EXPLORING MANUAL CORRECTION AS A SOURCE OF USER FEEDBACK IN PAY-AS-YOU-GO INTEGRATION

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN THE FACULTY OF SCIENCE AND ENGINEERING

2021

By
Nurzety Aqtar Ahmad Azuan
School of Computer Science
# Contents

Abstract .............................................. 9

Declaration .......................................... 11

Copyright ............................................ 12

Acknowledgements ..................................... 14

1 Introduction ........................................ 16
   1.1 Research Motivation .............................. 17
   1.2 Dataspaces ........................................ 20
   1.3 User Feedback ..................................... 22
   1.4 Research Problem ................................. 23
   1.5 Aim, Objectives & Contributions .................. 25
      1.5.1 Research Aims and Objectives ................ 25
      1.5.2 Research Contributions ........................ 25
   1.6 Published Work .................................... 26
   1.7 Thesis Outline .................................... 27

2 Literature Review .................................... 29
   2.1 Overview of Data Integration ....................... 30
      2.1.1 Schema Matching ............................... 31
      2.1.2 Schema Mapping ................................ 35
      2.1.3 Mapping Generation ............................ 35
   2.2 Data Integration Challenges ....................... 37
   2.3 Dataspaces Approach ............................... 40
   2.4 Challenges in the Dataspaces Approach ............ 41
   2.5 User Feedback ..................................... 44
2.5.1 Explicit User Feedback ........................................ 45
2.5.2 Implicit User Feedback ....................................... 48
2.6 User Feedback in Data Transformation .......................... 50
2.7 Feedback for Data Refresh ....................................... 55
2.8 Conclusion ......................................................... 57

3 Inferring Feedback from Manual Correction ..................... 59
  3.1 Framework for Manual Correction ............................ 61
     3.1.1 Broken Feedback Loop ................................... 61
     3.1.2 Reconnecting The Feedback Loop ....................... 63
  3.2 Integration Problems ......................................... 64
     3.2.1 Systematic Schema-Level ................................. 66
     3.2.2 Systematic Data-Level .................................... 68
     3.2.3 Unsystematic Data-Level ................................. 69
  3.3 Conclusion ...................................................... 72

4 Inferring False Positive/Negative Feedback ...................... 73
  4.1 Co-Integration .................................................. 74
  4.2 Inferring TP/FP Feedback From Manual Corrections .......... 75
  4.3 Experimental Setup ............................................ 80
  4.4 Evaluation Results ............................................ 83
  4.5 Result Discussion ............................................... 84
  4.6 Conclusions .................................................... 86

5 Inferring Format Transformation Rules ............................ 87
  5.1 Learning Transformations From Data Corrections ............ 89
  5.2 Example Pair Generation Techniques .......................... 91
     5.2.1 Example Pair Generation Strategies .................... 91
  5.3 Evaluation of the Generation Strategies ...................... 95
     5.3.1 Experimental Harness ................................... 95
     5.3.2 Experimental Design .................................... 98
     5.3.3 Results ................................................ 102
     5.3.4 Discussion .............................................. 107
  5.4 Validating the Learned Transformation Rules .................. 109
     5.4.1 The Rule Validation Problem .............................. 109
     5.4.2 Identifying Representative Cases in ManEd .............. 111
List of Tables

2.1 Explicit feedback in the current proposals with their corresponding objects ........................................................................... 47
2.2 Implicit feedback in the current proposals with their corresponding objects ........................................................................... 49
2.3 Implicit feedback in current data management proposals (outside the scope of data integration) .................................................... 50
3.1 Query result from the dataspace .................................................................................................................................................. 62
4.1 Feedback inference based on approach ...................................................................................................................................... 78
4.2 Example of result returned by dataspace .................................................................................................................................... 79
4.3 Example of result after manual correction .................................................................................................................................... 80
5.1 Example pairs generated from the manual corrections .................................................................................................................. 90
5.2 Transformation result from null filter (baseline) example pairs .................................................................................................... 93
5.3 Transformation result from edited value filtering example pairs .................................................................................................. 94
5.4 Transformation results from augmented edited value example pairs .......................................................................................... 95
5.5 Names ......................................................................................................................................................................................... 100
5.6 Phone Numbers ........................................................................................................................................................................ 100
5.7 Dates ....................................................................................................................................................................................... 100
5.8 Token classes and their description ............................................................................................................................................. 111
5.9 Example of before-edit dataset and its corresponding representative cases ................................................................................ 112
5.10 Example of before-edit dataset and its corresponding representative cases ................................................................................ 113
5.11 Representative Cases in the before-edit data set and example pairs ........................................................................................ 113
5.12 Transformation result when only parts of the representative cases are covered in example pairs .............................................................................. 114
5.13 Representative cases of the dataset in experimental harness expressed by case category .................................................................................. 115
6.1 Employee Information .................................................. 126
6.2 Employee Performance .................................................. 127
6.3 Estimation of Time Used with Updates to ‘Q9NPU4’ .......... 139
6.4 Estimation of Time Used with Updates to ‘B0UUJ3’ .......... 139
A.1 Result from manual approach group over ground truth. .... 171
A.2 Result from annotated approach group over ground truth. .. 171
List of Figures

2.1 Example of data integration application in hospitality domain . . . . . . . 39
2.2 Automated initialization of dataspace . . . . . . . . . . . . . . . . . 41
3.1 A dataspace with a broken feedback loop . . . . . . . . . . . . . . . 64
3.2 General framework on manual correction in dataspaces . . . . . . . . 65
3.3 Four types of integration problem in dataspaces . . . . . . . . . . . . 66
4.1 Performance aggregated by group, and by type of feedback . . . . . . 83
4.2 Precision for each volunteer . . . . . . . . . . . . . . . . . . . . . . 84
5.1 General Framework of Manual Correction Transformation (ManEd) . 90
5.2 Architecture of experimental harness . . . . . . . . . . . . . . . . . 97
5.3 Transformation Rule Scores for the Names Data Type . . . . . . . . 104
5.4 Transformation Scores for the Dates Data Type . . . . . . . . . . . . 105
5.5 Transformation Scores for the Phone Number Data Type . . . . . . 108
5.6 Effect of Increasing Case Coverage on Average Transformation Score
for Data Set 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
5.7 Effect of Increasing Case Coverage on Average Transformation Score
for Data Set 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122
5.8 Effect of Increasing Case Coverage on Average Transformation Score
for Data Set 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
6.1 Manual corrections made to query results by the data scientist . . . . 128
6.2 Architecture of data refresh rules . . . . . . . . . . . . . . . . . . . 129
6.3 Example of UniProt Version History . . . . . . . . . . . . . . . . . . 132
6.4 Example of a UniProt entry . . . . . . . . . . . . . . . . . . . . . . 134
6.5 Attributes of UniProt Entries . . . . . . . . . . . . . . . . . . . . . 135
6.6 Example of database cross reference (DR) values being added . . . 136
6.7 Example of database cross reference (DR) values being deleted . . 136
6.8 Example of database cross reference (DR) values being updates.

B.1 Transformation score average with min and max values for Names data type on 50,100 and 200 rows on all three strategies (null, edited and augmented filtering).

B.2 Transformation score average with min and max values for Dates data type on 50,100 and 200 rows on all three strategies (null, edited and augmented filtering).

B.3 Transformation score average with min and max values for Phone Numbers data type on 50,100 and 200 rows on all three strategies (null, edited and augmented filtering).

B.4 Results of 5 runs for rules validation based on representative case for 50, 100 and 200 rows (Set 1).

B.5 Results of 5 runs for rules validation based on representative case for 50, 100 and 200 rows (Set 2).

B.6 Results of 5 runs for rules validation based on representative case for 50, 100 and 200 rows (Set 3).
Abstract

EXPLORING MANUAL CORRECTION AS A SOURCE OF USER FEEDBACK IN PAY-AS-YOU-GO INTEGRATION
Nurzety Azuan
A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy, 2021

Current practice in data integration typically requires extensive upfront effort, such as defining a schema mapping before a useful result can be produced. Dataspase is introduced to minimise the initial design process of data integration by automating the parts of the data integration task that need technical know-how and taking an incremental approach to components that require domain knowledge through user feedback. An imperfect integration of data is constructed quickly and at a minimal cost. This approach is promising, but open problems remain. One issue is that the continuous cycle of feedback in dataspase may not instantly reflect expected results even after obtaining feedback from the user. Because dataspase may use the feedback to fix its underlying structure (such as matching), the outcome from the feedback is invisible to the end-user. As a result, the end-user chooses to manually fix the integration results outside the confine of dataspaces. Hence, the opportunity for dataspase to gather feedback and learn is lost.

In this thesis, we aim to leverage the manual correction effort performed by the user of data, such as data scientists, on query results that they manually improve. This dissertation demonstrated how manual correction could be used to infer feedback values without requiring extra effort from the user. This thesis proposes a general framework for manual correction as another source of implicit feedback. The proposed
general framework aims to determine the potential for manual correction and identify how manual correction can fit into dataspace settings. We then explore other areas of dataspace integration that can maximise the inferred value extracted from manual correction. First, we demonstrate an approach that uses manual correction to inferred feedback values for schema mapping. We compare our work with an existing practice that uses explicit feedback for schema mapping improvement to evaluate our proposed method. Next, we explore the usage of manual correction as example pairs for automated format transformation tasks. We devise three strategies for assessing the proposed approach, and we evaluate our strategies using existing format transformation tools. Lastly, we investigate ways to reuse the iterative effort of manual correction in dealing with changing data sets through a case study on a real-world database (UniProtKB). Our approach to tackling changes in data involves several tasks such as detecting and storing changes and re-apply changes inferred from previous manual correction. We measure our approach over existing work that also deals with real-world database. The results from all three works confirm that a manual correction approach is cost-efficient when dealing with the specific areas of data integration that we explored.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
This work is dedicated to:
Mohd Shahril
Ibu & Ayah
Mama & Papa
Ziqry & Soraya
I am forever thankful for every sacrifices, love and support.
Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Dr Suzanne Embury for her continuous support throughout this journey called a PhD. I am grateful for her immense patience, motivation, and extensive knowledge. Her guidance and encouragement helped me overcome many difficulties in all the time of research and writing of this thesis.

Sincere thanks to my co-supervisor, Prof Norman Paton, who is much involved in my research and writing of this thesis. I am grateful for the time, ideas, insightful comments, and also for the hard questions thrown to me to widen my research from different perspectives.

I thank my colleagues, Maryam, Renè, Iliada, Duhaill, Leanord, Siok and Phin for the support, friendship, and for all the fun we have shared.

I am deeply indebted and thankful to all the love and sacrifices from my two beautiful children Ziqry and Soraya, and my wonderful husband Shahril for supporting me in every step I take. Sincere thanks to my beloved parents, Ibu, Ayah, Mama, Papa and siblings for unconditional love and support then, now and ever.

Finally, to the most important, the most beneficent, the most kind, the most generous, the most knowing who make impossible possible, my Creator. Alhamdulillah.
Glossary and Acronyms

**Data Integration** A process that consolidates data that resides in multiple sources and provides the user with a unified view of these data [Len02].

**Dataspace** is an abstraction layer for a data management system that aims to reduce the upfront cost of data integration through an automatic approach to initialization of an integration system with the involvement of user feedback to incrementally improve the quality of integration in a pay-as-you-go approach.

**Data Wrangling** A process that transforms raw data into more useful forms for data analysis purposes. Data cleaning and data transformation are some of the tasks involved in data wrangling [Gul11, KPHH11a].

**Manual Correction** Manual editing effort made by a data consumer in correcting or refining data sets of a query result so as to fit their personal information needs in completing some specific task, such as for data analytics.

**User of data** Someone with an understanding of the data that they manually correct based on some requirements and/or knowledge. In the context of this thesis, this includes both end-users and domain experts such as data scientists.

**Data Scientist** A professional who uses scientific methods to create meaning from raw data. Data scientists carry out tasks on data such as data exploration, data preparation, data representation and modelling, and data visualization [Str16, Don17].

**ManEd** Manual Edit to Data

**Before Edit** A data set before manual correction take place

**After Edit** A data set after manual correction take place
Chapter 1

Introduction

“For the things we have to learn before we can do them,
we learn by doing them.”

Aristotle

Data integration is an approach that consolidates data from multiple heterogeneous sources and gives the impression that the data come from a single source. Halevy, in his literature survey [Hal01, p. 270], identifies data integration as a way to provide “a uniform query interface to a multitude of autonomous data sources, which may reside within an enterprise or on the World-Wide Web”. Data integration is an approach to providing the user with a unified view of combined data that reside in different sources [Len02]. The advantage of data integration is that it allows the user to pay attention to what information they need rather than having to spend time figuring out how to get to the data they require.

Despite the benefits data integration has to offer, it is difficult to have a perfect integration system. One of the difficulties of data integration lies in the large amount of domain knowledge that needs to be captured to build a data integration solution. Acquiring domain knowledge requires domain and technical experts’ involvement, which results in high upfront costs for data integration. Domain knowledge may be extensive and may contain other details, such as how individual data values are represented, the semantics of the data values and connections between data. Expecting domain and technical experts to provide information on the domain knowledge required upfront may be impractical and inefficient in terms of cost. Thus, other means of acquiring domain knowledge are essential.
1.1. RESEARCH MOTIVATION

The research reported in this thesis focuses on ways to acquire the domain knowledge needed for data integration. One way to gather domain knowledge is through user feedback, and our research explores new ways to obtain this feedback so that it can be used to continuously improve the quality of the integration.

1.1 Research Motivation

Information from the unified view provided by integration assists companies and data-driven organisations in their decision-making processes. Data integration helps improve work processes, achieve goals, make better predictions and achieve better performance due to its ability to bring together disparate data sources so that end-users, such as data scientists, can provide valuable insights on the integrated data [WKB18, PBT+16]. As a result, data integration is now a commonly used approach that benefits many fields, such as business, the life sciences, public services and healthcare. Although integration can benefit an organisation, some challenges need to be addressed and overcome before organisations can attain a useful integration system for supporting their requirements and needs since there is no one-size-fits-all solution for integration. It is instead a process that evolves due to the changing data landscape and business requirements.

We will now present a series of data integration initiatives in several domains, such as in retail businesses, healthcare and smart cities. We highlight the difficulties each integration initiatives may face during the integration process. Although these examples describe integration in different domains, the challenges often boil down to the quality of integrated data. Data accuracy, inconsistency, formats and transformation of data, and heterogeneity of the sources are among the concern in the quality of data integration. These examples demonstrate the significance of manual correction effort in meeting the data requirements of end-users.

- **In Retail:** In retail businesses, combining sales data from the marketing department with warehouse inventory data from the operations team allows businesses to manage and limit the number of products they have in stock. The association of data from both the marketing and operations departments allows the administration of just-in-time stock control to become more efficient for the warehouse, as less space is needed for storage, limiting the chances of overstocking. At the same time, combining these two data sources also assists buyers in making informed purchasing decisions [STD+14].
**Challenges:** Combining multiple data sources is essential since individual data sets alone could not provide the values expected. Common challenges with this kind of integration are, among others, different schema structures, different names for the same attributes and significant differences in data representation. For example, sales data may have product names stored under a schema element called ‘Product’ while the warehouse inventory stores product’s names under ‘Item’. Another example is in the representation of data: for example, where ‘Product’ data in the sales data is in the form of the product name and brand, whereas in the warehouse inventory, ‘Item’ data contains only the product name while the brand is in a separate field. Thus, domain knowledge may be required to determine the semantic relationships in data, specifically across integrated data sources, since the automated approach is not always reliable and often causes uncertainty, such as when looking for matching attribute values of ‘Item’ and ‘Products’ [DHY09, MM10].

- **In Healthcare:** Integrating patient personal information, their medical history, allergies and laboratory test results into one complete and comprehensive record helps to raise the standard of patient care and enable better-informed clinical decisions. As health data becomes more easily accessible and more extensive, health and wellness services can be offered at reduced cost without compromising the outcome. For example, a report by Kayyali et al. [KKVK13] highlighted how integration initiatives in a healthcare setting had reduced the total cost of care for heart disease patients by combining early cholesterol screening data with smoking cessation data and Aspirin intake. As a result, thirty billion dollars in costs have been saved in the United States alone. By integrating electronic patient records with laboratory records, results from imaging centres and data from pharmacies, quick and informed decision-making can be carried out, especially during critical life events.

**Challenges:** In healthcare, data may come from various operations such as administrative, legal, finance, consultation, outpatient services, surgeries and diagnostics. Each of these operations often has its own systems and subsystems, making integrating healthcare applications to enable them to communicate with each other a significant challenge. Hence, this results in inconsistency across the individual systems and misunderstanding of data governance procedures and rules [LR07]. In particular, integration of healthcare data requires people, concepts and technologies to construct robust, secure and scalable infrastructure.
1.1. RESEARCH MOTIVATION

while tackling data accessibility, ownership, privacy and security. Many of these issues require human intervention – for example, aggregating data from diverse systems with different standards, protocols, terminologies and policies usually involve domain knowledge [NBSV13]. Semantic understanding of data is essential in adding context to the integration results and facilitates a more in-depth and meaningful integration of information [SLY+17].

• **In Smart Cities:** Smart city initiatives integrate “systems-of-systems” by utilising information and communication technologies from multiple areas, such as the environment, government and technology. Given this, the potential of the smarter city is enormous [BAW13]. As a result of robust analysis, valuable insights can be derived to further enhance urban business processes such as planning, operations and management [NBH+11]. A smart city connects information and technology (ICT) infrastructure with physical and social infrastructure and businesses to improve quality of life and service operation. The goal is to provide better environmental, social and environmental performance for today’s and tomorrow’s generations [KRJ+14]. For instance, the city of Nice in France offers an efficient parking management system by utilising data retrieved from a network of sensors available throughout its public roads. Such an effort is made possible by using the information gathered from different systems and services available across the city [DD13, KRJ+14].

**Challenges:** Heterogeneity is one of the key concerns in smart city initiatives. Integration and cooperation to aggregate heterogeneous technologies and services at the application layer of smart cities is a tedious and challenging task. For example, platform incompatibilities across the subsystems could be among the obstacles that hamper the development of smart cities [NBH+11]. So, to overcome technological challenges in the smart city, such as problems relating to the interconnection of subsystems and consideration of diverse sources of heterogeneous data, highly skilled workers with specific domain expertise (for example, data analysts) are very much needed to analyse and coordinate these data to extract new and valuable information for the cities and their population [JSG+16].

Despite more than a decade of research work, data integration remains a difficult problem. Among the reasons are:

• The diversity of data sources (since data from multiple sources may have different formats and structure), and difficulties in understanding the behaviour of
CHAPTER 1. INTRODUCTION

data from different sources and targeted systems.

- Different users’ requirements and information needs must also be considered.

- The large volume of related data and unforeseen costs involved in changes after integration is complete is problematic.

All of the reasons mentioned earlier result in the constant involvement of expensive human labour. To overcome problems related to continuous change in data integration, the database research community has proposed a new approach to integration, known as a dataspaces. Dataspaces aim to overcome some of the problems in data integration systems, particularly in the high up-front cost of manual work.

1.2 Dataspaces

The examples given in Section 1.1 involve diverse data sets in areas where data integration needs both technical and domain knowledge to be successful. The requirement for technical expertise plus the high up-front cost of integration activities can delay delivering a first version of the integration. Hence, there is a potential benefit to be found from approaches that provide the first integration quickly and reduce the need for ongoing technical input, which is expensive. For this reason, dataspaces have been proposed to reduce the upfront costs of setting up a data integration system [FHM05, HFM06]. Specifically, dataspaces allow the initial integration to be created using automated tools. As a result, it quickly produces a “best effort” integration that can be used and improved gradually through user feedback.

A dataspaces is a data management abstraction that provides an alternative approach to data integration that does not require full integration to provide useful services. Dataspaces initialisation of the integration system is done automatically using tools rather than human involvement often, domain expert to run the integration tasks. For instance, the semantic mapping between the initial data sources can be done automatically without the knowledge of the domain expert through automated tools [DSDH08]. This automated approach results in a quick but low-quality data integration. Nonetheless, the quality of dataspaces integration can be improved over time whenever the opportunity or need arises by taking advantage of user feedback [HFB+13]. Soliciting feedback from end-users, particularly domain experts or data scientists, creates an opportunity for dataspaces to improve integration quality as it
1.2. DATASPACES

is used [DSDH08, JFH08, HBF+09]. During the improvement phase, acquiring feedback from the end-users is essential. Feedback can be obtained from the accuracy of schema matching deriving from the automated tools of dataspace or on the correctness of the query results [Ser19].

However, the use of user feedback in improving a dataspace is not an easy task. Some of the challenges in gathering such feedback are:

i. Knowing what to ask the user.

ii. In which order the feedback-driven questions should be asked.

iii. Knowing how much feedback is sufficient and when more needs to be requested.

iv. Identify feedback with the most significant effect on the quality of the integration system to be extracted so it can be used for integration refinement.

v. Another challenge faced by dataspaces, which is related to feedback, is knowing what kind of feedback is required, in what amount and whether or not the feedback offered is reliable. Hence, the quality of the integrated data is valid for a long time.

Because businesses and organisations constantly change, this results in constantly changing data and possibly changes in end-user expectations and needs. Feedback can be a useful source of information for coping with the continual changes in data meaning and representation in an incremental way [BPF+11, KK09, ERA17, SRL+07, CVDN09].

From the data integration challenges described earlier, it is clear that knowledge from domain experts is essential in building better data integrations. Dataspaces build on this human knowledge by making use of user feedback in an attempt to reduce the up-front costs of integration (such as when knowledge of a domain expert is needed in identifying the semantic equivalent of two data sources that need to be integrated) and at the same time make the most of human involvement in improving the integration over the system lifetime. In this work, we address the challenges mentioned above by focusing on a particular type of user feedback as one of the essential information sources in data integration.
1.3 User Feedback

Finding and integrating data from disparate sources when the structures, extent and semantics of those data sets are unknown to the data scientist at the outset remains a difficult and costly activity. Tools can help, but a substantial up-front effort is often required to gather and encode the domain knowledge needed. Typically, significant technical know-how must be combined with significant domain expertise, leading to substantial human time costs, and the solutions arrived at through all this effort may not remain valid for long, given the rapidly changing data landscape on which they are built.

A user feedback approach to dataspace integration has the potential to allow usable (though imperfect) data integrations to be created rapidly. The method is also flexible enough to react gracefully to changes in the underlying data sets. But this feedback-driven approach to reaching the required quality of output also has some limitations. It leaves the end-user (the holder of the domain knowledge) in a somewhat secondary role, able to control the form of the integration produced by the dataspace only indirectly, through feedback on individual data items or results sets. The dataspace uses this feedback to tweak internal parameters, adjust ranked orderings of mappings or matchings and nudge the thresholds used in blocking strategies (among other things). The (non-technical) user typically has little chance of knowing quite what effect any single piece of feedback will have on the results or guessing how much feedback they will need to provide to achieve the quality of data they need.

Often, however, data scientists are under pressure to produce data sets of a certain quality level by a specific time. Data may be needed for an upcoming meeting of senior management at which key strategic decisions will be made, for presentation to a government working group setting national policy, or for a ward meeting to decide the treatment policy for the current group of patients. Data scientists who need to produce data for a specific deadline will do what users of data have always done: they will take the best data the tools can provide them with, copy them into some convenient tool (Microsoft Excel, for example) and then fix the problems they can see by hand (manually). For example, a data scientist may resort to an off-the-shelf data transformation tool rather than providing feedback to the dataspace so that the format inconsistency of the integration can be fixed quickly. Or rather than supplying feedback to the dataspace on correct and incorrect tuples retrieved by the integration results so that any incorrect tuples can be removed by correcting the schema mapping, the data scientist, instead, may choose to delete the incorrect tuples manually using
1.4 RESEARCH PROBLEM

a spreadsheet-based tool. The manual corrections performed in the quest to get to the correct and expected result do allow the user’s task to be completed, but they are also a missed opportunity. How much of this manual integration work will need to be repeated the next time this or another end user requires a similar data set? The problem here is not that the user takes over the integration work, but that they do so out of sight of the dataspace in a separate tool. Because the manual work is performed outside the confines of the dataspace, the learning opportunity to improve the query answer of data integration for the subsequent and future request is lost.

1.4 Research Problem

User feedback can improve the quality of data integration, but acquiring reliable and valuable feedback can be a difficult task. Feedback requires the end user to dedicate some time in providing it, which could have been used on other more critical tasks that cannot be performed through tools. Existing research has investigated different ways of soliciting feedback, such as through annotations and crowd-sourcing, and in various aspects of data integration such as source selection, schema matching and mapping. However, the majority of current work in user feedback focuses on obtaining the feedback explicitly. Explicit feedback has limitations, other than being costly; it is also intrusive because it usually interferes with the user’s daily tasks.

Below we list the problems we found in the existing approach, where feedback is acquired explicitly from the user of data. In our work, we consider a “user of data” to be someone who understands the data that they manually correct based on some requirements and/or knowledge. Also, “users of data” include end-users and domain experts such as data scientists:

- Explicit feedback is expensive and time-consuming for the end-user to provide however, feedback is needed in order to improve the integration quality of the dataspace.

- In a dataspace, feedback comes from the data user, such as a data scientist. Acquiring feedback from the end-user limits the amount of explicit feedback that can be expected to receive, and in many cases, it is speculated that the dataspace may not be sufficient to improve the quality of integration to the level required.

As such, how can we acquire more feedback without adding to the work of end-users? In contrast, less attention has been given to implicit feedback. In implicit
feedback, feedback is acquired indirectly as a side benefit of tasks that the user chooses or needs to do without putting the burden of acquiring the feedback on the end-user. In implicit feedback, although users may be aware that feedback gathering activities are taking place, the users do not have to change what they are doing or how they are doing it to provide the feedback makes it non-intrusive compared to its counterpart explicit feedback.

Implicit feedback is also inexpensive because no extra work is needed from the user. However, many assumptions may be required to understand the purpose behind the series of activities performed by users to allow helpful feedback to be derived from them. We consider manual correction performed by data scientists on sets of queries answers to be a valuable source of implicit feedback since the work done during manual correction may reflect the domain knowledge of the data scientist. If manual corrections could be extracted and transformed into feedback, some additional aspects of the improvement phase of a dataspace could be done at no extra cost.

In general, feedback is essential for dataspace as feedback is needed to improve the integration quality. However, acquiring valuable feedback is difficult. In dataspaces, feedback comes from the user of the data (e.g. the data scientists). Therefore, users need to dedicate time to supply feedback to data when, in fact, the time dedicated to providing feedback can be used on more critical tasks. Thus, explicit feedback is considered expensive and time-consuming for the end-user to provide. For this reason, it limits the amount of explicit feedback we can expect to receive, and in many cases, it is not sufficient to improve the quality of the integration to the level required.

Based on acquiring feedback explicitly in dataspace, how can we acquire more feedback without adding end users’ work? To understand the potential of manual correction as an implicit source of feedback in dataspaces, we try to answer a couple of questions. We seek to determine:

- Can manual corrections be used to infer feedback that is useful in improving the quality of an integration?
- Can feedback from manual corrections be applied in different integration problem scenarios?
- What is the most appropriate way to use manual correction in a dataspace integration setting?

To the best of our knowledge, no work has considered capturing manual corrections for use as a source of implicit feedback for dataspace-style integration.
1.5 Aim, Objectives & Contributions

This research aims to understand whether we can infer useful information for improving data integration using end-users manual corrections. Specifically, we look into the manual correction (or manual editing) of the data where the user tidies up and corrects the integration results once the integration tools have finished. Very often, with dataspaces or other forms of integration, the integration tools will not get it all right. They may do lots of things correctly, but some other things might be problematic. Before the data is usable, the user must make some manual corrections to it.

1.5.1 Research Aims and Objectives

We set out to develop an understanding of different ways to infer useful information from manual corrections to be used as a source of feedback to help address data integration problems. The main factor in choosing manual correction as a relevant source of feedback is that manual correction is cost-effective because we are not expecting more input from the user than they choose to put in for themselves and is less obstructive as it is done implicitly. The idea is to obtain added value from actions that end users are going to take regardless. Below we list the objectives of our research:

1. To devise a mechanism that uses manual corrections to obtain the necessary feedback to improve the schema aspects of the integration.

2. To devise techniques that use manual corrections to obtain the feedback that is necessary to reduce formatting inconsistencies.

3. To design a technique that helps to minimise the cumulative effort of manual corrections carried out by the user in dealing with a changing data set over time.

1.5.2 Research Contributions

The following research contributions reported in this dissertation are:

1. A general framework for inference of feedback from manual corrections in a data integration context.

   We propose a high-level framework that describes how manual correction could be incorporated into dataspace management systems as a source of implicit feedback.
2. **An algorithm for inferring true/false positive-style feedback from manual corrections to query results**
   
   We propose Co-Integration, an algorithm that focuses on inferring true/false positive feedback values from manual corrections performed on data sets to improve schema mapping. It should be noted that Co-Integration is our first step in an attempt to better understand manual correction and its potential benefits.

3. **An algorithm for inferring example pairs for learning format transformation rules from manual corrections to query results**
   
   We propose ManEd, an algorithm that focuses on extracting example pairs for format transformation from manual correction. The mechanism does not require the user to understand the transformation rules and the processes that derive those rules but instead focus on strategies that could automate the generation of example pairs from manual corrections.

4. **An approach for re-applying manual corrections to continually changing data sets.**
   
   We propose Data Refresh Rules, a method that automates the re-application of manual corrections to updated data sets. The re-application approach involves several tasks: first, detect changes in data; second, extract changes as a manual correction; third, store information on changes in a knowledge-base and repeat the process while re-applying any necessary changes. The mechanism automatically re-uses relevant information from the stored manual correction. This mechanism offers new ways of dealing with the iterative manual effort often performed on long-lived data sets in improving the quality of data integration over time.

### 1.6 Published Work

The work presented in Chapter 4 and 5 of this thesis is supported by two conference publications:

**Chapter 4 :Inferring Feedback Values from Manual Correction**

1.7. THESIS OUTLINE

Chapter 5: Inferring Format Transformation Rules


1.7 Thesis Outline

The rest of the thesis is organised as follows.

1. **In Chapter 2**, we discuss the scientific literature related to dataspace integration. We present the technical background that underpins this research. In particular, we describe conventional data integration and dataspaces and highlight each approach’s advantages and challenges. We emphasise the benefits of user feedback as one of the mechanisms that overcome the imperfection of dataspace integration. We divide user feedback into two categories *implicit* and *explicit*, and we discuss the related work for these two types of feedback.

2. **In Chapter 3**, we show the general framework we designed to fit manual correction into a dataspace setting. We outline the use and benefits of manual correction approaches as user feedback in the dataspace setting. We start by explaining how manual correction approaches can fit into a dataspace architecture. We also describe the correlation of manual correction with standard components of dataspaces. We also introduce a general framework for manual correction. Finally, we anticipate and identify possible ways to infer user feedback values from manual corrections, presented in later chapters.

3. **In Chapter 4**, we present the inference algorithm we developed for inferring true/false positive feedback from manual correction. We first explore the manual correction approach that underpins the quality of dataspaces through some problem scenarios. Then, based on the work by Belhajjame [BPE+10], we show how to incorporate manual correction as user feedback by automatically inferring feedback values from the manual correction work performed by users. Following the preliminary approach to inferring feedback, we present and demonstrate that the proposed work can improve two different schemas using real-world data and scenarios.
4. **In Chapter 5**, we present the inference algorithm we built to infer example pairs for format transformation from manual corrections. We first review the research on data cleaning, specifically format transformation. We then proceed to introduce the method of soliciting manual correction as example pairs for format transformation. We describe the harness needed and a new framework for deriving example pairs for format transformation. We propose three strategies for assessing the proposed approach. Lastly, we present the evaluation results of all the strategies that we proposed.

5. **In Chapter 6**, we present an approach to reapply manual correction on constantly changing data sets. We first review the research on capturing changes in data. We also identify the types of changes that exist from manual corrections. We then present the basis for reapplying manual corrections to future versions of the data set by recording user actions in correcting the data. We evaluate our approach by comparing our automated approach to reapplying manual corrections with manual work often performed by data scientists using a public data set.

6. **In Chapter 7**, we review the contributions of this research and opportunities for future work.
Chapter 2

Literature Review

“Be curious, not judgmental.”

Walt Whitman

This chapter presents background material on common approaches to data integration. We describe the underlying concepts of data integration, such as schema matching and schema mapping, and human involvement. Next, we present an alternative approach to data integration done in a pay-as-you-go manner, called dataspaces. As proposed by the database research community, dataspaces aim to overcome difficulties in data integration, especially in dealing with the high upfront cost of building data integration systems. We focus our research on one of the essential building blocks of dataspaces: user feedback. We consider the different types of user feedback often solicited by dataspaces and discuss the existing proposals in the literature related to user feedback. We also present challenges in user feedback to highlight the difficulty of feedback gathering in dataspaces, and point out the importance of exploring alternative ways of gathering feedback to support continuous improvement of integration quality.

The remainder of this chapter is organised as follows. In Section 2.1, we present a brief introduction to data integration and the challenges surrounding the integration process. We present a description of processes essential to data integration: schema matching, schema mapping, and mapping generation in Section 2.1.1, 2.1.2 and 2.1.3 respectively. We describe data integration problems in Section 2.2. Then we introduce dataspaces in Section 2.3, an approach proposed to reduce the high upfront costs of data integration. We describe challenges associated with the dataspace approach in Section 2.4. Next, we describe the two type of user feedback available to dataspaces in
Section 2.5. We also discuss user feedback in the context of our research, specifically in data transformation in Section 2.6 and in constantly changing in Section 2.7. Finally, we conclude the chapter in Section 2.8

2.1 Overview of Data Integration

Data integration consolidates data that resides in multiple sources and provides the user with a unified view of these data [Len02]. Through data integration, the user is presented with the impression that the data come from a single source, while in fact they are distributed across multiple resources, possibly stored using different tools, in different formats, and for different purposes. The data integration process is usually difficult due to the heterogeneity of the data sources. One of the main challenges in data integration is that it involves a high upfront cost in setting up an integration application [FHM05]. In establishing semantic relationships between disparate data sources, a great deal of manual work is often required in reconciling the various forms of heterogeneity. Some of the heterogeneity problems encountered during integration are addressed through schema matching and schema mapping components that are at the heart of the integration system.

In data integration, data sources are referred to as local sources, and the schemas of the local sources are referred to as local schemas. The discovery of possible semantic relationships between elements across local schemas is accomplished through a process called schema matching. The user of an integration system may assess and transform data in local sources that conform to different data models and structures by querying local sources through a global (or mediated) schema [HRO06]. A global schema provides a reconciled view of data defined over multiple local schemas, which allow the user to access information as if it comes from a single source; this is made possible through schema mapping.

Here we list steps involved in a typical data integration process [Lev98, Ull00]:

- **Source selection**: relevant source data that fits the integration task is identified. Middleware (such as wrappers or mediators) are constructed that produce a logical connection between each local source and the data integration system.

- **Schema Matching**: collections and/or attributes within the local and global
2.1. OVERVIEW OF DATA INTEGRATION

schemas that are semantically related are systematically identified. Similar concepts in the real world are identified during this phase, such as client and customer are representing corresponding entities and zipcode and postcode representing corresponding attributes.

- **Semantic Mapping**: mappings that describe the semantic relationships between the global schema and the set of local schemas are identified. In essence, these mappings describe how to populate the global schema from the local schemas. For a long time, the generation of mapping has been done manually by domain experts who understand the semantics of all schemas involved. However, in our approach, we pay more attention to automated mapping generation (or semi-automatic, assisted by tools). Details on mapping generation facilitated by tools will be given in Section 2.1.2.

- **User Query Reformulation**: a process that translates user queries over the global schema into queries linked directly to the local schemas.

- **Entity Resolution**: a task that detects duplication of instances within a collection of data sources. The outcome for entity resolution is a grouping of records into those representing the same real world instance. The results are used in the construction of query results.

In the next section, we review existing proposals on different phases involved in building a data integration application. Specifically, we describe works related to schema matching, schema mapping, and human involvement to improve the quality of integration.

### 2.1.1 Schema Matching

The outcome of schema matching is a set of matches that determines the semantic relationships across elements between two given schemas, a source schema and a target schema. Matches do not specify a view; however, they are used in the process of view generation as they specify the corresponding elements between the two schemas. For example, schema matching will help us decide whether data in a client table in one schema refers to the same real world entities as data in a customer table in another, and whether the zipcode attribute in one source corresponds to the attribute postcode in another.
Schema matching is a process that can be performed manually, automatically, or semi-automatically. In the earlier generations of data integration applications, matching was performed manually and was, therefore, expensive [KCGS93]. Since the process is tedious, time-consuming, and prone to human error, automatic schema matching approaches have been proposed [DR02, DLD+04, MGMT02, MBR01]. Extensive surveys of research related to schema matching can be found in the literature [BMR11, RB01, SE05]. We present the classification of automatic schema matching approaches from one of the well-cited works by Rahm et al. [RB01, BMR11]. From their set of classifications, we focus on schema-level matching and instance-level matching in particular.

**Schema-level matching**

Schema-level matching techniques identify matches between source and target schemas using only schema information such as name, data type, relationship type (is-a, part-of, etc.), schema structure (such as XML models and relational), and constraints (e.g., integrity and referential constraints). For matching schemas of various kinds with different modeling languages and varieties of application domains, schema matching tools are usually implemented with several matching algorithms, known as *matchers*. Schema matching use techniques including the following:

- **Element-level matching techniques** consider only single elements and are therefore confined to identifying semantic relationships with 1:1, 1:n and n:1 cardinalities. Element-level techniques are often used before structure-level techniques are applied; hence, this technique acts as the groundwork for matching element structures. The main matching techniques based on elements are string-based techniques, lexical techniques and semantic-based techniques. String-based techniques take lexical structures into account by working on the element’s name and description string. An example of a string-based technique is *edit distance*, where the number of edit operations (such as insertion, deletion, and substitution) needed to transform the name of one element in the compared pair into the other is calculated to compute the similarity between two strings. Lexical techniques also apply at the element level and take advantage of the structure and form of words from natural language processing (NLP). Using methods from NLP, lexical techniques parse the name and description of string elements into the following: (1) Tokens, where a tokeniser cuts the original string at upper case letters, punctuation and digits (for instance, *BookingNumber* is parsed...
2.1. **OVERVIEW OF DATA INTEGRATION**

into Booking, and Number), (2) Stemming, where root words are identified by removing prefixes and suffixes (for instance, friendly and befriend are both converted to their root word friend) and (3) Elimination, where tokens identified as articles, prepositions and conjunctions are discarded so that words that describe real-world concepts can be compared. **Semantic-based techniques** are another group of element-level matching techniques that operate on the meaning of the two elements to compute their similarity. These techniques depend on the semantic relationships between the two compared strings, such as synonymy, where the strings describe the same concept (e.g., “job” and “occupation”), or hyponymy, where both strings describes a subtype of a common entity (e.g., “office” and “house” are both kinds of “building”).

- **Structure-level matching techniques** are based on the matching of combined elements that emerge together in a structure. This type of matching technique addresses relationships with m:n cardinalities [ZR07]. In structure-level matching, the similarity of two elements in terms of their structure is inferred from element-level similarity and supported by combined similarities of the neighbour and analysis of the schema’s positions. There are two types of approach to structure-level matching: scope and neighbourhood. Scope refers to the coverage of a set of schema elements in a match, and allows for partial matching of schema elements. For neighbourhood, matched elements might be impacted by the similarity of nearby elements, and occasionally, matched elements may also be affected by the position of nearby elements. Constraint-based matchers have also been used at the structure-level, as an alternative approach.

**Instance-level matching**

Information about the content and meaning of schema elements can also be acquired from data at the instance-level. In instance-level techniques, matching tasks are carried out using the underlying data content of the source and target systems. Often, instance-level information supplements schema-level techniques to improve matching accuracy, specifically when little information is available at the schema level. In the event where no schema is available, instance data can be used to form a schema. This can be done manually or automatically, and often occurs on semi-structured data. Many existing approaches directly compare instances to find match elements [DDH01, EM07, WT06, DKS+08, BN05]. For complex attribute matches, matches are trained
using machine learning techniques by using subsets of instances before matching ele-
ments [DMDH02, DDH01, RNX12, BM02, MIA17]. Since higher-level similarities
can be derived from low-level similarities, some instance-level approaches identify
matches between the lowest-level elements that relate directly with data values, such
as attribute, instead of between high-level elements such as the data model. Work on
low-level similarity, particularly on attribute, row and term similarity, has been imple-
mented using horizontal matching performed on heterogeneous data sets with unknown
schemas [MKR14]. Instance-level matchers identify one-to-one matches and many-to-
many matches and most classifications of schema-level technique can also be used at
the instance-level. However, below we list approaches that are designed specifically
for instance-level matching, as categorised by Rahm et al. [RB01, BMR11]:

- **Linguistic approaches**: used for text elements, keywords and themes extracted
  based on relative frequencies of words and combination of words. The main
goal of a linguistic matcher is to change the attribute names into a more basic
form using a tokenization approach so that these names can be more accurately
compared for equality [CSH06].

- **Constraint-based approaches**: used for structured data. This group of approaches
  aim to identify properties such as the range and average of a set of numerical val-
ues or character patterns in text data. When enough information regarding the
constraints on the source and target input is available, these approaches can assist
the matcher in determining the more accurate matches between schemas. For ex-
ample, similarity of constraints on the data types/domains used by schemas can
be used to compute a similarity score for matches produced by other means.
Other constraints, such as primary and foreign keys, can also be used in comput-
ing similarity scores. However, solely relying on constraint information is
not always appropriate in gaining an accurate matching result. In some cases,
a constraint may result in an imprecise match because of a similar constraint
among the schema’s other attributes. Even so, taking advantage of constraint
information assists in reducing the number of matching candidates that need to
be considered. Constraint-based matching techniques can also be used together
with other matchers, such as those based on linguistic analyses [MIA17].
2.1. OVERVIEW OF DATA INTEGRATION

2.1.2 Schema Mapping

While schema matchings describe the relationships between elements in different data sources, _schema mappings_ on the other hand, express how data in one format can be transformed into data of another format. For data integration (also schema integration), mapping is used to specify how individual source schemas are used to construct an integrated schema [BLN86]. An integrated schema can be built manually or automatically using the combined source schemas by establishing a unified view over the global schema. Also, schema mappings are the primary feature in query translation since queries posed by a user against the global schema are transformed into a query language understood by every source relying on the relevant mapping. Retrieval of correct data from different data sources as query answers results from reconciling schema heterogeneity enabled by schema mapping. Fundamentally, the mapping must determine how the global schema represents the data entities stored in the source data when they are brought into the data integration system. Schema mapping also is essential to other aspects of data management. For instance, in an information integration system, mapping is considered an essential component because it describes the semantic relationship between a global schema and a set of local schemas [Len02]. Mappings are also used to describe how data in one source could be translated and restructured to data of another source in, for example, data exchange and peer-to-peer settings [FKMP05, HIST03]. In data migration, schema mapping is essential as it is used to transform data from an old database format ready to be saved in the target database [FHH+09].

In the early generation of data integration systems, schema mappings were derived from the output of schema matching, with input from domain and technical experts. However, in recent years, several researchers have proposed prototypes where mapping applications have been built to ease the generation of schema mappings [BMC06, FHH+09, BBF+10]. In the next subsection, we will look into mapping definition and its generation.

2.1.3 Mapping Generation

In the earlier generation of data integration systems, mapping was often done manually in a tedious, time-consuming and expensive process. In general, the involvement of a domain expert and the technical expert is essential during mapping formulation. Mappings created manually tend to be of high quality and can be expected to fulfil
the specified requirements. However, the creation of these high-quality mappings may cause the initialisation phase of integration to be delayed because a long time can be required using this manual approach. This delay is a significant contributor to the high upfront effort required to create data integration systems.

As a result, automatic and semi-automatic mapping generation methods have been proposed, in which users or experts are not required to be involved (or heavily involved) in the mapping generation process. However, because generation of views is complicated and non-trivial, there may be uncertainty about whether the mappings are correct or not. Because of this, many existing proposals depend on a complete schema definition (containing, for example, referential integrity contraints) or external resources (such as a rich specification for the relationships in the schema) for assistance. In practice, relying on a complete schema definition or external resources is not realistic as both of these aspects may not always be available, or could be uncertain and not clearly defined. In the following, we will briefly present some of the notable proposals for automatic mapping generation.

IBM’s Clio system seeks to tackle automatic generation of mapping as required by data integration and data exchange [FHH+09, MHH+01]. Clio takes two schemas as input and matches between the schemas to specify views through a graphical user interface. To join or union schema elements, Clio uses schema information, such as referential integrity constraints and relational and nesting structure. If schema information is incomplete (such as foreign key not being defined) or unavailable, automatic generation of views becomes difficult for Clio.

Comparable to Clio, is Model Management, an infrastructure for tools built to support data management application such as data warehousing, e-commerce and object-to-relational wrappers, to name a few. The primary goal for model management is to offer a system that can manipulate models and mappings between models using high-level operators [BBM07]. Some of the operators included in model management are Diff schemas to uncover differences between two models, Merge schemas to combine two models into one using the mapping between them and Match schemas to identify similar elements between models. There are also other operators such as Compose mapping and translate schemas. Again, the availability of primary and foreign key as joins is essential for model management.

Both Clio [FHH+09] and Model Management [Ber03] need a complete schema definition to be available before they can generate mappings. In particular, details relating to primary keys and foreign keys are needed but are not always available. The
2.2 DATA INTEGRATION CHALLENGES

generated mappings may not be accurate and at the same time may not address the information needs of the user. For this reason, several refinement processes might be needed to deal with the accuracy of mapping. Clio refines mappings through interactions with the users. Through user feedback, users can choose the appropriate views. However, when the source and target schemas are ill-defined, or where information on the primary or foreign key is not available (a frequent occurrence in independent, such as data lakes), other information such as inferred profiling data can be used, as proposed in work by Mazilu et al. [MPFK19]. The authors proposed a mapping generation algorithm that takes advantage of automated data profiling to infer information about sources and their relationship.

Motivated by the test examples approach in debugging computer programs, some work has explored the use of data examples for schema mapping [ACKT11]. Early work on schema mapping with data examples has been done by Yan et al. [YMHF01], who use source data as examples. In their work, the authors define the concept of mapping examples and a mapping operator, used to exploit examples. Another recent work on data examples is the work by Alexa et al. [ACM+08], in which data examples are used to assist the expert user to understand, design and refine a mapping specification. Further work by Alexa et al. [ACKT11] uses data examples to characterise schema mappings uniquely. The authors also explore the potential and limitations of data examples to better interpret and understand schema mappings. Another recent proposal is by Bonifati et al. [BCCT19, BCCT17], who leverage exemplar tuples provided by non-expert users to determine the underlying mappings. The authors also propose an interactive schema mapping framework that would produce more general mappings that closely comply with the users’ requirements.

2.2 Data Integration Challenges

One of the main challenges in the traditional data integration process is the high up-front costs required to build and maintain a fully-functional data integration system. Doan et al. [DHI12] claimed that data integration is challenging, and this is supported by Hedeler et al. [HBF+09], who emphasise that the difficulty of data integration lies in specifying and determining the correct mappings. Also, Halevy et al. [Hal01] stated that during initialisation of the data integration process, specifically the schema mapping task, the identification of the semantic relationship between a source schema and global schema is difficult and time-consuming and thus is one of the main obstacles to
data integration set up.

To minimise uncertainty and determine relevant records from the mapping result, schema mapping for data integration is often carried out manually by a domain expert—someone who has knowledge of the application domain and understands the business rules, user requirements and semantics of the mapping. Knowledge of the application domain is essential in writing and maintaining the semantic mapping between the source data and the global schema. Moreover, what makes mapping more challenging is the frequent changes in data sources or other exceptions or specific cases such as amendments in business rules or shifts in user requirements which also trigger the need for changes in the mappings. Traditional data integration approaches are insufficient in coping with the changing environment (such as change of schemas) due to the lengthy process of manual work required before the data integration system becomes useful, which may cause the mapping that has been determined so far to become outdated.

Figure 2.1 provides an example of a typical data integration scenario in the hospitality domain. Consider a user posting a query over the user-interface of the data integration system, looking for a “Hotel room in Manchester with a rate of not more than £200 per night”. To obtain an answer to such query without the data integration in place, the user needs to pose queries on several heterogeneous and independent data sources such as on Booking.com, PremierInn.com, TravelLodge.com and possibly several other hotel directories. With the data integration, the query is posed once over the global schema. Through semantic mapping, this query will then be reformulated to be translated into a set of queries that correspond to the source schemas. As presented in Figure 2.1, each data source is bound to a wrapper. Interaction between data sources and the data integration system is made possible through this wrapper. A query posted by the user from the data integration system (possibly through the system’s user interface) is forwarded to data sources via the wrapper. One of the wrapper’s duties is to transform the query results from the sources into a format manageable by the query processor [DHI12].

For a data integration application to retrieve the query results, the connections between schemas of the sources and the global schema play an important role and need to be established beforehand. Mappings stipulate ways to reconcile the discrepancies between data value representations and formats that appear in different sources. Since the global schema, specifically the data records, are not physically stored in the data integration system but instead rely on the mapping specification defined during the schema mapping phase, the integration is made “virtually” through the global schema.
2.2. DATA INTEGRATION CHALLENGES

acting as the “view”. It is also important to note that, if there is a growth in the number of data sources that need to be integrated, this will result in a high number of returned records as query answer sets. Consequently, a large number of data sources leads to uncertainty, since the search space in identifying the correct mapping candidate is now amplified [DHI12, Hal01]. As a human expert is needed to identify the most appropriate mappings out of all possible mappings between a set of source schemas, this can be a delaying step and becomes another of the obstacles for data integration systems. Because human labour is usually expensive, thus requesting experts to manually specify the correct mapping from an extensive collection of possible mappings is not feasible.

Furthermore, before any analysis can take place, data from the heterogeneous sources needs to be cleaned and transformed into a uniform format. Cleaning and transforming data is a fundamental element of the data integration process as these tasks deal with incorrect, missing and inconsistent values in data from multiple sources.

Cleaning and transforming data is also an iterative process, as some errors in data can only be detected after several rounds of cleaning and transformation. For instance, outliers in the data set are detected only when missing values have been replaced. Raman et al. [RH01], claimed that data cleaning and transformation is made up of three components: (1) data auditing, to detect errors in the data, (2) transformation selection, to fix the identified error and (3) applying transformations on the data set. The authors also stated that the tasks of auditing, correcting and transforming data to the format

![Figure 2.1: Example of data integration application in hospitality domain](image)
needed for analysis may require significant user effort since user knowledge is essential specifically in verifying the data repairs, especially when there are no reference or gold standard data sources available to measure the accuracy of detecting and repairing data errors.

Today, many tools have been developed in an attempt to handle the labour-intensive effort of data cleaning. Some of data cleaning systems include KATARA [CMI+15], Holo Clean [RCIR17], Tamr System [SBI+13], NADEEF [EEI+13]. There are also existing systems that focus on data transformation such as Trifacta Wrangler [HHK15, KPHH11a], Talend [Tal09], Open Refine [VDW13], Foofah [JACJ17b], FlashFill [SG16] and BlinkFill [Sin16], to name a few. Data transformation is the task of transforming data from one format to another. Despite many tools available to support transformation work, human effort is still required specifically in specifying the transformation rules. Manual transformation by the expert user may lead to an expensive data transformation task, especially when it involves large data sets from many sources.

Ilyas argued that much existing research in data cleaning concentrates on techniques and algorithms for discovering specific types of data error, and that only a small portion of these solutions offers techniques (either automatic or semi-automatic) to fix the discovered errors [Ily16]. The author also pointed out that the disconnection between error detection and error correction is one of the problems that need to be addressed, since the issue is closely related to the user's involvement in data cleaning solutions because user expertise is usually needed, especially in guiding the data cleaning solution and verifying data repairs. Even with tool support, the task is still time-consuming, especially when the sources are diverse, extensive and contain many points of inconsistency.

### 2.3 Dataspaces Approach

A more recent data integration approach, known as dataspaces, seeks to distribute the high upfront costs of data integration system creation to the other stages of the data integration process. To accomplish this goal, dataspaces gather feedback from the user on selected artefacts of the dataspace. The feedback supplied by the user can be explicit or implicit. The approach carried out to gather feedback in dataspaces is done in a pay-as-you-go manner, where the user continuously supplies feedback to the data integration system to obtain a better quality data integration.
2.4 Challenges in the Dataspace Approach

Because of this, a dataspace is also capable of accommodating a constantly changing environment (e.g. changes to source schemas, additions and deletions of sources) enabled by a user-feedback approach performed incrementally in a pay-as-you-go manner [FHM05]. The idea is to start the initialisation of dataspaces at the earliest opportunity. Therefore, in dataspaces, the initialisation of integration is often performed automatically (or semi-automatically); the integration is automated on a best-effort basis. This results in the earlier completion and operation of a data integration system with lower upfront-cost at the expense of quality. The integration quality is then improved gradually using user-feedback gathered throughout the entire lifespan of the dataspace. Figure 2.2 provides an overview of the elements involved during initialisation of a dataspace, as proposed by Hedeler et al. [HBP+10].

Because of the automated semantic integration of dataspaces, partial semantic integration is sufficient as long as it fulfils the requirements and purpose of the integration at that particular time. However, over time, as the dataspace management system gathers and acts on user feedback, the integration will be refined to provide better integration quality according to the user feedback supplied by the user or domain expert. For example, user feedback can improve schema design and mapping, sources selection or the formatting of attribute values.

![Figure 2.2: Automated initialization of dataspace [HBP+10]](image)

2.4 Challenges in the Dataspace Approach

Like traditional data integration, the dataspace approach also has its challenges. Studies by Mirza et al. [MCC10] point out four difficulties surrounding the dataspace setting. The problems specified by the authors were based upon the original work on dataspaces proposed by Halevy et al. [HFM06]. As pointed out by Mirza et al. [MCC10], the challenges are (1) schema heterogeneity, (2) indexing of loosely-coupled data collection, (3) querying across data models and, (4) need for user feedback. The authors stated that most of the work on user feedback in data integration focused primarily
CHAPTER 2. LITERATURE REVIEW

on one type of integration task such as the work by Doan et al. [DDH01], where the authors used user feedback to assist in schema matching. In contrast, work by Wu et al. [WK15] mainly focuses on an active learning method which supplies feedback from the user to the system to train a classifier. Therefore, Mirza et al., argued that in obtaining an improved query result, it is essential to combine user feedback gathered from multiple data integration tasks.

Other relevant studies that indicate dataspace challenges are by Maskat [Mas15] and Singh et al. [SJ11]. In general, Maskat and Singh et al. list challenges comparable with the ones presented by Mirza et al. Maskat’s study reported that a dataspace has to cope with difficulties that lie in three main areas: data sources and semantic relationships, queries and user feedback. Singh et al. [SJ11], suggested that dataspaces need to deal with issues such as data modelling and querying, local storage and indexing, and dataspace discovery as well as re-usability of human attention or, in other words, user feedback. Based on the challenges mentioned in these studies, it would seem that the challenges in a dataspace are around the heterogeneity of its data sources, query-processing mechanisms and the application of user feedback. We briefly discuss each of the obstacles that cause difficulties these areas below, supported with evidence from the literature.

• **Uncertainty in Heterogeneous Data Sources**
  An essential task in data integration is to identify data sources that can meet user requirements. For example, we might need sources that could allow us to find the best school in England or sources could identify a hotel room in London with an average price below £100 per night. Bootstrapping of a new source into the dataspace is done automatically. The semantic relationships derived from these automatic methods are created based on a best effort basis, and are hence approximate. This uncertainty could have significant implications, such as causing the selected sources to only poorly fulfil the user’s needs for the task in hand. Deciding which sources to choose for the integration system depends primarily not on the schemas of the sources but also on their other characteristics and the requirements of the database user, whether these are correctness, completeness or freshness of data, to name a few examples.

Moreover, source selection becomes more complicated when many sources are involved, which gives rise to semantic relationships that are unknown. Also, in some cases, not every semantic relationship is required in an attempt to offer services that are of user’s interest [MCC10]. Therefore, it is vital for dataspaces
2.4. CHALLENGES IN THE DATASPACE APPROACH

To automatically discover data sources that contain only information that is relevant to the user’s needs. Aboulnaga et al. [AEG07], argued that mapping data sources to a defined global schema might not be the best approach, particularly for large-scale data integration. As a result, the authors proposed a pay-as-you-go approach to data source selection where feedback on search results is utilised to co-relate the new and current data sources. The feedback assists in gathering information about the selected sources and how much they manifest the requirements of the user so that sources with less significant impact on the outcome can be used. This is done by asking a domain expert or a non-expert user of the data.

A further challenge is that managing changes in data spread across many databases can cause problems for the user as they seek the data they need based on their requirements and interest. Feedback does help to reduce uncertainty about the quality of data sources and because the dataspace needs to accommodate ongoing changes in multiple data sources. At some point in the lifetime of the dataspace, the global schema will likely require modification so that it can fit into the latest data sources that have changed. Even with feedback, in some cases, it is still difficult to select the most relevant sources given that sources for the integration may be large with many characteristics. Furthermore, because users may have different criteria regarding the integration result they need, this may result in a costly feedback approach. Although there has been some work on selecting sources in dataspaces in a cost-effective way, further research is needed to determine new ways of finding the most significant sources for the integration that meets the needs of the user in solving the task at hand.

- Reformulate Query Process

Although dataspaces can sometimes have a fixed target schema, not all have a one-size-fits-all schema that is suitable for all the queries that all users need to post. Thus, user interaction with a dataspace can be seen as a data exploration effort, particularly in terms of finding possible schemas and sources that allow the desired results to be obtained from the posed queries. Due to the non-uniformity of data models such as structured data (e.g., relational data model), unstructured data (e.g., email, audio and video files) and semi-structured data (e.g., CSV), queries in dataspaces need to be reformulated. Thus, a different form of query, such as structured queries, keyword queries or meta-data queries, need to be supported to provide an efficient query process. As a result, providing a powerful search and query mechanism in a dataspace and reformulating the query against
the mapped data sources remains a significant challenge in dataspaces. Query reformulation is more challenging when the query that needs to be reformulated is a complex query of a structured data model to be answered using data from unstructured or semi-structured sources.

- Soliciting User Feedback

In dataspaces, the integration is initially configured automatically on a best-effort basis and results in a system that runs with basic functionality and low (though for immediate needs, sufficient) quality. The intention is to encourage the user to supply feedback so the integration system can be improved over time through a pay-as-you-go approach. Feedback is also envisioned to be the primary mechanism by which the dataspace seeks to reduce the high upfront costs of setting up an integration system [FHM05]. Feedback is a key building block of dataspaces that helps to guide the formation of the integration and in some cases, assist in improving the quality of services the system provides. Existing proposals on feedback in dataspaces often target specific data integration tasks. For example, existing work seeks feedback to support the user in building integration queries [TJM+08]. However, gathering user feedback in a dataspace is not an easy task and may be expensive, considering there may be a high number of potential matchings (and/or mappings) that need to be evaluated by the user for the overall benefits of dataspace system. What remains a challenge in this context is to formulate the best approach so that, when there is a need to demand feedback from the user, the system should only include potential matchings or mappings in order of its benefits to the quality of the dataspace. Another challenge concerns the correctness of the feedback, as the quality of the user feedback may be uncertain since the degree of expertise and individual requirements of users who supply feedback may vary. For instance, the user may not be an expert in the domain of the integration or the user may only supply feedback on a small subset of tuples from a query result that may contain thousands of other tuples, making the feedback insignificant in terms of improving the overall performance of the dataspace.

2.5 User Feedback

This section describes two types of user feedback in dataspaces: implicit and explicit. In general, user feedback is information on a particular subject that depends on the
2.5. USER FEEDBACK

user’s knowledge, interaction and behaviour. Feedback can be gathered in a way that is either explicit or implicit. We define explicit user feedback as information gathered about the user’s domain knowledge and requirements by directly requesting it. In contrast, implicit user feedback is feedback gathered in a way that is invisible to the user, by observing the user’s interactions and behaviours on specific activities and inferring the feedback from that. Explicit user feedback is more obtrusive as it may interfere with the user’s time and duties, while implicit user feedback is more discreet and usually does not interrupt the user’s daily routines. More on this distinction will be described in the following subsections.

2.5.1 Explicit User Feedback

Prior work on dataspaces mainly focuses on explicit user feedback. Feedback can be given explicitly on objects such as matchings, mappings, queries or query answers. Such feedback is not restricted to identifying whether or not the specific object is appropriate or inappropriate, given the user’s needs, but it also can be gathered from tasks such as asking a user to supply a series of relevant constraints as well as seeking users to rank the results of tuple with ‘before’ and ‘after’ information [MPE12].

Jeffery et al. [JFH08], developed a decision-theoretic framework that seeks feedback from users to verify the ranking of candidate mappings to improve the query results. Focusing on schema mapping improvement, Belhajjame et al. [BPF+11], captured feedback on false positive and false negative query results. The feedback instances then are used to improve schema mappings to better meet users’ expectations. Both these studies focus primarily on how the feedback is used, and, in principle, the same feedback could be obtained through different forms of user interaction. Talukdar [TIP10a] developed a system called Q that allows the user to pose a query through an interface for data stored across multiple data sources. The system uses a keyword search from the posed queries and returns records of results generated over various alternative queries. The user supplies feedback by specifying the preferred order of the results based on their knowledge and requirements.

In contrast, Alexe et al. [ACM+08] focus on schema mapping in a tool called MUSE. MUSE uses an example-based approach to improve how well partially correct mappings meet a specific requirement. Using data as examples, MUSE takes into account the mapping requirement from the user as an alternative and distinguished it over the mapping process by asking the user yes/no questions to learn from user interaction with the proposed examples.
In addition to improving schema matchings and mappings, user feedback has also been used in query refinement, as proposed by Coa et al. [CQCS10]. They classified feedback into soft feedback (representing the user’s preferences) and hard feedback (representing requirements). The feedback is collected on path expressions over a semi-structured data collection. Algorithms were proposed to explore the space of candidate queries and to rank the results. In the same vein, Islam et al. [ILZ13], suggest a framework called FlexiIQ, which aims to improve the ratio of expected to unexpected query results by refining query conditions. In both these cases, end-users provide feedback as part of an iterative querying process.

Several teams have investigated the use of feedback for source selection. Ríos et al. [RPFB16, RPF+17], aimed to achieve cost-effective source selection by soliciting user feedback on targeted data items. Users are asked to annotate the items as true or false positives to show if the data items specify certain criteria. Feedback on the relevance of data items is used to estimate the quality of data sources in terms of their precision and recall, and feedback is sought to distinguish between candidate sources. Likewise, Yan et al. [YZI+15], focus on adding new sources and mappings in the context of keyword search over structured. Once again, feedback on specific items is sought: in this case, query answers where users can provide feedback by determining whether the answer is good or bad or in the form of ranking answers that they preferred. The feedback provided suggests whether tuples across the results set should be removed or re-ordered.

Ashraf and Karem, have proposed another approach related to the selection of sources [AEG07], in the context of a tool known as μBE. μBE uses an iterative exploratory process to guide the system in selecting the best data sources for the user and the best global schema for the source. This approach is achieved using feedback supplied by the user by iteratively solving a constraint optimisation problem formulated by the tools through a simple user interface.

One of the main advantages of using explicit feedback is that the feedback given by the user is expressed clearly; hence, confidence in the provided feedback is higher compared to the implicit feedback where assumptions are often needed in interpreting the user’s intention. One of the reasons for this is that feedback given explicitly by the user is based on some specific object, such as on mappings or matchings (for example, when users annotate tuples in query results with ‘true positive’/‘false positive’ labels). However, it incurs costs for the user for every piece of feedback given, including the time spent in providing the feedback needed. Therefore, it is crucial to only ask for
feedback when it is necessary. Hence the questions of what feedback to ask for, when to ask for it and on which objects feedback should be solicited require serious consideration, so that maximum benefit can be obtained from the feedback gathered and time and other costs spent gathering the feedback.

Table 2.1: Explicit feedback in the current proposals with their corresponding objects [BPF⁺11]

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Object on which feedback is given</th>
<th>Set of terms used for annotating objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa et al. [ACM⁺08]</td>
<td>an instance of a given schema and the instance obtained by its transformation into another schema</td>
<td>{‘yes’, ‘no’}</td>
</tr>
<tr>
<td>Belhajjame et al. [BPF⁺11]</td>
<td>a result tuple</td>
<td>{‘true positive’, ‘false positive’, ‘false negative’}</td>
</tr>
<tr>
<td></td>
<td>an attribute and its value</td>
<td>{‘true positive’, ‘false positive’, ‘false negative’}</td>
</tr>
<tr>
<td>Coa et al. [CQCS10]</td>
<td>a candidate query</td>
<td>{‘true positive’, ‘false positive’, ‘false negative’}</td>
</tr>
<tr>
<td></td>
<td>a pair of candidate query</td>
<td>{‘before’, ‘after’}</td>
</tr>
<tr>
<td>Chai et al. [CVDN09]</td>
<td>a view result tuple</td>
<td>{‘insert’, ‘update’, ‘delete’}</td>
</tr>
<tr>
<td>Jeffery et al. [JFH08]</td>
<td>a mapping</td>
<td>{‘true positive’, ‘false positive’}</td>
</tr>
<tr>
<td>Islam et al. [ILZ13]</td>
<td>a result tuples</td>
<td>{‘expected’, ‘unexpected’}</td>
</tr>
<tr>
<td>Talkudar et al. [TIP106]</td>
<td>a result tuple</td>
<td>{‘true positive’, ‘false positive’}</td>
</tr>
<tr>
<td></td>
<td>a pair of result tuple</td>
<td>{‘before’, ‘after’}</td>
</tr>
<tr>
<td>Rios et al. [RPFB16]</td>
<td>a subset of sources</td>
<td>{‘true positive’, ‘false positive’}</td>
</tr>
<tr>
<td>McCann et al. [MSD08]</td>
<td>a relation attribute</td>
<td>set of attribute data type</td>
</tr>
<tr>
<td></td>
<td>two attribute of a given relation</td>
<td>set of constraints</td>
</tr>
<tr>
<td></td>
<td>a match</td>
<td>{‘true positive’, ‘false positive’}</td>
</tr>
<tr>
<td>Yan et al. [YZI⁺15]</td>
<td>set of query answers</td>
<td>{‘correct’, ‘incorrect’}</td>
</tr>
</tbody>
</table>

Crowdsourcing can be viewed as another form of explicit feedback that provides a method for soliciting user feedback from a group of people who can provide a solution to tasks that cannot entirely be addressed through the automated approach [DRH11, Bra08]. Work by Maskat [MPE12] viewed crowd workers who engaged in crowdsourcing activity based on their interest either as (1) Stakeholders, for crowd workers who are interested in the content of the information, and (2) Non-stakeholders, for crowd workers whose main concern is the incentives offered. Usually, the incentives are in a monetary form.

In the context of data management, recent surveys [CFMP17, LWZF16] have studied existing proposals that used a crowdsourcing approach in addressing different data management tasks, including integration tasks such as matching and mapping [DDCM12, PF13, MSD08, ZCJC13, HTMA13], data cleaning [WKFF12, CMI⁺15, TCZ⁺14], queries processing [FKK⁺11] and knowledge construction [SLB12, AGMS13].
McCann et al. [MSD08], studied the potential of the collective voice among crowd workers as feedback for improving schema matching. The goal is to identify correct and incorrect matches based on the feedback supplied to several different questions generated by the proposed system. The system validates answers to the given question and determines schema matching results according to the feedback supplied. In other work related to feedback, a correctly formatted example is used to inform the synthesis of format transformation rules [KPHH11b, GHS12a]. The work focuses on the direct interaction of users with result values.

Similar to the key limitation of explicit feedback mentioned earlier, cost is one of the main limitations of crowdsourcing, since obtaining feedback from crowdsourcing requires payment to be made to crowd workers, just as feedback consumes valuable human effort. Besides, crowdsourcing has specific issues to consider with respect to the human involvement in the process of soliciting feedback, such as how to differentiate experienced from non-experienced crowd workers, as well as interest in the work and the accuracy of the information they provide. Because of the monetary aspect, it is also vital to identify the types of tasks and strategies that will give the most benefit from the crowd. Sizes of the recruited team of crowd workers also need careful consideration as it impacts the cost.

Due to all the reasons discussed, we argue that gathering feedback explicitly should not be the only way of requiring feedback from users and an alternative approach is essential to complement the limitation of explicit feedback. We believe this can be carried out through implicit feedback, an approach that is less obstructive and inexpensive, and we address this approach in Section 2.5.2.

2.5.2 Implicit User Feedback

Despite implicit feedback being a great complementary approach to explicit feedback, as useful information from users can be gathered at no extra cost, this area of study remains briefly addressed in the literature. One of the reasons may be due to its lower accuracy compared to explicit feedback [Nic98]. Existing artefacts of an integration system can be used to gather feedback implicitly. There are two ways for an integration artefact to offer information: first, from the information it carries, such as a phrase that exists in the clause of a query’s condition; and second, from within the artefact itself, such as tagging frequency [MPE12]. Another possible source of implicit feedback is the actions of end-user. For example, in the context of dataspace, if the user’s course of action on what they do with the dataspace and the data it provides could be observed
and learned, this could also be one way of gathering implicit feedback.

In data integration thus far, only a few examples of implicit feedback have been proposed. Work by Maskat et al. [MPE12], utilised query logs by extracting information from them to assist in ordering candidate mappings. Elmeleegy et al. [EEL11] proposed $U$–MAP, a system that generates schema mappings by extending an existing schema mapping technique by exploiting the metadata extracted from query logs. The information on system usage that exists in the query logs is used to determine mappings that are in line with frequently asked queries.

On the other hand, much of the work on implicit feedback for data integration relates to ranking, and there seems to be an opportunity to investigate how feedback that has previously been obtained explicitly can be inferred from other sources. For classification of implicit feedback, we extend the list proposed by Maskat [MPE12] but divide the implicit feedback into two categories. Table 2.2 lists implicit feedback proposals related to data integration and Table 2.3 focuses on proposals that are more general within the data management context.

Table 2.2: Implicit feedback in the current proposals of data integration with their corresponding objects within the scope of data integration [MPE12].

<table>
<thead>
<tr>
<th>Proposals</th>
<th>Artefact used to imply feedback</th>
<th>Part of Artefact used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elmeleegy et al.</td>
<td>query logs</td>
<td>transaction data</td>
</tr>
<tr>
<td>[EEL11]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maskat et al.</td>
<td>query logs</td>
<td>terms in a query’s condition clause</td>
</tr>
<tr>
<td>[MPE12]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thran et al.</td>
<td>list of entity description</td>
<td>description itself</td>
</tr>
<tr>
<td>[TMC12]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yakout et al.</td>
<td>transaction logs</td>
<td>transaction data</td>
</tr>
<tr>
<td>[YEN+11]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outside the scope of data integration, several forms of implicit user feedback have been proposed in data management, particularly in information retrieval. Joachims et al. [JGP+05], compare implicit feedback based on click-through data from query results against explicit feedback for assessing the relevance of web pages. The results show that, in this case, the implicit feedback can be challenging to interpret, but that it can be useful for inferring relative rather than absolute measures of relevance. Bullock et al. [NJH11], explored data tagging in improving relevance in retrieval of web documents. Other work that takes advantage of tagging activities is the work by Iofciu et al. [IFAB11], in which the authors investigated whether or not social media users can be identified based on their tagging practice. Radlinski et al. [RJ05], infer information on preference choices from search engine log files based on query sequences. Using click-through data, they introduce a technique to identify sequences of related queries and propose an algorithm that uses the preference choices to learn ranking functions.
for web search results. To overcome the diversity of results offered by search engines and recommender systems, Raman et al. [RSJ12], introduced an online learning model and algorithm. The learning algorithm uses implicit feedback in which the user reads documents to learn a diversified recommendation and retrieval function.

Table 2.3: Implicit feedback in current data management proposals (outside the scope of data integration)

<table>
<thead>
<tr>
<th>Proposals</th>
<th>Object used to imply feedback</th>
<th>Part of Object used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bullock et al.</td>
<td>set of tags</td>
<td>search queries</td>
</tr>
<tr>
<td>Iofciu et al.</td>
<td>set of tags</td>
<td>tagging itself</td>
</tr>
<tr>
<td>Joachims et al.</td>
<td>eyetracking and click-through</td>
<td>retrieved search result</td>
</tr>
<tr>
<td>Radlinski et al.</td>
<td>tagging data</td>
<td>search queries and ranking of search engine</td>
</tr>
<tr>
<td>Raman et al.</td>
<td>click-through</td>
<td>retrieve documents</td>
</tr>
</tbody>
</table>

As illustrated by these examples, implicit feedback has the potential to be beneficial, since users are not required to perform any additional tasks and since the information it can provide is potentially significant. However, to the best of our knowledge, the implicit feedback approach has been only investigated to a preliminary degree in data integration settings. As far as we know, no previous research has considered manual editing work carried out by users on data returned from a data integration as form of implicit feedback. To extend our understanding on whether or not manual correction as implicit feedback to overcome the challenges of gathering feedback so feedback could be gathered at no extra cost and without expecting additional effort from the end-user further studies on the topic are desirable, specifically in PAYG data integration.

Because implicit feedback could be interpreted in many different ways, imprecision in implicit feedback is a challenge depending on the user’s interaction with an integration system. Therefore, further investigation is needed to observe, determine and interpret the information extracted implicitly from the user’s interaction with the data set.

### 2.6 User Feedback in Data Transformation

Even with advanced data preparation solutions, data scientists still spend a substantial amount of time diagnosing problems with data quality and manipulating and transforming it. Surveys have established that data scientists typically spend up to 80% of their time on data preparation. In this section, we review existing work on supporting

---

tools for cleaning and correcting integrated data, focusing primarily on *user involvement in data transformation*, but also considering a smaller set of results relating to *user feedback approaches to data cleaning*.

This section aims to describe different ways of handling data transformation in an integrated setting. We divide this section into three subsections that cover existing works on format transformation, the part of the data transformation task closely related to the research contribution presented in Chapter 5. In the subsections, we present work related to data transformation in general and further describes the usage of user feedback in the context of data cleaning and transformation.

**General Data Transformation.** Concerning *user involvement in data transformation*, an important body of related work involves Potter’s Wheel [RH01] and its successor Wrangler [KPHH11a]. Potter’s Wheel is a menu-based system that allows users to directly construct and manipulate a sequence of data transformation operations on a subset of data, which can then be applied to the rest of the data or newly arriving data. Building on the transformation language of Potter’s Wheel, Wrangler is an interactive data cleaning system that is capable of suggesting candidate transformations to the user through an automatic inference mechanism. Wrangler seeks to reduce the time taken to manually specify data transformations using a learning-based approach and predictive model [HHK15]. Through its graphical user interface, the user interacts directly with a tabular view of the data in Wrangler. A similar approach is followed by OpenRefine [VDW13], formerly GoogleRefine [HM11]. OpenRefine allows a user to specify some aspects of their intended result graphically, although most of the user’s intention still needs to be expressed through a command language.

With the same goal, which is to reduce hefty data wrangling costs during integration, Talend [Tal09] is another system that allows the user to interact through its graphical workspace to improve the quality of integrated data. Talend enables the user to perform validation using a set of standard validation patterns and to execute data profiling tasks such as schema analysis and table analysis. It also supports the generation of tables, graphs and data sets for various range of data type. In Talend, a pattern is defined as a set of strings where the user can specify its content, structure and quality. Talend offers two types of pattern: (1) regular expressions or REGEX, a predefined regular pattern prepare by the user to search specific strings to match complex patterns of data content. For example, if users need to check if the National Insurance Number is valid and follow the correct pattern, they can redefine their expression to match the data with the pattern they want. and (2) SQL patterns: a personalised pattern used in
SQL queries and often involving a *LIKE* clause and *wildcard* sign.

Transformation in Talend can be done through a processing component known as a *tMap*. As a core component for transformation in Talend, tMap is used to map one schema to another as it transforms input field to output field. tMap also transform data from single or multiple sources to single or multiple destinations. Through tMap, users specify the *mapping expression* that they require, such as mapping of Postcode format to *uppercase* or to transform Date format from “yyyy/mm/dd” to “dd-mon-yyyy”. The can also be validated against data from the profiling process [Nau14].

Another tool comparable to Talend is Pentaho Kettle [Pen18], also known as Pentaho Data Integration, which is an ETL tool that builds upon several sub-applications and does a lot more than just Extract, Transform and Load processes. Among PDI’s other intended purposes is data cleaning, migrating data between an application and sources, and exporting data from one file format to another, just to name a few. *Spoon* is a Kettle component that allows the user to design data transformation jobs from combinations of built-in functions using a drag-and-drop interface. The user can choose from a wide range of data input-output source types, including text file formats, database tables and datasheets. Kettle’s transformation engine is called Pan, a wrapper for integration engine of Kettle built to execute a command-line program for launching transformation jobs that can be run in an XML file directly from its repository. In Kettle, the transformation feature is limited to 26 individual transformations. and 15 lookup methods, including database queries, file system operations and web service calls. An example of transformation in Kettle is transforming *FirstName* and *LastName* columns into one column called *Name*; this can be done by concatenating the two columns, *FirstName* and *LastName* through some regular expression. Also, in Kettle, regular expressions can be used for validation and transformation purposes. Hence some technical knowledge is required in handling Kettle tools.

**Format Transformation.** Under this section, we will focus on existing tools developed to deal with format transformation specifically. One of the reasons to discuss format transformation tools rather than existing research literature is because the field has now advance and mature where many of the research prototypes have now appeared in commercial and also available as open-source tools such as Flash Fill [Gul11], which is now one of the feature available in Microsoft Excel 2013.

While the tools listed so far support the user in performing general data transformation tasks, a small number of tools and approaches have been proposed that specifically focus on the problem of *format transformation*. Flash Fill is a tool built into Microsoft
Excel that learns syntactic string transformation programs from example pairs provided by the user [Gul11]. The user provides examples of data values of the kind that appear in the data set but expressed in an incorrect format (e.g., ‘Barack Obama’) and gives the desired formatting for each one (‘B. Obama’). Flash Fill applies a set of binary classifiers to the example pairs, looking for one that matches the incorrectly formatted member of each pair. Because multiple classifiers are applied, Flash Fill is capable of handling several formats at a time, in the same column of data [Gul11, GHS12b]. Briefly, Flash Fill first learns data transformation rules from the supplied examples, and then it applies the learnt rules to the data set on which the data scientist is working. The user fixes a small portion of the data set manually, and then only has to check over the fixes Flash Fill has applied for the rest of the data, saving considerable time.

Synthesis of rules in Flash Fill begins with learning traces, and then inferring loops and conditionals from these traces of input-output examples (rather than trying to learn the whole rule in one go). The algorithm creates a directed acyclic graph (DAG) from the input-output examples traces. This can be used to represent a huge collection of transformations. Conditionals are used to allow a wide range of formats to be considered for a single column in the source spreadsheet. Flash Fill first partitions the set of examples and the uses a Boolean classification approach based on scoring functions.

Because Flash Fill operates only on the example pairs given, these examples need to cover all formats present in the input. A robust set of transformation rules has to be learnt from the examples, though it may not be able to transform all data to the desired format. This explains Flash Fill’s limitations in transforming long and complicated strings. Various extensions to Flash Fill have been developed to increase its ability to learn correct transformations from small amounts of user input.

In Flash Fill, example pairs provided may be under specification of the intended task and this cause a large number of programs conform to the example given. Therefore, finding the desired programs among the large search space is difficult and result in scalability issues on the synthesis algorithm of Flash Fill. To improve Flash Fill’s limitations in scalability issues of the synthesis algorithm, BlankFill [Sin16] takes the Flash Fill approach one step further by minimising the search space of ambiguous choices experienced by Flash Fill. This is done through a semi-supervised learning strategy which leverages the logical representation that exists in the spreadsheet data. In contrast to Flash Fill, which focuses only on the input-output example pairs and neglects other data items contained in the spreadsheet. Singh et al. hypothesised that the
CHAPTER 2. LITERATURE REVIEW

use of supplementary data such as providing given input-output examples of “Mumbai, India” → “India” and also additional example of “New Delhi, India” → “India” is essential during rule learning, in addition to the example pairs [Sin16], while Flash Fill and its successors synthesise programs based on positive examples. Wu et al. [WK16] are able to efficiently obtain strong conditional statements for data transformation by taking advantage of user interaction with a programming-by-example (PBE) system. The proposed approach acquires the necessary domain knowledge by requiring the user to examine at least one possible incorrect pair supplied by the machine learning technique, thus obtaining positive and negative examples.

Another data transformation system that is similar to Flash Fill is Foofah [JACJ17a], which aims to reduce user effort and enable naive or/and non-technical users to synthesise real-world data transformation program, focusing primarily on unstructured spreadsheet tables. Like Flash Fill, Foofah also requires example pairs to be specified by the user. It generates consistent transformation scripts based on a heuristic search.

All the above proposals require explicit input from the user, to provide the examples and other information needed. The costs and delays involved in obtaining sufficient examples from users have been addressed by Bogatu et al. [BPF17]. In their approach, Flash Fill is used for transformation program synthesis, but examples are obtained from existing rather than from end-users. Candidate example pairs are identified by searching for existing that align with the data to be transformed and contain values in the desired format.

Data Cleaning. Format transformation is a form of data cleaning since it aims to make the data more fit for purpose. Proposals for the use of feedback to support data repair (rather than format transformation) include KATARA [CMI+15], Guided Data Repair [YEN+11] and Continuous Data Cleaning [VCSM14]. All of these make use of user feedback approaches to data cleaning, as we will now briefly summarise.

KATARA [CMI+15], an end-to-end data cleaning system, adopts crowd-sourcing to collect user feedback, whereas Guided Data Repair employs an active learning approach in which the user feedback is acquired through an iterative process. Continuous Data Cleaning, proposed by Valkovs et al. [VCSM14], is a framework for adaptive and constant data repair guided by a classifier that takes advantage of user involvement during data repair, where the previous repair made by the user is used as feedback to learn repair preferences. The classifier uses user feedback to suggest more acceptable repairs in the future. KATARA, Guided Data Repair and Continuous Data Cleaning all involve explicit feedback collection, and thus the expenditure of valuable human effort.
As the volume of available data increases and our dependence on data analysis results grows, ways to achieve the same data cleaning effect with less human involvement are much in demand.

## 2.7 Feedback for Data Refresh

In this final section, we review the related work connected with the third contribution of this thesis: user feedback in the context of handling data refresh challenges. In relation to our proposed work, data refresh deal with changes observed between current and previous version of data set after user done correcting the data set manually. The aim is to extract and reapply the gathered changes to fix the same mistake in the future (when subjected to the same data sets) as data sets usually undergo many forms of change over their lifetime. Changes in data can be viewed as a dynamic process that occurs for various reasons [CSMM07, CSP05]. Because the world changes and the data set needs to change to keep consistency with it, thus the portion of the world about which we can and need to store data also grows. For instance, data change may occur due to corrections made to incorrect data values or because business requirements have now shifted, or perhaps the company have now downsized.

Today, most databases and other data management tools come with built-in features capable of tracking and reacting to changes in data. In a data warehouse and data integration context, changes to data sets can be tracked down through Change Data Capture functionality (CDC, for short).

CDC is a set of software processes that capture changes made on data sources so necessary action can be taken on the changed data. Often used by a data warehouse, CDC mechanisms have been implemented in most databases and data integration tools, such as Microsoft SQL-Server and PostgreSQL, and off-the-shelf integration tools, such as Talend. There are several CDC mechanisms, including snapshot differences [LGM96, DZ15, CGM97], log-based approaches [RD00, MY15, SBLY08], triggers on tables [VMZ+13] and timestamps [BBK+18]. Work proposed by Labio et al. [LGM96], and further extended by Sudarshan.S.C et al. [CGM97] focused on snapshot differences. **Snapshot differential** is an approach that compares the current snapshot of the source with it previous version. Detection of differences between the two snapshots of current and previous versions is known as snapshot differential.

It is important to note that our contribution related to data refresh challenges, as described in detail in Chapter 6, focuses on snapshot differences in identifying changes.
between the current and previous version of data sets.

In [LGM96], the author proposed a simple algorithm to detect changes in a data set which works well on large data sets in the context of data warehousing. The algorithm proposed by Labio is built using a traditional join-merge method. The work is further elaborated by Chawathe et al. [CGM97], where the author aims to detect changes on more challenging data such as on nested-object of hierarchical data rather than on flat-file or relational data. For better detection of changes and to interpret changes in a semantically meaningful way, the work is established on the traditional insert, update, delete operation and on an operation that moves subtree of a node, copy of it and entire subtree.

On the other hand, Du et al. [DZ15] argue that algorithms based on sort, merges and joins, such as the one proposed by Labio [LGM96], is vulnerable to error and not so useful for differential snapshots, especially when computing resource is limited. They proposed a more efficient approach by cross-breeding the query summary technique of MySQL, an open-source database, with the parallel programming technique of Hadoop MapReduce.

Ram et al. [RD00], introduce a method of extracting changes to the data at the source systems. in a data warehousing setting, called Op-Delta. Their proposed approach attempts to capture deltas (or changes) in data through an operation that cause the changes. Since the method proposed is a log-based approach, no downtime should be experienced during the maintenance process. Despite claims that Op-Delta is effective on large and competitive in terms of cost, this technique requires constant modification of the integration artefacts, which does not fit the dataspace context and is likely to be inconvenient in any setting.

Ma and Yang proposed a log-based CDC approach for document-oriented NoSQL databases by using a schema-free document compiled into a cell state model using MapReduce [MY15]. The cell state models represent changes to data using a copy-modify-merge technique that allows all change revisions to be easily retrieved. Shi et al. [SBLY08] introduce a framework for CDC in the context of a real-time data warehouse. Their work introduces a mechanism to process changes and improve the quality of changed data through scheduling strategy of captured data.

Although most current works deal with changes between data sets extracted directly from data sources with no chance of human involvement, the CDC approach offers an opportunity to extract changes from the manually corrected data set, which

---

2https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist.
could then be used to gather feedback. One recent proposal that is in-line with our goals is Data Diff, a tool implemented by Sutton et al. [SHGC18]. It aims to assist data scientists by minimising the repetitive nature of data cleaning during data analysis tasks. Through Data Diff, multiple versions of the data set generated over time are compared with each other to derive changes or Diffs. Changes in distributions and data transformations are extracted and then used to “patch” inconsistencies in data format and incompleteness in the data set. The “patch” helps uncover changes and clarifies what type of changes occur; hence, a data scientist can make informed decisions and reduce the time spent in wrangling data to make it appropriate for analysis.

2.8 Conclusion

In this chapter, we have reviewed the work within the scope of our research. We have presented the data integration context and identified several issues in building an integration system, including showing how the concept of a dataspace can address them. Due to the focus of our own research, we paid particular attention to the user feedback aspect of dataspaces and examined related work on different approaches to gathering user feedback. We also described work that underpins our research proposal, discussing the advantages of user feedback in data preparation tasks and tools, with specific reference to user feedback in handling data format transformations. Finally, we review work on capturing changes to data, and how these changes can provide a form of user feedback for tackling certain data integration problems.

To sum up, several dataspace improvement tasks require user feedback, but existing work mainly focuses on explicitly captured feedback. In practice, gathering user feedback can be an expensive approach due to the need for (often costly) human involvement. As a result, although the user is not obligated to provide feedback, the quality of the dataspace may not be improved and may even deteriorate. Whereas in data cleaning and transformation, existing proposals and tools mainly focus on explicit feedback of various forms such as through annotations, user rating, ranking and validation and crowdsourcing, to name a few. However, explicit captured feedback as well as costly is also intrusive as it requires time and effort from the users. In response to these shortcomings, implicit feedback could be an alternative approach to address the challenges in gathering feedback in dataspace and other areas in data management tasks such as data cleaning and transformation.

We note that, thus far, there is no framework for user feedback in dataspaces that
we could observe and learn from. In the next chapter, we proposed such a framework for implicit user feedback based on observing and analysing manual corrections made by end-users, which serves as the groundwork to the proposed approach.
Chapter 3

Inferring Feedback from Manual Correction

“We all need people who will give us feedback. That’s how we improve.”

Bill Gates

In the previous chapter (Chapter 2), we highlighted the challenge of obtaining user feedback in dataspaces. We analysed existing techniques and tools for gathering user feedback and demonstrated the need for an approach to feedback collection that is inexpensive and less intrusive. Considering that the data landscape in which a dataspace exists often changes as data accumulates over time, feedback may not remain useful for long. Thus, low cost feedback mechanisms are required, so that a constant supply of helpful feedback could be acquired to steadily improve the quality of integration offered by the dataspace as the requirements for and constraints on the dataspace evolve. Through the continuous supply of feedback, the need to implement a new integration from scratch whenever the data or requirements change can be avoided.

Additionally, the number of heterogeneous data sources within a large and complex organisation can also make the gathering of feedback challenging, given that large numbers of data sources usually results in a high amount of candidate mappings, which in turn leads to uncertainty in the data integration. For instance, identifying an exact mapping between data sources may be difficult for some domain areas such as bioinformatics, given that biological data sets are often diverse and geographically distributed, depending on the data access method, dissemination formats, structure and functions.

In many cases, uncertainty in integration requires knowledge from domain experts.
In practice, however, it is impractical to expect an expert user—in this case, a data scientist—to continually provide feedback on every data integration issue, when their efforts and knowledge could be more wisely used in other critical areas, such as devising models and algorithms. Besides, the never-ending process of feedback gathering could result in the user opting to stop providing feedback altogether. Thus, a good strategy is essential in knowing what to ask for as feedback, when to ask for it and in which order. Ideally, feedback will be extracted without a substantial cost on the user’s time and effort. Approaches capable of gathering feedback from the user at a reduced cost are opportunities worth considering. In this thesis, we examine one such approach: the extraction of feedback implicitly from corrections made to the integrated query results, by an expert user, to prepare the data for use.

In this chapter, we set out a road map for our exploration of whether there is merit in this idea. From the existing work summarised in Chapter 2, we know that other forms of implicit feedback have shown promising results in improving integration quality. In practice, implicit approaches to gathering feedback have proved to be inexpensive and able to be carried out in a less intrusive manner compared to explicit feedback. This lies in the fact that users are not expected to put additional time and effort into providing feedback to improve integration quality. Instead, the feedback is collected indirectly, by using other external resources such as log-files [MPE12], or by otherwise observing and learning from the user’s actions and behaviour. As we argued in in Chapter 2, to the best of our knowledge, no work has been conducted to implicitly gather feedback from manual corrections. In this chapter, we analyse this as a general notion and set out the plan for testing this approach, to access its success or otherwise in a dataspace context.

Specifically, in Section 3.1 we discuss the notion of manual correction in dataspaces, and the general framework of manual correction. We then show how our approach can be embedded within a dataspace setting in Section 3.2. A definition of the manual correction approach is also given in this section along with example scenarios from an integration problem. We provide several potential usages and benefits of manual correction across different types integration problem. Finally, Section 3.3 concludes the chapter.
3.1 Framework for Manual Correction

We define manual correction as:

\[ \text{A process in which the user of a dataspace makes changes to a data set produced by the dataspace to correct or refine it to allow completion some specific task using that data.} \]

We begin by considering the mechanisms by which manual correction can be incorporated into the standard dataspace architecture. We propose a common architectural framework for a range of applications of manual corrections, which is also one of our contributions to this research project.

To provide a clear idea of how our manual corrections can fit into the architecture of a dataspace, we direct the reader’s attention to the example we briefly described in Section 1.3 and use this example to describe a broken feedback loop and how we could reconnect it.

3.1.1 Broken Feedback Loop

In the examples we described in Section 1.3, we highlighted the problem that is often encountered while gathering feedback from users in a dataspace setting. We here present this problem scenario in more detail, as our running example for this chapter.

In this running example, we consider a data scientist’s task in providing feedback to a dataspace in an attempt to improve the quality of integration, so the result meets her information needs. The data scientist needs to prepare a report on school performance around Manchester city centre for a school board meeting. She is working with a dataspace to obtain the data she needs to carry out the analysis. She executes queries to retrieve, amongst other data, the headteacher’s name and the school’s name, phone number, address and postcode for the schools of interest. The dataspace generates a query result as shown in Table 3.1. But this data is unsuitable for the task in hand. For instance, the values of the phone number attribute, for Row 3 and Row 4 are inconsistent with the rest of the records, since they lack the country code—information which is required by the data scientist’s task. The same goes with the inconsistent format of the postcode attribute of Row 3. Another example is in the headteacher’s name; the data scientist is expecting the name to be in the form initial, dot, surname, such as A. Anderson, instead of Anne Lynn Anderson or A. Lynn Anderson. In addition, she also realised that the value for school name of Row 6 is inaccurate. Based on her
knowledge, the correct name of “Abbott Primary” is “Abbott Hey Primary”, a fact that she confirms through verification against external sources. Determined to make the dataspace query answer accurate and meet her information needs, she begins to provide feedback to the dataspace with the expectation that the feedback she supplies will eventually improve the query answer.

Table 3.1: Query result from the dataspace

<table>
<thead>
<tr>
<th>RowID</th>
<th>HeadTeacherName</th>
<th>SchoolName</th>
<th>PhoneNo</th>
<th>Address</th>
<th>Postcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>New Moston Primary</td>
<td>+441616813321</td>
<td>Moston Road</td>
<td>M41 3JQ</td>
</tr>
<tr>
<td>2</td>
<td>Anne Lynn Anderson</td>
<td>Ladybarn Primary</td>
<td>+44 212 321 4433</td>
<td>Ladybarn Lane B</td>
<td>M20 4ST</td>
</tr>
<tr>
<td>3</td>
<td>Dean R Ross</td>
<td>St John Primary</td>
<td>07434212234</td>
<td>Claire St</td>
<td>m192kk</td>
</tr>
<tr>
<td>4</td>
<td>Julia Brown</td>
<td>St Mary Primary</td>
<td>1314342768</td>
<td>Thorn Rd</td>
<td>M2 3FL</td>
</tr>
<tr>
<td>5</td>
<td>D.Amy Ann</td>
<td>Park View Primary</td>
<td>+44 222 321 3121</td>
<td>Springview Rd</td>
<td>ML7 4RR</td>
</tr>
<tr>
<td>6</td>
<td>M.Graham</td>
<td>Abbott Primary</td>
<td>+44 161 681 3321</td>
<td>Maine Rd</td>
<td>M40 6PP</td>
</tr>
</tbody>
</table>

Suppose that the gathering of feedback for the scenario above is performed explicitly, and is driven by the dataspace. The data scientist is expected to give “right” and “wrong” annotations on specific data values picked out by the dataspace. Despite responding to many such feedback-related questions, the query result may remain unchanged or far from her expectations. Technically, it is natural for a feedback gathering process to go back and forth between the dataspace and data scientist in an iterative cycle before the dataspace can meet the user’s information needs. One of the reasons is that the dataspace often uses feedback given by the data scientist to improve its underlying structure, such as reordering candidate mappings or restructuring the mapping between data elements. Thus, feedback given by the data scientist may not be reflected directly in the query answer, even though some improvements have taken place.

However, since no explanation may be given on what changes have been made in improving the quality of the dataspace, this results in the data scientist having no way of knowing what is being improved within the dataspace based on the feedback she provides. Explanation is seen as an integral part of human behaviour in understanding the surrounding world, especially in justifying choices and action. The number of data integration systems providing explanations (for example, a causal explanation, that determines how and why results were obtained from the feedback given) is still small [WHM18]. Lack of explanation on how the feedback improved the query results may discourage users from continuing to provide it, since users are unable to understand or relate to the query results based on the feedback they supply. And, in the
long run, the opportunity for the dataspace to improve its integration quality may be compromised by the data scientist’s poor user experience.

Returning to the scenario described earlier, discouraged with the query results from the dataspace, despite providing feedback to all feedback-related questions given to her, the data scientist finally decides to stop providing feedback to the dataspace. Instead, she makes use of the best query result the dataspace has to offer and improves the query result herself, manually, using off-the-shelf editing tools. The feedback loop that existed between the data scientist and the dataspace ended once the data scientist stopped providing feedback to the dataspace. Figure 3.1 illustrates the broken dataspace feedback loop, showing the disconnection of the partnership between the data scientist and the dataspace after the decision made by the data scientist to improve the query results manually, out of the reach of the dataspace. As a result, the dataspace’s potential to continuously enhance its integration quality is now compromised as it is now being replaced with manual correction.

The problems that result from this scenario are twofold: first, the dataspace may no longer obtain feedback from the data scientist if she continues to fix the integration query results outside the dataspace. Consequently, it is impossible for the dataspace to improve its integration quality. Second, the manual effort carried out by the data scientist in manually fixing the query result to fit her requirements is inefficient, as it often requires a considerable amount of time and valuable knowledge. Moreover, the data scientist needs to repeat all of the manual work whenever she requests an answer for the same query at some other time.

3.1.2 Reconnecting The Feedback Loop

One potential solution to these problems is to reconnect the broken feedback loop by exploring the potential of manual corrections on query results to act as a source of implicit feedback for the dataspace. This is illustrated in Figure 3.2, which incorporates a new component in our framework, called “implicit feedback from manual correction”. The new component is responsible for transforming the manual corrections performed by the data scientist outside the dataspace back into the dataspace as feedback. To realise this, we first need to find the answer to the question we formulated at the beginning of this chapter, where we ask whether or not feedback values can be inferred from manual corrections, and what types of feedback can be so inferred.

To answer this, we must consider the range of feedback types that can be consumed by dataspaces, to solve a range of integration problems. More of this will be discussed
in the next subsection.

3.2 Integration Problems

To provide a road-map for our research, we divided the set of integration problems encountered in dataspaces along two axes, giving four quadrants delineating the search space we need to explore (Figure 3.3). In this figure, the $y$-axis expresses the level at which the integration problem occurs: either at the data level or the schema level. The $x$-axis expresses the extent of integration problem: either systematic or unsystematic. Only three of types of integration problem were considered in our research work, since schema level changes are of necessity systematic (a change to the schema affects all
3.2. INTEGRATION PROBLEMS

Before examining the role of feedback in the three types of integration problem we mentioned earlier, we will first describe them in more detail, and give examples of how feedback from manual corrections could help to address them.

We begin with the levels of integration problem: data-level and schema-level. Data-level, also known as instance-level, refers to the integration problems in the instances or records. Corrections made at the data-level are evidence that conflicting or incorrect data values have resulted from the integration of multiple sources, giving rise to poor data quality. Examples of integration problems at data-level will be given in subsections 3.2.2 and 3.2.3, where we will discuss data-level problems of both extents.

Figure 3.2: General framework on manual correction in dataspaces

values in the affected columns).

Before examining the role of feedback in the three types of integration problem we mentioned earlier, we will first describe them in more detail, and give examples of how feedback from manual corrections could help to address them.

We begin with the levels of integration problem: data-level and schema-level. Data-level, also known as instance-level, refers to the integration problems in the instances or records. Corrections made at the data-level are evidence that conflicting or incorrect data values have resulted from the integration of multiple sources, giving rise to poor data quality. Examples of integration problems at data-level will be given in subsections 3.2.2 and 3.2.3, where we will discuss data-level problems of both extents.
In contrast, schema-level refers to the structuring of the integrated data, as described by its schema. It includes (1) properties of schema elements, such as names, descriptions, data types as well as constraints (e.g., integrity and referential constraints), (2) relationships between schema elements, such as part-of and is-a and (3) schema structures, such as relational and XML models. Examples of the integration problem at the schema-level are described in section 3.2.1.

### 3.2.1 Systematic Schema-Level

For an example systematic schema-level integration problem, we focus on the relationships between different source schemas. Such integration problems include naming and structural conflicts between schemas, where for example schema elements that describe similar concepts are given different names, such as Headteacher and Principal or Student and Pupil. This type of conflict is known as a synonym. Other naming conflicts, homonyms, happen when the same name is used to describe two or more distinct concepts. For example, the schema element called Grade describes exam grades in one schema and salary grades in another. Sometimes different structures are used to represent the same concept, such as Name being an attribute in one schema but split
3.2. INTEGRATION PROBLEMS

into FirstName and LastName attributes in another. Other structural conflicts involve inconsistent or different data types for the same schema elements across two or more schema, such as phone number, represented as a number in one schema and as a characters in another.

These problems usually require human insight and/or domain knowledge to accurately identify whether synonyms or homonyms represent the same real-world objects or different ones.

To demonstrate how manual correction can be used as a source of feedback in handling systematic schema-level integration problems, let us assume that a data scientist has deleted all tuples of query result that have values in the Grade column other than “pass” and “fail”. One possible reason for this deletion is because all the deleted tuples are related to the salary grade of staff instead of the desired student grade. This correction by the data scientist may indicate the presence of a naming conflict in the schemas, where two sources with an attribute Grade do not describe the same real-world concept. From the deletions performed on the query result, we can infer true positive/false positive style feedback where all deleted tuples are interpreted as false positive annotations and tuples that were not deleted as true positive annotations. This feedback can then be used to improve the schema mapping of the integration, possibly by identifying that the Grade column in one source does not match semantically with the Grade column in another, hence allowing the dataspace to improve the mapping of elements between two schemas and avoid the same mistake in the future.

Existing work has explored the potential of user feedback in refining integration problems at schema-level. For instance, Roby and Abulnaga translate user actions into refinement operations on the mediated schema and mappings [ERA17]. In their work, the user can provide feedback on the answers to queries they issue by distinguishing whether or not a record should exist in the query answer. The user’s responses were then used to fix both mediated schemas and mappings through a probabilistic technique. Similar to this is work by Belhajjame et al. [BPF+11, BPE+13] in which the user is asked to give feedback on the query answers provided by a dataspace. The feedback is then used to derive better-quality mappings for the global schema.

Both these existing proposals require the user to supply feedback explicitly. As we discussed earlier, requests for explicit user feedback can be intrusive, since the request for feedback may interrupt the user while carrying out their work, and costly, since the time spent providing feedback might be better used on other tasks. We hypothesise that manual corrections could be another source of implicit feedback for improving
integration problem at schema-level. In Chapter 4, we describe a study of this idea.

3.2.2 Systematic Data-Level

To further understand the systematic data-level integration problem, let us return to the example presented in Section 3.1.1, where a data scientist is preparing a report on school performance. Suppose that the data scientist has spotted the inconsistent formats in the names of the headteachers. She might correct this by manually editing those names that aren’t in the desired format. Other examples of this class of integration problems in this context are standardisation of the phone number and postcode values. For instance, the data scientist may also require all data concerning phone numbers be presented with a country code at the beginning of the phone number, such as +44 for the United Kingdom; while, for postcodes, the data scientist may require all alphabet letters in postcode values to be represented with capital letters, for example, correcting “m14 6yj” to “M14 6YJ”.

In respect of the format of headteachers’ names, all the values stated earlier are correct names for the individual, but the differences in representation can make analysis of the data much harder. When we want to search for an individual, for example, which of the available formats do we need to use to express the search term? Similarly, joining and filtering on the values in the column are both made more difficult when many data formats are present. A key task for the data scientist, therefore, is to make the integrated data more usable by converting the values into a common, suitable format. We refer to this process as format transformation.

In reality, manually correcting data format errors seems feasible only if a small number of such corrections are required. Often, the data scientist is required to repeat much of the manual effort she has expended previously in correcting the data format whenever she encounters a similar formatting problem in the future. With the broken feedback loop, the dataspace cannot help by formatting data correctly in the future.

The question now arises as to whether it is possible to re-use the information from manual correction to improve the dataspace’s handling of formats in future integrated queries. Once the need for systematic format correction is discovered, it could be performed automatically using off-the-shelf data transformation tools such as Flash Fill [SG16], Trifacta [KPHH11a] and Talend [Bar13]. These tools need to be trained on example pairs of incorrectly and correctly formatted values. This raises the possibility of using manual corrections to data format as a means to reduce the effort of providing such examples. For example, if the data scientist modifies the name of a headteacher
from Anne Lynn Anderson to A. Anderson, the before edit value and the after edit value together provide exactly the kind of example pairs needed to learn the format transformation rules [JACJ17b, WK16, SG16]. For instance, during manual correction, the modification of Anne Lynn Anderson to A. Anderson could become an example of a corrected pair of values to be used in learning. Thus, instead of explicitly asking the user to provide examples for format transformation purposes, the examples are now extracted from the manual corrections.

In our work, we explore the possibility of extracting example pairs for format transformation by using the before and after values of data from manual corrections, as evidence that our implicit feedback approach can handle this form of integration problem. Specifically, we propose several generation and filtering techniques for extracting format transformation examples from manual corrections, as described in Chapter 5.

### 3.2.3 Unsystematic Data-Level

Unsystematic data-level integration problems affect individual values, without any clear pattern across whole columns. Handling such problems usually relies on the domain knowledge and preference of the user of data. Manual corrections to data by the data scientist can be turned to an opportunity to improve the dataspace’s handling of particular values, by using them as a source of feedback on the handling of particular data values.

For example, suppose in 2019 the data scientist is asked to analyse primary school performance, using data in the same format given earlier. In the data set provided by the dataspace, the data scientist finds the following school records:

\{M.Graham, Abbott Primary, +44 161 681 3321, Maine Rd, M40 6PP\}

and

\{Michelle Johnson, Abbott Hey Primary, 1616813321, Maine Street,\}

The data scientist needs to identify whether these two records are semantically equivalent or not. For instance, the two records may refer to the same school, with one of the records having a complete school name while the other has an incomplete version. One of the records could have an more up-to-date last name for the headteacher if the school has changed its headteacher or if the earlier headteacher happens to have
changed her name recently. If the records are duplicates, one of them needs to be deleted, and the other record may need to be updated with more accurate information about the school name or the headteacher’s name. Alternatively, it may be that the two records are not duplicates but represent two different schools near each other with different headteachers and similar names. In this case, both records should remain, though some corrections may need to be made (for example, if the name of the second school is actually “Priory Hey Primary”).

In practice, to determine whether the two records mentioned earlier are duplicates, a data scientist first needs to examine if they match. One way to find out if they match is for the data scientist to manually inspect whether the source data shares identical primary key values. However, if primary key values are not available, then the next best approach is to establish a correlation between the two records, such as seeing if any common attributes share similar values that could link the two records together. For instance, based on the two school information records mentioned earlier, one of their common attributes is PhoneNumber. Although the representation of phone number is different for the two records, as one of the records has a Country Code and the other does not, the content of the phone numbers match, and this can be used as an indicator to confirm that the two records could represent information about the same school. If the postcode values are available for both of the records, then the postcode information could also be an excellent guide to determine whether or not both records represent the same real-world object.

Other than examining the phone number (or postcode) of the two records, further investigation of the school’s current headteachers’ name is also significant. The best possible way is for the data scientist to quickly search on the school website to determine the school’s headteacher. This is to establish knowledge and validate whether the two records represent a single real-world object or not, and if they do, then which of the two records is accurate? Given that the school name’s information is slightly different in the two records and one of the record is missing its postcode, a more thorough investigation is required. Suppose, after some digging, the data scientist now knows that the two records are in-fact represent the same school and that the second record has the most up-to-date information. Based on this knowledge, the data scientist deletes the first record, making sure any foreign key links now point to the second.

The manual deletion effort carried out by the data scientist in this example may seem small and simple if considered individually. However, similar reports may be required again at intervals, such as every week, every month, every quarter or even every
3.2. INTEGRATION PROBLEMS

year, meaning that the same integrated query must be issued repeatedly. The results will not always be the same, even though they come from the same sources and organisations, because of new or changed rows reflecting changes in the real world since the last query run, or because of changes in how the data is extracted and represented. Each new arrival of the query results is unlikely to be a carbon copy of the previous versions [SHGC18], and yet is subject to many of the same corrections already made.

In addition, each time when a new version of the data set arrives to prepare for the new report, it is highly likely the data scientist will need to repeat a lot of the correction they have done before. As a result, this repeated correction needs to be done and redone for each new report that they need to prepare. As can be observed, this is a failing of the integration done by a dataspase. These corrections are examples of unsystematic data-level changes. They are individual to the records to which they apply, and can’t be applied wholesale across a data set, or table or even within a column. Ideally, dataspase would know how the user wants to see the data, and would present it in that way in the first place. Our proposed manual correction approach seeks to address the shortcomings of dataspaces mentioned earlier. For unsystematic data level problems, the manual corrections made by the user to one version of the data set helps the dataspase to present future versions without the problems mentioned earlier.

Because manual correction is a form of unsystematic feedback that happens at the data level, this raises a question about how changes made by manual correction at the data level could be used to effect systematic and unsystematic improvements at the data and schema levels. It is essential to highlight that existing work has demonstrated that feedback at the data level (unsystematic data level feedback) given explicitly by the user could be used to improve the schema level integration, and our work is built up from this body of research work [BPE+10, MSD08, TJM+08].

One piece of existing work that inspired the research work described in this thesis is the work by Belhajjame et al. [BPE+10], where the authors demonstrated that explicit feedback through unsystematic annotation could be used to improve the schema mapping choices of the dataspase (a schema level aspect of the integration). In so doing, users are encouraged to annotate at the data level, which tuples are true positives and which are false positives as a form of explicit feedback. He further explains through an example that considers \( r \) to be a relation in the integration schema, populated using data from the source through a mapping, \( m \), where \( m \) is used to retrieve tuple \( t \) of \( r \). Suppose a particular user identifies that \( t \) is part of \( r \) by annotating true positives as his feedback. Based on specific feedback instances proposed by the author, elements of
the integration schema populated through candidate mapping can now be annotated, selected and refined. Established from Belhajjame’s work, we seek to gather the same kind of feedback implicitly rather than explicitly. Therefore, it is worth emphasising that our primary concern is not with the schema level improvement itself and how the unsystematic data level feedback leads to it. Instead, we are interested in understanding whether manual correction can be used as implicit feedback for this line of work.

3.3 Conclusion

This chapter presented a general approach to *Manual Correction* as a source of implicit feedback for dataspace integration. We also presented part of our roadmap for the study of whether manual corrections can act as a source of feedback for dataspace improvement. Based on our investigation into the potential for gathering implicit feedback from manual corrections, it can be seen that all three of our solutions share a common basis, and this suggests a common framework for working with manual correction. We illustrated the framework for manual correction in the dataspace setting in Section 3.1, in which the framework shows how manual correction could be used as implicit feedback alongside a typical dataspace architecture. We also identify the common basis of integration problems that we describe in detail in Section 3.2, in which we have identified three types of integration problem that dataspaces face, based on the *level* of integration problem as well as the *extent* of the problem. We used these quadrants to design a research roadmap, giving us target areas in which to find a use for feedback inferred from manual corrections. If usefulness can be shown in all three quadrants, then we will have shown the broad applicability of the technique.

We can conclude that the manual correction approach is promising as a source of implicit feedback in dataspaces. Furthermore, the framework introduced in this chapter can be integrated with a dataspace architecture as an additional component that could extract manual corrections and infer feedback relevant to a variety of integration tasks. The scenario examples presented in this chapter addressed significant integration problems such as uncertainty in schema mapping and iterative work on data cleaning. The manual correction approach also seems to have the potential to deal with other issues relating to data preparation, such as dealing with iterative effort in managing constantly changing data sets.
Chapter 4

Inferring False Positive/Negative Feedback from Manual Corrections

“No great discovery was ever made without a bold guess.”

Isaac Newton

This chapter describes our work on inferring true positive and false positive feedback from manual corrections. As mentioned in Section 1.5, the objective for this research is to develop and evaluate a range of methods for inferring useful feedback from manual corrections. If we can achieve this, the manual effort put into improving the query results should not go to waste. We will refer to the “users of data” as end-users or data scientists interchangeably across this thesis. By manual corrections we mean tasks such as insertions, updates and deletions performed by data scientists on data sets produced by the dataspace in order to make them fit for purpose. Our approach captures this manual work and converts it into useful values used as feedback for dataspaces. The feedback values derived from manual corrections can then be used to assist in refining the integration at no extra cost to the data scientist. Throughout this thesis, we will, in general, call our proposed approach manual correction.

In this chapter, we target a specific type of feedback, aimed at improving the schema-level integration of the dataspace. We chose the proposal by Belhajjame et al. [BPE+10] for using user feedback to refine and improve mapping selection by the dataspace. In this work, the authors require the user to annotate selected tuples in query results as being true positive, false positive or false negative based on the knowledge
the users have of the data. Through proposed feedback instances these annotated values were used to select and refine the automatically created schema mappings used by the dataspace in query answering. The authors claim that good quality mappings are constructed, and if more feedback instances are given, the mapping gets better.

In contrast to the work carried out by Belhajjame et al., where feedback is gathered explicitly through annotated values, we set out to discover whether the same kinds of feedback could be inferred implicitly from the manual corrections made by the data scientist to query results, to make them suitable for use in the current data analysis task.

This chapter is organised as follows: Section 4.1 briefly describes our proposed feedback inference method, which we have named co-integration. In Section 4.2, we specify the different forms of manual correction that our feedback inference method can operate on: namely insertions, updates and deletions of tuples. We then show how to determine the inferred feedback from these forms of manual correction. Section 4.3 presents our experimental setup, and Section 4.4 describes the results of our experimental evaluation of the co-integration method over an actual data set and a different scenario. The results provide some evidence that our approach is able to produce valid feedback values implicitly from manual corrections to query results. Finally, the chapter concludes with Section 4.6.

4.1 Co-Integration

Based on the scenario presented in Section 1.3 and expanded in Chapter 3, a solution is needed to close the broken loop of feedback collection in dataspaces. Conceptually, in co-integration, the end-user, a person who knows the data sets, works together with the dataspace and carries out integration work as an equal partner. Thus, each can request feedback from and provide feedback to the other. In particular, when the end-user carries out manual correction tasks, the dataspace observes the changes made and adjusts its understanding of how the data should be integrated as a result. In this way, the integration work on some part of the integration moves gradually from the end-user to the dataspace until the integration quality is sufficiently high and the end user’s focus moves to some other areas.

We set out to discover whether we could use manual corrections to integrated
query results to infer the true/false positive feedback of the kind used by Belhaj-jame et al. [BPE+13] to refine the schema mappings generated by the DSToolkit dataspace [HBM+12]. DSToolkit is the first dataspace management system to follow the fundamentals of model management [Ber03].

In the next section, we will discuss our strategies for inferring feedback values from the manual correction forms that we mentioned earlier. We will also discuss our proposed co-integration process and present our strategy for identifying and extracting manual corrections based on the manual work performed by the end-user on the query results they have in hand.

### 4.2 Inferring TP/FP Feedback From Manual Corrections

In DSToolkit, automated schema matching and mapping tools are used to create a set of mappings linking the global schema elements to those of the registered source schemas [HBM+12]. DSToolkit does not typically have enough information to select a single correct mapping for each global schema element, so it retains a ranked list of candidate mappings for each global schema table instead. When queries are processed, several mappings are selected from this list, and the query is evaluated using the union of the results they provide. Since most of the candidate mappings are incorrect somehow, the result of the query will contain errors. Such errors comprise missing values and incorrect values and tuples created by joining incorrect tuples from unexpected parts of the source schemas. To address this, DSToolkit asks the end-user to give feedback on the query results. Since the end-user has domain expertise but does not typically know about writing and correcting schema mappings, DSToolkit asks users to comment on the tuples included in the result rather than on the mappings directly. For example, the end-user can annotate individual tuples as false positives (i.e., results returned in the query answer that should not have been). Or they can add additional rows that they annotate as false negatives (i.e., results that should have been returned as part of the query answer but were not). They can also annotate tuples that they believe to be correct results as true positives. DSToolkit then takes this feedback (which, of course, is only available for some of the tuples in the result) and uses it to re-rank the candidate mappings. Further, it refines them by adding or removing conditions to increase the consistency of the query result and the feedback provided by the end-user. The aim is to find mappings that remove the false positive tuples, include the
false-negative tuples and retain the true positive tuples.

In our modification of this approach, rather than asking the end-user to annotate query results, they instead take a copy of the query results and manually correct it using some editing tool. We assume the use of a simple tool like Microsoft Excel for editing, due to its familiarity and ease of use for non-technical domain experts, and also because it allows rows to be added easily. Other tools we considered, including OpenRefine and Trifacta Wrangler, do not allow rows to be added to the data set being edited straightforwardly. If a tuple contains data the end-user knows to be incorrect, they modify the values to match their view of reality. If a tuple is missing, they add a row containing the values that should be present. We then pass the changes made to our algorithm, which infers false positive/false negative annotations for the original query result from the changes. We will now describe how this inference is made.

**Identifying What Has and Has Not Changed** The first step is to determine which tuples the end-user has made changes to when correcting the data. If a tuple is left unchanged by the correction process, then the end-user is happy for it to remain in the result. This can be either because it is a true positive or because the end-user does not have any information that would cause them to doubt the correctness. Unfortunately, we typically have no way of knowing which is the case. In this work, we generated a true positive annotation for all tuples that emerge from the correction process unchanged, although it is easy to envisage situations in which this behaviour is inappropriate. In future work, we could explore options such as asking the end-user for a confidence assessment of their corrections or a hybrid approach combining explicit and implicit feedback mechanisms.

To discover other kinds of feedback, we need to extract tuples that have been added and ones that have been deleted due to the manual correction process. This tuple extraction includes converting updates to individual tuples into an inserted tuple and a deleted tuple. Initially, we planned to get this information from the change log available in Excel, which gives a time-ordered list of all the changes made to a spreadsheet by a user. This log required significant additional processing, however, to extract the net changes at a tuple level. (We are not interested in corrections that the user made and then changed their mind about at this stage.) We, therefore, found it more practical to adopt a snapshot comparison approach, in which the result provided by the dataspace was compared with the corrected version produced by the end-user. However, if the data sets to be compared are very large and the number of corrections few, then the
Inferring TP/FP feedback from manual corrections would become more attractive.

**Inferring Feedback from Inserted Tuples** If the data scientist adds tuples to the query result provided by the dataspace, then it is because (in their view) the data is missing. That is, the data scientist is correcting a false negative result. However, our experience asking users to fix query results showed that our inference algorithm could not simply package all added tuples as false negative feedback. We must consider the possibility that a duplicate tuple may be added without the data scientist realising that they have done so (especially if the data set is large). Such duplicates can take several forms. There may be exact copies of some tuples, or several tuples may exist with the same secondary key values but differing non-key attributes. Tuples with distinct keys may also, nonetheless, represent the same real-world entity.

A new tuple can therefore only be considered indicative of a false negative in the query result if it was not already included in it. This version of the algorithm checked that no other tuple with the same key value exists in the original query result before classifying the addition as a false negative. We assumed that the global schema declares a secondary (human-readable) key for tuples, in addition to the usual surrogate primary key, and used this secondary key in detecting duplicates. Detecting tuples with different keys representing the same real-world entity is a challenging task requiring access to the dataspace’s entity resolution mechanism. We have not explored this possibility as of yet. Nor have we made use of the information about the duplication. Ideally, we would return additional feedback noting the duplicated tuples, but at present, DSToolkit does not accept this kind of feedback.

If the inserted tuple is concluded to be a genuine addition to the query result, we interpret this as an indication that it is a false negative and generate the corresponding explicit feedback. Algorithm 1 shows the generation of feedback, in the form described by Belhajjame et al. in their original paper [BPE+10]. Feedback is required in the form of a 4-tuple, \(< t, T, ext, source >\). Here, \( t \) is the tuple for which the feedback is generated (in the form of a list of attribute-value pairs) and either comes from the original query result or the manually corrected results. \( T \) is the table in the global schema from which the query results are taken. We follow the simplifying assumption in the DSToolkit system that query results are taken from a single global schema table rather than coming from a join of 2 or more tables. \( ext \) is a Boolean value indicating whether the user believes the tuple \( t \) should be present in the query result or not, and \( source \) gives information about how \( t \) got into the query result. If the user added the
Algorithm 1 Inferring feedback from tuple insertions [BPE+10]

**Input:** Q: The query result returned by dataspace
Q': The query result after manual correction
T: The global schema table from which Q comes

**Output:** EF: the set of inferred feedback annotations

1: \( I = \text{tuplesInserted}(Q, Q') \)
2: \( \text{for each } t' \text{ in } I \) do
3: \( kv = \text{keyValue}(t') \)
4: \( \text{if some } t \text{ in } Q \text{ has keyValue}(t) = kv \) then
5: \( EF = EF \cup \{ < t, T, \text{true}, \text{mappings}(t)> \} \)
6: \( \text{else} \)
7: \( EF = EF \cup \{ < t', T, \text{false}, \text{mappings}(t)> \} \)
8: \( \text{end if} \)
9: \( \text{end for} \)

tuple, then source takes the value “userSpecified”. If it comes from the original query result, then source should contain all the mappings that produced \( t \). For simplicity, we assume the availability of a method (mappings) that can return the mappings that produced a given tuple. In practice, providing this facility would only require that the provenance of query results in terms of mappings be retained for some time after the query result was created, at least until the manual corrections to the data set have been processed.

**Inferring Feedback from Deleted Tuples**  
Next, we consider the behaviour of the algorithm when the data scientist deletes tuples during the manual correction. In general, we can say that the removal of a tuple indicates that the domain expert believes that it should not have been included in the query result – that is, it is a false positive.

Table 4.1: Feedback inference based on approach

<table>
<thead>
<tr>
<th>Feedback Inference</th>
<th>Manual Correction</th>
<th>Annotated</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive (TP)</td>
<td>– No changes</td>
<td>– Correct</td>
</tr>
<tr>
<td></td>
<td>– Match with ground truth</td>
<td></td>
</tr>
<tr>
<td>False Positive (FP)</td>
<td>– Deletion</td>
<td>– Outdated Info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Incorrect Column Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Missing Column Value</td>
</tr>
<tr>
<td>False Negative (FN)</td>
<td>– Insertion</td>
<td>– Provide Remarks</td>
</tr>
</tbody>
</table>

However, we once again have to consider the possibility of duplicated tuples in the result before we can correctly convert a deletion into false negative feedback. Suppose
the full set of deletions made during manual correction leave a duplicate of the tuple still present in the query result. That could be read as evidence in favour of including the real-world object represented by these duplicates in the data set rather than the opposite. For example, Table 4.3 shows the results of a sequence of manual corrections made to the query result set given in Table 4.2. The rows are labelled to allow us to discuss the differences between the tables.

The data scientist has deleted tuples \( T_2 \) and \( T_6 \), causing the information to be removed from the result. She has also deleted \( T_8 \), however, we cannot characterise it as false negative feedback, because tuple \( T_1 \) contains the same information and is still present. On the contrary, this deletion gives evidence that tuple \( T_1 \) is a true positive for the query result.

Algorithm 2 shows the process of converting tuple deletions to feedback, using the secondary key to determine whether the information is still present in the query result after manual correction or not.

Table 4.2: Example of result returned by dataspace

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
<th>Position</th>
<th>Designation</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Frank</td>
<td>Campbell</td>
<td>Senior Lecturer</td>
<td>Deputy Dean (Academic Student Development Affair)</td>
<td>Networking</td>
</tr>
<tr>
<td>T2</td>
<td>Julia</td>
<td>Wright</td>
<td>Professor</td>
<td>Dean</td>
<td>Graphic Multimedia</td>
</tr>
<tr>
<td>T3</td>
<td>Megan</td>
<td>Baker</td>
<td>Assoc. Professor</td>
<td>Deputy Dean (Master)</td>
<td>Networking</td>
</tr>
<tr>
<td>T4</td>
<td>Rachel</td>
<td>Clarkson</td>
<td>Senior Lecturer</td>
<td>Head of Program (Computer Science)</td>
<td>Computer Science</td>
</tr>
<tr>
<td>T5</td>
<td>Robert</td>
<td>Cornish</td>
<td>Lecturer</td>
<td>Head of Program (Software Engineering)</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>T6</td>
<td>Joan</td>
<td>Allen</td>
<td>Lecturer</td>
<td>Head of Program (Networking)</td>
<td>Networking</td>
</tr>
<tr>
<td>T7</td>
<td>Joanne</td>
<td>Hodges</td>
<td>Assistant Registrar</td>
<td>Assistant Registrar</td>
<td>Administration</td>
</tr>
<tr>
<td>T8</td>
<td>Frank</td>
<td>Campbell</td>
<td>Senior Lecturer</td>
<td>Deputy Dean (Academic Student Development Affair)</td>
<td>Networking</td>
</tr>
</tbody>
</table>
Table 4.3: Example of result after manual correction

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
<th>Position</th>
<th>Designation</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Frank</td>
<td>Campbell</td>
<td>Senior Lecturer</td>
<td>Deputy Dean (Academic Student Development Affair)</td>
<td>Networking</td>
</tr>
<tr>
<td>T3</td>
<td>Megan</td>
<td>Baker</td>
<td>Assoc. Professor</td>
<td>Deputy Dean (Master)</td>
<td>Networking</td>
</tr>
<tr>
<td>T4</td>
<td>Rachel</td>
<td>Clarkson</td>
<td>Senior Lecturer</td>
<td>Head of Program (Computer Science)</td>
<td>Computer Science</td>
</tr>
<tr>
<td>T5</td>
<td>Robert</td>
<td>Cornish</td>
<td>Lecturer</td>
<td>Head of Program (Software Engineering)</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>T7</td>
<td>Joanne</td>
<td>Hodges</td>
<td>Assistant Registrar</td>
<td>Assistant Registrar</td>
<td>Administration</td>
</tr>
</tbody>
</table>

Algorithm 2 Inferring Feedback from Tuple Deletions

**Input:**
- Q: The query result returned by dataspace
- Q': The query result after manual correction
- T: The global schema table from which Q comes

**Output:**
- EF: the set of inferred feedback annotations

1. $D =$ tuplesDeleted($Q, Q'$)
2. for each $t$ in $D$ do
3. $kv =$ keyValue($t$)
4. if some $t'$ in $Q'$ has keyValue($t'$) = $kv$ then
5. $EF = EF \cup \{ < t', T, true, mappings(t') > \}$
6. else
7. $EF = EF \cup \{ < t, T, false, mappings(t) > \}$
8. end if
9. end for

4.3 Experimental Setup

To gain insight into the practical value of this approach to feedback generation, we designed and carried out an experiment to compare the feedback inferred from manual corrections to that obtained when end users give explicit feedback annotations on the data. The objective was to determine whether the feedback inferred from the manual correction is similar to that obtained when feedback on true/false positives/negatives is requested explicitly, or whether we get less useful information from feedback requested explicitly.

The hypothesis we set out to test is as follows (with its corresponding null hypothesis):
4.3. EXPERIMENTAL SETUP

- **Hypothesis 1 (HO)** There is no difference in extracting user feedback from manual corrections and the explicit annotation approach.

- **Hypothesis 2 (H1)** There is a difference in extracting user feedback from manual corrections and the explicit annotation approach.

To test these hypotheses, we needed to set up an experiment which gathered feedback on data values by the two approaches, so that they could be compared. We needed a data set covering a domain for which we could easily find study participants who would have sufficient expertise to be able to correct errors in the data quickly, and for which we could recruit sufficient participants with a similar level of expertise.

To meet this need, we used a data set that contains information on the management team of an organisation, using the schema in Table 4.2. The data initially contains information about all the staff in the organisation. For this experiment, we only selected tuples with information of persons in the organisation that holds a top management position. Member of staff with position dean, associate dean and all heads of departments are shortlisted, and their information is selected to be in the data set used for the experiment. One of the reasons to select members of staff that hold a top management positions is because we require participants with a specific set of domain knowledge, and the top management team are usually known by many, and recruited participants are familiar with this information. In total, there are 10 tuples that have information about staff who belong to the top management team, and this data acted as our original data set.

We injected ten errors into the data set, covering various error types, in particular outdated data, missing data, incorrect values and incomplete values. The original (error-free) version of the data set was retained for use as the ground truth. To give an overview on how we injected errors on the original data set, firstly, for error type outdated data, we inserted information of staff that used to hold a top management position; however, their tenure has expired and they are no longer in the management team. In fixing this kind of error, participants are expected to delete the tuples of outdated information. Then, for injected errors on missing data, we deleted some rows from the original data set. In addressing this type of error, participants are expected to insert new tuples to the data set based on the missing values. Next, to inject error of incorrect values, we provide wrong information in some of the fields and participants are expected to update this value to correct values such as updating specific tuples that have incorrect values on it Department field, by updating the value from Networking
CHAPTER 4. INFERRING FALSE POSITIVE/NEGATIVE FEEDBACK

to Computer Science. Finally, to inject error of incomplete values, we deleted some values from some fields of some tuples from the original data set and participants are expected to update the empty field with correct values. Participants start with a data set of 15 rows that contain different types of error described earlier. For every manual correction performed in fixing the injected error, we define a feedback inference for it as presented in Table 4.1.

We called for volunteers using email advertisements requesting members of staff who regularly work with data and have moderate knowledge and skills in using Microsoft Excel. We managed to recruit 12 participants, all professional and working in a public university. Existing studies that are similar to ours have a comparable number of participants [KPHH11a].

Then, we split them into two groups: one group gives feedback through manual corrections (implicit feedback), and the other group gives feedback through annotation (explicit feedback). Each group contained three recently appointed members of staff who worked less than five years and three more extended serving members who worked more than five years. These participants responded to our call for volunteers, and hence we grouped them as such. As the recruited participants have varying years of experience working in their institutions, we need to equalise distribution of them in each group fairly.

The experiment did not require formal ethical approval since no personal information, vulnerable group, or risk disclosure were involved. However, sensitive and confidential information, specifically staff names and positions, have been changed before publishing this thesis’s work to protect privacy.

Before anything else, we provided a brief one-to-one explanation of what was expected from participants. We also provided written guidelines on the conduct of the experiment to all the participants. Both groups were asked to examine the data to look for errors, using their knowledge of the organisation’s current staff. To allow control for differences in motivation and perfectionism in the task, each volunteer was asked to complete the task during a 10-minute individual Skype conversation.

Group A was asked to make manual corrections to the data, using Excel spreadsheets configured to record their actions. Microsoft Excel is chosen as the tool as it is one of the data manipulation tools often used by the members of the group in their daily work tasks. Group B was asked to annotate each tuple with a description of the errors found, using a drop-down menu containing a list of false-positive/false-negative
4.4 Evaluation Results

As stated, by this experiment, we aimed to test the hypothesis that implicit feedback inferred from manual corrections is as useful as explicit feedback annotations in identifying errors in data sets. Based on the outcomes from the two groups of volunteers, we compiled two sets of feedback, one implicitly and one explicitly obtained, and compared each with the ground truth. For each volunteer, we calculated the number of true and false positives correctly and incorrectly identified. (For a volunteer \( v \), \( TT(v) \) denotes the number of true positives (TPs) correctly identified by \( v \); \( TF(v) \) denotes the number of TPs incorrectly identified by \( v \); \( FT(v) \) and \( FF(v) \) denote the number of false positives correctly and incorrectly identified by \( v \)). We used these counts to calculate the precision of each volunteer as follows:

\[
\text{Precision}(v) = \frac{TT(v) + FF(v)}{TT(v) + TF(v) + FF(v) + FT(v)}
\]

The results are shown in Figure 4.2. We also aggregated the results, to give a precision score for each group. The results of this are shown in Figure 4.1.
CHAPTER 4. INFERRING FALSE POSITIVE/NEGATIVE FEEDBACK

Figure 4.2: Precision for each volunteer

4.5 Result Discussion

Our results show that the feedback produced by the group making manual corrections correctly identified all true positives when matched against the ground truth. However, three of the group members made mistakes and failed to identify some of the false positives. Only two members of this group managed to identify all false negatives correctly. In total, two of the manual correction group members managed to produce corrected data sets equal to the ground truth.

The group making explicit feedback annotations made more errors than the manual correction group. Only one member from the explicit feedback group (Group B), managed to produce a data set equal to the ground truth. All members from this group managed to identify all true positives correctly, but three members from the group incorrectly identified a true positives and this results in them having more true positives than the ground truth. Four members from manual correction group (Group A), managed to identify all false-negatives correctly. We observe that four out of the six members gave different characterisations for the annotations. The most common errors as characterising previous employment as “correct” instead of using the “outdated information” annotation. Based on our interviews with the participants, we found out that subjects overlooked the option of “outdated information” in the annotation list. The situation is understandable as they need to scroll down to the bottom of the list to reach the “outdated information” option. This failure to observe and use all the available annotations could be a possible explanation why the explicit feedback approach yielded poorer results than the manual correction approach. There is also the possibility that participants interpreted the annotations provided differently, and that there is
less subjectivity involved in manually editing the data than in explicit feedback through
data labelling.

We initially expected the manual correction approach to perform as well as the
explicit feedback approach, but we found that it outperforms the explicit approach.
However, the results gathered must be interpreted with caution since the number of
data points in our experiment was small, and we can therefore derive only tentative
conclusions from the study. For this reason for supporting the result we have, we used
a two-tailed t-test of equal variance to determine if there is any significance difference
between the two groups we tested. We wanted to test the possibility in both directions
to prevent bias.

In total we have a sample size of ten that contains information about top manage-
ment staff such as first name, last name, position designation and department. And the
sample consists of ten errors with corrections coming from each of the six participants
in each of the two groups, manual correction and explicit feedback. The three types of
eroerror injected to the data set that are covered in the \( n \) values are true positive, false pos-
itive and false negative. However, with a small sample size, caution must be applied
as the finding might be less conclusive. For cross validation, we use the value of our
precision as presented in Figure 4.2.

To proceed with the t-test, we first need to determine whether our data is of equal
or unequal variance. For this, we use the variance rule of thumb by performing F-test.
The F-test result gives a p-value of 0.122, and because the p-value is larger than the
alpha value of .05, this implies that we have two groups of data of equal variance. In
addition, we also calculate the ratio based on the variance between the two groups.
Our calculation returns a ratio of less than 4; hence, we are confident that our two
data groups are of equal variance. We then proceed with a two-sample t-test of equal
variance. Based on the t-test, we acquire \( t(\text{stat}) = 1.246 < 1.86 \ t(\text{critical}) \) (or p-value =
0.248 > .05 - \( \alpha \ ) therefore we keep the null hypothesis. As a result, we are confident
there is no significant difference in extracting user feedback from manual corrections
and the explicit annotation approach as stated in our null hypothesis.

It could be argued that the manual correction approach performed better due to the
mistakes made by the participants in identifying the correct values in the annotation
list. If all values of the annotations list had been visible to the participants (without
them having to scroll to find the best answer), it is possible that the participants from
the explicit feedback group may have performed on a par with or even better than
the manual correction group. On the other hand, the differences might be caused by
4.6 Conclusions

In this chapter, we proposed a method for generating true/false positive feedback from manual corrections to data. This gives an approach in which the end-user and the dataspace share the integration work. We proposed an algorithm that infers feedback from manual corrections to query results, for use by a dataspace in improving its schema mappings. We compared inferred feedback from manual corrections with explicitly given feedback by conducting an experiment. In so doing, we pursued our Objective 1 of identifying and evaluating a method for inferring useful feedback on the schema level integration from manual corrections made to query results produced by the dataspace.

In our small scale study, those volunteers who manually corrected data gave feedback as good as or better than that of those annotating errors explicitly. The reason for the difference is not immediately clear. It may be that the ten annotation options supplied to the volunteers for making explicit feedback were too numerous or too confusing. Certainly, the volunteers tended to select annotations from the top of the list, when the correct annotation was towards the end of the list. On the other hand, it may be that the manual annotation route avoids the need to create a set of concepts or phrases that must be used, such as for communicating information about errors between the person giving feedback and the person using it. In relation to our contributions for this chapter, the experimental results show that manual correction can be a good source of implicit feedback for data integration information in meeting user’s information’s needs, though more studies would be needed to confirm this in the general case.
Chapter 5

Inferring Format Transformation Rules from Manual Corrections

“Knowledge without transformation is not wisdom.”

— Paulo Coelho

This chapter presents the design and implementation of an approach to gather feedback from manual corrections from which to address systematic data level integration problems. We focus on the specific problem of regularising the formatting of data values in integrated query results. More specifically, we aim to determine whether we can use the changes made by the data scientist when manually correcting query results to generate input-output example pairs for format transformation rule learning. The rules learnt from these example pairs can be used by the dataspace to inform the formatting of similar kinds of data in the answers to future queries produced using the affected table, thus improving the quality of the integration over the long term.

Handled manually, format transformation efforts (such as converting names, addresses or phone numbers from one format to another) are usually repetitive. Authoring effective and safe transformation rules requires significant technical skills. In many cases, data scientists often resort to custom scripting in order to handle formatting problems. Thus, certain technical skill sets have often been necessary when dealing with format transformation work—skills that the data scientist may not have. Moreover, custom-made scripts for transformation can be difficult to manage, particularly when the transformation rules become invalid due to changes in requirements or amendments in business rules. In addition, whenever incorrect transformations occur, the data scientist may need to construct another set of scripts to offset or reverse
any transformation mishaps that have happened. This may require them to write and maintain many versions of scripts and data [RH01].

To counter this, there have recently been proposals for an alternative to scripting in which input-output pairs are used to synthesise format transformation programs using Programming-By-Examples (PBE) techniques. Off-the-shelf transformation and editing tools such as Flash Fill (embedded in Microsoft Excel [SG16]), Talend [Bow12, Tal09] and Trifacta and its predecessor Wrangler [KPHH11a, KPHH11b] now contain such helpful facilities. But even then, the manual provision of examples for such tools to be able to learn the required rules can be time consuming [WFW+20, KPHH12, HHK15], and must be done for each data type involved in queries.

We propose to infer input-output examples from manual corrections to query results, to make the discovery of example pairs for format transformation rule learning more cost-effective. The manual corrections will be made anyway, to make the data fit for use, so this approach provides example pairs for the dataspace to use without any additional work being required on the part of the data scientist. This contrasts with the explicit construction of suites of input-output examples, requiring the data scientist to put time in to construct representative training data systematically. In addition, example pairs derived from manual corrections could also be reused by the dataspace over time, resulting in more correct answers to future queries and consequently lessening the amount of manual correction work needed in the future.

When data is brought together from many sources, we may have columns containing data items of the same kind but formatted differently. For instance, we may have a column containing the names of people formatted in different ways, for example including names in the form initial, dot, surname, such as B.Obama, alongside names in other formats, such as Barack Obama, Barack H. Obama and Barack Hussein Obama.

Of course, all these values are correct names for the specific individual, but the differences could make the analysis of data much harder. For example, if we want to search for an individual, which available formats do we need to use to express the search term? Similarly, joining and filtering on the values in the column are both made more difficult when many data formats are present. Therefore, an essential task for data scientists is to make the integrated data more usable by converting the values into a common, standard format. We refer to this process as format transformation.

Tools such as Flash Fill have been proposed to help with this process [Gul11, GHS12b]. Users of Flash Fill provide examples of incorrectly and correctly formatted data values, and the tool learns a set of generalised format transformation rules that
5.1 LEARNING TRANSFORMATIONS FROM DATA CORRECTIONS

can be applied across the whole data set.

In this chapter, we explore the possibility of extracting format transformation examples from manual format corrections. The aim is not to reduce the work for data scientists on their first encounter with a data set or data type but to gather information from that work that can be used to reduce the manual effort needed for future format transformation tasks. The question we set out to answer in this chapter is: can format transformation examples suitable for learning useful format transformation rules be extracted from manual data corrections? Specifically, we wanted to see if we could use manual corrections as a source of training examples for Flash Fill [Gul11], instead of requiring the user to provide examples explicitly. We describe several techniques for obtaining examples from corrections, which are empirically evaluated in various scenarios.

The rest of the chapter is organised as follows. Our proposed architecture for acquiring format transformation example pairs from manual corrections is presented in Section 5.1. Then, Section 5.2 provides the detailed approach for generating and extracting examples from manual corrections while the experimental design for our evaluation and the results we obtained are given in Section 5.3. Next, based on the results from the previous section, we discuss the validation of the learned transformation rules in Section 5.4, and from the findings, we discuss in Section 5.5 how we have extended our experimental harness to consider the coverage of representative cases—a key success factor in learning effective format transformation rules. We also evaluate our proposed validation approach, and discuss the results. Finally, conclusions are given in Section 5.6.

5.1 Learning Transformations From Data Corrections

We propose to extract examples for learning format transformation rules from corrections made by data scientists on data sets. In this section, we describe our approach. The system we have built is called ManEd (short for “manual edits to data”).

We assume a scenario in which a data scientist is working on a single query result to prepare it for use in some later integration or analysis task. She first extracts the query answer from the dataspace to a CSV format. Next, she examines the query results using some preferred editor (such as a spreadsheet) and fixes errors where she sees them. As part of this, she changes the format of a sample of the data values to bring them into line with the formats used by other data sets involved in the analysis.
Due to time constraints she leaves other rows unchanged (even though they may not be in the desired format).

Figure 5.1: General Framework of Manual Correction Transformation (ManEd)

At this point, she saves the changes she has made and submits the two versions of the data set (the one before her changes and the one after them) to the ManEd tool. Figure 5.1 shows the architecture of the tool: a pipeline of three main components. The first component examines the two versions of the data set supplied and generates a set of before-and-after examples for each column in the data set. To illustrate this process, Table 5.1 shows the example pairs that would be generated for a column containing people’s names from the values the column contains before the user edits have been made and after they have been made. In this case, the data set is small, and the data scientist has been able to correct all format errors present in it, so we get an example pair from each row.

These example pairs are then passed to the second component, which uses them to generate general format transformation rules. In our case, we use an implementation of the Flash Fill algorithm, in Java by Wu and Knoblock [WK16]. This component learns a transformation rule from the generated examples for each column. In the case of the example pairs shown in the table, it learns a rule that abbreviates the first name to an initial followed by a full stop, removes any spaces between the first name and the last names, removes any middle initials, and leaves the last names unchanged.

<table>
<thead>
<tr>
<th>Row</th>
<th>Before-Edit Value</th>
<th>After-Edit Value</th>
<th>Example Pair Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>B.Kenyon</td>
<td>Ben Kenyon,B.Kenyon</td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>G.O’Conor</td>
<td>Gina O’Conor,G.O’Conor</td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>B.Hywel</td>
<td>Branwen Hywel,B.Hywel</td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>A.Perks</td>
<td>A K Perks,A.Perks</td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>E.Davidson</td>
<td>E Davidson,E.Davidson</td>
</tr>
<tr>
<td>7</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt,N.Buckhardt</td>
</tr>
</tbody>
</table>

The learnt rules are then passed to the final component: the Rule Validator. Since, in general, the Example Generator component will only have been able to guess at a
suitable set of before-after format pairs, the rules learnt by the Flash Fill component may be incorrect. Therefore, this final component has the task of checking that the rules seem sensible. Only format transformation rules that pass the final validation checks are output by ManEd.

The rules learnt in this way can be applied by the data scientist to the data set on which she is working to correct the format of rows she has not explicitly corrected herself. Alternatively, they can be passed back to the dataspace, stored in its metadata for use on later data sets, to allow the data scientist to be presented with data sets in the correct format from the outset, rather than requiring them to be corrected each time explicitly.

This gives an overview of the basic approach, but two questions remain: how can we generate good examples from corrections made by data scientists (when, for example, only a sample of rows will have been corrected), and how do we validate the learnt rules to ensure that only sensible rules are output from ManEd? The following two sections provide answers to these questions and describe the results of our experiments to assess how successful our proposed strategies are.

5.2 Example Pair Generation Techniques

Although from our simple example, it may seem trivial to generate example pairs from the corrections made by the data scientist, the situation is generally more complicated. For large data sets, the data scientist may only correct a sample of the rows. In addition, in editing the data set, the data scientist is not only concerned with format transformations; she may also need to correct errors in the data (such as correcting the spelling of “Davidson” to “Davison” in our example), or she may need to add in missing elements. Finally, the data scientist may introduce errors into the data set during manual editing or use inconsistent formats for different parts of the data. All these factors complicate the task of the Example Generator component of ManEd.

5.2.1 Example Pair Generation Strategies

We propose three example pair generation strategies: null filtering, edited value filtering and augmented edited value filtering, which we will now describe. Each one takes a list of the full before/after edit pairs from the data set versions supplied by the user and outputs a set of example pairs to be used in learning format transformation
rules. We assume that the data set has a set of key columns that can allow us to find corresponding rows in the before-edit and after-edit versions, even if the after-edit version has been reordered before saving. As mentioned previously, we do not concern ourselves with newly added rows (for which we have no before-edit value) or deleted rows (for which we have no after-edit value). To simplify the presentation and without loss of generality, we assume that the data scientist is interested in the contents of only a single non-key column. Multiple non-key columns can be handled by iterating over several calls to ManEd, and edits to key columns can be handled by associating a unique surrogate key with each row before manual corrections are made.

Null Filtering (Baseline) The first and most straightforward strategy is proposed only as a baseline case, against which to assess the performance of the other strategies. In it, we use all before/after pairs as examples, whether the data scientist edited them or not. This has the virtue of increasing the chances that our set of example pairs includes at least one of each representative case of the format transformation desired - a property known to be important if Flash Fill is to produce a reliable transformation rule. For example, when transforming name data, we need examples of the before and after transformations for names with one forename, with two and with three. We would also need surnames with prefixes (like von and O’) and double-barrelled names. Of course, this strategy also has a serious disadvantage in that it will likely include some (possibly many) incorrectly formatted values, making it difficult for Flash Fill to learn an accurate rule.

To illustrate this strategy in action, we return to our running example from Section 5.1. Table 5.2 shows (in the Example Pair column) the pairs that would be sent to the Flash Fill Rule Learner component by ManEd using this strategy. For this case, the user only corrected a sample of rows (rows 1, 2, 4 and 5) but both corrected and uncorrected rows are used to create example pairs to pass to the rule learner.

The final column (labelled “Transformed Result”) shows what result we get if we allow Flash Fill to learn a transformation rule from these examples and then apply the rule to the values in the version of the data set that has been manually corrected, the after-edit set. We can see that the learnt rule manages to transform only Row 4 correctly and (also correctly) leaves row 7 unchanged. Rows 3 and 6 (unsurprisingly) retain their original incorrectly formatted values, while rows 1, 2 and 5 were transformed to an empty string. Generally, one of the reasons Flash Fill returns empty strings is that it was not sure what to do since the examples provided are not consistent throughout
and do not cover every representative case that exists in the data sets.

Table 5.2: Transformation result from null filter (baseline) example pairs

<table>
<thead>
<tr>
<th>Row</th>
<th>Before-Edit Value</th>
<th>After-Edit Value</th>
<th>Example Pair</th>
<th>Transformed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>B.Kenyon</td>
<td>Ben Kenyon, B.Kenyon</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>G.O’Conor</td>
<td>Gina O’Conor, G.O’Conor</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>Branwen Hywel</td>
<td>Branwen Hywel, Branwen Hywel</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>A.Perks</td>
<td>A K Perks, A.Perks</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>E.Davidson</td>
<td>E Davidson, E.Davidson</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt, N.Buckhardt</td>
<td></td>
</tr>
</tbody>
</table>

**Edited Value Filtering** Ideally, we need a filtering strategy that only uses correctly formatted values in the example pairs. The problem is, of course, that ManEd does not know which of the values are correctly formatted. In this strategy, we lean heavily on the user’s domain expertise and assume that all values that the user has edited are now correctly formatted. We create example pairs from any row where there is a difference in the value of the column of interest in the after-edits version of the data and filter out any rows where the value remains the same. Table 5.3 illustrates how this strategy works on our running example.

As can be seen, this approach produces fewer example pairs than the null filtering strategy. This has the advantage that fewer incorrect example pairs are used for learning the rules but, where data sets are large, and only a sample of rows can be corrected, we will have the disadvantage that some representative cases may not be included. Nevertheless, we can see from this small example (where most of the rows have been edited) that this strategy learns a rule that can correct the formatting of the unedited rows (3 and 6) using the rule learned.

Based on the null and edited value filtering strategies, we realised that some columns have an empty cell as their output values. We speculate that such an outcome is due to the absence of input for that particular format. In the literature on Flash Fill, researchers have pointed out that for good results it is usually necessary for the user to go through several interactive rounds of example provision [Gul11, SG16]. Typically, between two to three rounds of input example provisions (with a maximum of four rounds) are needed. In the literature, the researchers provide one example for a different representative sample. So if a couple of rounds are needed, meaning that, for one representative sample, typically two to three examples per representative sample are required before a good transformation rule can be learned. However, because our
running examples are based on small data sets, most representative formats managed to be covered at least once.

Table 5.3: Transformation result from edited value filtering example pairs

<table>
<thead>
<tr>
<th>Rows</th>
<th>Before-Edit Value</th>
<th>After-Edit Value</th>
<th>Example Pair</th>
<th>Transformed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>B.Kenyon</td>
<td>Ben Kenyon,B.Kenyon</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>G.O’Conor</td>
<td>Gina O’Conor,G.O’Conor</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>Branwen Hywel</td>
<td></td>
<td>B.Hywel</td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>A.Perks</td>
<td>A K Perks,A.Perk</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>E.Davidson</td>
<td>E Davidson,E.Davidson</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt</td>
<td></td>
<td>N.Buckhardt</td>
</tr>
</tbody>
</table>

**Augmented Edited Value Filtering**  Early trials of the Edited Filtering strategy highlighted an unexpected feature of the examples we had generated. They performed well on incorrectly formatted results (of the kind we were providing as “before” examples in the example pairs) but performed very poorly when asked to transform correctly formatted values. These values were converted into an empty string. We hypothesised that this was because we were not providing Flash Fill with example pairs showing that correctly formatted values needed to retain their formatting.

To address this, we propose a third example generation strategy that extends the Edited Filtering strategy. It creates not one but two example pairs from every edited row in the data set: one pair contains the before value and the after value, while the second pair consists of the after value and the after value. The Rule Learner is given examples of how to correct the formatting of incorrect examples and of how to retain the formatting of examples that are already correct.

As before, we illustrate this with our running example. Table 5.4 shows the example pairs generated when the Augmented Edited Value strategy is used. With this new strategy, the learnt rule manages to transform all the rows in the data set correctly. Nevertheless, the next question that we may have to consider based on the transformation outcome of this approach is: **What if the uncorrected values are not manually corrected for a reason and on purpose?** We leave this challenge for future work, and for now, we assume that the uncorrected value is there because the data scientist did not look at it and it should also be transformed to the format that matches the left-hand side format of the example pair’s list.
5.3 Evaluation of the Generation Strategies

We now describe how we have evaluated the three example pair generating strategies, and give the results of our experiment comparing them.

5.3.1 Experimental Harness

To assess the performance of the proposed example pair generation strategies described in the earlier sections, we needed to create an experimental harness that would provide ManEd with corrected data sets to generate format transformation rules. Note that it is not our aim to determine the Flash Fill algorithm’s ability to learn rules accurately from small numbers of examples. This has already been demonstrated by the work by Wu et al. [WK16]. Instead, we aim to assess whether our strategies for inferring example pairs from manually corrected data sets can lead to accurate transformation rules being learnt by Flash Fill and how they perform in comparison to each other. Therefore, the experimental harness should also allow us to assess the accuracy of the generated rules to the “ground truth” format rules, which the corrections should reflect.

Our harness provides this by mimicking some of the actions taken by a data scientist using ManEd. It requires a seed data set that can be used as the basis for the performed experiments. Since we expect that the harness will be used with real data sets containing format inconsistencies, we need to give it some way of distinguishing

Table 5.4: Transformation results from augmented edited value example pairs

<table>
<thead>
<tr>
<th>Rows</th>
<th>Before-Edit</th>
<th>After-Edit</th>
<th>Example Pair</th>
<th>Transformed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>B.Kenyon</td>
<td>Ben Kenyon, B.Kenyon</td>
<td>B.Kenyon</td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>G.O’Conor</td>
<td>Gino O’Conor, G.O’Conor</td>
<td>G.O’Conor</td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>Branwen Hywel</td>
<td>Branwen Hywel</td>
<td>B.Hywel</td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>A.Perks</td>
<td>A K Perks, A.Perks</td>
<td>A.Perks</td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>E.Davidson</td>
<td>E Davidson, E.Davidson</td>
<td>E.Davidson</td>
</tr>
<tr>
<td>7</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt</td>
<td></td>
<td>N.Buckhardt</td>
</tr>
</tbody>
</table>
correctly formatted values in the column of interest from incorrectly formatted ones. We do this by requiring the harness user to provide a version of the seed data set that is identical except that all values in the column of interest have been corrected to match a single format. To avoid possible confirmation bias, we avoid using Flash Fill itself to make this transformation and instead perform the task manually. (This necessarily puts a practical limit on the size of seed data sets that can be used with the harness.) In what follows, we refer to the version that has been manually corrected for consistent format in the column of interest as the formatted seed data set. Both these data sets are loaded into a PostgresSQL database for convenient access by the experimental harness.

The next step is to use the seed data to create the data sets that will be given as input to ManEd: the original version of the data that the data scientist sees and the version with corrections to selected rows. To allow us to experiment with different sizes and properties of data set, the user of the harness can set:

- the size of the before data set (in rows, up to and including the size of the seed data set),
- the proportion of rows in this set that will be modified in the before data set, and
- the proportion of rows in this set that will be modified in the after data set, and
- the proportion of modifications that correctly match the ground truth formatting rule.

These parameters allow us to experiment with different manual correction behaviours on the part of the (simulated) data scientist. First, the harness creates the before-edit data set by taking a sample of rows at random from the seed data set. It then creates the after-edit data set by selecting at random the rows that will be modified correctly and the rows that will be modified incorrectly. Finally, the values for the column of interest in rows that are modified correctly are taken from the equivalent rows in the formatted seed data set. It is important to note that the before-edit data set also contains the correctly formatted rows. The proportion of correctly formatted rows in the after-edit data set grows at twice the size of the correctly formatted rows in the before-edit data set so that we can observe if the number of manual corrections has any significant effect on the result. The rationale of this is that a high percentage of manual corrections means an increased number of example pairs and presumably will show a better outcome from the transformation.
5.3. EVALUATION OF THE GENERATION STRATEGIES

Values for rows that are modified incorrectly are created by changing characters at random in the original. All other rows are copied across from the before-edit data set unchanged. (The after-edit data set contains the same number of rows, with the duplicate row keys, as the before-edit data set.)

Both the generated before-edit and the after-edit datasets are then passed on to ManEd, which attempts to learn a format transformation rule from the corrections injected by the experimental harness. This is done three times, once for each example pair generation strategy, and the learning rules are recorded.

The harness’ final task is to determine how closely the rule learnt by ManEd in each case matches the ground truth rule. Because Flash Fill rules are highly procedural in form, it would be challenging to compare rules by analysing their semantics statically. Instead, we opt for the conceptually more straightforward approach of comparing rules dynamically by assessing how correctly they transform a second data set. We already have the ground truth for this comparison in the form of the formatted seed data set.

To assess the accuracy of each rule, the harness extracts a second sample from the seed data set at random. The size of this second sample is the same as the before-edit data set size. It applies the learnt rule to the column of interest in this new data set and then compares the resulting transformed value in each row with the same column value in the corresponding row of the formatted seed data set. We count up the number of
correctly and incorrectly transformed values to give precision and recall scores for the
rule. These are the scores that are output by the harness as the result of an individual
experiment run. They can be collected from a series of runs, including repeated runs
using the same experimental parameters, and averaged to give an idea of the stability
of performance of the ManEd strategies over data sets with different properties.

5.3.2 Experimental Design

We now present the experiment that we carried out using the experimental harness
described in the preceding section. As stated, our aim was to assess the relative per-
f ormance of our three example pair generation strategies in inferring example pairs
for use by Flash Fill. We predicted that the augmented edited value filtering strategy
would perform best, with the null filtering strategy performing worst.

For the seed data set, we selected a open government data set called “Schools
 in England”\(^1\). We based our experiment around a scenario in which primary school
performance around the United Kingdom is analysed based on publicly available data,
 and in which there is a need to integrate headteacher and school information from
different sources.

To design the experiment, we need to set the parameters that will be used to con-
figure the experimental harness for each run. These are listed below.

**The seed data sets that the experiment will be run against.** Since each experiment
 runs against one column of data, this choice affects the number of data types that
 the experiment will be run against.

**The size of the before data set in rows.** This affects the number of format transfor-
mation tasks that our method needs to solve simultaneously in each experiment.
 It also affects the variety of format transformations that need to be fixed.

**The proportion of rows in the before data set that will be modified in the after data set.**
 This parameter also affects the number and variety of format transformations to
 be fixed in each run.

**The proportion of modifications that correctly match the ground truth formatting rule.**
 This determines the amount of information on correct format transformations
 available to each process, and consequently the degree of difficulty for the learn-
ing task.

\(^1\)https://data.gov.uk/dataset/schools-in-england
The threshold used by the validation component for accepting or rejecting rules.  
In this version of the ManEd system, the rule validation component needs to be 
given a threshold accuracy score to be used for deciding whether to accept or 
reject the learned rules.

The number of times the experiment will be run for each configuration. Since the 
harness randomly creates a new data set with new format transformation chal-
enges on each run, we will run the experiment for each configuration multiple 
times to be able to see the general performance, and so that the results aren’t 
skewed by an anomalous selection of data in a single run.

The major choices to be made related to the set of data types to use in the experi-
ments and the data set sizes over which to run the experiments. To choose values 
for these parameters, we examined the literature for studies assessing the accuracy 
of format transformation by example approaches, published in high quality venues. 
We selected six studies that are closely related to ours and examined the experimental 
configurations they used. These studies were: Foofah\(^2\) [JACJ17b], TDE\(^3\) [HCG\(^+\)18], 
UDATA\(^4\) [PKP19], SynthEdit\(^5\) [BFPK19], Iterative data transformation\(^6\) [WK15] and 
CLX [JCJ\(^+\)18].

Data Types Five out of the six studies we examined used data sets with a range 
of diverse data types. The common ones are transformation tasks related to people’s 
names, phone numbers, addresses, and dates. Comparing our seed data set contents 
with the data types used in the selected studies, We chose to use columns containing 
person names, dates and phone numbers.

The Schools data set we are using as the basis for the experiment contains person 
names in the “headteacher” column in a wide variety of formats, since the data provider 
has not attempted to regularise the formats. We selected this column as the seed data 
set for one group of experiments. For the ground truth, we manually created a version 
of the data set in which all names are formatted consistently, using an initial, full 
stop, capitalised surname (as in B.Obama) as its desired format for transformation. 
Examples of the different name formats that exist in our seed data set for this data type,

\(^2\)https://github.com/umich-dbgroup/foofah  
\(^3\)https://github.com/Yeye-He/Transform-Data-by-Example  
\(^4\)https://github.com/minhtpx/ieee-bigdata2019-transformation  
\(^5\)https://microsoft.github.io/prose/  
\(^6\)http://bit.ly/1GtZ4Gc
CHAPTER 5. INFERRING FORMAT TRANSFORMATION RULES

<table>
<thead>
<tr>
<th>Seed Data</th>
<th>Formatted Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben Kenyon</td>
<td>B.Kenyon</td>
</tr>
<tr>
<td>Gina O’Conor</td>
<td>G.O’Conor</td>
</tr>
<tr>
<td>Branwen Hywel</td>
<td>B.Hywel</td>
</tr>
<tr>
<td>A K Perks</td>
<td>A.Perks</td>
</tr>
<tr>
<td>E Davidson</td>
<td>E.Davidson</td>
</tr>
</tbody>
</table>

Table 5.5: Names

<table>
<thead>
<tr>
<th>Seed Data</th>
<th>Formatted Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>01617705333</td>
<td>(44)01617705333</td>
</tr>
<tr>
<td>01885483238</td>
<td>(44)01885483238</td>
</tr>
<tr>
<td>01787228344</td>
<td>(44)01787228344</td>
</tr>
<tr>
<td>02084430708</td>
<td>(44)02084430708</td>
</tr>
<tr>
<td>01344426413</td>
<td>(44)01344426413</td>
</tr>
</tbody>
</table>

Table 5.6: Phone Numbers

<table>
<thead>
<tr>
<th>Seed Data</th>
<th>Formatted Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>28/12/2011</td>
<td>2011-12-28</td>
</tr>
<tr>
<td>1/4/2019</td>
<td>2019-04-01</td>
</tr>
<tr>
<td>23/09/2018</td>
<td>2018-09-23</td>
</tr>
<tr>
<td>01/06/2015</td>
<td>2015-06-01</td>
</tr>
</tbody>
</table>

Table 5.7: Dates

and their target format are given in Table 5.5. This data type represents a challenging format transformation objective for the tool.

To contrast with this data type, we selected two further data types with different challenges. We selected the Phone Number column to give an example of a largely numerical type with minimal formatting requirements. The Schools data set was already formatted consistently for this data type, giving us an example seed data set containing a very small number of representative format cases (in contrast with the Headteacher column). We chose a target format that prepends the country code (all UK) in brackets to the numbers, to again give a relatively easy challenge for our method. Examples of formats for Phone Numbers available in the data set and the desired correctly formatted values are presented in Table 5.6.

As an intermediate challenge between these two we selected one of the Date type columns. This data gives an example of a more complex format transformation task and, although the data is largely consistently formatted, there are several representative cases for transformation present in the data (due to the different choices being made as to whether to use leading zeroes in the day and months or not). For the target format, we chose a textually very different format of YYYY-MM-DD (as in 2019-09-22). Examples of the date formats present in the seed data and their target formats are shown in Table 5.7.

Data Set Size  Based on our examination of the experimental data sets available in publications’ open-source repositories, data sets used range from 3 to 300 rows.
Following this review, we selected our experimental configurations as given below:

**The seed data sets that the experiment will be run against.** As described above, we ran the experiment over three data types, presenting different degrees of transformation challenge: phone numbers, dates and people’s names.

**The size of the before data set in rows.** We chose to run the experiments over several contrasting data set sizes, to be able to see the effects of the example selection strategies in the presence of different amounts of manual correction. We selected data sizes of 50 rows, 100 rows and 200 rows.

**The proportion of rows in the before data set that will be modified in the after data set.**
We vary the number of rows that are manually corrected, starting at 5% of rows to 10%, 20%, 30%, 40% and finally 50%.

**The proportion of modifications that correctly match the ground truth formatting rule.**
This parameter is a combination of two choices for the experimental harness: the proportion of correctly formatted rows (from the ground truth data set) that will be included in the *before-edit* data set and the proportion of injected incorrectly formatted rows (set in the previous parameter) that will be modified by the simulated manual corrections. Across the data sets of each size of 50, 100 and 200 rows, we cumulatively increase the proportion of correctly formatted values in the *after-edit* data set starting with 5%, followed by 10%, 20%, 30%, 40% and finally 50%. While for the correctly formatted values in the *before-edit* data set started with 0% of correctly formatted values in the first run and increase it 5%, 10%, 15%, 20% and finally 25%. As the proportion of manual corrections increase, the correctly formatted values in the *before-edit* data set also increases by half the size of the manual correction. For justification, the reasons to have correctly formatted values on the *before-edit* data sets are first to determine whether the learned transformation rules would transform the originally correctly formatted values to an incorrect format and second to imitate real world scenarios as in practice data sets are usually composed of a mixture of correct and incorrect formats.

**The threshold used by the validation component for accepting or rejecting rules.**
We used a threshold of 0.6 accuracy for rule acceptance.
The number of times the experiment will be run for each configuration. We chose to run the experiment five times for each configuration. The harness reports the mean rule accuracy score for each group of five runs.

5.3.3 Results

For each data type, we created three result sets showing the behaviour of each of the three example pair generation strategies (null filtering (baseline), edited value filtering and augmented edited value filtering). Each result set gives the transformation accuracy scores (averaged across five runs) for each of the six different percentages of manually corrected rows, for the three data sets sizes. Below, for clarity, we present the data in summary graphs. Across five runs, the values of average (mean), min and max are depicted in Appendix B.1.

Results for Name Data Type

For the Name data type, the results for all three strategies are presented in Figure 5.3. Figure 5.3(a) shows the performance of the null filtering strategy (which also acts as the baseline), 5.3(b) shows performance of the edited value filtering strategy and 5.3(c) shows the performance of the augmented edited value filtering strategy. For the null filtering strategy, as shown in Figure 5.3(a), as the number of manually edited rows increases, better performance can be seen in terms of the transformation score. This outcome is broadly consistent across the different data set sizes, though the larger data sets show slightly better performance than smallest. This is expected since the number of manually corrected rows is a function of the number of rows in the starting data set, so the larger data sets have more corrected rows than the smaller ones. A similar situation can be seen for the augmented edited value filtering strategy in Figure 5.3(c). While we did not observe significant differences in terms of data set size, the proportion of corrected rows does play a role in the quality of the outcome. For both these strategies, the transformation score increases as the percentage of corrected rows increases.

The edited value filtering strategy, however, performs surprisingly poorly across all data sets sizes (Figure 5.3(c)). Here, the performance gradually decreases as the proportion of corrected rows increases. On investigation, we found that this is because when the format transformation rules learnt by this strategy are applied to the after-edit data sets, a significant proportion of the manually corrected values were transformed to the empty string.
5.3. EVALUATION OF THE GENERATION STRATEGIES

We wanted to see whether this was a feature of the specific implementation of Flash Fill that we were using, and so tested the example pairs generated on the most up-to-date commercial implementation of Flash Fill that we had access to (embedded in Microsoft Office Excel 2013). We observed the same empty string behaviour with that version of the tool. A possible explanation might be that, in Flash Fill, if an input value is supplied that does not match against any of the cases considered by the learnt rule then an empty string is output, rather than just outputting the input value unchanged.

These results for the simple edited filtering strategy confirm what we discovered when designing the strategies (and that led to us proposing the augmented edited value filtering strategy): that it is important to supply examples of correctly formatted values as well as incorrect-to-correct formatting examples. Adding more editing rows to the data set simply created more negative examples for the simple edited filtering strategy, while not providing any examples of correctly formatted values. Hence, its performance decreases with additional editing.

Overall, the augmented edited value filtering strategy demonstrated much better results than the other strategies for this data type, including having the desirable property of becoming more accurate as more manual corrections are provided.

Moreover, this strategy performs equally well on all data sizes and with comparatively small amounts of manual correction. At just 20% manually corrected rows, augmented edited filtering is already consistently showing a transformation score of above 0.6, the value we selected for the ManEd harness to accept the transformation rules. In contrast, the null filtering strategy shows transformation accuracy increasing more slowly in proportion to the amount of manual correction. The cause of this is probably the fact that since null filtering uses all rows as input-output example pairs, not all example pairs are valid and useful values to generate good transformation rules.

Further analysis on results of individual rows reveals that correctly formatted values injected into the before-edit data set retained their format for all three strategies, indicating that the learned rules manage to preserve correctly formatted values even when the values are not provided as input-output of example pairs.

Results for Dates Data Type

We now turn to the performance of Dates data type, the results for which are shown in Figure 5.4. This data type was selected as presenting a transformation challenge of intermediate difficulty, between Names and Phone Numbers.
CHAPTER 5. INFERRING FORMAT TRANSFORMATION RULES

(a) Null filtering strategy

(b) Edited filtering strategy

(c) Augmented filtering strategy

Figure 5.3: Transformation Rule Scores for the Names Data Type

The performance of the null filtering strategy for Dates is given in Figure 5.4(a), the edited value filtering strategy performance in Figure 5.4(b) and the augmented edited value filtering strategy in Figure 5.4(c). From these results, we can see that the transformation scores obtained from both the null and augmented edited value filtering strategies display similar patterns as observed for the Names data type, although the
augmented edited value filtering strategy performs less well for Dates at lower manual correction rates than it did for Names. For Dates, the transformation rule scores obtained from augmented edited value filtering rise more gradually as the percentage of manually corrected tuples increase and from a lower starting point, closer to the behaviour of the null filtering strategy.

Figure 5.4: Transformation Scores for the Dates Data Type
CHAPTER 5. INFERRING FORMAT TRANSFORMATION RULES

This finding was unexpected; the date transformation challenge presented by our data set is much more straightforward for the human brain than the name formatting challenge, and we expected it to be easier for ManEd to handle. The results could suggest that the transformation rules are really only transforming results that have been provided as example pairs correctly, and hence the results improve as more such pairs are given to the rule learner to work from. However, we did find that a small number of manually corrected tuples were not transformed as desired by the learned results.

Interestingly, although the null and augmented filtering strategies did not manage to generate as good transformation rules for Dates as for Names, they certainly did not make the rules any worse for the correctly formatted values injected in the before-edit. All these were transformed correctly, across all strategies and data sizes.

For simple edited filtering, again we see a pattern of gradually decreasing accuracy as the proportion of manual corrections increases, though starting from a much lower level of accuracy. At 5% of rows being manually corrected, the average transformation score is 0.4 and declines as the percentage of manual correction increase.

One possible explanation for the difference in results between the data types may be caused by how the Flash Fill algorithm works based on a given set of input-output example pairs. Because input-output example pairs for Dates could be ambiguous because the date values could have multiple interpretations (such as mm/dd/yyyy or m/d/yyyy or d/mm/yyyy), and Flash Fill uses joint learning for data type interpretation using other string values that exist in the spreadsheet. As a result, there is a possibility the synthesised program generates undesired output for Dates [SG16].

It is possible, therefore, that the overall lower performance of the strategies is dominated by a feature of the Date type as handled by our Flash Fill implementation, while the influence of our filtering strategies is shown in the overall shapes of the graphs.

Results for the Phone Number Data Type

For the phone number data type, the results for all three strategies are as presented in Figure 5.5. Figure 5.5(a) shows the performance of the null filtering strategy, while 5.5(b) shows the performance of the edited value filtering strategy, and lastly 5.5(c) shows the performance of the augmented edited value filtering strategy. As shown on all three line charts, and similar to the Names data type, the results for each strategy are broadly the same for all sizes of data set. In addition, the graphs for all three filtering strategies follow exaggerated versions of the same trends as we saw for the
Names data type. The results for the null filtering strategy shows a positive trend with transformation score rising with the amount of manual corrections made, while the results for and augmented edited value filtering quickly reach a maximum and remain there. And, for the edited value filtering strategy, the average transformation score shows declining performance as manual corrections increase, comparable to Names and Dates data types, regardless of data size. These results suggests that regardless of data type and size, the filtering strategies have consistent effects on the transformation score in terms of whether it increases or decreases, though some features of particular data types may change the overall level of accuracy and the slope of the increase or decrease as more manual corrections are made.

What stands out for the Phone Numbers data type is the performance of the augmented, edited value filtering strategy. Compared to the Names and Dates data types, where the transformation score average rose gradually as the percentage of manual correction increases, for the Phone Number data type, quite unexpectedly, the transformation score average of augmented edited value filtering strategy is at 1.0 even when the percentage of manually corrected tuples is as low as 5%. Another unanticipated finding from the outcome of the Phone Number data type is seen in the individual data for the five runs for each percentage of manual correction. We observed that each individual run gives a consistent transformation score. Thus, the minimum, maximum and average for all the five runs on each incremental percentage of manually corrected tuples have the same result for all filtering strategies. This can be seen in the full data set given in Appendix B.1.

We predicted that the augmented edited value filtering strategy for Phone Numbers would surpasses the performance for both the Names and Dates data types with this same filtering strategy due to the straightforward transformation our data set for this type requires, with a very limited number of representative cases to consider.

5.3.4 Discussion

Overall, we observe from these results that for some data types (Names and Phone Numbers) the augmented edited value filtering strategy gave the most useful behaviour out of the three strategies. This finding is in line with our expectation, since the augmented edited value filtering strategies was designed to make best use of the manual corrections provided by the data scientist. However, despite exhibiting similar trends across all filtering strategies and data sizes to the other data types, the results for the Dates data type, unfortunately, give a lower overall score across all strategies, with
CHAPTER 5. INFERRING FORMAT TRANSFORMATION RULES

(a) Null filtering strategy

(b) Edited filtering strategy

(c) Augmented filtering strategy

Figure 5.5: Transformation Scores for the Phone Number Data Type

even augmented filtering giving broadly the same performance as null filtering.

Further investigation on the results for the Dates data type shows that the learned transformation rules only managed to transform data values that were provided as example pairs. Data values that are not provided as example pairs will either be returned as an empty string or transformed into incorrect formats. We also found that a small number of the manually corrected values are also transformed into incorrect formats.
5.4. VALIDATING THE LEARNED TRANSFORMATION RULES

However, this happened sufficiently rarely that it does not materially affect the results of the transformation score in general.

This was a surprising result as we were expecting the Dates data type to be of intermediate difficulty as a transformation task, but it appears that some features of the specific format transformation it requires were not handled well by our implementation of Flash Fill.

On the other hand, the good performance of null and especially augmented filtering for the Phone Number data type across all data sizes was as we expected, and most likely results from the limited number of representative formats in the Phone Number data set. Therefore, further experimentation needs to be undertaken before the association between representative format and performance of learned rules is more clearly understood. In addition, further study with more focus on other data types may also be needed to understand the effects of the specific implementation of the transformation rule learner component when trained on results from manual correction.

Our results suggest that the augmented edited value filtering strategy gives the best performance overall, while edited filtering should be avoided. One surprising outcome of our experiments is that null filtering (which we expected to give the worst performance) actually performs better than edited filtering, giving an improvement in the transformation score as more manual corrections are provided. This shows the importance of providing correctly transformed values (that should not be changed by the rules) when giving example pairs for the rule learner component. However, because of the observed differences in performance of all strategies for the different data types, we recommend that any implementation of our approach in a dataspace should allow users to switch this transformation rule learning facility off for certain data types, so that it is only used in cases where it is giving useful results.

5.4 Validating the Learned Transformation Rules

5.4.1 The Rule Validation Problem

Even with a careful choice of example generation strategy, it is likely that some proportion of the values edited by the data scientist were changed for some reason other than in an attempt to regularise the formatting of the column. Guessing at which edits correspond to format changes is extremely challenging, with any approach likely to be error prone. There is therefore no guarantee that the transformation rules generated
from our examples will be reliable when applied to the whole data set. A further validation of the learned rules is needed to try to avoid unhelpful transformation rules from being output for use in data integration.

We cannot, in practice, assess the accuracy of a transformation rule without requiring more information from the data scientist. We could present the rules to the data scientist and ask them to select the one that most closely matches their needs. But Flash Fill rules are highly procedural in form, and are hard to understand. We cannot ask a non-technical user to validate them (or even a technical user, since the rules are complex and very low level). We might observe the actions of the user over several sessions, to see whether the effects of the rules we output are undone by the user in subsequent manual correction steps. But this does not help us to determine which rules to output in each new run of ManEd. We therefore need some other way to judge the likely value of the generated rules, based on the information we have available to us at run-time: only the original data set and the edited (manually corrected) data set.

If we consider the conditions needed for Flash Fill to give a good result, we see that an important consideration is whether the examples provided cover a representative set of format cases for the data set under investigation. If we can infer whether the set of examples we used to generate the rule is likely to give such good coverage, we can use that as a proxy for the quality of the rule. We do not have access to information about such cases, but since we are only interested in formatting and not semantics we can make a guess at what the cases might be by looking at the lexical structure of the values in the before-edit data set. For example, we saw earlier how a representative set of name examples would need to cover certain common name forms: one or more forenames, one or more initials, an optional name prefix, and one or more surnames, possibly joined by hyphens. Considered in terms of lexical tokens, these forms would have different representations: \texttt{WORD, SPACE, WORD}, for example, or \texttt{WORD, SPACE, WORD, HYPHEN, WORD}.

This suggests a possible approach to assessing whether a rule is based on a good coverage of representative examples or not. We can take the values in the before-edit data set, and calculate their lexical token structure. Once done, we can cluster the tokens, identifying data values which share a common structure. We can consider each cluster a representative case. At transformation rule generation time, we can then compare the set of examples on which the rule was built, and compare it to the set of lexical token clusters in the input file space. We say that a rule is based on representative cover if the examples from which it was generated contain at least one
5.4. VALIDATING THE LEARNED TRANSFORMATION RULES

example of every such cluster in the before-edit data set.

We implemented this approach, and carried out an experiment to determine whether
the number of covered lexical cases could act as a suitable proxy measure for the ac-
ccuracy of generated rules, where each case is considered “covered” if one example
matching that lexical case is in the set of examples used to generate the rule. We also
set out to test whether one example of each case was sufficient, or whether having
more examples covering more cases more often was beneficial, and if so how many
examples are needed to gain good reliability of rules.

We will now describe the approach to testing rules for their format representative-
ness in the next section.

5.4.2 Identifying Representative Cases in ManEd

To study the feasibility of estimating representative case coverage as a proxy for rule
validity, we implemented the approach in the Rule Validation component of the ManEd
pipeline (Figure 5.2). We used a simple tokenising strategy shown in Table 5.8, which
distinguishes strings of characters unbroken by white space or punctuation as the main
token form. We distinguish sequences consisting only of digits, sequences consisting
only of lower-case alphabetic letters, sequences consisting only of upper-case alpha-
betic letters, sequences consisting of alphabetic letters of any case, and sequences con-
sisting only of white space characters. We add to these basic tokens additional tokens
distinguishing punctuation elements, with each punctuation character acting as its own
token.

<table>
<thead>
<tr>
<th>Num</th>
<th>Elements Description</th>
<th>Character Elements</th>
<th>Token Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NUM</td>
<td>[0-9]+</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>WORD</td>
<td>([A-Z]+[a-z]+)+</td>
<td>w</td>
</tr>
<tr>
<td>3</td>
<td>LWRD</td>
<td>[a-z]+</td>
<td>l</td>
</tr>
<tr>
<td>4</td>
<td>UWRD</td>
<td>[A-Z]+</td>
<td>u</td>
</tr>
<tr>
<td>5</td>
<td>SPACE</td>
<td>s+</td>
<td>s</td>
</tr>
</tbody>
</table>

Values in the before-edits data set are converted into strings of these tokens. This
process is illustrated in Table 5.9, which shows the before-edit values in our running ex-
ample together with their tokenised forms. While more sophisticated similarity-based
clustering algorithms can be envisaged, we use a very simple scheme that counts each
distinct sequence of tokens as a separate representative case. This very likely over-estimates how many representative cases there are, splitting some cases into multiple narrower cases. This may lead to good transformation rules being discarded, but is unlikely to result in poor-quality rules mistakenly being seen as strong, and then being passed on to the data integration system.

Table 5.9: Example of before-edit dataset and its corresponding representative cases

<table>
<thead>
<tr>
<th>Num</th>
<th>Before-Edit (BE)</th>
<th>BE Representative Cases</th>
<th>Case Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>wsw</td>
<td>M_A</td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>wsu’w</td>
<td>M_B</td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>wsw</td>
<td>M_A</td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>ususw</td>
<td>M_C</td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>usw</td>
<td>M_D</td>
</tr>
<tr>
<td>6</td>
<td>Amy Jim-Wilson</td>
<td>wsw-w</td>
<td>M_E</td>
</tr>
<tr>
<td>7</td>
<td>N. Buckhardt</td>
<td>u.w</td>
<td>M_F</td>
</tr>
</tbody>
</table>

For convenience of explanation, we label these cases with identifying letters, shown in the final column. We can see that there are six representative cases in our example data set. We next want to see whether the set of example pairs, generated by the first component of the ManEd pipeline, is a good cover for all these cases. This is easily checked by converting the examples themselves into their tokenised form, in the same way that the before-edits values were. Table 5.10 shows the tokenised example pairs extracted from manual corrections using the augmented edited value filtering strategy. As mentioned in Section 5.2.1, this strategy creates not one but two example pairs from each manually corrected cell in the column of interest. One pair contains the before-edit and after-edit values, while the other pair consists of the after-edit value paired with itself. Both examples are used to generate tokenised forms.

The tokens from the example pairs can now be compared with the tokens generated from the before-edits data set. Any omissions may indicate that the generated example pair does not cover the representative cases of the data set well, and therefore may not be suitable for generating transformation rules. In our example, only three of the six representative cases are covered by the generated examples, and therefore the rule learnt from them may not be of high quality and should not be passed on to the data integration system.

To check this, we examined how the rule learnt from the example pairs fared when applied to the before-edits data set in this small example. The table below shows the results. It can be seen that the rule is unable to transform all of the results. Only the names that were already manually corrected were transformed accurately, while some
5.4. VALIDATING THE LEARNED TRANSFORMATION RULES

Table 5.10: Example of before-edit dataset and its corresponding representative cases

<table>
<thead>
<tr>
<th>Rows</th>
<th>Before-Edit (BE)</th>
<th>AfterEdit (AE)</th>
<th>Example Pair (EP)</th>
<th>EP Representative Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>B.Kenyon</td>
<td>Ben Kenyon, B.Kenyon</td>
<td>wsw,u.w</td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>G.O’Conor</td>
<td>G.O’Conor, G.O’Conor</td>
<td>wsu’w,u.w</td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>B.Hywel</td>
<td>Branwen Hywel, B.Hywel</td>
<td>wsw,u.w</td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>A.Perks</td>
<td>A K Perks, A.Perks</td>
<td>ususw,u.w</td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>E.Davidson</td>
<td>E Davidson, E.Davidson</td>
<td>usw,u.w</td>
</tr>
<tr>
<td>6</td>
<td>Amy Jim-Wilson</td>
<td>A.Jim-Wilson</td>
<td>Amy Jim-Wilson, A.Jim-Wilson</td>
<td>wsw-w,u.w</td>
</tr>
<tr>
<td>7</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt</td>
<td>N.Buckhardt</td>
<td>u,w,u,w</td>
</tr>
</tbody>
</table>

Table 5.11: Representative Cases in the before-edit data set and example pairs

<table>
<thead>
<tr>
<th>Representative Cases in Data Set</th>
<th>Representative Cases in Example Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsw</td>
<td>wsw</td>
</tr>
<tr>
<td>wsu’w</td>
<td>wsu’w</td>
</tr>
<tr>
<td>u.w</td>
<td>u.w</td>
</tr>
<tr>
<td>ususw</td>
<td></td>
</tr>
<tr>
<td>usw</td>
<td></td>
</tr>
<tr>
<td>wsw-w</td>
<td></td>
</tr>
</tbody>
</table>

of the other names are transformed to the empty string. The name in Row 7, however, is correctly transformed; note that although it was not manually corrected by the user, its tokenised form matches one of the forms covered by the example pairs set, lending support to our hypothesis.

The approach we have taken here is quite strict, requiring one example pair for each tokenised form. It is possible that we can manage with rather fewer examples than this in many cases. We may be able to set a threshold for representative case coverage, so that every rule that reaches that threshold is passed to the data integration system, and every rule that does not reach it is rejected. In the next section, we discuss how we have evaluated this rule validation approach, and look for evidence as to what the coverage threshold should be in practice.
Table 5.12: Transformation result when only parts of the representative cases are covered in example pairs

<table>
<thead>
<tr>
<th>Rows</th>
<th>Before-Edit</th>
<th>Before-Edit Cases</th>
<th>Example Pair</th>
<th>After-Edit Cases</th>
<th>Transform Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>wsw</td>
<td>Ben Kenyon, B.Kenyon, B.Kenyon</td>
<td>wsu,u.w</td>
<td>B.Kenyon</td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>wsu’w</td>
<td>G.O’Conor, G.O’Conor, G.O’Conor</td>
<td>wsu’w, u.w</td>
<td>G.O’Conor</td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>wsw</td>
<td>Branwen Hywel, B.Hywel, B.Hywel</td>
<td>wsw,u.w</td>
<td>B.Hywel</td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>ususw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>usw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Amy Jim-Wilson</td>
<td>wsw-w</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>N.Buckhardt</td>
<td>u.w</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.5 Evaluation of Rule Validation Based On Representative Case Coverage

We now describe our approach to evaluating the learned rules based on the representative cases that exist in the before-edit data set and examples pairs. Since the Date and Phone Number data sets we created for the previous set of experiments contained very few representative cases, we focused our attention on the Names data set for this experiment.

5.5.1 Extension to the Experimental Harness

Experimental evaluation of this approach to rule validation requires that our experimental harness should allow us to control the representative cases present in the input data set, so that we can vary the extent to which they are covered by the manual corrections artificially generated by the harness. This requires us to generate labelled input data sets for the experiments that include known sets of representative cases. We cannot use the simple form of representative case we use in ManEd, based on the lexical tokens, as this would bias the experiments. Instead, we need some independent way of determining the representative cases. If there was another easy way of doing this programmatically, we would have used it in ManEd (perhaps comparing it with the lexical token approach we are already using). Instead, therefore, we chose to make a manual assignment of cases for the input data for the experiments, using domain knowledge about the data type in question (people’s names). The author examined each of the values in the input data set, and classified the different parts of the name into first name, middle name, middle initial, last name, etc. This is illustrated in Table 5.13, where the
identified cases are shown for the names in our running example. Again, as a convenience for the purposes of explanation, we assign each case a letter, shown in the final column of the table.

Table 5.13: Representative cases of the dataset in experimental harness expressed by case category

<table>
<thead>
<tr>
<th>Rows</th>
<th>Before-Edit</th>
<th>String Pattern</th>
<th>Representative Case Category</th>
<th>Case Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ben Kenyon</td>
<td>first name, space, last name</td>
<td>fsl</td>
<td>U_A</td>
</tr>
<tr>
<td>2</td>
<td>Gina O’Conor</td>
<td>firstname, space, lastname, punctuation</td>
<td>fslp</td>
<td>U_B</td>
</tr>
<tr>
<td>3</td>
<td>Branwen Hywel</td>
<td>first name, space, last name</td>
<td>fsl</td>
<td>U_A</td>
</tr>
<tr>
<td>4</td>
<td>A K Perks</td>
<td>initial first name, space, initial middle name, space, last name</td>
<td>ifsimsl</td>
<td>A_C</td>
</tr>
<tr>
<td>5</td>
<td>E Davidson</td>
<td>initial first name, space, last name</td>
<td>ifsl</td>
<td>U_D</td>
</tr>
<tr>
<td>6</td>
<td>Amy Jim-Wilson</td>
<td>first name, space, last name, punctuation</td>
<td>fslp</td>
<td>U_B</td>
</tr>
<tr>
<td>7</td>
<td>N. Buckhardt</td>
<td>initial first name, dot, last name</td>
<td>ifpl</td>
<td>U_E</td>
</tr>
</tbody>
</table>

It will be noted that we have slightly fewer manually-identified representative cases than ManEd generates. This is expected, and is related to the fact that the cases identified by lexical tokens are very fine-grained, and (as mentioned) probably split some of the cases into smaller sub-cases.

On the assumption that input data sets can be provided that are annotated with these manually identified cases, the experiment harness was extended to allow the user to set:

- the number of manually-identified representative cases in the before-edits data set, and
- the proportion of rows of each case that will be corrected manually by the simulated user.

User are able to set the number of manually-identified representative cases in the before-edits data set through the system harness that we built using a Java program. Based on the percentage the user specified, the harness computes a set of representative cases the data sets will cover before manual correction takes place.

5.5.2 Experiment Design

We now describe the experiment we undertook using the extended harness to evaluate our approach to rule validation. If the use of lexical tokens can successfully approximate the representative cases in the data, then we would expect manually corrected data sets with higher coverage of the real representative cases to result in transformation rules being learned that are more accurate, and manually corrected data sets with
lower coverage of the real representative cases to produce less accurate transformation rules.

As mentioned earlier, for this experiment we focus on the Names data set used in the previous experiment. The Phone Numbers data set contains only one representative format, while the Dates data set has two. Therefore, these data sets do not allow us to vary the number of representative cases covered in any meaningful way.

For this experiment, we configured the harness to work with the Names seed data set, as before. However, since this experiment needs a data set with each value labelled with its user-identified case, we needed to create some dedicated seed data sets. We created three such sets, as follows. We allowed the harness to select three subsets of rows from the seed data set. Each of these were captured and stored for use in the experiments. Each value in these data sets was labelled with the user-identified representative case that it best matches.

This gave us three randomly selected, labelled seed data sets. A set of experiments was run against each one, as described (in the configuration of the harness) below.

We have three data sets for each data set size, giving a total of nine data sets. The justification for such an approach is that if the result shows no significant differences across all three data sets in each data size, we are confident that a random selection of the data set does not have significant bias for or against the transformation rule. We predict that all different data sets for each data size will yield consistent results subject to the percentage of representative cases it covers.

We set the user-identified cases in the before-edits data set to start at full coverage, means that all the user-identified representative cases existing in the data sets will be included in all before-edits data sets that the harness creates. It is important to note that configuring the harness to give full user identified case coverage is not the same as having full coverage of the representative cases returned by ManEd. The coverage of user-identified cases can be set based on user preferences. In this particular experiment, we set the amount of case coverage to increase by 10% on each run. Increasing the number of user-identified cases during our simulated manual correction give more chances to have high coverage of the representative cases identified by ManEd.

We run the experiment five times on each of the three input data sets across three data sizes. Thus, we can be sure that the randomly selected sample covers fair distribution of population in the dataset.

On all three input data sets of different data sizes, the harness selects at random the rows that will have correctly formatted values, the rows that will have incorrectly
5.5. EVALUATION OF RULE BASED ON REPRESENTATIVE CASES

formatted values, and the rows that will be the subject of the simulated manual corrections. With each set of runs on each input data set, we cumulatively increase the number of user-identified cases that are affected by manual corrections by one row in each run.

For instance, if the before-edits data set contains values covering only four user-identified representative cases, then the first experiment will make manual corrections to four rows, one from each of the cases thus giving 100% coverage of the user-identified cases. By the last of the 5 experiment runs, the harness will be making five manual corrections for each case, giving a total of 20 manually corrected rows.

It is essential to highlight that the “full coverage of cases” referred to above means the coverage of the user-identified cases that is configured in our harness. The input data set the harness produces may cover quite a different number of ManEd’s own internal cases. For example, an input data set with 100% coverage of the user-identified representative cases may cover only 70% of the cases identified by ManEd using the simple lexical token approach, since (as discussed) the latter are typically more fine-grained than the user-identified cases.

For each run, users are allowed to configure:

- The number of user-identified cases to be covered in the generated query result (input data set). We selected full coverage of 4 cases for the experiment (all the cases present in the input data set).

- The number of examples to be included in the query result (input data set) for each user identified case. For this experiment, we start with one example for every user-identified case and increase to 2, 3, 4 and finally 5 examples for every user-identified case included in the input data set.

In each experimental run, ManEd learns a transformation rule from the manual corrections generated by the harness, and the harness then calculates a score for its accuracy by comparing the results it produces with the ground truth data set.

5.5.3 Results and Discussion

We report the results of the experiment for each of the three seed data sets separately\(^7\). For each data set, at each size, we show the effect on the average transformation score

\(^7\)In this section, we give the graphs showing the results averaged across the repeated runs. Full data sets are given in Appendix B.2
of increasing the coverage of the user-identified representative cases (shown as the orange line in the graphs). For comparison with this, we also show ManEd’s own assessment of the proportion of representative cases that are covered, using its own identified cases based on the lexical token approach described in Section 5.4.2. Recall that our aim is to assess whether ManEd can use the proportion of ManEd-identified cases to decide whether the rules learnt from the input will be accurate or not, and to determine the case coverage threshold that should be used to configure ManEd to only return accurate learnt rules.

Figures 5.6, 5.7 and 5.8 show the results obtained for the three data sets, each in their three variants of different sizes.

The results are broadly consistent across each data set. They show that the average transformation score does indeed increase as the number of user-identified cases included in the input data set increases, ramping up slightly quicker with the first few increases in coverage per case and then slowing down. We can also see that the coverage of the ManEd-identified cases also broadly follows the amount of user-identified cases covered, though at less than full coverage. This reduced coverage, when we know that the user-identified coverage is 100% is to be expected, and is caused by the more fine-grained nature of the cases that are identified by ManEd. Raw data for the results is available in Appendix B.2.

The average scores for the learnt transformation rules improve in line with the proportion of representative cases identified by the ManEd system. For example, for all three sizes of data set, when the manual corrections cover more than 30% of representative cases of ManEd, the majority of runs show an average transformation score of more than 0.7, while with 40% of case coverage gives a high score in all cases, and a score of close to one for the 50 rows data sets as seen in Figure 5.6(a).

We can use these figures to find the coverage threshold at which transformation rules can be accepted for wider use or rejected. We can see from the charts that to gain a transformation rule that is correct for half the time, the manual corrections need to cover around 30% of the representative cases. If manual corrections cover 50% of the cases or more, then the average transformation scores rise sharply to between 0.8 to 1.0, and this can be seen in Figure 5.6(a) and 5.6(b). However, this is not happening for 200 rows data size, as the manual corrections only cover a maximum of 30% of the representative cases of ManEd.

However, the percentage values of representative cases across all data sizes are slightly different as can be seen it gets smaller as the data sizes increase from 50 rows
to 100 rows and 200 rows. For example, at three examples per user identified case, the percentage of representative cases on 50 rows data set reaches around 50%. On the contrary, it is only 40% for 100 rows data set. Subsequently, 200 rows data sets reach almost 30% of representative cases only. This finding is not surprising considering as the size of the data set increases, it is highly likely the representative cases that exist in the data sets also would increase. And because the number of examples per user identified case remains the same for all data sizes and incrementally increases by one example on each run up until five examples per case coverage. Therefore it is understandable why the percentage of ManEd representative cases is lower for 200 rows data size than 50 and 100 rows data size. However, a closer examination of the results of each runs shows that the average representative cases identified by ManEd for 50 rows are around 16 cases, 100 rows has an average of 17 ManEd representative cases and 200 rows data size have 28 representative cases on average. Based on these findings, we could interpret why the percentage of representative cases becomes lower across all data sizes.

Providing manual corrections for half the representative cases seems to be asking a lot from the user, mainly when data sets are highly diverse and/or large. Therefore, based on the results presented in this chapter, we propose a coverage cut-off of 30%, since below this transformation rules perform poorly, while above it, there is a chance that a good rule could be learnt. However, individual users of ManEd may wish to adjust the threshold higher, depending on the uses to which the data transformed by the rules will be put.

5.6 Conclusion

We have presented an approach to extracting examples for learning format transformations from manual corrections to data. The main idea is to obtain examples of format transformation corrections implicitly, from the work the data scientist is already doing with the data, rather than asking them to undertake an additional explicit data preparation task. The approach generates pairs of incorrect/correct values from the changes made to the data by the data scientist, using three different strategies: null filtering, edited value filtering and augmented edited value filtering. It then uses an implementation of the Flashfill algorithm as a black box to learn a transformation rule from the example pairs.
We evaluated the three strategies against one another, using data from an open government data set. The results show that the *augmented edited value filtering* strategy is promising since the transformation rules derive from this strategy manage to correctly transform manually edited datasets using transformation rules synthesised from example edits of transformation rules.

However, the proportion of manual corrections performed on the dataset significantly improves the correctness and reliability of the transformation rules. This is because, as more format examples are provided, more representative cases are likely to be included and better transformation rules will be learnt. Based on this observation, we explored the likelihood of using *representative cases* as another means of validating the transformation rules. Learnt rules are passed on for further use by the integration system if the examples they are derived from are found to give good coverage of the representative cases in the input data set. Otherwise, they are thrown away. The coverage is approximated by using a simple lexical tokenising system to identify groups of similarly formatted values. We found that this was a reasonable validation strategy, with our experiments showing that rule accuracy scores increased as the coverage of representative cases by the corrections increased. We also used an experiment to determine the threshold for accepting the learnt rules. High quality rules can be learnt if just 50% of the representative cases are covered by the examples, but good results can also be found with a more reasonable 30% coverage.

In our future work, we will build on this foundation to understand how this approach can be applied across multiple columns, taking account of corrections made by data scientists over multiple interactions with the data set and, eventually, will look at combining corrections from multiple data scientists working on the same type of data. There is also an open question regarding how we distinguish manual corrections that fix formatting problems (which should be included in the example pair generation) from those that fix one-off data defects such as spelling errors and data incompleteness (which should not be included in the example pairs).
5.6. CONCLUSION

(a) 50 rows

(b) 100 rows

(c) 200 rows

Figure 5.6: Effect of Increasing Case Coverage on Average Transformation Score for Data Set 1
CHAPTER 5. INFERRING FORMAT TRANSFORMATION RULES

Figure 5.7: Effect of Increasing Case Coverage on Average Transformation Score for Data Set 2
5.6. CONCLUSION

Figure 5.8: Effect of Increasing Case Coverage on Average Transformation Score for Data Set 3
Chapter 6

Inferring Data Refresh Rules from Manual Correction - A Case Study

“The only thing that is constant is change.”

Heraclitus

In the previous two chapters, we studied the potential usage of manual corrections as a source of implicit feedback in dataspace integration related-tasks. In Chapter 4, we showed how to use the domain information contained in manual corrections to improve the dataspace integration at the systematic schema level, by inferring true/false positive feedback from manual corrections that can be used to improve query mappings. And in Chapter 5, we showed how manual corrections can assist with a systematic data-level integration task, by extracting training data from manual corrections for use by components that learn format transformation rules for use during data cleaning.

In this chapter, we turn our attention to the problem of handling unsystematic data level integration issues. These relate to the handling of specific data values. They are particularly challenging as they cannot be handled through generic rules; instead, during data fusion, the dataspace must recognise these special case values and somehow work out how to transform and include them within query results correctly.

The specific problem we focus on is the data refresh problem: situations where an integrated query is run repeatedly at regular intervals, over data sources whose contents change over time. When a query is rerun, much of the data will be the same as when it was run the previous time, but there will be updates, additions and deletions to take into account. Any unsystematic data level problems will typically have to be fixed by hand, since data integration systems don’t deal well with this kind of problem at
present. And when the query is run again, the same problems will reappear. In some cases, the data scientist can report the change to the owner of the original source data that the query result was computed from, and the problem will be fixed at source and not be seen again. But processes for such fixes can be slow or non-existent. And some of the unsystematic data level changes we are talking about are a product of the integration, not the quality of the source data. For example, the source database might classify a new model of fridge as "white goods", but for the integration it needs to be considered as a "smart appliance". This change will need to be made on the data every time, because the classification of the fridge in the source is correct for the other users of the source. This repeated fixing of problems already encountered and solved can be very time consuming for the data scientist.

We set out to discover whether we could use our manual correction approach to allow the dataspace to detect these unsystematic changes and to reapply such corrections that are still applicable to the query result each time it is refreshed. The data scientist has to put effort into fixing these problems manually when they are first encountered, to make the data fit for whatever purpose it is intended for, but after that the dataspace can take up some of the work. Thus, the data scientist gets extra long-term benefit from any time spent manually correcting the query results, and the task of working with future refreshed versions of the query results is made less time-consuming.

The rest of the chapter is organised as follows, In Section 6.1 we present our research motivation with a scenario example. Section 6.2, describes the architecture of Data Refresh Rules we propose. Then, we continue to describe the detection of changes from manual corrections and how we can store the changes for future use in a knowledge-base in Section 6.3. Description of the experiment design is given in Section 6.4. Explanation of the experiment process is given in Section 6.5. We present the results of the case study evaluation in Section 6.4. Finally, Section 6.7 describes the threats to validity of the experiment and concludes the chapter.

6.1 Research Motivation

We propose to re-apply changes from manual correction made on a query result by a data scientist. In this section, we describe and motivate our approach, which is called “Data Refresh Rules”.

Consider a scenario in which a data scientist is working to prepare an analysis report for an upcoming board meeting on staff performance across all the company
CHAPTER 6. INFERRING DATA REFRESH

branches. The information required for the analysis purposes comes from a variety of databases, covering different business functions and different geographic regions. The analyst therefore uses a dataspace to perform the integration for them. For simplicity, we start with data sets *Employee* and *Performance*, produced by two integrated queries, as illustrated in Tables 6.2 and 6.1. The *Performance* table is assessed by the data scientist to ensure that the data is accurate and satisfies the needs of her current analysis tasks. The relationship between the *Employee* and *Performance* tables are established through the primary key/foreign key relationship of column *NIN* (short for National Insurance Number), which associates employee profiles to their performance results.

Because this data is created by integration from multiple sources, there may not be a complete match between the primary key column of *Employee* and the foreign key of *Performance*. There may be differences in formats or structuring of records and the presence of homonyms and synonyms may cause inaccuracies and inconsistencies. For example, Table 6.2 contains two representation of “Branwen Hywel” (an example of a synonym), appearing as “Branwen Hywel” in tuple 3 (with ID 102) and “B.Hywel” in tuple 8 (with ID 107). These two name values can be interpreted as matches through a string similarity approach, which is done automatically and therefore not always correctly. Another example is when a string similarity approach may identify “M.Robinson” in tuple 6 (with ID 105) and “M.Robinson” in tuple 9 (with ID 108) as the same person or struggle to identify any match at all in the case when the primary key (NIN information) is missing in either query result.

Table 6.1: Employee Information

<table>
<thead>
<tr>
<th>EMPID</th>
<th>NIN</th>
<th>EmpName</th>
<th>DOB</th>
<th>PhoneNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP_001</td>
<td>QQ123456C</td>
<td>Michelle Robinson</td>
<td>5-Jan-1964</td>
<td>07-404956690</td>
</tr>
<tr>
<td>EMP_002</td>
<td>BR223456D</td>
<td>Gina O’Conor</td>
<td>13-Feb-1972</td>
<td>07-404212234</td>
</tr>
<tr>
<td>EMP_003</td>
<td>XD334768S</td>
<td>Branwen Hywel</td>
<td>21-Mar-1962</td>
<td>07-787654428</td>
</tr>
<tr>
<td>EMP_004</td>
<td>BN879995H</td>
<td>Amanda Perks</td>
<td>23-May-1977</td>
<td>07-943234455</td>
</tr>
<tr>
<td>EMP_005</td>
<td>VG998751G</td>
<td>Endrew Davidson</td>
<td>19-Dec-1967</td>
<td>07-545675530</td>
</tr>
<tr>
<td>EMP_006</td>
<td>RT215567W</td>
<td>Amy Jim-Wilson</td>
<td>19-Jan-1978</td>
<td>07-545321112</td>
</tr>
<tr>
<td>EMP_007</td>
<td>QQ123456C</td>
<td>Mary Robinson</td>
<td>5-Jan-1964</td>
<td>07-404956690</td>
</tr>
</tbody>
</table>
6.1. RESEARCH MOTIVATION

Table 6.2: Employee Performance

<table>
<thead>
<tr>
<th>ID</th>
<th>NIN</th>
<th>EmpName</th>
<th>KPI Group</th>
<th>Weightage</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>QQ123456C</td>
<td>M.Robinson</td>
<td>Sales Performance</td>
<td>80%</td>
</tr>
<tr>
<td>101</td>
<td>BR223456D</td>
<td>G.O’Conor</td>
<td>Sales Performance</td>
<td>78%</td>
</tr>
<tr>
<td>102</td>
<td>XD334768S</td>
<td>B.Hywel</td>
<td>Sales Performance</td>
<td>88%</td>
</tr>
<tr>
<td>103</td>
<td>BN879995H</td>
<td>A.Perks</td>
<td>Sales Performance</td>
<td>87%</td>
</tr>
<tr>
<td>104</td>
<td>VG998751G</td>
<td>E.Davidson</td>
<td>Sales Performance</td>
<td>82%</td>
</tr>
<tr>
<td>105</td>
<td>RT215567W</td>
<td>M.Robinson</td>
<td>Customer Service</td>
<td>92%</td>
</tr>
<tr>
<td>106</td>
<td>QG332151F</td>
<td>E.Davidson</td>
<td>Sales Performance</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>XD334768S</td>
<td>Branwen Hywel</td>
<td>Customer Service</td>
<td>85%</td>
</tr>
<tr>
<td>108</td>
<td>GG123456C</td>
<td>M.Robinson</td>
<td>Customer Service</td>
<td>85%</td>
</tr>
</tbody>
</table>

Not all problems of this kind can be fixed through an automated approach. Consider tuples 1 and 7 of Table 6.1 where several potential candidates exist between these two tuples, (“M.Robinson”, “Michelle Robinson”) and (“M.Robinson”, “Mary Robinson”). The problem that remains to be solved is which of these two tuples is the correct match? In deciding which of the the tuples holds the right values, the knowledge of a domain expert who is aware of the employee information and their performance is usually helpful. Data scientists who may not have this knowledge can also carry out some groundwork in identifying which of the two tuples holds the correct values by going through other tables for references or directly asking the owner of the source data to determine the correct values. Often, data scientists manually modify the data values based on information gathered from external sources to make the data fit for its intended purpose, as illustrated in Figure 6.1.

When an updated version of this report is next requested (for the next board meeting, for example) a new version of the query results will be provided by the dataspace. There may be some new records, and some of the records may have been deleted or updated, but many of the records may be exactly the same as when the query was previously generated, including the same unsystematic data issues. So the correction effort needs to be repeated. Ideally, data quality issues would be reported to the source data owner, who would fix them so they do not reappear. But this can be a slow process in a large enterprise environment, where data is passed from system to system through a network of constantly refreshing systems. And there may be different opinions on whether the issue is even an error in the context of the source system. The whole point of a federated data integration system is that the owners of data can continue to store it
in a form that meets their needs, while not blocking its use by other people.

6.2 From Manual Corrections to Data Refresh Rules

Figure 6.2 shows the architecture of our proposed Data Refresh Rule approach. The components in grey indicate the elements of the architecture that have been added to the manual approach shown in Figure 6.1. The role of the Editor— the data scientist/user of the query results who has a good understanding of the domain and who is preparing the data for some analysis task by correcting errors, omissions and inconsistencies—remains the same in both approaches.

In our version, we distinguish three versions of the query results. The process begins with the generation of the source data (SD for short): the original query results that the dataspace generates in response to the data scientist’s request. Next is the working data (WD). This is a copy of the source data that the data scientist is working on to make it fit for purpose. This process may take several hours, days or even weeks, depending on the data update rates and the frequency of the request for the report the data scientist is creating. The third is the confirmed version of the working data, that has been verified as correct by the editor and is to be used in the analysis task. We call this version the confirmed working data (or CWD for short). One copy of the CWD exists for each instance of the analysis problem that the data scientist works on.

Once the analysis task is completed, the dataspace can take over to gather information about the changes made by the data scientist, and to store them as refresh rules for the future. To do this, it compares the SD and the CWD to obtain the list of corrections that were deemed necessary by the data scientist in order to prepare the query result for its intended task. These are recorded in a knowledge base called the EditsKB, short for Knowledge-Base for Manual Edits.
6.3 DETECTING CHANGES FOR EDITSKB

The first time the Data Refresh Rules mechanism is used, the EditsKB will be empty and the dataspace will not be of much use. But after the data scientist submits the first CWD to the system, some refresh rules will be able to be created from the corrections made by the data scientist. The next time the query result is regenerated, the dataspace will be able to use the refresh rules to reapply the corrections from before that are still applicable, meaning there is less editing work that the data scientist needs to do. Once the new version of the CWD is created, more refresh rules can be recorded and the process can start again.

6.3 Detecting Changes For the EditsKB

We view the EditsKB as a file containing records of manual correction made to the query result in the past. (For convenience, we will sometimes use the term edits to refer to these stored corrections). For the purposes of this exploratory study, we assume that the manual corrections performed by the Editor are correct and are a true reflection of the information requirements underpinning the analysis task. Correct data in our context is a value confirmed and verified by the Editor.

Based on the approach of snapshot differentials proposed by Labio and Garcia-Molina [LGM96], we define the EditsKB file as a set of records:

\[ \{ R_1, R_2, \ldots, R_n \} \]

where each \( R_i \) is a record of a manual correction. Labio and Garcia-Molina further define each \( R_i \) to be of the form:
\langle K, B \rangle

where K is the key and B is the rest of the record representing one or more fields. Our goal is to automatically re-apply manual corrections on the next incoming SD to create a first version of the WD for the user, wherever applicable based on the list of manual corrections recorded in the EditsKB. Therefore, in our EditsKB, we make a slight modification of the representation proposed by Labio and Garcia-Molina. Each record, \( R_i \), may have one of the following three forms which represent the different kinds of manual correction performed on the records:

- **For Updates**: \( \langle \text{Update}, K, E, P, V_1, V_2 \rangle \)
  This form is produced when a record \( \langle K_1, V_i \rangle \) in the SD (an entity with id \( E \) and secondary key \( K \) and attribute values \( V \)) was updated so that one of its attributes (property \( P \)) has had its value changed from its old value \( V_1 \) to its new value \( V_2 \).

- **For Insertions & Deletions**: \( \langle \text{Insert}, K, E, V_2 \rangle \)
  This form is produced when a record \( \langle K, V \rangle \), with id \( E \), is present in CWD but not present in SD (i.e., the row was added since the last run) or vice versa (i.e. the row was present on the last run but has since been deleted).

The list of changes that is stored in EditsKB can be re-applied whenever needed. The Editor can choose whether to apply only selected changes, only the delete list, only the insert list, only the update list or apply everything in the changes. For each list, Editor can also choose which records in the list they wanted to re-apply.

### 6.4 Experiment Design

To get a better understanding of the potential benefits and limitations of this data refresh approach, we used a public data set that is regularly updated to estimate the time that is potentially saved with our approach. We were able to do this with only one system, and therefore do not claim that these results will be representative of the gains and losses from the use of Data Refresh on all systems. The results are only indicative of what benefits might be reasonably predicted in systems with similar properties to the one used in our test.

Our aim was to make some objective measurement of the time saved (or lost) when manual corrections to a changing data set are captured and reapplied automatically.
compared to the current situation, when the editor must reapply the manual corrections for themselves.

To achieve our goal, we need to be able to observe the refresh process, so that we can measure the edits that are made and capture them in our knowledge base for re-application on future versions. There are a number of possible options to set up this situation for the purpose of this experiment. Below we list some plausible approaches to acquire the data we need:

- **Real Editor**: We could recruit an editor and set up a task for the editor to correct the data, so that edits could be extracted and accumulated for the experiment. The recruited editor may start by learning what is the ground truth of the data set that they are asked to edit and use their knowledge on the ground truth to, later on, edit the provided data set accordingly. A new version of the data set would then be provided, and the editor would be asked to repeat the process of making it fit for purpose. However, this approach may seem unnatural and artificial since the editor may already know what to edit and expect. The care and attention paid to the edits may be different when the editor knows they are in an artificial experiment, compared to preparing data to fulfil a real task. Moreover, we did not have the resources needed to pay someone with real data scientist skills to complete what could be a lengthy task.

- **Simulate an Editor**: We could program an artificial editor that follows some rules to simulate the process of making and reapplying changes to an input data set. Since the editor in this case is a program, we can run it many times as we liked, and we could try to set up the simulated edits to correspond to what happened in real life. However, there is a strong risk of confirmation bias with this approach, since we would inevitably code certain kinds of editing patterns into the simulator, with no guarantee that these correspond to the kind of patterns experienced in real life.

- **A Retrospective Study Based on Open Data**: to try to assess our technique against real data update patterns, we could take an open data set that is publicly versioned, and use it as the basis of a retrospective study. This would involve choosing a starting version, and imposing a view onto it to simulate the integration process. We could then roll forward through the public versions to see the effects of storing and re-applying the corrections needed to convert the latest data version into the integrated view.
Based on the possible approach listed above, we proceed with the second approach where we simulate an editor. The simulated editor uses publicly versioned data sets.

6.4.1 Datasets

We decided to use data from the Universal Protein Resource\(^1\) (Uniprot for short) for our study. Uniprot is made up of three databases, known as the Uniprot Knowledge-base (UniProtKB), the UniProt Archive (UniParc) and the UniProt Reference Cluster (UniRef). UniProtKB itself comprises two components: Swiss-Prot, a database of protein information that is manually annotated by human experts, and TrEMBL, a database of protein information that has been automatically annotated by software.

Data from UniProt seems suitable for our case study in that:

- It is a real-world data that is curated by a human expert and is updated regularly (every four weeks). The means it is a significant source of real world data changes that we can use to evaluate our data refresh approach.

- UniProt keeps the history of every change made to individual protein records. This assists us in identifying individual changes made in the dataset so that we can simulate the manual work of the data scientist. For example, Figure 6.3 provides the history of changes that have been made to the protein with code Q96MT7.

---
\(^1\)www.uniprot.org
• UniProt is an open data set, so that we are able to access both its current and previous versions. (Many other open data sets make only the latest version available.)

### 6.4.2 Schema

For this experiment, we use data from UniProtKB-SwissProt that keep records of protein information that is of high quality as it is manually annotated and curated. Each protein record is known as an “entry”, and each entry includes information such as ID, protein name, date of creation and accession number. The protein entries are provided in a text file format through a web interface. Figure 6.4 provides an example of a protein entry in UniProt and Figure 6.5 provides a description for each line available in UniProt entries.

### 6.4.3 Assessing UniProt

Now, for each UniProt entry, every line starts with a two-character line code that represents the type of data contained in the line [Unia]. Figure 6.5 shows the line type, line codes and order it appear in an entry. Because every entry that is available in UniProt usually contains a large number of lines of code that contain information on protein entries, for simplicity, we only targeted specific information instead of all attributes that are available in the entries. To do this, we needed to first manually analyse the potential data that we will use for the experiment. In each version, different attributes may change, and every version has different sets of changes. Some versions may only have deleted information, while another version only consists of newly inserted entities. There are also cases where a version has both added and deleted information. However, based on our analysis of the protein data of UniProt, we realise that there is no specification for updates to entries. For this reason, we assume that whenever there is a delete follow by added protein information, with an identical line code name, we assume this was an update. Figure 6.6 shows an example of the addition of a new line with new attribute information. Figure 6.7 shows versions with a deleted line/attribute and Figure 6.8 shows changes in protein entries that we assume are an update.

We first identified the version of data that we wanted to use as the starting point for the experiment. We chose two protein data entries from the same species, *homo*
Figure 6.4: Example of a UniProt entry

sapiens. The protein entries are ‘Q9NPU4’\(^2\) and gene name ‘B0UUJ3’\(^3\). For each

\(^2\)https://www.uniprot.org/uniprot/Q9NPU4
\(^3\)https://www.uniprot.org/uniprot/B0UUJ3
Figure 6.5: Attributes of UniProt Entries (Taken from [Unia])

<table>
<thead>
<tr>
<th>Line code</th>
<th>Content</th>
<th>Occurrence in an entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Identification</td>
<td>Once; starts the entry</td>
</tr>
<tr>
<td>AC</td>
<td>Accession number(s)</td>
<td>Once or more</td>
</tr>
<tr>
<td>DT</td>
<td>Date</td>
<td>Three times</td>
</tr>
<tr>
<td>DE</td>
<td>Description</td>
<td>Once or more</td>
</tr>
<tr>
<td>GN</td>
<td>Gene name(s)</td>
<td>Optional</td>
</tr>
<tr>
<td>OS</td>
<td>Organism species</td>
<td>Once or more</td>
</tr>
<tr>
<td>OG</td>
<td>Organelle</td>
<td>Optional</td>
</tr>
<tr>
<td>OC</td>
<td>Organism classification</td>
<td>Once or more</td>
</tr>
<tr>
<td>OX</td>
<td>Taxonomy cross-reference</td>
<td>Optional</td>
</tr>
<tr>
<td>OH</td>
<td>Organism host</td>
<td>Optional</td>
</tr>
<tr>
<td>RN</td>
<td>Reference number</td>
<td>Once or more</td>
</tr>
<tr>
<td>RP</td>
<td>Reference position</td>
<td>Once or more</td>
</tr>
<tr>
<td>RC</td>
<td>Reference comment(s)</td>
<td>Optional</td>
</tr>
<tr>
<td>RX</td>
<td>Reference cross-reference(s)</td>
<td>Optional</td>
</tr>
<tr>
<td>RG</td>
<td>Reference group</td>
<td>Once or more (Optional if RA line)</td>
</tr>
<tr>
<td>RA</td>
<td>Reference authors</td>
<td>Once or more (Optional if RG line)</td>
</tr>
<tr>
<td>RT</td>
<td>Reference title</td>
<td>Optional</td>
</tr>
<tr>
<td>RL</td>
<td>Reference location</td>
<td>Once or more</td>
</tr>
<tr>
<td>CC</td>
<td>Comments or notes</td>
<td>Optional</td>
</tr>
<tr>
<td>DR</td>
<td>Database cross-references</td>
<td>Optional</td>
</tr>
<tr>
<td>PE</td>
<td>Protein existence</td>
<td>Once</td>
</tr>
<tr>
<td>KW</td>
<td>Keywords</td>
<td>Optional</td>
</tr>
<tr>
<td>FT</td>
<td>Feature table data</td>
<td>Once or more in Swiss-Prot, optional in TrEMBL</td>
</tr>
<tr>
<td>SQ</td>
<td>Sequence header</td>
<td>Once</td>
</tr>
<tr>
<td>(blanks)</td>
<td>Sequence data</td>
<td>Once or more</td>
</tr>
<tr>
<td>//</td>
<td>Termination line</td>
<td>Once; ends the entry</td>
</tr>
</tbody>
</table>

entry, we gathered updates from the five subsequent versions.

We accessed information on the protein entries in the text file formats available through the UniProt website. Then we performed some data preprocessing tasks to transform the entries into a relation to make it a more understandable format.

6.5 The Experimental Process

In this section, we demonstrate the applicability of the data refresh rule mechanism that we presented in this chapter. To assess the potential benefits of this against UniProt entries we selected, we compared the effort required when rules are reapplied automatically against later versions with the effort required when the refresh rules must be repeatedly applied manually. We hypothesise that the data preparation task for the Editor when using the data refresh approach will require less human effort compared to the Editor working alone since some of the manual work of the Editor is now being
Figure 6.6: Example of database cross reference (DR) values being added

Figure 6.7: Example of database cross reference (DR) values being deleted

done by the refresh rule system.
6.5. THE EXPERIMENTAL PROCESS

We wrote a Java program to simulate the manual correction process of a data scientist working with the UniProt entries we selected. The tool for applying the data refresh rules was also implemented using Java. However, due to time and resource constraints in imitating the steps of a real data scientist, our simulated data scientist is set to verify and update protein entries in a simpler manner. In practice, curating protein entries specifically for UniProtKB and other established curated databases requires substantial effort. A well-defined curation process is essential in ensuring the quality of data entries is at the highest level and is done in a systematic manner. For example, to update a specific entry in UniProtKB, one must undergo automated and manual quality assurance checks [Unib]. For automated checks, the entries will be verified over many biological rules and corrected if needed. Then, under manual checks, an experienced curator is engaged to review the entries manually and verify that citations of all relevant literature and correct annotation are added, relevant sequences are merged, and every sequence analysis result is incorporated. The entry will only be integrated into the database once it passes the automated and manual quality control checks [Unic]. However, we opted for a simpler process for our simulated data scientist due to time and resource constraints. Below we describe how we set up our simulated data scientist and data refresh rules to work:
• **Protein Entries Dataset**: To simulate constantly changing data, we use the actual version history of the entries we used as the basis for the study. We collected five consecutive versions from two different protein entries ‘Q9NPU4’ and ‘B0UUJ3’. Both simulated data scientist and data refresh rules use the first version of data, to begin with, and the most recent version of the protein entries are used as the final version, a version expected by both approaches as a current and updated version, which is also the ground truth for our work. It is important to note that both approaches require a simulated Editor, but that the actions of this Editor are informed by the changes made to UniProt records.

• **Simulated Editor Approach**: For the simulated Editor approach, we attempt to impersonate the manual work performed by a data scientist in handling updates in protein entries, such as checking, analysing and verifying changes required from one version of protein entry to its subsequent version. Thus in the simulated editor approach, we compare one version of protein entry with the one before it. Differences extracted between two subsequent protein entry versions are considered as changes and are recorded in a file named change file that is versioned. Thus, one protein entry with several versions of updates will have multiple versions of its change file. Change files record differences between two subsequent entries, and we consider the differences between subsequent protein entries as manual corrections since changes (or updates) in a curated database usually go through manual checking. Over time, when more updates take place, the change file for the protein entry expands. In the case of our simulated data scientist, to update the current version of protein entry with all changes from the versions before the current entry, the simulated version will go through each change file that exists for that particular protein entry and confirm whether to apply the changes in the change files to the current protein entry or not and move on to the following versions of the change file. For example, five consecutive versions of protein entry in our experiment generated four versions of change files. Thus, to update the current protein entry with changes recorded before it, with the first version of the change file as a starting point, our program is set to update the current protein entry with changes available in each version of the change file consecutively. In this example, when confirmed, changes available in the four versions of change files will be applied to the current protein entry in

---

4https://www.uniprot.org/Uniprot/Q9NPU4
5https://www.uniprot.org/Uniprot/B0UUJ3
an orderly manner from one version of the change file to the other

- **Data Refresh Approach**: In this approach, using our data refresh rules, the program we set up processed each protein version, adding the changes found from one version of protein entry to the EditsKB as described earlier as manual corrections. Changes extracted in each conservative version are accumulated and stored in EditsKB files. In the experiment setup for data refresh rules, the manual work of the Editor is now reduced as the Editor just needs to confirm whether they want to apply the edits from EditsKB or otherwise.

### 6.6 Results

We assess the approach relative to only one aspect, which is the Editor’s time. We cannot compare the accuracy of the approaches, as both are based around a simulated data scientist. For pragmatic reasons, we assume both that the Editor knows what they are doing and that the changes done are based on the ground truth. This assumption precludes any assessment on the basis of accuracy. The time recorded is based on the elapsed duration of the experiment program run for both of the approaches. For clarification, it must be mentioned that the time recorded for the simulated editor is the time taken by the execution of the simulated editing steps and is not intended to reflect the time taken by a human editor. Data scientists usually conduct a thorough investigation in checking, analysing and verifying any updates before integrating them into the curated database such as UniProt thus the time taken is expected to be longer.

<table>
<thead>
<tr>
<th>Versions</th>
<th>Total Changes</th>
<th>Update</th>
<th>Delete</th>
<th>Insert</th>
<th>DataRefresh &amp; Editor</th>
<th>Editor Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0.56s</td>
<td>0.56s</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0.41s</td>
<td>0.51s</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.35s</td>
<td>0.42s</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0.52s</td>
<td>0.56s</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.41s</td>
<td>0.52s</td>
</tr>
<tr>
<td><strong>Total Time Taken</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.4: Estimation of Time Used with Updates to ‘B0UUJ3’**

<table>
<thead>
<tr>
<th>Versions</th>
<th>Total Changes</th>
<th>Update</th>
<th>Delete</th>
<th>Insert</th>
<th>DataRefresh &amp; Editor</th>
<th>Editor Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.60s</td>
<td>0.60s</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.42s</td>
<td>0.52s</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.35s</td>
<td>0.42s</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0.40s</td>
<td>0.50s</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.41s</td>
<td>0.50s</td>
</tr>
<tr>
<td><strong>Total Time Taken</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Despite its exploratory and small scale nature, the experiment offers some insight into how the data refresh approach might play out in practice. We were able to see only small time savings in this study, but then the numbers of changes were very small and we were looking at query results that returned only records of one protein at a time. However, our estimates for the manual change times were also very conservative. In practice, the data scientist would need to take much longer to decide what the correct data should be than they would to make the actual update. Such decisions may require a phone call with a colleague or finding and reading a portion of a research paper. These costs are unavoidable the first time the correction must be made, but some similar costs may be incurred for later refresh cycles, especially if the time difference between cycles is long. Therefore it is likely that we have under-estimated the costs for the repeated manual changes somewhat too.

One of the reason may be caused by the small portion of changes that we have in each cycle of the historical versions. If the changed list is more substantial and EditsKB stored most of these changes, then the manual correction effort performs by the Editor can be substantially reduced.

### 6.7 Threats To Validity & Conclusions

This chapter proposed a data refresh rule approach to capturing, recording and reapplying manual corrections to query results that must be regularly rerun over time. We reported on the results of an exploratory study to understand the potential benefits from this approach when applied on a real data set. This study compared a simulation of the data updates that would need to be made in two cases: 1) when the changes made by an editor are not captured and must be repeated when the query result is updated, and 2) when the changes are captured and can be reapplied without requiring input from the editor. We were able to see a small time saving from the data refresh approach in this study.

However, the small scale nature of this study means that we can draw only limited conclusions from it. The threats to validity that are present are listed below, along with some suggestions for how they could be overcome in future work:

- **Small size of the data set and updates:** because the study focused on only a small number of records and a small number of versions of those records, the results are only indicative of what benefits might be seen in one very particular
6.7. THREATS TO VALIDITY & CONCLUSIONS

In addition, we can only see the final version of the edits. This contrasts with the challenges faced by a real Editor who might have to make many more changes to make the query results fit for use. Because the versions of the UniProt records we see are only final edits (and don’t include edits that are later overwritten by others) we may be under-estimating the savings achievable from data refresh rules (since in real scenarios we can expect there to be more edits). Going forward, we should increase the data sample size (from UniProt KB) to be more confident of the results.

- **Study is limited to a single domain**: another limitation of the study is that we do not know how generalisable the results may be to other domains, even ones close to protein structure. Even within UniProt different records are subject to very different update patterns, with some entries being updated frequently and others (especially older or more obscure ones) being less frequently changed. The most we can claim is that we found benefits in even this small scale study, without having to look very far or for a data set with particular characteristics.

- **Assumptions regarding manual edits**: In our experiment, we take manual edits by comparing previous and current version of records and re-applied them to subsequent versions of the records. In simulating this process, we assumed that the same manual edits would be applied to all versions of the data set. In practice, this is not likely to be the case. This means that the control case we designed for our experiment could be different from what would be experienced in a natural setting. Ideally, the experiment should incorporate a human editor (data scientist) to perform a routine of correcting and editing datasets, although arranging for this to happen in a way that both mirrors the real life situation of data scientists and gives us ways to control all the experimental variables is likely to be challenging. In our case, even with a human editor, manually editing real datasets may still result in some artificiality of results because learning the domain knowledge needed to carry out the task well may require a longitudinal study which is not feasible in the scope of this study.

Due to these limitations, a firm conclusion on the benefits and costs of the data refresh approach can only be made by repeating the experiment on more extensive dataset entries, covering more domains and more refresh cycles, and with a more realistic editor simulation. Another possible option that may require long-term investigation is to have a scheduling plan where edits are extracted gradually whenever updates...
are made to the Uniprot database for a more accurate estimation of the total time cost-saving since new UniProt releases do not occur often (every four weeks).
Chapter 7

Conclusion

“When you reach the end of what you should know, you will be at the beginning of what you should sense”.  

Khalil Gibran

Existing work in dataspace integration has explored different types of feedback, focusing almost entirely on explicit approaches to feedback gathering. Because feedback can be expensive to obtain, and because the amount of feedback required to achieve meaningful integration improvement may be high, there is a need to establish an alternative feedback collection approach that is more cost-effective. One potential solution is to gather feedback implicitly: that is, by inferring the needed feedback from actions that the dataspace user chooses to do anyway, for their own purposes.

In this thesis, we set out to determine whether manual corrections, performed by the data scientist on the results of queries performed over a data integration, can become a source of implicit feedback for improving dataspace integration.

As we described in Section 2.4, because dataspace integration is rarely perfect, data scientists need to perform manual corrections to the results provided by the dataspace to obtain a data set that fulfils their needs and expectations. If the dataspace can observe and learn from these manual corrections, useful information could be extracted implicitly, as a form of feedback, and applied to improve the integration provided by the dataspace.

To understand the potential of manual corrections as a source of implicit feedback in dataspaces, we set out to answer the following questions that we previously stated in Section 1.4, page 23

- Can manual corrections be used to infer feedback that is useful in improving the
quality of an integration?

- Can feedback from manual correction be applied in different integration problem scenarios?

- What is the most appropriate way to use manual correction in a dataspace integration setting?

The answer to these questions might be different for different kinds of integration improvement. For our research, we needed to check our premise in several different integration improvement contexts. To help us guide our research, we developed a framework showing the broad categories of improvement challenges that are encountered in dataspaces (see Section 3.1). Figure 3.2 on page 65, shows the four-quadrant framework we proposed. We organised our research by designing and carrying out one study for each of the three challenge categories that were amenable to improvement by feedback.

These were:

- **Systematic Schema-Level** integration problems, in which feedback improves some aspect of the schema handling in the dataspace,

- **Systematic Data-Level** integration problems, in which feedback improves some handling of data-level integration uniformly across the whole data source, and

- **Unsystematic Data-Level** integration problems, in which feedback improves the handling of specific data values or groups of data values in the integration.

We built a mechanism to apply the manual correction approach for each of these different types of integration problem. In particular, this thesis focused on integration problems related to improving schema mapping results through true/false positive feedback (Systematic Schema Level), learning new format transformations to reconcile data from multiple sources (Systematic Data Level) and learning refresh rules for constantly updating source data sets (Unsystematic Data Level). In the next section, we will list the contributions of the work in these areas.

### 7.1 Research Contributions

Our main contributions in this thesis are the results from each study showing evidence that useful feedback can be extracted from manual corrections for each type of integration improvement.

We proposed a framework for manual correction based on common integration problems in the dataspace setting. We identified three types of integration problem that dataspaces face based on the level and extent of the problems. Our proposed framework is divided into three quadrants: Systematic Schema-Level, Systematic Data-Level and Unsystematic Data-Level. The quadrants allow us to design the direction of our research on manual corrections as it gives us target areas in which to seek evidence for the usefulness of feedback inferred from manual correction.

2. **Co-Integration**: An algorithm for inferring true positive/false positive feedback from manual corrections to query results.

Under the Systematic Schema-Level of the framework, we showed that we could infer feedback about whether query results were true or false positives from manual corrections made by the end-user. True/false positive feedback has been shown to be useful by other researchers in automatically adjusting query mappings and other schema-level artefacts [BPE+10, MDVK03, TIP10b]. But previous work has only considered the explicit collection of this kind of feedback. Our research showed how that an example of this kind of feedback can be inferred at no extra cost to the user when users manually correct query results produced by the dataspace.

We demonstrated the feasibility of this approach in an experiment that compared explicit false/true positive information given by human participants with feedback inferred from manual corrections to the data. Our finding shows that inferred feedback from manual correction was just as good as the explicitly supplied feedback. In our experiment, two out of six participants from the manual correction group manage to identify all injected errors correctly, and one out of six users from the annotated group manage to identify all injected errors correctly.

3. **ManEd**: An algorithm for inferring examples for learning format transformation rules from manual corrections to query results.

For the Systematic Data-Level component of the general framework, we proposed and compared a suite of strategies for gathering feedback to allow the dataspace to learn format transformation rules. The feedback was inferred from
CHAPTER 7. CONCLUSION

manual corrections made by the user, and took the form of before-and-after example pairs, of the kind required by format transformation learning engines, such as Flash Fill [Gul11]. In an experimental study, we found that:

• Different filtering strategies for deriving example pairs from manual corrections produce different results. The augmented values filtering strategy shows the best outcome across all data sets of different type and size; the transformation rules derived from this strategy correctly transformed more than 70% of the format errors on the data set tested on Names and Phone Number data type. However, the result for the Phone Number data type is not as good as the Names and Dates data type since the transformation rules derived from this strategy correctly transformed values only at the proportion of how much manual correction was supplied.

• The proportion of representative cases of format transformation included in the inferred feedback set is important in giving a promising result. Our results suggest that 30% coverage of the total set of representative cases manages to correctly transformed data set of Names data type of more than 60%. From the experiment result, at 30% coverage of the ManEd representative case, the average transformation score is above 0.6. While some ManEd users might be okay with a transformation score of around 60%, others may expect a transformation score of 100%. Thus, ManEd needs to provide flexibility to its user in accepting and rejecting the transformation rules derived from manual corrections based on the percentage of representatives cases covered during the manual correction. For this reason, we proposed a way of approximating the set of representative cases in a data set to allow the proportion of representative cases to be calculated at run time and be used for deciding whether to apply the learnt transformations rules or not.

4. Data Refresh: An approach for re-applying manual corrections to continually changing data sets.

For the Unsystematic Data-Level component of the general framework, we proposed a technique for the extraction of data refresh rules from manual corrections. These rules describe the unsystematic changes needed to prepare the results of the integration for use. When the underlying data sources change, and the integration is run again, the same errors will reappear and need to be fixed again,
unless the dataspace can learn to avoid them. The data refresh rules learned by our approach prevent this. Our method includes a preliminary approach for detecting conflicting manual corrections when used over time.

We used a case study using real data and updates made by expert curators (from the Uniprot protein database) to give an idea of the potential cost savings of using data refresh rules in practice. The results showed a small time saving with our approach.

7.2 Limitations

Although we were able to find some evidence for the broad usefulness of the idea of extracting feedback from manual corrections to integrated data, as with any research study, there are some limitations to our work that need to be reflected over, and we provide ways to address these limitations for future research:

- **Lack of Consideration of the Reasons Behind Changes to Data**
  Since we take as input only the raw data changes made by the data scientist, we have no way of knowing the reason behind each change (or lack of change). Previous work on implicit feedback shows that interpreting it relies heavily on the specific domain and may be speculative, so that we cannot be always sure if we have interpreted the feedback correctly [JLZ18, JGP+05]. In our work, we consider edits made to query results as corrections or true values expected by the editor, and data that is observed by the editor but left unchanged to be correct. There might be a case, however, where the editor does not make changes to some values not because the data is correct but because they have no information that would cause them to doubt its correctness. In addressing the correctness of unchanged data, we may need to ask the user to confirm whether unchanged data is correct or whether that is unknown. Asking users for validation is outside the scope of this thesis, as our work focuses on implicit feedback rather than explicit.

- **Manual corrections assumed to be from a single data scientist**
  Currently, our work only considers manual correction from one data scientist. In Chapter 4, although the experiment involves several participants performing edits on a data set, we did not explicitly consider discrepancies between editors since this was not the goal of the experiment. In Chapters 5 and 6, the experiments were designed to replicate the work of a single data scientist in manually
editing datasets for a specific purpose. Therefore, although the experiment results improve our understanding of manual correction, the outcome could not generalize to the large population of data scientists that may have different goals while handling and editing a dataset.

- **Scope of manual corrections limited to single columns**
  At present, our work analyses manual corrections on a column by column basis. While edits may be made to several columns in a tuple, when we analyse them, we extract knowledge from edits in a single column, relating only to that column. For example, in Chapter 4, we presented an approach that evaluates the performance of manual correction over an annotation approach. And although we require the editor to evaluate a full data set, where each tuple may contain several columns yet, we measure the correctness of edits based on the value given by the editor for the respective column, hence a single column. Whereas, in Chapter 5 we carried out an experiment that evaluates the accuracy of edits made on data sets consist of names. However, in practice, data sets are usually made up of multiple columns. Thus our approach should be extended to consider manual corrections to multiple columns at a time to conduct a more thorough analysis of the result.

### 7.3 Future Work

Below we list some future research directions that lead on from this work.

- **A hybrid approach to manual corrections.**
  Identifying the reasons for changed and unchanged data is challenging, relying on possibly unfounded assumptions about the user’s actions and intentions. Further studies on this matter need to be undertaken, so that the reason why data values get change and/or unchanged resulting from the editor’s edits become more clearly understood especially when the change is unsystematic or when the changes cause conflicts. As future work, we could explore the possibility of combining implicit manual correction as feedback with explicit feedback. For example, we could ask data scientists to explicitly annotate the reason behind the manual correction they perform. Deleting a tuple could be because the data is incorrect, outdated or not needed for the analysis purposes, and this help dataspace to improve. The hybrid approach of inferring feedback implicitly from manual
correction and gathering feedback explicitly for examples through annotations could offer a better argument for why such changes happen and the intention behind them, especially on unsystematic or conflicting changes.

- **Manual correction on multiple columns and multiple tables**
  To further improve the generalization of our proposed approach, longitudinal studies that include a larger sample size with edits across multiple tables and columns performed by a group of data scientists would be interesting to be investigated for a deeper analysis of the experiment results presented in this thesis.

- **Deriving feedback from manual correction on crowd source worker**
  This thesis explores the potential of inferring user feedback from a manual correction in a data integration setting. Specifically in query mapping, data format cleaning and data refresh. However, in the existing literature, we can find other data management tasks that could use feedback derived from manual correction. One of the potential areas is crowdsourcing. Existing work on crowdsourcing has investigated timely, task-specific feedback that assists crowds to produce better results [DKKH12]. Based on the task-specific feedback introduced in work by Dow et al., it may be necessary to have a mechanism that converts the manual work/corrections performed by multiple crowd members into values of task-specific feedback.

### 7.4 Overall Conclusion

In this thesis, we asked the following question:

> Can manual corrections performed on the query results produced by dataspace integration be used as a source of implicit feedback to improve the quality of the integration?

Based on the results we obtained from our study of this approach in several integration contexts, the broad answer is: yes, but with some reservations. We were able propose approaches for deriving feedback value from manual corrections in a range of data integration settings without needing to search for niches with particular features. But the usefulness in practice depends on whether or not the feedback values that are inferred from the manual correction are worth the time required in setting up such approach.
We were only able to show value in an experimental setting in this thesis, and therefore this question remains open.

Complicated and time-consuming data processes cause their owners to adopt automation tools and systems. While such automation can be cost-effective and offer predictable and consistent outcomes, it can be inflexible and unable to cope with special cases, something that human experts and their judgement could provide. The question is, do we focus on improving the quality of outcome of automation tools and systems? Or do we make the most out of human involvement despite it being costly? A likely answer could be in the middle, where we take advantage of the cost-effective automation tools and systems and make the most of human involvement by continuously observing and learning from the human expert. This is what the approach proposed in this thesis tries to do: to allow humans and programs to each contribute to the process where they are best able to.

Therefore, from a broader perspective, observing and learning from a human expert through their actions, such as manual corrections and subsequently re-applying the learned knowledge, is deemed beneficial. Firstly, the dataspace could extract and learn expertise at a reduced cost, and secondly, we could re-apply the learned knowledge to reduce the repetitive manual work often performed by a human experts. We also believe that the know-how learned from manual edits does not restrict only to data integration-related tasks but also in other areas such as in education, manufacturing or any other discipline so long as the knowledge of a human expert is needed. For example, assessment evaluation and grading, and personalized feedback from a teacher customarily carried out manually have now started to become automated, aimed to reduce teachers’ tasks. Especially in classes with significant sizes so teachers could dedicate more time in supporting student learning experience [YM19, YM20, ZH18]. We speculate that if the teacher’s expertise in marking assessment and providing feedback to students could be observed and learned, that knowledge could be extracted and applied to further improve the automated tools to complement the training data often used to train such automated systems - especially for automated assessment such as multiple choice exams. Inclusion of the teacher’s knowledge and style of feedback upon automated evaluation system could offer more comprehensive feedback to learners and at the same time increase the student’s confidence and trust in adopting such systems [ZM21].

The manufacturing industry is another possible area that could benefit from our
proposed work. Currently, the introduction of Industry 4.0 has promoted smart manufacturing where a great deal of manual operation and production processes are now digitized, automated and make smarter using machine learning and artificial intelligence to support production, operation and decision-making processes [BDM+19, Gho20]. Despite that, technologies for intelligent manufacturing still require considerable investigation, especially in ensuring trusted and secure, intelligent manufacturing systems that includes humans-in-the-loop so the desired outcome could be delivered through human-centred intelligent applications [CPPC20, RZK+21, PLTRM17]. Recent studies [RZK+21, EGGS20, LNP17] have highlighted the significance of user feedback in industrial processes to improve the prediction and decision-making recommendation of the intelligent manufacturing system, and it seems that our proposed approach could be one of the ways forward.
Bibliography


[PKP19] Minh Pham, Craig A. Knoblock, and Jay Pujara. Learning data transformations with minimal user effort. In Chaitanya Baru, Jun Huan, Latifur Khan, Xiaohua Hu, Ronay Ak, Yuanyuan Tian, Roger S. Barga, Carlo


[Unib] UniProtKB.

[Unic] UniProtKB.


[VMZ+13] Carlos Roberto Valêncio, Matheus Henrique Marioto, Geraldo Francisco Donega Zafalon, José Márcio Machado, and Julio César Momente.
Real time delta extraction based on triggers to support data warehousing.


Appendix A

A.1 Inferring False Positive/Negative Feedback from Manual Corrections

Table A.1 shows the collection of data gathered from the experiment describe in Section 4.3 for manual correction group. While, Table A.2 depicted the results gathered from the same experiment but from the annotated group.

Table A.1: Result from manual approach group over ground truth.

<table>
<thead>
<tr>
<th>ManualCorrection</th>
<th>True Positive(TP)</th>
<th>False Positive(FP)</th>
<th>False Negative(FN)</th>
<th>Duplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User 2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User 3</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User 4</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>User 5</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User 6</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Ground Truth</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table A.2: Result from annotated approach group over ground truth.

<table>
<thead>
<tr>
<th>Annotated</th>
<th>True Positive(TP)</th>
<th>False Positive(FP)</th>
<th>False Negative(FN)</th>
<th>Duplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>User 2</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>User 3</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>User 4</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>User 5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User 6</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ground Truth</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Appendix B

B.1 Format Transformation Filtering Strategies

Three sizes of data sets with 50, 100 and 200 rows respectively have been randomly selected from “Schools in England” \(^1\) data set to be use as our seed data sets to evaluate the performance of the three proposed strategies, *Null Filtering*, *Edited Filtering* and *Augmented Edit Filtering*. Three sizes of data sets cover three category of data types, Names, Dates and Phone Number. In this section, we depicted the values across five run for each of the 6 different percentage that are increment by 10% across filtering strategies and data sizes. Figure B.1 illustrated average, min and max across the five runs for Names data type, Figure B.2 demonstrated for Dates datatype and Figure B.3.

\(^1\)https://data.gov.uk/dataset/schools-in-england
B.1. FORMAT TRANSFORMATION FILTERING STRATEGIES

Figure B.1: Transformation score average with min and max values for Names data type on 50, 100 and 200 rows on all three strategies (null, edited and augmented filtering)
Figure B.2: Transformation score average with min and max values for Dates data type on 50, 100 and 200 rows on all three strategies (null, edited and augmented filtering)
Figure B.3: Transformation score average with min and max values for Phone Numbers data type on 50, 100 and 200 rows on all three strategies (null, edited and augmented filtering)
B.2 Evaluation on Representative Case Coverage

Figure B.4 shows the first set of collection results of five runs (randomly) used for the experiment in Section 5.5.2 across data size 50, 100 and 200 rows respectively for Names data type. The averaged results are demonstrated in Section 5.5.3. Figure 5.6 shows the five runs of the first set across three data sizes of 50, 100 and 200 rows respectively. While Figure 5.7 and 5.8 illustrated for Set 2 and 3 respectively.
### B.2. Evaluation on Representative Case Coverage

<table>
<thead>
<tr>
<th>Num of Example per run</th>
<th>50 rows (DS1)</th>
<th>ManEd Identified Representative Case</th>
<th>User Identified Case Coverage</th>
<th>% of ManEd Representative Case Covered</th>
<th>Transformation Score (Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Example</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Run1</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Run2</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Run3</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Run4</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Run5</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>15</td>
<td>4</td>
<td>26.71</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>2 Example</td>
<td>16</td>
<td>7</td>
<td>43.75</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Run1</td>
<td>16</td>
<td>7</td>
<td>43.75</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Run2</td>
<td>16</td>
<td>7</td>
<td>43.75</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Run3</td>
<td>16</td>
<td>7</td>
<td>43.75</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Run4</td>
<td>16</td>
<td>7</td>
<td>43.75</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Run5</td>
<td>16</td>
<td>7</td>
<td>43.75</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>16</td>
<td>7</td>
<td>43.75</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Num of Example per run</td>
<td>100 rows (DS1)</td>
<td>ManEd Identified Representative Case</td>
<td>User Identified Case Coverage</td>
<td>% of ManEd Representative Case Covered</td>
<td>Transformation Score (Average)</td>
</tr>
<tr>
<td>1 Example</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>Run1</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>Run2</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>Run3</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>Run4</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>Run5</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.59</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.59</td>
<td></td>
</tr>
</tbody>
</table>

**Figure B.4:** Results of 5 runs for rules validation based on representative case for 50, 100 and 200 rows (Set 1)
<table>
<thead>
<tr>
<th>Num of Example per run</th>
<th>50 rows (DS2)</th>
<th>Manifold Identified Case</th>
<th>User Identified Case Coverage</th>
<th>% of Manifold Representative Case Covered</th>
<th>Transformation Score (Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Example Run1</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.46</td>
<td></td>
</tr>
<tr>
<td>Run2</td>
<td>14</td>
<td>4</td>
<td>28.57</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>Run3</td>
<td>15</td>
<td>3</td>
<td>20.00</td>
<td>0.66</td>
<td></td>
</tr>
<tr>
<td>Run4</td>
<td>14</td>
<td>4</td>
<td>28.57</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>Run5</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.56</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>13</td>
<td>5</td>
<td>38.46</td>
<td>0.76</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Num of Example per run</th>
<th>100 rows (DS2)</th>
<th>Manifold Identified Case</th>
<th>User Identified Case Coverage</th>
<th>% of Manifold Representative Case Covered</th>
<th>Transformation Score (Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Example Run1</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.27</td>
<td></td>
</tr>
<tr>
<td>Run2</td>
<td>17</td>
<td>4</td>
<td>23.53</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Run3</td>
<td>15</td>
<td>4</td>
<td>26.67</td>
<td>0.69</td>
<td></td>
</tr>
<tr>
<td>Run4</td>
<td>17</td>
<td>4</td>
<td>23.53</td>
<td>0.46</td>
<td></td>
</tr>
<tr>
<td>Run5</td>
<td>16</td>
<td>4</td>
<td>25.90</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>16</td>
<td>5</td>
<td>31.25</td>
<td>0.88</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Num of Example per run</th>
<th>200 rows (DS2)</th>
<th>Manifold Identified Case</th>
<th>User Identified Case Coverage</th>
<th>% of Manifold Representative Case Covered</th>
<th>Transformation Score (Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Example Run1</td>
<td>20</td>
<td>4</td>
<td>20.00</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>Run2</td>
<td>20</td>
<td>4</td>
<td>20.00</td>
<td>0.65</td>
<td></td>
</tr>
<tr>
<td>Run3</td>
<td>19</td>
<td>4</td>
<td>21.05</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>Run4</td>
<td>19</td>
<td>4</td>
<td>21.05</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>Run5</td>
<td>20</td>
<td>4</td>
<td>20.00</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td>20</td>
<td>7</td>
<td>35.00</td>
<td>0.87</td>
<td></td>
</tr>
</tbody>
</table>

Result from five runs for user identified case for 50 rows

Result from five runs for user identified case for 100 rows

Result from five runs for user identified case for 200 rows

Figure B.5: Results of 5 runs for rules validation based on representative case for 50, 100 and 200 rows (Set 2)
### B.2 Evaluation on Representative Case Coverage

<table>
<thead>
<tr>
<th>Num of Example per run</th>
<th>50 rows (DS5)</th>
<th>ManEnt Identified Representative Case</th>
<th>User Identified Case Coverage</th>
<th>% of ManEnt Identified Case Covered</th>
<th>Transformation Score (Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Example</td>
<td>Run1</td>
<td>4</td>
<td>30.77</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>4</td>
<td>28.57</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>4</td>
<td>28.57</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>4</td>
<td>28.57</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>4</td>
<td>30.77</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>29.45</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>2 Example</td>
<td>Run1</td>
<td>4</td>
<td>35.71</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>7</td>
<td>58.33</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>6</td>
<td>42.86</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>6</td>
<td>42.86</td>
<td>0.72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>6</td>
<td>40.00</td>
<td>0.72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>43.95</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>3 Example</td>
<td>Run1</td>
<td>4</td>
<td>50.00</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>5</td>
<td>33.33</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>8</td>
<td>53.33</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>9</td>
<td>69.23</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>7</td>
<td>53.85</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>51.95</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>4 Example</td>
<td>Run1</td>
<td>9</td>
<td>60.00</td>
<td>0.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>10</td>
<td>71.43</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>7</td>
<td>53.85</td>
<td>0.86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>8</td>
<td>66.67</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>9</td>
<td>69.23</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>64.23</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>5 Example</td>
<td>Run1</td>
<td>10</td>
<td>70.92</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>10</td>
<td>70.92</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>9</td>
<td>64.23</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>9</td>
<td>64.23</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>7</td>
<td>58.33</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>68.15</td>
<td>0.87</td>
<td></td>
</tr>
</tbody>
</table>

Result from five runs for user identified case for 50 rows

<table>
<thead>
<tr>
<th>Num of Example per run</th>
<th>100 rows (DS5)</th>
<th>ManEnt Identified Representative Case</th>
<th>User Identified Case Coverage</th>
<th>% of ManEnt Identified Case Covered</th>
<th>Transformation Score (Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Example</td>
<td>Run1</td>
<td>18</td>
<td>22.22</td>
<td>0.93</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>18</td>
<td>22.22</td>
<td>0.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>17</td>
<td>23.53</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>19</td>
<td>21.05</td>
<td>0.62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>19</td>
<td>21.05</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>22.02</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>2 Example</td>
<td>Run1</td>
<td>17</td>
<td>29.41</td>
<td>0.72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>17</td>
<td>35.29</td>
<td>0.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>18</td>
<td>38.89</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>18</td>
<td>27.38</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>19</td>
<td>26.32</td>
<td>0.73</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>33.54</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>3 Example</td>
<td>Run1</td>
<td>18</td>
<td>38.89</td>
<td>0.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>17</td>
<td>47.06</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>18</td>
<td>50.00</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>17</td>
<td>41.38</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>19</td>
<td>38.44</td>
<td>0.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>42.79</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>4 Example</td>
<td>Run1</td>
<td>17</td>
<td>47.06</td>
<td>0.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>18</td>
<td>44.44</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>18</td>
<td>44.44</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>17</td>
<td>64.71</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>18</td>
<td>50.00</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>50.33</td>
<td>0.86</td>
<td></td>
</tr>
<tr>
<td>5 Example</td>
<td>Run1</td>
<td>18</td>
<td>50.00</td>
<td>0.86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>17</td>
<td>47.06</td>
<td>0.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>18</td>
<td>61.11</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>18</td>
<td>50.00</td>
<td>0.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>16</td>
<td>56.25</td>
<td>0.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>52.88</td>
<td>0.87</td>
<td></td>
</tr>
</tbody>
</table>

Result from five runs for user identified case for 100 rows

<table>
<thead>
<tr>
<th>Num of Example per run</th>
<th>200 rows (DS5)</th>
<th>ManEnt Identified Representative Case</th>
<th>User Identified Case Coverage</th>
<th>% of ManEnt Identified Case Covered</th>
<th>Transformation Score (Average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Example</td>
<td>Run1</td>
<td>26</td>
<td>15.38</td>
<td>0.605</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>26</td>
<td>15.38</td>
<td>0.735</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>27</td>
<td>14.81</td>
<td>0.835</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>26</td>
<td>15.38</td>
<td>0.42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>26</td>
<td>15.38</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>15.37</td>
<td>0.652</td>
<td></td>
</tr>
<tr>
<td>2 Example</td>
<td>Run1</td>
<td>25</td>
<td>24.00</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>25</td>
<td>24.00</td>
<td>0.925</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>25</td>
<td>28.00</td>
<td>0.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>26</td>
<td>23.08</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>26</td>
<td>26.92</td>
<td>0.73</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>25.20</td>
<td>0.861</td>
<td></td>
</tr>
<tr>
<td>3 Example</td>
<td>Run1</td>
<td>26</td>
<td>26.92</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>25</td>
<td>28.00</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>26</td>
<td>23.08</td>
<td>0.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>26</td>
<td>23.08</td>
<td>0.72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>26</td>
<td>34.62</td>
<td>0.855</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>27.14</td>
<td>0.854</td>
<td></td>
</tr>
<tr>
<td>4 Example</td>
<td>Run1</td>
<td>26</td>
<td>38.46</td>
<td>0.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>26</td>
<td>26.92</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>26</td>
<td>23.08</td>
<td>0.835</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>26</td>
<td>23.08</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>26</td>
<td>34.62</td>
<td>0.685</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>29.23</td>
<td>0.820</td>
<td></td>
</tr>
<tr>
<td>5 Example</td>
<td>Run1</td>
<td>26</td>
<td>35.71</td>
<td>0.775</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run2</td>
<td>26</td>
<td>38.46</td>
<td>0.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run3</td>
<td>26</td>
<td>38.46</td>
<td>0.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run4</td>
<td>26</td>
<td>30.77</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run5</td>
<td>27</td>
<td>29.63</td>
<td>0.69</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td>34.61</td>
<td>0.799</td>
<td></td>
</tr>
</tbody>
</table>

Result from five runs for user identified case for 200 rows

Figure B.6: Results of 5 runs for rules validation based on representative case for 50, 100 and 200 rows (Set 3)