from strings, deleting or merging columns, interpolating values, and reshaping tables by folding and unfolding. Table 4.1 shows the most common operators used and their descriptions in Data Wrangler. A critical feature of the Data Wrangler's language design is its compactness: with only a dozen operators, analysts can complete a wide variety of data transformation tasks. The language is based on SchemaSQL [Lakshmanan et al. 2001], and it has been proven that the language can express any one-to-one or one-to-many transformations of cell values [Raman and Hellerstein 2001]. The declarative nature of the data transformation language facilitates the implemen-

1. https://www.knime.com/

RIGHTSLINK

Table 4.1 The Data Wrangler transformation language [Kandel et al. 2011]

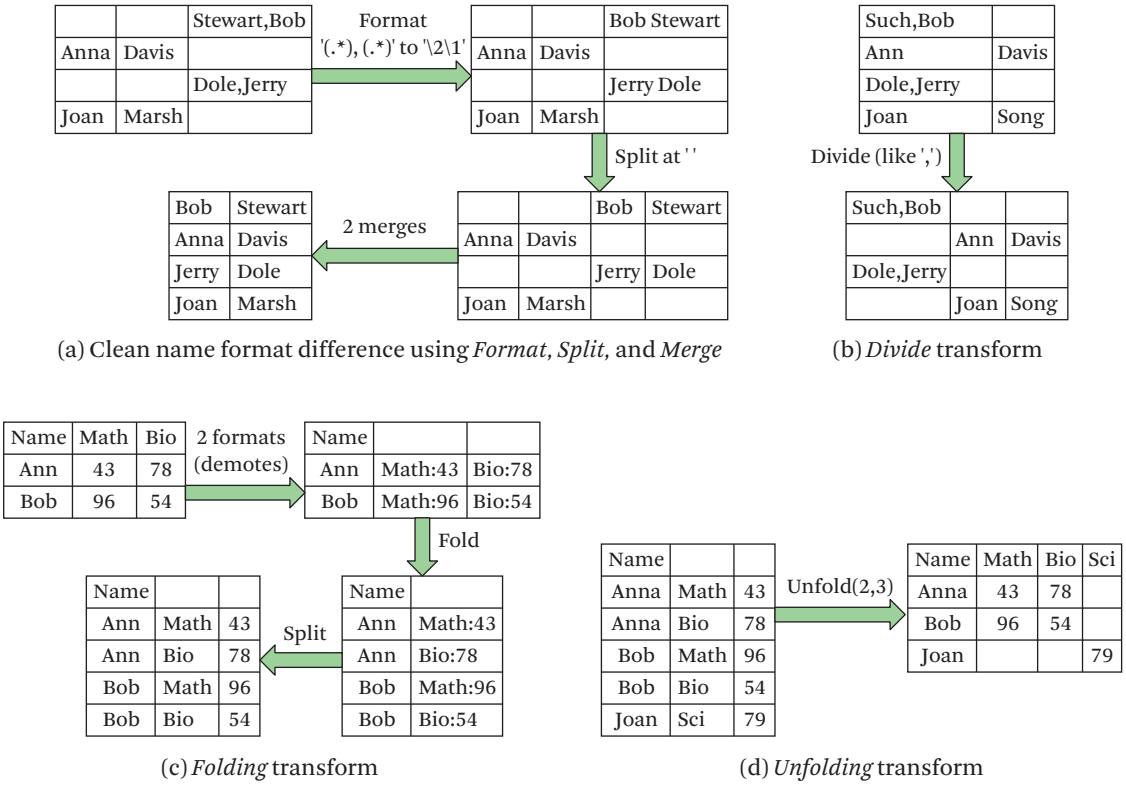
Operations	Description
Cut	Remove selected text from cells in specified columns
Delete	Remove rows that match given indexes or predicates
Divide	Conditionally split a column into two, sending values into one of the two new columns based on a predicate
Drop	Remove specified columns from table
Edit	Edit the text in each cell of the specified columns
Extract	Copy text from cells in a column into a new column
Fill	Fill empty cells using values from adjacent cells
Format	Apply a function to every value in a column
Fold	Reshape a table into columns of key-value sets: selected rows map to keys, selected columns to values
Merge	Concatenate multiple columns into a single column
Promote	Promote row values to be the column names
Split	Split a column into multiple columns by delimiters
Translate	Shift the position of cell values by a given offset
Transpose	Transpose the rows and columns of the table
Unfold	Reshape a table by mapping key-value sets to a collection of new columns, one per unique key

tation across multiple platforms, for example, by generating executable code for multiple runtimes, including Python and JavaScript. Oftentimes, users "wrangle" a subset of the data using the interface and then export a resulting Python script to transform the much larger dataset on a server.

Example 4.2 When integrating data from different sources, values of the same type might exist in different formats. Figure 4.3(a) shows an example of using the *Format, Split,* and *Merge* operators to clean name format discrepancies. Figure 4.3(b) shows an example of using the *Divide* operator to separate various name formats; the tabular data after applying the Divide operator can then be transformed similar to Figure 4.3(a).

Many-to-many row mappings, such as *Fold* and *Unfold*, are useful for resolving higher-order schematic heterogeneity where information is stored partially in the schema and partially as data values. The Fold operator flattens the table by converting one row into multiple rows, folding a set of columns into one column, and

RIGHTSLINK



(a) Clean name format difference using *Format, Split,* and *Merge*

(b) *Divide* transform

(c) *Folding* transform

(d) *Unfolding* transform

Figure 4.3 Example transformations in Data Wrangler [Raman and Hellerstein 2001].

replicating the rest of the columns. Figure 4.3(c) shows an example of the Folding operator, where the subject names are demoted into rows using the *Format* operator, then the last two columns are folded together while the first column is being replicated. Lastly, the *Split* operator is used to split the subject from the grade. The Unfold operator, on the other hand, unflattens the table by taking two columns, collecting rows that have the same values for all the other columns, and unfolding the two chosen columns. Values in one of the chosen columns are used as column names to align the values in the other chosen column. Figure 4.3(d) shows an example of unfolding the second and third column.

While Data Wrangler aims at providing a set of transformation operators to manipulate tabular data, QuickCode [Gulwani 2011] defines a transformation language that is focused on transforming strings. The domain-specific string processing language QuickCode adopts supports limited conditionals and loops, syntactic

RIGHTSLINK

Table 4.2 **The QuickCode syntactic transformation language**

String program $P :=$	$Switch((b_1, e_1), \ldots, (b_n, e_n)) \mid e$
Boolean condition $b :=$	$d_1 \vee \ldots \vee d_n$
Conjunction $d :=$	$\pi_1 \wedge \ldots \wedge \pi_n$
Predicate $\pi :=$	$Match(v_i, r, k) \mid \neg Match(v_i, r, k)$
Trace expression $e :=$	$Concatenate(f_1, \ldots, f_n) \mid f$
Atomic expression $f :=$	$ConstStr(s) \mid SubStr(v_i, p_1, p_2) \mid Loop(\lambda w : e)$
Position $p :=$	$CPos(k) \mid Pos(r_1, r_2, c)$
Regular expression $r :=$	$TokenSeq(T_1, \ldots, T_n) \mid T \mid \epsilon$

string operations, such as substring and concatenation, and matching based on regular expressions. Table 4.2 shows the string transformation language adopted by QuickCode. A string program P in QuickCode is an expression that maps the input string values $v_1, \ldots, v_m$ to an output string value s. To better explain QuickCode's language, we start by introducing the basic constructs, including regular expression r, position p, trace expression e, and Boolean expression b. Then we introduce the program P. Readers who are familiar with manipulating strings with regular expressions may jump directly to Example 4.4.

QuickCode limits regular expression r to an empty string ϵ, a token T, or a sequence of tokens $TokenSeq(T_1, \ldots, T_n)$. QuickCode also adopts the following finite set of tokens: (i) *StartTok*, which matches the begining of a string; (ii) *EndTok*, which matches the end of a string; (iii) a token for each special character, such as dot, semicolon, hyphen; and (iv) two tokens for each character class C, one that matches a sequence of one or more characters in C, and one that matches a sequence of one or more characters that are not in C. Character classes of C include numerical digits, alphabetic characters, uppercase alphabetic characters, lowercase alphabetic characters, alphanumeric characters, and whitespace characters. For example, *UpperTok* represents a sequence of one or more uppercase characters.

A positional expression p is either a constant integer $CPos(k)$ that denotes the kth position in an input string v_i, or another expression $Pos(r_1, r_2, c)$, where r_1 and r_2 are two regular expressions and c is an integer. We use $v_i[t_1, t_2)$ to denote the substring of v_i between positions t_1 and t_2. $Pos(r_1, r_2, c)$ refers to a position t in v_i, such that r_1 matches some suffix of $v_i[0 : t)$ and r_2 matches some prefix of $v_i[t : length(v_i))$. $Pos(r_1, r_2, c)$ returns the cth such position.

A trace expression e is the concatenation of atomic expressions $f_1, \ldots, f_n$, where each atomic expression f is either a constant-string *ConstStr*, a substring expression

RIGHTSLINK

SubStr, or a loop expression constructed from *Loop*. The substring expression $SubStr(v_i, p_1, p_2)$ relies on two positional expressions p_1 and p_2, which evaluate to a position within v_i.

A Boolean expression b is a propositional formula in disjunctive normal form. A propositional formula has a unique truth value if all values of all variables in the formula are given. A predicate $Match(v_i, r, k)$ in the formula is evaluated to be true if v_i contains at least k non-overlapping matches of regular expression r.

The top-level string expression P is a *Switch* constructor whose arguments are pairs of Boolean expression b_i that return *True* or *False* based on the input values, and a trace expression e_i, which returns an output string s. The set of Boolean expressions are disjoint; for any set of input values, there is at most one Boolean expression evaluated to be true. The return value of P is the value of the trace expression e_i that corresponds to the Boolean expression b_i satisfied by the input set of values. If none of the Boolean expressions are unsatisfied, then the trace expression e is returned.

Example 4.3 Consider an example string v_i to be "Very Large Data Bases", which has 21 characters indexed from 0–20.

The positional expression $Pos(\epsilon, UpperTok, 1)$ evaluates to position 0, since ϵ matches the suffix $v_i[0, 0)$, which is an empty string, and *UpperTok* matches the prefix of $v_i[0, 21)$, which is "Very Large Data Bases." The position 0 is the first position that satisfies such a condition.

The positional expression $Pos(\epsilon, UpperTok, 2)$ evaluates to position 5, since ϵ matches the suffix $v_i[0, 5)$, which is "Very," and *UpperTok* matches the prefix of $v_i[5, 21)$, which is "Large Data Bases." The position 5 is the second position that satisfies such a condition.

Similarly, the positional expression $Pos(UpperTok, \epsilon, 1)$ evaluates to position 1, and positional expression $Pos(UpperTok, \epsilon, 2)$ evaluates to position 6.

The expression $SubStr(v_i, Pos(\epsilon, UpperTok, 1), Pos(UpperTok, \epsilon, 1))$ is thus $SubStr(v_i, 0, 1)$, which returns the first uppercase token "V".

The expression $SubStr(v_i, Pos(\epsilon, UpperTok, 2), Pos(UpperTok, \epsilon, 2))$ is thus $SubStr(v_i, 5, 6)$, which returns the second uppercase token "L".

There are two additional expressions that QuickCode allows for convenience. A commonly used expression is $SubStr(v_i, Pos(\epsilon, r, c), Pos(r, \epsilon, c))$, which evaluates to the cth occurrence of regular expression r in v_i; this is abbreviated to $SubStr2(v_i, r, c)$. The loop expression $Loop(\lambda w : e)$ is the concatenation of $e_1, e_2, \ldots, e_n$, where e_i is obtained from e by replacing all occurrences of integer w in e by i, and n is the smallest integer such that e_{n+1} is undefined.

RIGHTSLINK

Example 4.4 The following transformation task aims at extracting conference name abbreviations from conference names by concatenating the CAPITAL letter of every word, and can be expressed in QuickCode's domain specific language as: $Loop(\lambda w : SubStr2(v_1, UpperTok, w))$.

Input v_1	Output s
Very Large Data Bases	VLDB
International Conference on Data Engineering	ICDE
International Conference on Machine Learning	ICML

4.1.2 Transformation Authoring

Syntactic transformation tools must allow for easy and effective authoring of transformation programs. We classify the interaction models for transformation authoring into *declarative transformations*, *transformation by example*, and *proactive transformations*. Most modern tools offer one or more of these authoring capabilities to solicit desired transformations.

Declarative authoring allows users to specify transformation directly, usually through a visual interface that contains a table view as well as a set of menu-accessible transformation operators. Users specify desired transformations by selecting appropriate transformations using drop-down menus, or by highlighting certain data for the tool to infer potential transformations. Both Potter's Wheel and Data Wrangler provide a menu-based interface, and KNIME provides a drag-and-drop graphical interface. Declarative authoring allows skilled practitioners to transform data rapidly; however, they usually have a learning curve as users need to be familiar with the transformation task as well as the set of available operators.

Example-driven authoring requires uses to give a few input-output examples, based on which the tools automatically infer plausible transformations. Transformation by example can significantly reduce the users' burden in authoring transformations; however, they might limit the set of allowed transformations, either because learning transformations in a large space is expensive or because some transformations, such as table reshaping in Figure 4.1(a), lack intuitive interfaces to provide examples.

Proactive authoring automatically suggests potential transformations, without requiring (or with minimal) user input. These tools often assume a "goodness" criterion about the data, for example, depending on how the data is consumed by downstream analytics. Candidate transformations are then suggested to bring the data to a desired state with respect to the "goodness" criterion.

RIGHTSLINK

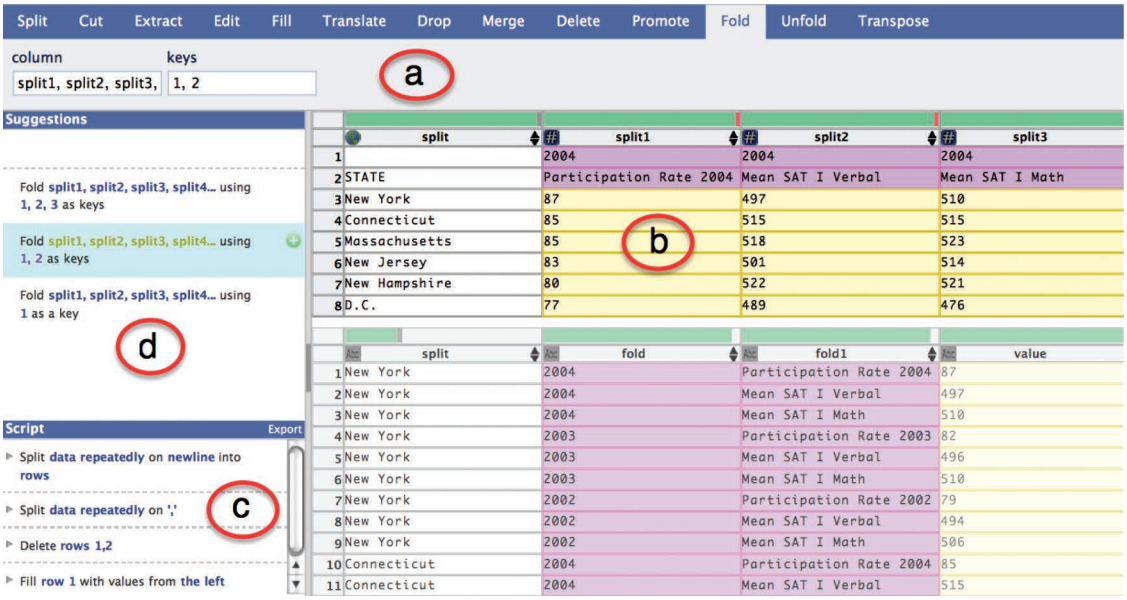


Figure 4.4 The Data Wrangler interface [Kandel et al. 2011, Guo et al. 2011, Heer et al. 2015]. Panel descriptions: (a) tool bar for operator specification; (b) data table display; (c) history of data transformations, which can be exported into a script; and (d) suggested transformation operators. The effect of the selected Fold operator is previewed in the data table display (b) before and after the transformation.

Use Case 1: Declarative Authoring in Data Wrangler

The Data Wrangler user interface, shown in Figure 4.4, allows users to specify transformations in the underlying language. The interface has four main components: the top of the screen shows a tool bar for the set of operators; the right panel shows an interactive data table, displaying the current state of the table as well as a preview of the table after applying the selected operator; and the left panel contains both automated operator suggestions and the history of transformations. Users can specify transforms in multiple ways. A transformation can be specified manually by selecting a operator type from the tool bar and then entering the desired parameter values.

Data Wrangler can also reactively infer transformations though user interaction with the interface, including selecting rows and columns by clicking their headers, or selecting and manipulating texts within cells. While these actions can be used to specify parameters for a transformation type selected from the tool bar, manually choosing the transform type can be tedious and requires the user to be familiar with

RIGHTSLINK

the available transforms. Data Wrangler automatically infers applicable transforms directly from the user selection on the table display in three steps.

1. Data Wrangler infers a set of possible parameter values. Example parameters include regular expressions matching selected text substrings, and row predicates matching selected rows, such as "row is empty" or "row [equals | contains | starts with | ends with] selected values".
2. Data Wrangler enumerates a list of transforms that accepts the inferred parameters. For example, a split transform can accept a parameter set containing a text selection, but an unfold transform cannot accept such a parameter set.
3. The list of transforms is filtered and ranked according to historical usage statistics and heuristics to improve result quality. For example, one of the ranking criteria is diversity, namely, the top suggested transforms should contain multiple types of transforms rather than one type of transform.

Use Case 2: Declarative Authoring in KNIME

KNIME is a general purpose open source data analytics tool allowing users to visually create a workflow by assembling data processing nodes, selectively executing some or all analysis steps, and later inspecting the results. Figure 4.5 shows the graphical interface of KNIME. The space of all possible data processing nodes, shown as the node repository in Figure 4.5, are grouped into categories. For example, the *IO* category includes data processing nodes for reading data from various sources, such as CSV files and relational database engines; the *Manipulation* category includes data processing nodes for manipulating data, such as splitting a column based on regular expressions and filling in missing values based on provided rules; the *View* category includes nodes for different ways of visualizing the data, such as Pie charts and histograms; and the *Analytics* category includes nodes for common data mining and machine learning algorithms, such as decision trees and neural networks, and common statistical analytics, such as hypothesis testing and regression analysis. Every node has a configuration window for the parameter settings. For example, a CSV file reader node has parameters to set for which file to read and which character should be used as a separator. Users can compose a data analytics workflow by selecting desired nodes from the node repository and connecting them into a workflow.

Users can easily use KNIME to perform data transformation tasks. The node repository is essentially the transformation language. Users can compose a transformation program by building a workflow using the data processing nodes.

RIGHTSLINK

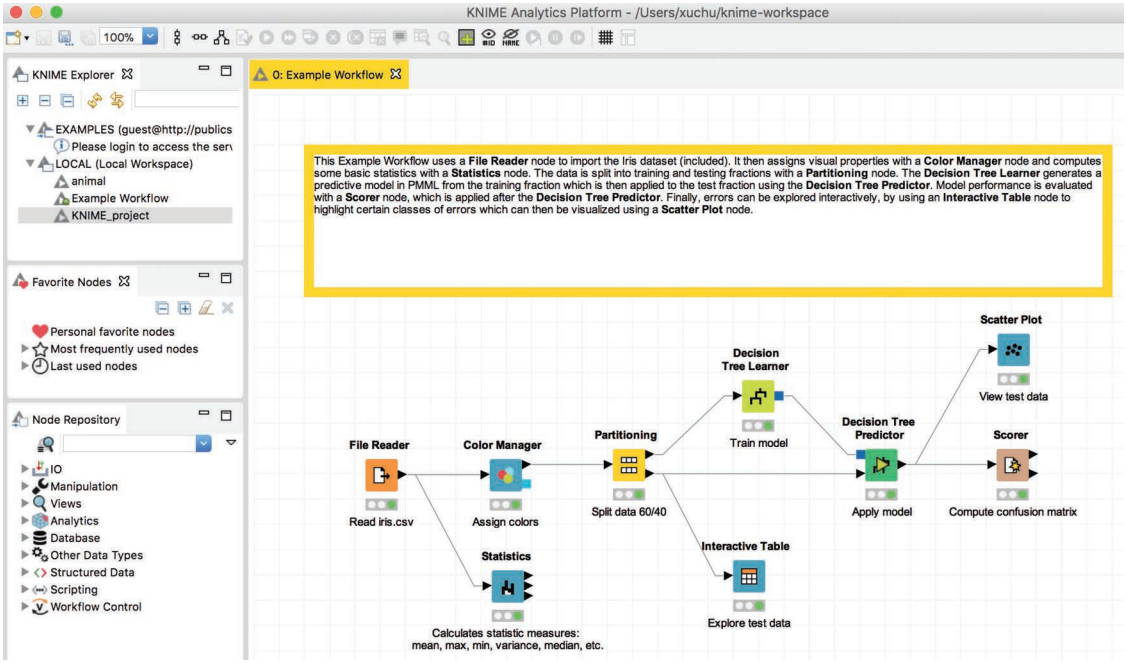


Figure 4.5 KNIME graphical interface. The right panel shows the current workflow composed. The top-left panel shows different workspaces for different workflows. The bottom-left panel shows the node repository, grouped into categories. The middle-left panel shows favorite and recently used nodes.

Use Case 3: Example-Driven Authoring in QuickCode

Programming by example (also known as programming by demonstration and inductive synthesis) refers to a methodology that infers a generalized program based on user-provided examples that describe the expected behavior of the program [Cypher and Halbert 1993]. It has been used in a variety of settings, such as building regular expressions from example texts [Blackwell 2001], creating mashups from multiple web sources by providing examples on how to extract and transform web sources [Tuchinda 2008], and integrating data automatically based on a program learned from example copy-and-paste actions performed by users [Ives et al. 2009]. Unsurprisingly, programming by example is also useful in expressing transformation tasks [Gulwani et al. 2012].

QuickCode [Gulwani 2011] is an Excel plug-in that automatically synthesizes a program for manipulating string transformation tasks based on a few user-provided input-output examples. Table 4.3 shows a syntactic transformation task

RIGHTSLINK

Table 4.3 **Programming by example with five user-provided input-output examples**

Input	Output
323-708-7700	323-708-7700
(425)-706-7709	425-706-7709
510.220.5586	510-220-5586
235 7654	425-235-7654
745-8139	425-745-8139
(519)-781-8816	
. . .	
781-0987	

with five input-output examples, where the user would like to transform phone numbers to a consistent format, adding a default area code "425" if the area code is missing. For each input-output example, QuickCode first computes the set of all expressions that map the input to the output. It then intersects these sets for similar examples and learns conditionals to handle different cases. It finally ranks the set of synthesized programs mainly according to one principle: *a simpler program is more likely to be correct than a more complicated one*. Figure 4.6 shows a synthesized program that QuickCode synthesizes using its domain-specific language for the transformation task in Table 4.3. It checks if the first three letters of an input v_1 are numbers. If they are numbers, it applies expression e_1, which concatenates the first sequence of number tokens in the input, a constant token "-", the second sequence of number tokens, a constant "-", and the third sequence of number tokens; otherwise, it concatenates a constant prefix "425-", the first sequence of number tokens, a constant token "-", and the second sequence of number tokens.

Use Case 4: Proactive Authoring in Data Wrangler

Proactive transformations suggest possible transformations automatically without (or with minimal) user input. They are intended to facilitate the specification of complex wrangling tasks, such as table reshaping operations, that are difficult for users to specify. The intuition is that users find it easier to deal with "recognition" tasks, such as confirming suggested transformations, than to come up with complex transformations. The suggested transformations aim at converting the input data to a more "suitable" state for downstream analytics.

RIGHTSLINK

```
Switch ((b1, e1), (b2, e2)), where
b1 = Match(ν1,NumTok, 3)
b1 = ¬Match(ν1,NumTok, 3)
e1 = Concatenate(SubStr2(ν1,NumTok, 1),ConstStr("-"),
                 SubStr2(ν1,NumTok, 2),ConstStr("-"),
                 SubStr2(ν1,NumTok, 3))
e2 = Concatenate(ConstStr("425-"),SubStr2(ν1,NumTok, 1),
                 ConstStr("-"),SubStr2(ν1,NumTok, 2))
```

Figure 4.6 A synthesized program [Gulwani 2011].

Data Wrangler defines the *suitability* score of table to indicate the degree to which a table adheres to a relational format usably by downstream analytics tools, such as Tableau.[2] These tools often expect the data to be in a relational format, where every column has atomic/simple data type and each row describes a single entity. The suitability score considered in Data Wrangler rewards column type homogeneity (H), fewer empty values (E), and smaller number of delimiters (D), including comma, colon, pipe, or tab characters. Poor column type homogeneity of a column can cause that column to be interpreted only as strings when imported by downstream tools. A table with many empty values may indicate there are replicated values in adjacent columns, and a column with many delimiters may also be treated as strings, rather than composite values of more specific types. For a table T with rows R and columns C, the suitability score is defined as follows:

$$S(T) = \left(1 - \frac{\sum_{c \in C} H_c(T)}{|C|}\right) + \frac{E(T) + D(T)}{|R||C|}.$$

The number of rows and number of columns are denoted as $|R|$ and $|C|$, respectively. H_c is the homogeneity of Column c, and is defined as the sum of squares of the proportions of each type present in that column. Data Wrangler parses each cell into the most specific type, including default built-in types, such as *integer, double, or date*, and user-defined domains, such as zip or country code. If a cell does not match any specific type, then the most generic string type is assigned to that cell. For example, if 75% of the values in a column are integers and 25% are dates, the column homogeneity score is $0.75^2 + 0.25^2 = 0.625$. A column of only one type has the maximum homogeneity score of 1.0. $E(T)$ is the number of empty cells in the table, and $D(T)$ is the number of cells that contain delimiters in the table. The suitability score is invariant of the size of the table, since it is normalized.

2. https://www.tableau.com

RIGHTSLINK

Data Wrangler makes proactive suggestions that improve a table's suitability score by first generating suggestions and then ranking these candidates before presenting them to the user. Data Wrangler considers the following operators as candidate proactive suggestions: (1) *Delete* all empty columns; (2) *Delete* all empty rows; (3) *Delete* all mostly empty rows (more than 75% empty cells); (4) *Fill* all empty cells of column C with values from the above; (4) *Fill* all empty cells of row R with values from the left; (5) *Split* column C into multiple columns using a delimiter; (6) *Fold* columns $C_1, \ldots C_n$ using rows $R_1, \ldots, R_n$ as keys; and (7) *Unfold* column C_1 on column C_2. Since most of these operators take row and/or column indices as parameters, enumerating all possibilities for every type of operator is expensive. For example, there are $O(|R|)$ possible row fills, $O(|C|)$ possible column fills, $O(2^{|R|} \times 2^{|C|})$ possible folds, and $O(|C|^2)$ possible unfolds. Data Wrangler adopts a set of heuristics inspired by real-world use cases from a corpus of public datasets to reduce the set of proactive transformations considered.

1. Data Wrangler only suggests *Fill* candidates if the majority of the cells in a given row or column are empty. This type of transform is useful for filling in sparse rows or columns before performing a fold or unfold operation.
2. Data Wrangler only generates *Fold* candidates using all columns except for the leftmost one, and using at most the first three rows as keys, limiting the number of possible folds to three. In addition, fold candidates are only generated if the leftmost column contains all unique and non-null values in rows other than the key rows; otherwise, there is conflict in the result table.
3. Data Wrangler also only generates *Unfold* candidates of the form "Unfold C_1 on C_2" if all columns except C_2 are completely homogeneous and non-empty; otherwise, the header of the resulting table will not be well typed. Also, unfold candidates are generated if the table has exactly 3, 4, or 5 columns, which corresponds to 1, 2, or 3 key columns, respectively.

Example 4.5 To see how proactive authoring in Data Wrangler alleviates the burden of users, Figure 4.7 illustrates an end-to-end scenario, where a user transforms a table, imported from an Excel file (Table 1 in Figure 4.7), into a dense relational table suitable for analytics (Table 6 in Figure 4.7). Table 1 in Figure 4.7 is unsuitable for analytics since it contains unusable header rows, empty rows, and it also mixes telephone and FAX number in a complex data type. In contrast, Table 6 is much more suitable for downstream analytics, as it lacks these anomalies.

First, Data Wrangler analyzes the input table and generates three suggestions, as shown under Table 1 in Figure 4.7. The user ignores the suggestions and instead

RIGHTSLINK

1.

Bureau of I.A.	
Regional Director	Numbers
Niles C.	Tel: (800)645-8397
	Fax: (907)586-7252
Jean H.	Tel: (918)781-4600
	Fax: (918)781-4604
Frank K.	Tel: (615)564-6500
	Fax: (615)564-6701

Manually select and delete the first two rows

Proactive suggestions (none chosen):
1. Fill column 1 from above
2. Split column on ':'
3. Delete empty rows

2.

Niles C.	Tel: (800)645-8397
	Fax: (907)586-7252
Jean H.	Tel: (918)781-4600
	Fax: (918)781-4604
Frank K.	Tel: (615)564-6500
	Fax: (615)564-6701

Proactive suggestions:
1. Fill column 1 from above
2. Split column on ':'
3. Delete empty rows

3.

Niles C.	Tel	(800)645-8397
	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
	Fax	(615)564-6701

Proactive suggestions:
1. Delete empty rows
2. Fill column 1 from above

4.

Niles C.	Tel	(800)645-8397
	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
	Fax	(615)564-6701

Proactive suggestions:
1. Fill column 1 from above

5.

Niles C.	Tel	(800)645-8397
Niles C.	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
Jean H.	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
Frank K.	Fax	(615)564-6701

Proactive suggestions:
1. Unfold column 1 on column 3
2. Unfold column 2 on column 3

6.

	Tel	Fax
Niles C.	(800)645-8397	(907)586-7252
Jean H.	(918)781-4600	(918)781-4604
Frank K.	(615)564-6500	(615)564-6701

Result: cleaned table ready for DB import

Figure 4.7 Proactive data wrangling example. The proactive suggestion chosen at each step is marked in bold [Guo et al. 2011].

chooses to manually delete the first two rows of the table, turning Table 1 into Table 2. The user then chooses the second proactive suggestion for Table 2 (shown in bold), which transforms Table 2 into Table 3. Data Wrangler makes two proactive suggestions based on Table 3, as shown under Table 3, and asks the user to pick one. The process goes on until the data is transformed into Table 6, which is now suitable for downstream analytics.

4.1.3 Transformation Execution

There might be multiple-authored transformation programs, declared by users or inferred by the tool as we discussed in Section 4.1.2. Transformation tools usually

RIGHTSLINK

offer assistance to the users in selecting the correct transformation, for example, by displaying the effect of the specified transformation immediately on sample data, or by providing natural language descriptions of the expected outcomes of the specified transformations, to guide the execution of these transformations.

For example, Data Wrangler executes the set of proactively suggested transformations on the current table and calculates the suitability score for each resulting table. All the suggested transformations are sorted by the improvements on the suitability score of the tables before and after applying the suggestions. As shown in Figure 4.4, Data Wrangler displays the current state of the table as well as a preview of the table after applying the selected transformations. The user can preview and execute one of these suggestions, or choose to ignore them and make a selection on the data table to surface reactive transform suggestions.

Another example is QuickCode, which may produce multiple programs that match the examples. QuickCode offers two ways to interact with users to determine which synthesized program best fits the transformation task [Gulwani et al. 2012]: the first interaction model requires users to examine the results of applying a transformation program on the data. If any output is deemed incorrect, the user can fix it and reapply the synthesizer using the fix as an additional example; and the second interaction model requires users to examine the actual programs generated by the synthesizer and to pick a program. The programs can be shown using programming language syntax, or they can be described in natural languages. The differences between different programs can be highlighted by showing users different output values for an input on which the programs behave differently.

4.2 Semantic Data Transformations

Semantic transformations usually involve understanding the meaning/semantics or the typical use of the data, instead of simply as a sequence of characters. In contrast to syntactic transformations, semantic transformations cannot be computed by solely looking at input values; rather, they usually require external data sources to look up the values that need to be transformed. Example-driven techniques have been mostly used for semantic transformations [Arasu et al. 2009, Singh and Gulwani 2012, Abedjan et al. 2016b, Jin et al. 2017]. We discuss semantic transformations by example in Section 4.2.1 using DataXFormer [Abedjan et al. 2016b], which uses external web tables and web forms to match user-provided examples.

RIGHTSLINK

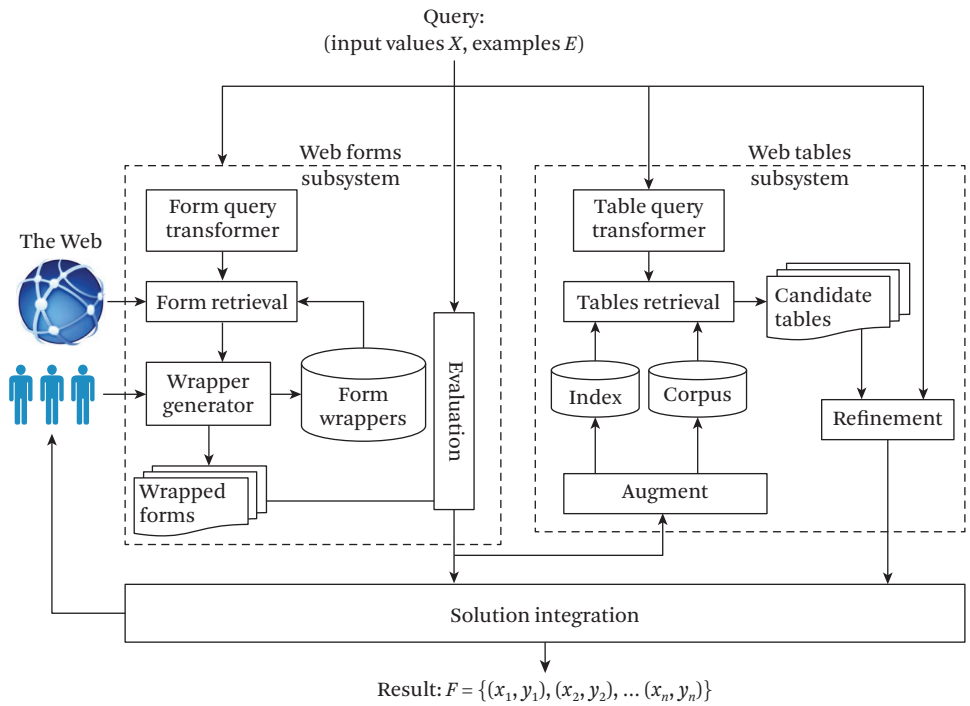


Figure 4.8 DataXFormer architecture [Abedjan et al. 2016b].

Data exchange [Fagin et al. 2005a], a process that transforms structured data under a source schema to structured data under a target schema based on semantic mappings between the two schemas, can also be considered as a semantic transformation. We briefly discuss data exchange and refer interested readers to relevant surveys in Section 4.2.2.

4.2.1 Semantic Transformations by Example

DataXFormer [Abedjan et al. 2016b] is an example-driven technique for semantic data transformation based on two external data sources: locally stored static web tables and dynamically discovered web forms. The architecture of DataXFormer is illustrated in Figure 4.8. The user starts by submitting a transformation query that includes a few input-output examples, such as the query that transforms airport codes to city names. DataXFormer converts the user query into an internal form to retrieve candidate web tables and candidate web forms, respectively. While the table retrieval component works on top of a local repository, the form retrieval

RIGHTSLINK

component uses the entire web to find relevant web forms. DataXFormer assumes that the user provides as input a set X of n values and a set E of m example pairs, $E = \{(x_i, y_i) \mid x_i \in X, 1 \leq i \leq m\}$. The goal is to discover the missing Y values and output a solution F, $F = \{(x_i, y_i) \mid x_i \in X, 1 \leq i \leq n\}$.

Using an inverted index, the table retrieval algorithm retrieves a set of relevant candidate tables for the given user query. Candidate results are further analyzed by the *refinement* component, which verifies the coverage of the candidate tables with respect to the query examples. If the coverage is above the user-defined threshold, DataXFormer extracts the rest of the required transformation values. In the case of web forms, DataXFormer uses a web search engine to retrieve relevant URLs. By examining the results of the web search, DataXFormer identifies candidate web forms. Then, for each candidate form, DataXFormer generates a "wrapper" that will allow it to query the form and to obtain the transformation values. Candidate web forms are then queried using the examples present in the user query. Those with sufficient coverage are then invoked with the remaining input values. In a final step, the *solution integration* component consolidates and ranks multiple transformation solutions for a given query, as obtained from the two subsystems, and outputs the desired transformation.

Web Tables for Transformation

A table T is *relevant* to a transformation query if it contains at least τ of the examples provided in that query, where τ is a predefined threshold. Based on the number of contained examples, DataXFormer assigns scores to the tables. These scores weigh in when reconciling conflicting transformations of different tables to ensure high quality results.

Scanning every table in the corpus for the given examples can be prohibitively expensive. To mitigate such a cost, DataXFormer uses a filter-refine approach illustrated by multiple steps in Figure 4.9. In Step 1, a query is submitted to the web table repository, which maintains millions of n-ary relations. Step 2 corresponds to the *filter* phase. Here, DataXFormer locates a candidate set of relevant tables.

In Figure 4.9, Tables 1, 2, and 4 cover $\tau = 2$ examples and are therefore candidate tables. The *filter* phase significantly affects the recall of the results. For example, a coarse-grained index will return many false positives, while a very restrictive query can result in low recall because it misses relevant tables.

In the *refine* phase (Step 3 in Figure 4.9), DataXFormer validates the row correspondences of found examples and compute scores of found transformations and tables that contain them. DataXFormer follows an iterative and inductive approach. An iteration refers to one pass of filter and refine. Since the initial examples might

RIGHTSLINK

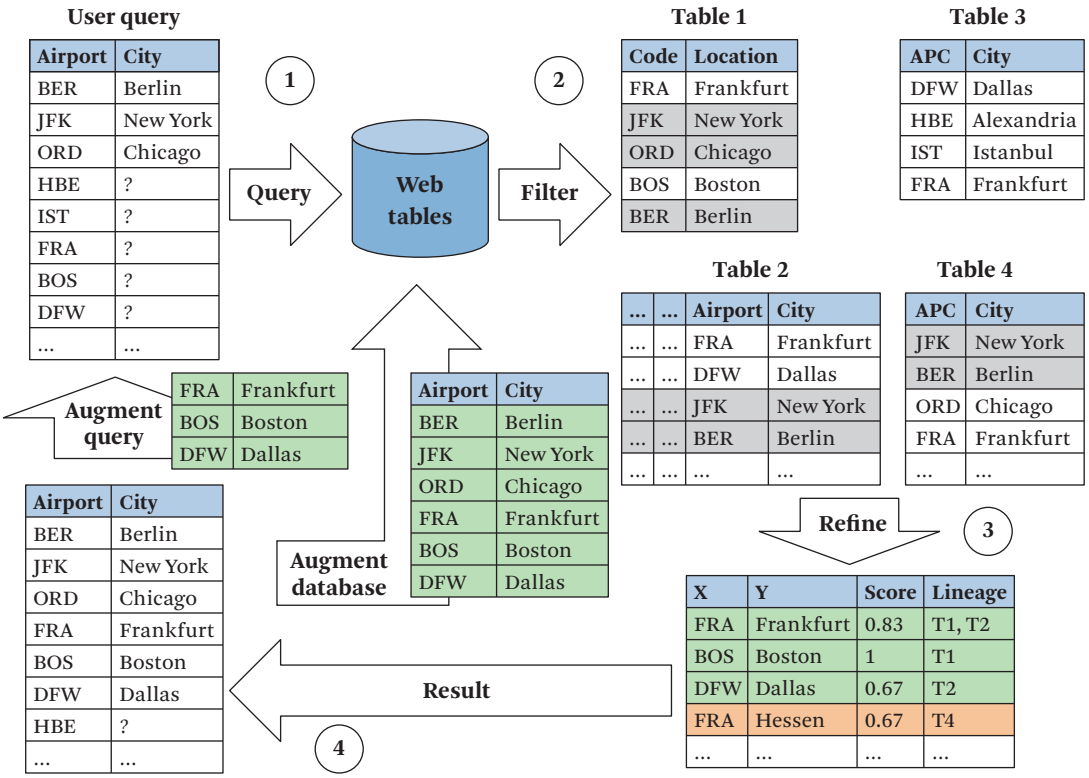


Figure 4.9 Workflow of DataXFormer's filter-refine approach [Abedjan et al. 2016b].

not be representative or might have a low recall, newly discovered mappings are used as examples in subsequent iterations. When the iterations converge, the final transformation results are presented to the user (Step 4 in Figure 4.9). Furthermore, if users are satisfied with the result or any subset of the results, they can trigger the system to store the results as a new table with a higher initial confidence score in the database.

The simplest way to identify the tables that support a transformation is to use an inverted index that maps cell values in web tables to the indexed tables and columns. DataXFormer stores the tables within a universal main table (relation *Cells* in Figure 4.10), where every cell of a web table is represented by a tuple that records the table, column, and row IDs, along with the tokenized, stemmed form of its value. The relation is ordered by *tableid*, *columnid*, and *rowid*, simultaneously achieving two advantages: (i) every column from a web table is stored contiguously, and (ii) the space requirement of this schema can be alleviated by compression, which

RIGHTSLINK

Tables

tableid	url	Title	Initial weight
1	www...	World airports	0.8
2	http...	–	0.5
3	http...	Airports	0.5
...	...	...	0.5

Columns

tableid	colid	Header
1	1	Code
1	2	Location
...	...	...
4	1	apc
4	2	Location
...	...	...

Cells

tableid	colid	rowid	Term	Term tokenized
1	1	1	FRA	fra
1	1	2	JFK	jfk
...	...	...	...	...
3	2	1	Dallas	dallas
...	...	...	...	...
4	2	4	Hessen	hessen

Projection on cells
Sort order from left to right

Term tokenized	tableid	colid	rowid	Term
ber	1	1	5	BER
ber	2	1	4	BER
ber	4	1	2	BER
berlin	1	2	5	Berlin
...	...	...	...	...
ord	1	1	3	ORD

Figure 4.10 Schema for storing web tables in a column-store [Abedjan et al. 2016b].

is provided by most modern column-stores. DataXFormer stores web tables in a multi-node Vertica instance. Vertica employs projections on tables in place of an inverted index. A projection is a specialized materialized view that is efficient to maintain and load.

Besides table content, DataXFormer also stores other metadata such as the number of rows in a table to easily prune tables with fewer rows than the required number of covered examples. Additional dimension tables are maintained (Figure 4.10). Relation *Tables* stores metadata of tables, such as the URL where the table came from, the table title, and an initial table weight that may vary depending on the authoritativeness of the table. The initial weight influences the confidence score that is computed later to rate transformation results. Relation *Columns* stores column metadata, such as column headers.

Given the index built for the web tables, DataXFormer looks for column pairs that contain at least τ of the given examples. To maximize coverage, DataXFormer tokenizes and stems every value x and y from the input. Using the *Cells* relation described earlier, DataXFormer filters all relevant column pairs with a single SQL query:

RIGHTSLINK

```
SELECT col1.tableid, col1.colid, col2.colid
FROM
(SELECT tableid, colid
 FROM Cells
 WHERE term_tokenized IN (<x1>,<x2>,..., <xm>)
 GROUP BY tableid, colid
 HAVING COUNT(DISTINCT term_tokenized) >= tau)
 AS col1,

(SELECT tableid, colid
 FROM Cells
 WHERE term_tokenized IN (<y1>,<y2>,..., <ym>)
 GROUP BY tableid, colid
 HAVING COUNT(DISTINCT term_tokenized) >= tau)
 AS col2

WHERE col1.tableid = col2.tableid
AND col1.colid <> col2.colid
```

The query joins two subqueries, one to find the columns that contain the X values and another to identify the columns that contain the Y values, where a column is uniquely identified using the table and the column IDs.

In the *refine* phase, DataXFormer loads the content of each candidate column pair and checks the row correspondence between the values. Note that the candidate generation does not ensure the transformation examples are aligned in the corresponding rows. If τ examples still match after considering the row correspondence, DataXFormer collects all transformation results that are provided by the corresponding table.

The retrieved tables might provide conflicting answers, i.e., returning different Y values for the same X value. For example, in Figure 4.9, the airport code "FRA" has been assigned to two different values. A naïve approach to resolve such conflicts is to apply a majority vote. However, the authoritativeness of tables should also be considered. For example, while *(JFK, New York)* might appear in more tables of the database, a table from a more reliable source might provide *(JFK, New York City)*, which is more accurate. Therefore, it is necessary to score tables according to their authoritativeness as well as their coverage of examples with a confidence score. DataXFormer adopts an iterative expectation-maximization (EM) approach that incorporates confidence scores. The confidence score of each table (i.e., the probability that an answer it provides is correct) is estimated based on the current belief about the probability of the answers. Initially each table is assigned with a confidence score based on the number of user examples it covers. The score of the

RIGHTSLINK

table is weighted with its initial weight assigned by experts and stored in relation *Tables*. The answer scores are updated based on the newly computed scores, and the process is repeated until convergence is reached (i.e., the sum of all score changes is below a very small value ϵ). In the end, for each $x \in X$, the scores of possible answers form a probability distribution, and DataXFormer reports the highest scoring value as the answer.

Web Forms for Transformation

As with web tables, a web form is relevant to a query if it covers at least τ of the example transformations. As there is no common repository of web forms, DataXFormer needs to dynamically search for relevant forms from the web. In addition, a new web form appears as a black box, and an invocation strategy (i.e., wrapper) has to be developed to use the form to produce the desired transformations.

DataXFormer dynamically searches for relevant forms from the web by issuing search queries (on existing engines) with the attribute names I_X and I_Y, and it also maintains a repository of web forms that have been successfully wrapped and previously used to answer transformation queries. Each web form is stored as a document that contains the attribute names I_X and I_Y, frequent terms from the form's webpage, and examples from previous transformation tasks. In addition to querying the web, DataXFormer also queries this repository for candidate matches along with their corresponding wrappers.

Since the number of web forms is far less than the number of regular pages retrieved by a search, DataXFormer issues multiple keyword queries and filters the results coming from the underlying search engine to find web forms. Example keyword queries include terms such as "convert," "detect," or "lookup," in addition to using column names as keyword queries.

To be able to wrap web forms, DataXFormer simulates a web browser and probes the forms by using the given example values $(x, y) \in E$ to identify the relevant input field to fill in the x values, the output field that contains the desired transformation result y, and the request method for invoking the form.

Figure 4.11 shows a candidate web form that has been retrieved for transforming stock names to companies, e.g., `AMZN` to `AMAZON.COM INC`. The screenshot on the left shows the form's input fields that comprise various radio buttons, a selection field, an input field, and a button. By analyzing the HTML code, DataXFormer discovers that the form can be invoked by a GET request URI with the relevant parameter `?symbol` that has to be set to our input value `AMZN`. Submitting the GET request returns the page that is illustrated on the right of Figure 4.11. The desired output `AMAZON.COM INC` appears on the page. DataXFormer discovers the XPath of

RIGHTSLINK

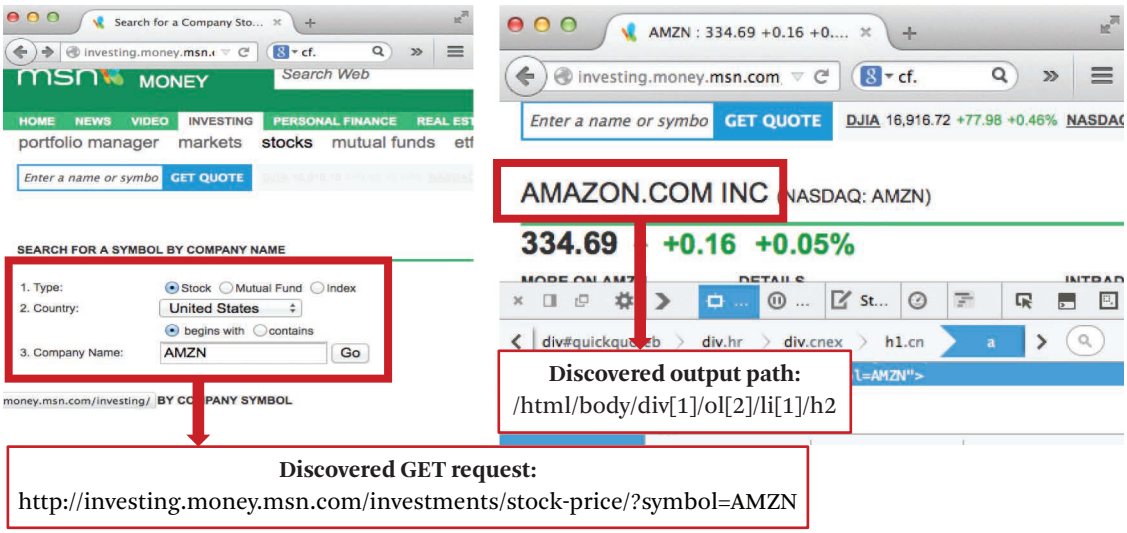


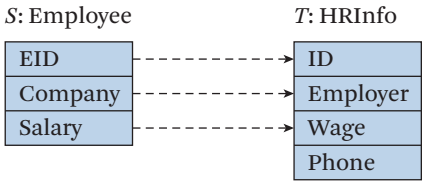
Figure 4.11 The wrapper for this web form consists of a GET URI, the request parameter `?symbol`, and the output path [Abedjan et al. 2016b].

the output field. Given the path, DataXFormer can identify the transformation for any subsequent stock symbol.

4.2.2 Data Exchange

Data exchange [Fagin et al. 2005a, Libkin 2006, Arenas et al. 2014] is the process of taking data under a source schema and transforming it into data under a target schema, and thus can be seen as a form of semantic data transformation. In a data exchange setting, there is a source schema S and a target schema T. The relationship between S and T is captured by *source-to-target dependencies* Σ_{st} that specify how and which source data should be translated into the target. The source-to-target dependencies are of the form $\forall \bar{x}\, \phi(\bar{x}) \rightarrow \exists \bar{y}\, \psi(\bar{x}, \bar{y})$, where $\phi(\bar{x})$ and $\psi(\bar{x}, \bar{y})$ are atoms defined over S and T, respectively. An atom in mathematical logic is an atomic formula with no deeper propositional structure. In addition, since T may be an independently created schema, it may have some additional constraints Σ_t. Given a data exchange scenario determined by S, T, Σ_{st}, and Σ_t, the *data exchange problem* is as follows: given a source instance I over the source schema S, materialize a target instance J over the target schema T, such that the source-to-target dependencies Σ_{st} are satisfied by I and J and the target dependencies Σ_t are satisfied by J. The target instance J is called a *solution* for the data exchange problem. For a data

RIGHTSLINK



I

EID	Company	Salary
1001	Corp 1	110,000
1002	Corp 2	120,000

J_1

ID	Employer	Wage	Phone
1001	Corp 1	110,000	$NULL_1$
1002	Corp 2	120,000	$NULL_2$

J_2

ID	Employer	Wage	Phone
1001	Corp 1	110,000	123-456-0000
1002	Corp 2	120,000	123-456-1111

J_3

ID	Employer	Wage	Phone
1001	Corp 1	110,000	NULL
1002	Corp 2	120,000	NULL

J_4

ID	Employer	Wage	Phone
1001	Corp 1	110,000	$NULL_1$
1002	Corp 2	120,000	$NULL_2$
1003	Corp 3	130,000	$NULL_3$

Figure 4.12 A data exchange example.

exchange problem, there might exist multiple solutions. Hence, several technical problems arise: When does a solution exist? If there are multiple solutions, which one is a "good" solution? How to compute a good solution efficiently?

Example 4.6 Figure 4.12 shows a data exchange scenario where the source schema S contains an employee table with EID, company, and salary columns, and the target schema T contains an HRInfo table with ID, employer, wage, and phone columns. The correspondences between the two schemas are marked in dash lines, which can either be manually annotated by users or detected automatically using schema matching techniques. The mapping is then expressed using source-to-target tuple-generating dependencies Σ_{st} as: $\forall e, c, s\ Employee(e, c, s) \rightarrow \exists p\ HRInfo(e, c, s, p)$.

Note that Σ_{st} may not completely specify the target instance. In this example, the Phone column of T is not specified. Figure 4.12 shows four different solutions for the input I, all of which satisfy Σ_{st}, where $NULL$, $NULL_1$, $NULL_2$, and $NULL_3$ represent "unknown" values, or labeled nulls. Solution J_2 does not use labeled nulls; instead, it uses concrete values ("123-456-0000" and "123-456-1111") for the Phone column, and hence is less general than Solution J_1. While Solution J_3 uses labeled nulls, it is also less general than J_1 since it assumes that the Phone values of two tuples are the same, which was not specified by Σ_{st}. Similarly, Solution J_4 has an additional

RIGHTSLINK

tuple that was not specified Σ_{st}. In contrast, Solution J_1 is a "good" solution, since it contains no more and no less than what Σ_{st} specifies.

The "good" solution that contains no more and no less information than what Σ_{st} specifies is called a *universal solution* [Fagin et al. 2005a]. To introduce the formal definition of a universal solution, we first need to introduce the notion of homomorphism. Let K_1 and K_2 be two database instances over the same schema R with values being a set of constant values and labeled nulls. Let h be a function that maps from the set of constant values and labeled nulls to the set of constant values and labeled nulls. h is a homomorphism from K_1 to K_2 if: (1) $h(c) = c$, for every constant value; and (2) if $R(a_1, a_2, \ldots, a_n)$ is a fact in K_1, then $R(h(a_1), h(a_2), \ldots, h(a_n))$ is also a fact in K_2. Given a data exchange problem, a solution J is a *universal solution* if, for every possible solution K, there exists a homomorphism from J to K.

Example 4.7 In Figure 4.12, Solution J_1 is a universal solution. It has a homomorphism to J_2, J_3, and J_4. For instance, the homomorphism h from J_1 to J_2 is as follows: $h(c) = c$, for $c \in \{1001, 1002, \text{Corp1}, \text{Corp2}, 110{,}000, 120{,}000\}$, $h(NULL_1) = 123 - 456 - 0000$, and $h(NULL_2) = 123 - 456 - 1111$.

The classical chase procedure [Maier et al. 1979] can be used to compute a universal solution if one exists. If the chase procedure succeeds, the result is a universal solution. If the chase fails, then no solution exists. However, for an arbitrary set of dependencies, there may not exist a finite chase. Fagin et al. [2005a] showed that if Σ_{st} and Σ_t satisfy certain conditions (Σ_{st} is a set of tuple-generating dependencies and Σ_t is the union of a weakly acyclic set of tuple-generating dependencies with a set of equality-generating dependencies), then a universal solution exists if and only if a solution exists. They also gave a polynomial-time chase algorithm that determines whether a solution exists and gives a universal solution.

Another important problem in a data exchange setting is query answering, specifically how to answer a query Q against the target schema T, given a source instance I. The answer to Q over the target schema is ambiguous since, as we discussed earlier, there might be multiple solutions J, and each solution might produce a different answer. This difficulty was also present in the context of query answering over incomplete databases [van der Meyden 1998], where the query can be answered over many possible instances of the database. The "certain answer" semantics was adopted to resolve this problem; namely, a tuple appears in the final answer of the query if that tuple appears as an answer of the query against every possible instance of the database. This "certain answer" semantics was also used in the context of data exchange [Fagin et al. 2005a], where every possible solution J is a possible database instance. Given the semantics of query answering in data

RIGHTSLINK

exchange, the question naturally becomes: What is the complexity of answering a query against the target schema? Can we compute the certain answers directly using the "good" solution we choose to materialize? Fagin et al. [2005a] showed that if the target query Q is a union of conjunctive queries, then certain answers of Q can be obtained by directly evaluating Q on a universal solution; they also showed multiple intractability results when Q contains inequalities.

The aforementioned basic data exchange framework is exemplified in the Clio system [Fagin et al. 2009], and it has much follow-up research, including designing more efficient algorithms for computing universal solutions [Gottlob and Nash 2008], answering different types of queries against the target schema [Libkin 2006, Afrati and Kolaitis 2008], studying how target dependencies affect data exchange complexity [Kolaitis et al. 2006, Hernich and Schweikardt 2007], creating schema mappings to generate the source-to-target dependencies [Rahm and Bernstein 2001], using more expressive mapping languages [Fagin et al. 2005b], and developing data exchange solutions in other data models such as XML [Fagin et al. 2009]. A detailed discussion about those topics is outside the scope of this book, and we refer interested readers to a recent survey on data exchange [Arenas et al. 2014].

4.3 ETL Tools

Extract, Transform, and Load (ETL) refers to a three-step data processing pipeline commonly used to extract data from data sources, transform and clean data to a proper format, and load the data in a target environment. ETL tools enable organizations to make their data accessible, meaningful, and usable across disparate data systems. The data transformation functionalities discussed in this chapter are usually offered as part of the ETL tools. Different ETL tools have many different features and functionalities with ever changing new versions and releases, and there are many dimensions for comparing different ETL tools, such as interface usability, batch processing capability, streaming processing capability, available connectors to different data sources, and so on. It is beyond the scope of this book to provide a detailed comparison of features of all existing tools. Rather, we provide prominent examples of current ETL tools with a focus on their data cleaning capabilities according to their respective product documentations at the time of writing.

IBM InfoSphere Information Server[3] is advertised to enable users to clean data and monitor data quality on an ongoing basis, helping to turn data into trusted information. The data cleaning solution offers end-to-end data quality tools to

3. https://www.ibm.com/analytics/information-server

RIGHTSLINK

help users understand data and its relationships, analyze and monitor data quality continuously, clean, standardize and match data, and maintain data lineage.

Informatica Data Quality[4] is advertised to achieve holistic data stewardship for data, regardless of size, format, or platform. Its key data cleaning features include enterprise discovery, search, and profiling that understands the nature of the data and identifies relationships between various data objects, a rich set of data quality transformations, including data standardization, validation, enrichment, and deduplication, a rules builder for business analysts that allows users to build and test logical business rules without relying on IT, and an exception handling that incorporates humans into the workflow, allowing business users to review, correct, and approve exceptions throughout the process.

Microsoft SQL Server Integration Services[5] is a platform for building enterprise-level data integration and data transformations solutions; it has a component called data quality services (DQS).[6] The DQS provides the following features to resolve data quality issues: (1) data cleaning, which is described as the modification, removal, or enrichment of data that is incorrect or incomplete; (2) data matching, the identification of duplicate records; (3) data profiling, the analysis of a data source to provide insights into the quality of data; and (4) data quality monitoring.

Oracle Enterprise Data Quality[7] offers a family of products to help organizations achieve maximum value from their business-critical applications by delivering fit-for-purpose data. Instead of targeting any data domain, it mainly focuses on customer (or more generally party data including suppliers, employees, and so on) and product data. The major features of Oracle Enterprise Data Quality include data profiling, data parsing and standardization, data matching and merging, case management facility for manual cleaning, address verification, and specific rules to handle different product data categories such as resistors, switches, fasteners, and so on.

4.4 Conclusion

Data transformation is a very important data cleaning task that can be applied at various stages of a data analytics pipeline. It can be used to standardize data formats

4. https://www.informatica.com/products/data-quality/informatica-data-quality.html

5. https://docs.microsoft.com/en-us/sql/integration-services/

6. https://docs.microsoft.com/en-us/sql/data-quality-services/introduction-to-data-quality-services

7. http://www.oracle.com/us/products/middleware/data-integration/enterprise-data-quality/overview/index.html

RIGHTSLINK

(e.g., to convert dates expressed in various conventions to a uniform representation), to extract patterns from long strings (e.g., to extract time and codes from system logs), and even to fill in missing values (e.g., to fill in the state values based on the ZIP codes by looking up a external dictionary that maps ZIP to state). This chapter classifies the space of data transformations into syntactic transformations and semantic transformations. Syntactic transformations aim at transforming a table from one syntactic format to another, often without requiring external knowledge or reference data. In contrast, semantic transformations usually involve understanding the meaning/semantics of the data, usually by referencing external data sources, to perform the desired transformations.

We discuss syntactic data transformation systems based on three essential components: *language*, *authoring*, and *execution*. The language component defines the space of allowed syntactic transformations, which could consist of either a finite set of operations allowed or a set of functions for string manipulations. Practical syntactic transformation systems need to achieve a balance between the expressiveness of the language and the efficiency of searching for the right transformation program using a given language. The authoring component defines how the user can author a transformation program using the language. We showed three models for authoring: (1) users specify the transformations directly through a visual interface; (2) users provide a few transformation examples; and (3) the transformation tool proactively suggests potential transformations for users to confirm. The execution component applies the authored transformation programs to the dataset. Practical transformation tools usually offer certain ways to assist users in verifying the correctness of the program, for example, by displaying the results of the transformation on sample data.

For semantic data transformations, example-driven techniques have mostly been used, where users provide several example transformations and then the transformation tool searches for the external data sources that match the provided examples. Those identified data sources are then used to perform the rest of the data transformations. Semantic transformation tools differ mostly on the kind of external data sources used, such as web tables, web forms, and knowledge bases. We also discussed a relevant topic of data exchange, which is a process that transforms structured data under a source schema to structured data under a target schema based on semantic mappings between the two schemas, and can also be considered as semantic transformation.

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

5 Data Quality Rule Definition and Discovery

So far, we have covered some of the most conventional data cleaning tasks, including outlier detection, data deduplication, and data transformation. While these techniques can detect and repair many common errors, they can fall short at cleaning logic errors (e.g., impossible ZIP code and state combination in an address, or when an employee's salary is greater than their manager's, contradicting a predefined business rule). These logic errors are captured by data quality rules, which state certain logics that the data must conform to. Integrity constraints (ICs) provide formal languages to describe various data quality rules. In this chapter, we survey the formal definitions of the integrity constraints proposed to express data quality rules, as well as techniques proposed to discover them automatically. In Chapter 6, we discuss how to use the ICs to clean the data, detecting and repairing errors in a dataset with respect to a set of declared integrity constraints.

Historically, ICs were not proposed to specifically address data quality issues, but rather to study database schema design, including key constraints, functional dependencies, and foreign key constraints [Abiteboul et al. 1995]. Recently, ICs have been increasingly used for data cleaning purposes. To make practical use of dependencies in data quality management, classical dependency theory has to be extended, as traditional types of ICs are often insufficient in capture real-world data errors. The following example illustrates a real-world tax record database that has various data quality problems due to the violations of different data quality rules, some of which cannot be expressed by traditional ICs.

Example 5.1 Consider the U.S. tax records in Table 5.1. Each record describes an individual's address and tax information with 15 attributes: first and last name (FN, LN), gender (GD), area code (AC), mobile phone number (PH), city (CT), state (ST), zip code (ZIP), marital status (MS), has children (CH), salary (SAL), tax rate (TR),

RIGHTSLINK

Table 5.1 Tax data records

TID	FN	LN	GD	AC	PH	CT	ST	ZIP	MS	CH	SAL	TR	STX	MTX	CTX
t_1	Mark	Ballin	M	304	232-7667	Anthony	WA	25813	S	Y	70000	3	2000	0	2000
t_2	Chunho	Black	M	206	154-4816	Seattle	WA	98103	M	N	60000	4.63	0	0	0
t_3	Annja	Rebizant	F	636	604-2692	Cyrene	MO	64739	M	N	40000	6	0	4200	0
t_4	Annie	Puerta	F	501	378-7304	West Crossett	AR	72045	M	N	85000	7.22	0	40	0
t_5	Anthony	Landram	M	319	150-3642	Gifford	IA	52404	S	Y	15000	2.48	40	0	40
t_6	Mark	Murro	M	970	190-3324	Denver	CO	80251	S	Y	60000	4.63	0	0	0
t_7	Ruby	Billinghurst	F	501	154-4816	Kremlin	AR	72045	M	Y	70000	7	0	35	1000
t_8	Marcelino	Nuth	F	304	540-4707	Kyle	WV	25813	M	N	10000	4	0	0	0
t_9	Ann	Puerta	F		378-7304	West Crossett	AR	72045	M	N	86000	7.22	0	40	0

RIGHTSLINK

tax exemption amount if single (STX), tax exemption amount if married (MTX), and tax exemption amount if having children (CTX).

The following data quality rules can be defined:

1. area code and phone identify a person;
2. two persons with the same zip code live in the same state;
3. a person who lives in Los Angeles lives in California;
4. if two persons live in the same state, the one with lower salary has a lower tax rate; and
5. single tax exemption is less than the salary.

While Constraints (1) and (2) can be expressed as a key constraint $Key\{AC, PH\}$ and an FD $ZIP \rightarrow ST$, respectively, the other three constraints cannot be expressed by traditional ICs. We see that Constraint (3) involves a constant value, Constraints (4) and (5) involve order predicates ($>$, $<$), and (5) compares values from different attributes. We introduce more advanced types of ICs, specifically proposed for capturing data quality issues, which can capture these data quality rules.

Deriving a comprehensive set of integrity constraints that accurately reflects an organization's policies and domain semantics is a primary task in using rules for data cleaning. To address this task, many organizations employ consultants and experts who have specific knowledge of business policies to identify integrity constraints. This effort can take a considerable amount of time and cost a lot of money. Furthermore, there may exist domain specific rules in the data that users are not aware of, but that can be useful towards enforcing semantic data consistency. Therefore, developing automatic or semi-automatic techniques to mine for integrity constraints has attracted attention.

There is a natural trade-off between the expressiveness of an IC language and the complexity of an algorithm to mine rules expressed using an IC language. The more expressive an IC language is, the more data quality rules can be captured. With increasing expressiveness of an IC language, it also becomes more complex to discover data quality rules due to the larger search space. For example, domain constraints are a well known type of constraints in databases; they state the possible values a particular column can take (e.g., "*salary of any employee should be within 10K to 100K*"). Domain constraints have limited expressiveness; for example, they cannot capture correlations between two different attributes and they cannot capture conflicts among two different records. However, discovering domain constraints is fairly easy; one can iterate over every column and summarize the values that appear in that column. On the end of the spectrum, programming languages

RIGHTSLINK

such as Python scripts can be used to capture any data quality rules. However, it is extremely difficult to mine for Python scripts.

It is worth noting that the discovery algorithms discussed in this chapter assume the input database is clean. In practice, this can be achieved by asking users to clean a sample of the entire database and using the cleaned sample for discovery. In cases where a sufficiently large clean sample cannot be obtained, the discovery algorithms discussed here can be easily extended or modified into algorithms to discover "approximate" ICs; we discuss one such example using denial constraints [Chu et al. 2013a], the most expressive IC discussed in this chapter. Whether the input dataset is clean or dirty, the discovered ICs may still not be correct as they can be overfitting on the input dataset—they happen to be correct on the input instance, but do not hold on any instance of the same schema. Therefore, it is important to verify any discovered ICs by domain experts before they can be used for data cleaning.

5.1 Functional Dependencies

Consider a relational schema R with attributes $attr(R)$.

Definition 5.1 A functional dependency (FD) φ is defined as $X \rightarrow Y$, where $X \subseteq attr(R)$ and $Y \subseteq attr(R)$. An instance I of R satisfies FD φ, denoted as $I \models \varphi$ if for any two tuples t_α, t_β in I, such that $t_\alpha[X] = t_\beta[X]$, then $t_\alpha[Y] = t_\beta[Y]$.

In other words, if there exist any two tuples, t_α, t_β, in any instance I that have the same value for attributes X but different values for Y, then there must be some errors present in t_α or t_β. We call X the left-hand side (LHS) and Y the right-hand side (RHS). In Example 5.1, the second data quality rule is an FD $ZIP \rightarrow ST$.

Example 5.2 Consider two tuples t_1 and t_8 in Table 5.1; they have the same value "25813" for ZIP, but t_1 has "WA" for ST and t_8 has "WV" for ST. At least one of the four cells $\{t_1[ZIP], t_8[ZIP], t_1[ST], t_8[ST]\}$ has to be incorrect. To limit the space of possible repairs, certain data cleaning algorithms only allow changes on the RHS. In this case, changing $t_1[ST]$ to "WV" or changing $t_8[ST]$ to "WA" would fix the violation of the FD. If changes on the LHS are allowed, changing $t_1[ZIP]$ or $t_8[ZIP]$ to any value other than "25813" would also resolve the violation.

5.1.1 FD Discovery

An FD $\varphi : X \rightarrow A$ is valid with respect to a database instance I if there is no violation of φ on I. FD φ is said to be minimal if removing any attribute from X would make it invalid. Moreover, an FD is trivial if its RHS is a subset of its LHS. Since FDs with multiple attributes in the RHS can be equivalently decomposed into multiple

RIGHTSLINK

FDs with one attribute in the RHS, only FDs with one attribute in the RHS need to be considered. Thus, given a database instance I of schema R, the FD discovery problem is to find all valid minimal nontrivial FDs with one attribute in the RHS that hold on I.

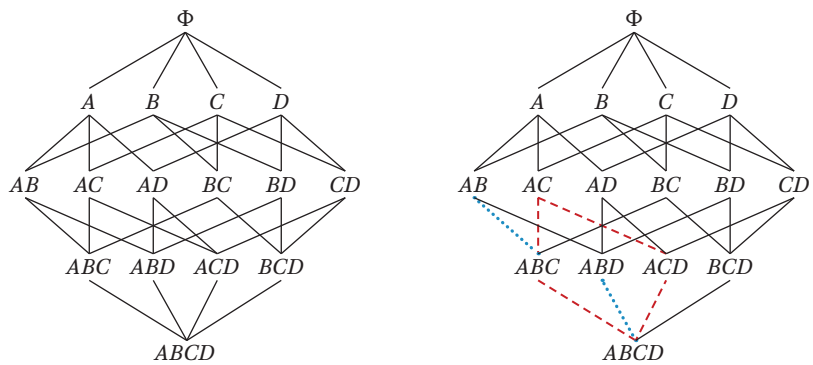
Current FD discovery approaches can be divided into schema-driven and instance-driven approaches. Schema-driven approaches usually have a systematic way of enumerating all the candidate FDs, and also have a way to efficiently check whether a candidate FD is valid or not based on the enumeration procedure. Data-driven approaches start by building a summary of the datasets, and then search for valid FDs directly based on the summary. We present an example technique for each category: TANE [Huhtala et al. 1999], an example of a schema-driven approach, and FASTFD [Wyss et al. 2001], an example of an instance-driven approach. TANE adopts a level-wise candidate generation and pruning strategy and relies on a linear algorithm for checking the validity of FDs. FASTFD first computes difference sets from data, then adopts a heuristic-driven depth-first search algorithm to search for covers of difference sets. TANE is sensitive to the size of the schema, while FASTFD is sensitive to the size of the instance. Papenbrock et al. [2015a] presented an experimental comparison of different functional dependency discovery algorithms.

TANE. Assume the relational schema R has m attributes; $|R| = m$. Selecting an attribute as the RHS of an FD, any subset of the remaining $m - 1$ attributes could serve as the LHS. Thus, the space to be explored for FD discovery is $m \times 2^{m-1}$. Figure 5.1(a) shows the space of candidate FDs organized in a lattice for a table with four columns, A, B, C, and D, where every edge in the lattice represents a candidate FD. For example, edge A to AC represents the FD $A \rightarrow C$.

Algorithm 5.1 describes TANE [Huhtala et al. 1999]. TANE searches the lattice level by level. The level-by-level traversal ensures that only minimal FDs are in the output. There are three types of pruning employed by TANE: (1) if $X \rightarrow A \in \Sigma$, then all FDs of the form $XY \rightarrow A$ are implied, and hence they can be pruned; (2) if $X \rightarrow A \in \Sigma$, then all FDs of the form $XAY \rightarrow B$ can be pruned. The reason is that if $XY \rightarrow B$ is a valid FD, then $XAY \rightarrow B$ is implied by $XY \rightarrow B$, which would be discovered earlier due to level-by-level traversal, and if $XY \rightarrow B$ is not a valid FD, then $XAY \rightarrow B$ is also not valid due to $X \rightarrow A$; and (3) if X is a key, then any node containing X can be pruned.

Partitioning I by X produces a set of nonempty disjoint subsets denoted as Π_X, and each subset contains identifiers of all tuple in I sharing the same value for attributes X. An FD $X \rightarrow A$ is valid if and only if $|\Pi_X| = |\Pi_{X \cup A}|$, where $|\Pi_X|$ denotes the number of disjoint subsets in Π_X. The partitions need not be computed from

RIGHTSLINK



(a) Space of FDs (b) Candidate FDs pruned if $A \rightarrow C$ is valid

Figure 5.1 TANE.

Algorithm 5.1 **TANE**

Input: One relational instance I, schema R
Output: All minimal FDs Σ
$L_1 \leftarrow \{\{A\} | A \in attr(R)\}$
$l \leftarrow 1$
while $L_l \neq \emptyset$ **do**
  for all Node $Y \in L_l$ **do**
    for all Parent node X of Y **do**
      if $X \rightarrow Y - X$ is valid **then**
        add $X \rightarrow Y - X$ to Σ
      end if
    end for
  end for
  pruning L_l based on the three pruning rules
  $L_{l+1} \leftarrow$ generate next level based on L_l
  $l \leftarrow l + 1$
end while

scratch for every set of attributes; rather, TANE computes Π_{XY} from two previously computed partitions, Π_X and Π_Y. Note that Π_{XY} contains all subsets of tuples, where each subset is in both Π_X and Π_Y. For example, if $\Pi_X = \{\{t_1\}, \{t_2, t_3\}, \{t_4\}\}$ and $\Pi_Y = \{\{t_1, t_2, t_3\}, \{t_4\}\}$, then $\Pi_{XY} = \{\{t_1\}, \{t_2, t_3\}, \{t_4\}\}$. Therefore, TANE needs only to compute partitions for every single attribute $A \in R$; partitions for every set

RIGHTSLINK

Algorithm 5.2 **FASTFD**

Input: One relational instance I, schema R
Output: All minimal FDs Σ
for all $A \in attr(R)$ **do**
  calculate D_I^A
end for
for all $A \in attr(R)$ **do**
  Finding all minimal set covers of D_I^A using a depth-first search
  For every cover X, add $X \rightarrow A$ to Σ
end for

of attributes X can be computed from a previous level following the level-by-level traversal.

FASTFD. FASTFD [Wyss et al. 2001] is an instance-based FD discovery algorithm. We start by defining the difference set of two tuples t_1, t_2 as $D(t_1, t_2) = \{A \in attr(R) \mid t_1[A] \neq t_2[A]\}$. The difference sets of I are $D_I = \{D(t_1, t_2) \mid t_1, t_2 \in I,\ D(t_1, t_2) \neq \emptyset\}$. Given a fixed $A \in attr(R)$, the difference sets of I modulo A are $D_I^A = \{D - \{A\} \mid D \in D_I,\ \text{and } A \in D\}$. An FD $X \rightarrow A$ is a valid FD if and only if X covers D_I^A, i.e., X intersects with every element in D_I^A. The intuition is that if X intersects with every element in D_I^A, then X distinguishes any two tuples that disagree on A.

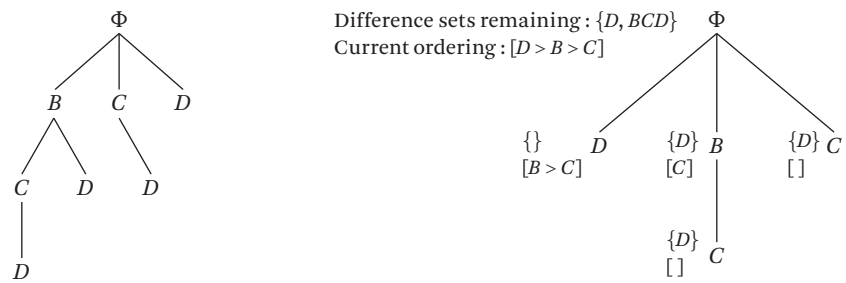
Example 5.3 Consider a table I of R with four attributes as follows:

	A	B	C	D
t_1	a_1	b_1	c_1	d_1
t_2	a_2	b_1	c_1	d_2
t_3	a_1	b_2	c_2	d_1

We have $D(t_1, t_2) = \{AD\}$, $D(t_1, t_3) = \{BC\}$, and $D(t_2, t_3) = \{ABCD\}$. Thus, $D_I = \{AD, BC, ABCD\}$ and $D_I^A = \{D, BCD\}$. Since $\{D\}$ is a minimal cover of D_I^A, we have $D \rightarrow A$.

Therefore, the problem of finding all valid FDs is transformed to the problem of finding all minimal set covers of D_I^A for every attribute $A \in attr(R)$. Every subset of $attr(R) - A$ is a potential candidate minimal cover of D_I^A. Algorithm 5.2 describes FASTFD. In the following, we describe the depth-first search (Line 4) of Algorithm 5.2 using the table in Example 5.3.

RIGHTSLINK



(a) Static ordering of attributes

(b) Dynamic ordering of attributes

Figure 5.2 FASTFD.

To generate all possible minimal set covers for D_I^A, that is, all subsets of $\{BCD\}$, without repetition, the attributes are lexically ordered, i.e., $B > C > D$, and arranged in a depth-first search tree, as shown in Figure 5.2(a). An improved version of the search orders the remaining attributes dynamically according to how many difference sets they cover. Ties are broken lexically. For example, to search for minimal covers of D_I^A using $\{BCD\}$, the attributes are ordered $D > B > C$, since D covers two difference sets while B and C cover one difference set, as shown in Figure 5.2(b). If the algorithm reaches at a node where there are no remaining difference sets left, we have reached a cover X, which may not be minimal. If every immediate subset of X is not a cover, then X is minimal. If a node is reached where there are still remaining difference sets but no attributes left, the depth-first search procedure terminates.

HYFD. As discussed before, schema-based approaches such as TANE scale well with respect to the number of records (row-efficient), but are sensitive to the size of the schema; instance-based approaches such as FASTFD scale well with respect to the number of columns (column-efficient), but are sensitive to the size of the instance. HYFD [Papenbrock and Naumann 2016] is a hybrid FD discovery algorithm that combines the best of both worlds. HYFD proceeds in two phases. In the first phase, HYFD samples a set of records from the entire database and calculates FDs based on this sample using a column-efficient algorithm. Since a small sample is used, the first phase is also efficient with respect to the number of records. The FDs derived from the sample are either valid or almost valid on the entire dataset. In the second phase, HYFD validates the FDs derived in the first phase based on the entire dataset and refines those FDs that do not hold on the entire dataset using a row-

RIGHTSLINK

efficient algorithm. Since only the FDs derived from the first phase are validated, in contrast to schema-based approaches, which need to validate all candidate FDs, the second phase is also efficient with respect to the number of columns.

For Phase 1, HYFD tracks the sampling efficiency, which measures the number of new observations made per sample tuple pair. If the sampling efficiency falls under a certain threshold, the algorithms switches to the second phase. For Phase 2, HYFD tracks validation efficiency, which measures the number of discovered valid FDs per validation. If the validation efficiency falls under a certain threshold, HYFD switches back to the first phase with a relaxed sampling efficiency threshold. In this way, HYFD ensures that both phases are both row-efficient and column-efficient.

The goal of Phase 1 is to derive candidate FDs by eliminating all non-FDs. An FD $X \rightarrow Y$ can be invalidated by only one violation, that is, two records with the same X but different Y. Therefore, the sampling strategy in Phase 1 is to sample record pairs rather than individual records, and it aims to select record pairs that can reveal new violations. The sampling strategy will yield a set of non-FDs, which are then converted into a set of FDs represented via a prefix tree called FDTree, introduced by Flach and Savnik [1999]. A node in the FDTree represents multiple FDs with the same LHS, which is denoted by the node's path in the FDTree; the RHS attributes that can form FDs with the current LHS are marked in the bitset vector attached to the node. Figure 5.3 shows the resulting FDTree with four attributes and a set of non-FDs $\{D \not\rightarrow B, A \not\rightarrow D, B \not\rightarrow D, C \not\rightarrow D\}$. Initially, all FDs are considered valid, that is, $\emptyset \rightarrow A, B, C, D$, as shown in Figure 5.3(0). To eliminate an FD $X \rightarrow Y$ from the tree, all generalizations of the FDs that are of the form $X' \rightarrow Y$ with $X' \subset X$ must be eliminated as well, since all the generalizations must also be invalid FDs. Once all the generalizations are removed, the non-FD $X \rightarrow Y$ is specialized, meaning that the LHS of the FD is extended to generate still valid FDs. Figure 5.3(1) shows the FDTree after processing the non-FD $D \not\rightarrow B$. In this case, $\emptyset \rightarrow B$ is the only generalization and thus is eliminated; specializations including $A \rightarrow B$ and $C \rightarrow B$ are added to the FDTree. Figure 5.3(1) shows the FDTree after processing all the non-FDs.

The goal of Phase 2 is to validate all the FDs stored in the FDTree produced at Phase 1 following a level-wise traversing strategy. Usually, schema-based approaches such as TANE need to traverse a huge lattice containing all candidate FDs. HYFD, however, simply needs to validate a much smaller set of FDs, which are stored in the FDTree. HYFD traverses the FDTree level by level, and stops and switches back to Phase 1 if the validation efficiency goes below a threshold.

RIGHTSLINK

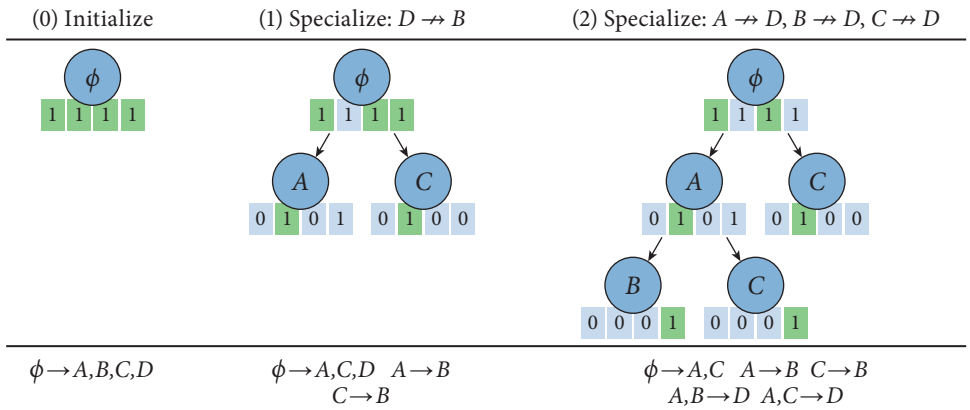


Figure 5.3 Converting non-FDs to FDTree [Papenbrock and Naumann 2016].

5.2 Conditional Functional Dependencies

FDs are not sufficient to capture certain semantics of data. Conditional functional dependencies (CFDs), an extension of FDs, are capable of capturing FDs that hold partially on the data [Bohannon et al. 2007].

Definition 5.2 A CFD φ on R is a pair $(R : X \rightarrow Y, T_p)$, where:

- $X, Y \subset R$;
- $X \rightarrow Y$ is an FD, called *embedded FD* in the context of CFD; and
- T_p is called a *pattern tableau* of φ, where for every attribute $A \in X \cup Y$ and each pattern tuple $t_p \in T_p$, either $t_p[A]$ is a constant in the domain $Dom(A)$ of A, or $t_p[A]$ is a wild card '-'.

A tuple $t_\alpha \in I$ is said to *match* a pattern tuple $t_p \in T_p$, denoted as $t_\alpha \approx t_p$, if for every attribute $A \in X \cup Y$, $t_\alpha[A] = t_p[A]$, in case $t_p[A]$ is a constant. A relation instance I of R is said to satisfy a CFD φ, denoted as $I \models \varphi$, if for every tuple $t_\alpha, t_\beta \in I$, and for each tuple $t_p \in T_p$, if $t_\alpha[X] = t_\beta[X] \approx t_p[X]$, then $t_\alpha[Y] = t_\beta[Y] \approx t_p[Y]$.

Intuitively, a CFD is a traditional FD with an added constraint of the pattern tableau. If, for two tuples $t_\alpha, t_\beta \in I$, $t_\alpha[X]$, and $t_\beta[X]$ are equal and they both match $t_p[X]$, then $t_\alpha[Y]$ and $t_\beta[Y]$ must also be equal and must both match the pattern $t_p[Y]$.

Example 5.4 Consider a table of sales records in Table 5.2 and an FD {name, type, country} $\rightarrow$ {price, tax}. The FD does not hold on the entire relation (e.g., t_6 and t_7 are violating

RIGHTSLINK

Table 5.2 Sales data records [Golab et al. 2008]

TID	Name	Type	Country	Price	Tax
t_1	Harry Potter	book	France	10	0
t_2	Harry Potter	book	France	10	0
t_3	Harry Potter	book	France	10	0.05
t_4	The Lord of the Rings	book	France	25	0
t_5	The Lord of the Rings	book	France	25	0
t_6	Algorithms	book	USA	30	0.04
t_7	Algorithms	book	USA	40	0.04
t_8	Armani suit	clothing	UK	500	0.05
t_9	Armani suit	clothing	UK	500	0.05
t_{10}	Armani slacks	clothing	UK	250	0.05
t_{11}	Armani slacks	clothing	UK	250	0.05
t_{12}	Prada shoes	clothing	USA	200	0.05
t_{13}	Prada shoes	clothing	USA	200	0.05
t_{14}	Prada shoes	clothing	France	500	0.05
t_{15}	Spiderman	DVD	UK	19	0
t_{16}	Star Wars	DVD	UK	29	0
t_{17}	Star Wars	DVD	UK	25	0
t_{18}	Terminator	DVD	France	25	0.08
t_{19}	Terminator	DVD	France	25	0
t_{20}	Terminator	DVD	France	20	0

this FD), but it holds if (1) type is "clothing"; (2) country is "France" and type is "book"; or (3) country is "UK".

The constraints, however, can be expressed by the CFD ({name, type, country} $\rightarrow$ {price, tax}, T_p), with T_p, as shown in Figure 5.4.

While it requires two tuples to have a violation of an FD, one tuple may also violate a CFD. A single tuple t violates a CFD if t matches the LHS of a tuple t_p in the pattern tableau but not the RHS, where t_p consists of all constants, i.e., no wild cards, traditionally referred to as "tuple check constraint."

RIGHTSLINK

Name	Type	Country	Price	Tax
—	clothing	—	—	—
—	book	France	—	0
—	—	UK	—	—

Figure 5.4 T_p for the CFD ({name, type, country} → {price, tax}, T_p).

5.2.1 CFD Discovery

Similar to FD discovery, we discover nontrivial, minimal CFDs with only one attribute in the RHS that hold on a given database instance. The CFD discovery problem is challenging for two reasons: (1) the number of all possible embedded FDs is exponential in the number of attributes in the schema, which is shared by the problem of FD discovery; and (2) the number of all possible constants in the pattern tableau is huge, a challenge unique to CFD discovery.

Candidate CFDs can be generated according to the same lattice used in FD discovery (cf. Figure 5.1) [Chiang and Miller 2008]. Unlike FD discovery, where each edge in the lattice corresponds to one candidate FD, in CFD discovery, each edge corresponds to multiple candidate CFDs. Specifically, an edge (X, XA) generates CFDs of the form $[Q, P] \rightarrow A$ consisting of *variable attributes* P and *conditional attributes* Q, where $X = P \cup Q$. The conditional attributes are those attributes that appear as constants in T_p. The same strategy used in TANE is employed to traverse the lattice level by level, and to reduce the computation at each level by using the results from previous levels [Chiang and Miller 2008]. Several interestingness measures, e.g., support, χ^2 test, and conviction, are proposed for discovered CFDs to avoid returning an unnecessarily large number of CFDs.

Three CFD discovery methods that require that the number of tuples matching the pattern tableau should be above a minimum threshold were proposed [Fan et al. 2011a]. The first method, CFDMiner, aims at discovering constant CFDs, i.e., CFDs with pattern tableau containing only constants. CFDMiner leverages the similarity between the problem of discovering constant CFDs and the problem of mining free and closed itemsets: constant CFDs correspond to association rules with 100% confidence. The second method, CTANE, extends TANE for FD discovery and uses a level-wise search strategy, which is similar to the search strategy used in Chiang and Miller [2008]. The third method, FASTCFD, extending FASTFD for FD discovery, employs a depth-first search strategy. A novel pruning strategy used in FASTCFD is to use constant CFDs already discovered by CFDMiner. CTANE is sensitive to the

RIGHTSLINK

number of attributes in the schema, while FASTCFD is sensitive to the number of tuples in the database.

If the embedded FD is given, the CFD discovery problem becomes that of generating a near-optimal pattern tableau [Golab et al. 2008]. The "goodness" of the pattern tableau is defined by the support and the confidence, where the support of a pattern tableau is defined as the fraction of tuples in the database that match the LHS of the pattern tuples in the pattern tableau, and the confidence of a pattern tableau is defined as the maximum fraction of tuples in the database that do not violate the pattern tableau. The optimal pattern tableau generation problem will be further discussed in Section 6.2.3 under the context of repairing existing data quality rules such that data conforms to the newly modified rules.

5.3 Denial Constraints

As powerful as CFDs are, they are still not capable of capturing many real-life data quality rules, such as the fourth rule, that is "if two persons live in the same state, the one with lower salary has a lower tax rate," and the fifth rule, that is "single tax exemption is less than salary," in Example 5.1. Denial constraints (DCs), a universally quantified first order logic formalism, which subsume FDs and CFDs, describe all data quality rules in Example 5.1. The formal definition of DCs is given as follows.

Definition 5.3 A denial constraint (DC) φ on R is defined as: $\forall t_\alpha, t_\beta, t_\gamma, \ldots \in R, \neg(P_1 \wedge \ldots \wedge P_m)$, where each *predicate* P_i is of the form $v_1 \theta v_2$ or $v_1 \theta c$ with $v_1, v_2 \in t_x.A$, $x \in \{\alpha, \beta, \gamma, \ldots\}$, $A \in R$, c is a constant in the domain of A, and $\theta \in \{=, <, >, \neq, \leq, \geq\}$.

A relation instance I of R is said to satisfy a DC φ, denoted as $I \models \varphi$, if for every ordered list of tuples $\forall t_\alpha, t_\beta, t_\gamma, \ldots \in I$, at least one of P_i is false.

For a DC φ according to the definition, if $\forall P_i$, $i \in [1, m]$ is of the form $v_1 \phi v_2$, then we call such DC a *variable denial constraint* (VDC); otherwise, φ is a *constant denial constraint* (CDC).

A DC states that all the predicates cannot be true at the same time, otherwise we have a violation. Single-tuple constraints (such as check constraints), FDs, and CFDs are special cases of unary and binary denial constraints with equality and inequality predicates.

Example 5.5 DCs are expressive enough to capture all data quality rules in Example 5.1:

1. area code and phone identify a person

$$c_1 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.AC = t_\beta.AC \wedge t_\alpha.PH = t_\beta.PH);$$

RIGHTSLINK

2. two persons with the same zip code live in the same state

$$c_2 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.ZIP = t_\beta.ZIP \wedge t_\alpha.ST \neq t_\beta.ST);$$

3. a person who lives in Los Angeles lives in California

$$c_3 : \forall t_\alpha \in R, \neg(t_\alpha.CT = \text{'Los Angeles'} \wedge t_\alpha.ST \neq \text{'CA'});$$

4. if two persons live in the same state, the one with lower salary has a lower tax rate

$$c_4 : \forall t_\alpha, t_\beta \in R, \neg(t_\alpha.ST = t_\beta.ST \wedge t_\alpha.SAL < t_\beta.SAL \wedge t_\alpha.TR > t_\beta.TR); \text{ and}$$

5. single tax exemption is less than salary

$$c_5 : \forall t_\alpha \in R, \neg(t_\alpha.SAL < t_\alpha.STX).$$

5.3.1 DC Discovery

A DC φ is valid with respect to a database instance I if there is no violation of φ on I. A DC φ is said to be minimal if removing any predicate from φ would make it invalid. Moreover, an DC is trivial if it is satisfied by any instance I. Thus, given a database instance I of schema R, the DC discovery problem is to find all valid minimal nontrivial DCs that hold on I.

While there are both schema-driven and instance-driven approaches for FD discovery, pure schema-driven approaches for DC discovery are less applicable. The main reason is that the structure of DCs is more complicated than that of FDs; FDs only involve equality, while DCs involve predicates "greater than" and "less than". Therefore, it is difficult to design an efficient way to systematically check the validities of the space of all DCs. There are currently two main algorithms for DC discovery: FASTDC [Chu et al. 2013a] and Hydra [Bleifuß et al. 2017]. FASTDC is an instance-driven algorithm, similar to FASTFD for FD discovery, while Hydra is a hybrid algorithm, similar to HYFD for FD discovery.

FASTDC. FASTDC [Chu et al. 2013a, Chu et al. 2014] is proposed as an extension of FASTFD for DC discovery. To define the space of DCs, we first need to define the *space of predicates* **P**, since DCs are composed of predicates. Then, the *evidence set* Evi_I is built. The *evidence set* Evi_I is a set where each element in Evi_I is a subset of predicates in **P** that are satisfied by a tuple pair in I. The evidence set has a similar

RIGHTSLINK

functionality to the difference set in FASTFD in that each minimal set cover for Evi_I corresponds to a valid minimal DC.

Example 5.6 Consider the following employee table with three attributes: Employee ID (I), Manager ID (M), and Salary (S)

TID	I(String)	M(String)	S(Double)
t_9	A1	A1	50
t_{10}	A2	A1	40
t_{11}	A3	A1	40

For each attribute in the schema, we add two equality predicates ($=, \neq$) between two tuples on it. In the same way, for each numerical attribute, we add order predicates ($>, \leq, <, \geq$). For every pair of attributes in R, they are *joinable* (*comparable*) if equality (order) predicates hold across them, and we add cross column predicates accordingly. We build the following predicate space $\mathbf{P}$ for it:

$$
\begin{array}{lll}
P_1 : t_\alpha.I = t_\beta.I & P_5 : t_\alpha.S = t_\beta.S & P_9 : t_\alpha.S < t_\beta.S \\
P_2 : t_\alpha.I \neq t_\beta.I & P_6 : t_\alpha.S \neq t_\beta.S & P_{10} : t_\alpha.S \geq t_\beta.S \\
P_3 : t_\alpha.M = t_\beta.M & P_7 : t_\alpha.S > t_\beta.S & P_{11} : t_\alpha.I = t_\alpha.M \\
P_4 : t_\alpha.M \neq t_\beta.M & P_8 : t_\alpha.S \leq t_\beta.S & P_{12} : t_\alpha.I \neq t_\alpha.M \\
P_{13} : t_\alpha.I = t_\beta.M & P_{14} : t_\alpha.I \neq t_\beta.M &
\end{array}
$$

Given a pair of tuples $\langle t_x, t_y \rangle \in I$, the satisfied predicate set for $\langle t_x, t_y \rangle$ is $SAT(\langle t_x, t_y \rangle) = \{P | P \in \mathbf{P}, \langle t_x, t_y \rangle \models P\}$, where $\mathbf{P}$ is the predicate space and $\langle t_x, t_y \rangle \models P$ means $\langle t_x, t_y \rangle$ satisfies P. The *evidence set* of I is $Evi_I = \{SAT(\langle t_x, t_y \rangle) | \forall \langle t_x, t_y \rangle \in I\}$. A set of predicates $\mathbf{X} \subseteq \mathbf{P}$ is a *minimal set cover* for Evi_I if $\forall E \in Evi_I, \mathbf{X} \cap E \neq \emptyset$, and $\nexists \mathbf{Y} \subset \mathbf{X}$, such that $\forall E \in Evi_I, \mathbf{Y} \cap E \neq \emptyset$.

Example 5.7 $Evi_{Emp} = \{\{P_2, P_3, P_5, P_8, P_{10}, P_{12}, P_{14}\}, \{P_2, P_3, P_6, P_8, P_9, P_{12}, P_{14}\}, \{P_2, P_3, P_6, P_7, P_{10}, P_{11}, P_{13}\}\}$. Every element in Evi_{Emp} has at least one pair of tuples in I such that every predicate in it is satisfied by that pair of tuples.

$\mathbf{X}_1 = \{P_2\}$ is a minimal cover; thus $\neg(\overline{P_2})$, i.e., $\neg(t_\alpha.I = t_\beta.I)$ is a valid DC, which states I is a key.

$\mathbf{X}_2 = \{P_{10}, P_{14}\}$ is another minimal cover; thus $\neg(\overline{P_{10}} \wedge \overline{P_{14}})$, i.e., $\neg(t_\alpha.S < t_\beta.S \wedge t_\alpha.I = t_\beta.M)$ is another valid DC, which states that a manager's salary cannot be less than their employee's.

Algorithm 5.3 describes FASTDC, which first builds the space of predicates and the evidence set, then searches for all minimal set covers for the evidence set.

RIGHTSLINK

Algorithm 5.3 **FASTDC**

Input: One relational instance I, schema R
Output: All minimal DCs Σ
$\mathbf{P} \leftarrow$ building the predicate space based on R and I
$Evi_I \leftarrow$ building the evidence set based on I and P
$\mathbf{MC} \leftarrow$ search for all minimal covers of Evi_I
for all $\mathbf{X} \in MC$ **do**
    $\Sigma \leftarrow \Sigma + \neg(\overline{\mathbf{X}})$
end for
rank DCs in Σ based on their interestingness

Every minimal set cover corresponds to a minimal DC. FASTDC follows a depth-first search procedure to exhaustively search for all minimal set covers. It includes multiple optimizations and pruning opportunities based on the properties of DCs to speed up the search procedure.

Although FASTDC is able to prune the search space effectively, the number of DCs returned can still be too large. To address this problem, FASTDC proposes a scoring function to rank DCs based on their size and their support from the data. Given a DC φ, we denote by $Inter(\varphi)$ its *interestingness* score. FASTDC recognizes two different dimensions that influence $Inter(\varphi)$: *succinctness* and *coverage* of φ, which are both defined on a scale between 0 and 1. Each of the two scores represents a different yet important intuitive dimension that should be taken into account when ranking discovered DCs. Given a DC φ, FASTDC defines the interestingness score as a linear weighted combination of the two dimensions. Succinctness is motivated by the Occam's razor principle, which suggests that among competing hypotheses, the one that makes fewer assumptions is preferred. Minimum description length (MDL), which measures the code length needed to compress the data [Bishop 2006], is a formalism to realize the Occam's razor principle. Inspired by MDL, we measure the length of a DC $Len(\varphi)$, and we define the *succinctness* of a DC φ, i.e., $Succ(\varphi)$, as the minimal possible length of a DC divided by $Len(\varphi)$, thus ensuring the scale of $Succ(\varphi)$ is between 0 and 1. Coverage or support measures the statistical significance of discovered DCs. For example, in frequent itemset mining [Agrawal et al. 1993], the support of an itemset is defined as the proportion of transactions in the data that contain the itemset. Only if the support of an itemset is above a threshold is it considered to be frequent. The coverage of DCs is different from that of frequent itemsets—counting the number of tuple pairs that do not violate a DC is a meaningless measure for coverage. Therefore, we need a different way of defining

RIGHTSLINK

coverage. Intuitively, for every tuple pair, not all the predicates in a DC should be satisfied. Depending on the number of satisfied predicates, different tuple pairs give different support to the statistical significance score of a DC. The larger the number of satisfied predicates in a DC φ given a tuple pair, the more support that tuple pair gives to φ. The coverage of φ is thus defined as the average support it gets from all the tuple pairs.

Algorithm FASTDC consumes the entire input dataset and requires no violations for a DC to be declared valid. However, in real world settings, the input data might be dirty, and thus a potentially valid DC might contain violations in the input data. The main assumption used to deal with dirty input data is that although the input data is dirty, most of it should be clean. Therefore, a candidate DC can still be considered to be a valid DC if most of the data conforms to it. Based on this assumption, besides looking for exact covers for the evidence set, FASTDC also looks for approximate covers, which correspond to approximately valid DCs.

Hydra. Hydra [Bleifuß et al. 2017] is a hybrid DC discovery algorithm that overcomes the quadratic runtime complexity in the number of tuples in the FASTDC algorithm by avoiding comparing all tuple pairs. This is possible because if a tuple pair already violates a candidate DC, then any other tuple pairs are redundant in checking the validity of that DC. Furthermore, a tuple pair is also redundant if all predicates that are fulfilled by that pair are also fulfilled by another pair. While in theory it is possible that no tuple pair is redundant in a dataset, Hydra empirically shows that the vast majority of tuple pairs is in fact redundant. To avoid considering all tuple pairs, the general idea of Hydra is to determine a preliminary set of DCs on a sample of tuple pairs. Hydra discovers all tuple pairs that are in violation of the preliminary DCs. Eventually, the combination of the sampled tuple pairs and the violating tuple pairs will determine all valid DCs.

Figure 5.5 shows the basic steps in Hydra. Hydra takes as input a relational table and a predicate space. Hydra first samples tuple pairs from the relational table ((1) in Figure 5.5). For each sampled tuple pair, evidence is computed, namely, the set of predicates in the predicate space that are evaluated to be true for the sampled tuple pair. Since duplicate evidence does not provide new information for discovering DCs, the sampling stage attempts to maximize the number of non-redundant tuple pairs sampled. At the end of the sampling stage, Hydra obtains a "close-to-complete" evidence set, called preliminary evidences, from only a fraction of all tuple pairs in the dataset ((2) in Figure 5.5). The preliminary evidence is then used to compute a set of preliminary DCs ((3) in Figure 5.5) using an evidence inversion process, which is an iterative process similar to the depth-first search procedure used

RIGHTSLINK

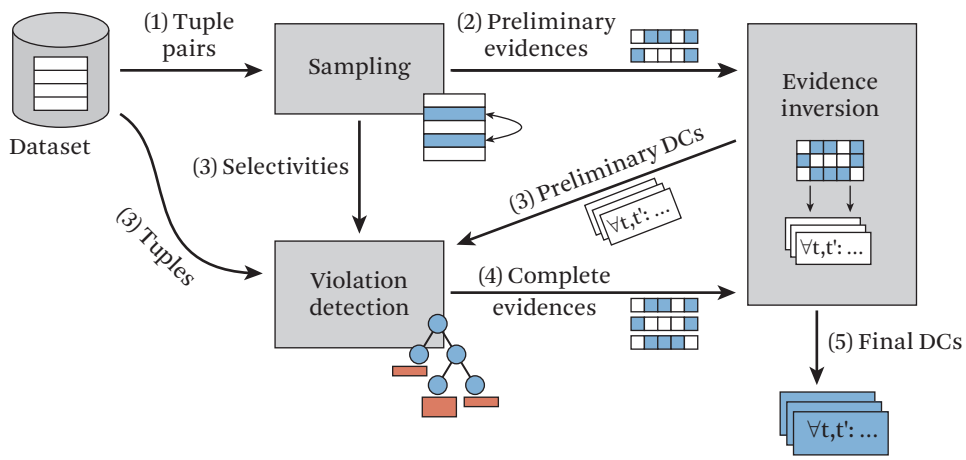


Figure 5.5 Overview of Hydra [Bleifuß et al. 2017].

in FASTDC to discover minimum set covers. This preliminary set of DCs, of course, could be violated by tuple pairs that are not included in the sample. Therefore, Hydra employs an efficient scheme to discover tuple pairs that violate preliminary DCs using the violation detection procedure in Figure 5.5. Instead of naively enumerating all tuple pairs to find violating tuple pairs for the preliminary DCs, Hydra employs specific data structures and dedicated evaluation algorithms for various predicate types. For example, Hydra uses clusters of tuples to efficiently evaluate equality predicates involving some attribute A, where each cluster represents tuples that have the same value for A, and uses cluster pair (c_1, c_2) to efficiently evaluate inequality predicates involving some attribute A, where $t[A] < t'[A]$ for $t \in c_1$ and $t' \in c_2$. Finally, the preliminary set of evidences combined with the violating tuple pairs found with respect to the preliminary DCs forms the complete evidence set ((4) in Figure 5.5). The complete set of evidences is then used to compute the final DCs ((5) in Figure 5.5)) using again the evidence inversion procedure.

5.4 Other Types of Constraints

Multiple types of constraints have been proposed for different purposes. Inclusion dependencies (INDs) [Abiteboul et al. 1995] can be used for detecting inconsistencies or information incompleteness and schema matching. Matching dependencies (MDs) [Fan et al. 2009] use similarity measures to generalize the equality condition used in FDs, to support record linkage across two tables. Metric functional dependencies (MFDs) [Koudas et al. 2009] can be considered as special MDs

RIGHTSLINK

defined on one table, to capture small variations in the data. Numeric functional dependencies (NFDs) [Fan et al. 2014a] can capture interesting constraints involving numeric attributes, since NFDs allow arithmetic operations; editing rules (eRs) [Fan et al. 2010] not only provides a way to detect errors, but also tells how to fix errors by referencing a master table. Fixing rules [Wang and Tang 2014] precisely capture which attribute is wrong and how to correct the error when enough evidence is present. Sherlock rules [Interlandi and Tang 2015] annotate the correct and erroneous attributes, and precisely tell how to fix the errors by referencing master tables.

5.4.1 Inclusion Dependency

Inclusion dependencies (INDs) [Abiteboul et al. 1995], which are a generalization of referential constraints, can be used for detecting inconsistencies or information incompleteness [Bohannon et al. 2005] and in schema matching systems (e.g., Haas et al. [2005]).

Definition 5.4 An inclusion dependency φ for two relations (R_1, R_2) is defined as $R_1[X] \subseteq R_2[Y]$, where (X_1, X_2) are lists of attributes in (R_1, R_2), and $|X| = |Y|$. An instance pair (I_1, I_2) satisfies IND φ if for any tuple $t_\alpha \in I_1$, there exists a tuple $t_\beta \in I_2$, such that $t_\alpha[X] = t_\beta[Y]$.

For any $t_\alpha \in I_1$, if there is no tuple $t_\beta \in I_2$ that satisfies $t_\alpha[X] = t_\beta[Y]$, then either $t_\alpha \in I_1$ is incorrect or there is a tuple missing in R_2. Inclusion dependencies may also refer to only one relation, i.e., R_1 is the same as R_2.

Example 5.8 Denote the relation about tax records in Example 5.1 as Relation R_1, and a second employee relation R_2(EID, FirstName, LastName) that keeps all the employee IDs and their first and last names. A valid IND would be R_1[FN,LN] $\subseteq$ R_2[FirstName, LastName], which means that if there is a tax record about a person, then that person must appear in the employee table.

De Marchi et al. [2009] discover unary INDs, that is, INDs with one attribute in X and Y, using a two-step process by first building an inverted index pointing every value in the database to the set of all attributes containing the value, and then retrieving valid INDs using set intersections. N-ary INDs are discovered following a level-wise approach similar to TANE. BINDER [Papenbrock et al. 2015b] does not assume that the dataset fits into main memory, and discovers INDs in a scalable manner based on a divide and conquer strategy. INDs are later extended to conditional inclusion dependencies (CINDs) [Ma et al. 2014], which are INDs that hold

RIGHTSLINK

on subset of the tuples. Just like the extension of CFDs to FDs, CINDs have more expressive power than INDs.

5.4.2 Matching Dependency

MDs [Fan et al. 2009] use similarity measures to generalize the equality condition used in FDs. While FDs are defined on a single relation, MDs are defined on two relations.

Definition 5.5 A matching dependency φ for two relations (R_1, R_2) is defined as follows:

$$\wedge_{j\in[1,k]}(R_1[X_1[j]] \approx_j R_2[X_2[j]]) \rightarrow R_1[Z_1] \rightleftharpoons R_2[Z_2],$$

where

- (X_1, X_2) are lists of attributes in (R_1, R_2), $X_1[j]$, $X_2[j]$ denotes the jth attribute in X_1, X_2;
- for every $j \in [1, k]$, $X_1[j]$ and $X_2[j]$ are comparable, i.e., they belong to the same domain;
- for every $j \in [1, k]$, $\approx_j$ is a similarity operator, which can be any similarity metric used in record matching, e.g., q-grams or edit distance; and
- $\rightleftharpoons$ is a matching operator. For any two values x, y, $x \rightleftharpoons y$ indicates that x and y are changed to be identical.

Intuitively, an MD φ states that if $R_1[X_1]$ and $R_2[X_2]$ are similar with respect to some similarity metrics, then $R_1[Z_1]$ and $R_2[Z_2]$ should be changed to be identical.

Example 5.9 Consider two relational tables from a bank in the U.K.: card(FN, LN, St, city, AC, zip, tel, dob, gd) maintains customer information collected when credit cards are issued; tran(FN, LN, St, city, AC, post, phn, gd, item, when, where) consists of transaction records of credit cards, which may be dirty. Here a card tuple specifies a U.K. credit card holder identified by first name (FN), last name (LN), address (street (St), city, zip code), area code (AC), phone (tel), date of birth (dob), and gender (gd). A tran tuple is a record of a purchased item paid by a credit card at place where and time when, by a U.K. customer who is identified by name (FN, LN), address (St, city, post code), AC, phone (phn), and gender (gd).

A possible matching rule is that for any tuple in card and any tuple in tran, if they have the same last name and address and, moreover, if their first names are *similar*, then their phone and FN attributes can be identified. This rule can be expressed by an MD ψ: tran[LN, city, St, post] = card[LN, city, St, zip] $\wedge$ tran[FN] $\approx$ card[FN] $\rightarrow$ tran[FN, phn] $\rightleftharpoons$ card[FN, tel].

RIGHTSLINK

Table 5.3 Movie data records integrated from multiple data sources

Source	Title	Duration
movies.aol.com	Aliens	110
finnguide.fi	Aliens	112
amazon.com	Clockwork Orange	137

The MD discovery problem also discovers interesting MDs with high support and confidence [Song and Chen 2009], which is similar to CFD discovery [Chiang and Miller 2008]. Only interested MDs are selected as output.

5.4.3 Metric Functional Dependency

While MDs are for capturing small variations on string attributes, metric functional dependencies (MFDs) [Koudas et al. 2009] are usually used to capture small variations on numerical data.

Definition 5.6 A metric functional dependency (MFD) φ is defined as $X \xrightarrow{\delta} Y$, where X and Y denote subsets of attributes of $attr(R)$. An instance I of R satisfies this FD φ, denoted as $I \models \varphi$ if for any two tuples t_α, t_β in I such that $t_\alpha[X] = t_\beta[X]$, then $d(t_\alpha[Y], t_\beta[Y]) \leq \delta$, where d is a metric function defined on the domain of Y.

An MFD can be seen as a special case of MD where R_1 and R_2 are the same relation and the LHS is exactly matching.

Example 5.10 Consider a table of movies in Table 5.3, resulting from integrating movies from multiple websites. A plausible constraint is that the same movie should have the same duration. However, different websites may have different ways of calculating the duration, for example, depending on whether extra materials, such as the advertisement, are included. An MFD Title $\xrightarrow{5}$ Duration would be more suitable than the FD Title $\rightarrow$ Duration to capture such a constraint.

5.4.4 Numeric Functional Dependency

NFDs [Fan et al. 2014a] are another type of constraint for capturing constraints involving numeric attributes. They are able to capture errors in numeric attributes that FDs, CFDs, and DCs cannot capture.

Definition 5.7 A numeric functional dependency (NFD) φ defined on a relational table $R(A_1, \ldots, A_m)$ is a pair of tables:

RIGHTSLINK

Name	YoB	YoD	Town	Country
—	x	y	—	—

(a) Pattern table T_{P1}

Condition
$y - x \leq 120$

(b) Condition table T_{C1}

SS#	Name	cno	hw	Tests	Lab	Proj
—	—	—	x_1	x_2	x_3	x_4

(c) Pattern table T_{P2}

Condition
$x_1 + x_2 + x_3 + x_4 = 100$

(d) Condition table T_{C2}

	CC#	Name	Street	City	Zip	When	Where	Amnt
p_1:	x_c	—	—	—	—	x_t	Edi	—
p_2:	x_c	—	—	—	—	y_t	NYC	—

(e) Pattern table T_{P3}

Condition
$\lvert x_t - y_t \rvert \geq 2$

(f) Condition table T_{C3}

Figure 5.6 NFDs examples [Fan et al. 2014a].

- a *pattern table* T_p of schema R that has two tuples p_1 and p_2; for $i \in [1, 2]$ and $j \in [1, m]$, $p_i[A_j]$ is a constant in $dom(A_j)$, a variable x, or a wildcard '_'; and
- a *condition table* T_c with a single *condition tuple* of the form *eopz*, where e is either a variable in T_p or a linear arithmetic expression of variables in T_p, *op* is one of the operations in $\{=, \neq, <, \leq, >, \geq\}$, and z is either a constant or a variable in T_p.

We see that NFDs are defined on at most two tuples and can express more constraints involving numeric attributes than FDs, CFDs, and DCs because NFDs allow arithmetic operations.

Example 5.11 Figure 5.6 shows three NFDs on three different tables. (1) The first table specifies a person with his name, year of birth (YoB), year of death (YoD), and origin (country, town). The first NFD with T_{P1} in Figure 5.6(a) and T_{C1} in Figure 5.6(b) says that no one can live more than 120 years. (2) The second table specifies the academic report of courses of a student, with the distribution of the score into homework (*hw*), tests, lab, and projects (proj). The second NFD with T_{P2} in Figure 5.6(c) and T_{C2} in Figure 5.6(d) says that the total percentage has to be equal to 100. (3) The third table is about credit card transactions with each tuple specifying the card number (CC#), the card holder information (name, street, city, zip), when and where the card was used, and the amount charged to the card (amnt). The third NFD, with T_{P3} in Figure 5.6(e) and T_{C3} in Figure 5.6(f), asserts that two transactions involving the

RIGHTSLINK

same credit card, one happening in Edinburgh (Edi) and one happening in New York (NYC), have to be at least two hours apart.

Editing Rules

eRs [Fan et al. 2010] not only provide a way to detect errors but also tell how to fix errors by referencing a master table.

Definition 5.8 An editing rule (eR) φ defined on a relation R and a master relation R_m is a pair $\Big((X, X_m) \rightarrow (B, B_m), t_p[X_p]\Big)$, where:

- X and X_m are two lists of distinct attributes, in R and R_m respectively, with the same number of attributes;
- B is an attribute in $attr(R) - X$, and B_m is an attribute in $attr(R_m) - X_m$; and
- t_p is a pattern tuple over a set of distinct attributes X_p in R, such that for each $A \in X_p$, $t_p[A]$ is one of $_$, a or $\overline{a}$, where a is a constant from the domain of A, $\overline{a}$ is any constant other than a, and $_$ is an unnamed variable.

An eR φ is said to be applicable to a tuple $t \in I$ and a tuple $t_m \in I_m$ to update t to t', denoted as $t \rightarrow t'$, if:

- t and t_m matche on the LHS of φ, i.e., $t[X] = t_m[X_m]$;
- t matches t_p, i.e., $t[X_p] \approx t_p[X_p]$; and
- t' is obtained from t by updating $t[B]$ to be $t_m[B_m]$.

CFDs and eRs are both based on pattern tuples. CFDs are defined on a single relation, while eRs are defined on an input tuple and a master relation. In addition, while CFDs have *static semantics*, i.e., they only tell whether two tuples are in violation or not, eRs have *dynamic semantics*, i.e., they update t to t' if an eR is applicable.

eRs are also similar to MDs in that they both share dynamic semantics. Although both MDs and eRs are defined on two relations, MDs neither have pattern tuples nor master data relation. For two tuples t_1 and t_2, an MD $\wedge_{j \in [1,k]}(R_1[X_1[j]] \approx_j R_2[X_2[j]]) \rightarrow R_1[Z_1] \rightleftharpoons R_2[Z_2]$ only states that $t_1[Z_1]$ and $t_2[Z_2]$ should be identified but does not tell what values are to be taken; however, eRs directly dictate that values from the master relation should be taken.

Example 5.12 Consider the two relational tables defined in Example 5.9, namely, card(FN, LN, St, city, AC, zip, tel, dob, gd) and tran(FN, LN, St, city, AC, post, phn, gd, item, when, where). Assume that card is a *clean master relation*, that is, tuples in card are correct.

A plausible eR φ is that for any tuple t in tran, if there exists a master tuple s in card with t[LN, FN, city, St, post] = s[LN, FN, city, St, zip], the t[phn] should be updated to

RIGHTSLINK

be s[phn], which can be written as φ: $\Big((X, X_m) \rightarrow (\text{phn}, \text{phn}), t_p[X_p] = ()\Big)$, where X ranges over LN, FN, city, St, post, and X_m ranges over LN, FN, city, St, zip.

Editing rule discovery is studied in Diallo et al. [2012] and adapts techniques from CFD discovery.

5.4.5 Fixing Rules

Similar to eRs, fixing rules [Wang and Tang 2014] not only precisely capture which attribute is wrong but also indicate how to correct the error, when enough evidence is present.

Definition 5.9 A fixing rule φ defined on a schema R is formalized as $((X, t_p[X]), (B, T_p^-[B])) \rightarrow t_p^+[B]$, where:

- X is a set of attributes in attr(R) and B is an attribute in attr(R) $\setminus X$ (i.e., B is not in X);
- $t_p[X]$ is a pattern with attributes in X, referred to as the *evidence pattern* on X, and for each $A \in X$, $t_p[A]$ is a constant value in dom(A);
- $T_p^-[B]$ is a finite set of constants in dom(B), referred to as the *negative patterns* of B; and
- $t_p^+[B]$ is a constant value in dom(B) $\setminus T_p^-[B]$, referred to as the *fact* of B.

Intuitively, the evidence pattern $t_p[X]$ of X, together with the negative patterns $T_p^-[B]$, imposes the condition to determine whether a tuple contains an error on B. The fact $t_p^+[B]$ in turn indicates how to correct this error. The last condition in the definition enforces that the correct value (i.e., the fact) is different from known wrong values (i.e., negative patterns) relative to a specific evidence pattern.

A tuple t of R *matches* a rule $\varphi : ((X, t_p[X]), (B, T_p^-[B])) \rightarrow t_p^+[B]$ if (i) $t[X] = t_p[X]$ and (ii) $t[B] \in T_p^-[B]$. Tuple t matches rule φ indicates that φ can identify errors in t.

Example 5.13 Consider a table of travel records, shown in Figure 5.7, for a research institute specified by the following schema: Travel (name, country, capital, city, conf). A Travel tuple specifies a person, identified by name, who has traveled to conference (conf), held at the city of the country with capital. All errors are highlighted and their correct values are given between brackets. For instance, r_2[capital] = `Shanghai` is wrong and its correct value is `Beijing`.

Consider the following two fixing rules defined on Travel: φ_1 : (([country], [`China`]), (capital, {`Shanghai`,`Hongkong`})) $\rightarrow$ `Beijing`, and φ_2 : (([country], [`Canada`]), (capital, {`Toronto`})) $\rightarrow$ `Ottawa`.

RIGHTSLINK

	Name	Country	Capital	City	Conf
r_1:	George	China	Beijing	Beijing	SIGMOD
r_2:	Ian	China	Shanghai (Beijing)	Hongkong (Shanghai)	ICDE
r_3:	Peter	China (Japan)	Tokyo	Tokyo	ICDE
r_4:	Mike	Canada	Toronto	Toronto	VLDB

Figure 5.7 Database D: an instance of schema Travel [Wang and Tang 2014].

In both φ_1 and φ_2, X consists of country and B is capital. Here, φ_1 states that, if the country of a tuple is `China` and its capital value is in {`Shanghai`, `Hongkong`}, its capital value is wrong and should be updated to `Beijing`. Similarly for φ_2, Tuple r_1 does not match rule φ_1, since r_1[country] = `China` but r_1[capital] $\notin$ {`Shanghai`, `Hongkong`}. As another example, tuple r_2 matches rule φ_1, since r_2[country] = `China`, and r_2[capital] $\in$ {`Shanghai`, `Hongkong`}. Similarly, we have r_4 matches φ_2. After applying φ_1 and φ_2, two errors, r_2[capital] and r_4[capital], can be repaired.

To the best of our knowledge, automatic discovery for fixing rules has not been studied.

5.4.6 Sherlock Rules

Sherlock rules [Interlandi and Tang 2015] annotate the correct and erroneous attributes and precisely tell how to fix the errors by referencing master tables. Let I be a table over schema R and M a reference table with schema R_m. The relational schema R is often different from R_m.

Definition 5.10 A Sherlock rule (sR) φ defined on schemas (R, R_m) is formalized as $\varphi : ((X, X_m), (B, B_m^-, B_m^+), \vec{\approx})$, where:

- X and X_m are lists of distinct attributes in schemas R and R_m, respectively, where $|X| = |X_m|$;
- B is an attribute such that $B \in R \setminus X$, and B_m^-, B_m^+ are two distinct attributes in $R_m \setminus X_m$; and
- $\vec{\approx}$ is a vector of similarity operators over comparable attributes, (A, A_m), (B, B_m^-), and (B, B_m^+), where $A \in X$, and A_m is the corresponding attribute in X_m.

RIGHTSLINK

	Name	Dept	Nation	Capital	Bornat	OfficePhn
t_1	Si	DA	China	Beijing	CRChenYang	28098001
t_2	Yan	DA	China	Shanghai	Chengdu	24038698
t_3	Ian	ALT	Chine	Beijing	Hangzhou	33668323

Figure 5.8 I_{EMP}: An instance of the schema EMP [Interlandi and Tang 2015].

	Country	Capital
s_1	China	Beijing
s_2	Japan	Tokyo
s_3	Chile	Santiago

Figure 5.9 M_{cap}: An instance of the schema CAP [Interlandi and Tang 2015].

	Name	OfficePhn	Mobile
r_1	Si	28098001	66700541
r_2	Yan	24038698	66706563
r_3	Ian	27364928	33668323

Figure 5.10 M_{phn}: An instance of the schema PHN [Interlandi and Tang 2015].

Rule φ says that for a pair of tuples t in I and t_m in M, if both $(t[X], t_m[X_m])$ and $(t[B], t_m[B_m^-])$ are similar with respect to some similarity metrics, φ validates that $t[X]$ is correct, $t[B]$ is wrong, and, moreover, the correct value of $t[B]$ is $t_m[B_m^+]$. Intuitively, given that $t[X]$ and $t_m[X_m]$ are similar, $t[B]$ should take its value from $t[B_m^+]$ rather than $t[B_m^-]$. In other words, the rule explicitly captures the possible errors $t[B]$ can make; for example, attributes B and B_m^+ might be an office phone number, and B_m^- might be a mobile phone number.

Example 5.14 Consider the employee table in Figure 5.8 and two reference tables: the capital table in Figure 5.9 and the phone table in Figure 5.10. Three sRs are defined as follows, where φ_1 and φ_2 are defined on (EMP, PHN), and φ_3 is defined on (EMP, CAP):

RIGHTSLINK

$$\varphi_1 : ((\text{name}, \text{name}), (\text{officePhn}, \text{mobile}, \text{officePhn}), (=, =, =))$$
$$\varphi_2 : ((\text{name}, \text{name}), (\text{officePhn}, \text{mobile}, \perp), (=, =, \not\approx))$$
$$\varphi_3 : ((\text{nation}, \text{country}), (\text{capital}, \perp, \text{capital}), (=, \not\approx, =)),$$

where "$\perp$" indicates that a field is missing and "$\not\approx$" that the two corresponding attributes are not comparable, e.g., when some attribute is missing from reference tables.

(1) Rule φ_1 states that for a tuple t in I_{EMP}, if its name matches the name of a r tuple in M_{PHN} and t[officePhn] matches r[mobile], then φ_1 validates that t[name] is correct and t[officePhn] is wrong. Moreover, it will rectify t[officePhn] to r[officePhn]. Consider t_3 in I_{EMP} and r_3 in M_{PHN}; φ_1 works as follows. Firstly, t_3[name] is matched with r_3[name] and t_3[officePhn] with r_3[mobile]. It then detects that t_3 is about Ian, but someone confused his office number with his mobile number. Consequently, t_3[name] is marked as correct and t_3[officePhn] as wrong. Since the office number of Ian is available in r_3[officePhn], φ_1 will update t_3[officePhn] to r_3[officePhn], which is 27364928.

(2) Often, not all evidence is available. Assume that the column officePhn is missing in PHN, and consider a revised schema PHN′ (name, mobile). Rule φ_2 states that given a tuple t in I_{EMP}, if its name matches the name of a tuple r in $M_{\text{PHN}'}$, and t[officePhn] matches r[mobile], then φ_2 validates that t[name] is correct and t[officePhn] is wrong. Again, consider t_3 in I_{EMP} and r_3 in $M_{\text{PHN}'}$; φ_2 works similarly to φ_1, which validates that t_3[name] is correct and t_3[officePhn] is wrong. However, due to the missing column officePhn in PHN′, φ_2 cannot update t_3[officePhn].

(3) Rule φ_3 states that for a tuple t in I_{EMP}, if t[country, capital] matches s[country, capital] of an s tuple in M_{CAP}, it will mark t[country, capital] as correct. Consider t_1 in I_{EMP} and s_1 in M_{CAP}. Since both country (i.e., China) and capital (i.e., Beijing) match, φ_3 will mark t_1[country, capital] as correct.

5.5 Conclusion

Data quality rules provide a declarative way to specify what is not allowed in correct data instances. Enforcing data quality rules can be done either through checking the validity of the data with respect to the defined rules upon data addition or update, or by detecting and repairing violations in the dirty data at various data analytics stages. Since designing data quality rules through consultation with domain experts is an expensive and time-consuming process, automatically mining data quality rules is an appealing alternative. To mine data quality rules, we need a formal language to capture the space of rules. ICs, while originally designed to

RIGHTSLINK

guide database schema normalization [Abiteboul et al. 1995], have been increasingly used to capture data inconsistencies because they provide formal languages to capture data quality rules. The more expressive the language is, the more rules we can potentially capture. However, with increasing expressiveness, it usually becomes harder to mine for rules due to a larger search space. In this chapter, we discussed commonly used integrity constraint languages, including FDs, CFDs, and DCs, and algorithms for discovering rules expressed in those languages.

We classify IC discovery algorithms into schema-driven approaches and data-driven approaches. Schema-driven approaches are sensitive to the size of the schema, namely, number of attributes, since they need to enumerate the space of all possible ICs based on the schema, which is usually exponential with respect to the size of the schema. Schema-driven approaches are only appealing if there exist efficient procedures to check the validities of candidate ICs. For instance, instead of checking the validity of each candidate FD by looking up the data I, the schema-driven algorithm TANE [Huhtala et al. 1999] partitions I by X to produce a set of nonempty disjoint subsets denoted as Π_X, where each subset contains identifiers of all tuples in I sharing the same value for attributes X. To check whether an FD $X \rightarrow A$ is valid, TANE simply needs to check $|\Pi_X| = |\Pi_{X \cup A}|$. Furthermore, Π_{XY} can be efficiently computed from two previously computed partitions, Π_X and Π_Y. However, to the best of our knowledge, no schema-driven algorithms exist for discovering DCs, mainly because it is unclear whether there exists an efficient way to check the validities of candidate DCs. Data-driven approaches are sensitive to the size of the instance and less sensitive to the size of the schema, since they avoid enumerating all ICs by building a data structure that captures all the information needed to discover ICs. FASTFD [Wyss et al. 2001] and FASTDC [Chu et al. 2013a] are examples of instance-driven algorithms. Hybrid IC discovery algorithms combine the best of both schema-driven approaches and instance-driven approaches by first discovering ICs on a sample dataset and then verify the discovered ICs using the entire dataset. HYFD [Papenbrock and Naumann 2016] for FD discovery and Hydra [Bleifuß et al. 2017] for DC discovery are examples of hybrid algorithms.

RIGHTSLINK

6 Rule-Based Data Cleaning

In this chapter, we discuss rule-based techniques to clean a dirty database I of schema R. We assume that a set of data quality rules Σ has been specified for R. This can be achieved either by asking domain experts to manually curate a set of rules, or by employing the automatic discovery algorithms discussed in Chapter 5. Since I is assumed to be dirty, the discovery algorithms should aim at discovering "approximate" ICs that hold on most parts of the data (cf. Section 5.3). As discovered ICs can overfit I and thus are not correct ICs for R in general, to verify the correctness of any discovered ICs by domain experts before using them for data cleaning is required.

Given a dirty database instance I of schema R and a set of ICs Σ, rule-based techniques clean I in two main steps: *violation detection*, where errors in I are detected by identifying violations in the dataset with respect to the specified data quality rules, and *error repair*, where the input data is updated to resolve the violations. In what follows, we discuss various challenges and design choices in performing these two steps.

6.1 Violation Detection

Detecting errors is an essential first step in any data cleaning activity. Errors found by using data quality rules are commonly referred to as *violations*. A commonly used definition of a violation with respect to data quality rules with no existential quantifiers is as follows: "A violation is a minimal set of cells that cannot coexist together." We can infer from this definition the following facts. First, not all the cells in a violation are actually wrong. However, at least one of them is wrong. Second, the set of cells that form the violation has to be minimal—removing any cells from the set would result in a smaller set of cells that can coexist together. Third, at least one of the cells from the set of cells that form a violation has to be changed to resolve the violation.

RIGHTSLINK

Table 6.1 **Employee data records** [Chu et al. 2013b]

TID	FN	LN	LVL	ZIP	ST	SAL
t_1	Anne	Nash	5	10001	NM	90K
t_2	Mark	White	6	87101	NM	80K
t_3	Mark	Lee	4	10001	NY	80K

Example 6.1 Consider Table 6.1: every tuple specifies an employee in a company with his or her identification (TID), name (FN, LN), role (ROLE), ZIP (ZIP), state (ST), and salary (SAL).

Consider two data quality rules. The first is a functional dependency (FD) stating that ZIP determines ST. We can see that a set e_1 of four cells $\{t_1[ZIP], t_1[ST], t_3[ZIP], t_3[ST]\}$ in t_1 and t_3 presents a violation for this FD: they have the same value for the city, but different states.

The second rule states that for two employees in the same state, the employee with a higher level cannot earn less than the other employee. In this case, a set e_2 of six cells $\{t_1[LVL], t_1[ST], t_1[SAL], t_2[LVL], t_2[ST], t_2[SAL]\}$ in t_1 and t_2 is violating the rule, since employee t_1, who is a level-5 employee, is earning more than t_2, who is a level-6 employee.

Given a dataset and a set of data quality rules, we need to detect all the violations with respect to the provided rules. Violation detection is a fairly straightforward procedure given the formalism of the data quality rules. For example, detecting violations in a dataset with respect to the functional dependency ZIP $\rightarrow$ ST essentially requires enumerating all pairs of tuples in the dataset and checking whether two tuples have the same ZIP and different ST for every tuple pair.

There are multiple challenges associated with violation detection: (1) a violation only states that a set of cells cannot co-exist together—some cells in that set might be correct and some might be wrong. Treating each violation individually in a piecemeal manner will not pinpoint exactly which cells are erroneous; (2) data usually needs to go through multiple stages of processing in order to derive value. While data errors originate in the source data, violations are usually discovered much later in the data processing pipeline, where more business logic becomes available. Therefore, violations detected in later stages need to be traced back to identify errors in the data source; and (3) detecting violation with respect to a data quality rule involving k tuples will require enumerating all k-tuple combinations from n input tuples, a polynomial cost that could become very high when n is large.

RIGHTSLINK

We discuss the literature of violation detection that addresses the three aforementioned challenges.

1. We introduce holistic data cleaning [Chu et al. 2013b], which compiles all violation detections, regardless of which data quality rules they violate, into one homogeneous representation, called the *conflict hypergraph*. Based on the conflict hypergraph, we can then identify which cells are more likely to be wrong.
2. We survey techniques that aim at propagating errors detected in transformation results to the data sources. Those techniques differ mainly because of the type of data transformations assumed, such as Boolean expressions [Meliou et al. 2011], aggregation on numerical attributes [Wu et al. 2012, Wu and Madden 2013], and general select-project-join-aggregate (SPJA) queries [Chalamalla et al. 2014].
3. We discuss the BigDansing [Khayyat et al. 2015] system, which detects violations in a distributed manner, leveraging a common shared-nothing parallel computing environment such as Hadoop or Spark.

6.1.1 Holistic Error Detection

Given a dirty dataset, a variety of data quality rules expressed in different IC languages could be used for detecting errors in that dataset. One naive way to use all the data quality rules would be to cascade them in a pipeline where different algorithms are used as black boxes to be executed sequentially or in an interleaved way. However, this piecemeal approach would compromise the quality in data cleaning.

Consider the two violations detected with respect to two different data quality rules in Example 6.1. We can see that violation e_1 consists of four cells, $\{t_1[ZIP], t_1[ST], t_3[ZIP], t_3[ST]\}$. However, we are not sure which cells out of the four are actually erroneous. Similarly, the violation e_2 consists of six cells, $\{t_1[LVL], t_1[ST], t_1[SAL], t_2[LVL], t_2[ST], t_2[SAL]\}$, and we are not sure which cells out of the six are actually erroneous.

Holistic data cleaning [Chu et al. 2013b] aims at pinpointing which cells are more likely to be wrong by compiling all violations detection regardless of which data quality rules they violate into one homogeneous representation, called the conflict hypergraph. Every cell in the original database corresponds to a vertex in the conflict hypergraph, and every violation detected corresponds to a hyperedge, consisting of a set of cells forming the violation. Figure 6.1 shows the two hyperedges corresponding to the two violations e_1 and e_2, as well as the conflict hypergraph by

RIGHTSLINK

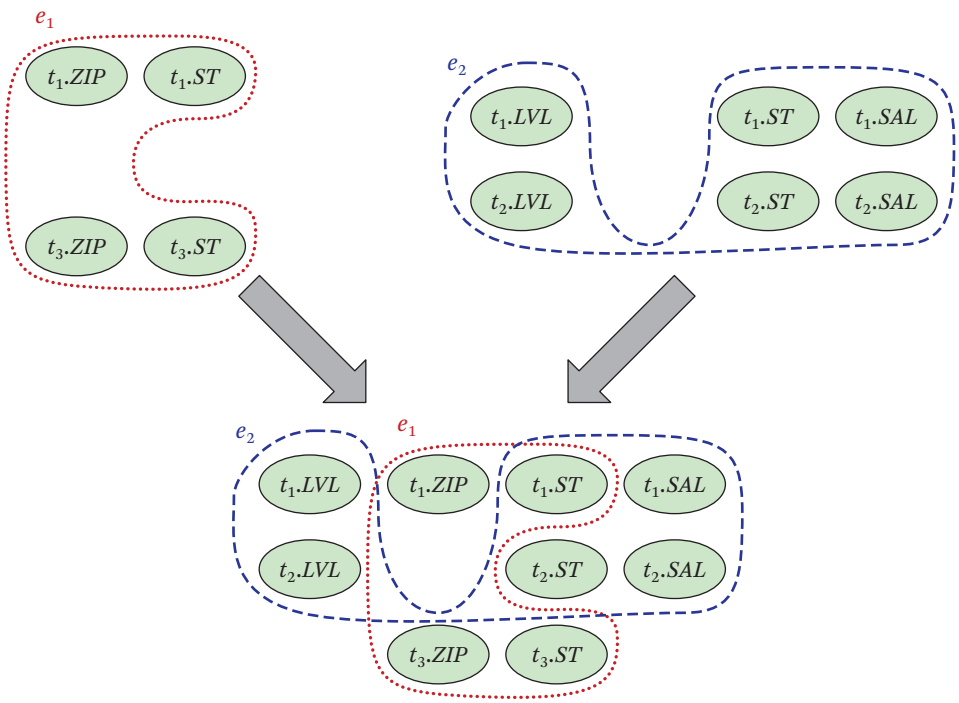


Figure 6.1 Holistic error detection.

putting the two hyperedges together. The conflict hypergraph contains all the violations and gives a holistic overview of how different violations interact with each other. The main intuition holistic data cleaning uses to identify erroneous cells is as follows: a cell involved in multiple violations is more likely to be wrong than another cell involved in fewer violations. In Figure 6.1, only cell $t_1[ST]$ is involved in both violations; therefore, $t_1[ST]$ is considered more likely to be the actual erroneous cell. In general, holistic data cleaning finds a minimal vertex cover of the conflict hypergraph and considers the cells in the minimal vertex cover as more likely to be wrong; those cells are passed on to the data repair step, as discussed in Section 6.2.

6.1.2 Error Propagation

The problem of error detection is further complicated by the fact that errors are usually discovered much later in the data processing pipeline, where more business

RIGHTSLINK

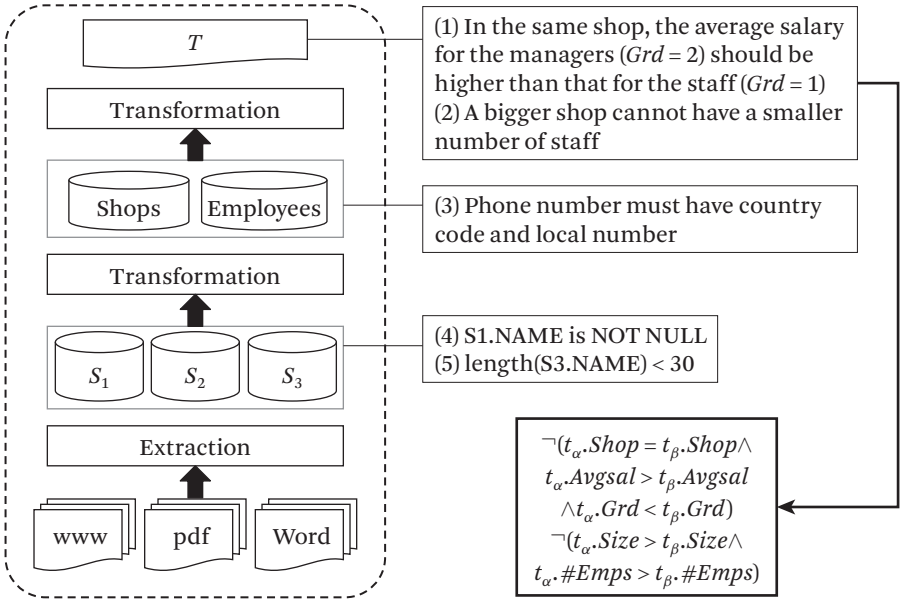


Figure 6.2 The ETL stack [Chalamalla et al. 2014].

logic becomes available. Consider a simple example of employee and department source tables. Detecting that the sum of employee salaries in a department exceeds the budget allocated for that department cannot be done before joining the two tables and aggregating the salaries of each group of employees.

In many applications, errors are detected in a target database (or a report) that is the result of data transformations applied on a source database. Figure 6.2 shows a typical data Extract-Transform-Load (ETL) processing stack. In each of the layers, various integrity constraints are defined as more semantics are added to the data. For example, while Constraint (4) (S1.NAME is NOT NULL) can be defined directly on the sources, Constraints (1) and (2) can only be defined at the application and reporting layer after the necessary aggregation and joins have been performed.

Propagating errors detected in transformation results to the data sources is essential for both repairing these errors and preventing them from recurring in the future. Techniques for error propagation vary according to the type of data transformations assumed, such as Boolean expressions [Meliou et al. 2011], aggregation on numerical attributes [Wu et al. 2012, Wu and Madden 2013], and more general SPJA queries [Chalamalla et al. 2014].

RIGHTSLINK

Causality Analysis

Causality analysis that tries to reason about the responsibility of a source in causing errors in query results is an intuitive way to handle the aforementioned problem. Work in this area [Meliou et al. 2011] models the source database as a set of variables $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ and the target database as another set of variables $\mathbf{Z} = \{Z_1, Z_2, \ldots, Z_m\}$. $\mathbf{X}$ can be seen as columns of an input database, and $\mathbf{Z}$ can be seen as columns of an output database. Each input variable X_i takes value from a discrete or continuous domain. Each output variable Z_j is a Boolean variable. The data transformation Φ_j for Z_j is a Boolean expression over threshold predicates of the form $X_i \; op \; c$, where $X_i \in \mathbf{X}$, $op \in \{<, \leq, =, \neq, \geq, >\}$, and c is a constant value in the domain of X_i. For example, a simple transformation Φ_1 is $Z_1 = (X_1 > 10) \wedge (X_2 < 3)$. Let $\mathbf{\Phi} = \{\Phi_1, \ldots, \Phi_m\}$ denote the m transformations for m output variables. See that $\mathbf{\Phi}$ takes input vector $\mathbf{x}$ of values for $\mathbf{X}$ and computes the output vector $\mathbf{z}$ of values for $\mathbf{Z}$. $\mathbf{x}$ can be seen as a tuple of the input database, and $\mathbf{z}$ can be seen as a tuple of the output database. Let $\hat{\mathbf{z}}$ be the ground truth values for $\mathbf{Z}$. Given $\mathbf{X}, \mathbf{Z}, \mathbf{\Phi}, \mathbf{x}, \mathbf{z}, \hat{\mathbf{z}}$, we would like to identify sources of errors by ranking the input variables $\mathbf{X}$ according to how much each variable contributes to the error in $\mathbf{z}$, also referred to as the *responsibility of* X_i.

Meliou et al. [2011] propose using a *view-conditioned counterfactual cause* to quantify the *responsibility of* X_i. As a first attempt, one might try to say that X_i is a view-conditioned counterfactual cause of $\mathbf{z}|\hat{\mathbf{z}}$ if a change in the value x_i "corrects" the output $\mathbf{z}$ to the ground truth $\hat{\mathbf{z}}$. The problem is that this requirement is hard to meet—a change in X_i alone can correct simultaneously all erroneous output values. Instead, a more relaxed definition is proposed: a *view-conditioned counterfactual cause* (VCC cause) of $\mathbf{z}|\hat{\mathbf{z}}$ is a minimal subset of input variables for which there exists a changed assignment that can change the output from $\mathbf{z}$ to $\hat{\mathbf{z}}$. Based on this more relaxed definition, the variable X_i is a *view-conditioned cause* (VC cause) of $\mathbf{z}|\hat{\mathbf{z}}$ if there exists a set $\Gamma \subset X$, called the *contingency set* of X_i, such that $X_i \cup \Gamma$ is a VCC cause of $\mathbf{z}|\hat{\mathbf{z}}$. The responsibility of X_i is then defined as $\rho_{X_i} = \frac{1}{1+\min_\Gamma |\Gamma|}$, where Γ is a contingency for X_i. Intuitively, the responsibility of X_i is the minimal number of input variables that need to be changed together with X_i to change the output from $\mathbf{z}$ to $\hat{\mathbf{z}}$.

Example 6.2 Consider $\mathbf{\Phi}$ given in Figure 6.3. Assume the ground truth is $\hat{\mathbf{z}} = \{F, T, T\}$, which indicates that Z_1 and Z_3 are errors in the output. $\{X_1, X_3\}$ is a VCC cause of $\mathbf{z}|\hat{\mathbf{z}}$, since changing X_1 from 5 to 11 and X_3 from 2 to 4 will change the output values from $\mathbf{z}$ to $\hat{\mathbf{z}}$ and changing only X_1 or X_3 will not flip both Z_1 and Z_3. The minimal sized contingency sets for X_1, X_2, and X_3 are $\{X_3\}$, $\{X_1, X_3\}$, and $\{X_1\}$, respectively.

RIGHTSLINK

$\boldsymbol{X}$	$\boldsymbol{x}$
X_1	5
X_2	T
X_3	2

(a)

$\boldsymbol{Z}$	$\boldsymbol{\Phi}$	$\boldsymbol{z}$
Z_1	$(X_1 < 10) \wedge X_2$	T
Z_2	$(X_3 > 0) \wedge X_2$	T
Z_3	$(X_3 > 3)$	F

(b)

Figure 6.3 (a) Value assignment **x** for **X**. (b) The transformations **Φ** and values **z** for **Z** [Meliou et al. 2011].

Therefore, the responsibilities of X_1, X_2, and X_3 are $\rho_{X_1} = \frac{1}{2}$, $\rho_{X_2} = \frac{1}{3}$, and $\rho_{X_3} = \frac{1}{2}$, which show that X_1 and X_3 contribute more than X_2 to the errors in the output variables.

Since computing the causality and responsibility are NP-hard [Meliou et al. 2011], the problem of computing causality is reduced to the SAT problem, and the problem of computing responsibility is reduced to a partial weighted MaxSAT problem. There exist several highly optimized tools to solve both SAT and weighted MaxSAT, which allow for efficient execution.

Scorpion

Another approach for computing the responsibility of sources in target errors is the Scorpion system [Wu and Madden 2013, Wu et al. 2012]. Scorpion assumes a single relational table as the source database and an SQL aggregate query as the data transformation. The target database is then a group of aggregate values, one for each group according to the aggregate query. The errors in the target database are those aggregate values that are considered outliers by users. Scorpion finds common properties of the set of tuples in the source database that cause the outliers in the target database. The common properties of a set of tuples are described by *predicates* over the attributes of the source database.

Scorpion uses *sensitivity analysis* to identify predicates that are most influential over the aggregate values. For example, given a function $y = f(x_1, \ldots, x_n)$, the influence of x_i is defined by the partial derivative, $\frac{\Delta y}{\Delta x_i}$. Similarly, the *influence* of a predicate p on one group α, denoted as $inf(p, \alpha)$, is defined as the ratio between the change in the output if the tuples satisfying the predicates are deleted from the input and the number of tuples satisfying the predicate. Thus, the influence of predicate p, denoted as $inf(p)$, is the average influence of p on all groups.

Example 6.3 Figure 6.4(a) shows some readings from an Intel sensor dataset, with each row corresponding to readings from a certain sensor at a given time. Figure 6.4(b) is

RIGHTSLINK

(a) Example reading from sensors

TIDs	Time	SensorID	Voltage	Humidity	Temperature
t_1	11AM	1	2.64	0.4	34
t_2	11AM	2	2.65	0.5	35
t_3	11AM	3	2.65	0.4	35
t_4	12AM	1	2.7	0.3	35
t_5	12AM	2	2.7	0.5	35
t_6	12AM	3	2.3	0.4	100
t_7	1PM	1	2.7	0.3	35
t_8	1PM	2	2.7	0.5	35
t_9	1PM	3	2.3	0.5	80

(b) Query results (left three columns) and user annotations (right column)

ResultIDs	Time	AVG(temp)	Label
α_1	11AM	34.6	Normal
α_2	12AM	56.6	Outlier
α_3	1PM	50	Outlier

Figure 6.4 Explaining errors in a sensor dataset [Wu and Madden 2013].

generated by the following aggregate query, which groups the readings by the hour and computes the mean temperature.

```
Q:  SELECT AVE(temp), time
    FROM sensors
    GROUP BY time
```

The rightmost column in Figure 6.4(b) represents user annotations. The user thinks two groups α_2 and α_3 have unusual results and group α_1 is a normal result.

Consider a predicate $p : Voltage < 2.4$. The average temperature of group α_2 after deleting tuples satisfying the predicate, that is, t_6 is 35. Thus, $inf(p, \alpha_2) = \frac{56.6-35}{1} = 21.6$. Similarly, $inf(p, \alpha_3) = \frac{50-35}{1} = 15$. Therefore, $inf(p) = 18.3$. Scorpion searches all possible predicates over attributes that are not involved in the query, e.g., Voltage and Humidity in this example, and returns the predicate with the largest influence.

To efficiently compute the influence of a predicate and to avoid testing the exponential number of all possible predicates, Scorpion identifies several proper-

RIGHTSLINK

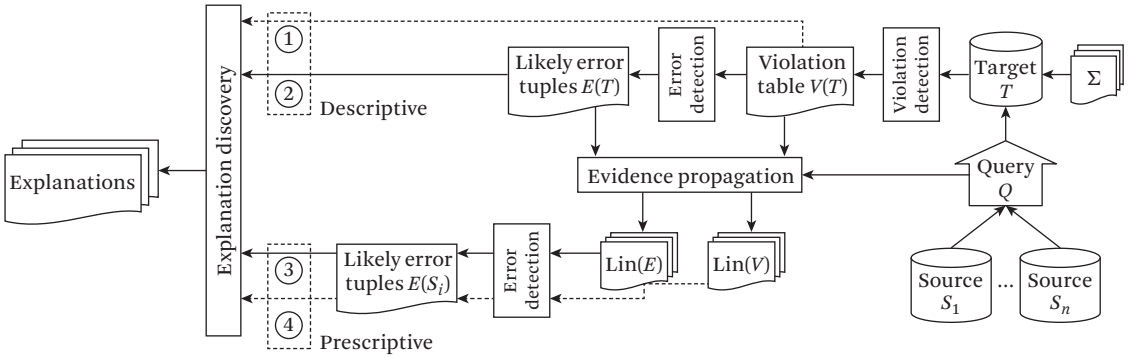


Figure 6.5 DBRx architecture [Chalamalla et al. 2014].

ties of aggregate operators, i.e., incrementally removable, independent, and anti-monotonic influence, that enables the algorithms to find the most influential predicate efficiently.

DBRx

To handle a more general class of transformations, the DBRx system [Chalamalla et al. 2014] considers SPJA (select, project, join, and aggregate) queries as the reporting and transformation language. Figure 6.5 shows the architecture of DBRx that takes quality rules defined over the output of a transformation and computes explanations of the errors. Given a transformation scenario (sources S_i, $1 < i < n$, and query Q) and a set of quality rules Σ, DBRx computes a violation table VT of tuples not complying with Σ. VT is mined to discover a descriptive explanation of the violations (1). The description explanation should cover the most likely erroneous tuples, while minimizing the clean tuples being covered. The lineage of the violation table over the sources enables the computation of a prescriptive explanation on the source tables (4). When applicable, a repair is computed over the target, thus allowing the possibility of a more precise description (2) and a more precise prescriptive explanation (3), based on propagating errors to the sources.

Example 6.4 Consider a target database T in Figure 6.6 which lists shops in an international franchise and information about employees working in those shops. T is generated by the following query.

```
Q:  SELECT SId as Shop, Size, Grd, AVG(Sal) as
    AvgSal, COUNT(EId) as #Emps,'US' as Region
    FROM US.Emps JOIN US.Shops ON SId
    GROUP BY SId, Size, Grd
```

RIGHTSLINK

(a) Target database T (Dirty)

T	Shop	Size	Grd	AvgSal	#Emps	Region
t_a	**NY1**	46 ft^2	**2**	**99** $	1	US
t_b	**NY1**	*46* ft^2	***1***	**100** $	*3*	US
t_c	NY2	62 ft^2	2	96 $	2	US
t_d	NY2	*62* ft^2	*1*	90 $	*2*	US
t_e	LA1	35 ft^2	2	105 $	2	US
t_f	LND	38 ft^2	1	65 £	2	EU

(b) Source Relation *Emps*

Emps	EId	Name	Dept	Sal	Grd	SId	JoinYr
t_1	e4	John	S	91	1	NY1	2012
t_2	e5	Anne	D	99	2	NY1	2012
t_3	e7	Mark	S	93	1	NY1	2012
t_4	e8	Claire	S	116	**1**	NY1	2012
t_5	e11	Ian	R	89	1	NY2	2012
t_6	e13	Laure	R	94	2	NY2	2012
t_7	e14	Mary	E	91	1	NY2	2012
t_8	e18	Bill	D	98	2	NY2	2012
t_9	e14	Mike	R	94	2	LA1	2011
t_{10}	e18	Claire	E	116	2	LA1	2011

(c) Source Relation *Shops*

Shops	SId	City	State	Size	Started
t_{11}	NY1	New York	NY	46	2011
t_{12}	NY2	New York	NY	62	2012
t_{13}	LA1	Los Angeles	CA	35	2011

Figure 6.6 A view T on data sources *Emps* & *Shops* [Chalamalla et al. 2014].

RIGHTSLINK

Explanation	Coverage	Preciseness	Conciseness
$Dept = s$?	No	Yes	Yes
$eid = e_4 \vee eid = e_7 \vee eid = e_8 \vee eid = e_{14}$?	Yes	Yes	No
$Grd = 1$?	Yes	No	Yes

Figure 6.7 Explanation discovery [Chalamalla et al. 2014].

The HR department has a set of policies (ICs) pertaining to the franchise workforce and these ICs are enforced on T. The first rule states that, in the same shop, the average salary of the managers (Grd=2) should be higher than that of the staff (Grd=1). Cells *Shop*, *Grd*, *AvgSal* of tuples t_a and t_b, labeled in bold, violate this rule. The second rule states that a bigger shop cannot have a smaller staff (Grd=1), which induces a violation between cells of t_b and t_d, labeled in italic.

Tuples t_a and t_b are in violation and their lineage is $\{t_1 - t_4\}$ and $\{t_{11}\}$ in sources *Emps* and *Shops*, respectively. Similarly, another violation between t_b and t_d has the combined lineage $\{t_5 - t_8\}$ and $\{t_{12}\}$. For each cell and tuple in the lineage, the cell contribution score (CSV) and tuple removal score (RSV) is computed. The CSV (resp. RSV) is a m-length vector to represent the contribution (resp. removal) scores of a cell (resp. tuple), where m is the number of violations in T. For instance, the RSV of t_4 is [1,1] since the removal of t_4 would resolve both violations.

Based on the CSV and RSV, DBRx identifies t_1, t_3, t_4, t_7 to be the most likely erroneous tuples. To explain these tuples, DBRx summarizes the tuples in terms of source attribute predicates with three desirable properties, namely, *coverage*, *preciseness*, and *conciseness*. Coverage requires an explanation to cover erroneous tuples; preciseness requires an explanation to cover mostly erroneous tuples, not correct tuples; and conciseness requires an explanation to have a small number of predicates. Figure 6.7 lists three different explanations. The first explanation lacks coverage, since t_7 is not covered; the second lacks conciseness, since it required four predicates to describe the four tuples; and the third lacks preciseness, since it also incorrectly covers the correct tuple t_5. These explanations can then be inspected by users, and similar errors can be prevented from happening.

6.1.3 Scalable Violation Detection

Violation detection can be expensive for large datasets as it often requires costly computations such as enumerating pairs of tuples, handling inequality joins, and

RIGHTSLINK

dealing with user-defined functions. BigDansing [Khayyat et al. 2015] is a distributed data cleaning system that runs on top of common data processing platforms, such as DBMS or MapReduce frameworks. BigDansing receives a data quality rule either in a user-defined function form or in a declarative form such as FDs or DCs. BigDansing architecture consists of three layers: logical, physical, and execution layers.

Logical layer. One major goal of BigDansing is to allow users to express a variety of data quality rules without having to worry about how to make data cleaning with respect to the rules scalable. To this end, BigDansing provides five logical operators, namely, Scope, Block, Iterate, Detect, and GenFix, to express a data quality rule. Scope operator defines the relevant data for the rule; Block operator defines the group of data units among which a violation may occur; Iterate operator enumerates all the candidate violations; Detect operator determines whether a candidate violation is indeed a violation; GenFix operator generates a set of possible fixes for each violation. GenFix operator is used to actually change the data so that new data is no longer in violation of the provided data quality rules, which is a topic discussed in Section 6.2. Users define these logical operators as well as the sequence in which BigDansing has to run them. Alternatively, users provide a declarative rule and BigDansing translates it into a job with these five operators automatically.

Example 6.5 We illustrate the five logical operators automatically generated when the user specifies an FD ZIP $\rightarrow$ ST in Table 6.1. The Scope operator generated would remove irrelevant data units from a dataset; in this case, it would project on attributes ZIP and ST. The Block operator generated automatically would partition the dataset to a disjoin set of tuples, with each partition having the same values for the ZIP attribute. The Iterate operator generated would iterate over all the partitions, and emit all the tuple pairs within each partition. The Detect operator would take every tuple pair emitted by the Iterate operator and check whether that tuple pair has different values for the ST attribute.

Physical layer. In this layer, BigDansing receives a logical plan and transforms it into an optimized plan of physical operators, which specifies how logic operators should be implemented. For example, Block operator could be implemented by either hash-based or range-based methods. BigDansing processes a logical plan through two main optimization steps. First, BigDansing consolidates redundant logical operators into a single logical operator. Hence, by applying the same logical operator several times on the same set of data units using shared scans, BigDansing is able to increase data locality and reduce I/O overhead. For example, if there

RIGHTSLINK

are two FDs $X \rightarrow Y$ and $X \rightarrow Z$, instead of scanning the input dataset twice to check for violations of these two FDs, BigDansing would only scan the dataset once, apply the Block operator once, and check for violations for both FDs in a single job. Second, BigDansing also provides specialized physical join operators that are commonly used in detecting violations with respect to data quality rules. For example, a specialized self-join operator is provided to perform tuple pairwise comparisons that involve greater than or less than.

Execution layer. In this layer, BigDansing determines how a physical plan will be actually executed on the underlying parallel data processing framework. It transforms a physical plan into an execution plan, which consists of a set of system-dependent operations, e.g., a Spark or MapReduce job. BigDansing runs the generated execution plan on the underlying system.

This three-layer architecture allows BigDansing to: (i) support a large variety of data quality rules by abstracting the rule specification process; (ii) achieve high efficiency when cleaning datasets by performing a number of physical optimizations; and (iii) scale to big datasets by fully leveraging the scalability of existing parallel data processing frameworks.

6.2 Error Repair

Given a relational database instance I of schema R which is in violation of a certain set of data quality rules, error repair refers to the process of finding another database instance I' that conforms to that set of data quality rules. Many error repair techniques have been proposed. Figure 6.8 depicts the classification we adopt to categorize the proposed error repair techniques. Below we discuss our classification dimensions and their impact on the design of underlying error repair techniques. The three adopted dimensions address the three main questions when repairing an erroneous databases.

Repair Target (What to Repair?). Business logic is not static; it often evolves over time. Previously correct integrity constraints may become obsolete quickly. Practical data repair techniques must consider possible errors in the data as well as possible errors in the specified constraints. Repairing algorithms make different assumptions about the data and the quality rules: (1) trusting the declared integrity constraints, and only allowing data to be updated to remove errors; (2) trusting the data completely and allowing the relaxation of the constraints, for example, to address schema evolution and obsolete business rules; and finally (3) exploring the possibility of changing

RIGHTSLINK

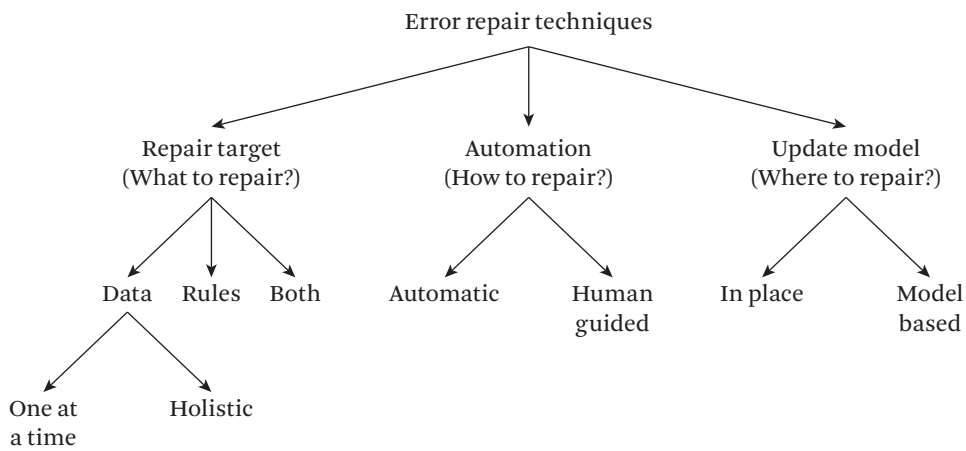


Figure 6.8 Classification of error repair techniques.

both the data and the constraints. For techniques that trust the rules and change only the data, they can be further divided according to the driver of the repairing exercise, that is, what types of errors they are targeting. A majority of techniques repair the data with respect to one type of error only (one at a time), while other emerging techniques consider the interactions among multiple types of errors and provide a holistic repair of the data (holistic).

Automation (How to Repair?). We classify proposed approaches with respect to the tools used in the repairing process. More specifically, we classify current repairing approaches according to whether and how humans are involved. Some techniques are fully automatic, for example, by modifying the database such that the distance between the original database I and the modified database I' is minimized according to some cost function. Other techniques involve humans in the repairing process to verify the fixes, to suggest fixes, or to train ML models to carry out automatic repairing decisions.

Repair Model (Where to Repair?). We classify proposed approaches based on whether they change the database in situ or build a model to describe the repair. Most proposed techniques repair the database in place, thus destructing the original database. For non-in-situ repairs, a model is often built to describe the different ways to repair the underlying database. Queries are answered against these repairing models using, for example, sampling from all possible repairs and other probabilistic query answering mechanisms.

RIGHTSLINK

Table 6.2 shows a sample of error repair techniques using the taxonomy. In the following, we discuss one-at-a-time data-only repairing techniques in Section 6.2.1, holistic data-only repairing techniques in Section 6.2.2, rule-only repairing techniques in Section 6.2.3, and both data and rule repairing techniques in Section 6.2.4. We introduce different cost functions adopted by automatic repairing techniques in Section 6.2.5, and we explore how humans can be involved to resolve ambiguities in the automatic repairing and to improve repairing accuracy in Section 6.2.6. Finally, we present model-based repairing techniques in Section 6.2.7 where original databases are not destroyed and a space of possible repairs is used to answer queries.

6.2.1 What to Repair: One-at-a-Time Data Repair

Data repair techniques in this category assume there is a set of ICs Σ defined on the database schema R, and any database instance I of R should conform to these constraints. Multiple proposed approaches aim at changing a minimum number of cells in the database such that a set of FDs are satisfied, either by changing the data directly [Bohannon et al. 2005, Kolahi and Lakshmanan 2009] or by generating samples from the space of all possible minimal repairs [Beskales et al. 2010]. We give multiple definitions of minimal repairs in Section 6.2.5 when we discuss automatic data repair techniques. Other approaches that generate minimal repairs have also been proposed to address other types of ICs, such as CFDs [Cong et al. 2007] and the more general denial constraints (DCs) [Chu et al. 2013b]. Approaches that avoid the minimal repair heuristic generate verified fixes, for example, by using reference master data and editing rules [Fan et al. 2010] or by heavily involving human experts [Yakout et al. 2011, Stonebraker et al. 2013]. In this section, we give one example data repair algorithm with respect to a given set of FDs.

Algorithm 6.1 gives the details of a data repair algorithm with respect to a given set of FDs [Bohannon et al. 2005] as an example. The details of other examples will be described in other sections when we discuss other dimensions and design options. Algorithm 6.1 takes as input a set of FDs and a dirty database instance I and produces a repaired database instance I'. At a high level, Algorithm 6.1 initially puts every database cell in its own equivalence class. An equivalence class is a set of cells that should have the same value. Then it greedily merges the equivalence classes in $\mathcal{E}$ until all FDs in Σ are satisfied. The unResolved sets keeps tracks of all tuples that participate in violations of FDs in Σ. At each step, a tuple t and an FD $X \rightarrow A$ are picked; the chosen FD has the lowest cost of merging all equivalence classes that should have the same value according to the FD. The termination of

RIGHTSLINK

Table 6.2 A sample of data repair techniques

	Repair Target—What				Automation—How		Repair Model—Where	
	Data—One at a Time	Data—Holistic	Rules	Both	Automatic	Human Involved	In Place	Model Based
FDs value modification [Bohannon et al. 2005]	✓				✓		✓	
FDs hypergraph [Kolahi and Lakshmanan 2009]	✓				✓		✓	
CFDs value modification [Cong et al. 2007]	✓				✓		✓	
Holistic data cleaning [Chu et al. 2013b]		✓			✓		✓	
LLUNATIC [Geerts et al. 2013]		✓			✓		✓	
Record matching and data repair [Fan et al. 2011b]		✓			✓		✓	
NADEEF [Dallachiesa et al. 2013]		✓			✓		✓	
Generate optimal tableaux [Golab et al. 2008]			✓		✓		✓	
Unified repair [Chiang and Miller 2011]				✓	✓		✓	
Relative trust [Beskales et al. 2013]				✓	✓		✓	
Continuous data cleaning [Volkovs et al. 2014]				✓	✓		✓	
GDR [Yakout et al. 2011]	✓					✓	✓	
KATARA [Chu et al. 2015]	✓					✓	✓	
Editing rules [Fan et al. 2010]	✓					✓	✓	
Sampling FDs repairs [Beskales et al. 2010]	✓				✓			✓
Sampling CFDs repairs [Beskales et al. 2014]	✓				✓			✓

RIGHTSLINK

Algorithm 6.1 **GenFDsRepair**

Input: Database instance I, a set of FDs Σ
Output: Another instance I' such that $I' \models \Sigma$
$\mathcal{E} \leftarrow \{t[A] : t \in I, A \in R\}$
Initialize unResolved sets for Σ
while unResolved is not empty **do**
  pick the next tuple t and next FD $X \rightarrow A$ to repair with the lowest repair cost
  resolve tuple t and update $\mathcal{E}$
  update unResolved sets affected by $\mathcal{E}$
end while
return I' obtained by picking a value that results in the lowest cost of $cost(eq)$
  for each $eq \in \mathcal{E}$

Algorithm 6.1 is based on the fact that the number of equivalence class merges is bounded by the total number of equivalence classes.

Similar to other data repair algorithms, the key difficulty addressed in Algorithm 6.1 is that repairing one constraint might introduce violations for other constraints. To capture and track the effect of changing one database cell on the possible values other cells can take, equivalence classes of database cells are used. An equivalence class eq is a set of database cells (e.g., $\{t_1[A], t_2[A], t_3[A]\}$ (where t_i is a tuple identifier and A is an attribute in R) that should have the same value. The algorithm maintains a global set of equivalence classes $\mathcal{E}$. For a given cell $t_i[A]$, let $eq(t_i[A])$ denote the current equivalence class containing $t_i[A]$ in $\mathcal{E}$. The cost of updating an equivalence class with respect to a value v, $cost(eq, v)$, is the cost of changing the values of all cells in eq to v. The cost of an equivalence class $cost(eq)$ is the minimal cost for all possible values v. The cost of merging a set E of equivalence classes is $mgcost(E) = cost(\cup_{eq \in E} eq) - \sum_{eq \in E} cost(eq)$.

Example 6.6 Consider an equivalence class $eq_1 = \{t_1[A], t_2[A], t_3[A]\}$, and assume $t_1[A], t_2[A], t_3[A]$ have values a_1, a_2, and a_2, respectively. Assuming that the cost of changing any cell is 1, we have $cost(eq_1, a_1) = 2$ and $cost(eq_1, a_2) = 1$, and thus $cost(eq_1) = 1$. Consider another equivalence class $eq_2 = \{t_4[A]\}$ with $t_4[A] = a_3$. The cost of merging eq_1 with eq_2 to form $eq_3 = \{t_1[A], t_2[A], t_3[A], t_4[A]\}$ is given by $mgcost(\{eq_1, eq_2\}) = cost(eq_3) - (cost(eq_1) + cost(eq_2)) = 2 - (1 + 0) = 1$.

The costs of updating and merging equivalence classes become the main building blocks in finding a minimal-cost repair while capturing the dependency among

RIGHTSLINK

cells when assigning a new value. Algorithm 6.1 was later extended to handle violations of CFDs based on the concept of equivalence classes [Cong et al. 2007].

6.2.2 What to Repair: Holistic Data Repair

Data-only repairing techniques make different assumptions about the driver of the repairing process. We discussed one technique in Section 6.2.1, addressing violations of FD constraints. Similarly, violations of CFD constraints have been addressed in multiple proposals, either automatically [Cong et al. 2007] or by involving humans in the loop [Yakout et al. 2011]. We describe these techniques as *one-at-a-time* techniques. Most available data repair solutions are in this category. They address one type of error, either to allow for theoretical quality guarantees or to allow for a scalable system.

However, data anomalies are rarely due to a single type of error; multiple data quality problems, such as missing values, typos, the presence of duplicate records, and business rule violations, are often observed in a single dataset. These heterogeneous types of errors interplay and conflict on the same dataset, and treating them independently would miss the opportunity to correctly identify the actual errors in the data. We call the proposals that take a more holistic view of the data cleaning process *holistic* cleaning approaches [Fan et al. 2011b, Chu et al. 2013b, Dallachiesa et al. 2013, Geerts et al. 2013, Fan et al. 2014b, Rekatsinas et al. 2017a].

Holistic repairing algorithms consider violations coming from different types of ICs simultaneously, while suggesting updates to repair the underlying data. For example, Chu et al. [2013b] consider a wide range of ICs, including FDs, CFDs, and DCs, as long as the violations of ICs can be encoded as a hyperedge in the conflict hypergraph. NADEEF [Dallachiesa et al. 2013], an open-source data cleaning system which provides an interface for the users to define their own data quality rules, also uses techniques from Chu et al. [2013b] to holistically resolve violations. LLUNATIC [Geerts et al. 2013] considers all constraints that can be expressed as equality-generating dependencies. Fan et al. [2011b] integrate data repair based on CFDs and record matching based on MDs, and show that these two tasks benefit from each other when coupled together. HoloClean [Rekatsinas et al. 2017a] uses probabilistic graphical models to accumulate all signals to reason about possible repairs.

As holistic cleaning approaches usually generate more accurate repairs than piecemean approaches, we describe multiple proposals on this topic, including the holistic data cleaning algorithm [Chu et al. 2013b], the LLUNATIC data cleaning framework [Geerts et al. 2013], and how to integrate CFDs with MDs [Fan et al. 2011b]. HoloClean [Rekatsinas et al. 2017a] can also be considered a holis-

RIGHTSLINK

Algorithm 6.2 **Holistic data cleaning**

Input: Database instance I, a set of ICs Σ
Output: Repaired database instance I'
build the conflict hypergraph for I w.r.t. Σ
loop
  find the minimum vertex cover for the conflict hypergraph
  using a recursive procedure to recursively collect repair requirements,
    starting from the minimum vertex cover:
  using a determination procedure for finding updates to satisfy the repair
    requirement according to a cost function:
  update the database I'
  build the conflict hypergraph again for I' w.r.t. Σ
until the conflict hypergraph is null

tic data cleaning approach that leverages multiple signals, which we describe in detail in Chapter 7 when we discuss how ML techniques can be used for data cleaning.

Holistic Data Cleaning

The holistic data cleaning algorithm is shown in Algorithm 6.2. The system takes as input a relational database I and a set of ICs Σ, which express the data quality rules that have to be enforced over the input database. It first projects the violations coming from different types of ICs into one homogeneous representation, i.e., a conflict hypergraph. Each node in the conflict hypergraph is a cell in the database; each edge is a set of cells participating in a violation of an IC. A minimum vertex cover for the conflict hypergraph is found. The minimum vertex cover contains the cells that are mostly likely to be wrong, i.e., those participating in multiple violations of different ICs. Anchoring on the cells in the minimum vertex cover, a set of repair requirements is collected, such that if they are satisfied all the violations will be resolved. The set of repair requirements is fed into a determination procedure, which computes a set of cell updates to the database such that all repair requirements are satisfied. Depending on the repair requirements, different determination procedures can be devised to suit different cost functions (cf. Section 6.2.5). For example, if there are $>$, $\geq$, $<$, $\leq$ operators in the repair requirements, a quadratic or linear programming solver may be used as the determination procedure. The database is then updated accordingly. The process is repeated until there are no violations of any ICs.

RIGHTSLINK

Example 6.7 Figure 6.1 shows the two hyperedges corresponding to the two violations e_1 and e_2 in Example 6.1, as well as the conflict hypergraph by putting the two hyperedges together. The conflict hypergraph contains all the violations and gives a holistic overview of how different violations interact with each other.

A minimum vertex cover of this graph is $\{t_1[ST]\}$. Anchoring on $t_1[ST]$, we collect repair requirements for fixing e_1 and e_2. To fix e_1 by changing $t_1[ST]$, $t_1[ST]$ has to be changed to not equal $t_3[ST]$. To fix e_2 by changing $t_1[ST]$, $t_1[ST]$ has to be changed to not equal $t_2[ST]$. Thus, the two repair requirements are: $t_1[ST] = t_3[ST]$ and $t_1[ST] \neq t_2[ST]$. Given these two repair requirements, and supposing the cost function is to change the minimum number of cells, a determination procedure is invoked to change the minimum number of cells out of these three cells, $t_1[ST]$, $t_2[ST]$, $t_3[ST]$, such that two requirements are satisfied. In this case, changing only one cell, i.e., updating $t_1[ST]$ from "NY" to "AZ," would satisfy the two requirements.

As stated before, different determination procedures can be used for different cost functions.

Example 6.8 Consider an instance of I of a relational schema $R(A, B, C)$, where I has only one tuple t_1 with $t_1[A] = 0$, $t_1[B] = 3$, and $t_1[C] = 2$. Suppose there are two repair requirements: $t_1[A] < t_1[B]$ and $t_1[B] < t_1[C]$.

If one would like to minimize the squared distance between a repaired instance I' and I, a quadratic programming problem can be used with the objective function $(t_1[A] - 0)^2 + (t_1[B] - 3)^2 + (t_1[C] - 2)^2$ and the two repair requirements as two constraints. The optimal solution to the quadratic is $t_1[A] = 1$, $t_1[B] = 2$, and $t_1[C] = 3$.

However, if one would like to minimize the number of changed cells, only one cell needs to be changed (change $t_1[B]$ to 1) to satisfy the two repair requirements.

LLUNATIC

LLUNATIC [Geerts et al. 2013] is a data cleaning framework that considers different kinds of integrity constraints, as well as different strategies, to repair conflicting values. In particular, it specifies constraints based on equality-generating dependencies, which can express FDs, CFDs, MDs, and eRs. To specify different repair strategies it introduces the notion of *cell group*, which is a set of cells that should take on the same value along with the lineage of the value to take, e.g., coming from a master relation. A *cell group* is similar to the notion of an *equivalence class*, as discussed in Section 6.2.1. A partial order is introduced to cell groups; the partial order specifies the typical strategies to select a value for a cell group, including master data, certainty, accuracy, freshness, and currency, as well as user specified preferences. A parallel chase engine is developed to compute the repair. The chase

RIGHTSLINK

procedure includes a cost manager, which decides which repair to retain or discard based on different repairing objectives, e.g., cost minimal or cardinality minimal. Example 6.9 shows several cases where one value is preferred to another in resolving a violation. LLUNATIC has been extended to include user input into the chase procedure by allowing users to resolve conflicting values for which there is no clear preference, or by allowing users to discard unwanted repairs [Geerts et al. 2014a, Geerts et al. 2014b].

Example 6.9 Consider the database shown in Figure 6.9, containing customer data (CUSTOMERS) with addresses and credit card numbers of customers, and medical treatments paid by insurance plans (TREATMENTS). The following constraints are defined on the database: an FD φ_1 : SSN, NAME $\rightarrow$ PHONE defined on CUSTOMERS, an FD φ_2 : SSN, NAME $\rightarrow$ CC# defined on CUSTOMERS, and an FD φ_3 : SSN $\rightarrow$ SALARY defined on TREATMENTS.

Tuples t_2 and t_3 are violate φ_1 and one may want to equate t_2[PHONE] and t_3[PHONE] to fix the violation. However, φ_1 does not tell which value ("122-1876" or "000-0000") t_2[PHONE] and t_3[PHONE] should take. If the PHONE attribute of CUSTOMERS comes with a confidence CONF, shown in CUSTOMERS in Figure 6.9, and the value with higher confidence is preferred, the violation is repaired by changing t_3[PHONE] to "122-1876".

Tuples t_4 and t_5 are violate φ_3 and if the more recent value for SALARY attribute of a person is preferred, the violation can be repaired by changing t_4[SALARY] to the

	Customers						
	SSN	Name	Phone	Conf	Str	City	CC#
t_1	111	M. White	408-3334	0.8	Red ave.	NY	112321
t_2	222	L. Lemon	122-1876	0.9	NULL	SF	781658
t_3	222	L. Lemon	000-0000	0.0	Fry Dr.	SF	784659

	Treatments				
	SSN	Salary	Insurance	Treat	Date
t_4	111	10K	Abx	Dental	10/1/2-11
t_5	111	25K	Abx	Cholest.	8/12/2012
t_6	222	30K	Med	Eye Surg.	6/10/12

Figure 6.9 Customers and treatments [Geerts et al. 2013].

RIGHTSLINK

value of t_5[SALARY], which is more recent than the value of t_4[SALARY], according to Attribute DATE in TREATMENTS.

It is not always clear how to choose preferred values. For example, when repairing t_2[CC#] and t_3[CC#] for φ_2, there is no information available to resolve the conflict. The best one can do is to mark the conflict and then, perhaps, ask for user interaction to solve it.

Repairing CFDs and MDs

Fan et al. [2011b] integrate data repair based on CFDs and record matching based on MDs and show that these two tasks benefit when they are considered together. For a relation I of schema R, a master relation I_m of schema R_m, a set Σ of CFDs defined on R, and a set Γ of MDs defined on R and R_m, a repair I' of I is another instance of R such that (1) I' satisfies Σ; (2) I' and I_m satisfy Γ, and (3) $cost(I, I')$ (cf. Section 6.2.5) is minimal. Example 6.10 gives a scenario where these two tasks interplay and benefit from each other.

Example 6.10 Consider two relational tables card and tran defined in Example 5.9, that is: card(FN, LN, St, city, AC, zip, tel, dob, gd), and tran(FN, LN, St, city, AC, post, phn, gd, item, when, where). Table 6.3 shows an instance D_m of card and an instance D of tran. The following constraints are defined on tran and card: a CFD φ_1 tran([AC = 020] $\rightarrow$ [city = Ldn]), an FD φ_2 tran([city, phn] $\rightarrow$ [St, AC, post]), a CFD φ_3 tran([FN = Bob] $\rightarrow$ [FN = Robert]), and an MD ψ tran[LN, city, St, post] = card[LN, city, St, zip] $\wedge$ tran[FN] $\approx$ card[FN] $\rightarrow$ tran[FN, phn] $\rightleftharpoons$ card[FN, tel].

Consider Tuples t_3 and t_4 in D. The bank suspects that the two records refer to the same person. If so, then these transaction records show that the same person made purchases in the U.K. and in the U.S. at about the same time (taking into account the five-hour time difference between the two countries). This indicates that a fraud has likely been committed.

Tuples t_3 and t_4 are quite different in their FN, city, St, post, and Phn attributes. No rules can identify the two directly. Nonetheless, they can indeed be matched by a sequence of *interleaved* matching and repairing operations: (a) get a repair t'_3 of t_3 such that t'_3[city] = Ldn via CFD φ_1, and t'_3[FN] = Robert by normalization with φ_3; (b) match t'_3 with s_2 of D_m, to which ψ can be applied; (c) as a result of the matching operation, get a repair t''_3 of t_3 by correcting t''_3[phn] with the master data s_2[tel]; and (d) find a repair t'_4 of t_4 via the FD φ_2: since t''_3 and t_4 agree on their city and phn attributes, φ_2 can be applied to enrich t_4[St] and fix t_4[post] by taking corresponding values from t''_3, which have been confirmed correct with the master data in Step (c).

RIGHTSLINK

Table 6.3 Example credit card and transaction records [Fan et al. 2011b].

(a) Master data D_m: An instance of schema `card`

	FN	LN	St	City	AC	Zip	Tel	Dob	Gd
s_1 :	Mark	Smith	10 Oak St	Edi	131	EH8 9LE	3256778	10/10/1987	Male
s_2 :	Robert	Brady	5 Wren St	Ldn	020	WC1H 9SE	3887644	12/08/1975	Male

(b) Database D: An instance of schema `tran`

	FN	LN	St	City	AC	Post	Phn	Gd	Item	When	Where
t_1:	M.	Smith	10 Oak St	Ldn	131	EH8 9LE	9999999	Male	watch, 350 GBD	11am 28/08/2010	UK
cf	(0.9)	(1.0)	(0.9)	(0.5)	(0.9) (0.9)	(0.0)	(0.8)	(1.0)	(1.0)	(1.0)	
t_2:	Max	Smith	PO Box 25	Edi	131	EH8 9AB	3256778	Male	DVD 800 INR	8pm 28/09/2010	India
cf	(0.7)	(1.0)	(0.5)	(0.9) (0.7)	(0.6)	(0.8)	(0.8)	(1.0)	(1.0)	(1.0)	
t_3:	Bob	Brady	5 Wren St	Edi	020	WC1H 9SE	3887834	Male	iPhone 599 599 GBP	6pm 06/11/2009	UK
cf	(0.6)	(1.0)	(0.9)	(0.2) (0.9)	(0.8)	(0.9)	(0.8)	(1.0)	(1.0)	(1.0)	
t_3:	Robert	Brady	null	Ldn	020	WC1E 7HX	3887644	Male	necklace 2,100 USD	1pm 06/11/2009	USA
cf	(0.7)	(1.0)	(0.0)	(0.5) (0.7)	(0.3)	(0.7)	(0.8)	(1.0)	(1.0)	(1.0)	

RIGHTSLINK

At this point, t_3'' and t_4' agree on every attribute. It is now evident enough that they indeed refer to the same person; hence, a fraud. Observe that not only repairing helps matching, for example, from Step (a) to (b), but matching also helps to repair the data; for example, Step (d) can be done only after the matching in (b).

6.2.3 What to Repair: Rules-Only Repair

The techniques in this category assume data is clean and ICs need to be changed such that data conforms to the changed ICs. A particular example is the pattern tableau problem discovery for CFDs (cf. Section 5.2), where an embedded FD is given [Golab et al. 2008]. Recall that a CFD $(R : X \rightarrow Y, T_p)$ consists of an embedded FD $X \rightarrow Y$ and a pattern tableau T_p, where for every attribute $A \in X \cup Y$ and each pattern tuple $t_p \in T_p$, either $t_p[A]$ is a constant in the domain $Dom(A)$ of A, or $t_p[A]$ is a wild card "-".

Given a clean database instance I of schema R and an embedded FD for a CFD, the pattern tableau discovery problem aims at discovering a "good" pattern tableau for the CFD. A good pattern tableau should maximize the number tuples matching the pattern tuples in the tableau while minimizing the number of violations. Furthermore, a tableau should be concise to capture the semantics of the data in a readable way. For example, if every tuple in the database is a pattern tuple in the pattern tableau, then the pattern tableau would match all the database tuples; however, the large tableau size makes it almost unusable in practice.

The cover of a pattern tuple $t_p \in T_p$ is defined as all the tuples in I that match t_p, i.e., $Cover(t_p) = \{t | t \in I \text{ and } t \approx t_p\}$. The local support of t_p is thus defined as $LS(t_p) = \frac{|Cover(t_p)|}{|I|}$. Let $Keepers(t_p)$ be the subset of tuples in $Covers(t_p)$ after removing the fewest tuples to remove all violations of t_p. The local confidence of t_p is thus defined as $LC(t_p) = \frac{|Keepers(t_p)|}{|Cover(t_p)|}$. The global support of the pattern tableau T_p is defined as the fraction of tuples in I matching any pattern tuple in T_p, i.e., $GS(T_p) = \frac{|\cup_{t_p \in T_p} Covers(t_p)|}{|I|}$. The global confidence of the pattern tableau T_p is defined as $GC(T_p) = \frac{|\cup_{t_p \in T_p} Keepers(t_p)|}{|\cup_{t_p \in T_p} Covers(t_p)|}$.

Example 6.11 Consider the CFD $(\{\text{name, type, country}\} \rightarrow \{\text{price, tax}\}, T_p)$ in Figure 5.4 for Table 5.2. Consider the third pattern tuple $(-, -, UK | -, -)$ in Figure 5.4; it covers seven tuples t_8, t_9, t_{10}, t_{11}, t_{15}, t_{16}, and t_{17}. Those seven tuples violate this pattern tuple; removing either t_{16} or t_{17} will resolve the violation. Hence, the local support of this pattern tuple is $\frac{7}{20}$ and its local confidence is $\frac{6}{7}$. Similarly, the local support of $(-, \text{clothing}, - | -, -)$ is $\frac{7}{20}$ and its local confidence is 1. The local support of $(-, \text{book, France} | -, 0)$ is $\frac{5}{20}$ and its local confidence is $\frac{4}{5}$ (t_3 causes the violation).

RIGHTSLINK

Algorithm 6.3 **Tableau generation with GS and LC**

Input: Database instance I of schema R, two thresholds (s, c),
and an embedded FD $X \rightarrow Y$
Output: Pattern tableau T_p
Generate all possible pattern tuples from active domain of R,
and compute their local support and local confidence
remove pattern tuples whose local confidence is below c
iteratively choose the pattern tuple t_p with the highest marginal support,
add t_p to T_p
stop when the global support of T_p is greater than s
return T_p

The global support of all three pattern tuples is $\frac{15}{20}$ with a global confidence of $\frac{13}{15}$.

Given the definition of GS, GC, LS, LC, two versions of the pattern tableau generation problem are defined: the first version is called *pattern tableau generation with GS and GC*, i.e., given an embedded FD $X \rightarrow Y$ on R, an instance I of R, and two thresholds (s, c), find the T_p of the smallest size such that $GS(T_p) \geq s$ and $GC(T_p) \geq c$. Not only is the problem NP-complete, but it is also provably hard to approximate with $|I|^{0.5-\epsilon}$, $\epsilon > 0$ [Golab et al. 2008]. The second version is called *pattern tableau generation with GS and LC*, i.e., given an embedded FD $X \rightarrow Y$ on R, an instance I of R, and two thresholds (s, c), find the T_p of the smallest size such that $GS(T_p) \geq s$ and $LC(t_p) \geq c$, $\forall t_p \in T_p$. The problem is reduced to a variant of the partial set cover problem. Algorithm 6.3 gives a greedy approach to the problem.

Algorithm 6.3 computes the support and confidence of every possible candidate pattern tuple and then iteratively chooses pattern tuples with the highest marginal support (and those which are above the confidence threshold), adjusting the marginal supports for the remaining candidate patterns after each selection, until the global support threshold is met or until all candidate patterns are exhausted.

6.2.4 What to Repair: Both Data and Rule Repair

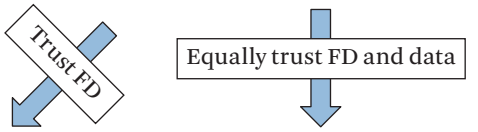
Cleaning techniques in this category assume data and rules can be dirty simultaneously [Beskales et al. 2013, Chiang and Miller 2011, Volkovs et al. 2014]. Given a database instance I and a set of FDs Σ such that $I \not\models \Sigma$, we need to find another I' and Σ' such that $I' \models \Sigma'$.

Example 6.12 Figure 6.10 shows a table with an FD stating that given name and surname determine income. There are three violations of the FD, i.e., the first and the second

RIGHTSLINK

GivenName	Surname	BirthDate	Gender	Phone	Income
Danielle	Blake	9 Dec 1970	Female	817-213-1211	120k
Danielle	Blake	9 Dec 1970	Female	817-988-9211	100k
Hong	Li	27 Oct 1972	Female	591-977-1244	90k
Hong	Li	27 Oct 1972	Female	498-214-5822	84k
Ning	Wu	3 Nov 1982	Male	313-134-9241	90k
Ning	Wu	3 Nov 1982	Male	323-456-3452	95k

Surname, GivenName → Income



GivenName	...	Income
Danielle	...	120k
Danielle	...	120k
Hong	...	90k
Hong	...	90k
Ning	...	95k
Ning	...	95k

Surname, GivenName → Income

GivenName	...	Income
Danielle	...	120k
Danielle	...	120k
Hong	...	90k
Hong	...	84k
Ning	...	90k
Ning	...	95k

Surname, GivenName, BirthDate → Income

GivenName	...	Income
Danielle	...	120k
Danielle	...	100k
Hong	...	90k
Hong	...	84k
Ning	...	90k
Ning	...	95k

Surname, GivenName, BirthDate, Phone → Income

Figure 6.10 Relative trust of FDs and data.

tuple, the third and the fourth tuple, and the fifth and the sixth tuple. If the FD is completely trusted, three cell changes are required, shown in the bottom left table in Figure 6.10. If the data is completely trusted, two attributes are added to the LHS of the FD, shown in the bottom middle table in Figure 6.10. If the FD and data have equal trustworthiness, a repair is to only change one cell value and add one attribute to the LHS of the FD, shown in the bottom right table in Figure 6.10.

In what follows, we discuss three approaches for repairing FDs and data, relying on the notion of relative trust [Beskales et al. 2013], unified cost [Chiang and Miller 2011], and continuous cleaning [Volkovs et al. 2014].

Relative Trust Between Data and Constraints

Beskales et al. [2013] model the universe of all possible repairs of (I, Σ) to include all pairs of (I', Σ'), such that $I' \models \Sigma'$. Let $\Delta(I, I')$ denote the distance between I and I' (e.g., the number of different cells between I and I'). Let $\Delta(\Sigma, \Sigma')$ denote the distance between Σ and Σ' (e.g., the number of attributes added to the LHS of the

RIGHTSLINK

Algorithm 6.4 **Relative trust of FDs and data**

Input: Database instance I, a set of FDs Σ, a threshold τ
Output: Another instance I' and Σ', such that $I' \models \Sigma'$
obtain Σ' such that $\delta_{opt}(\Sigma', I) \leq \tau$, and no other Σ'' with $\delta_{opt}(\Sigma'', I) \leq \tau$ has
  $\Delta(\Sigma, \Sigma') < \Delta(\Sigma, \Sigma'')$, where $\delta_{opt}(\Sigma', I)$ is the minimum number of cells that need to be changed in I for I to satisfy Σ'
if $\Sigma' \neq \emptyset$ **then**
  obtain I' that satisfies Σ' with at most $\delta_{opt}(\Sigma', I)$ changes
  return (I', Σ')
else
  no repair
end if

FDs in Σ). A repair (I', Σ') is said to be *minimal* if there does not exist another repair (I'', Σ''), such that $\Delta(I, I'') \leq \Delta(I, I')$ and $\Delta(\Sigma, \Sigma'') < \Delta(\Sigma, \Sigma')$, or $\Delta(I, I'') < \Delta(I, I')$ and $\Delta(\Sigma, \Sigma'') \leq \Delta(\Sigma, \Sigma')$. In other words, a repair (I', Σ') is *minimal* if and only if no other repair (I'', Σ'') dominates it in terms of the two distances. Minimal repairs cover a wide spectrum of repairs, ranging from completely trusting I and only changing Σ to completely trusting Σ and only changing I. The relative trust on I is defined as the maximum number of allowed cell changes τ. A *τ-constrained repair* is a repair that has the minimum distance to Σ across all repairs with distance to I less than or equal to τ.

Algorithm 6.4 describes the procedure for computing a repair (I', Σ') given a relative trust level τ on I. It consists of two major steps: (1) find the closest Σ' to Σ, such that there exists a repair for I that at most changes τ cells; and (2) obtain the actual repair I' given Σ. The space of possible repairs is modeled as a state space, where each state represents extending the LHS of the FDs in the original Σ. Figure 6.11 depicts an example search space for $\Sigma = \{A \rightarrow B, C \rightarrow D\}$. The root node represents adding nothing to the LHS of the two FDs, while the child node (C, ϕ) represents adding C to the LHS of the first FD and not adding anything to the LHS of the second FD, representing the new $\Sigma' = \{A \rightarrow BC, C \rightarrow D\}$.

To perform Step (1), searching the space for optimal Σ', the algorithm effectively navigates the space of all possible repairs of Σ, while computing $\delta_{opt}(\Sigma', I)$, i.e., the minimum number of cells that need to be changed in I for I to satisfy Σ', without actually performing the cleaning. Since computing $\delta_{opt}(\Sigma', I)$ is an NP-hard problem [Bohannon et al. 2005], a tuple-based *conflict hypergraph* is used to approximate $\delta_{opt}(\Sigma', I)$ by a factor of $2 \times \min\{|R| - 1, |\Sigma|\}$. An $A^\star$ based algorithm is

RIGHTSLINK

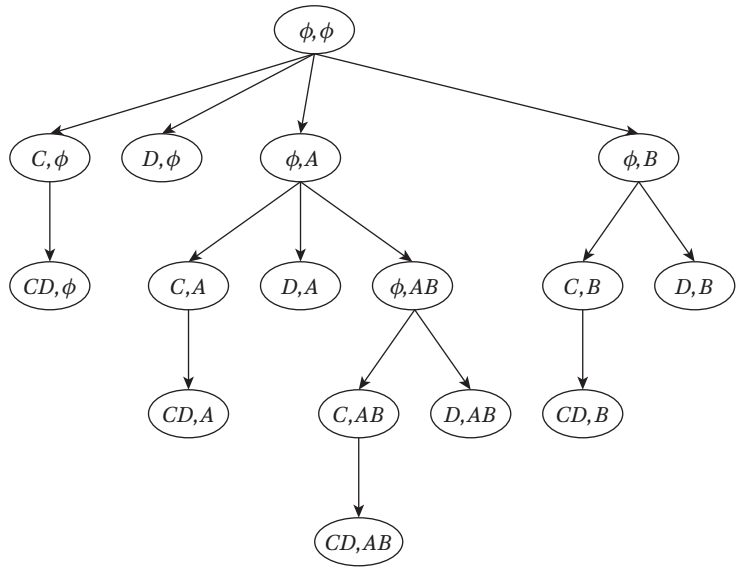


Figure 6.11 A space for $R = \{A, B, C, D\}$ and $\Sigma = \{A \rightarrow B, C \rightarrow D\}$.

used to navigate the space of all possible repairs of Σ. Step (2), the actual repairing of I with respect to Σ' found in Step (1), is performed using any automatic FD violation repairing algorithm, such as those described in Section 6.2.1.

Unified Cost of Changing Data and Constraints

In contrast to the aforementioned approach [Beskales et al. 2013], which treats the cost of repairing constraints and data separately, Chiang and Miller [2011] propose a unified cost model for repairing data and FDs on an equal footing based on the minimum description length (MDL) principle. Based on the cost model, Chiang and Miller [2011] associate a cost for a database instance I and a set of FDs Σ. Given a database instance I and Σ such that $I \not\models \Sigma$, the goal is to find another database instance I' and Σ' such that $I' \models \Sigma'$ and the cost associated with I' and Σ' is minimized.

We describe how the model is built and used if there a single FD in Σ. Assume an FD $\varphi : X \rightarrow Y$ defined over relational schema R and an instance I of R; a model M is built for φ. The model M consists of a set of *signatures*, where each signature s is a single tuple in I projected on XY, i.e., $s \in \Pi_{XY}(I)$. M uses a unit cost for each cell in a relation. The description length DL for M is defined as the length of the model $L(M)$ plus the length to encode the data values in the relation I, given the model

RIGHTSLINK

$L(I|M)$. See that $L(M)$ is calculated as $L(M) = |XY| \times S$, where $|XY|$ is the number of attributes in XY and S is the number of signatures in M, and $L(I|M)$ is calculated as $L(I|M) = |XY| \times E$, where E is the number of tuples in I whose projection on XY is not represented by any signature s in M. If the model is empty, namely, $L(M) = 0$, then $DL = L(I|M) = |XY| \times |I|$, where $|I|$ is the number of tuples in I. As more signatures are added to M that do not conflict with existing ones, $L(M)$ increases while $L(I|M)$ decreases. The goal is to find an M, such that $DL = L(M) + L(I|M)$ is minimized. Given an FD φ and an instance I of R, an initial model M is built by adding those signatures into M whose support is more than a predefined threshold, where the support of a signature s is the number of tuples having s as their values for XY attributes. To resolve the violations of φ, either the data repair or the constraint repair is chosen depending on which repair results in a larger reduction in DL.

If there are multiple FDs in Σ, they are processed in an order depending on (1) the number of violations of an FD, and (2) the potential conflict an FD shares with other FDs defined based on the number of overlapping attributes.

Continuous Data Cleaning

Rather than repairing data and constraints in a single snapshot of the database, continuous data cleaning [Volkovs et al. 2014] considers repairing both FDs and the data in a dynamic environment, where data may change frequently and constraints may evolve. The continuous data cleaning framework is shown in Figure 6.12.

In the first stage, a probabilistic classifier that predicts the repair type (data, FDs, or a hybrid of both) is trained using repairs that have been selected and validated by a user (Figure 6.12(a)). These repairs provide a baseline to the classifier representing the types of modifications that align with the user and the application preferences. As the data and the constraints change, inconsistencies may arise that need to be resolved. Once the classifier is trained, it predicts the types of repairs needed to address the violations (Figure 6.12(b)). A set of statistics is calculated to describe the properties of the violations, such as the number of violating tuples and the number of violating FDs. The classifier generates predications and computes the probability of each repair type (data, FD, or a hybrid of both). These repair predictions are passed to the repair search algorithm, which narrows the search space of repairs based on the classifier's recommendation (Figure 6.12(c)). The repair search algorithm includes a cost model that determines which repairs are best to resolve the inconsistencies. The repair search algorithm recommends a set of data and/or FD repairs to the user, who will decide which repairs to apply (Figure 6.12(d)). The applied repairs are then used to re-train the classifier and the process is repeated. Incremental changes to the data and to the constraints

RIGHTSLINK

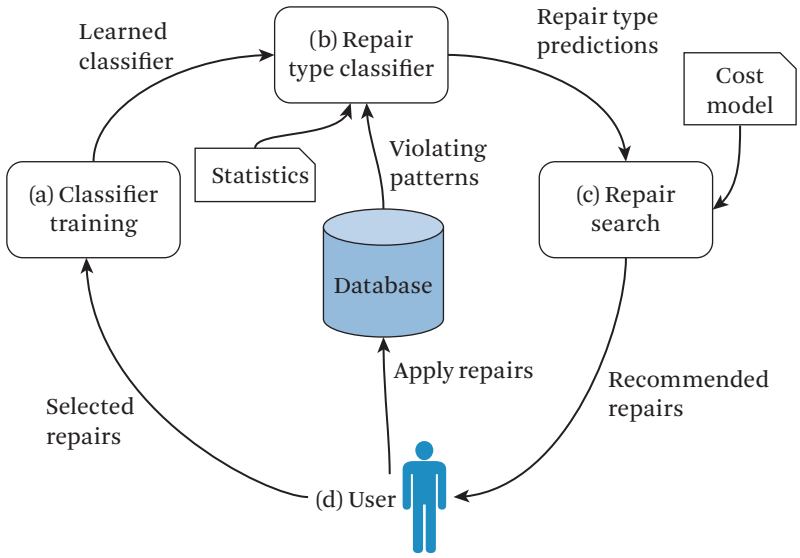


Figure 6.12 Continuous data cleaning [Volkovs et al. 2014].

are passed to the classifier (Figure 6.12(b)) and reflected via the statistics and the patterns.

6.2.5 How to Repair: Automatic Repair

In this section, we focus on automatic date repairing techniques, where humans are not involved. There exist multiple theoretical studies [Bohannon et al. 2005, Chomicki and Marcinkowski 2005, Afrati and Kolaitis 2009] and surveys [Bertossi 2011, Fan and Geerts 2012] studying the complexity of data repairing parameterized by different classes of ICs, such as FDs, CFDs, and DCs, and different repairing operations, such as value updating and tuple deleting. We discuss data repairing techniques that aim at updating the database in a way such that the distance between the original database I and the modified database I' is minimized. With a lack of ground truth, the main hypothesis behind the minimality objective function is that a majority of the database is clean and, thus, only a relatively small number of updates need to be performed compared to the database size.

Let $\Delta(I, I')$ denote the set of cells that have different values in I and I', i.e., $\Delta(I, I') = \{C \in CIDs(I) : I(C) \neq I'(C)\}$.

Definition 6.1 Cardinality-Minimal Repair. A repair I' of I is cardinality-minimal if and only if there is no repair I'' such that $|\Delta(I, I'')| < |\Delta(I, I')|$.

RIGHTSLINK

A repair I' of I is cardinality-minimal if and only if the number of changed cells in I' is minimum among all possible repairs of I. The automatic repairing algorithm in Kolahi and Lakshmanan [2009] aims to find cardinality-minimal repairs for FD violations.

A weighted version of the cardinality-minimal repair associates a weight with each cell, reflecting the confidence in the correctness of the cell [Bohannon et al. 2005, Cong et al. 2007, Chu et al. 2013b]. In addition, the distance between cell $I(C)$ and $I'(C)$ is measured using a distance function $dis(I(C), I'(C))$ instead of binary 0 or 1 in cardinality-minimal repair. The cost of a repair I' of I is defined as $cost(I, I') = \sum_{C \in \Delta(I,I')} dis(I(C), I'(C))$.

Definition 6.2 Cost-Minimal Repair. A repair I' of I is cost-minimal if and only if there is no repair I'' such that $cost(I, I'') < cost(I, I')$.

In Section 6.2.1, we gave the details of an example cost-minimal repairing algorithm (Algorithm 6.1 [Bohannon et al. 2005]) in the context of discussing data repair while trusting the declared quality rules. It has been shown [Bohannon et al. 2005] that, even if the set of ICs Σ are FDs only, the problem of determining if there exists a repair I' such as $cost(I, I') < W$, for a given constant W, is NP-complete. Obviously, for constraints that are more expressive than FDs, such as DCs, the data repair problem is even harder.

Definition 6.3 Set-Minimal Repair. A repair I' of I is set-minimal if and only if there is no repair I'' such that $\Delta(I, I'') \subset \Delta(I, I')$ and for each $C \in \Delta(I, I''), I''(C) = I'(C)$.

A repair I' of I is set-minimal if and only if no subset S of the changed cells in I' can be reverted to their original values while keeping the current values of other cells in $Delta(I, I') \setminus S$ unchanged. Most existing literature on consistent query answering assumes set-minimal repairs [Arenas et al. 1999, Lopatenko and Bertossi 2007, Bertossi 2011].

Definition 6.4 Cardinality-Set-Minimal Repair. A repair I' of I is cardinality-set-minimal if and only if there is no repair I'' such that $\Delta(I, I'') \subset \Delta(I, I')$.

A repair I' of I is cardinality-set-minimal if and only if no subset S of the changed cells in I' can be reverted to their original values, even if the current values of cells in $\Delta(I, I') \setminus S$ are allowed to be changed to other values.

Example 6.13 Figure 6.13 shows several examples of different notions of minimal repairs. Repair I_1 is cardinality-minimal because no other repair has fewer changed cells. By definition, Repair I_1 is also cardinality-set-minimal and set-minimal. Repairs I_2 and I_3

RIGHTSLINK

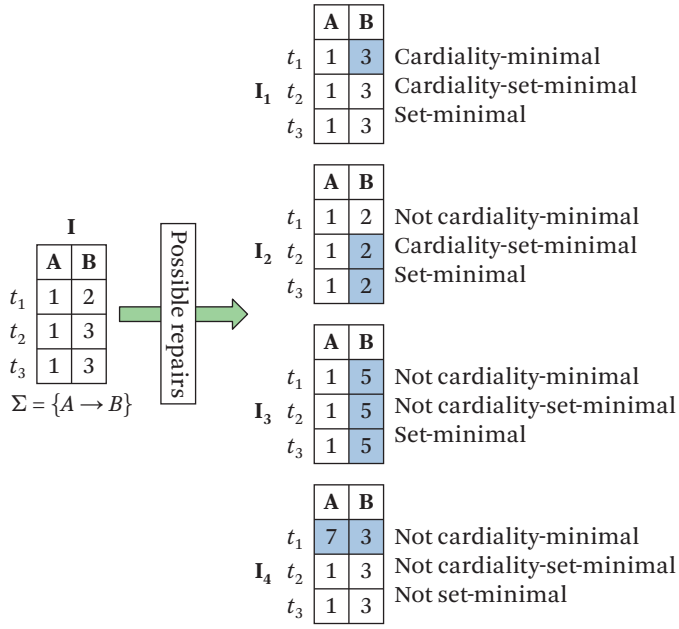


Figure 6.13 Examples of various types of repairs.

are set-minimal because reverting any subset of the changed cells to the values in I will violate $A \rightarrow B$. On the other hand, I_3 is not cardinality-set-minimal (hence not cardinality-minimal) because reverting $t_2[B]$ and $t_3[B]$ to 3 and changing $t_1[B]$ to 3 instead of 5 gives a repair of I, which is the same as I_1. I_3, however, is set-minimal, since reverting any subset of the changed cells to the values in I will still violate the FD. Repair I_4 is not set-minimal because I_4 still satisfies $A \rightarrow B$ after reverting $t_1[A]$ to 1.

Figure 6.14 shows the relationships between different notions of minimality in a graph [Beskales et al. 2014]. The set of cardinality-minimal repairs is a subset of cardinality-set-minimal repairs. The set of cardinality-set-minimal repairs is a subset of set-minimal repairs. Finally, the set of cost-minimal repairs is a subset of set-minimal repairs if for each cell $C \in CIDs(I)$, $w(C) > 0$. In general, cost-minimal repairs are not necessarily cardinality-minimal or cardinality-set-minimal, and vice versa. However, for a constant weighting function w (all cells are equally trusted) and a constant distance function *dis* (the distance between any pair of values is the same), the set of cost-minimal repairs and the set of cardinality-minimal repairs coincide.

RIGHTSLINK

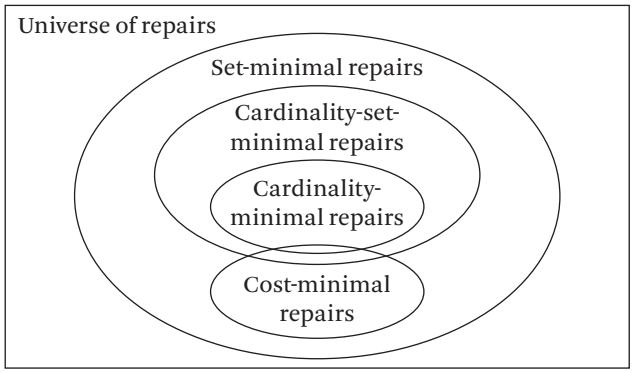


Figure 6.14 The relationships among different notions of minimality [Beskales et al. 2010].

We give the details of an algorithm that produces cardinality-minimal repairs [Kolahi and Lakshmanan 2009]. A technique that samples from the space of cardinality-set-minimal repairs for FDs [Beskales et al. 2010] is discussed in Section 6.2.7 in the context of creating repairing models, instead of cleaning data in situ.

Generating Cardinality-Minimal Repairs

Algorithm 6.5 finds a repair I' whose distance to I (number of changed cells) is within a constant factor of the optimum repair distance, where the constant factor depends on the set of FDs [Kolahi and Lakshmanan 2009]. The algorithm captures the interplay among the defined FDs in a hypergraph, where each node represents a cell in the database, and a hyperedge comprises multiple cells that cannot coexist together. We call this data structure a *conflict hypergraph*. The algorithm uses the notion of *core implicant* to ensure the termination of the algorithm. A *core implicant* of an attribute A w.r.t. a set of FDs Σ is a minimal set C_A of attributes such that C_A has at least one common attribute with every implicant X of A, where X is an implicant of A if Σ implies a nontrivial FD $X \rightarrow A$. A *minimal core implicant* of an attribute A is the core implicant with the smallest number of attributes. Intuitively, by putting variables in Attribute A and all attributes in the core implicant of A, all violations involving A are resolved and no more new violations can be introduced, where a variable denotes an unknown value that is not in the active domain where two different variables will have different values.

The algorithm works as follows. First, an initial conflict hypergraph $\mathcal{G}_I$ is built for I. Then, an approximate minimum vertex cover, VC, in $\mathcal{G}_I$ is found. For each cell

RIGHTSLINK

Algorithm 6.5 **FindVRepairFDs**

Input: Database instance I, a set of FDs Σ
Output: Another instance I' such that $I' \models \Sigma$
create an initial conflict hypergraph $\mathcal{G}_I$ for I
find an approximation VC for minimum vertex cover in $\mathcal{G}_I$
$change \leftarrow VC$
$I' \leftarrow I$
while there exist two tuples $t_1, t_2 \in I'$ violating an FD $X \rightarrow A \in \Sigma$ and $t_1[A]$ is the only cell in VC **do**
  $t_1[A] \leftarrow t_2[A]$
  $change \leftarrow change - t_1[A]$
end while
for all Cell $t[B] \in change$ **do**
  $I'(t[B]) \leftarrow$ fresh variable
end for
if there are new violations **then**
  let $t[B] \leftarrow$ a cell in VC with the largest number of violations involving $t[B]$
  let CI be the set of attributes in the *minimal core implicant* of Attribute B w.r.t. Σ
  for all Attribute $C \in CI \cup B$ **do**
    $I'(t[C]) \leftarrow$ fresh variable
  end for
end if

in VC, a value from the active domain (values that appear in the instance) is chosen if it satisfies the set of defined FDs; otherwise, a new variable is chosen. After all cells in VC have been changed, the resulting I' may contain new violations. A new violation of an FD $X \rightarrow B$ is resolved by putting variables in one of the violating tuple for Attributes B and the attributes in the minimal core implicant of B, which ensures that no more violations are introduced [Kolahi and Lakshmanan 2009].

Example 6.14 Consider a relational schema $R(A, B, C, D, E)$ with FDs $\Sigma = \{A \rightarrow C, B \rightarrow C, CD \rightarrow E\}$. An instance I is shown in Figure 6.15 with three hyperedges of three different types (not all hyperedges are shown). The first type of hyperedge is due to violation of a single FD, such as hyperedge e_1 that consists of four cells: $t_1[B]$, $t_2[B]$, $t_1[C]$, and $t_2[C]$, which together violate the FD $B \rightarrow C$. The second type of hyperedge is due to the interaction of two FDs that share the same RHS attribute, such as hyperedge e_2 that consists of six cells: $t_1[A]$, $t_1[B]$, $t_2[B]$, $t_2[C]$, $t_3[A]$, and $t_3[C]$, which cannot coexist due to the two FDs $A \rightarrow C$ and $B \rightarrow C$. The third type of hyperedge is due to the

RIGHTSLINK

	A	B	C	D	E
t_1	a_1	b_1	c_1	d_1	e_1
t_2	a_2	b_1	c_2	d_2	e_2
t_3	a_1	b_3	c_3	d_3	e_3
t_4	a_4	b_4	c_4	d_4	e_4
t_5	a_5	b_4	c_5	d_5	e_5
t_6	a_6	b_6	c_4	d_6	e_6

Figure 6.15 An initial conflict hypergraph [Kolahi and Lakshmanan 2009].

interaction of two FDs, where the RHS of one FD is part of the LHS of the other, such as hyperedge e_3 that consists of eight cells: $t_4[B]$, $t_4[C]$, $t_5[B]$, $t_5[D]$, $t_5[E]$, $t_6[C]$, $t_6[D]$, and $t_6[E]$, which cannot coexist due to the two FDs $B \rightarrow C$ and $CD \rightarrow E$.

There are two other hyperedges not shown in that consists of four cells: $t_1[A]$, $t_1[C]$, $t_3[A]$, and $t_3[C]$, and hyperedge e_5 that consists of four cells: $t_4[B]$, $t_4[C]$, $t_5[B]$, and $t_5[C]$.

Suppose $VC = \{t_2[C], t_3[C], t_4[B]\}$. Algorithm 6.5 enforces $t_2[C]$ to be the value c_1 of $t_1[C]$ because $t_2[C]$ is the only cell in VC among all cells in hyperedge e_1. Similarly, $t_3[C]$ is assigned the value c_1 of $t_1[C]$. $t_4[B]$ is changed to a fresh variable. Algorithm 6.5 terminates after all cells in VC are changed, because there are no more new violations introduced.

6.2.6 How to Repair: Human Guided Repair

Automatic data repair techniques use heuristics such as minimal repairs to automatically repair the data in situ, and they often generate unverified fixes. Worse still, they may even introduce new errors during the process. It is often difficult, if not impossible, to guarantee the accuracy of any data repair techniques without external verification via experts and trustworthy data sources.

Example 6.15 Consider two tuples, t_1 and t_8, in Table 5.1; they both have the same values "25813" for ZIP attribute, but t_1 has "WA" for ST attribute and t_8 has "WV" for ST attribute. Clearly, at least one of the four cells—$t_1[ZIP]$, $t_8[ZIP]$, $t_1[ST]$, $t_8[ST]$—has to be incorrect. Lacking other evidence, existing automatic repairing techniques [Bohannon et al. 2005, Chu et al. 2013b] often randomly choose one of the four cells to

RIGHTSLINK

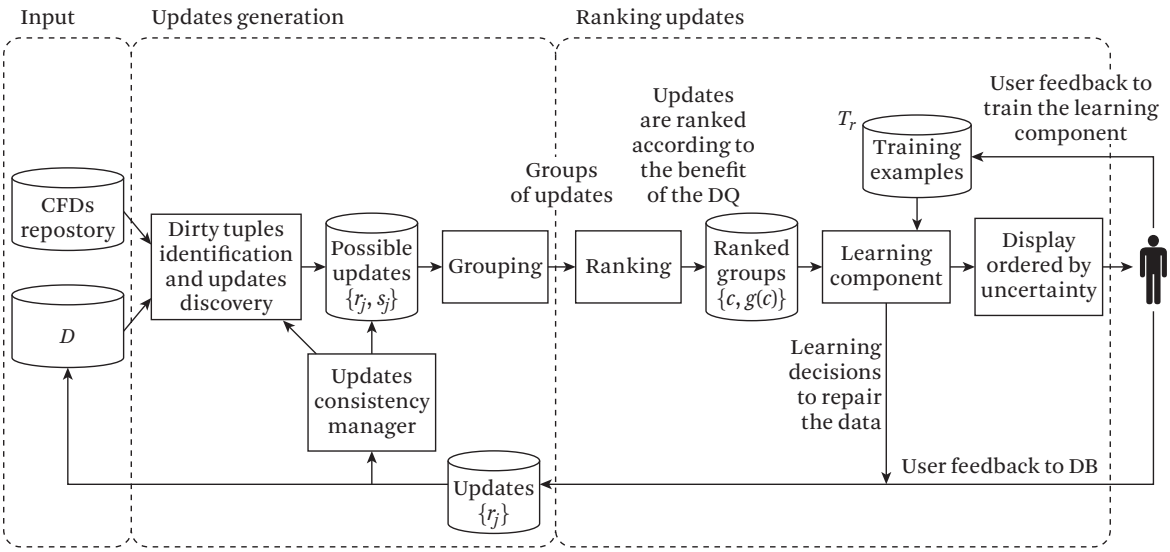


Figure 6.16 Guided data repair framework [Yakout et al. 2011].

update. Some of them [Bohannon et al. 2005] even limit the allowed changes to be $t_1[ST]$, $t_8[ST]$, since it is unclear which values $t_1[ZIP]$, $t_8[ZIP]$ should take if they are to be changed.

This shooting-in-the-dark approach, adopted by most automatic data cleaning algorithms, motivated new approaches that effectively involve humans or experts in the cleaning process to generate reliable fixes. We list a few examples: guided data repair (GDR) [Yakout et al. 2011] shows how to effectively incorporate user feedback into CFDs repairing algorithms; editing rules [Fan et al. 2010] uses tabular master data and humans to generate verified fixes; and KATARA [Chu et al. 2015] combines KBs (e.g., Yago and DBPedia), which is a collection of curated facts, such as *China hasCapital Beijing*, and crowdsourcing to discover and verify table patterns, identify errors, and suggests possible fixes.

Guided Data Repair

GDR [Yakout et al. 2011] incorporates user feedback in the data cleaning process to enhance and accelerate automatic data repair techniques for CFDs while minimizing user involvement. Figure 6.16 shows the GDR framework.

Given a database instance and a set Σ of CFDs, violations of Σ are considered *dirty* tuples; they are identified and stored in a *DirtyTuples* list. For an attribute A

RIGHTSLINK

in a dirty tuple t violating a CFD $\varphi \in \Sigma$, an *on demand* update discovery process based on the mechanism described in Cong et al. [2007] for resolving CFDs violations and generating candidate updates is used to suggest an update for the cell $t[A]$. Initially, the process is called for all dirty tuples and their attributes. Later, during the phase of user interaction, a *consistency manager* triggers the repairing process upon receiving user feedback. The generated updates are tuples in the form $r_j = \langle t, A, v, s_j \rangle$ stored in the *PossibleUpdates* list, where v is the suggested value for $t[A]$ and s_j is the *update score* assigned to each update r_j to reflect the confidence of the repairing technique in the suggested update. Once an update $r = \langle t, A, v, s \rangle$ is confirmed to be correct either by the user or by the *learning component*, it is immediately applied to the database resulting into a new database instance. A set of updates is grouped together if they are updating the same attribute to the same value; grouping provides contextual information to make it easier for the user to verify the suggested repairs. The groups are ranked according to the expected quality gain of each group. The quality gain is estimated by computing the difference between the expected number of violations before and after processing the updates in that group. The cost of acquiring user feedback for verifying each update is reduced by training an ML classifier (using an active learning technique) to replace the user later in the process. The use of a learning component in GDR is motivated by the correlation between the original data and the correct updates. If this correlation can be identified and represented in a classification model, then the model can be trained to predict the correctness of a suggested update and hence replace the user for similar (future) situations.

Example 6.16 Consider the following example. Let Relation `Customer(Name, SRC, STR, CT, STT, ZIP)` specify personal address information Street (`STR`), City (`CT`), State (`STT`), and (`ZIP`), in addition to the source (`SRC`) of the data. An instance of this relation is shown in Figure 6.17 along with a set of CFDs.

Assume that a cleaning algorithm gives two groups of updates: the first group suggests assigning attribute `CT` to the value 'Michigan City' for t_2, t_3, and t_4, and the second group suggests assigning attribute `ZIP` with the value 46825 for t_5 and t_8. Assume further that the user provides the correct values for these tuples; the user has confirmed "Michigan City" as the correct `CT` for t_2, t_3 but an incorrect `CT` for t_4, and 46825 as the correct `ZIP` for t_5 but an incorrect `ZIP` for t_8. The hypothesis in GDR is that consulting the user on the first group, which has more correct updates, is better and would allow faster convergence to a cleaner database instance as desired by the user.

RIGHTSLINK

(a) Data

	Name	SRC	STR	CT	STT	ZIP
t_1	Jim	H1	Redwood Dr	Michigan City	MI	46360
t_2	Tom	H2	Redwood Dr	Westville	IN	46360
t_3	Jeff	H2	Birch Parkway	Westville	IN	46360
t_4	Rick	H2	Birch Parkway	Westville	IN	46360
t_5	Joe	H1	Bell Avenue	Fort Wayne	IN	46391
t_6	Mark	H1	Bell Avenue	Fort Wayne	IN	46825
t_5	Cady	H2	Bell Avenue	Fort Wayne	IN	46825
t_5	Sindy	H2	Sheriden Rd	Fort Wayne	IN	46774

(b) CFD Rules

ϕ_1	ZIP $\rightarrow$ CIT, STT, {46360 ‖ Michigan City, IN}
ϕ_2	ZIP $\rightarrow$ CIT, STT, {46774 ‖ New Haven, IN}
ϕ_3	ZIP $\rightarrow$ CIT, STT, {46825 ‖ Fort Wayne, IN}
ϕ_4	ZIP $\rightarrow$ CIT, STT, {46391 ‖ Westville, IN}
ϕ_5	STR, CT $\rightarrow$ ZIP, {- ,Fort Wayne ‖ -}

Figure 6.17 Guided data repair example [Yakout et al. 2011].

There could be a correlation between the attribute values in a tuple and the correct updates. For example, when SRC = 'H2', CT is incorrect most of the time, while ZIP is correct. This is an example of a recurrent mistake that exists in real data. Patterns such as this with a correlation between the original tuple values and the correct updates, if captured by an ML algorithm, can reduce user involvement.

Editing Rules

Another example of involving humans to generate verified fixes is to use editing rules [Fan et al. 2010]. Given an input tuple $t \in I$ of schema R to be fixed, a set of eRs Σ, and a master data relation I_m of schema R_m, a certain fix t' of t needs to be found by interacting with the users and t' satisfies the following properties: (1) no matter how Σ and tuples in I_m are applied, Σ and I_m will yield a unique t', and (2) all attributes of t' are guaranteed to be correct. Figure 6.18 depicts the framework for using Σ and I_m to derive t' for t. Specifically, the framework works as follows.

RIGHTSLINK

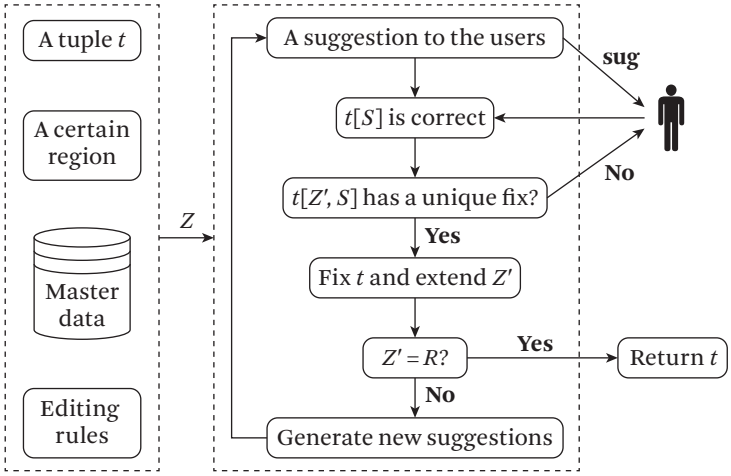


Figure 6.18 Repairing framework for using editing rules and master data [Fan et al. 2010].

Initialization. Given t, it picks a precomputed certain region Z and recommends Z to the user. A certain region is a set of attributes that are guaranteed to be correct. Z' denotes a set of attributes that have already been validated by the user to be correct.

Generating Correct Fixes. In every interaction with the user, a set of attributes, initially Z, is shown to the user, who chooses a subset S of asserted correct attributes. If $t[S \cup Z']$ leads to a unique fix, t is fixed and Z' is extended; otherwise, users are asked to provide new suggestions.

Generating New Suggestions. If Z' covers all attributes, a certain fix has been found; otherwise, a new set of suggestions are computed for the next round of user interaction.

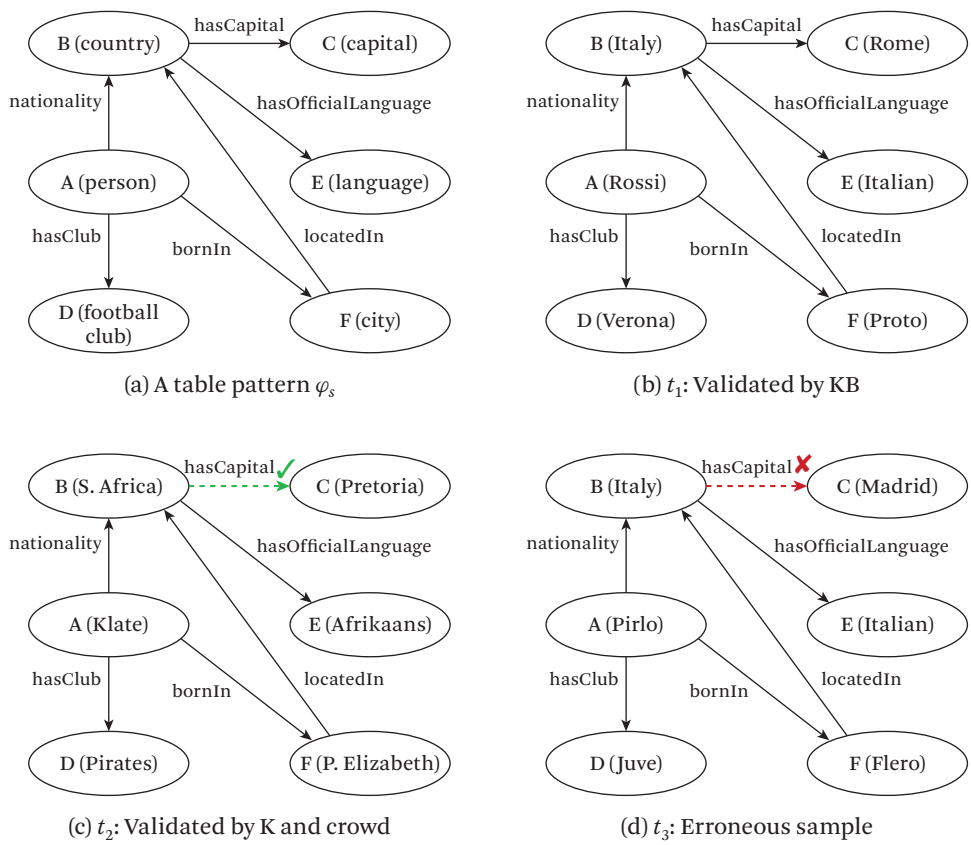
KATARA

KATARA [Chu et al. 2015] aims at producing accurate repairs by relying on two authoritative data sources, namely, knowledge bases (KBs) and domain experts. KATARA first discovers table patterns to map the table to a KB such as Yago or DBPedia. With table patterns, KATARA annotates tuples as either correct or incorrect by interleaving the KB and humans. For incorrect tuples, KATARA will extract top-k mappings from the KB as possible repairs that are to be examined by humans.

RIGHTSLINK

	A	B	C	D	E	F	G
t_1	Rossi	Italy	Rome	Verona	Italian	Proto	1.78
t_2	Klate	S. Africa	Pretoria	Pirates	Afrikaans	P. Eliz.	1.69
t_3	Pirlo	Italy	Madrid	Juve	Italian	Flero	1.77

Figure 6.19 A table $\mathcal{T}$ for soccer players [Chu et al. 2015].



(a) A table pattern φ_s (b) t_1: Validated by KB

(c) t_2: Validated by K and crowd (d) t_3: Erroneous sample

Figure 6.20 KATARA patterns [Chu et al. 2015].

Consider a table $\mathcal{T}$ for soccer players (Figure 6.19). Table $\mathcal{T}$ has no table header, thus its semantics are completely unknown. Assume that a KB $\mathcal{K}$ (e.g., Yago) contains some information related to $\mathcal{T}$. KATARA [Chu et al. 2015] works as follows:

RIGHTSLINK

Pattern discovery. KATARA first discovers table patterns that contain the types of the columns and the relationships between them. A table pattern is represented as a labeled graph (Figure 6.20a) where a node represents an attribute and its associated type, e.g., "C (capital)" means that the type of attribute C in KB $\mathcal{K}$ is capital. A directed edge between two nodes represents the relationship between two attributes, e.g., "B hasCapital C" means that the relationship from B to C in $\mathcal{K}$ is hasCapital. A column could have multiple candidate types, e.g., C could also be of type city. However, knowing the relationship from B to C is hasCapital indicates that capital is a better choice. Since KBs are often incomplete, the discovered patterns may not cover all attributes of a table, e.g., attribute G of table $\mathcal{T}$ is not described by the pattern in Figure 6.20a.

Pattern validation. Consider a case where pattern discovery finds two similar patterns: the one in Figure 6.20(a) and its variant with type location for column C. To select the best table pattern, we send the crowd the question "*Which type (*capital *or* location*) is more accurate for values (*Rome, Pretoria *and* Madrid*)?*" Crowd answers will help choose the right pattern.

Data annotation. Given the pattern in Figure 6.20(a), KATARA annotates each tuple with one of the following three labels:

(i) Validated by the KB. By mapping tuple t_1 in table $\mathcal{T}$ to $\mathcal{K}$, we found a full match, shown in Figure 6.20(b), indicating that Rossi (resp. Italy) is in $\mathcal{K}$ as a person (resp. country), and the relationship from Rossi to Italy is nationality. Similarly, all other values in t_1 with respect to attributes A-F are found in $\mathcal{K}$. We consider t_1 to be correct with respect to the pattern in Figure 6.20(a) and to attributes A-F.

(ii) Jointly validated by the KB and the crowd. Consider t_2 about Klate, whose explanation is depicted in Figure 6.20(c). In $\mathcal{K}$, we find that S. Africa is a country, Pretoria is a capital. However, the relationship from S. Africa to Pretoria is missing. A positive answer from the crowd to the question "*Does* S. Africa hasCapital Pretoria*?*" completes the missing mapping. We consider t_2 correct and generate a new fact "S. Africa hasCapital Pretoria."

(iii) Erroneous tuple. Similar to case (*ii*). For tuple t_3, there is no link from Italy to Madrid in $\mathcal{K}$. A negative answer from the crowd to the question "*Does* Italy hasCapital Madrid*?*" confirms that there is an error in t_3. At this point, however, we cannot decide which value in t_3 is wrong, Italy or Madrid. KATARA extracts evidence from $\mathcal{K}$, e.g., Italy hasCapital

RIGHTSLINK

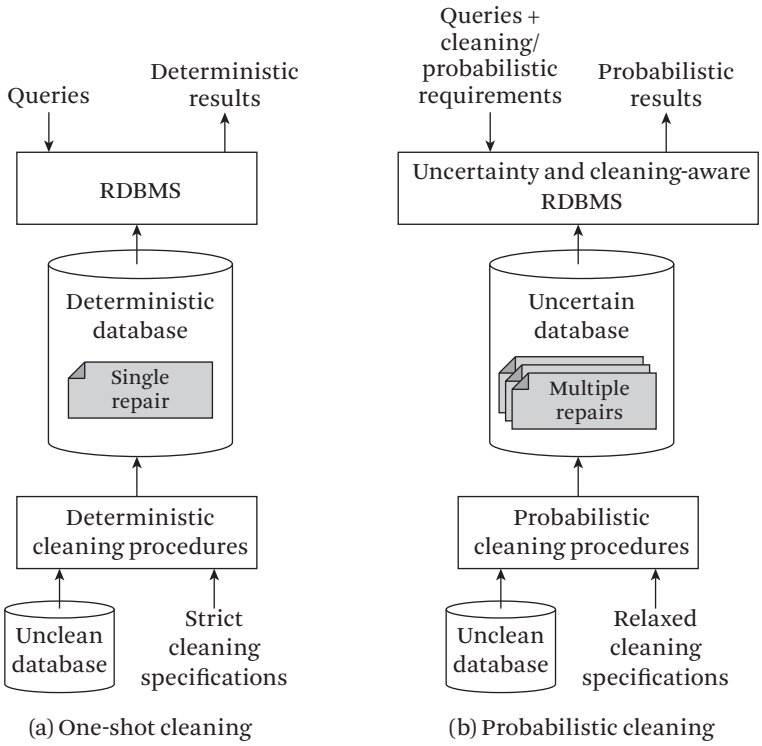


Figure 6.21 One-shot vs. probabilistic cleaning.

Rome and Spain hasCapital Madrid, joins them, and generates a set of possible repairs for this tuple.

6.2.7 Where to Repair: Model-Based Repair

Data repair techniques are classified based on whether the database will be changed in place by the repairing techniques or by a model that describes the possible changes that will be used to answer queries against the dirty data. Most of the proposed data repair techniques (all discussed so far) identify errors in the data and find a unique fix of the data either by minimally modifying the data according to a cost function or by using human guidance (Figure 6.21(a)).

In what follows, we describe a different model-based approach for non-destructive data cleaning. Data repair techniques in this category do not produce a single repair for a database instance; instead, they produce a space of possible repairs (Figure 6.21(b)). The space of possible repairs is used either to answer

RIGHTSLINK

queries against the dirty data probabilistically (e.g., using possible worlds semantics) [Beskales et al. 2009] or to sample from the space of all possible clean instances of the database [Beskales et al. 2010, Beskales et al. 2014].

We give details of one algorithm, which considers the space of possible repairs of FD and CFD violations and provides a sampling technique to sample possible clean instances with certain minimality guarantees, more specifically, with cardinality-set-minimal repairs (cf. Section 6.2.5) [Beskales et al. 2010, Beskales et al. 2014]. Techniques from consistent query answering [Arenas et al. 1999, Lopatenko and Bertossi 2007] also fall in this category, since they consider a tuple in the original dirty database instance I to be in the answer of a query if that tuple is present in every possible repair I' of I. We refer readers to a survey [Bertossi 2011] for a comprehensive treatment of consistent query answering.

For any two tuples t_1, t_2 that violate an FD $X \rightarrow A$, the violation can be repaired by either changing $t_1[A]$ to be the value of $t_2[A]$ (or vice versa) or modifying an attribute $B \in X$ in either t_1 or t_2 so that $t_1[B] \neq t_2[B]$. Generalizing this observation, if a set of CleanCells does not violate any FD in Σ, the consistency of CleanCells $\cup\, C$, for any cell C, can always be ensured by modifying C if necessary. To systematically determine whether a set of cells is clean, the equivalence classes $\mathcal{E}$ for that set of cells are built. An equivalence $eq \in \mathcal{E}$ denotes a subset of cells that should be equal according to Σ. Thus, to check if a set of cells is clean or not, it is sufficient to check if any two cells in an equivalent class $eq \in \mathcal{E}$ indeed have the same value.

Example 6.17 For example, to determine if the set of six cells, $t_1[A]$, $t_1[C]$, $t_2[A]$, $t_2[B]$, $t_3[B]$, and $t_3[C]$, in Figure 6.22 is clean or not, a set of equivalence classes $\mathcal{E}$ is built. Initially, $t_1[A]$, $t_2[A]$ belong to the same equivalence class, since they have the same value. Similarly $t_2[B] and t_3[B]$ also belong to the same equivalence class. However, $t_1[C]$, and $t_3[C]$ form two separate equivalence classes. According to the FD $A \rightarrow C$, $t_1[C]$ and $t_2[C]$ should belong to the same equivalence class. According to the FD $B \rightarrow C$,

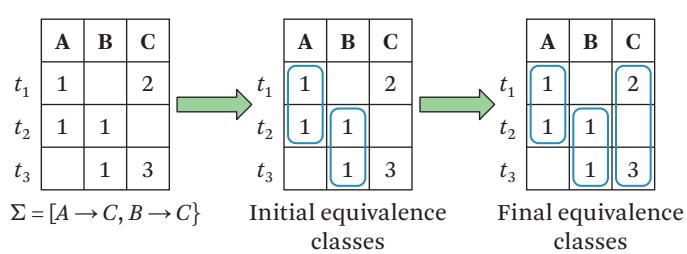


Figure 6.22 An example of checking whether a set of cells is clean.

RIGHTSLINK

Algorithm 6.6 **Sampling FDs repairs**

Input: Database instance I, a set of FDs Σ
Output: Possible repairs I'
$I' \leftarrow I$
CleanCells $\leftarrow \emptyset$
while CleanCells $\neq CIDs(I')$ **do**
  insert a random cell $t[A] \in CIDs(I') \setminus$ CleanCells to CleanCells,
  where $t \in I$ and $A \in R$
  build the equivalence classes $\mathcal{E}$ of CleanCells according to Σ
  if CleanCells is not clean w.r.t. $\mathcal{E}$ **then**
    Build the equivalence classes $\mathcal{E}_p$ of CleanCells$t[A]$ according to Σ
    if $t[A]$ belongs to a non-singleton equivalence class in $\mathcal{E}_p$ **then**
      set $I'(t[A])$ to the value of other cells that are in the same equivalence class in $\mathcal{E}_p$
    else
      randomly set $I'(t[A])$ to one of the three alternatives: a randomly selected constant from $Dom(A)$, a randomly selected variable that appears in I', or a new variable such that CleanCells is clean with respect to $\mathcal{E}$
    end if
  end if
end while
return I'

$t_2[C]$ and $t_3[C]$ should belong to the same equivalence class. Thus, $t_1[C]$ and $t_3[C]$ end up in the same equivalence class. However, $t_1[C]$ and $t_3[C]$ have different values. Therefore, the set of six cells is not clean.

Algorithm 6.6 describes the procedure for generating repairs. Cells are inserted into CleanCells in random order. At each iteration, the algorithm checks whether CleanCells is clean by building the equivalence classes $\mathcal{E}$ according to Σ. If CleanCells is not clean, the last inserted Cell C is changed so that CleanCells is clean again.

An example of executing Algorithm 6.6 is shown in Figure 6.23. At each step, the cells that have been selected so far by the algorithm are shown. Equivalence classes are shown as rectangles. The cells $t_1[A]$, $t_1[B]$, $t_2[A]$, and $t_3[B]$ are added to *CleanCells* in step (a). They are all clean and do not need to change. The cell $t_2[B]$ is added to *CleanCells* in step (b). Because the cells $t_1[B]$ and $t_2[B]$ belong to the same equivalence class, the value of $t_2[B]$ must be changed to the value of $t_1[B]$, which is

RIGHTSLINK

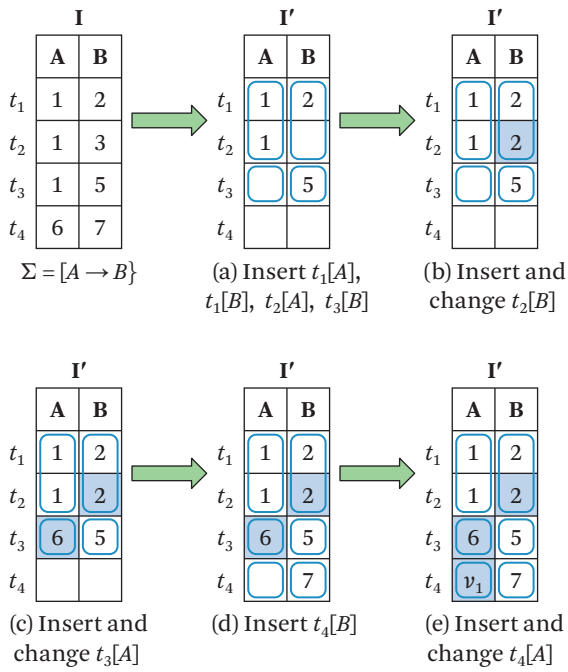


Figure 6.23 An example of executing Algorithm 6.6 [Beskales et al. 2010].

2. In step (c), the cell $t_3[A]$ is added to *CleanCells*. The value of $t_3[A]$ is changed to a randomly selected constant, namely 6, to resolve the violation. We continue adding the remaining cells and modifying them as needed to make sure that *CleanCells* is clean after each insertion. Finally, the resulting instance I' represents a repair of I. Algorithm 6.6 is extended to sampling from the space of all cardinality-set-minimal repairs for CFDs [Beskales et al. 2014].

6.3 Conclusion

Given a database instance I and a set of data quality rules Σ, rule-based data cleaning proceeds in two steps: detecting violations in I with respect to Σ and updating I (and potentially also Σ) such that I conforms to Σ.

While violation detection is a fairly straightforward procedure algorithmically, namely, enumerating tuple combinations to determine whether they violate rules in Σ, we identified three practical challenges and discussed proposals addressing those challenges. First, detecting a violation does not necessarily pinpoint the exact erroneous cell since a violation may involve multiple cells. We introduced the idea

RIGHTSLINK

of holistic data cleaning that leverages conflict hypergraphs to identify cells that are more likely to be erroneous. Second, there can be decoupling between the space and the time errors are detected; errors usually happen in the data source, but may not be detected until after the data has gone through multiple stages of processing. We discussed techniques that propagate errors detected in transformation results to the data sources. Finally, to tackle the scalability challenges associated with violation detection, we presented a proposal that distributes the task in the common shared-nothing parallel computing environment such as Hadoop.

We presented a classification of various data repair proposals using three dimensions: what to repair, how to repair, and where to repair. The "what to repair" dimension captures the evolving nature of both the data and the business logic (integrity constraints). It classifies repairing algorithms into those that update the data only, those that update the rules only, and those that update both. The "how to repair" dimension classifies error repair techniques into fully automatic techniques and those that involve human experts. In real-world scenarios, automatic techniques are rarely used and humans often need to either manually update the data or approve the updates suggested by automatic tools. The "where to repair" dimension classifies proposed approaches based on whether they change the database in situ or build a model to describe the repair. While most techniques update the data in place, some techniques recognize the fact that there may not exist a "golden" repair and hence use a model to describe the space of repairs.

RIGHTSLINK

7 Machine Learning and Probabilistic Data Cleaning

In this chapter, we discuss how data cleaning can be viewed as a probabilistic database problem, where statistical and probabilistic interpretation of data errors can lead to more general and holistic error detection and repair solutions. Most of the data repair techniques described in previous chapters view inconsistencies in databases as violations of logical rules. These techniques resolve constraint violations by taking actions to bring the data into conformance with these *hard* integrity constraints, including, for example, heuristically changing an attribute value of a tuple to eliminate a function dependency violation or entirely removing a set of tuples.

While statistical and probabilistic solutions have been employed in data cleaning for decades, they have been limited to mostly numerical outlier detection techniques [Barnett and Lewis 1994, Hawkins 1980, Hawkins et al. 2002], as we discussed in Chapter 2, and building binary classifiers for de-duplication [Fellegi and Sunter 1969, Sarawagi and Bhamidipaty 2002], as we briefly discussed on Chapter 3. With the popularity and the availability of resources to build large-scale machine learning solutions, leveraging ML techniques for data cleaning is becoming a popular and a promising direction. Since these ML models are built on a probabilistic view of the underlying data, a deeper understanding of the interaction between cleaning and probabilistic modeling of data becomes a necessity.

One of the biggest advantages of adopting a probabilistic view of data cleaning is the fact that this view enables a holistic treatment of a variety of data errors in one platform. We discussed in previous chapters many examples of data errors and anomalies, including outliers, duplicates, violations of integrity constraints, and misalignments. While it is easier to formulate and tackle these data problems

RIGHTSLINK

in isolation, in reality, most dirty datasets suffer from most or all of these problems combined. Furthermore, these different errors interact in non-trivial ways. For example, we cannot find duplicates effectively without mapping schemas, wrangling data sources, imputing missing values, and removing outliers. Studies have shown that there does not exist a single best sequence of cleaning exercises due to their dependencies; even when multiple tools are applied, the performance is often poor because none has captured the holistic nature of the data cleaning process [Abedjan et al. 2016a, Stonebraker and Ilyas 2018]. Treating the data cleaning problem as a large-scale ML problem is a principled and promising way to address these issues.

Multiple ML concepts and techniques have been used or are currently proposed as natural choices to build data cleaning solutions. For example, factor graphs (a type of probabilistic graphical models; Koller and Friedman [2009]) are used to capture the correlation among the various features and signals involved in predicting the most likely value of an erroneous database cell (attribute value of a tuple) [Rekatsinas et al. 2017a]. Probabilistic graphical models have been also used to estimate the accuracy and fidelity of sources when integrating multiple data sets [Rekatsinas et al. 2017b]. Active learning was used to effectively involve human experts to obtain labeled data to train ML models for the record linkage problem [Sarawagi and Bhamidipaty 2002]. In this chapter, we explore how ML and statistical inference have been used for large-scale data cleaning problems through concrete algorithms and examples. In Section 7.1, we focus on the deduplication problem as a classical example of using ML in a data cleaning task, where we highlight traditional active learning techniques and more recent proposals based on deep learning (DL). In Section 7.2, we discuss in detail new advances in using ML and statistical learning for general data repair. We also discuss in Section 7.3 how cleaning can serve as an important data preparation step for downstream analytics, including building effective ML models.

7.1 Machine Learning for Data Deduplication

ML solutions have been applied to data deduplication, as deciding whether two records are duplicates or not is by definition a binary classification problem. Indeed, we have seen how ML techniques are used in data deduplication to learn a classifier to determine whether two records are duplicates using similarity scores as features (cf. Section 3.2). In this section, we explore how active learning is used for data deduplication when there are not enough labeled matches and non-matches to build a good classifier, and how DL can also be useful for certain data deduplication scenarios.

RIGHTSLINK

7.1.1 Active Learning in Data Deduplication

As discussed in Section 3.2, data deduplication involves training a binary classifier to predict whether a pair of records are duplicates or distinct. To train a binary classifier with good prediction accuracy, we need a sufficiently large training set, namely, a set of duplicate record pairs and non-duplicate record pairs. However, obtaining such a large training set usually requires human labeling, which can be an expensive process. Active learning techniques have been explored to judiciously solicit training examples for training a classifier in the data deduplication problem [Tejada et al. 2001, Sarawagi and Bhamidipaty 2002, Arasu et al. 2010, Stonebraker et al. 2013, Konda et al. 2016]; the general idea is to solicit user feedback for unlabeled record pairs which, when labeled, will provide the highest utility to the training process.

There are certain difficulties that are specific to learning a classification model for data deduplication. First, in a typical setting, the number of non-matches far exceeds the number of matches; consider a record matching task involving two different tables with 1000 records each, and on average, each record matches with one or two records of the other table. The number of matches for this task is about 2×10^3, while the number of non-matches is about 10^6. This imbalance between matches and non-matches makes it difficult to identify a suitable set of training examples to learn a classifier with high accuracy. Second, even if we manage to find an equal number of matches and non-matches, the success of deriving a high accuracy classifier for data deduplication depends on the quality of those training examples; for example, it would be more beneficial to include non-matches that are likely to be confused as matches than those obvious non-matches.

We discuss in detail one seminal solution, *ALIAS* [Sarawagi and Bhamidipaty 2002], to solicit training data. *ALIAS* demonstrates the core challenges of using active learning in data deduplication. Algorithm 7.1 gives the procedure of the *ALIAS* active learning procedure. It takes as input a database of records I, an initial set of training data L arranged in pairs of matches and non-matches, and a set of similarity functions F. The set of similarity functions, when applied to a record pair, would generate a comparison vector γ. In the initialization step, *ALIAS* first creates the pairs L_p and I_p from the labeled data L and the unlabeled data I by applying the set of similarity functions F. The record pairs in L_p are then used as an initial training set. In the active learning session, i.e., the loop in Algorithm 7.1, *ALIAS* chooses from the set I_p a subset S, such that the learner would benefit the most if S is labeled. The user is shown examples in S, along with the current predictions; the user can then correct any predictions in S that are wrong. The newly labeled examples in S are added to the training set T and the classifier is retrained. The

RIGHTSLINK

Algorithm 7.1 **ALIAS Active Learning Algorithm**

Input: Database instance I, initial training data L, and similarity functions F
Output: A classifier C
create pairs L_p from the labeled data L and F
create pairs I_p from the unlabeled data I and F
initialize training set $T = L_p$
loop until user satisfaction **do**
  train classifier C using T
  use C to select a set S of instances from I_p for human labeling
  exit if S is empty
  collect user feedback on the labels of S
  add S to T and remove S from D_p
end loop
output classifier C

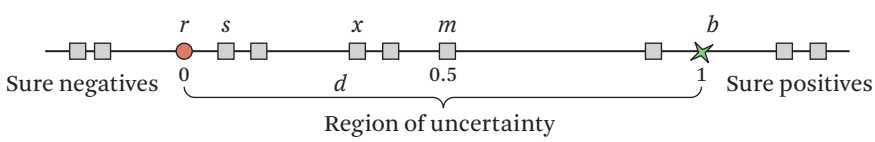


Figure 7.1 Examples of active learning to reduce confusion region [Sarawagi and Bhamidipaty 2002].

loop goes on until the user is satisfied with the classifier, for example, by evaluating its performance on a held-out test dataset.

The core step in *ALIAS* is to choose a subset S that would be most beneficial if labeled. Based on the current training data T, the classifier will be certain about some pairs in I_p and will be uncertain about other pairs in I_p. The active learner in *ALIAS* picks those pairs the classifier is most unsure about, which can help reduce a classifier's confusion. The following example demonstrates the intuition using a simple classification scenario [Sarawagi and Bhamidipaty 2002].

Example 7.1 Consider a simple learner for separating points from two different classes: positive (P) and negative (N) on a straight line, as shown in Figure 7.1. Assume that the points in the line are separable by a single point on the line. The hypothesis space consists of all points on the line, and for a given point (hypothesis), all points to the left of that point will be labeled negative and all points to the right of that point will be labeled positive. The classification task is thus to learn that point on the line that perfectly separates the positives and negatives.

RIGHTSLINK

The initial training set consists of one positive point b (green star) and one negative point r (red circle) picked randomly from the line. The rest of the points (squares) are unlabeled. At this point, the classifier is certain about the points to the left of r and is also certain about the points to the right of b. The confusion region includes the points between r and b, from which the active learner will select points for human labeling.

For any point x in this confusion region, assume that the probability that it is negative is inversely proportional to its distance from r. For simplicity, assume r has a coordinate of 0 and b has a coordinate of 1. Thus, if x has a coordinate of d, the probability that its class is negative (N) is $Pr(N|x) = 1 - d$ and the probability that its class is positive (P) is $Pr(P|x) = d$. If x were negative, the size the confusion region would reduce by d, and if it were positive, the size of the confusion region would reduce by $1 - d$. Hence, the expected reduction in the size of the confusion region when adding x to the training set is $Pr(N|x)d + Pr(P|x)(1 - d) = (1 - d)d + d(1 - d) = 2d(1 - d)$, which achieves the maximum value when $d = 0.5$ (point m in Figure 7.1). By including m in the training set, the size of the uncertain region will reduce by half no matter what its label.

If we were to choose another point (such as s in the figure) for human labeling, which is close to the negative boundary but far from the positive boundary, the confusion region could be reduced more if s turned out to be positive. However, this is unlikely given the current training data. Therefore, the expected reduction of the confusion region of choosing s is less than that of choosing m.

The above example shows a toy scenario of how to calculate the uncertainty of a prediction of an extremely simple classifier (i.e., choosing a separating single point on the line). For other more complicated classifiers, how do we quantify the uncertainly of a prediction? There are generally two categories of approaches, namely, classifier-specific approaches and classifier-independent approaches. Examples of classifier-specific approaches include using posterior probabilities of predications for Bayesian classifiers [Tong and Koller 2002] and using the inverse of the distance between an instance and the separator for SVM [Schohn and Cohn 2000]. A classifier-independent approach to derive the uncertainty is done by measuring the disagreement among the predications of a set of classifiers, also known as a *committee* [Freund et al. 1997]. The classifiers in the committee are different from each other, but have similar accuracies on the training data. A certain record pair would get the same predications from almost all committee members, while an uncertain record pair would result in disagreement, which can be quantified in various

RIGHTSLINK

ways, such as entropy on the fraction of committee members that predicate either duplicates or non-duplicates.

The uncertainty score is not the only factor considered by *ALIAS* when picking instances for human labeling, as the most uncertain point could be an outlier point that is not representative of a larger number of unlabeled instances. Therefore, *ALIAS* also considers how representative an instance is of the underlying data distribution. The main challenge in incorporating the representativeness factor is figuring out how to combine it with the uncertainty factor. Two different methods have been proposed for this in *ALIAS* [Sarawagi and Bhamidipaty 2002]. The first approach explicitly measures the representativeness of an instance by estimating the density of unlabeled instances. The instances are then scored using a weighted sum of their density and the uncertainty value. The top scoring instances are then selected for human labeling. This method requires us to have an accuracy estimate of the density of unlabeled instances, which itself could be a challenging task. The second approach relies on weighted sampling to preserve the underlying data distribution. Every unlabeled instance is weighted by its uncertainty score. Then a set of instances are selected from all unlabeled instances using weighted sampling. The premise is that outlying points are a small fraction of all unlabeled points and thus will be filtered out by the sampling procedure.

7.1.2 Deep Learning for Data Deduplication

We introduce recent work that examines different design choices available when designing a DL solution for matching (EM) [Mudgal et al. 2018], and when DL becomes a favorable approach.

Figure 7.2 shows an architecture template for building DL solutions for EM. While previously we have assumed that attribute similarities are obtained by simply applying a similarity function over attribute values from two entities, Figure 7.2 details other possible choices for obtaining attribute similarities. In particular, the template in Figure 7.2 consists of three main modules and each module has a set of choices. The combination of these choices forms a design space of possible DL solutions for EM. The template assumes as input a pair of entities (e_1, e_2), which follow the same schema with attributes $A_1, A_2, \ldots, A_N$. Textual data can be represented using a schema with a single attribute. The value of attribute A_j for an entity e is a sequence of words $w_{e,j}$; the length of these sequences could be different for different entities. Given this setup, we discuss the three modules as follows.

The Attribute Embedding Module. A word embedding maps words or phrases from a vocabulary to vectors of real numbers. For every attribute A_j, this module takes two sequences of words, $w_{e_1,j}$ and $w_{e_2,j}$, and converts them into two sequences of word

RIGHTSLINK

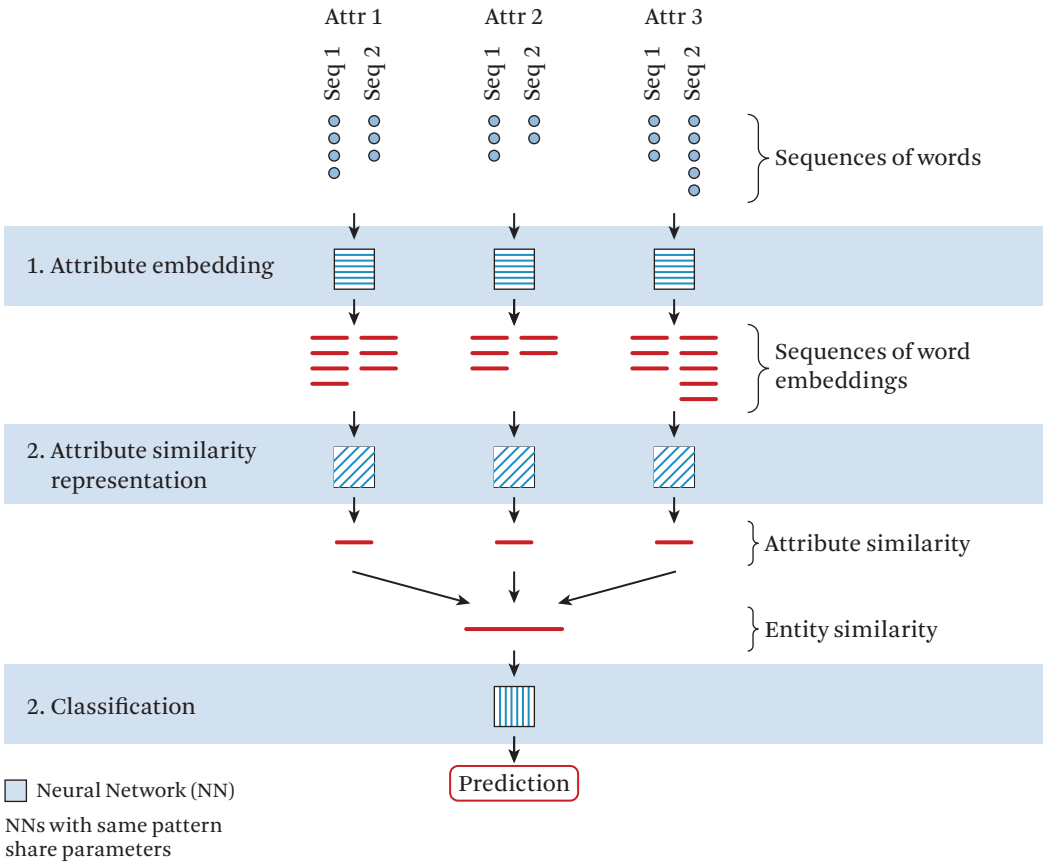


Figure 7.2 Template for DL solutions for EM [Mudgal et al. 2018].

embedding vectors $u_{e_1,j}$ and $u_{e_2,j}$ whose elements correspond to d-dimensional embeddings of the corresponding words. For example, if there are m words in $w_{e_1,j}$, then we have $u_{e_1,j} \in R^{d \times m}$.

The Attribute Similarity Representation Module. The goal of this module is to learn a representation that captures the similarity of two entities (e_1, e_2) given as input. For each attribute A_j and a pair of attribute embeddings $u_{e_1,j}$ and $u_{e_2,j}$, this module performs two operations: *Attribute Summarization* and *Attribute Comparison*. The attribute summarization operation summarizes the information contained in the attribute embeddings. More precisely, assume $u_{e_1,j} \in R^{d \times m}$ and $u_{e_2,j} \in R^{d \times k}$; this operation outputs two summary vectors of the same dimension $s_{e_1,j} \in R^h$ and $s_{e_2,j} \in R^h$. The role of attribute summarization is to aggregate information across all

RIGHTSLINK

Architecture Module		Options	
Attribute embedding		*Granularity:* (1) Word-based (2) Character-based	*Training:* (3) Pre-trained (4) Learned
Attribute similarity representation	(1) Attribute summarization	(1) Heuristic-based (3) Attention-based	(2) RNN-based (4) Hybrid
	(2) Attribute comparison	(1) Fixed distance (cosine, Euclidean) (2) Learnable distance (concatenation, element-wise absolute difference, element-wise multiplication)	
Classifier		NN (multi-layer perception)	

Figure 7.3 The design space of DL solutions for EM [Mudgal et al. 2018].

tokens in the attribute value. The attribute comparison part takes the two summary vectors $s_{e_1,j} \in R^h$ and $s_{e_2,j} \in R^h$ and applies a comparison function D to obtain a similarity measure $s_j = D(s_{e_1,j}, s_{e_2,j})$.

The Classifier Module. This module takes as input the similarity representations $\{s_1, s_2, \ldots, s_N\}$ and uses them as features to train a classifier M that determines if two input entities e_1 and e_2 are duplicates.

Given the above architecture template, multiple choices exist for each of the three modules. Figure 7.3 describes these choices, which are inspired by how DL is used in other matching tasks in NLP such as entity linking and question answering. We refer readers to the original study [Mudgal et al. 2018] for detailed discussions of these choices. In summary, the study found that DL solutions do not outperform traditional ML solutions on structured EM tasks, namely, when entities to be matched follow the same schema and the attribute values are generally clean. However, DL solutions significantly outperform traditional ML solutions on textual EM tasks and dirty EM tasks, where textual EM tasks are those with input entities containing only raw textual attributes, and dirty EM tasks are the same as structured EM tasks, except that the input entities can have dirty values. These results are not surprising, as it would be difficult for conventional ML to capture textual similarity without embedding or to capture similarity correctly if the inputs are dirty. DL approaches, especially attribute embeddings, are still able to capture entity similarities well in textual EM and dirty EM tasks.

RIGHTSLINK

7.2 Machine Learning for Data Repair

In this section we explore the use of ML models for performing general data repair. Unlike data deduplication, most data repair methods focus on violations of integrity constraints and, as we mentioned in the beginning of this chapter, adopt a logical interpretation of the repairing process, as we extensively reviewed in Chapter 6. We highlight a recent proposal of adopting a probabilistic dirty database model for data repair in Section 7.2.1, and we demonstrate in Section 7.2.2 a repair framework based on this model, which uses statistical inference to suggest data repair to erroneous cells.

7.2.1 Probabilistic Data Cleaning Models

Before we discuss example ML and statistical solutions for data repair, we highlight a formal framework for unclean databases [De Sa et al. 2019], which better explains how to incorporate statistical reasoning to build effective real-world data cleaning solutions. The model introduced in De Sa et al. [2019] represents two types of statistical knowledge: the first represents a belief of how intended (clean) data is generated, and the second represents a belief of how noise is introduced in the actual observed database. Explicitly representing the clean data generation process and the realization process that introduced errors before data was consumed allows for a variety of cleaning tasks, including error detection, probabilistic repair and even answering queries over dirty databases. The model is based on the concept of probabilistic noisy databases, which consists of three main components: (1) a probabilistic database called the intention (I); (2) a probabilistic process called realization (R) that captures how noise is introduced; and (3) a dirty observed database called the observation (J^*).

Cleaning the observed data instance (J^*) is to find the intended database I^* (among all possible clean instances I) from which this observed instance J was produced via the realization process. Adopting Bayesian inference, this intended instance (I^*) is the one with the highest probability given J: $I^* = Argmax_I Pr(I).Pr(J^*|I)$. $Pr(I)$ captures the prior model for a clean database instance, and $Pr(J^*|I)$ captures the noisy channel that produces the observed instance. Hence, a probabilistic noisy database $\mathcal{D}$ is a triple $(\mathcal{I}, \mathcal{R}, J)$, where $\mathcal{I}$ is the intention model, which is the distribution that generates clean instances I describing the prior probability $Pr(I)$; $\mathcal{R}$ is the noisy realization model, which introduces errors and gives the probability $Pr(J^*|I)$ and maps each instance I generated by $\mathcal{I}$ to probabilistic database $\mathcal{R}_I$; and J^* is the observed instance.

RIGHTSLINK

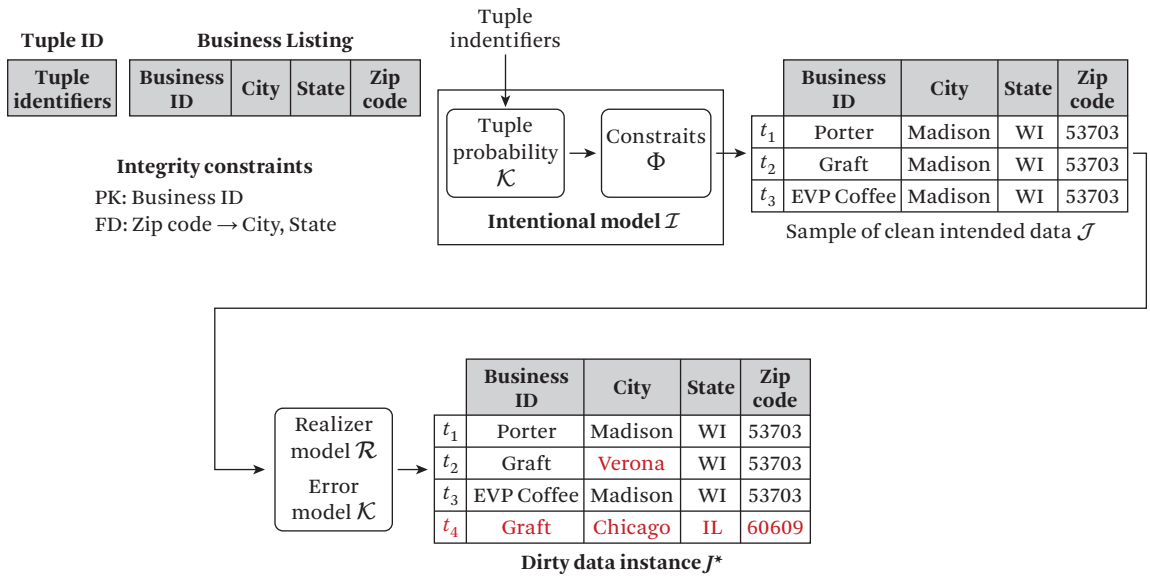


Figure 7.4 The probabilistic noisy database model.

To learn these models, the proposal in De Sa et al. [2019] adopts a parametric model representation, where Model $\mathcal{I}$ is parametrized by a set of parameters Θ and Model $\mathcal{R}$ is parametrized by a set of parameters Δ. The challenging task is to generate and use training data from the observed instance J^* to estimate the parameters Θ^* and Δ^* that maximize this training data. In a supervised learning approach, both unclean data and their clean versions will be provided as training examples (unclean data will be drawn from J^*), while in an unsupervised learning approach, only J^* will be used to generate training examples. In both cases, a large enough observed instance J^* will be decomposed to provide many training examples to learn these parameters. Once these parameters are learned, data cleaning becomes an inference exercise to estimate the most likely intention I^*, which is the maximum a posteriori (MAP) assignment of I.

An example of parametrized model for $\mathcal{I}$ which generates the instances I is a probabilistic database with two components.

- A tuple-independent probabilistic database assuming a discrete domain of identifiers $ids(I)$, where a tuple t in entry i (where i is an identifier) has a probability $\mathcal{K}[i](t)$ (with $\mathcal{K}[i](\perp)$ denoting no tuple generated for identifier i)

RIGHTSLINK

$$\mathcal{K}(I) \stackrel{\text{def}}{=} \prod_{i \in ids(I)} \mathcal{K}[i](I[i]), \qquad \prod_{i \notin ids(I)} \mathcal{K}[i](\perp).$$

- Parametric factors [Sen et al. 2009], a special case of Markov Logic Networks [Richardson and Domingos 2006], describe the (soft) integrity constraints Φ defined on the data instance, where each constraints φ in Φ is assigned a weight $w(\varphi) > 0$ and each violation of the constraints contributes a factor $exp(-w(\phi))$ to the probability of I.

Hence, the probability of an instance I in this model is defined as:

$$\Pr(I) \stackrel{\text{def}}{=} \frac{1}{Z} \times \mathcal{K}(I) \times \exp\left(-\sum_{\varphi \in \Phi} w(\varphi) \times |V(\varphi, I)|\right).$$

$|V(\varphi, I)|$ is the number of violations of constraint φ when applied to I and Z is a normalization factor (also known as a partition function) that normalizes the sum of probabilities to 1.

A concrete example of the realization model $\mathcal{R}$ that generates J^* can be a polluter that picks cells at random in I and changes the value to another value randomly picked from the domain of that attribute. Another realization model picks a tuple at random from I and deletes it. De Sa et al. [2019] draw a connection to minimal repairs discussed extensively in Chapter 6 and the most likely repair I^*, and shows that minimal repaired data is the most likely repair only under very strict conditions on the intention and realization models. Figure 7.4 summarizes the aforementioned details of a probabilistic cleaning model.

7.2.2 Machine Learning Frameworks for Data Repair

ML and probabilistic models have been used in multiple data cleaning activities. To recap, a graphical model or probabilistic graphical model (PGM) or structured probabilistic model is a model for which a graph expresses the conditional dependence structure between random variables [Koller and Friedman 2009]. Different variants and extensions of PGMs have been used for different data cleaning problems: conditional random fields (CRF) and Markov logic networks (MLN), which combine first-order logic and PGMs in a single representation, have been used for entity resolution [McCallum and Wellner 2004, Singla and Domingos 2006, Poon and Domingos 2008, Rastogi et al. 2011]; factor graphs have been used for fusing conflicting records from multiple data sources [Rekatsinas et al. 2017b], which not

RIGHTSLINK

only can reason about conflicted values from different sources but also can incorporate domain knowledge into the reasoning process (e.g., a particular data source dated before the year 2000 is inaccurate).

Based on the probabilistic error model we described in Section 7.2.1, we give an example of a statistical inference framework, the HoloClean system, that uses the observed data set to build a probabilistic model capable of predicting the most likely value for cells identified as (possibly) noisy. The HoloClean system also uses factor graphs to encode various kinds of features and inputs to the repair process, such as denial constraints, statistical properties, minimality, and external master data [Rekatsinas et al. 2017a].

Example 7.2 [Rekatsinas et al. 2017a] Consider a dataset from the city of Chicago with information on inspections of food establishments. A snippet is shown in Table 7.1(a). The dataset is populated by transcribing forms filled out by city inspectors, and as such contains multiple errors. Records can contain misspelled entries, report contradicting ZIP codes, and use different names for the same establishment.

The dataset has a set of functional dependencies (see Table 7.1(b)) and an external dictionary of address listings in Chicago (Table 7.1(d)). Co-occurrence statistics can also be obtained by analyzing the original input dataset in Figure 7.1(a).

First, data repairing methods based on integrity constraints, such as Bohannon et al. [2005], assume the majority of input data to be clean and use the principle of *minimality* as an operational principle to perform repairs (cf. Section 6.2.5). Nevertheless, minimal repairs do not necessarily correspond to correct repairs: an example minimal repair is shown in Table 7.1(e). This repair chooses to update the ZIP code of tuple $t1$ so that all functional dependencies in Table 7.1(b) are satisfied. This particular repair introduces an error as the updated zip code is wrong. This approach also fails to repair the ZIP code of tuples $t2$ and $t3$ as well as the "DBAName" and "City" fields of tuple $t4$ since altering those leads to a non-minimal repair.

Second, methods that rely on external data [Fan et al. 2009, Chu et al. 2015] match records of the original dataset to records in the external dictionaries or knowledge bases to detect and repair errors in the former. The matching process is usually described via a collection of matching dependencies (see Table 7.1(c)) between the original dataset and external information. A repair using such methods is shown in Table 7.1(f). This repair fixes most errors, but fails to repair the "DBAName" field of tuple $t4$ as no information for this field is provided in the external data. In general, the quality of repairs performed by methods that use external

RIGHTSLINK

Table 7.1 A variety of signals can be used for data cleaning [Rekatsinas et al. 2017a]

	DBA Name	AKAName	Address	City	State	Zip
t_1	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	**Chicago**	IL	**60608**
t_2	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60609**
t_3	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60609**
t_4	**Johnnyo's**	Johnnyo's	3465 S Morgan St	**Cicago**	IL	60608

Conflicts due to c_2

Does not obey data distribution

Conflict due to c_2

(a) Input database external information (Chicago food insspections)

c_1: DBAName → Zip
c_2: Zip → City, State
c_3: City, State, Address→ Zip

(b) Functional dependencies

m_1: Zip = Ext_Zip → City = Ext_City
m_2: Zip = Ext_Zip → State = Ext_State
m_3: City = Ext_City ∧ State = Ext_State∧
∧ Address = Ext_Address → Zip = Ext_Zip

(c) Matching dependencies

Ext_Address	Ext_City	Ext_State	Ext_Zip
3465 S Morgan St	Chicago	IL	60608
1208 N Wells St	Chicago	IL	60610
259 E. Erie St	Chicago	IL	60611
2806 W Cermak Rd	Chicago	IL	60623

(d) External information (address listings in Chicago)

	DBA Name	AKAName	Address	City	State	Zip
t_1	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60609**
t_2	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60609**
t_3	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60609**
t_4	**Johnnyo's**	Johnnyo's	3465 S Morgan St	**Cicago**	IL	60608

(e) Repair using minimality w.r.t. FDs

	DBA Name	AKAName	Address	City	State	Zip
t_1	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	60608
t_2	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60608**
t_3	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60608**
t_4	**Johnnyo's**	Johnnyo's	3465 S Morgan St	**Chicago**	IL	60608

(f) Repair using matching dependencies

	DBA Name	AKAName	Address	City	State	Zip
t_1	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	60608
t_2	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60609**
t_3	John Veliotis Sr.	Johnnyo's	3465 S Morgan St	Chicago	IL	**60609**
t_4	**John Veliotis Sr.**	Johnnyo's	3465 S Morgan St	**Chicago**	IL	60608

(g) Repair that leverages quantitative statistics

RIGHTSLINK

data can be poor due to the limited coverage of external resources, or these methods may not be applicable as for many domains a knowledge base may not exist.

Finally, data repairing methods that are based on statistical analysis [Mayfield et al. 2010, Yakout et al. 2013] leverage quantitative statistics of the input dataset, e.g., co-occurrences of attribute values, and use those for cleaning. These techniques overlook integrity constraints. Table 7.1(g) shows such a repair. As shown, the "DBAName" and "City" fields of tuple $t4$ are updated as their original values correspond to outliers with respect to other tuples in the dataset. However, this repair does not have sufficient information to fix the zip code of tuples $t2$ and $t3$.

In this example, if the zip code and city repairs from Table 7.1(f) are combined with the DBAName repair from Table 7.1(g), all errors can be repaired in the input dataset. Nonetheless, combining heterogeneous signals can be challenging. This is not only because each type of signal is associated with different operations over the input data (e.g., integrity constraints require reasoning about the satisfiability of constraints while external information requires efficient matching procedures) but different signals may suggest conflicting repairs. For instance, naively combining the repairs in Table 7.1 produces conflicts on the zip code of tuples $t2$ and $t3$. The repairs in Table 7.1(e) and (g) assign value "60609" while the repair in Figure 7.1(f) assigns value "60608".

Instead of considering each signal in isolation, HoloClean uses all available signals to suggest data repairs. HoloClean considers the input dataset as a *noisy version* of a hidden clean dataset and treats each signal as *evidence* on the correctness of different records in that dataset. To combine different signals, HoloClean relies on probability theory to reason about inconsistencies across those signals. HoloClean consists of two main stages: *Compilation* and *Repairing*.

In the compilation stage, HoloClean automatically generates a probabilistic model (a factor graph) [Koller and Friedman 2009] whose random variables capture the uncertainty over cells in the input dataset. Signals are converted to factors and are used to describe the distribution characterizing the input dataset D. Specifically, HoloClean associates each cell $c \in D$ with a random variable T_c that takes values from a finite domain $dom(c)$ and compiles a probabilistic graphical model that describes the distribution of random variables T_c for cells in the dataset D.

A factor graph is a hypergraph (T, F, θ) in which T is a set of nodes that correspond to random variables and F is a set of hyperedges. Each hyperedge $\phi \in F$, where $\phi \subseteq T$, is referred to as a *factor*. Each hyperedge ϕ is associated with a *factor function* and a real-valued weight θ_ϕ and takes an assignment of the random variables in ϕ and returns a value in $\{-1, 1\}$ (i.e., $h_\phi : \mathbb{D}^{|f|} \rightarrow \{-1, 1\}$). Hyperedges f,

RIGHTSLINK

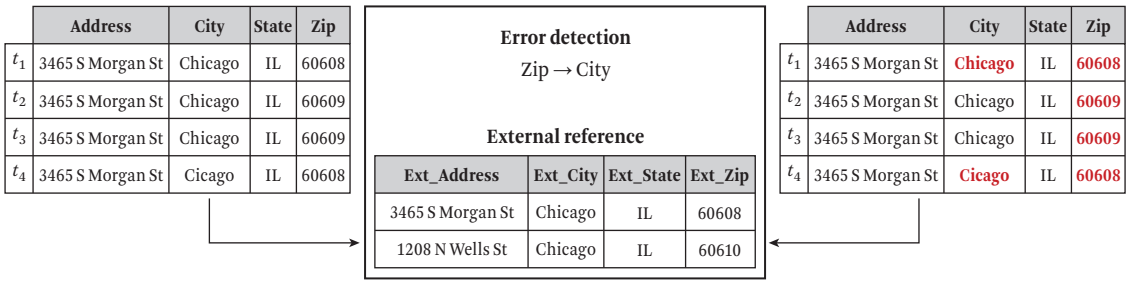


Figure 7.5 Error detection methods are used in HoloClean to identify uncertain cells.

functions h_ϕ, and weights θ_ϕ define a *factorization* of the probability distribution $P(T)$ as:

$$P(T) = \frac{1}{Z} \exp\left(\sum_{\phi \in F} \theta_\phi \cdot h_\phi(\phi)\right), \tag{7.1}$$

where Z is called the *partition function* and corresponds to a normalization constant ensuring we have a valid distribution.

Any error detection methods can be used to identify the uncertain cells. Hence the set of random variables T_c is split into T_c^u and T_c^k, where T_c^u is the set of random variables that correspond to uncertain cells (and hence their values need to be inferred) and T_c^k is the set of random variables that correspond to the correct cells (whose observed value will be used for training). For example, in Figure 7.5, a functional dependency and master data were used to identify that some cities and zip codes in the table might be incorrect. Figure 7.6 shows an example subset of the factors defined on four uncertain cells: $t1.City$, $t4.City$, $t1.Zip$, and $t4.Zip$. Two example types of factors are shown:

Co-occurrence factors capture the importance of co-occurring with the specific address value mentioned in the same tuple. For example, possible values for the random variable $t4.City$ that co-occur with the address "3465 S Morgan St" in the whole relation will be more probable than other values. The factor function (h_1 in the figure) can be computed as the $Pr(t4.City = x | t4.Address =$ "3465 *S Morgan St*"), where x is a possible value for a city in the domain of $t4.City$.

Constraint factors capture the dependency among multiple random variables involved in a violation of a given integrity constraint. In Figure 7.6, a factor

RIGHTSLINK

	Address	City	State	Zip
t_1	3465 S Morgan St	Chicago	IL	60608
t_2	3465 S Morgan St	Chicago	IL	60609
t_3	3465 S Morgan St	Chicago	IL	60609
t_4	3465 S Morgan St	Cicago	IL	60608

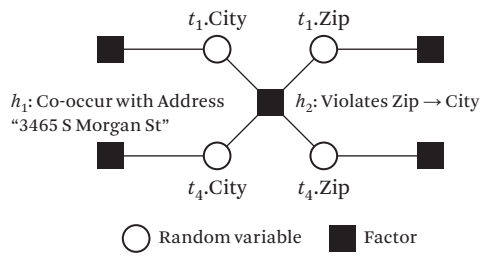


Figure 7.6 HoloClean represent uncertain cells as random variables in a factor graph.

that connects the four random variables will make values that do not violate the FD *Zip* → *City* more probable than other values in the domain of these variables. The factor function in this case (h_2 in the figure) can be simply a binary function that gives 1 if the four values violate the constraint and 0 otherwise.

Once the model is built, the model parameters (factor weights) are learned in a weakly supervised fashion; HoloClean uses the set of cells identified as correct to generate a large number of correct and incorrect example assignments of random variables in T_c^k, where the current observed values correspond to the "correct" examples and multiple generated values from the domain correspond to a set of "incorrect" examples. Let T be the set of all variables T_c. HoloClean uses empirical risk minimization (ERM) over the likelihood $\log P(T)$ to compute the parameters of its probabilistic model. Approximate inference via Gibbs sampling [Zhang and Ré 2014] is used to estimate the value $\hat{v}_c$ of the noisy uncertain cells T_c^u, and each of these cells is assigned to the MAP estimates of variables T_c^u.

Each repair by HoloClean is associated with a calibrated marginal probability. For example, if the proposed repair for a record in the initial dataset has a probability of 0.6, it means that HoloClean is 60% confident about this repair. As a result, these marginal probabilities can be used to solicit user feedback to further improve the repairing accuracy. For example, HoloClean can ask users to verify repairs with low marginal probabilities and use those as labeled examples to retrain the parameters of HoloClean's model using standard incremental learning and inference techniques [Shin et al. 2015].

The HoloClean original paper [Rekatsinas et al. 2017a] describes multiple ways to generate these factor graphs and suggest tools such as DeepDive [Shin et al. 2015] to declaratively represent the random variables involved, and how to compile various signals such as minimality, denial constraints, and co-occurrence statistics as

RIGHTSLINK

factors. The paper also highlights that naively declaring all possible factors will generate a massive graph with exponential number of weights that need to be learned to produce the final model. For example, the constraint factor in Figure 7.6 is the only one of many that correspond to the quadratic number of tuple combinations in the data. Moreover, the size of the weight vector associated with such a factor is exponential in the size of the domains of the involved random variables. Hence, HoloClean employs multiple techniques to allow learning such a model and to achieve scalable inference for cleaning.

Domain Pruning. It is easy to see that many possible values for the random variables in the model can be easily pruned as they have very little chance of being picked as the most likely value. Pruning the domain of random variables simplifies inference by limiting the possible grounding of these variables.

Parameter tying. All factors from the same type are assigned the same weight in the spirit of *template-based graphical models* [Kimmig et al. 2015], and as discussed in De Sa et al. [2019] for learning the probabilistic noisy model. This significantly reduces the number of model parameters that need to be learned, which allows using the available observations to learn the repair model.

Model relaxation. During the inference process, Gibbs sampling may require exponential iterations to converge. However, if a factor graph has only independent random variables then Gibbs sampling requires a polynomial number of samples to mix [Sa et al. 2015]. Motivated by this result, HoloClean introduces an optimization that relaxes the factors used to encode denial constraints to obtain a model with independent random variables. Instead of enforcing denial constraints for any assignment of the random variables corresponding to uncertain cells in D, HoloClean generates features that provide evidence on random variable assignments that lead to constraint violations. This builds upon two assumptions: (i) erroneous cells in D are fewer than correct cells, i.e., there is sufficient redundancy to fix errors in D; and (ii) each integrity constraint violation can be fixed by updating a single cell in the participating tuples. Figure 7.7 shows an example relaxation where a four-variable factor representing the given FD applied on four cells is transformed into four single-variable factors, where in each of the new factors the other three variables are replaced by their observed values.

RIGHTSLINK

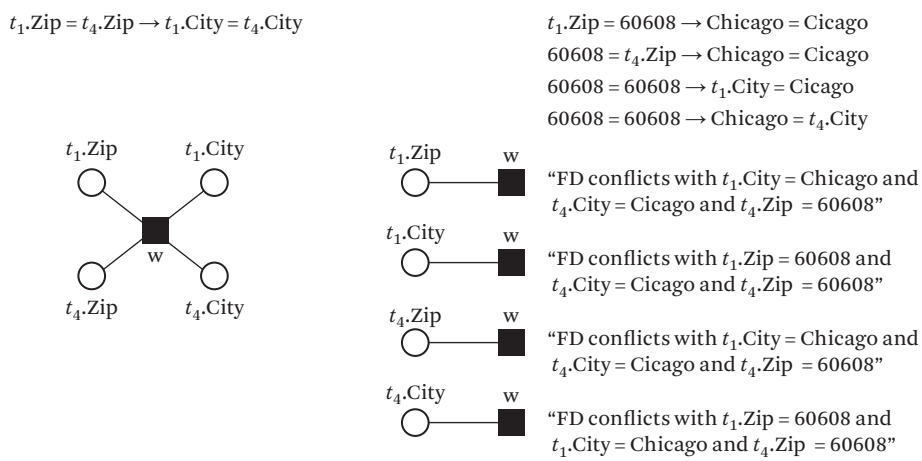


Figure 7.7 HoloClean relaxed model.

The relaxed model comes with two desired properties: (i) the factor graph generated by relaxing the original DDlog rules contains only independent random variables; hence, Gibbs sampling is guaranteed to mix in $o(n \log n)$ steps; and (ii) since random variables are independent, learning the parameters of Equation 7.1 corresponds to a convex optimization problem, which is efficient to solve. HoloClean shows empirically that this model not only leads to more scalable data repairing methods, but achieves the same quality repairs as the non-relaxed model.

7.2.3 Comments on Using Deep Learning in Data Cleaning

Conventional ML models, such as linear regression, decision trees, random forests, and SVMs, are well understood and often require elaborate feature engineering exercise to provision. In the last few years, there has been dramatic interest in "deep learning" (neural networks), which have shown great results in image understanding and natural language processing problems [LeCun et al. 2015, Goodfellow et al. 2016]. However, applying DL in enterprise data cleaning applications remains challenged by the scarcity of training data, since most of these models require large quantities of labeled data to learn the classification task, and is further complicated by the lack of reasonable explanations of the output decisions.

Training Data. In general, generating training data in enterprise data integration tasks is a huge problem. Consider three plausible terms: IBM-SA, IBM

RIGHTSLINK

Inc., and IBM. Intuitively, these three terms might represent the Spanish subsidiary, the U.S. subsidiary, and the overall company, respectively. Deciding whether these are duplicated entities or separate related entities can be performed by a domain expert, and even then, the decision to consolidate or not could be depend on the question being asked. However, automatically "learning" these complex relationships among these entities and the right merge/separate decision will probably require massive labeled data in a deep neural network model.

Domain expert time is a scarce commodity that must be ruthlessly economized and cannot be simply used to label millions of training examples. These experts are typically well-paid, busy business experts who view generating training data as a low priority task. Hence, it is easy to see why ML models with fewer training data requirements are highly preferable to those with large training data demands (e.g., DL). Advances in automating training data collection, such as the Snorkel project [Ratner et al. 2017], might relax this constraint. However, Snorkel still requires users to write many highly overlapping and high-coverage "labeling functions" to produce high-quality labeled data. Until we effectively solve the problem of generating large and high-coverage training data, enterprise solutions will likely depend more on conventional ML classifiers with modest training requirements.

Explainability. In many enterprise integration problems, one must be able to explain why the ML model took a particular action. For example, a predictive model that generates approvals for real estate loans must be explainable: a loan was denied because of such and such reasons. If this explanation is not forthcoming, then lawsuits are inevitable, and adoption of these models is highly unlikely. Conventional ML models are at least marginally explainable, while DL models are not. Again, this will be an impediment to the applicability of DL in enterprise integration applications. It is conceivable that DL models will become more explainable in the future, and there is indeed considerable research effort in this area. However, until the explanability of DL is truly solved, we do not foresee the adoption of these models on a wide scale or as a primary classification method in enterprise data integration tasks.

However, DL outperforms conventional ML models or can be incorporated as part of the cleaning solutions in certain data cleaning scenarios. We discussed in Section 7.1.2 a promising example of using deep learning for the entity resolution problem. It is also natural to use deep learning to capture latent data features (such as language models for text and strings) during error detection and error repair

RIGHTSLINK

ML models, for example as features in the HoloClean framework [Rekatsinas et al. 2017a] discussed in Section 7.2.2.

7.3 Data Cleaning for Analytics and Machine Learning

While ML techniques are useful for designing high-accuracy data cleaning solutions, ensuring high-quality data is also important for downstream data analytics. In this section, we discuss how dirty data and data cleaning affect both SQL analytics and machine learning models.

7.3.1 Cleaning for SQL Aggregate Queries

SampleClean [Wang et al. 2014] targets the problem of answering aggregate queries when the input data is dirty. Since cleaning a large input dirty dataset is usually time consuming and requires human expertise, SampleClean aims at answering a query only by cleaning a sample of the dirty dataset, and at the same time, providing confidence interval guarantees for the query results.

SampleClean addresses aggregate queries of the following form:

```
SELECT F(A) FROM R WHERE PREDICATE GROUP BY GB_ATTRS
```

where F(A) is AVG, SUM, COUNT on attribute a. SampleClean assumes that there are two types of errors in the input R: attribute error and duplication error. A row $r \in R$ is said to have an attribute error if one of the attributes of r is incorrect or has a missing value, and a row $r \in R$ is said to have a duplication error if there exist other records in R that refer to the same entity. For every dirty relation R, SampleClean assumes there is a clean relation R_{clean} where attribute errors are corrected and duplicates are merged into a canonical representation. Furthermore, SampleClean assumes an oracle black-box data cleaning function $C(.)$ which takes a input row r and returns the cleaned version $Correct(r)$, as well as the number of times $Numdup(r)$ the record is duplicated in the entire dataset.

Before we describe how SampleClean provides a confidence interval for aggregate queries, consider a simple problem of estimating the mean value of a set of real numbers R from a sample S. If S is randomly sampled from R, the central limit theorem states that these estimates follow a normal distribution. Because of this, the mean of R can be estimated by $mean(S)$ with a confidence interval $mean(S) \pm \lambda\sqrt{\frac{var(S)}{k}}$ parameterized by λ (e.g., 95% indicates $\lambda = 1.96$), where k is the number of rows in the sample S. To leverage the existing results, SampleClean reformulates the aggregate function $F(A)$ (SUM, COUNT, AVG) on an attribute A of

RIGHTSLINK

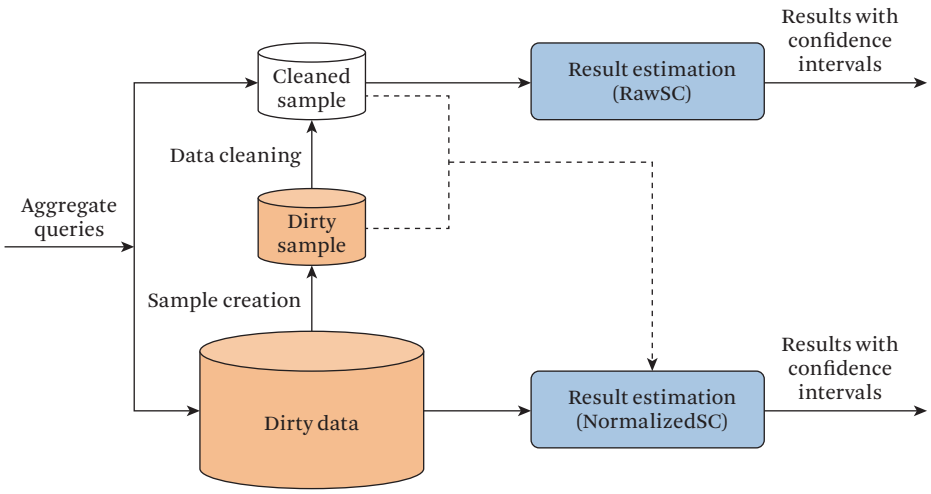


Figure 7.8 SampleClean Framework [Wang et al. 2014].

R as calculating a mean value with $F(S) = \frac{1}{k}\sum_{r \in S} \phi(r)$, where $\phi(r)$ expresses the necessary scaling to translate the query into the mean value calculation:

- COUNT: $\phi(r) = \text{PREDICATE}(r) \cdot N$
- SUM: $\phi(r) = \text{PREDICATE}(r) \cdot N \cdot r[A]$
- AVG: $\phi(r) = \text{PREDICATE}(r) \frac{k}{k_{pred}} \cdot r[A]$,

where PREDICATE(r) returns 1 or 0 depending on whether r satisfies the predicate, and k_{pred} is the number of rows in the sample that satisfies the predicate.

Figure 7.8 illustrates the SampleClean framework. SampleClean first generates a sample of dirty data, and then invokes the oracle data cleaning function to clean the dirty sample. SampleClean then uses the cleaned sample to answer aggregate queries and gives unbiased estimates with confidence intervals, meaning that in expectation the estimates are equal to the query results if the entire dataset was first cleaned and then used to answer the query. SampleClean provides two types of estimates for a query, namely, RawSC and NormalizedSC. RawSC directly estimates the true query result based on the cleaned sample, and NormalizedSC uses the difference between the cleaned sample and the dirty sample to correct the error in a query result over the entire dirty data. To obtain unbiased estimates, SampleClean has to account for duplicates when sampling. For example, consider a query asking for the average number of citations of all papers published every year, grouping by

RIGHTSLINK

year. Assume that the table has duplicates and papers with higher citation counts tend to have more duplicates. The more duplicates a paper has, the more likely that paper is sampled, leading to over-estimated average citation count per year when random sampling is used. Therefore, SampleClean samples a tuple that has more duplicates with a smaller probability. Specifically, a tuple with no duplicates is c times more likely to be included in the sample than a tuple with c duplicates. We now describe RawSC and NormalizedSC in detail.

RawSC runs the query directly on a cleaned sample S drawn from R, which is not equivalent to computing the query result on a sample directly drawn from the clean data R_{clean}. Consider the case where data has duplication errors. Sampling from the dirty data R leads to an overrepresentation of the duplicated rows in the sample. Even if data cleaning is applied on the sample S, the sample S is not uniform. RowRC, thus, defines a new function $\phi_{clean}(r)$, similar to $\phi(r)$, that corrects attribute values and rescales to ensures that the estimate is unbiased. As discussed, SampleClean considers two types of errors.

- Attribute errors only affect individual rows, and thus the probability a given tuple is sampled is not changed by cleaning. Therefore, for attribute errors, $\phi_{clean}(r)$ can be defined as $\phi_{clean}(r) = \phi(Correct(r))$.
- The duplication error, on the other hand, affects the uniform sampling since duplicated rows are more likely to be sampled. This problem is addressed by SampleClean with a weighting scheme. Specifically, if a tuple r is duplicated $m = Numdup(r)$ times, then the tuple should have a factor of $\frac{1}{m}$ weight compared with other tuples in the sample. SampleClean derives the following $\phi_{clean}(r)$ for duplication errors to achieve unbiased estimate: (1) for SUM, COUNT queries, applying $\phi_{clean}(r) = \frac{\phi(r)}{m}$ yields an unbiased estimate; and (2) for AVG query, the result needs to be scaled by the duplication rate $d = \frac{k}{k'}$, where $k' = \sum_i \frac{1}{m_i}$, with m_i denoting the number of duplicates for row r_i in the sample. Applying $\phi_{clean}(r) = d \cdot \frac{\phi(r)}{m}$ yields an unbiased estimate (see Wang et al. [2014] for proof).

Given $\phi_{clean}(r)$, RawSC estimates the aggregate query f in the following way given a sample S of the input R.

1. Apply $\phi_{clean}(r)$ to every row $r \in S$ in the sample and call the resulting set $\phi_{clean}(S)$.
2. Calculate the mean μ_c and the variance δ_c^2 of the set $\phi_{clean}(S)$.
3. Return $\mu_c \pm \lambda\sqrt{\frac{\delta_c^2}{k}}$.

RIGHTSLINK

While RawSC runs the aggregate directly on the sample, NormalizedSC uses the difference between the cleaned sample and the dirty sample to correct the error in a query result over the entire dirty data. The difference is defined as $\epsilon = f(R) - f(R_{clean}) = \frac{1}{N}\sum_{r \in R}(\phi(r) - \phi_{clean}(r))$. In other words, the difference is the average of how much $\phi_{clean}(r)$ changes $\phi(r)$ for every row r. To estimate the difference for a sample S, SampleClean constructs the set of differences between $\phi_{clean}(r)$ and $\phi(r)$ for every row $r \in S$: $Q = \{\phi(r_1) - \phi_{clean}(r_1), \phi(r_2) - \phi_{clean}(r_2), \ldots, \phi(r_k) - \phi_{clean}(r_k)\}$. The mean value of the set Q is an unbiased estimate of ϵ, the difference between $f(R)$ and $f(R_{clean})$. SampleClean subtracts this estimate from an existing aggregation of data to get an estimate of $f(R_{clean})$ as follows.

1. Apply $\phi_{clean}(r)$ to every row $r \in S$ in the sample and calculate the set of differences $Q(S)$.
2. Calculate the mean μ_q and the variance δ_q^2 of the set $Q(S)$.
3. Return $(f(R) - \mu_q) \pm \lambda\sqrt{\frac{\delta_q^2}{k}}$.

Comparing RawSC with NormalizedSC shows that both achieve unbiased estimates. However, we can see that RawSC gives an estimate that is proportional to the variance of the cleaned sample δ_c^2, while NormalizedSC gives an estimate that is proportional to the variance of the differences before and after cleaning δ_q^2. If there are small errors in the sample, S_{clean} will be highly similar to S, and NormalizedSC will give high accuracy estimate. If errors are large but the absolute variances of values in the aggregate attribute are small, RawSC gives a more accurate estimate.

7.3.2 Cleaning for ML Analytics

We discuss the need of handling errors for ML analytics. While it is widely recognized that errors in training data affect model accuracy, the study of how different types of errors affect different ML models and how to handle errors in ML datasets is in its infancy. The following example highlights the possible effects of dirty data on ML models and the challenges of cleaning for ML.

Example 7.3 Consider training a linear regression model on a dataset that has systematic errors, for example, due to measuring equipment malfunctioning. Figure 7.9(a) shows a scenario where one variable (x-axis) is systematically corrupted, leading to a shifted regression model. Training the model on a mix of dirty and clean data is also problematic. Figure 7.9(b) shows a scenario where two of the dirty data points are cleaned, leading to a learned model completely different from the correct model. This is a well-known phenomenon called Simpson's Paradox [Simpson

RIGHTSLINK

Figure 7.9 Systematic biases in training data lead to inaccurate models [Krishnan et al. 2016].

1951]. Training the model only a sample of the cleaned data, similar to SampleClean, can also result in incorrect models, as shown in Figure 7.9(c).

We classify existing approaches for dealing with errors in the training data into three types and postulate that future approaches would also fit into these categories: (1) by using an error-robust ML model; (2) by cleaning errors in the training data; and (3) by modeling the errors in the model learning process.

Error-Robust ML Models. This type of approach relies on ML algorithms that are naturally robust or not too sensitive to errors in the training data. Indeed, studies have shown that different ML algorithms are influenced by errors in different ways [Nettleton et al. 2010, Zhu and Wu 2004]. For example, AdaBoost [Freund and Schapire 1997] is known to be very sensitive to erroneous labels, as it tries to fit noisy instances by increasing their weights in training; decision trees are known to be able to handle missing values natively, while linear models cannot; and least square regression is highly sensitive to outliers. In essence, for approaches in this category, error handling is by done by choosing the appropriate ML models that are known to be more robust to the errors at hand.

Cleaning for Learning. This category of approaches detects and fixes errors before or during the learning process. Techniques in this category can further be divided into model-agnostic cleaning and model-specific cleaning. All data cleaning algorithms discussed in previous chapters can be considered as model-agnostic cleaning methods, as the cleaning solutions are not based on the downstream ML models. The advantage of model-agnostic cleaning methods is that the cleaning is done only once and can potentially benefit multiple ML models. However, as we have shown, cleaning itself is an iterative and expensive process, and it is hard to assess whether a dataset is 100% clean without ground truth. In contrast, model-

RIGHTSLINK

specific cleaning methods save cleaning efforts by focusing on cleaning parts of the data so as to maximize gains on specific models. The downside for model-specific cleaning is that the cleaning efforts may not be reusable for a different ML model. In the following, we describe ActiveClean [Krishnan et al. 2016], a model-specific cleaning method.

ActiveClean [Krishnan et al. 2016] is an example of cleaning data intelligently for convex models that are trained using gradient descent methods. Formally, the class of predictive modeling problem ActiveClean addresses is as follows: Given a set of labeled training examples $\{(x_i, y_i)\}_{i=1}^{N}$, the training problem is to find a vector of *model parameters* θ by minimizing a loss function ϕ over all training examples:

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \phi(x_i, y_i, \theta),$$

where ϕ is a convex function in θ. Convexity is a required property to ensure that the iterative optimization converges to a true global optimum, and ActiveClean applies convergence arguments from convex optimization theory to show that ActiveClean converges. Without loss of generality, all regularizations are treated as part of the loss function. Given a dirty relation R, where every record $r \in R$ can be mapped into a feature vector x and a label y, a convex loss function ϕ, and a black-box cleaner $C(\cdot)$ which takes a dirty row r and returns a cleaned version $C(r)$, ActiveClean returns a *reliable* estimate $\hat{\theta}$ given a limit k on the number of invocations of $C(\cdot)$. A reliable estimate, in this context, means that the expected error in the estimate $\hat{\theta}$ is bounded by a monotonically decreasing function in k.

The key insight of ActiveClean is that convex loss models (e.g., linear regression) can be trained and cleaned simultaneously. Mini-batch stochastic gradient descent (SGD) is an algorithm for finding the optimal θ, which starts with an initial $\theta^{(0)}$ and iteratively $\theta^{(t+1)}$ with $\theta^{(t)}$ and gradient taken at $\theta^{(t)}$. Specifically, in each iteration of mini-batch SGD, a random subset of R is selected and the average gradient is computed. $\theta^{(t+1)}$ is then updated using $\theta^{(t+1)} = \theta^{(t)} - \lambda \nabla\phi(\theta^{(t)})$, where λ is the learning rate and $\nabla\phi(\theta^{(t)}) = \frac{1}{|S|}\sum_{i=i}^{|S|} \nabla\phi(x_i, y_i, \theta^t)$ is the average gradient for the batch S with respect to the dirty data. ActiveClean modifies the mini-batch SGD in the following way: instead of calculating average gradient with respect to the dirty data, ActiveClean calculates the average gradient with respect to the cleaned data in the batch, namely, $\nabla\phi(\theta^{(t)}) = \frac{1}{|S|}\sum_{i=i}^{|S|} \nabla\phi(C(x_i), C(y_i), \theta^t)$. The iterative process is guaranteed to converge if every record has a non-zero probability of being sampled in a batch, which follows from the convergence properties of mini-batch SGD algorithm.

RIGHTSLINK

Modeling Errors in the Learning Process. This category of techniques involves actually adjusting the learning algorithms to model the errors in the training data. The difficulties in these approaches involve how to model the errors in the training data, as they are often not known in advance. In the following, we use Snorkel [Ratner et al. 2017] as an example of how the errors in the training data are modeled as probabilistic training data, and how the learning algorithm takes the probabilistic labels into training.

Snorkel is developed to generate labeled training data for learning a supervised classification model h_θ which, given a data point $x \in \mathcal{X}$, predicates a label $y \in \mathcal{Y}$. Snorkel targets settings when there is not enough training data for learning a good model and generates training labels by soliciting a set of labeling functions from users. A labeling function $\lambda : \mathcal{X} \rightarrow \mathcal{Y} \cup \{\emptyset\}$ takes as input a unlabeled data point and outputs a label, where $\emptyset$ denotes that the labeling function abstains (i.e., cannot provide a label for a data point). Users can write many labeling functions according to their domain knowledge, and each labeling function provides a weak signal to the true label of a data point. Given m unlabeled points and n labeling functions, Snorkel applies all labeling functions to all points and produces a $m \times n$ label matrix Λ, where $\Lambda_{ij} \in \mathcal{Y} \cup \{\emptyset\}$. Given Λ, Snorkel denoises these labeling functions to produce *probabilistic training labels* $\tilde{Y} = (\tilde{y_1}, \ldots, \tilde{y_m})$, where $\tilde{y_i} \in [0, 1]$. The probabilistic training labels are then used to train a supervised classification model. We now describe how Λ is used to obtain $\tilde{Y}$ and how $\tilde{Y}$ is used for model training.

Given Λ, Snorkel aims at modeling and integrating the noisy signals provided by n labeling functions and produces the probabilistic labels $\tilde{Y}$. Essentially, Snorkel uses the agreements and disagreements of the labeling functions on different data points to learn the accuracy and dependencies of labeling functions via factor graph models [Ratner et al. 2016]. There are three factors used: the labeling propensity, accuracy, and pairwise correlations of labeling functions:

$$\phi_{i,j}^{Lab} = \mathbb{1}\{\Lambda_{ij} \neq \emptyset\}$$

$$\phi_{i,j}^{Acc} = \mathbb{1}\{\Lambda_{ij} = y_i\}$$

$$\phi_{i,j,k}^{Corr} = \mathbb{1}\{\Lambda_{ij} = \Lambda_{ik}\}, (j, k) \in C.$$

For a given point x_i, let $\phi_i(\Lambda, y_i)$ is used as the concatenated vector of these factors for all labeling functions, and let $w \in R^{2n+|C|}$ be the vector of parameters. This defines the probabilistic model used in Snorkel:

RIGHTSLINK

$$p_w(\Lambda, Y) = \frac{1}{Z_w} \exp\left(\sum_{i=1}^{m} w^T \phi_i(\Lambda, y_i)\right).$$

To learn this model, i.e., w, Snorkel minimizes the negative log marginal likelihood given the observed label matrix Λ:

$$\hat{w} = \arg\min_{w} -\log \sum_{Y} p_w(\Lambda, Y).$$

The probabilistic training labels are then defined as $\hat{Y} = p_{\hat{w}}(Y|\Lambda)$.

Given $\tilde{Y}$, Snorkel simply trains a supervised model h_θ by minimizing a *noise-aware* variant of the loss function $l(h_\theta(x_i), y)$, i.e., the expected loss with respect to $\tilde{Y}$:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{m} \mathbb{E}_{y\sim Y}[l(h_\theta(x_i), y)].$$

Snorkel shows that as the amount of unlabeled data increases, the generalization error of the supervised model will decrease at the same asymptotic rate as traditional supervised learning models do with additional hand-labeled data [Ratner et al. 2016].

7.3.3 Conclusion

In this chapter, we explored the synergies between machine learning and data cleaning, in terms of both how ML can help with data cleaning activities and how data cleaning is important for data analytics, including ML.

ML techniques hold great promise for developing better data cleaning solutions. We discussed how ML is used for data deduplication, a data cleaning problem that is a natural fit for applying a binary classifier. We showed how to use leverage active learning to judiciously solicit informative training examples, and also how recent developments in deep learning (e.g., word embeddings) can increase classifier accuracies in certain cases. We then introduced a probabilistic unclean database model, and presented HoloClean as an example of using that model for data repair. HoloClean [Rekatsinas et al. 2017a] uses probabilistic graphical models to accumulate various signals to determine the most likely repair for an erroneous cell.

Ensuring high quality data, for example, through data cleaning is important for downstream data analytics as well. We first discussed SampleClean [Wang et al. 2014] which targets SQL aggregate queries. It shows how cleaning only a sample of the dirty data is sufficient to provide confidence intervals on answers to aggregate queries. We then surveyed different strategies for handling errors in ML

RIGHTSLINK

analytics, including using error-robust ML models, cleaning errors before training, and modeling errors in the learning process.

We believe that the recent rapid developments in both ML and data cleaning will spark promising research at the intersection of these two fields. This chapter provides a starting point for readers interested in this direction, and we hope to see many more ML-inspired data cleaning solutions and model-specific cleaning solutions in the near future.

RIGHTSLINK

8 Conclusion and Future Thoughts

Data cleaning is a complex process; we shed some light on some of the foundational aspects of data cleaning efforts. In particular, we focus on some of the most commonly encountered topics, namely, outlier detection, data dedupilcation, data transformation, rule-based data cleaning, and ML-guided data cleaning.

Outlier detection. Outlier detection has many important applications, including credit card fraud detection and network intrusion detection. We introduced a taxonomy for outlier detection techniques that includes three broad categories: statistics-based techniques, distance-based techniques, and model-based techniques. We discussed example techniques in each category and the pros and cons of each category of techniques. In addition, we showed how contextual outlier detection techniques and subspace outlier detection techniques can be used to address the "curse of dimensionality" in high-dimensional outlier detection.

Data deduplication. Data deduplication is perhaps the most well-studied problem in data cleaning. Due to the amount of work in this area, we discussed different aspects of designing a data deduplication workflow, including similarity metrics, classifiers used to determine duplicates, clustering algorithms used to identify clusters of records referring to the same real world entity, different fusion strategies to consolidate multiple records into a canonical representation, blocking strategies to reduce the number of comparisons by avoiding comparing record pairs that are unlikely to be matches, distribution strategies to scale out the data deduplication process, and human involvement in the process. We also highlighted some open-source and commercial data deduplication tools.

RIGHTSLINK

Data transformation. Data transformation has many uses in different stages of an ETL process. We categorized data transformation tasks into syntactic transformations and semantic transformations; their main differences are whether external data sources are needed to perform the transformation. We identified three major components of a syntactic transformation tool: language, authoring, and execution. We used different syntactic transformation tools to illustrate these three dimensions. For semantic transformation, we discussed example-driven techniques as well as data exchange, which can be seen as a form of semantic transformation.

Rule-based data cleaning. Rule-based data cleaning techniques detect and repair errors based on data quality rules, which are often expressed as certain forms of integrity constraints. Since manually designing data quality rules requires human expertise and is often time consuming, we discussed techniques for automatically discovering data quality rules expressed in a variety of languages. We discussed three main challenges associated with detecting violations with respect to a set of rules. We also presented a taxonomy to classify different error repair techniques based on what to repair, how to repair, and where to repair.

ML-guided data cleaning. ML-guided data cleaning techniques leverage ML to detect and repair errors. In particular, we discussed in detail how active learning can be used to solicit training examples from users that are most beneficial to train a classifier for predicting duplicate record pairs. We also showed that probabilistic graphical models can be used to holistically reason about various kinds of signals, which together determine the probabilities of different fixes.

Despite the advances in data cleaning research, academic solutions are still rarely adopted in practice. First, while holistic data cleaning proposals have started to appear, most current data cleaning solutions still consider one type of error at a time. They are generally not adequate to deal with real-world scenarios, where errors interact with each other in complex ways. Second, most of the techniques do not scale well. They either have quadratic complexity or require multiple passes on the whole dataset. Third, most of the parameters of these tools are hard to configure or are non-existent. For example, rule-based cleaning requires a rich set of ICs that is expensive to obtain. Fourth, there is usually a space decoupling between where errors are generated and where errors are detected. While errors are easily spotted downstream near final reports, data sources actually need fixing, which means that

RIGHTSLINK

practical solutions need to maintain and navigate complex provenance information across heterogeneous systems. Finally, human involvement is expensive. However, adding humans in the loop is still a necessary precondition for adopting data cleaning solutions to tune, verify, and approve various automatic decisions.

These challenges—and many other pragmatic reasons—explain the dearth of deployable and industry-strength data cleaning solutions. The gap between data cleaning solutions used in large enterprises and what the scientific community publishes in terms of assumptions, setup, and applicability expands daily. Based on our experience with real data cleaning customers, the real competitor of academic proposals are millions of lines of code in scripts and case-specific wrangling rules written by IT departments struggling to solve their business problems. The good news is that enterprises are starting to appreciate the need for scalable and principled cleaning solutions as opposed to their home-grown rule-based systems, which are starting to be a major technical debt and a burden to maintain. The scientific community is also getting better at understanding real data cleaning challenges (including the engineering and pragmatic ones) because of the recent work of companies like Alation,[1] Tamr,[2] and Trifacta[3] and open-source tools like Magellan [Konda et al. 2016]. They are used by a number of businesses and enterprises. We envision multiple lines of future research directions that both push the frontiers of data cleaning research and make the solutions more practical in industry.

Error detection. While we have discussed several ways to detect errors in the data, many data errors may still remain undetected. One possible line of research in this direction is to devise more expressive integrity constraint languages that allow data owners to easily specify data quality rules. Another direction is to leverage many error detection techniques at the same time for detecting errors (even non-deterministic errors) and reasoning about them in a holistic way.

Master data curation. To perform reliable data repair, master data often needs to be referenced, for example in performing semantic data transformations and in performing trustworthy data repairing in rule-based data cleaning. Relevant master data can also greatly reduce human involvement. However, existing master data sources, such as knowledge bases, often cannot provide a comprehensive coverage for the data to be repaired. Automatic creation and

1. https://alation.com
2. https://www.tamr.com
3. https://www.trifacta.com

RIGHTSLINK

maintenance of relevant master and authoritative data catalogs are essential tasks in enabling high-quality repairs.

Human-involved data repairing. Although much research has been done on involving humans to perform data deduplication, involving humans intelligently in other data cleaning tasks, such as repairing integrity constraint violations, is yet to be explored.

Scalability. Large volumes of data render most current techniques unusable in real settings. The obvious trade-off between accuracy and performance has to be taken more seriously in designing the next generation cleaning algorithms that take time and space budgets into account. Example tools include sampling and approximate cleaning algorithms, with clear approximation semantics that can be leveraged by analytics applications.

Privacy and anonymization. In the process of integrating and repairing data, input from multiple parties often has to be combined or shared, which could be problematic if data to be integrated or cleaned is private or sensitive, such as medical history or private customer databases. The obvious research question is: "How can existing data cleaning solutions be adapted or modified to preserve privacy?"

Semi-structured and unstructured data. A significant portion of data is residing in semi-structured formats, such as JSON, and unstructured formats, such as text documents. Data quality problems for semi-structured and unstructured data largely remain unexplored. There are many possible routes to take, given the variety of techniques already developed for cleaning structured data (e.g., can we design rules that enforce certain structures of a JSON document or a graph dataset?).

RIGHTSLINK

References

Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. 2016a. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12): 993–1004. DOI: 10.14778/2994509.2994518. 196

Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. 2016b. Dataxformer: A robust transformation discovery system. In *Proceedings of the 32nd International Conference on Data Engineering*, pp. 1134–1145. DOI: 10.1109/ICDE.2016.7498319. 91, 107, 108, 110, 111, 114

S. Abiteboul, R. Hull, and V. Vianu. 1995. *Foundations of Databases*. Addison-Wesley. 121, 138, 139, 148

D. Achlioptas. 2001. Database-friendly random projections. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pp. 274–281. ACM. DOI: 10.1.1.28.6652. 33

F. Afrati and P. G. Kolaitis. 2008. Answering aggregate queries in data exchange. In *Proceedings of the 27th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 129–138. ACM. DOI: 10.1145/1376916.1376936. 117

F. N. Afrati and P. G. Kolaitis. 2009. Repair checking in inconsistent databases: algorithms and complexity. In *Proceedings of the 12th International Conference on Database Theory*, pp. 31–41. DOI: 10.1145/1514894.1514899. 178

F. N. Afrati and J. D. Ullman. 2010. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*, pp. 99–110. DOI: 10.1145/1739041.1739056. 68, 69

C. C. Aggarwal. 2013. *Outlier Analysis*. Springer, 2013. 2, 12, 25, 35, 36, 44

C. C Aggarwal and P. S. Yu. 2001. Outlier detection for high dimensional data. In *ACM SIGMOD Rec.*, 30, pp. 37–46. ACM, 2001. DOI: 10.1145/376284.375668. 33, 37, 40

R. Agrawal and R. Srikant. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, 1215, pp. 487–499. 42

R. Agrawal, T. Imieliński, and A. Swami. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD Rec.*, 22, pp. 207–216. DOI: 10.1145/170036.170072. 136

RIGHTSLINK

N. Ailon, M. Charikar, and A. Newman. 2008. Aggregating inconsistent information: ranking and clustering. *J. ACM*, 55(5): 23. 59

R. Ananthakrishna, S. Chaudhuri, and V. Ganti. 2002. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 586–597, 2002. DOI: 10.1016/B978-155860869-6/50058-5. 66

Fabrizio Angiulli, Fabio Fassetti, and Luigi Palopoli. 2009. Detecting outlying properties of exceptional objects. *ACM Trans. Database Syst.*, 34(1): 7. DOI: 10.1145/1508857.1508864. 33, 40, 43

A. Arasu, S. Chaudhuri, and R. Kaushik. 2009. Learning string transformations from examples. *Proceedings of the VLDB Endowment*, 2(1): 514–525. DOI: 10.14778/1687627.1687686. 107

A. Arasu, M. Götz, and R. Kaushik. 2010. On active learning of record matching packages. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 783–794. DOI: 10.1145/1807167.1807252. 197

M. Arenas, L. Bertossi, and J. Chomicki. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 68–79. DOI: 10.1145/303976.303983. 179, 191

M. Arenas, P. Barceló, L. Libkin, and F. Murlak. 2014. *Foundations of Data Exchange*. Cambridge University Press. 114, 117

I. Assent, R. Krieger, E. Müller, and T. Seidl. 2007. Dusc: Dimensionality unbiased subspace clustering. In *Proceedings of the 2007 IEEE International Conference on Data Mining*, pp. 409–414. IEEE. DOI: 10.1109/ICDM.2007.49. 39

I. Assent, R. Krieger, E. Müller, and T. Seidl. 2008. Edsc: efficient density-based subspace clustering. In *Proceedings of the 17th ACM International Conference on Information and Knowledge Management*, pp. 1093–1102. ACM. DOI: 10.1145/1458082.1458227. 39

P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. 2017. Macrobase: Prioritizing attention in fast data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 541–556. 5

G. V. Bard. 2007. Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric. In *Proceedings of the Fifth Australasian Symposium on ACSW Frontiers*, pp. 117–124. 50

V. Barnett and T. Lewis. 1994. *Outliers in Statistical Data*. Wiley, New York. 4, 11, 15, 19, 44, 195

M. Baudinet, J. Chomicki, and P. Wolper. 1999. Constraint-generating dependencies. *J. Comp. and System Sci.*, 59(1): 94–115. DOI: 10.1006/jcss.1999.1632. 7

S. D. Bay and M. Schwabacher. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 29–38. DOI: 10.1145/956750.956758. 28

RIGHTSLINK

R. J. Bayardo, Y. Ma, and R. Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th International World Wide Web Conference*, pp. 131–140. ACM. DOI: 10.1145/1242572.1242591. 64

P. Beame, P. Koutris, and D. Suciu. 2014. Skew in parallel query processing. In *Proceedings of the 33rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 212–223. DOI: 10.1145/2594538.2594558. 67, 69

L. E. Bertossi. 2011. *Database Repairing and Consistent Query Answering.* Morgan & Claypool Publishers. 2, 178, 179, 191

G. Beskales, M. A. Soliman, I. F. Ilyas, and S. Ben-David. 2009. Modeling and querying possible repairs in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1): 598–609. DOI: 10.14778/1687627.1687695. 76, 78, 191

G. Beskales, I. F. Ilyas, and L. Golab. 2010. Sampling the repairs of functional dependency violations under hard constraints. *Proceedings of the VLDB Endowment*, 3(1–2): 197–207. DOI: 10.14778/1920841.1920870. 163, 164, 181, 191, 193

G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. 2013. On the relative trust between inconsistent data and inaccurate constraints. In *Proceedings of the 29th International Conference on Data Engineering*, pp. 541–552. DOI: 10.1109/ICDE.2013.6544854. 164, 173, 174, 176

G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. 2014. Sampling from repairs of conditional functional dependency violations. *VLDB J.*, 23(1): 103–128. DOI: 10.1007/s00778-013-0316-z. 164, 180, 191, 193

G. Beskales, i. F. Ilyas, and L. Golab. 2010. Sampling the repairs of functional dependency violations under hard constraints. *Proceedings VLDB Endowment*, 3(1–2): 197–207. DOI: 10.14778/1920841.1920870.

K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. 1999. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, pp. 217–235. Springer. DOI: 10.1007/3-540-49257-7_15. 12, 32

M. Bilenko and R. J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 39–48. ACM. DOI: 10.1145/956750.956759. 55

M. Bilenko, B. Kamath, and R. J. Mooney. 2006. Adaptive blocking: Learning to scale up record linkage. In *Proceedings of the 2006 IEEE International Conference on Data Mining*, pp. 87–96. DOI: 10.1109/ICDM.2006.13. 66

C. M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag. 136

A. F. Blackwell. 2001. SWYN: A Visual representation for regular expressions. In H. Lieberman, editor, *Your Wish is My Command*, pp. 245–270. Morgan Kaufmann Publishers Inc. 102

RIGHTSLINK

T. Bleifuß, S. Kruse, and F. Naumann. 2017. Efficient denial constraint discovery with Hydra. *Proceedings of the VLDB Endowment*, 11(3): 311–323. DOI: 10.14778/3157794.3157800. 134, 137, 138, 148

J. Bleiholder and F. Naumann. 2008. Data fusion. *ACM Computing Surveys*, 41(1). 74, 75

P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 143–154. ACM. DOI: 10.1145/1066157.1066175. 139, 163, 164, 175, 178, 179, 183, 184, 206

P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering*, pp. 746–755. DOI: 10.1109/ICDE.2007.367920. 130

M. M. Breunig, H.-P. Kriegel, R. T Ng, and J. Sander. 2000. LOF: identifying density-based local outliers. In *ACM SIGMOD Rec.*, 29, pp. 93–104. ACM. DOI: 10.1145/335191.335388. 14, 26, 27, 28, 30

A. Z. Broder. 1997. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pp. 21–29. IEEE. DOI: 10.1.1.24.779. 62

A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. 2014. Descriptive and prescriptive data cleaning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 445–456. DOI: 10.1145/2588555.2610520. 151, 153, 157, 158, 159

V. Chandola, A. Banerjee, and V. Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3): 15. DOI: 10.1145/1541880.1541882. 2, 12, 31, 44

M. S. Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pp. 380–388. ACM. DOI: 10.1145/509907.509965. 62

S. Chaudhuri, V. Ganti, and R. Motwani. 2005. Robust identification of fuzzy duplicates. In *Proceedings of the 21st International Conference on Data Engineering*, pp. 865–876. DOI: 10.1109/ICDE.2005.125. 55

S. Chaudhuri, V. Ganti, and R. Kaushik. 2006. A primitive operator for similarity joins in data cleaning. In *Proceedings of the 22nd International Conference on Data Engineering*, pp. 5–16. IEEE. DOI: 10.1109/ICDE.2006.9. 64

S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. 2007. Example-driven design of efficient record matching queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pp. 327–338. 55

F. Chiang and R. J. Miller. 2008. Discovering data quality rules. *Proceedings of the VLDB Endowment*, 1(1): 1166–1177. DOI: 10.14778/1453856.1453980. 7, 132, 141

F. Chiang and R. J. Miller. 2011. A unified model for data and constraint repair. In *Proceedings of the 27th International Conference on Data Engineering*, pp. 446–457. DOI: 10.1109/ICDE.2011.5767833. 164, 173, 174, 176

J. Chomicki and J. Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1): 90–121. DOI: 10.1016/j.ic.2004.04.007. 178

RIGHTSLINK

P. Christen. 2008. Febrl: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1065–1068. ACM. DOI: 10.1145/1401890.1402020 . 85

P. Christen. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9): 1537–1555. DOI: 10.1109/TKDE.2011.127. 61, 66

X. Chu, I. F. Ilyas, and P. Papotti. 2013a. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13): 1498–1509. DOI: 10.14778/2536258.2536262. 7, 124, 134, 148

X. Chu, I. F. Ilyas, and P. Papotti. 2013b. Holistic data cleaning: Putting violations into context. In *Proceedings of the 29th International Conference on Data Engineering*, pp. 458–469. DOI: 10.1109/ICDE.2013.6544847. 7, 150, 151, 163, 164, 166, 179, 183

X. Chu, I. F. Ilyas, P. Papotti, and Y. Ye. 2014. Ruleminer: Data quality rules discovery. In *Proceedings of the 30th International Conference on Data Engineering*, pp. 1222–1225. DOI: 10.1109/ICDE.2014.6816746. 134

X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. 2015. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1247–1261. DOI: 10.1145/2723372.2749431. 164, 184, 187, 188, 206

X. Chu, I. F. Ilyas, and P. Koutris. 2016. Distributed data deduplication. *Proceedings of the VLDB Endowment*, 9(11): 864–875. DOI: 10.14778/2983200.2983203. 67, 71, 72

G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. 2007. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pp. 315–326. 163, 164, 166, 179, 185

J. P. Cunningham and Z. Ghahramani. 2015. Linear dimensionality reduction: survey, insights, and generalizations. *Journal of Machine Learning Research*, 16(1): 2859–2900. 32

A. Cypher and D. C. Halbert. 1993. *Watch What I do: Programming by Demonstration*. MIT Press. 102

M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. 2013. Nadeef: a commodity data cleaning system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 541–552, 2013. DOI: 10.1145/2463676.2465327. 164, 166

T. Dasu and T. Johnson. 2003. *Exploratory Data Mining and Data Cleaning*. J. Wiley & Sons, Inc. 2

M. Davy and S. Godsill. 2002. Detection of abrupt spectral changes using support vector machines an application to audio signal segmentation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pp. II–1313. IEEE. DOI: 10.1109/ICASSP.2002.5744044. 32

RIGHTSLINK

F. De Marchi, S. Lopes, and J.-M. Petit. 2009. Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems*, 32(1): 53–73. DOI: 10.1007/s10844-007-0048-x. 139

C. De Sa, I. F. Ilyas, B. Kimelfeld, C. Ré, and T. Rekatsinas. 2019. A formal framework for probabilistic unclean databases. *In Proceedings of the 22nd International Conference on Database Theory*. 8, 203, 204, 205, 211

C. De Stefano, C. Sansone, and M. Vento. 2000. To reject or not to reject: that is the question—an answer in case of neural classifiers. *IEEE Transactions on Systems, Man, and Cybernetics*, 30(1): 84–94. DOI: 10.1109/5326.827457. 31, 32

J. Dean and S. Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1): 107–113. DOI: 10.1145/1327452.1327492. 67

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38. 56

D. J DeWitt, J. F Naughton, D. A. Schneider, and S. Seshadri. 1992. Practical skew handling in parallel joins. In *Proceedings of the 18th International Conference on Very Large Data Bases*, pp. 27–40, 1992. 67

T. Diallo, J.-M. Petit, and S. Servigne. 2012. Discovering editing rules for data cleaning. In *10th International Workshop on Quality in Databases*, pp. 1–8. 144

H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2): 1542–1552. DOI: 10.14778/1454159.1454226. 32

A. Doan, Y. Lu, Y. Lee, and J. Han. 2003. Profile-based object matching for information integration. *IEEE Intelligent Systems*, 18(5): 54–59. DOI: 10.1109/MIS.2003.1234770. 55

X. L. Dong and F. Naumann. 2009. Data fusion: resolving data conflicts for integration. *Proceedings of the VLDB Endowment*, 2(2): 1654–1655. DOI: 10.14778/1687553.1687620. 2, 48, 74, 75, 79

X. L. Dong and D. Srivastava. 2015. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool. DOI: 10.2200/S00578ED1V01Y201404DTM040. 79

X. L. Dong, L. Berti-Équille, and D. Srivastava. 2009a. Integrating conflicting data: The role of source dependence. *Proceedings of the VLDB Endowment*, 2(1): 550–561. DOI: 10.14778/1687627.1687690. 79, 80

X. L. Dong, L. Berti-Équille, and D. Srivastava. 2009b. Truth discovery and copying detection in a dynamic world. *Proceedings of the VLDB Endowment*, 2(1): 562–573. DOI: 10.14778/1687627.1687691. 79, 81

A. K Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1): 1–16. DOI: 10.1109/TKDE.2007.250581. 2, 48

RIGHTSLINK

M. Elsner and E. Charniak. 2008. You talking to me? A corpus and algorithm for conversation disentanglement. In *Proceedings of the 46th Annual Meeting Association for Computational Linguistics*, pp. 834–842. 60

M. Elsner and W. Schudy. 2009. Bounding and comparing methods for correlation clustering beyond ILP. In *Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*, pp. 19–27. 6, 58, 59

R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. 2005a. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1): 89–124. DOI: 10.1007/3-540-36285-1_14. 108, 114, 116, 117

R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. 2005b. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems*, 30(4): 994–1055. DOI: 10.1145/1114244.1114249. 117

R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. 2009. Clio: Schema mapping creation and data exchange. In *Proceedings of the 28th International Conference on Conceptual Modeling*, pp. 198–236. Springer. DOI: 10.1007/978-3-642-02463-4_12. 117

G. Fan, W. Fan, and F. Geerts. 2014a. Detecting errors in numeric attributes. In *Proceedings of the 15th International Conference on Web-Age Information Management*, pp. 125–137. Springer. DOI: 10.1007/978-3-319-08010-9_15. 139, 141, 142

W. Fan, M. Miller, S. Stolfo, W. Lee, and P. Chan. 2004. Using artificial anomalies to detect unknown and known network intrusions. *Knowledge and Information Systems*, 6(5): 507–527. DOI: 10.1007/s10115-003-0132-7. 32

W. Fan and F. Geerts. 2012. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. 2, 7, 178

W. Fan, X. Jia, J. Li, and S. Ma. 2009. Reasoning about record matching rules. *Proceedings of the VLDB Endowment*, 2(1): 407–418. DOI: 10.14778/1687627.1687674. 138, 140, 206

W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. 2010. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment*, 3(1–2): 173–184. 139, 143, 163, 164, 184, 186, 187

W. Fan, F. Geerts, J. Li, and M. Xiong. 2011a Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering*, 23(5): 683–698. DOI: 10.1109/TKDE.2010.154. 132

W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. 2011b. Interaction between record matching and data repairing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 469–480. 164, 166, 170, 171

W. Fan, F. Geerts, N. Tang, and W. Yu. 2014b. Conflict resolution with data currency and consistency. *Journal of Data and Information Quality*, 5(1–2): 6:1–6:37. DOI: 10.1145/2631923. 166

I. P. Fellegi and A. B. Sunter. 1969. A theory for record linkage. *J. American Statistical Association*, 64(328): 1183–1210. DOI: 10.1080/01621459.1969.10501049. 2, 55, 56, 195

RIGHTSLINK

P. A. Flach and I. Savnik. 1999. Database dependency discovery: A machine learning approach. *AI Commun.*, 12(3): 139–160. 129

I. K. Fodor. 2002. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Lab., CA (US). 32

Y. Freund and R. E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1): 119–139. DOI: 10.1006/jcss.1997.1504. 218

Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. 1997. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2–3): 133–168. DOI: 10.1023/A:1007330508534. 199

J. Friedman, T. Hastie, and R. Tibshirani. 2001. *The Elements of Statistical Learning*, volume 1. Springer. 56

T. N. Gadd. 1990. Phoenix: the algorithm. *Program: Automated Library and Information Systems*, 24(4): 363–369. DOI: 10.1108/eb047069. 54

H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. 2001. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 371–380. 85

H. Galhardas, A. Lopes, and E. Santos. 2011. Support for user involvement in data cleaning. In *Proceedings of the 13th International Conference Data Warehousing and Knowledge Discovery*, pp. 136–151. DOI: 10.1007/978-3-642-23544-3_11. 85

E. Gan and P. Bailis. 2017. Scalable kernel density classification via threshold-based pruning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 945–959. ACM. DOI: 10.1145/3035918.3064035. 25

V. Ganti and A.-D. Sarma. 2013. Data cleaning: A practical perspective. *Synthesis Lectures on Data Management*, 5(3): 1–85. DOI: 10.2200/S00523ED1V01Y201307DTM036. 2

F. Geerts, G. Mecca, P. Papotti, and D. Santoro.2013. The LLUNATIC data-cleaning framework. *Proceedings of the VLDB Endowment*, 6(9): 625–636. DOI: 10.14778/2536360.2536363. 164, 166, 168, 169

F. Geerts, G. Mecca, P. Papotti, and D. Santoro. 2014a. Mapping and cleaning. In *Proceedings of the of the 30th International Conference on Data Engineering*, pp. 232–243. 169

F. Geerts, G. Mecca, P. Papotti, and D. Santoro. 2014b. That's all folks! LLUNATIC goes open source. *Proceedings of the VLDB Endowment*, 7(13): 1565–1568. 169

L. Getoor and A. Machanavajjhala. 2012. Entity resolution: theory, practice and open challenges. *Proceedings of the VLDB Endowment*, 5(12): 2018–2019. DOI: 10.14778/2367502.2367564. 2, 48

C. Gokhale, S. Das, A. Doan, J. F Naughton, N. Rampalli, J. Shavlik, and X. Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 601–612. DOI: 10.1145/2588555.2588576. 84

RIGHTSLINK

L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *Proceedings of the VLDB Endowment*, 1(1): 376–390. DOI: 10.14778/1453856.1453900. 131, 133, 164, 172, 173

I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*, vol. 1. MIT press Cambridge. 212

G. Gottlob and A. Nash. 2008. Efficient core computation in data exchange. *J. ACM*, 55(2): 9. DOI: 10.1145/1346330.1346334. 117

G. Grahne. 1991. *The problem of incomplete information in relational databases*, volume 554. Springer Science+Business Media. 2

S. Greco, C. Molinaro, and F. Spezzano. 2012. Incomplete data and data dependencies in relational databases. *Synthesis Lectures on Data Management*. DOI: 10.2200/S00435ED1V01Y201207DTM029. 2

F. E. Grubbs. 1969. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1): 1–21. DOI: 10.1080/00401706.1969.10490657. 13, 17

A. Gruenheid, X. L. Dong, and D. Srivastava. 2014. Incremental record linkage. *Proceedings of the VLDB Endowment*, 7(9): 697–708. DOI: 10.14778/2732939.2732943. 57

S. Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pp. 317–330. DOI: 10.1145/1925844.1926423. 91, 93, 94, 96, 102, 104

S. Gulwani, W. R. Harris, and R. Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8): 97–105. DOI: 10.1145/2240236.2240260. 102, 107

P. J Guo, S. Kandel, J. Hellerstein, and J. Heer. 2011. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. DOI: 10.1145/2047196.2047205. 94, 100, 106

L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. 2005. Clio grows up: from research prototype to industrial tool. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 805–810. DOI: 10.1145/1066157.1066252. 139

F. R. Hampel, E. M. Ronchetti, P. J Rousseeuw, and W. A. Stahel. 2011. *Robust Statistics: the Approach Based on Influence Functions*, volume 114. J. Wiley & Sons. 20

O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. 2009. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1): 1282–1293. DOI: 10.14778/1687627.1687771. 57

D. Hawkins. 1980. *Identification of Outliers*, volume 11. Chapman and Hall. 4, 11, 195

S. Hawkins, H. He, G. Williams, and R. Baxter. 2002. Outlier detection using replicator neural networks. In *Proceedings of the 4th International Conference Data Warehousing and Knowledge Discovery*, pp. 170–180. Springer. DOI: 10.1007/3-540-46145-0_17. 32, 195

RIGHTSLINK

J. Heer, J. Hellerstein, and S. Kandel. 2015. Predictive interaction for data transformation. In *Proceedings of the 7th Biennial Conf. on Innovative Data Systems Research*. DOI: 10.1.1.692.1613. 94, 100

J. M. Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*. 1, 2, 19, 44

M. A. Hernández and S. J. Stolfo. 1995. The merge/purge problem for large databases. *ACM SIGMOD Rec.*, 24(2): 127–138. DOI: 10.1145/568271.223807. 57, 64

M. A. Hernández and S. J. Stolfo. 1998. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1): 9–37. DOI: 10.1023/A:1009761603038. 55, 64

A. Hernich and N. Schweikardt. 2007. CWA-solutions for data exchange settings with target dependencies. In *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 113–122. ACM. DOI: 10.1145/1265530.1265547. 117

T. N. Herzog, F. J. Scheuren, and W. E. Winkler. 2007. *Data Quality and Record Linkage Techniques*. Springer Science+Business Media. 2, 48

V. J. Hodge and J. Austin. 2004. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2): 85–126. DOI: 10.1007/s10462-004-4304-y. 2, 12

H. Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6): 417. DOI: 10.1037/h0071325. 33

Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal*, 42(2): 100–111. DOI: 10.1093/comjnl/42.2.100. 125, 148

B. Iglewicz and D. C. Hoaglin. 1993. *How to detect and handle outliers*, volume 16. ASQC Quality Press. 19

I. F. Ilyas and X. Chu. 2015. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4): 281–393. DOI: 10.1561/1900000045. 2

M. Interlandi and N. Tang. 2015. Proof positive and negative in data cleaning. In *Proceedings of the 31st International Conference on Data Engineering*. DOI: 10.1109/ICDE.2015.7113269. 139, 145, 146

Z. Ives, C. Knoblock, S. Minton, M. Jacob, P. Talukdar, R. Tuchinda, J. L. Ambite, M. Muslea, and C. Gazen. 2009. Interactive data integration through smart copy & paste. In *Proceedings of the 4th Biennial Conf. on Innovative Data Systems Research*, 2009. 102

M. A. Jaro. 1976. Unimatch: A record linkage system: User's manual. *U.S. Bureau of the Census*. 51

W. Jin, A. K. H. Tung, and J. Han. 2001. Mining top-n local outliers in large databases. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 293–298. DOI: 10.1145/502512.502554. 30

RIGHTSLINK

W. Jin, A. K. H. Tung, J. Han, and W. Wang. 2006. Ranking outliers using symmetric neighborhood relationship. In *Advances in Knowledge Discovery and Data Mining, 10th Pacific-Asia Conf.*, pp. 577–593. DOI: 10.1007/11731139_68. 30

Z. Jin, M. R. Anderson, M. Cafarella, and H.V. Jagadish. 2017. Foofah: Transforming data by example. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 683–698. ACM. 91, 107

M. V. Joshi, R. C. Agarwal, and V. Kumar. 2001. Mining needle in a haystack: classifying rare classes via two-phase rule induction. *ACM SIGMOD Rec.*, 30(2): 91–102. 32

S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conf. on Human Factors in Computing Systems*, pp. 3363–3372. DOI: 10.1145/1978942.1979444. 91, 93, 94, 95, 100

Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. 2015. BigDansing: A system for big data cleansing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1215–1230. DOI: 10.1145/2723372.2747646. 151, 160

A. Kimmig, L. Mihalkova, and L. Getoor. 2015. Lifted graphical models: a survey. *Machine Learning*, 99(1): 1–45. DOI: 10.1007/s10994-014-5443-2. 211

E. M Knorr and R. T. Ng. 1998. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pp. 392–403. 14, 26, 27, 28

E. M Knorr and R. T. Ng. 1999. Finding intensional knowledge of distance-based outliers. In *Proceedings of the 25th International Conference on Very Large Data Bases*, volume 99, pp. 211–222. 14, 26, 27

S. Kolahi and L. V. S. Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*, pp. 53–62. DOI: 10.1145/1514894.1514901. 7, 163, 164, 179, 181, 182, 183

P. G. Kolaitis, J. Panttaja, and W.-C. Tan. 2006. The complexity of data exchange. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 30–39. ACM. DOI: 10.1145/1142351.1142357. 117

L. Kolb, A. Thor, and E. Rahm. 2012a. Dedoop: efficient deduplication with Hadoop. *Proceedings of the VLDB Endowment*, 5(12): 1878–1881. DOI: 10.14778/2367502.2367527. 67

L. Kolb, A. Thor, and E. Rahm. 2012b. Load balancing for MapReduce-based entity resolution. In *Proceedings of the 28th International Conference on Data Engineering*, pp. 618–629. DOI: 10.1109/ICDE.2012.22. 67

D. Koller and N. Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press. 196, 205, 208

G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. 2003. Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Transactions*

RIGHTSLINK

on Knowledge and Data Engineering, 15(5): 1170–1187. DOI: 10.1109/TKDE.2003.1232271. 27

P. Konda, S. Das, P. Suganthan G.C., A. Doan, A. Ardalan, J. R Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad, G. Krishnan, R. Deep, and V, Raghavendra. 2016. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12): 1197–1208. DOI: 10.14778/3007263.3007314. 6, 86, 87, 197, 225

H. Köpcke, A. Thor, and E. Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1): 484–493. DOI: 10.14778/1920841.1920904. 66

Y. Kou, C.-T. Lu, and D. Chen. 2006. Spatial weighted outlier detection. In *Proceedings of the SIAM International Conference on Data Mining*, pp. 614–618. SIAM. DOI: 10.1137/1.9781611972764.71. 40

N. Koudas, S. Sarawagi, and D. Srivastava. 2006. Record linkage: similarity measures and algorithms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 802–803. DOI: 10.1145/1142473.1142599. 2, 48

N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. 2009. Metric functional dependencies. In *Proceedings of the 25th International Conference on Data Engineering*, pp. 1275–1278. DOI: 10.1109/ICDE.2009.219. 138, 141

H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. 2009. Outlier detection in axis-parallel subspaces of high dimensional data. In *Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conf.*, pp. 831–838. Springer. DOI: 10.1007/978-3-642-01307-2_86. 33, 37, 39

S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12): 948–959. DOI: 10.14778/2994509.2994514. 218, 219

L. V.S. Lakshmanan, F. Sadri, and S. N. Subramanian. 2001. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Transactions on Database Systems*, 26(4): 476–519. DOI: 10.1145/503099.503102. 94

Y. LeCun, Y. Bengio, and G. Hinton. 2015. Deep learning. *Nature*, 521(7553): 436. 212

E. L. Lehmann and J. P Romano. 2006. *Testing Statistical Hypotheses*. Springer Science+Business Media. 16

J. Leskovec, A. Rajaraman, and J. D. Ullman. 2014. *Mining of Massive Datasets*. Cambridge University Press. 62, 63

V.I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, page 707. 49

J. Liang and S. Parthasarathy. 2016. Robust contextual outlier detection: Where context meets sparsity. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pp. 2167–2172. ACM. DOI: 10.1145/2983323.2983660. 33, 40, 43

RIGHTSLINK

L. Libkin. 2006. Data exchange and incomplete information. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 60–69. ACM. DOI: 10.1145/1142351.1142360. 114, 117

A. Lopatenko and L. E. Bertossi. 2007. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *Proceedings of the 11th International Conference on Database Theory*, pp. 179–193. DOI: 10.1007/11965893_13. 179, 191

J. Ma and S. Perkins. 2003. Time-series novelty detection using one-class support vector machines. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pp. 1741–1745. IEEE. DOI: 10.1109/IJCNN.2003.1223670. 32

S. Ma, W. Fan, and L. Bravo. 2014. Extending inclusion dependencies with conditions. *Theoretical Computer Science*, 515:64–95. DOI: 10.1016/j.tcs.2013.11.002 . 139

P. C. Mahalanobis. 1936. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55. 21

D. Maier, A. O. Mendelzon, and Y. Sagiv. 1979. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4): 455–469. DOI: 10.1145/320107.320115. 2, 116

C. Mayfield, J. Neville, and S. Prabhakar. 2010. Eracer: A database approach for statistical inference and data cleaning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 75–86. DOI: 10.1145/1807167.1807178. 208

A. McCallum and B. Wellner. 2004. Conditional models of identity uncertainty with application to noun coreference. In *Advances in Neural Information Proc. Systems 17, Proc. Neural Information Proc. Systems*, pp. 905–912. 205

A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. 2011. Tracing data errors with view-conditioned causality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 505–516. DOI: 10.1145/1989323.1989376. 151, 153, 154, 155

M. Michelson and C. A. Knoblock. 2006. Learning blocking schemes for record linkage. In *Proceedings of the 21st National Conf. on Artificial Intelligence and 18th Innovative Applications of Artificial Intelligence Conf.*, volume 21, page 440. 62

A. E. Monge and C. Elkan. 1996. The field matching problem: Algorithms and applications. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp. 267–270. 55

S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 19–34. ACM. DOI: 10.1145/3183713.3196926. 200, 201, 202

E. Muller, I. Assent, U. Steinhausen, and T. Seidl. 2008. Outrank: ranking outliers in high dimensional data. In *Proceedings of the Workshops of 24th International Conference on Data Engineering*, pp. 600–603. IEEE. DOI: 10.1109/ICDEW.2008.4498387. 33, 37, 39

F. Naumann and M. Herschel. 2010. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. 2, 48

RIGHTSLINK

D. F. Nettleton, A. Orriols-Puig, and A. Fornells. 2010. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4): 275–306. DOI: 10.1007/s10462-010-9156-z. 218

H. B. Newcombe, J. M. Kennedy, S.J. Axford, and A. P. James. 1959. Automatic linkage of vital records. *Science*, 130(3381): 954–959. 55

V. Ng and C. Cardie. 2002. Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting Assoc. for Computational Linguistics*, pp. 104–111. DOI: 10.3115/1073083.1073102. 60

A. Okcan and M. Riedewald. 2011. Processing theta-joins using mapReduce. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 949–960. ACM. DOI: 10.1145/1989323.1989423. 68

G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. 2014. Meta-blocking: Taking entity resolutionto the next level. *IEEE Transactions on Knowledge and Data Engineering*, 26(8): 1946–1960. DOI: 10.1109/TKDE.2013.54. 61

G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment*, 9(9): 684–695. DOI: 10.14778/2947618.2947624. 61

S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. 2003. LOCI: fast outlier detection using the local correlation integral. In *Proceedings of the 19th International Conference on Data Engineering*, pp. 315–326. DOI: 10.1109/ICDE.2003.1260802. 30

T. Papenbrock and F. Naumann. 2016. A hybrid approach to functional dependency discovery. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 821–833. ACM. DOI: 10.1145/2882903.2915203. 128, 130, 148

T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. 2015a. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10): 1082–1093. DOI: 10.14778/2794367.2794377. 125

T. Papenbrock, S. Kruse, J.-A. Quiané-Ruiz, and F. Naumann. 2015b. Divide & conquer-based inclusion dependency discovery. *Proceedings VLDB Endowment*, vol. 8, pp. 774–785. DOI: 10.14778/2752939.2752946. 139

L. Parsons, E. Haque, and H. Liu. 2004. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1): 90–105. DOI: 10.1145/1007730.1007731. 36

E. Parzen. 1962. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3): 1065–1076. DOI: 10.1214/aoms/1177704472. 25

K. Pearson. 1894. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110. 23

K. Pearson. 1901. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572. 33

L. Philips. 1990. Hanging on the metaphone. *Computer Language*, 7(12). 54

RIGHTSLINK

L. Philips. 2000. The double metaphone search algorithm. *C/C++ Users Journal*, 18(6): 38–43. 54

H. Poon and P. Domingos. 2008. Joint unsupervised coreference resolution with Markov logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 650–659. Association for Computational Linguistics. 205

E. Rahm and P. A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4): 334–350. DOI: 10.1007/s007780100057. 117

E. Rahm and H. H. Do. 2000. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23:2000. DOI: 10.1.1.98.8661. 2

V. Raman and J. M. Hellerstein. 2001. Potter's Wheel: An interactive data cleaning system. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *Proceedings of the 27th International Conference on Very Large Data Bases VLDB Endowment*, pp. 381–390. Morgan & Claypool Publishers, San Rafael, CA. 91, 93, 94, 96

S. Ramaswamy, R. Rastogi, and K. Shim. 2000. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 427–438. DOI: 10.1145/335191.335437. 27

V. Rastogi, N. Dalvi, and M. Garofalakis. 2011. Large-scale collective entity matching. *Proceedings of the VLDB Endowment*, 4(4): 208–218. DOI: 10.14778/1938545.1938546. 205

A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3): 269–282. DOI: 10.14778/3157794.3157797. 213, 220

A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. 2016. Data programming: Creating large training sets, quickly. In *Advances in neural information processing systems*, pp. 3567–3575. 220, 221

G. Ratsch, S. Mika, B. Scholkopf, and K.-R. Muller. 2002. Constructing boosting algorithms from SVMs: an application to one-class classification. *IEEE Trans. Pattern Analy. Machine Intell.*, 24(9): 1184–1199. DOI: 10.1109/TPAMI.2002.1033211. 31, 32

T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. 2017a. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11): 1190–1201. DOI: 10.14778/3137628.3137631. 8, 166, 196, 206, 207, 210, 214, 221

T. Rekatsinas, M. Joglekar, H. Garcia-Molina, A. G. Parameswaran, and C. Ré. 2017b. Slimfast: Guaranteed results for data fusion and source reliability. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1399–1414. DOI: 10.1145/3035918.3035951. 196, 205

M. Richardson and P. Domingos. February 2006. Markov logic networks. *Mach. Learn.*, 62(1–2): 107–136. DOI: 10.1007/s10994-006-5833-1. 205

M. Rosenblatt. 1956. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3): 832–837. 25

RIGHTSLINK

B. Rosner. 1983. Percentage points for a generalized ESD many-outlier procedure. *Technometrics*, 25(2): 165–172. 17

P. J Rousseeuw and K. V. Driessen. 1999. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3): 212–223. DOI: 10.1080/00401706.1999.10485670. 22, 23

R. C. Russell. 1918. Index, April 2 1918. URL http://www.google.com/patents/US1261167. U.S. Patent 1,261,167. 53

C. De Sa, C. Zhang, K. Olukotun, and C. Ré. 2015. Rapidly mixing Gibbs sampling for a class of factor graphs using hierarchy width. In *Advances in Neural Information Proc. Systems 28, Proc. 29th Annual Conf. on Neural Information Proc. Systems*, pp. 3097–3105. 211

S. Salvador, P. Chan, and J. Brodie. 2004. Learning states and rules for time series anomaly detection. In *FLAIRS Conference*, pp. 306–311. 40

S. Sarawagi. 2008. Information extraction. *Foundations and Trends in Databases*, 1(3): 261–377. 93

S. Sarawagi and A. Bhamidipaty. 2002. Interactive deduplication using active learning. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 269–278. DOI: 10.1145/775047.775087. 195, 196, 197, 198, 200

A.-D. Sarma, A. Jain, A. Machanavajjhala, and P. Bohannon. 2012. An automatic blocking mechanism for large-scale de-duplication tasks. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pp. 1055–1064. DOI: 10.1145/2396761.2398403. 62

G. Schohn and D. Cohn. 2000. Less is more: Active learning with support vector machines. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 839–846. 199

B. Schölkopf, J. C Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7): 1443–1471. DOI: 10.1162/089976601750264965. 31

D. W. Scott. 2008. The curse of dimensionality and dimension reduction. In *Multivariate Density Estimation: Theory, Practice, and Visualization*, pp. 195–217. 12, 32

P. Sen, A. Deshpande, and L. Getoor. 2009. Prdb: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5): 1065–1090. DOI: 10.1007/s00778-009-0153-2. 205

S. Shekhar, C.-T. Lu, and P. Zhang. 2001. Detecting graph-based spatial outliers: algorithms and applications (a summary of results). In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 371–376. ACM. DOI: 10.1145/502512.502567. 40

J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré. 2015. Incremental knowledge base construction using DeepDive. *Proceedings of the VLDB Endowment*, 8(11). DOI: 10.14778/2809974.2809991. 210

J. Shlens. 2014. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*. 33

RIGHTSLINK

J. S. Simonoff. 2012. Smoothing methods in statistics. Springer Science+Business Media. 25

E. H. Simpson. 1951. The interpretation of interaction in contingency tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 238–241. 217, 218

R. Singh and S. Gulwani. 2012. Learning semantic string transformations from examples. *Proceedings of the VLDB Endowment*, 5(8): 740–751. DOI: 10.14778/2212351.2212356. 91, 107

P. Singla and P. Domingos. 2006. Entity resolution with Markov logic. In *Proceedings of the 2006 IEEE International Conference on Data Mining*, pp. 572–582. DOI: 10.1109/ICDM.2006.65. 205

Q. Song, W. Hu, and W. Xie. 2002. Robust support vector machine with bullet hole image classification. *IEEE Transaction Systems, Man, and Cybernetics*, 32(4): 440–448. DOI: 10.1109/TSMCC.2002.807277. 32

S. Song and L. Chen. 2009. Discovering matching dependencies. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management*, pp. 1421–1424. DOI: 10.1145/1645953.1646135. 141

X. Song, M. Wu, C. Jermaine, and S. Ranka. 2007. Conditional anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 19(5). DOI: 10.1109/TKDE.2007.1009. 40, 41, 43

W. M. Soon, H. T. Ng, and D. C. Y. Lim. 2001. A machine learning approach to coreference resolution of noun phrases. *Proceedings of the 39th Annual Meeting Assoc. for Computational Linguistics*, 27(4): 521–544. 60

B. W. Steuart and Precision Indexing Staff. 1994. *The Daitch-Mokotoff Soundex Reference Guide*. Heritage Quest. 54

M. Stonebraker and I. F. Ilyas. 2018. Data integration: The current status and the way forward. *IEEE Data Eng. Bull.*, 41(2): 3–9. 196

M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. 2013. Data curation at scale: The data tamer system. In *Proceedings of the 6th Biennial Conf. on Innovative Data Systems Research*. DOI: 10.1.1.302.8817. 6, 85, 87, 163, 197

S. Suri and P. Bailis. 2017. Drop: Dimensionality reduction optimization for time series. *arXiv preprint arXiv:1708.00183*. 33

R. L. Taft. 1970. *Name Search Techniques*. Special report (New York State Identification and Intelligence System). Bureau of Systems Development, New York State Identification and Intelligence System. 54

G. Tang, J. Bailey, J. Pei, and G. Dong. 2013. Mining multidimensional contextual outliers from categorical relational data. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, pp. 43:1–43:4. DOI: 10.1145/2484838.2484883. 33, 40, 41, 43

J. Tang, Z. Chen, A. Wai-Chee Fu, and D. Wai-Lok Cheung. 2002. Enhancing effectiveness of outlier detections for low density patterns. In *Advances in Knowledge Discovery and Data Mining, 6th Pacific-Asia Conf.*, pp. 535–548. DOI: 10.1007/3-540-47887-6_53. 30

RIGHTSLINK

O. Taylor and D. Addison. 2000. Novelty detection using neural network technology. In *13th International Congress on Condition Monitoring and Diagnostic Engineering Management*, pp. 731–743. 32

S. Tejada, C. A. Knoblock, and S. Minton. 2001. Learning object identification rules for information integration. *Information Systems*, 26(8): 607–633. DOI: 10.1016/S0306-4379(01)00042-4. 197

W. D. Tietje, A. Y. Polyakov, V. Rolland, J. E. Hines, and M. K. Oli. 2018. Climatic influences on demography of the California mouse (peromyscus californicus) in semiarid oak woodland. *Journal of Mammalogy*, 99(5): 1149–1158. DOI: 10.1093/jmammal/gyy089. 7

G. L. Tietjen and R. H. Moore. 1972. Some Grubbs-type statistics for the detection of several outliers. *Technometrics*, 14(3): 583–597. DOI: 10.1080/00401706.1972.10488948. 13, 17

S. Tong and D. Koller. 2002. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66. DOI: 10.1162/153244302760185243. 199

R. Tuchinda. 2008. *Building Mashups by Example*. Ph.D. thesis. University of Southern California. 102

R. van der Meyden. 1998. Logical approaches to incomplete information: A survey. In *Logics for databases and information systems*, pp. 307–356. Springer. DOI: 10.1007/978-1-4615-5643-5_10. 116

V. S. Verykios, A. K Elmagarmid, and E. N. Houstis. 2000. Automating the approximate record-matching process. *Information Sciences*, 126(1): 83–98. DOI: 10.1016/S0020-0255(00)00013-X. 57

M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. 2014. Continuous data cleaning. In *Proceedings of the 30th International Conference on Data Engineering*, pp. 244–255. DOI: 10.1109/ICDE.2014.6816655. 164, 173, 174, 177, 178

J. Wang and N. Tang. 2014. Towards dependable data repairing with fixing rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 457–468. ACM. 139, 144, 145

J. Wang, T. Kraska, M. J. Franklin, and J. Feng. 2012a. CrowdER: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11): 1483–1494. DOI: 10.14778/2350229.2350263. 81, 83

J. Wang, G. Li, and J. Feng. 2012b. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 85–96. ACM. DOI: 10.1145/2213836.2213847. 64, 66

J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. 2014. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 469–480. DOI: 10.1145/2588555.2610505. 214, 215, 216, 221

RIGHTSLINK

M. S. Waterman, T. F. Smith, and W. A. Beyer. 1976. Some biological sequence metrics. *Advances in Mathematics*, 20(3): 367–387. DOI: 10.1016/0001-8708(76)90202-4. 50, 51

L. Wei, W. Qian, A. Zhou, W. Jin, and X. Y. Jeffrey. 2003. HOT: Hypergraph-based outlier test for categorical data. In *Advances in Knowledge Discovery and Data Mining, 7th Pacific-Asia Conf.*, pp. 399–410. Springer. DOI: 10.1007/3-540-36175-8_40. 33, 40, 41, 43

A. S. Weigend, M. Mangeas, and A. N. Srivastava. 1995. Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting. *International Journal of Neural Systems*, 6(04): 373–399. DOI: 10.1142/S0129065795000251. 40

M. Weis, F. Naumann, U. Jehle, J. Lufter, and H. Schuster. 2008. Industry-scale duplicate detection. *Proceedings of the VLDB Endowment*, 1(2): 1253–1264. DOI: 10.14778/1454159.1454165. 55

P. Wessa. 2012. Free Statistics Software, Office for Research Development and Education, version 1.2.2. 23-r7. http://www. wessa. net. 24, 26

W. E. Winkler. 1990. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. *Proceedings of the Section on Survey Research*. 51, 52

W. E. Winkler. 1993. *Improved decision rules in the Fellegi-Sunter model of record linkage*. Bureau of the Census, 1993. 56

W. E. Winkler. 1999. The state of record linkage and current research problems. In *Statistical Research Division, U.S. Census Bureau*. 55

E. Wu and S. Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment*, 6(8): 553–564. DOI: 10.14778/2536354.2536356. 151, 153, 155, 156

E. Wu, S. Madden, and M. Stonebraker. 2012. A demonstration of DBWipes: clean as you query. *Proceedings of the VLDB Endowment*, 5(12): 1894–1897. DOI: 10.14778/2367502.2367531. 151, 153, 155

C. M. Wyss, C. Giannella, and E. L. Robertson. 2001. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *International Conference on Big Data Analytics and Knowledge Discovery*, pp. 101–110. DOI: 10.1007/3-540-44801-2_11. 125, 127, 148

M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. 2011. Guided data repair. *Proceedings of the VLDB Endowment*, 4(5): 279–289. DOI: 10.14778/1952376.1952378. 163, 164, 166, 184, 186

M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. 2013. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 553–564. DOI: 10.1145/2463676.2463706. 208

C. Zhang and C. Ré. 2014. Dimmwitted: A study of main-memory statistical analytics. *Proceedings of the VLDB Endowment*, pp. 1283–1294. DOI: 10.14778/2732977.2733001. 210

RIGHTSLINK

J. Zhang, M. Lou, T. W. Ling, and H. Wang. 2004. HOS-miner: a system for detecting outlying subspaces of high-dimensional data. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pp. 1265–1268. DOI: 10.1016/B978-012088469-8/50123-6. 33, 37, 39, 40

C. Zhu, H. Kitagawa, S. Papadimitriou, and C. Faloutsos. 2004. OBE: outlier by example. In *Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conf.*, pp. 222–234. Springer. DOI: 10.1007/978-3-540-24775-3_29. 33, 40

X. Zhu and X. Wu. 2004. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3): 177–210. DOI: 10.1007/s10462-004-0751-8. 218

RIGHTSLINK

Index

τ-constrained repairs, 175

Active learning in data deduplication, 197–200
ActiveClean method, 219
AdaBoost algorithm, 218
Affine gap distance, 51
Aggregate queries, SQL, 214–217
AJAX framework, 85
ALIAS algorithm, 197–200
Alternative hypotheses, 16
Amazon Mechanic Turk, 82
Analytics
  data cleaning for, 214–221
  KNIME system, 101
Anchors in distributed data deduplication, 70
Anonymization, 226
Apriori algorithm, 42
Attribute Comparison operation, 201–202
Attribute Embedding Module, 200–202
Attribute errors in SampleClean, 216
Attribute Similarity Representation Module, 201–202
Attribute Summarization operation, 201–202
Authoring component in syntactic data transformations, 93–94
Automatic repairs
  cardinality-set-minimal repairs, 181–183
  description, 162
  human guided repair, 183–190
  overview, 178–181

Bandwidth property in KDE, 25–26
Bayesian inference problem, 55–56
Behavioral attributes in high-dimensional data, 33
BigDansing system, 160–161
BINDER discovery system, 139
Block operator in BigDansing system, 160–161
Blocking for deduplication
  distributed, 67–68, 71–73
  hash-based, 62–64
  overview, 60–62
  similarity-based, 64–66
Blocking keys in hash-based blocking, 62
Bottom-up clustering, 58
Box Plots, 18
Breakdown points in outlier detection, 19–20

Cardinality-minimal repairs, 178–179
Cardinality-set-minimal repairs, 179–183
Causality analysis in error propagation, 154–155
CDCs (constant denial constraints), 133
Cell groups in LLUNATIC system, 168
Central limit theorem, 16
Certain answer semantics in data exchange, 116–117
CFDMiner method, 132
CFDs (conditional functional dependencies)
  overview, 130–133
  repairing, 170–172

RIGHTSLINK

Character-based similarity metrics, 49–52
Chase procedure
  data exchange, 116
  incomplete data, 2
Chi-square test for independence, 16
CINDs (conditional inclusion dependencies), 139–140
Classifier Module in deep learning, 202
Classify tab in Febrl, 86
Clean master relation in editing rules, 143
Close-to-complete evidence sets in Hydra, 137–138
Clustering
  AJAX framework, 85
  data deduplication, 47–48, 57–60
  Hydra, 138
  probabilistic resolution, 76–78
  similarity-based blocking, 64
  subspace outliers, 36–37, 39
Co-occurrence factors in HoloClean, 209
COD algorithm, 41–42
Committees in ALIAS, 199
Compare tab in Febrl, 86
Compilation stage in HoloClean, 208
Conciseness in DBRx system, 159
Condition tables in numeric functional dependencies, 142
Conditional attributes, 132
Conditional functional dependencies (CFDs)
  overview, 130–133
  repairing, 170–172
Conditional inclusion dependencies (CINDs), 139–140
Conditional random fields (CRF), 205
Conflict avoidance strategies, 75
Conflict hypergraphs
  cardinality-set-minimal repairs, 181–183
  data and rule repair, 175–176
  holistic data cleaning, 167–168
  violation detection, 151–152
Conflict ignorance strategies, 74–75
Conflict resolution strategies
  advanced techniques, 78–81
  classification, 74–76
  probabilistic resolution, 76–78
Consistency manager in guided data repair, 185
Constant denial constraints (CDCs), 133
Constraints
  denial, 133–138
  HoloClean, 209–210
  integrity, 121–123
  relative trust with data, 174–176
  unified cost of changing, 176–177
Contextual attributes in high-dimensional data, 33
Contextual outlier detection
  high-dimensional data, 33–34
  overview, 40–43
Contingency sets in causality analysis, 154
Continuity property in KDE, 25–26
Continuous data cleaning, 177–178
Contradiction conflicts, 74
Core implicants in cardinality-set-minimal repairs, 181
Corleone deduplication system, 84
Correct fixes in human guided repair editing rules, 187
Correlation clustering, 58–59
Cost-minimal repairs, 179
Coverage
  DBRx system, 159
  FASTDC, 136–137
  sources in conflict resolution, 81
CRF (conditional random fields), 205
Critical regions for test statistics, 16
CrowdER tool, 81–84
CrowdFlower survey, 1
CTANE FD discovery, 132–133
Curse of dimensionality, 11–12, 32

Daitch-Mokotoff Soundex (D-M Soundex), 54
Damerau–Levenshtein distance, 50
Data
  relative trust with constraints, 174–176
  and rule repair, 173–178
  unified cost of changing, 176–177
Data annotation in KATARA, 189

RIGHTSLINK

Data cleaning workflow, 3–4
Data deduplication
  active learning in, 197–200
  blocking for, 60–66
  clustering, 57–60
  conclusion, 88–89, 223
  deep learning for, 200–202
  description, 5–6
  distributed, 66–73
  human-involved, 81–84
  machine learning for, 196–202
  overview, 47–49
  predicting duplicate pairs, 54–56
  record fusion and entity consolidation, 73–81
  similarity metrics, 49–54
  tools, 85–88
Data distribution, 15–16
Data exchange in data transformations, 116–117
Data quality rule definition and discovery
  conclusion, 147–148
  conditional functional dependencies, 130–133
  denial constraints, 133–138
  functional dependencies, 124–130
  miscellaneous constraints, 138–147
  overview, 121–124
Data quality services (DQS), 118
Data repair. *See* Error repair; Repair targets
Data tab in Febrl, 86
Data Tamer tool, 6, 87–88
Data transformation
  conclusion, 118–119, 224
  description, 6–7
  ETL tools, 117–118
  overview, 91–93
  semantic. *See* Semantic data transformations
  syntactic. *See* Syntactic data transformations
Data Wrangler system
  proactive authoring, 103–106
  syntactic data transformations, 94–96
  transformation authoring, 99–101
  transformation execution, 106–107
DataXFormer transformation engine, 107–114
DBRx system, 157–159
DCs (denial constraints)
  data quality rule definition and discovery, 133–138
  description, 7
Decision trees in predicting duplicate pairs, 55
Declarative transformations, 99
Dedoop tool, 67
Deduplication. *See* Data deduplication
Deep learning
  comments, 212–214
  data deduplication, 200–202
DeepDive tool, 210–211
Delete operation
  Data Wrangler system, 105
  similarity metrics, 50
Denial constraints (DCs)
  data quality rule definition and discovery, 133–138
  description, 7
Detect operator in BigDansing system, 160
Dice's coefficient, 52–53
Dimensionality reduction, 32–33
Dirty tuples in guided data repair, 184–185
Discovery
  conditional functional dependencies, 132–133
  data cleaning workflow, 3
  denial constraints, 134–138
  editing rules, 144
  functional dependencies, 124–130
  matching dependency, 141
  pattern, 189
DisDedup tool, 67–73
Distance-based outlier detection methods, 26–27
  global, 27–28
  local, 28–30
  overview, 13–14
Distributed data deduplication
  multiple blocking functions, 72–73

RIGHTSLINK

Distributed data deduplication *(continued)*
overview, 66–67
parallel computation model, 67–68
self-joins, 68–71
single blocking functions, 71–72
Divide operation in Data Wrangler, 95
Domain expert time in deep learning, 213
Domain pruning in HoloClean, 211
Double-Metaphone system, 54
DQS (data quality services), 118
Duplicate pairs, predicting, 54–56
Duplication. *See* Data deduplication
Duplication errors in SampleClean, 216
Dynamic semantics in editing rules, 143

Edit distance in similarity metrics, 50–51
Editing rules (eRs)
human guided repair, 186–187
overview, 143–144
Embedded functional dependencies, 130
Empirical risk minimization (ERM), 210
Environmental attributes in high-dimensional data, 33
Equality-based hash-based blocking, 62
Equi-depth histograms, 23
Equi-width histograms, 23–24
Equivalence classes in LLUNATIC system, 168
ERM (empirical risk minimization), 210
Erroneous tuples in KATARA, 189–190
Error detection overview
conclusion, 225
data cleaning workflow, 3–4
description, 1–2
Error propagation, 152–159
Error repair
automation, 178–190
description, 1
overview, 161–162
repair models, 162–163, 190–193
repair targets. *See* Repair targets
Error-robust ML models, 218
eRs (editing rules)
human guided repair, 186–187
overview, 143–144
ESD (Generalized Extreme Studentized Deviate) Test, 17
ETL (Extract-Transform-Load) tools and process
data transformation, 92, 117–118
error propagation, 153
surveys, 2
Evidence in HoloClean, 208
Evidence patterns in fixing rules, 144
Evidence sets
FASTDC, 134–136
Hydra, 137–138
Exactness of sources in conflict resolution, 81
Example
programming by, 102–103
semantic data transformations by, 108–113
Execution component in syntactic data transformations, 93–94
Execution layer in BigDansing system, 160–161
Explainability in deep learning, 213
Explore tab in Febrl, 86
Extract-Transform-Load (ETL) tools and process
data transformation, 92, 117–118
error propagation, 153
surveys, 2

Factor functions in HoloClean, 208–209
FASTCFD FD discovery, 132–133
FASTDC FD discovery, 134–135
FASTFD FD discovery, 125, 127–128
FASTMCD algorithm, 22–23
FDs (functional dependencies)
conditional, 130–133
discovery, 124–130
overview, 124
FDTrees, 129–130
Febrl (Freely Extensible Biomedical Record Linkage) system, 85–86
Fill operation in Data Wrangler, 105

RIGHTSLINK

Filter phase in DataXFormer, 109
FindVRepairFDs algorithm, 182
Fitting distribution
  nonparametric outlier detection, 23–26
  parametric outlier detection, 19–23
Fixing rules, 144–145
Fold operation in Data Wrangler, 95–96, 105
Format operation in Data Wrangler, 95–96
Fraud detection, 11
Freely Extensible Biomedical Record Linkage (Febrl) system, 85–86
Freshness of sources in conflict resolution, 81
Functional dependencies (FDs)
  conditional, 130–133
  discovery, 124–130
  overview, 124

Gap distance in similarity metrics, 51
Gaussian distribution, 15–16
GDR (guided data repair), 184–186
Generalized Extreme Studentized Deviate (ESD) Test, 17
GenFDsRepair algorithm, 165
GenFix operator in BigDansing system, 160
Gibbs sampling, 211–212
Global distance-based outlier detection, 14, 27–28
Grubbs Test, 13, 17
Guided data repair (GDR), 184–186

HAC (hierarchical agglomerative clustering), 58–59
Hamming distance in similarity metrics, 50
Hampel X84 technique, 20
Hash-based blocking, 62–64
Hierarchical agglomerative clustering (HAC), 58–59
Hierarchical clustering, 58–59
High-dimensional data, outlier detection in
  contextual outlier detection, 40–43
  overview, 32–34
  subspace outliers, 35–40
  use cases, 34–35
Histograms for outlier detection, 18, 23–26
HITs (Human Intelligence Tasks), 82–83
Holistic data cleaning algorithm, 167
Holistic data repair, 166–172
Holistic error detection, 151–152
HoloClean system
  holistic data repair, 166–167
  probabilistic data cleaning models, 208–212
HOS-Miner method, 39–40
HOT algorithm, 41–42
Human guided repair
  conclusion, 226
  editing rules, 186–187
  KATARA, 187–189
  overview, 183–186
Human Intelligence Tasks (HITs), 82–83
Human-involved cleaning, 8–9
Human-involved data deduplication
  Corleone, 84
  CrowdER, 81–84
Hydra FD discovery, 137–138
HYFD FD discovery, 128–129
Hypothesis testing, 13, 16–19

IBM InfoSphere Information Server, 117–118
ICs (integrity constraints), 7–8, 121–123
ILP (integer linear programming) problem, 59–60
Inclusion dependencies (INDs), 139–140
Incomplete data, 2
Index tab in Febrl, 86
Indicator attributes in high-dimensional data, 33
INDs (inclusion dependencies), 139–140
Informatica Data Quality, 118
Information extraction, 93
InfoSphere Information Server, 117–118
Initialization in human guided repair editing rules, 187
Insertion edit operation in data deduplication, 49

RIGHTSLINK

Instance-driven FD discovery, 125
Integer linear programming (ILP) problem, 59–60
Integrity constraints (ICs), 7–8, 121–123
Intention database in probabilistic data cleaning models, 203
Interestingness score in FASTDC, 136
Interleaved matching in repair, 170–172
Inverted indexes in similarity-based blocking, 65–66
Iterate operator in BigDansing system, 160

Jaccard coefficient, 52–53
Jaccard similarity, 62–63, 66
Jaro distance, 51–52
Jaro–Winkler distance, 51–52
Joinable attributes in FASTDC, 135
Jointly validated tuples in KATARA, 189

k-distance in outlier detection, 28–30
KATARA, 187–189
KD-trees in outlier detection, 27
Kernel Density Estimation (KDE), 25–26
KNIME system
  syntactic data transformations, 94
  transformation authoring, 99, 101–102

Labeling functions in Snorkel, 220
Lag Plots, 18
Language component in syntactic data transformations, 93–94
Learning
  cleaning for, 218–219
  guided data repair components, 185
  machine. *See* Machine learning
  modeling errors in, 220–221
Levenshtein distance, 50–51
LLUNATIC system, 166, 168–170
Local distance-based outlier detection, 14, 28–30
Local outlier factor (LOF) method, 28–30
Local reachability density in outlier detection, 29–30
Locality sensitive hashing (LSH) scheme, 62–64
Localized regions in subspace outliers, 37
LOF (local outlier factor) method, 28–30
Log tab in Febrl, 86
Logical layer in BigDansing system, 160
Lowest numbered blocking function, 73
LSH (locality sensitive hashing) scheme, 62–64

Machine learning
  classifiers, 48
  conclusion, 221–222, 224
  data cleaning for, 214–221
  data deduplication, 196–202
  data repair, 203–214
  deep learning, 121–123
  description, 8
  frameworks, 205–212
  KNIME system, 101
  overview, 195–196
  survey, 1
Macrobase, 5
MAD (median absolute deviation), 20
Magellan system, 6, 86–87
Mahalanobis distance, 21–22
Manipulation category in KNIME system, 101
Mapping in AJAX framework, 85
MapReduce tool, 67
Marginal probabilities in HoloClean, 210
Markov Logic Networks (MLNs), 205
Masking in outlier detection, 19
Master data curation, 225–226
Matching dependencies (MDs)
  overview, 140–141
  repairing, 170–172
Matching in AJAX framework, 85
MCD (Minimum Covariance Determinant), 22
MDL (minimum description length) principle, 176–177
MDs (matching dependencies)
  overview, 140–141
  repairing, 170–172
Mean, 15–16
Median absolute deviation (MAD), 20

RIGHTSLINK

Medians, 20
Merge operation
  AJAX framework, 85
  Data Wrangler system, 95
Meta-blocking process in data deduplication, 61
Metaphone system, 54
Metric attributes for high-dimensional data, 33
Metric functional dependencies (MFDs), 141
MinHash scheme, 62–63
Mini-batch SGD, 219
Minimal core implicants in cardinality-set-minimal repairs, 181
Minimal repairs, 175
Minimal set covers in FASTDC, 135–136
Minimal subspaces in outlier detection, 39
Minimum Covariance Determinant (MCD), 22
Minimum description length (MDL) principle, 176–177
Mixture of Gaussians in outlier detection, 23
MLNs (Markov Logic Networks), 205
Model-based outlier detection, 14–15, 30–32
Model-based repair, 190–193
Model parameters in ActiveClean, 219
Model relaxation in HoloClean, 211–212
Modeling errors in learning process, 220–221
Multi-class model-based outlier detection, 15, 31–32
Multi-reducer blocks, 71–72
Multimodal distributions in outlier detection, 23
Multiple blocking functions in data deduplication, 72–73
Multivariate normal distribution, 16
Multivariate outlier detection, 21–23

NADEEF system, 166
Naïve Bayes in predicting duplicate pairs, 55
Negative patterns in fixing rules, 144
Nested loop algorithms in outlier detection, 27–28
New York State Identification and Intelligence System Phonetic Code (NYSIIS), 54
NFDs (numeric functional dependency), 141–144
Noisy dataset versions in HoloClean, 208
Nonparametric approaches in outlier detection, 23–26
Normal distribution, 15–16
Normal Probability Plot technique, 18
NormalizedSC estimates in SampleClean, 215–217
Null hypotheses, 16
Null values in uncertainty conflicts, 74
Numeric functional dependency (NFDs), 141–144
NYSIIS (New York State Identification and Intelligence System Phonetic Code), 54

Observation database in probabilistic data cleaning models, 203
Occam's razor principle, 136
On demand update discovery process in guided data repair, 185
One-at-a-time data repair, 163–166
One-class model-based outlier detection, 15, 31–32
One-sample t-tests, 16
One-sided version of Grubbs' test, 17
Oracle Enterprise Data Quality, 118
Outlier detection, 2
  applications, 5, 11
  challenges, 11–12
  conclusion, 44–45, 223
  description, 4–5
  distance-based, 26–30
  high-dimensional data. *See* High-dimensional data, outlier detection in
  model-based, 14–15, 30–32
  outlier definition, 11

RIGHTSLINK

Outlier detection *(continued)*
statistics-based. *See* Statistics-based outlier detection methods
taxonomy, 12–15
OutRank method in outlier detection, 39
Overlap coefficient in token-based similarity metrics, 52–53
Overlap similarity in similarity-based blocking, 64–65

Parallel computation model for data deduplication, 67–68
Parameter tying in HoloClean, 211
Parametric approach and factors
outlier detection, 19–23
probabilistic data cleaning models, 205
Partition function in HoloClean, 209
Pattern discovery in KATARA, 189
Pattern tableaus
conditional functional dependencies, 130
rules-only repair, 173
Pattern tables for numeric functional dependencies, 142
Pattern validation in KATARA, 189
PCA (Principal Component Analysis), 33
pdf (probability density function) in outlier detection, 13
PGM (probabilistic graphical model), 205
Phonetics-based similarity metrics, 53–54
Phonix system, 54
Physical layer in BigDansing system, 160–161
Poisson distribution in outlier detection, 27
Possible worlds semantics, 77–78
Potter's Wheel
syntactic data transformations, 94
transformation authoring, 99
Preciseness in DBRx system, 159
Precision of blocking functions, 61
Predicates
denial constraints, 133, 137
FASTDC, 134–135
Hydra, 138
Scorpion system, 155–157
SQL queries, 214–215
Predicting duplicate pairs, 54–56
Prefix filtering in similarity-based blocking, 64–66
Preliminary evidences in Hydra, 137–138
Principal Component Analysis (PCA), 33
Principle of minimality in data cleaning models, 206
Privacy, 226
Proactive authoring in Data Wrangler system, 103–106
Proactive transformations, 99
Probabilistic classifiers, 177–178
Probabilistic data cleaning, 203–205
Probabilistic graphical model (PGM), 205
Probabilistic training labels in Snorkel, 220
Probability density function (pdf) in outlier detection, 13
Programming by example, 102–103

Q-grams, 52–53
Queries
data exchange, 116–117
error repair, 162–163, 190–191
probabilistic resolution, 77–78
Scorpion, 141–143
SQL, 214–217
transformations, 108–113
QuickCode system
syntactic data transformations, 94, 96–97
transformation authoring, 102–103
transformation execution, 107

R-trees in outlier detection, 27
Random projections in high-dimensional data, 33
RawSC estimates in SampleClean, 215–217
Reachability distance in outlier detection, 29
Realization process in probabilistic data cleaning models, 203–205
Recall of blocking functions, 61

RIGHTSLINK

Record fusion and entity consolidation in data deduplication
  conflict resolution advanced techniques, 78–81
  conflict resolution strategies, 74–76
  overview, 73–74
  probabilistic resolution, 76–78
Reducer allocation in distributed data deduplication, 72–73
Reduction ratio in blocking for deduplication, 61–62
Refine phase in DataXFormer, 109–110, 112
Refinement component in DataXFormer, 109
Region algorithms in outlier detection, 27
Relative trust between data and constraints, 174–176
Relevant subspace in outlier detection, 39
Relevant tables in DataXFormer, 109
Reliable estimate in ActiveClean, 219
Repair
  error. *See* Error repair; Repair targets
  machine learning, 203–214
Repair targets
  data and rule repair, 173–178
  holistic data repair, 166–172
  one-at-a-time data repair, 163–166
  overview, 161–162
  rules-only repair, 172–173
Repairing stage in HoloClean, 208
Responsibility in causality analysis, 154
Robust statistics
  description, 15
  statistics-based outlier detection methods, 19–23
Rule-based data cleaning
  conclusion, 193–194, 224
  description, 7–8
  error repair. *See* Error repair
  overview, 149
  violation detection, 149–161
Rules-only repair, 172–173
Run Sequence Plot technique, 18
SampleClean, 214–217
Sampling FDs repairs algorithm, 192–193
Scalability
  clustering, 57, 60
  conclusion, 226
Scalable violation detection, 159–161
Schema-driven FD discovery, 125
SchemaSQL, 94
Scope operator in BigDansing system, 160
Scorpion system, 155–157
Self-joins in data deduplication, 68–71
Semantic data transformations
  data exchange, 116–117
  description, 91
  by example, 108–113
  overview, 107–108
Semi-structured data
  conclusion, 226
  data transformations, 93
Sensitivity analysis in Scorpion system, 155
Set-minimal repairs, 179
SGD (stochastic gradient descent) algorithm, 219
Sherlock rules, 145–147
Shingling, 63
Signatures in unified cost model, 176–177
Significance level in hypothesis testing, 16
Similarity-based blocking for deduplication, 64–66
Similarity graphs, 47, 57
Similarity metrics
  character-based, 49–52
  overview, 49
  phonetics-based, 53–54
  token-based, 52–53
Single blocking functions, 71–72
Single-reducer blocks, 71–72
Singular value decomposition (SVD), 33
Smoothness property in KDE, 25–26
Snorkel project, 213, 220–221
SOD method for outlier detection, 39
Solution integration component in DataXFormer, 109
Solutions in data exchange, 116

RIGHTSLINK

Sorting keys in similarity-based blocking, 64
Soundex algorithm, 53–54
Source-to-target dependencies in data exchange, 114
Sources in conflict resolution
  accuracy, 79–80
  dependency, 80–81
  freshness, 81
Space of predicates in FASTDC, 134–135
Sparsity coefficient in subspace outliers, 37–38
Spatial index structures algorithms in outlier detection, 27
Split operation in Data Wrangler, 95–96, 105
Splitting clusters, 58
SQL aggregate queries, 214–217
SQL Server Integration Services, 118
Standard variance, 15–16
Statistics-based outlier detection methods, 15
  data distribution, 15–16
  fitting distribution, nonparametric approaches, 23–26
  fitting distribution, parametric approaches, 19–23
  hypothesis testing, 16–19
  overview, 12–13
Stochastic gradient descent (SGD) algorithm, 219
Student t-distribution in outlier detection, 27
Subspace outlier detection, 35–40
Substitution edit operation for similarity metrics, 50
Succinctness in FASTDC, 136
Suggestions in human guided repair editing rules, 187
Suitability scores in Data Wrangler system, 104
Supervised techniques in predicting duplicate pairs, 55–56
Support vector machines (SVMs)
  model-based outlier detection methods, 31–32
  predicting duplicate pairs, 55
SVD (singular value decomposition), 33
Syntactic data transformations
  description, 91
  overview, 93–94
  transformation authoring, 99–106
  transformation execution, 106–107
  transformation languages, 93–99

Tableaus
  conditional functional dependencies, 130
  rules-only repair, 173
TANE FD discovery, 125–127
Template-based graphical models, 211
Test statistic T, 16
Tietjen-Moore Test, 13, 17
Token-based similarity metrics, 52–53
Top-down clustering, 58
Training data in deep learning, 212–213
Transform-Data-by-Example, 6–7
Transformation. *See* Data transformation
Transformation authoring
  Data Wrangler system, 100–101, 103–106
  KNIME system, 101–102
  overview, 99–100
  QuickCode system, 102–103
Transformation by example, 99
Transformation languages, 93–99
Transitivity-based clustering, 57
Triangle data distribution, 69–71
Tuple check constraints in conditional functional dependencies, 131
Tuple-independent probabilistic database, 204–205
Two-sample t-tests, 16
Two-sided version of Grubbs' test, 17
Typographical errors in data deduplication, 49–50

Unary INDs, 139–140

RIGHTSLINK

Uncertain clean (U-clean) relations, 77–78
Uncertainty conflicts, 74
Uncertainty score in ALIAS, 200
Unfold operation in Data Wrangler, 95–96, 105
Unified cost of changing data and constraints, 176–177
Univariate outlier detection, 19–20
Universal solution in data exchange, 116
Unlabeled instances in ALIAS, 200
Unstructured data
  conclusion, 226
  data transformation, 93
Unsupervised techniques in predicting duplicate pairs, 55
Update scores in guided data repair, 185

Validated tuples in KATARA, 189
Variable attributes in conditional functional dependencies, 132
Variable denial constraints (VDCs), 133
VC (view-conditioned cause) in causality analysis, 154
VCC (view-conditioned counterfactual cause) in causality analysis, 154
VDCs (variable denial constraints), 133
Vertica instances, 111
View category in KNIME system, 101
View-conditioned cause (VC) in causality analysis, 154
View-conditioned counterfactual cause (VCC) in causality analysis, 154
Violation detection
  error propagation, 152–159
  holistic error detection, 151–152
  integrity constraints, 7
  overview, 149–151
  scalable violation detection, 159–161

Web forms in semantic data transformations, 113–114
Web tables in semantic data transformations, 109–113
Windowing methods in similarity-based blocking, 64

X-Trees
  global distance-based outlier detection, 27
  HOS-Miner method, 39

Z-scores, 19
Zipfian distribution in outlier detection, 23

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

Author Biographies

Ihab F. Ilyas

Ihab F. Ilyas is a professor in the Cheriton School of Computer Science and the NSERC-Thomson Reuters Research Chair on data quality at the University of Waterloo. His main research focuses on the areas of big data and database systems, with special interest in data quality and integration, managing uncertain data, rank-aware query processing, and information extraction. Ihab is also a co-founder of Tamr, a startup focusing on large-scale data integration and cleaning. He is a recipient of the Ontario Early Researcher Award (2009), a Cheriton Faculty Fellowship (2013), an NSERC Discovery Accelerator Award (2014), and a Google Faculty Award (2014), and he is an ACM Distinguished Scientist. Ihab is an elected member of the VLDB Endowment board of trustees, elected SIGMOD vice chair, and an associate editor of the ACM Transactions of Database Systems (TODS). He holds a Ph.D. in Computer Science from Purdue University, West Lafayette.

RIGHTSLINK

Xu Chu

Xu Chu is a tenure-track assistant professor in the School of Computer Science at Georgia Institute of Technology. He obtained his Ph.D. from the University of Waterloo in 2017. His research interests resolve around two themes: using data management technologies to make machine learning more usable, and using machine learning to tackle hard data management problems such as data integration. He won the Microsoft Research Ph.D. Fellowship in 2015. He also received the Cheriton Fellowship from the University of Waterloo, 2013–2015.

RIGHTSLINK

RIGHTSLINK
Copyright Clearance Center

RIGHTSLINK
Copyright Clearance Center

Data Cleaning

Ihab F. Ilyas
Xu Chu

Data quality is one of the most important problems in data management, since dirty data often leads to inaccurate data analytics results and incorrect business decisions. Poor data across businesses and the U.S. government are reported to cost trillions of dollars a year. Multiple surveys show that dirty data is the most common barrier faced by data scientists. Not surprisingly, developing effective and efficient data cleaning solutions is challenging and is rife with deep theoretical and engineering problems.

This book is about data cleaning, which is used to refer to all kinds of tasks and activities to detect and repair errors in the data. Rather than focus on a particular data cleaning task, we give an overview of the end-to-end data cleaning process, describing various error detection and repair methods, and attempt to anchor these proposals with multiple taxonomies and views. Specifically, we cover four of the most common and important data cleaning tasks, namely, outlier detection, data transformation, error repair (including imputing missing values), and data deduplication. Furthermore, due to the increasing popularity and applicability of machine learning techniques, we include a chapter that specifically explores how machine learning techniques are used for data cleaning, and how data cleaning is used to improve machine learning models.

This book is intended to serve as a useful reference for researchers and practitioners who are interested in the area of data quality and data cleaning. It can also be used as a textbook for a graduate course. Although we aim at covering state-of-the-art algorithms and techniques, we recognize that data cleaning is still an active field of research and therefore provide future directions of research whenever appropriate.

ABOUT ACM BOOKS

ACM Books is a series of high-quality books published by ACM for the computer science community. ACM Books publications are widely distributed in print and digital formats by major booksellers and are available to libraries and library consortia. Individual ACM members may access ACM Books publications via separate annual subscription.

ISBN 978-1-4503-7153-7
90000
9 781450 371537

BOOKS.ACM.ORG · WWW.MORGANCLAYPOOLPUBLISHERS.COM