Robust and Proactive Error Detection and Correction in Tables

by

Minh Tran Xuan Pham

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2022

Dedication

To my wife Oanh and my son Khiem for their encouragement, love, and patience.

To my Mom and my Dad, in loving memory.

Acknowledgements

First, I would like to thank my Ph.D. advisor, Craig A. Knoblock, for his valuable guidance and support throughout my Ph.D. What I learned the most from Craig is the ability to brainstorm research ideas in the big picture and align the research problems with real-world scenarios. Craig is also a very patient and understanding mentor. I struggled a lot during the second and third years of my Ph.D., and Craig has always encouraged me to keep trying and moving forward. He is also very understanding when I have personal issues and is very flexible with the schedule to allow me the time to take care of my family.

I also want to thank all members of my proposal and dissertation committees for providing valuable feedback on my dissertation: Professor Bistra Dilkina, Professor Xiang Ren, Professor Muhao Chen, and Professor Gerard Hoberg. I want to thank Muhao Chen for spending hours discussing research with me. Muhao's expertise in machine learning and natural language processing helps me tremendously in transitioning my research to focus more on machine learning approaches. I also want to thank Pedro Szekely and Jay Pujara for attending my weekly research meetings when they had time and giving valuable comments.

I want to thank my colleagues in the Center on Knowledge Graphs. Thanks to my officemate Basel Shbita, who is always ready to help. I would like to thank my USC Vietnamese friends, who made my Ph.D. life fun. Thank you, Binh Vu, for helping me with my research and spending time

hanging out with me. Thank you, Phong Trinh and Quynh Nguyen, for helping me in my first few years at USC.

I am thankful to my Mom and Dad, who were no longer with me but always supported me during my academic pursuits. Thank you, my brother, Khai, for always being here and playing games with me. Thank my aunts, uncles, and cousins for caring for my brother and me.

Finally, I want to thank my wife, Oanh, for her support and patience during my long 7-year journey, and I feel fortunate that I can spend my life with you. Thank my son Khiem for making the final three years of my Ph.D. joyful and full of smiles.

This research was supported in part by the Army Research Office and the Defense Advanced Research Projects Agency under contract number W911NF-18-1-0027, and was supported in part by the United States Air Force and the Defense Advanced Research Projects Agency under contract number FA8750-16-C-0045. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office, Defense Advanced Research Projects Agency, United States Air Force, or U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Table of Contents

List of Tables

List of Figures

Abstract

Millions of tables are available on the web, and they can provide valuable information for downstream applications in different domains. However, tables can contain detrimental errors, and cleaning these errors is an essential pre-processing phase before the tables can be used. Table errors can be classified into syntactic and semantic errors. Syntactic errors are erroneous values involved in data format and representation. These values do not follow the design rules of data sources and will yield fatal errors when input into database systems or applications. The most common syntactic errors are missing values, typos, format inconsistencies, and violated attribute dependencies. On the other hand, semantic errors are errors that are related to the semantic meaning of values in the tables and cannot be identified by traditional database approaches.

Traditional data cleaning requires extensive manual effort and is very time-consuming. Much recent research has focused on automating the data cleaning process and reducing users' workload. However, both supervised and unsupervised approaches to data cleaning face various difficulties. Supervised approaches require labeled data for training, and labeling tasks can be a significant burden for users to train their machine-learning models. On the other hand, unsupervised techniques introduce user bias into the process and usually only perform well for a limited set of error types. To address the weaknesses of existing methods, we present a robust and proactive approach that can perform comparably to supervised approaches while requiring minimal

human interaction. Our approach leverages closed-domain weak supervision and open-domain web text for automation. Closed-domain weak supervision includes different forms of user inputs, such as pre-defined detection functions, domain-specific languages or validation checking, while open-domain knowledge leverages available web text and structured knowledge sources in our training. Our semi-supervised approach targets error detection and correction for tables and can handle both syntactic and semantic errors.

In the evaluation, our syntactic error detection approach yields an improvement of 10%. Our syntactic error correction method performs comparably to state-of-the-art systems while requiring no parallel labeled input-output data. On the other hand, our semantic error detection achieves a performance of 0.86 in F1 score while the semantic error correction algorithm has a recall of 0.65.

Chapter 1

Introduction

Tables provide a massive set of informative data. However, tables can contain erroneous values, affecting the correctness of downstream applications. Therefore, data cleaning is essential in ensuring data quality in tables. Traditionally, data curators are in charge of manually performing data cleaning tasks. Such manual cleaning is done based on human knowledge and experience. Previous research defined an error as "a deviation from its given ground truth" [1]. In this thesis, we categorize these deviations into syntactic and semantic errors. Syntactic errors are erroneous values that violate the design rules or regulations of the data sources and will yield fatal errors when input into systems, databases, or applications. The most common syntactic errors in data sources are missing values, typos, format inconsistencies, and violated attribute dependencies [57]. On the other hand, semantic errors follow the design rule of the databases but contain incorrect factual information. Examples of syntactic and semantic errors are shown in Tables 1.1 and 1.2. In Table 1.1, "2.3 |" and "134 954 e" are syntactic errors because they have extra characters at the end of the strings and thus have different formats. In Table 1.2, the value "Ho Chi Minh City" follows the same format as the other capital cities, does not contain any typos and is a valid

Table 1.1: Table with syntactic errors

Beer name	style	ounces	abv
Pub Beer	American Pale Lager	12.0 oz	0.05
Rise of Phoenix	American IPA	12.0 ounce	0.07
Sinister	American Double	12.0 oz	0.09%

Table 1.2: Table with semantic errors

Country	Capital
United States	Washington D.C
Japan	Tokyo
Vietnam	Ho Chi Minh City
Germany	Berlin

city. However, "Ho Chi Minh City" is not actually the capital of "Vietnam," and the value needs to be corrected.

Figure 1.1 shows the workflow of data cleaning. In practice, error detection and error correction are an iterative process where a certain number of errors are identified and curated. The process then continues with other errors until no error can be found. There usually is human interaction to verify the results between the error detection and error correction phases.

There have been many studies on detecting syntactic errors [18, 32, 37, 46, 57, 59, 62, 90]. Supervised methods [32, 46, 57, 62] have focused on leveraging machine-learning techniques to characterize the properties of labeled errors and apply learned models to detect unseen errors. However, learning-based error detection systems require significant training data to achieve satisfactory performance, especially deep-learning methods [32]. Labeling errors in a large and noisy dataset is time-consuming and can burden users with a significant task required to train their models. On the other hand, unsupervised approaches [18, 37, 59, 90], which do not need labeled data, rely heavily on a fixed inductive bias for detection. However, erroneous values in

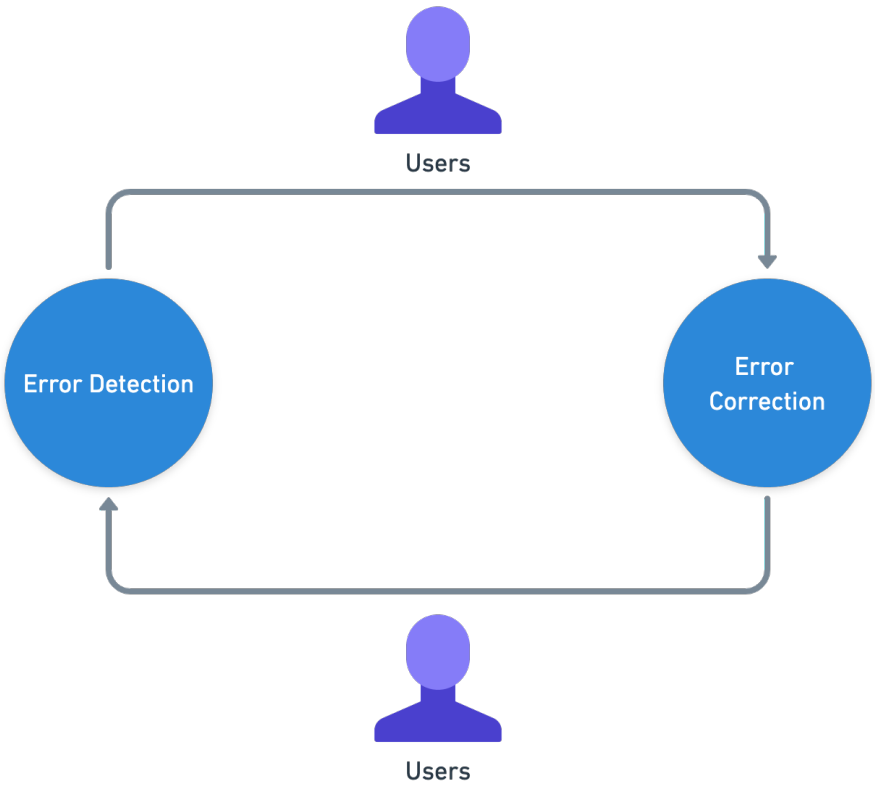


Figure 1.1: Data cleaning iterative process

one database may be normal in another. Therefore, existing methods that work intrinsically in one domain often fail to adapt extrinsically to other domains [57].

Besides syntactic error *detection*, syntactic error *correction* is also a challenge in data cleaning. Erroneous data can be represented in many formats. For instance, date/time data can be stored as "dd/mm/yyyy," "mm/dd/yyyy," and "MM, dd, yy"; or people's names can be written in many ways, such as "$\langle first\_name \rangle$ $\langle last\_name \rangle$" or "$\langle last\_name \rangle$, $\langle first\_name \rangle$." State-of-the-art methods in syntactic error correction, such as programming-by-example (PBE) or interactive data cleaning, still depend mainly on human interaction. For instance, interactive cleaning systems [46, 73] require users to specify transformation rules based on their suggestions. On the other hand, PBE [28, 80, 92], the state-of-the-art approach, achieves high transformation accuracy by relying

Table 1.3: Correcting erroneous data from different formats

Erroneous data	Transformed data
Messi; Lionel	Lionel Messi
Paul Pogba	Paul Pogba
Sergio R. Garcia	Sergio Garcia

Table 1.4: Table in Wikipedia about Iain Glen's awards

Year	Award	Category	Work	Result
1990	Silver Bear	Best Actor	Silent Scream	Won

on aligned input/output examples provided by users. Because of the dependency on human input, interactive data cleaning and PBE are rarely scalable when the volume of data and the number of data sources increase. Moreover, these existing approaches do not provide an easy way for users to verify the transformed results. Checking each transformed result becomes tedious with thousands or millions of records.

In contrast to syntactic errors, semantic errors attract little research. One major challenge in detecting semantic errors is its requirement for external knowledge sources to validate the data. For example, in Table 1.2, general geography knowledge is necessary to recognize the capitals of different countries and then identify the error. Structured knowledge sources, such as Wikidata, have been used to extract essential information for detecting semantic errors [16]. However, structured knowledge requires extensive manual effort to build and thus usually contains limited information compared to unstructured data. For example, Table 1.4 shows a Wikipedia table about "Iain Glen." By referring to the "Iain Glen" page in Wikidata, the current most extensive knowledge graph, we cannot verify the information specified in the tables since there are not enough details on the Wikidata page.

In this work, we aim to resolve the aforementioned issues. We present multiple approaches to the problem of data cleaning for both syntactic and semantic errors. The main goal of our

Table 1.5: Belgium statistical profiles dataset

GDP per capita	Voluntary expenditure	Household income	Passenger transport
41 450	2.3	-0.5	138 643
43 746	2.3	1.1	132 125
44 720	2.3	0.4	134 954 e

research is to reduce human effort in data cleaning while maintaining excellent performance so the approaches can be used in practical real-world applications.

1.1 Motivating Example

We provide three motivating examples to show the challenges of syntactic and semantic data cleaning.

1.1.1 Syntactic Error Detection

First, we provide a motivating example to explain how active learning can be used to detect syntactic errors. In this example, we use a snippet of a real-world dataset about Belgium statistical profiles from the Organisation for Economic Co-operation and Development (OECD) library,* as shown in Table 1.5.

Scenario 1. Suppose we want to detect errors in the *GDP per capita* column. It is easy to see that all *GDP per capita* column values have the same format, and their values seem comparable. Therefore, using only internal information, users may conclude that there is no error in the column. However, with external information, curators may notice that using a pattern like "XXX XXX" (where X represents a digit) to represent numbers is uncommon since popular formats for

*https://www.oecd-ilibrary.org/economic

numbers are “XXX.XXX” or “XXX,XXX” formats. In practice, these “XXX XXX” numbers should be flagged as errors for curation before being used in downstream tasks. In this case, we can learn from our Web table corpora that the pattern “XXX XXX” rarely appears and can conclude that values with “XXX XXX” pattern are more likely to be errors.

Scenario 2. We can look at the column *Voluntary expenditure* where cell values that end with “|” are rare cases in the column and are more likely to be errors. In this scenario, we can use an internal signal function that computes the format pattern frequency to detect the “2.3 |" since it is a rare pattern.

Scenario 3. There are situations when the initial potential errors are incorrect. For example, in the column *Household disposable income*, the value “-0.5” is a possible error based on internal signals since it is the only negative value. However, based on user verification, “-0.5” is a normal value. To allow our active learning model to handle these cases, we develop a probabilistic graphical model to model the agreement between our signals and user feedback.

1.1.2 Syntactic Error Correction

Here, we provide a real-world scenario where existing syntactic error correction systems have difficulties. Then we show how the transformations can be learned.

Scenario. The NYC Open Data portal[†] contains multiple data sources about places of interest. Tables 1.6 and 1.7 show two data sources concerning `Hotel` and `Restaurant` found in NYC Open Data. However, there are inconsistencies in the data that may become problematic when merging

[†]https://data.cityofnewyork.us

Table 1.6: Data source concerning NYC hotels

Name	Phone	Website	Location
Paramount Hotel	(212) 764-5500	http://www.nycparamount.com	(40.759132, -73.986348)
Doubletree Guest Suites	2127191600	www.nycdoubletreehotels.com	(40.759055, -73.98471)
The Westin New York at Times Square	(212) 868-1900 ext 245	www.westinny.com	(40.757482, -73.988309)

Table 1.7: Data source concerning NYC restaurants

Name	Address	Phone	Website	Latitude	Longitude
Ranch 1	832 Eighth Ave	2129561111		40.762444	-73.985983
Sosa Borella	832 Eighth Ave	(212) 262-8282	http://www.sosaborella.com/	40.762444	-73.985983
Starbucks	871-879 Eighth Ave,# 871	2122467699	www.starbucks.com	40.763644	-73.985134

records into a homogeneous database. For example, the phone numbers and the website URLs in both `Hotel` and `Restaurant` need to be normalized. In addition, the location column in the `Hotel` data source needs to be split into longitude and latitude for consistency with the `Restaurant` data.

Manual or programming approaches may be viable solutions if the number of sources is small. However, as the number of data sources grows, these approaches become intractable. PBE and interactive data cleaning systems allow semi-supervised operations to reduce user effort. However, a system that requires little or no human interaction is ideal.

1.1.3 Semantic Error Detection and Correction

We provide a motivating example for semantic error detection using two erroneous Wikipedia tables and show how textual evidence in Wikipedia articles can be exploited to fix these errors.

Club	Location	Stadium
Al-Ahli	Jeddah	King Abdullah Sports City
Al-Faisaly	Harmah	King Salman Sport City Stadium
Al-Fateh	Al-Hasa	Prince Abdullah bin Jalawi Stadium
Al-Hilal	Riyadh	King Fahd International Stadium Prince Faisal bin Fahd Stadium
Al-Ittihad	Jeddah	King Abdullah Sports City
Al-Khaleej	Saihat	Prince Saud bin Jalawi Stadium
Al Nassr	Riyadh	King Fahd International Stadium Prince Faisal bin Fahd Stadium
Al-Qadisiyah	Khobar	Prince Saud bin Jalawi Stadium
Al-Raed	Buraidah	King Abdullah Sport City Stadium
Al-Shabab	Riyadh	King Fahd International Stadium Prince Faisal bin Fahd Stadium
Al-Taawoun	Buraidah	King Abdullah Sport City Stadium
Al-Wehda	Makkah	King Abdul Aziz Stadium
Hajer	Al-Hasa	Prince Abdullah bin Jalawi Stadium
Najran	Najran	Al Akhdoud Club Stadium

Figure 1.2: A table in "Football league system in Saudi Arabia"

Example 1 Figure 1.2 shows a table of soccer teams extracted from the Wikipedia page for "Football league system in Saudi Arabia.[‡]". The table contains information about the soccer teams, locations, and home stadiums. However, suppose we cross-check the information with the Wikipedia pages of each soccer team. In that case, we can see that "Prince Saud bin Jalawi Stadium" is the home stadium of "Al-Qadisiya" instead of "Al-Khaleej" as stated in the table. It is quoted from the Wikipedia article of "Prince Saud bin Jalawi Stadium[§]" as "The venue is currently used mostly for football matches and it is the home stadium of Al-Qadisiya."

[‡]https://en.wikipedia.org/wiki/Football_league_system_in_Saudi_Arabia
[§]https://en.wikipedia.org/wiki/Prince_Saud_bin_Jalawi_Stadium

1 October	*The Increasingly Poor Decisions of Todd Margaret*	Channel 4
7 October	*PhoneShop*	
1 November	*Coppers*	
8 November	*Celebrity Coach Trip*	
30 November	*Frankie Boyle's Tramadol Nights*	

Figure 1.3: A table in "2010 in British television"

Example 2 Figure 1.3 shows a portion of another Wikipedia table about British television debuts in 2010 extracted from the Wikipedia page "2010 in British television.[¶]" However, as stated in the Wikipedia page of "PhoneShop,[‖]" "PhoneShop" was actually first broadcast on "Channel 4" on "13 November 2009".

The two above examples show that there are numerous errors in Wikipedia tables. However, these errors are difficult to identify since detecting these errors requires finding and understanding contradicting textual evidence. In Example 1, the table contents alone are insufficient to identify the error. In Example 2, the wrong value, "7 October 2010," was actually the broadcast day of the entire series after the pilot was broadcast on "13 November 2009."

1.2 Problem Definition

We formulate the problem of data cleaning as follows:

Definition 1.1 (Data Cleaning Problem) *Given a raw table S and a ground truth table S' with n rows and m columns, where S_{ij} and S'_{ij} refers to the cell in row i and column j of S and S',*

[¶]https://en.wikipedia.org/wiki/2010_in_British_television#Channel_4
[‖]https://en.wikipedia.org/wiki/PhoneShop

Table 1.8: An example of data errors in tables

(a) Raw data source with errors

GDP per capita	Voluntary expenditure	Household income	Passenger transport
41 450	2.3	-0.5	138 643
43 746	2.3 |	1.1	132 125
44 720	2.3	0.4	134 954 e

(b) Ground truth data source

GDP per capita	Voluntary expenditure	Household income	Passenger transport
41 450	2.3	-0.5	138 643
43 746	2.3	1.1	132 125
44 720	2.3	0.4	134 954

- *Error detection: predict if* $S_{ij} = S'_{ij}$ *where True means* S_{ij} *is a normal value and False means* S_{ij} *is an erroneous value.*

- *Error correction: if* $S_{ij} \neq S'_{ij}$*, correct* S_{ij}*.*

Tables 1.8a and 1.8b show an example of the data cleaning problem, where cell values in red indicate erroneous values. These errors can be identified by comparing them with the ground truth data source. In practice, the ground truth tables sometimes do not exist. However, they can be artificially created by following database design rules or user curation.

1.3 Proposed Approach

Our work on data cleaning for tables can be divided into three main parts. First, we present an active learning approach to syntactic error detection called SPADE [72]. SPADE introduces a weak supervision approach, where the system detects potential errors using indicative signals from the tables. SPADE then uses Probabilistic Soft Logic (PSL) [6] to create a probabilistic inference model that suggests examples to be labeled based on the agreements between user labels

and our indicative signals. Finally, SPADE uses a two-phase data augmentation process to enrich a dataset before training a deep-learning classifier to detect unlabeled errors.

In the second part of our work, we present UDATA [71], a novel method for data transformation and syntactic error detection. UDATA provides the ability to solve the data transformation problem with minimal human interaction. Users only need to provide a set of examples in the desired formats. Because the transforming data usually share common syntactic patterns, UDATA can cluster the data into different formats, learn the transformation programs to convert data between these common format patterns, and validate the output results. Users can optionally curate a representative set of examples to verify the results. We also describe DSL [70], a domain-independent approach for semantic matching. DSL uses similarity metrics as features to compare against labeled domain data and learns a matching function to infer the correct semantic labels for data. Since DSL's features are generated from similarity metrics and not the data, the system is domain-independent and only needs to be trained once to work effectively across multiple domains. DSL is used as the main component to infer pattern alignment in UDATA.

Finally, we present an approach for semantic error detection and correction called SEED (SEmantic Error Detector). SEED is a novel system for semantic error detection based on contrastive learning strategies for Dense Passage Retrieval (DPR) [42] and Natural Language Inference training. We leverage the ToTTo dataset [69] as our training and explore hard negative mining [25, 79] to generate negative examples to generate our training data. SEED is an end-to-end solution for semantic error detection in tables that contains: (i) multiple strategies to obtain positive and negative training examples from available datasets in relevant problems, (ii) an algorithm to convert the problem of semantic error detection into the traditional natural language inference (NLI) and leverage the available NLI pre-trained models.

1.4 Thesis Statement

Open-domain knowledge and closed-domain weak supervision can be leveraged to reduce human interaction and improve the accuracy in syntactic and semantic error detection and correction.

1.5 Contributions of the Research

This thesis presents a solution to syntactic and semantic error detection and correction in tables. Our research has the following main contributions:

- An active learning approach that uses weakly-supervised indicative signals and probabilistic models to detect syntactic errors with user feedback.
- A syntactic error correction method that leverages domain-specific languages (DSL), pattern clustering, and semantic alignment to automatically generate transformation programs to correct syntactic errors.
- A semantic error detection and correction system that can verify and correct factual information in tables by extracting evidence from open-domain web text.

The first two contributions are based on my earlier conference papers [70, 71, 72]. The final contribution is based on my latest paper, which is submitted and under review.

1.6 Outline of the Thesis

The thesis is organized as follows. In Chapter 2, we introduce our syntactic error detection approach. Chapter 3 presents our semantic labeling method, which is used as an algorithm in

Chapter 4. Chapter 4 explains the data transformation and syntactic error correction method. In Chapter 5, we describe our semantic error detection and correction solution. Related work is included in the chapters mentioned above, and Chapter 6 summarizes the thesis and presents some important topics for future work.

Chapter 2

Syntactic Error Detection

Supervised error-detection systems require a significant amount of training data to achieve satisfactory performance. Even though recent research [32, 46, 57, 62] focuses on reducing the amount of user labeling by using clustering and few-shot learning approach, the size of training data needed is still large. Labeling errors in a large and noisy dataset can require significant user effort and time. On the other hand, unsupervised approaches [18, 37, 59, 90], which do not need labeled data, rely heavily on a fixed inductive bias for detection. However, erroneous values in one database may be normal in another. Therefore, existing methods that work intrinsically in one domain often fall short of adapting extrinsically to other domains [57].

To cope with the lack of training data and support more generalizable error detection, we present a novel semi-supervised error detection system called SPADE ♠. As machine learning techniques have been proven to have high performance in error detection [32, 62], we integrate a machine learning classifier with an active learning process to reduce the amount of labeled data and an accompanying data augmentation algorithm to synthesize additional training data automatically.

SPADE introduces an active learning process where it initially improvises a set of signal functions to detect potential errors. We categorize our signal functions into two categories: internal and external. Internal signal functions analyze values within data attributes to determine the outliers. External signal functions leverage external knowledge, such as pre-trained language models and Web table data, to identify other errors. In the active learning process, the reliability scores of signal functions are equally initialized and continuously updated based on their agreement with user labels using Probabilistic Soft Logic (PSL) [6]. By accepting user feedback and progressively optimizing our active learning model, we can reduce the number of labeled examples required and improve our system's generalization.

To generate more training data, SPADE includes a two-phase data-augmentation module that propagates the labels of user-labeled examples based on the similarity of signals and PSL inference scores and generates synthetic examples with string transformation learned from user-corrected values. The combination of user-labeled and synthetically-augmented data ensures that our deep learning classifier has enough training data to achieve high performance, as shown in our evaluation.

Contribution. In this chapter, we present a semi-supervised solution for error detection that can accurately detect errors with limited user labels. Our solution provides two key technologies: (i) A *novel active-learning process* that uses probabilistic models to model the agreements between user labels and error signals to adaptively identify the actual errors; (ii) A *two-phase data augmentation algorithm*, which propagates user labels based on signal functions and generates synthetic errors using string transformations, to obtain enough training data for deep classification training.

2.1 Learning to Detect Errors

In this section, we define the error detection problem and discuss our approach in SPADE to solve the problem.

2.1.1 Problem Definition

This research focuses on solving the syntactic error detection problem. As syntactic errors can be identified by examining other values in the same ttribute, it can be formulated as a single-attribute error detection problem where errors can be found by examining each attribute independently. Errors that require examining the relationship between columns to detect (semantic errors) are also important but are not considered in this chapter. We can define the problem of syntactic error detection as follows:

Definition 2.1 (Syntactic Error Detection) *Given a data attribute A with n cells $\{A[1], A[2], ..., A[n]\}$, find a label vector L of size n so that $L[i] = 0$ if $A[i]$ is an error and $L[i] = 1$ if $A[i]$ is a normal cell value.*

Using the above definition, we can also formulate *error detection* as a binary classification where normal values are positive examples and errors are negative.

For the rest of the chapter, when we refer to error detection in a data source with n attributes $\{a_1, a_2, ..., a_n\}$, we consider it a set of n multiple error detection sub-problems where each sub-problem targets a single attribute a_i. Also, since attributes usually correspond to columns in tables, we will use the two terms interchangeably throughout the chapter.

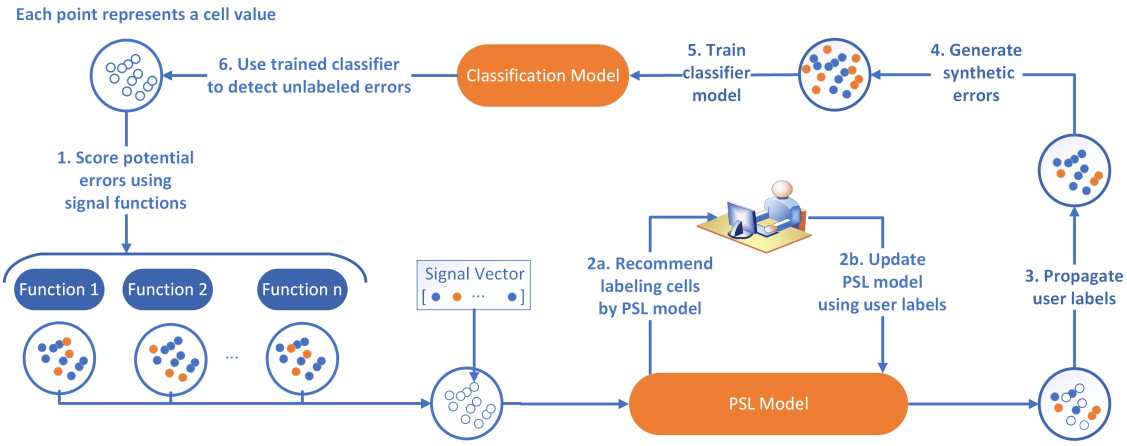


Figure 2.1: System Overview

2.1.2 Overall Approach

The main purpose of our active learning process is to quickly identify representative examples for user labeling. In SPADE, it can be achieved by using signal functions and the PSL model. Signal functions cover a set of indicative signals that can detect potential errors such as rarity in values, misspellings, or uncommon formats. Based on user feedback, the PSL model assigns a reliability score for each signal function and updates it throughout the active learning process. By focusing only on reliable signal functions, SPADE can select representative errors for user labeling in a large and noisy dataset.

As shown in Section 2.1.1, the *error detection* problem can be formulated as a binary classification problem. Since the number of labeled cells in active learning is limited, SPADE includes a two-phase data augmentation algorithm that enriches both positive and negative examples to ensure that our binary classifier has enough training data.

Figure 2.1 shows the overall workflow in SPADE. In the figure, blank circles represent unlabeled cells, while color-coded circles represent normal (blue) and erroneous (orange) cells. SPADE starts by initializing the set of unlabeled cells. For each active iteration, SPADE first computes

the signal values for each cell (Step 1). Next, these signal values are fed into our PSL model, and the model infers the error probabilities for all cell values. SPADE selects the cell with the highest error probability $A[i^*]$ for user labeling (Step 2a). The label is then used to update our PSL model (Step 2b) and the loop repeats until SPADE reaches the number of active iterations. The set of labeled cells is fed to our two-phase data augmentation algorithm (Steps 3 and 4) and the augmented data are then used to train a binary classifier (Step 5) to detect unlabeled errors (Step 6).

2.1.3 Signal Functions

SPADE contains two sets of signal functions that capture internal and external signals for potential errors. We define a *signal function* as follows:

Definition 2.2 (Signal function) *A signal function is a function that maps a cell value c_i to range $[0, 1]$ to indicate the error probability of c_i.*

This section explains the general ideas and reasons for including specific signal functions.

Internal Signal Functions — Internal signal functions are signal functions that can be computed using values within the column, and errors related to internal signals concern the rarity of the attribute data. Therefore, our internal signal functions focus on analyzing the frequencies of cell values within the column and then report the rarity. We apply the hierarchical syntactic patterns widely used in transformation learning systems [71, 80] and compute data frequency under four levels of abstractions: characters, symbols, grouped symbols, and punctuation marks. In symbol sequence, all characters are converted into their symbols using regex as follows: $[A - Z] \Rightarrow A$, $[a - z] \Rightarrow a$ and $[0 - 9] \Rightarrow 0$. In a grouped symbol sequence, we allow quantified regex so

that one symbol can replace a sequence of adjacent same-class characters. For example, with the string “1900, San Francisco CA", its symbol sequence is *“0000, Aaa Aaaaaaaaa AA"* and its grouped symbol sequence is *“0 Aa, Aa A"*. Our internal signal functions include:

- **Value frequency**: compute the frequency of cell values in a column.

$$f(A[i]) = \frac{|\{t = A[i] : t \in A\}|}{n}$$

- **Symbolic frequency**: compute the symbol frequency of cell values in a column.

$$f(A[i]) = \frac{|\{\text{sym}(t) = \text{sym}(A[i]) : t \in A\}|}{n}$$

- **Grouped symbolic frequency**: compute the grouped symbol frequency of cell values in a column.

$$f(A[i]) = \frac{|\{\text{sym}_+(t) = \text{sym}_+(A[i]) : t \in A\}|}{n}$$

- **Punctuation frequency**: compute the frequency of punctuation set for each cell value in a column.

$$f(A[i]) = \frac{|\{\text{punct}(t) = \text{punct}(A[i]) : t \in A\}|}{n}$$

External Signal Functions External signal functions aim to locate errors by leveraging information outside the tables, such as general human knowledge. We leverage the FastText embedding model [8] to characterize the information in general human knowledge. As FastText is trained using billions of tokens, their vocabulary can represent human vocabulary, and tokens missing in FastText have a high chance of errors. Moreover, we preprocess VizNet [35], an extensive Web table corpora, and compute statistics such as n-gram frequencies and word frequencies. As Web tables usually contain numerical or mixed data, analyzing VizNet can help identify potential errors, such as uncommon numerical formats or rare mixed values. SPADE currently supports the following external functions:

- **VizNet N-gram frequency**: compute the minimum 2-gram frequency of cell values in VizNet Web table corpora. In particular, a list of 2-gram character-based sequences is generated from a cell value. SPADE then computes the frequency of each n-gram and reports the minimum frequency.

$$f(A[i]) = \min \{\text{VizNetFreq}(s) \forall s \in \text{bigrams}(A[i])\}$$

- **FastText out-of-vocabulary**: check if any word in the cell values is out of FastText's vocabulary.

$$f(A[i]) = |\{\text{outOfFastText(t)} : t \in \text{tokenize}(A[i])\}| > 0$$

- **English out-of-vocabulary**: check if any word in the cell values is unknown in English vocabulary [*].

$$f(A[i]) = |\{\text{outOfVocab(t)} : t \in \text{tokenize}(A[i])\}| > 0$$

- **Missing Value**: check if a cell is a missing value

$$f(A[i]) = \mathbb{I}\{A[i] = \emptyset\}$$

2.1.4 Probabilistic Soft Logic

We use PSL to model the relationships between signal functions and user-annotated labels. PSL is a probabilistic graphical modeling framework built upon hinge-loss Markov Random Fields (HL-MRF) [6]. PSL eases HL-MRF modeling by allowing model definition using first-order logic syntax. A PSL model consists of a set of predicates and first-order logic weighted rules constructed from the predicates. An example of the PSL rule is shown below:

$$w : \textsc{label}(c, 1) \wedge \textsc{hasSignal}(s, c) \Rightarrow \textsc{badSignal}(s)$$

where w is the weight of the rule, LABEL, HASSIGNAL and BADSIGNAL are three predicates, c and s are variables, and 1 is a constant. In cases where w is undefined, the PSL rules become hard constraints and must be followed in the inference. During inference, the predicates are grounded

[*]https://github.com/barrust/pyspellchecker

by constants, and probabilities of the grounded predicates are inferred using convex optimization with relaxation.

2.1.4.1 PSL Model

In SPADE, we have four different predicates: $\textsc{error}(c)$ indicates if a cell c is error or not, $\textsc{badSignal}(s)$ denotes if a signal s is reliable or not, $\textsc{label}(c, l)$ shows the user label of cell c (normal value $l = 1$, error $l = 0$, or unlabeled $l = -1$), and $\textsc{hasSignal}(c, s)$ corresponds to the value of signal function s on cell value c. Based on these four predicates, we have a set of PSL rules in SPADE as follows:

$$\neg\textsc{error}(c) \tag{2.1}$$

$$\textsc{label}(c, 0) \Rightarrow \textsc{error}(c) \tag{2.2}$$

$$\textsc{label}(c, 1) \Rightarrow \neg\textsc{error}(c) \tag{2.3}$$

$$\textsc{label}(c, +l) = 1. \tag{2.4}$$

$$\textsc{label}(c, 1) \wedge \textsc{hasSignal}(c, s) \Rightarrow \textsc{badSignal}(s) \tag{2.5}$$

$$\textsc{label}(c, 0) \wedge \textsc{hasSignal}(c, s) \Rightarrow \neg\textsc{badSignal}(s) \tag{2.6}$$

$$\textsc{hasSignal}(c, s) \wedge \neg\textsc{badSignal}(s) \Rightarrow \textsc{error}(c) \tag{2.7}$$

$$\textsc{hasSignal}(c, s) \wedge \textsc{badSignal}(s) \Rightarrow \neg\textsc{error}(c) \tag{2.8}$$

Rule 2.1 is the prior rule that states that cell values are generally normal. We set the prior rule to have a low weight since we want them to be dominated by other PSL rules. Rules 2.2 and 2.3 are to enforce that user labeling is always correct, and Rule 2.4 ensures that the sum of

three probabilities, if a cell is either normal, error or unlabeled, is 1. These rules are unweighted since they are hard constraints in PSL. Rules 2.5 and 2.6 model the relationships between signal functions and user labels. Functions that agree with users are more reliable than functions that disagree with user labeling. Rules 2.7 and 2.8 require that cells scored highly by *good* signal functions are more likely to be errors and vice versa. We use the same weight for all these rules.

2.1.4.2 PSL Inference

As shown in Figure 2.1, SPADE conducts collective inference using the above PSL model and output probabilities of two predicates: ERROR and BADSIGNAL. Values of BADSIGNAL predicate denote the quality of each signal function and represent the importance of each signal function in active example selection. ERROR predicates indicate the error probability of cell values, and the cell with the highest error probability is reported to users for labeling. Two of the most common criteria for selecting active examples are informative and representative [36]. In SPADE, the cell with the highest error probability is selected since it is the most informative example for updating the PSL model. Our active learning relies on PSL inference to select a set of reliable signal functions to detect potential errors. By labeling the cell C_i with the highest error probability, users can help verify the correctness of a subset of signal functions that predict C_i as an error. Therefore, our PSL model can quickly select the most reliable signal functions. Example 2.1 illustrates how SPADE's PSL inference works.

Example 2.1 *Using the dataset in Table 1.5, in column Household disposable income, value "-0.5" is a potential error if we use a format frequency signal function since it is the only value with the minus sign "-". However, after user labeling, "-0.5" is a normal value. Using Rules 5 and 6, signal functions that conflict with user labels are more likely to become* BADSIGNAL. *Then using Rules 7 and 8, if a*

signal function is BADSIGNAL*, it will have a weight penalty in error inference and vice versa. For the column Household disposable income, signal functions that rely on format frequency will receive the penalty.*

After each active learning iteration, we update the values of LABEL predicates to reflect the labels of newly labeled cells before the inference.

2.1.5 Data Augmentation

Using the labeled data from users, SPADE performs a two-phase data augmentation process: *synthetic labeling* and *error generation*. The main goal of data augmentation is to increase the quantity and coverage of training data in SPADE. Existing supervised error detection approaches usually suffer from the lack of labeled data, and obtaining user labels is time-consuming. In this method, we use label propagation to extend the labeled data to similar values within close distances. Moreover, since the errors are usually rare in data, we generate additional synthetic errors to balance the data before binary classifier training.

2.1.5.1 Label Propagation

In SPADE, we propagate user labels of labeled cells to other cell values with similar signal vectors. We called two signal vectors similar if:

$$\textsc{SimVector}(c_i, c_j) = \begin{cases} 1 & \text{if } |f(c_i) = f(c_j)| \leq \epsilon \; \forall f \in F \\ 0 & \text{otherwise} \end{cases}$$

where F is the set of signal functions in SPADE and ϵ is a predefined threshold. Smaller ϵ means propagation has reached fewer examples and may require more active learning iterations to reach the maximum performance. In comparison, larger ϵ can cause labels spreading to values with different properties, resulting in wrong synthetic labels. We investigate the effect of different ϵ values in Section 2.3.4. SPADE uses ϵ with a value of 0.01.

2.1.5.2 Error Generation

Inspired by the idea of data augmentation in HoloDetect [32], we apply their error generation algorithm to enrich our training data. However, to increase the transformation accuracy, we extend the set of transformation operations to include the following:

- InsertAt(s, i): Insert s at position i.
- InsertAfter(s, c): Insert s after character c .
- InsertBefore(s, c): Insert s before character c.
- Replace(s, c): Replace an occurrence of string c with string s.
- ReplaceAll(s, c): Replace all occurrences of string c with string s.
- Delete(s): Delete an occurrence of substring s.
- DeleteAll(s): Delete all occurrences of substring s.

Example 2.2 shows how our system can generate better examples compared to HoloDetect.

Example 2.2 *To transform "Los Angeles" into "Los Angeles CA", HoloDetect can learn the transformation:* Insert(" CA"). *However, when applying the transformation function to a new string "San*

Jose," HoloDetect randomizes the inserting position and can create multiple different results such as "San JoCAse" or "SCAan Francisco." In SPADE, by using the operation InsertBefore("$") *where "$" is the end-of-string character, we can ensure that the generated errors are more accurate.*

2.1.6 Feature Extraction.

To create a representation for cell values and provide information to our classification model, we extract semantic and syntactic features to ensure that we can create the most informative cell value representation. We included both semantic and syntactic features since errors are usually related to one of these two aspects. For example, format inconsistencies are related to syntactic features, while typos are usually related to semantic features.

Syntactic Features. We follow the same hierarchical syntactic patterns used for internal signal functions to capture the syntactic patterns. SPADE calculates the Bag-of-character vectors for each cell value in three different syntactic abstraction levels: characters, symbols, and grouped symbols.

$$f_{char}(A[i]) = \{\text{count}(w)|w \in A[i]\}$$

$$f_{sym}(A[i]) = \{\text{count}(\text{sym}(w))|w \in A[i]\}$$

$$f_{sym+}(A[i]) = \{\text{count}(\text{sym}_{+}(w))|w \in A[i]\}$$

$$f_{syntactic}(A[i]) = \Big[f_{char}(A[i]), f_{sym}(A[i]), f_{sym+}(A[i])\Big]$$

Semantic Features. Pre-trained word embeddings are a common technique for capturing the meanings of string values. We use FastText [8] embedding to capture our cell semantic meanings.

The average character embeddings and word embeddings for each cell value are used as our semantic features.

$$f_{char\_ft}(A[i]) = \frac{\sum_{c \in A[i]} \text{FastText}(c)}{|A[i]|}$$
$$f_{word\_ft}(A[i]) = \frac{\sum_{c \in A[i]} \text{FastText}(c)}{|A[i]|}$$
$$f_{semantic}(A[i]) = \Big[f_{char\_ft}(A[i]), f_{word\_ft}(A[i]) \Big]$$

2.1.7 Classifier Training

The overall architecture of our classification model is shown in Figure 2.2. While the number of dimensions in syntactic features, which includes Bag-of-character vectors, can vary between different datasets, the size of semantic features is equal to the dimensionality of FastText (300) embedding. To ensure that the model is not biased toward the features with more dimensions, we provide a *feature compression* module where we reduce the dimensions of each feature group to a fixed size of 20. Our *compressor* module contains a linear fully-connected layer that reduces the number of dimensions and two intermediate ReLU activation functions. This encourages sparse representation in the data and prevents overfitting. The main network concatenates all *compressor* outputs and processes the concatenated vectors in another *compressor* module with one-dimension output. In the end, SPADE applies sigmoid activation so that the output is in the range of $[0, 1]$. All values with error probabilities higher than 0.5 are identified as errors. We train the model using binary cross-entropy loss function and AdamW optimization method [56]. We use the learning rate of $1e^{-5}$ and the batch size of 128.

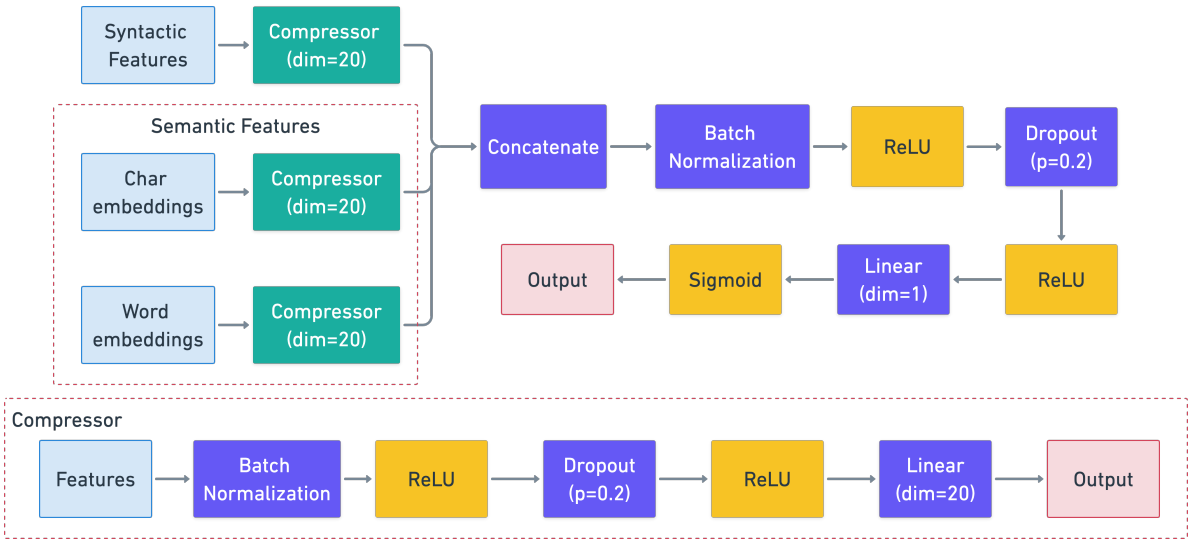


Figure 2.2: Classifier architecture

2.2 Related Work

Previous work on error detection includes supervised [32] and unsupervised [37, 90] approaches. The current trend in supervised error detection reduces the number of labeled examples required to detect errors while maintaining good performance. Heidari et al. [32] apply data augmentation to solve error detection as a few-shot learning problem. On the other hand, existing unsupervised methods exploit massive Web table corpora to understand the data distributions and statistically detect the values that are not aligned with the learned distributions. Auto-Detect [37] uses cooccurrence between values in Web table corpora to train an unsupervised model for detecting syntactic outliers. In contrast, Uni-Detect [90] leverages different statistical hypothesis tests for different types of errors on web table corpora to identify errors in pre-defined types. The major drawback of these unsupervised systems is their prior user settings, such as co-occurrence dependency in Auto-Detect or the set of pre-defined error types in Uni-Detect, which prevent them from generalizing. SPADE aims to solve the issues of both supervised and unsupervised

approaches by accepting a minimal number of labeled examples to remove the prior user settings while ensuring that the system can perform with minimal training data.

The idea of SPADE also applies to other interactive error detection systems. NADEEF [18] takes assertion rules as inputs and outputs the violated cell values. KATARA [16] marks errors that do not follow the entity relationships in given knowledge bases, and dBoost [59] uses histogram and Gaussian modeling to detect outliers. However, these systems, which require user inputs initially, usually suffer from low recall detection. Users need to understand the data errors before specifying the configurations, requiring going through the entire dataset. SPADE's active learning process is initialized by the system's signal functions, helping SPADE detect potential errors and overcome the low recall problem.

Another direction in interactive error detection is semi-supervised learning. In recent years, this research direction has drawn much attention from researchers. For example, ED2 [62] uses a double-dimensional active learning approach that recommends both suspicious columns and rows for user labeling. On the other hand, Raha [57] incorporates an ensemble method to apply detection strategies from existing systems to cluster data and produce representative examples for user labeling. Our study of SPADE also follows the methodology in this line of research. However, by incorporating PSL, we present a more reliable way of integrating information from users. As a result, SPADE can create a flexible active learning strategy compared to existing methods. In addition, SPADE proactively leverages data augmentation to generate more training data to overcome the small amount of labeled data in active learning systems.

2.3 Evaluation

We conduct multiple experiments to compare SPADE against the state-of-the-art error detection systems using different datasets and different quantities of labeled cells. We also investigate SPADE's running time, the importance of various features, and classification models in SPADE. Our system and datasets are available online[†].

2.3.1 Experimental Setup

We evaluate our system on five different datasets, as shown in Table 2.1. The Hospital dataset is a benchmark dataset used in several data cleaning papers [32, 57, 62]. Beers[‡], Rayyan [64], Flights [52] and Movies [19] are real-world datasets manually collected and cleaned by users. Rayyan, Flight, and Movies datasets contain multi-column errors, which are not covered in this approach. However, we want to include them in our evaluation since they are used in previous research [57, 62], and more evaluation datasets can show that SPADE is not fine-tuned to work in a specific domain.

We compare SPADE with Raha [57] and ED2 [62], the two state-of-the-art systems in active-learning error detection. We also include four other error detection tools that are widely used as baselines in error detection research: dBoost [59], NADEEF [18], KATARA [16] and ActiveClean [46]. We report precision, recall, and F1 score to evaluate the performance in the error detection task. Since the PSL model is probabilistic, we report the performance as the average of 10 independent runs. For readability, we omit the ± 0.01 in our tables when the standard deviations (SD) are less than 0.01 unless otherwise specified.

[†]https://github.com/minhptx/spade
[‡]https://www.kaggle.com/nickhould/craft-cans

Table 2.1: Information about our evaluation datasets

Dataset	Size	Error rate	Multi-column errors
Hospital	1000 x 20	0.048	No
Beers	2410 x 11	0.159	No
Rayyan	1000 x 11	0.086	Yes
Flights	2376 x 7	0.298	Yes
Movies	7390 x 17	0.062	Yes

Table 2.2: Performance comparison with baseline systems (# labeled cells = 20). * SD = ±0.02, ** SD=±0.03.

Approach	Hospital			Beers			Rayyan			Flights			Movies		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
dBoost	0.07	0.37	0.11	0.34	1.00	0.50	0.05	0.18	0.08	0.25	0.34	0.29	0.25	0.79	0.38
NADEEF	0.05	0.37	0.09	0.13	0.06	0.08	0.30	0.85	0.44	0.42	0.93	0.58	**1.00**	0.08	0.16
KATARA	0.44	0.11	0.18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ActiveClean	0.02	0.15	0.04	0.16	1.00	0.28	0.09	**1.00**	0.16	0.30	**0.99**	0.46	0.06	**1.00**	0.12
ED2	0.45	0.29	0.33	**1.00**	0.96	0.98	0.80	0.69	0.74	0.79	0.63	0.68	0.93	0.05	0.13
Raha	**0.94**	0.59	0.72	0.99	0.99	0.99	**0.81**	0.78	0.79	**0.82**	0.81	**0.81**	0.85	0.88	0.86
SPADE	0.93	**1.00**	**0.96**	**1.00**	**1.00**	**1.00**	0.80*	0.92*	**0.85**	0.81**	0.81**	**0.81***	**0.99**	0.83	**0.90**

In the evaluation, we set the maximum number of labeled cells for each column in SPADE to 20. We collect 20 labeled cells by running four active learning iterations with five active learning examples per iteration. To ensure fairness, for systems that detect errors at the dataset level, such as Raha and ED2, we set the maximum number of labeled cells in a dataset of n columns to be $20n$. We run all the experiments on a CentOS 7.8 machine with 72 cores, 754GB memory, and an NVIDIA RTX 2080 GPU.

2.3.2 Performance Evaluation

Table 2.2 shows that SPADE outperforms all baseline methods on five datasets in F1, except for Flights, where we have a comparable result with Raha. Specifically, SPADE excels over the best baseline system in each setting by 0.05 to 0.24 in F1 score. It is worth noting that both Raha and ED2 can detect multi-column errors, and thus they have a clear advantage compared to SPADE on

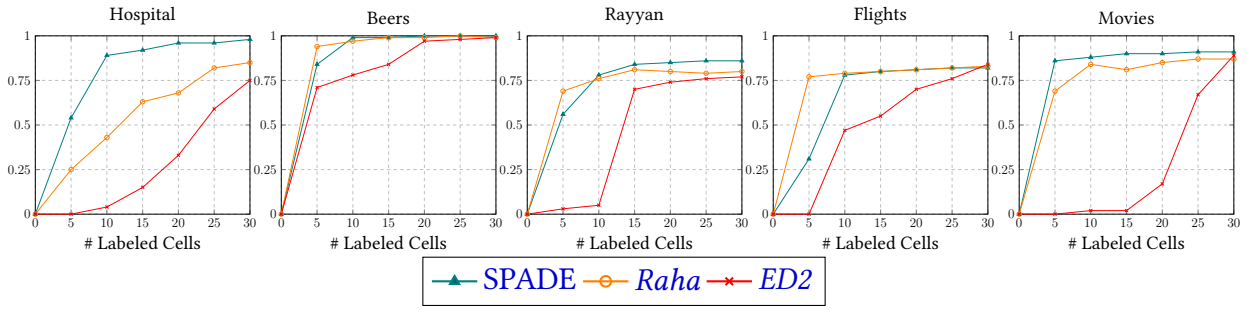


Figure 2.3: F1-score with different numbers of labeled cells

datasets with multi-column errors such as Flight, Rayyan, and Movies. Our high recall strongly supports SPADE's superior performance. SPADE's proposed data augmentation algorithm can generate significantly more unseen errors for the system. Therefore, SPADE is more generalizable than its competitors and is better at detecting unseen errors.

As observed in Table 2.2, in datasets with only single-column errors, such as Hospital and Beers, SPADE achieves a very high F1 score and a perfect recall. SPADE's performance drops in the other datasets with multi-column errors, which is understandable since there is always a set of multi-column errors that SPADE cannot detect in each dataset. The multi-column errors also contribute to SPADE's higher standard deviations in these datasets. There are multiple cases where a value can be both an error and a normal value within the same column depending on its functional dependency from other columns. This results in false positive and false negatives predictions in these datasets. Therefore, SPADE's performance in datasets with multi-column errors usually fluctuates.

Figure 2.3 shows that SPADE requires fewer labeled examples than the other two systems to reach its maximum F1 score. In Rayyan and Flights datasets, SPADE has a lower F1 score than Raha with five labeled cells since SPADE's first batch of active learning examples is suggested

purely on average signal function scores without user feedback. Starting from the second iteration (10 labeled cells), by leveraging user feedback and filtering reliable signals, SPADE reaches a near-peak performance and outperforms the baselines in all five datasets. Given more labeled examples, RAHA and ED2 can eventually catch up with SPADE in datasets with multi-column errors since SPADE is capped by our coverage of only single-column errors.

2.3.3 Running Time Evaluation

In this experiment, we evaluate SPADE's running time against two other active learning baselines: RAHA and ED2. To ensure fairness between different systems, we calculate the running time needed for each system to finish an iteration of active learning. As shown in Figure 2.4, RAHA has the fastest active learning time, and SPADE is the slowest system among the three. However, as we can see, the differences between SPADE and other baselines are marginal across most datasets. In the Hospital dataset, SPADE has a much higher active learning time than RAHA because SPADE handles error detection on each column separately, and the active learning time increases linearly with the number of columns. Since the Hospital dataset has the most columns in the five evaluation datasets, the differences in active learning time are the most noticeable.

2.3.4 Model Analysis

In this evaluation, we evaluate the importance of different components in SPADE and how removing or modifying these components will affect the system's performance.

Classifier Analysis. Table 2.3 shows SPADE's error detection performance using various classification models: Random Forest [12] and XGBoost [13]. Both classifiers are implemented using

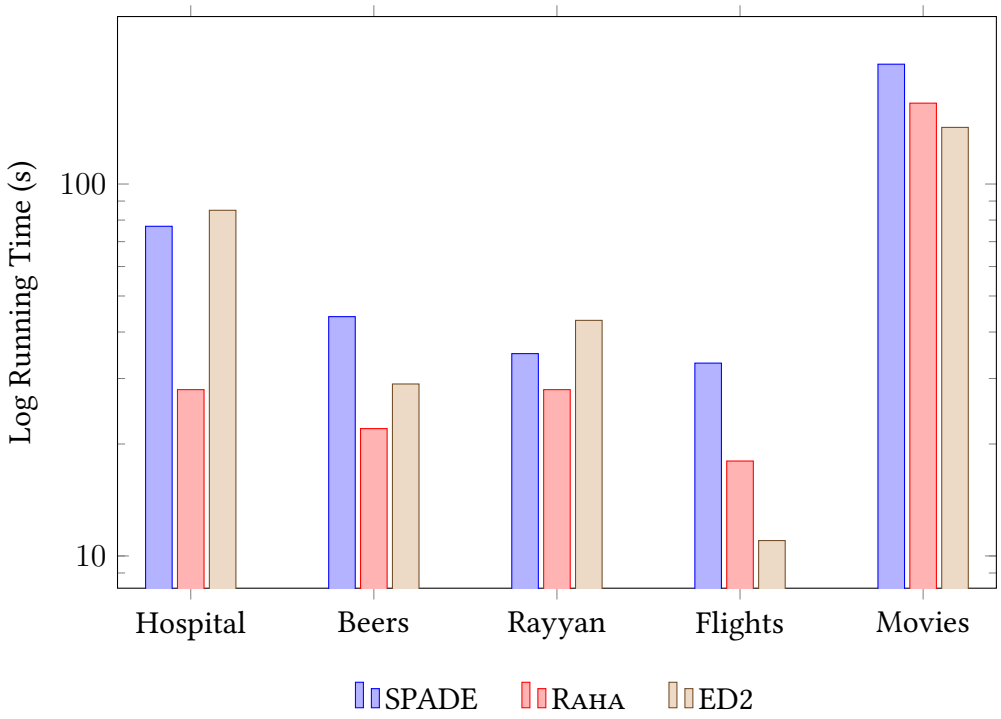


Figure 2.4: Active learning time in comparison with baseline systems

Table 2.3: Performance comparison between different classifiers (# labeled cells = 20)

Approach	Hospital			Beers			Rayyan			Flights			Movies		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
SPADE	**0.93**	**1.00**	**0.96**	**1.00**	**1.00**	**1.00**	**0.80**	**0.92**	**0.85**	0.81	**0.81**	**0.81**	**0.99**	0.83	**0.90**
Random Forest	0.57	0.99	0.73	0.94	**1.00**	0.97	0.79	0.85	0.82	0.86	0.76	**0.81**	**0.99**	0.83	**0.90**
XGBoost	0.86	0.99	0.92	0.99	0.99	0.99	**0.80**	0.85	0.83	**0.90**	0.70	0.79	0.85	**0.85**	0.85

the scikit-learn library [67] with the default parameters. As shown in Table 2.3, SPADE's deep-learning classifier outperforms all other classifiers in the five experimental datasets by 0.01 to 0.18 in terms of F1 score. As SPADE's deep classifier contains more parameters, it can better exploit features and capture errors' properties. Therefore, it is better at detecting unseen errors. When comparing RandomForest and XGBoost, the XGBoost classifier yields a more robust result across five experimental datasets. However, Random Forest's performance, while excels in Rayyan and Movies datasets, decreases severely in Hospital.

Table 2.4: Performance comparison with different sets of features

Features	Hospital	Beers	Rayyan	Flights	Movies
All features	**0.96**	**1.00**	**0.85**	**0.81**	0.90
Syntactic only	0.58	0.98	0.76	0.75	**0.91**
Semantic only	0.96	0.98	0.85	0.81	**0.91**

Table 2.5: Performance comparison with different ϵ values

ϵ	Hospital	Beers	Rayyan	Flights	Movies
$\epsilon = 0.01$	0.96	**1.00**	0.85	0.81	0.90
$\epsilon = 0.001$	0.96	0.99	**0.86**	0.81	0.86
$\epsilon = 0.005$	0.86	0.98	**0.86**	0.81	**0.91**
$\epsilon = 0.02$	**0.97**	**1.00**	**0.86**	0.81	0.89

Feature Analysis. In this experiment, we analyze the importance of features in SPADE's classifier. We run SPADE with all feature groups and then exclude each feature group, one at a time, to analyze its impact. As shown in Table 2.4, the system that uses only semantic features outperforms the one with syntactic features. Since we use character embeddings in our semantic feature group, our semantic features also include syntactic information, such as occurrences of characters and their frequencies. Therefore, using only semantic features gives superior performance compared to using only syntactic features and is comparable with SPADE.

Propagation Analysis. In this experiment, we investigate the effect of different values of ϵ in SPADE. Table 2.5 shows that changing epsilon values has a minimal effect on the performance of SPADE. It is also worth noting that despite the decrease in performance, using any ϵ reported in Table 2.5, SPADE still outperforms other baselines (Table 2.2).

2.4 Summary

In this chapter, we presented SPADE, a novel learning-based system for syntatic error detection that can effectively reduce the number of labels required for training while maintaining excellent performance in error detection. SPADE overcomes the weaknesses of existing supervised and unsupervised approaches by leverage active learning and data augmentation techniques. Active learning reduces the amount of required user labels while data augmentation increases our training data size. As a result, SPADE can achieve or outperform state-of-the-art results in syntactic error detection with less than 20 labeled examples.

Chapter 3

Semantic Labeling in Data Sources

Mapping attributes in data sources to a domain ontology is a necessary step in integrating different sources and mapping them to a domain ontology. The problem, which we call semantic labeling, requires annotating source attributes with classes and properties of ontologies. In this thesis, semantic labeling also plays an important role in our data transformation method, where semantic matching is required to infer the alignments between raw and groundtruth formats.

Several studies have been conducted to automate the process since labeling attributes manually is laborious and requires a sufficient amount of domain knowledge. However, automatic semantic labeling is difficult to perform accurately for several reasons. First, people have different ways of representing data with the same labels. Table 3.1 shows different formats that *PlayerPosition* can be found in soccer data. On the other hand, data from different labels can be very similar. For example, data of *NumberOfGoalsScores* and *NumberOfGamesPlayed* in soccer data are very similar because both are in numeric format with values ranging mainly from 0 to 50. Therefore, a good semantic labeling approach must deal with two different issues: distinguishing similar labels and recognizing the same labels from different data, both of which generally make the problem very hard.

Table 3.1: Different representations of *PlayerPosition*

Code	Abbreviation	Full form
1	GK	Goalkeeper
2	DF	Defender
3	MF	Midfielder
4	FW	Forward

To address these issues, we present a domain-independent machine-learning approach for semantic labeling. Our contribution is a novel way of using machine learning to solve semantic labeling as a combination of many binary classification sub-problems. Our machine learning model uses similarity metrics as features and learns a matching function to determine whether attributes have the same labels to infer the correct semantic labels. Because the matching function is not related to specific labels, our model is independent of labels and thus independent of the domain ontologies.

We evaluate our approach on many datasets from different domains. When the machine learning models are trained on another domain, the system achieves an average mean reciprocal rank (MRR) [17] of over 80% on four datasets. The results are even better if models are trained on the same domain. We also run experiments on the T2D Gold Standard data and achieve a higher F1 score compared to the property-matching approach in the T2K system [75].

3.1 Motivating Example

In this section, we provide an example to explain the problem of mapping source attributes to semantic types in a domain ontology. Suppose we want to map attributes in a data source named *WC2014* (Table 3.2), which contains information about players of national teams in World Cup

2014, to the DBpedia ontology. First, we define our target label, *semantic type*, as a pair of values consisting of a domain class and one of its properties <*class, property*>. For example, in Table 3.2, the correct semantic types of column *player*, *height* and *position* are <*dbo:SoccerPlayer, dbo:birthName*>, <*dbo:SoccerPlayer, dbo:height*> and <*dbo:SoccerPlayer, dbo:draftPosition*>. Semantic labeling systems attempt to identify these mappings automatically. However, this cannot be done without knowing about these semantic types in a domain.

Table 3.2: Sample data from World Cup 2014 players (WC2014)

player	**height**	**position**
Alan PULIDO	176	Forward
Robin VAN PERSIE	186	Forward
Miiko ALBORNOZ	180	Defender
Marouane FELLAINI	194	Midfielder

Therefore, the semantic labeling problem refers to a situation where we have already mapped one or more sources to a common ontology, and want to label new sources using the same ontology. For example, we have the data source *EPL* containing information about all England Premier League players, and it is already labeled with DBpedia semantic types (Table 3.3). Since we have information about the DBpedia ontology from the *EPL* source, we can label source attributes of *WC2014* based on this information. There are different ways to leverage domain data

Table 3.3: Sample data from England Premier League (EPL)

first name <*SoccerPlayer, birthName*>	**position** <*SoccerPlayer, draftPosition*>	**height** < *SoccerPlayer, height*>
Hazard, Eden	Midfielder	172
Cahill, Gary	Defender	191
Felliani, Marouane	Midfielder	194
Oezil, Mesut	Midfielder	180

from labeled sources for semantic labeling. Previous work uses labeled sources such as *EPL* as

training data to learn the characteristic of data in different attributes. Table 3.4 shows some feature values extracted from <*dbo:SoccerPlayer, dbo:birthName*> data. In our approach, we use *EPL* as our base data and compare attributes in *WC2014* with attributes in *EPL*. If these two attributes are similar such as column *first name* in EPL and column *player* in WC2014, we conclude that they have the same semantic types. Because we know that the semantic type of *first name* is <*dbo:SoccerPlayer, dbo:birthName*>, we infer that the semantic type of *player* is also <*dbo:SoccerPlayer, dbo:birthName*>.

Table 3.4: Some feature values extracted from <*SoccerPlayer, birthName*>

Feature	**Value**
all capitalized token	1
starts with char C	0.25
num len	0

The main difference between our approach and previous work is when faced with unseen semantic types. For example, consider the case where we have another labeled source named *BGL* containing information about players in Germany Bundesliga League (Table 3.5). *BGL* contains a column *salary* labeled as < *dbo:Person, dbo:salary* > - an unseen semantic type. In previous approaches, learned models need to be retrained to capture the data characteristic of < *dbo:Person, dbo:salary* >, and this process needs to be repeated for every unseen semantic type. There are a massive number of data sources and semantic types, which makes the possibility of facing new semantic types very high, and it is time-consuming to retrain the learning models each time. For our approach, we need to store data with the new semantic types for later comparison with unlabeled attributes.

Table 3.5: Sample data from Germany Bundesliga League (GBL)

name < *SoccerPlayer, birthName*>	**salary** < *SoccerPlayer, salary*>
Neuer; Manuel	150,000
Boateng; Jerome	90,000
Dante	100,000

3.2 Approach

In this section, we explain our approach to determine similarities between unlabeled and labeled attributes and use machine learning techniques to find the correct semantic type. Section 3.2.1 describes various similarity metrics used in our features and how we compute them. Finally, Section 3.2.2 describes details of how we use machine learning for semantic labeling.

3.2.1 Similarity metrics

In our approach, we exploit different similarity metrics that measure how attributes are similar to others. In this section, we describe these similarity metrics and explain how they can help in semantic labeling.

3.2.1.1 Attribute Name Similarity

In relational databases, web tables, or spreadsheets, tabular structures usually have titles for each column. We consider these titles as attribute names and use them to compare similarities between two attributes.

Definition 3.1 *Given two attributes named a and b, we have A and B as sets of character tri-grams extracted from a and b. The* ***attribute name similarity*** *is calculated using Jaccard similarity [58] as follows:*

$$S(a, b) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

In data sources, people usually name attributes based on the meaning of the data. Therefore, similarity in attribute names provides a good indication of the similarity in semantic types. However, as attribute names usually correspond only to ontology properties, using attribute names as the only metric can lead to false positives in labeling. For example, a column named *name* can refer to $<$ *dbo:Person, dbo:birthName* $>$ or $<$ *dbo:SportsTeam, dbp:clubName* $>$ depending on the sources. Collecting data sources from the web can also result in missing or noisy attribute names, which provide no information about the attributes.

3.2.1.2 Value Similarity

Value similarity is the most common similarity metric in different matching systems. In semantic labeling, attribute values play an essential role in identifying attributes with the same semantic types since they usually contain similar values. In our approach, we compute three value similarity metrics: Jaccard similarity and TF-IDF cosine similarity for textual data, and a modified version of Jaccard similarity for numeric values.

Definition 3.2 *Given two attributes named a and b with v_a and v_b as the corresponding sets of values, the* ***textual Jaccard similarity*** *[58] is computed as follows:*

$$S(a, b) = \frac{|v_a \cap v_b|}{|v_a \cup v_b|} \tag{3.2}$$

Definition 3.3 *Given set of attributes {a_1, a_2, ..., a_n} with a corresponding sets of values {v_1, v_2, ..., v_n}, the* ***TF-IDF cosine similarity*** *[58] is computed using the following steps:*

1. *We concatenate the values in {v_1, v_2, ..., v_n} by attribute to generate a set of documents: { D_1, D_2, ..., D_n}*

2. *For a document D_i, we calculate the corresponding TF-IDF vector W_i*

3. *We compute TF-IDF cosine similarity between two attributes a and b:*

$$S(a, b) = \frac{W_a \cdot W_b}{|W_a| \times |W_b|} \tag{3.3}$$

For numeric attributes, set-based similarity metrics such as Jaccard and cosine similarity do not work effectively because numeric data have continuous ranges of values. Therefore, we customize Jaccard similarity to work with ranges of values instead of sets of values.

Definition 3.4 *Given two attributes named a and b with v_a and v_b as the corresponding sets of values, the* ***numeric Jaccard similarity*** *is computed as follows:*

$$S(a, b) = \frac{\min(\max(v_a), \max(v_b)) - \max(\min(v_a), \min(v_b))}{\max(\max(v_a), \max(v_b)) - \min(\min(v_a), \min(v_b))} \tag{3.4}$$

For example, the numeric Jaccard similarity s of two attributes with values in the range [1912,1980] and [1940,2000] is computed as follows:

$$s = \frac{1980 - 1940}{2000 - 1912} = 0.45. \tag{3.5}$$

To reduce sensitivity to outliers, we only use the subsets containing values from the first to the third quartile instead of the all values in attributes.

3.2.1.3 Distribution Similarity

For numeric data, there are semantic types that we are unable to distinguish by using value similarity because they have the same range of values. However, since they have different underlying meanings, the distribution of their values may differ. For example, consider the example about *NumberOfGoalsScored* and *NumberOfGamesPlayed* in Table 3.1. Although they have the same range of values, *NumberOfGoalsScored* has skewed distribution because the high values are mainly distributed to *Forwards* and *Midfielders*. At the same time, *NumberOfGamesPlayed* is more likely to follow a near-uniform distribution.

Therefore, we analyze the distribution of numeric values contained in the attributes using statistical hypothesis testing as one of the similarity metrics. For statistical hypothesis testing in our approach, the null hypothesis is that the two sets of values are drawn from the same population (distribution), which may indicate that they come from the same semantic type. We use Kolmogorov-Smirnov test (KS test)[49] as our statistical hypothesis test based on the evaluation of different statistical tests in Ramnandan et al's research [74].

3.2.1.4 Histogram Similarity

Standard statistical hypothesis testing cannot be applied to textual data because there is no order in textual values. Moreover, we cannot use traditional correlation methods such as mutual information or KL-divergence since we are comparing attributes that do not appear in the same source.

Therefore, we calculate value histograms in textual attributes and compare their histograms instead. The statistical hypothesis test used for the histogram case is the Mann-Whitney test (MW test) [49]. The reason we use MW test instead of KS test is that histograms are not ordinal and using methods that compare two empirical value distributions, such as KS test, is not suitable. MW test computes distribution distances based on medians and, thus, is more appropriate to use for histograms.

When comparing a textual attribute with a numeric attribute, we also transform numeric data into histogram form and use the MW test to compute histogram similarity. For the example of *PlayerPosition* in Table 3.1, even though they have different representations, they have similar histogram forms because each position usually has similar frequencies over different data sources. For instance, because every soccer team usually has one goalkeeper, four defenders, four midfielders and two forwards, the histogram frequencies are likely to be $[\frac{1}{11}, \frac{4}{11}, \frac{4}{11}, \frac{2}{11}]$.

3.2.1.5 Mixtures of Numeric and Textual Data

As described above, similarity measures can only be applied to the textual part of attribute values, while some others only work on numeric parts (Table 3.6). Because textual similarity metrics are more important when comparing attributes with mostly text and numeric similarities are more important for attributes with numeric data, we need to adjust the values of these similarity measures based on the fraction of textual and numeric values contained in attributes.

Given r_1 and r_2 are fractions of textual data in the pair of attributes, the adjusted value of textual similarity value is computed as follows:

$$v_{adjusted} = \frac{(r_1 * r_2)}{(r_1 + r_2)} * v_{original} \tag{3.6}$$

Table 3.6: Similarity feature vector

Feature name	Explanation	Applied data types
ATT NAME	Jaccard similarity for attribute names	All
TEXT JACCARD	Jaccard similarity for textual data	Textual
TF-IDF COSINE	TF-IDF cosine similarity for textual data	Textual
NUM JACCARD	Modified Jaccard similarity for numeric data	Numeric
NUM KS	KS statistical test for numeric data	Numeric
MW HISTOGRAM	MW test for histogram	All

On the other hand, the adjusted value of numeric similarity value is computed as follows:

$$v_{adjusted} = \frac{[(1 - r_1) * (1 - r_2)]}{[(1 - r_1) + (1 - r_2)]} * v_{original} \tag{3.7}$$

The adjusted value is the product of the harmonic mean over r_1, r_2, and the original value. The reason for using harmonic mean follows the intuition that the corresponding similarity values are more reliable when two attributes have similar fractions of textual data or numeric data and vice versa.

3.2.2 Semantic Labeling

The overall framework is illustrated in Figure 3.1. The input of our system is an unlabeled attribute and a set of labeled attributes as domain data; the output is a set of top-k semantic types corresponding to the unlabeled attribute.

3.2.2.1 Overall Approach

Given a set of attributes {a_1, a_2, ... a_n}, we compute M-dimensional feature vectors f_{ij} ($i \neq j$). Each dimension k corresponds to a similarity metric, so $f[k]$ represents how similar attributes a_i and a_j are under metric k.

During the training phase, we label each f_{ij} as True/False, where True means that attributes a_i and a_j have the same semantic type and vice versa. To set up a new domain, we store a set of labeled attributes {a_1, a_2, ... a_n} as domain data and use them to compare against new attributes to infer the semantic types.

Given a new attribute a_0, the algorithm computes f_{0j} for all j ($j \neq 0$), and uses the learned classifier to label each f_{0j} as True/False. If the label of f_{0j} is yes, the algorithm says that the semantic type of a_0 is the semantic type that was recorded for a_j. From that, we can conclude the semantic type of a_0.

Previous approaches tried to predict the semantic label of a_0 based on the characteristic of recorded a_i. In contrast, our approach learns a classifier over similarity vectors. It is domain-independent because classification does not depend on the values in attributes but rather on the similarity scores of multiple metrics between the attributes.

Since there are no constraints on the number of True labels for each attribute, we develop a ranking method and only take the top-k results of semantic types. The ranking algorithm uses the predicted probabilities of the True class in classification as the confidence scores and ranks the candidate semantic types based on that.

3.2.2.2 Classifiers for Semantic Labeling

To choose the best classifier for semantic labeling, we run experiments on various classifiers and compare the results. Because we use class probabilities of classifiers as confidence scores, classifiers need to have class probabilities calculated from the feature vector to be applicable.

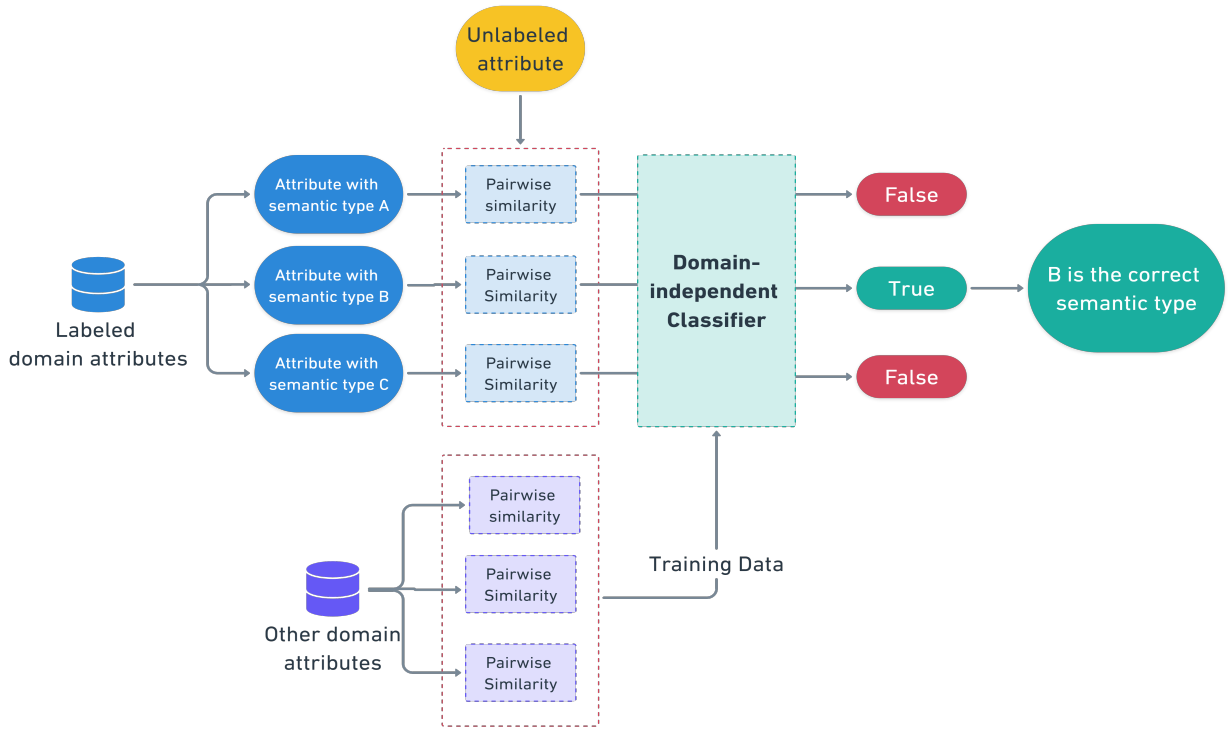


Figure 3.1: Overall framework of DSL

Therefore, we only consider Logistic Regression and Random Forests [11]. Details of the experiments are described in Section 3.4.2. According to Table 3.8-3.10, Logistic Regression achieves the best performance and is the selected classifier in our system.

3.3 Related Work

Ramnandan et al. [74] describe an approach that captures and compares distributions and properties of data corresponding to semantic types as a whole. They apply heuristic rules to separate numeric and textual data and then use TF-IDF and KS as measures to compare the data. In our

approach, we use more similarity features besides TF-IDF and KS test, which enables our system to better discriminate between semantic types. Our similarity metrics can be applied to both textual and numeric attributes by the method described in Section 3.2.1.5.

Ritze et al. [75] propose a new approach for annotating HTML tables with DBpedia classes, properties, and entities. Their system, T2K, uses metrics like Jaccard, Levenshtein, and deviation similarity to match attributes to properties and values. T2K also uses an iterative process to adjust property weights and filter the candidate sets until the similarity values converge. The system provides good results in entity and class matching but not property matching. However, since they exploit only value similarity for textual data and numeric similarity for numeric data for property matching, they face the same limitation as Ramnandan's work and achieve a lower performance compared to DSL.

Many approaches have used probabilistic graphical models to solve the problem of semantic labeling. Goel et al. [27] exploit the underlying relationships between attributes and values with attribute characteristics as features and use Conditional Random Fields (CRF) to label attributes. They assign semantic types to every attribute's value and then combine them to infer the semantic type for the whole attribute. Limaye et al. [53] use probabilistic graphical models in a broader problem as they annotate tables on the web by entities for cells, types for columns, and relationships for binary relations between columns. They exploit two feature functions that describe the dependency of column type with its values and header. The labels of all columns are then assigned simultaneously using a message-passing algorithm to maximize the potential function formulated by features and their weights. Mulwad et al. [60] extend the work of Limaye et al. by proposing a novel *Semantic Message Passing* algorithm that uses Linked Open Data (LOD) knowledge to improve the existing semantic message algorithm. These approaches require

the probabilistic graphical models to be retrained when handling new semantic types since their feature weights are associated with labels and need to be re-estimated for new semantic types. Also, graphical models do not scale well as the number of semantic types increases because of the explosion of different enumerations in the search space.

Mulwad et al. [61] implement their work into a complete system with multiple functions. They incorporated probabilistic semantic labeling with domain knowledge processing and data cleaning to produce a domain-independent semantic labeling system. However, their domain independence is limited since the system requires users to provide domain knowledge or apply preprocessing modules. In our semantic labeling system, the process is automatic, and the domain-independent learning models only require a small amount of domain data to perform well on semantic labeling.

Venetis et al. [88] present an approach to annotate tables on the web by leveraging existing data on the web. An isA database in the form of {instance, class} is extracted from the web using linguistic patterns and is used to produce column labels. The column labels are assigned by a maximum likelihood estimator that assigns a column with a class label that maximizes the fraction of column values in that label. Syed et al. [85] use Semantic Web data to infer the semantic models of tables. They annotate the table columns using the column names, if available, and values inside the columns to build a query to Wikitology. After that, columns are mapped to classes returned in the query result. Both the work of Venetis et al. and Syed et al. extract a vast amount of data from various sources to estimate the probability that a value belongs to a semantic type. Thus, their approach is restricted to domains where online data is widely available. In our approach, our learning model is not domain-specific; thus, we can use any domain as our training data, and the system can still label data from other domains effectively.

Gunaratna et al. [30] address a related problem called entity class resolution. Entity class resolution is similar to semantic labeling except that their targets are entity classes instead of semantic types. Their system, FACES, applies natural language processing (NLP) techniques to identify focus terms and uses text similarities to compare focus terms with entity class names in the ontology. Although FACES' approach works well in text documents because it is easy to detect focus terms in grammatical documents, it cannot be applied to most web data such as web tables, spreadsheets, or RDF stores because data values are unstructured and do not follow grammar rules such as numbers and named entity mentions. In contrast, our approach does not rely on NLP algorithms so that it can perform effectively in noisy data sources from the web.

3.4 Evaluation

In our experiments, we use four different datasets: city [74], weather [4], museum [86], and soccer. The soccer data set was created to provide a wide variety of semantic types and consists of numerous real-world data sets about soccer. The purpose of using many datasets from different domains is to evaluate our classifiers when applying a single learned classifier to multiple domains. Table 3.7 shows the general information about these data sets. The datasets and code used in our experiments have been published online*.

Table 3.7: Data sets from different domains in experiments

Data set	No. sources	No. semantic types	No. attributes
museum	29	20	217
city	10	52	520
soccer	12	14	97
weather	4	11	44

*https://github.com/minhptx/iswc-2016-semantic-labeling.git

3.4.1 Experimental Setup

In this section, we evaluate the performance of our system, which is called DSL (Domain-independent Semantic Labeler). The evaluation metric that we measure is the mean reciprocal rank (MRR) [17]. The details of the experimental setup are as follows:

1. Choose a labeling dataset A.
2. Suppose A consists of n sources {s_1, s_2, ..., s_n}, choose the number of labeled sources m in the dataset ($m < n$).
3. For every source s_i in A, perform semantic labeling using m labeled sources from s_{i+1} to s_{m+i+1}.

For example, the soccer dataset has 12 sources. If we have one labeled source, we label s_1 with labeled data from s_2, label s_2 with labeled data from s_3 and so on. Likewise, if we have five labeled sources, we label s_1 with labeled data in the set of sources s_2, s_3, ..., s_6 and continue through the entire data set.

For classifier training data, we follow the same process as above, but we manually label the computed feature vectors generated instead of running semantic labeling. To ensure that classifier training data is disjoint from labeling data, we choose distinct labeled sources for each process if the labeling dataset and training dataset are the same.

3.4.2 Classifier Analysis

In this experiment, we evaluate two classifiers: Logistic Regression and Random Forests, to choose the best classifier for semantic labeling.

Table 3.8 - 3.10 lists the results of two classifiers when being trained and tested on different datasets. We use city, museum, and soccer datasets to train Logistic Regression and Random Forest since we can generate sufficient samples for training data. For semantic labeling, we use all four datasets: soccer, museum, city, and weather, and set the number of labeled sources to 50% of the total number of sources in these datasets.

Table 3.8: MRR scores of different classifiers when training on soccer

Classifier	soccer	museum	city	weather
Logistic Regression	0.814	0.863	0.944	0.951
Random Forests	0.794	0.799	0.947	0.86

Table 3.9: MRR scores of different classifiers when training on museum

Classifier	soccer	museum	city	weather
Logistic Regression	0.815	0.845	0.940	0.951
Random Forests	0.820	0.778	0.830	0.898

Table 3.10: MRR scores of different classifiers when training on city

Classifier	soccer	museum	city	weather
Logistic Regression	0.782	0.807	0.965	0.955
Random Forests	0.802	0.728	0.912	0.807

Overall, Logistic Regression achieves a comparable performance to Random Forests, which is a surprising result, because Random Forests are the better classifier in other research. However, because of the issue where we need to use class probabilities as confidence scores, the results can be explained.

Logistic Regression class probabilities are computed using the following function:

$$P(y = 1|x) = sigmoid(w^T x) \tag{3.8}$$

where x is the feature vector and w are its coefficients. Because $P(y = 1|x)$ is a monotonically increasing function of $w^T x$, $P(y = 1|x)$ increases when $w^T x$ increases. Thus, feature vectors with higher similarity values have higher class probabilities in Logistic Regression models.

On the other hand, Random Forests calculate class probabilities based on fractions of samples of the same class in decision tree leaves. As long as the values are higher than splitting values in decision trees, feature vectors are split into the same branches and are likely to receive similar class probabilities. Therefore, using class probabilities of Random Forests as confidence scores performs worse.

Since the difference between accuracy in Logistic Regression and Random Forests are small, we consider the training time and labeling time of each classifier as additional measurements. Table 3.11 lists the average training time and labeling time of these classifiers.

Table 3.11: Training and labeling time of different classifiers

Classifier	**Training time**	**Labeling time**
Logistic Regression	144s	0.31s
Random Forests	157s	0.36s

Table 3.11 shows that Logistic Regression has a lower training and labeling time. Although the differences are minor, it provides an advantage, especially in real-world scenarios with large amounts of data. Using Logistic Regression also provides more meaningful insights of features because of its linear combination compared with a randomized algorithm such as Random Forests. Therefore, we use Logistic Regression as the classifier for the remaining experiments.

3.4.3 Feature Analysis

In machine learning classifiers, different features have different degrees of influence on the classification results. To analyze the importance of features in our similarity vectors, we train Logistic Regression on different datasets and extract coefficients of features. Table 3.12 shows coefficients of features when Logistic Regression models are trained on city, museum, and soccer data.

Table 3.12: Coefficients of features in Logistic Regression classifier

Feature	Train on soccer	Train on museum	Train on city
ATT NAME	4.41	6.08	0
TEXT JACCARD	1.88	0.88	9.16
TEXT TF-IDF	3.91	1.03	3.20
NUM JACCARD	4.21	3.28	12.68
NUM KS	1.78	0.78	7.25
MW HISTOGRAM	0.32	1.14	3.83

In general, all of our similarity features have a positive correlation with the classification result, which means that higher values in these similarity metrics result in higher probabilities that the attributes have the same semantic type. As we can see from the results, value similarity features play the most crucial role in Logistic Regression classifiers regardless of the training domain. Attribute names similarity has a good impact on soccer and museum data but not on city data because the city dataset does not have headers or titles for attributes. On the other hand, distribution and histogram similarity metrics have higher coefficients in city data because the city dataset contains mostly numeric attributes.

In conclusion, we have demonstrated that our similarity features contribute to the similarity in the semantic types of attributes. However, the importance of features in the learned classifiers can vary according to the training data, as shown in Table 3.12.

Table 3.13: MRR scores of DSL and SemanticTyper on soccer dataset

Classifier	Number of labeled sources				
	1	2	3	4	5
DSL (train on soccer)	0.625	0.782	0.777	0.800	0.815
DSL (train on city)	0.601	0.785	0.788	0.808	0.820
DSL (train on museum)	0.600	0.781	0.788	0.808	0.810
SemanticTyper	0.608	0.711	0.720	0.720	0.732

Table 3.14: MRR scores of DSL and SemanticTyper on museum dataset

Classifier	Number of labeled sources				
	1	2	3	4	5
DSL (trained on soccer)	0.471	0.665	0.719	0.755	0.790
DSL (trained on museum)	0.463	0.652	0.709	0.752	0.792
DSL (trained on city)	0.472	0.659	0.706	0.713	0.730
SemanticTyper	0.491	0.615	0.656	0.699	0.697

3.4.4 Semantic Labeling

In this experiment, we evaluate the performance of DSL (Domain-independent Semantic Labeler) compared with SemanticTyper [74]. Our experiments run on configuration with only one to five labeled data sources for every dataset (Weather dataset has only four sources, so the maximum number of labeled sources is three). This setting replicates real-world scenarios where labeled sources are hard to find, and manual labeling is tedious. For DSL, we follow the setup in section 3.4.1 while having soccer, city, and museum as our classifier training dataset iteratively. For SemanticTyper, the MRR scores reported are the MRR scores when being trained on the testing domains. The weather domain is only used in semantic labeling because it cannot provide a sufficient number of feature vectors for training classifiers.

It can be seen from Tables 3.13 - 3.16 that our approach outperforms SemanticTyper in all four evaluation datasets. Although there are slight changes in performance when the classifiers are trained on different domains, the changes are not significantly different, and it shows that

Table 3.15: MRR scores of DSL and SemanticTyper on city dataset

Classifier	Number of labeled sources				
	1	2	3	4	5
DSL (trained on soccer)	0.913	0.932	0.932	0.941	0.945
DSL (trained on museum)	0.912	0.927	0.928	0.941	0.944
DSL (trained on city)	0.914	0.928	0.930	0.939	0.944
SemanticTyper	0.856	0.893	0.893	0.913	0.919

Table 3.16: MRR scores of DSL and SemanticTyper on weather dataset

Classifier	Number of labeled sources		
	1	2	3
DSL (trained on soccer)	0.899	0.951	0.977
DSL (trained on museum)	0.899	0.951	0.977
DSL (trained on city)	0.902	0.955	0.977
SemanticTyper	0.852	0.920	0.955

our approach is robust across multiple domain datasets. According to the table, training the classifier from the same domain, which provides more information about the characteristic of data in domains, slightly improves the accuracy of the classifier.

We also evaluate our system on the T2D Gold Standard dataset[†] and compare our result with T2K system's approach for properties matching [75]. As described in Ritze's work, labeled sources are extracted from the DBpedia ontology. After that, they divided the T2D Gold Standard dataset into two equal-sized parts: an optimization set and an evaluation set. The optimization set is used to optimize the essential parameters for the system, and the result are evaluated on the evaluation set. Although we are unable to reconstruct the exact experiment, we approximated the result by using the following configuration as an alternative:

1. Collect DBpedia ontology data in table format as labeled sources.

[†]http://webdatacommons.org/webtables/goldstandard.html

Table 3.17: MRR scores of DSL and T2K on T2D Gold Standard dataset

DSL	T2K (evaluation)	T2K (optimization)
0.773	0.730	0.700

2. For every attribute in the ontology, extract only 1000 first values as the set of values for the attribute.

3. Train the classifiers on a combination of soccer, museum, and city datasets to enrich the training data.

4. Test semantic labeling (properties matching) on the entire T2D Gold Standard dataset.

Table 3.17 shows the results of DSL in comparison with T2K. Although our approach is not optimized on the optimization set as T2K, we achieve better accuracy on the dataset. Moreover, our classifiers have been trained on different domains, and we only use 1000 values for every attribute as domain data instead of the entire set of values. Because we exploit more similarity features, our approach achieves better discriminative ability for the various semantic types. The evaluation also shows that we have a robust, domain-independent system that only needs to be trained once before using it for semantic labeling in a wide range of domains.

3.5 Summary

In this chapter, we presented a novel domain-independent approach for semantic labeling that leverages similarity measures and machine learning techniques. In our system, we capture the patterns of matching decisions given the similarity scores between unlabeled attributes and labeled data to find the correct semantic types. The approach plays an important role in aligning semantic structures in our syntactic error correction approach (Chapter 4). Since our similarity

features are independent within a semantic type and across other semantic types, it allows us to train the machine learning model only once and use it in multiple different transformations.

Chapter 4

Syntactic Error Correction

The problem of syntactic error correction can be reformulated into a data transformation problem. Given that the erroneous data is the source data and the ground truth data is the target data, learning the program to transform erroneous data into ground truth data is equivalent to correcting the errors. In this chapter, we present a novel solution for data transformation, and this solution can be applied to syntactic error correction using the above formulation.

In recent years, much research has been undertaken to solve the data transformation problem with less human interaction. However, previous methods, such as programming-by-example (PBE) or interactive data cleaning, still require human labeling for input. Both interactive cleaning [46, 73] and PBE [28, 80, 92] systems require user input in different forms: transformation rules or sample input/output pairs. Because of the dependency on human input, interactive data cleaning and PBE are rarely scalable when the volume of data and the number of data sources increase. These systems have two main difficulties: 1) handling format diversity; and 2) result verification. Source attributes can contain multiple data formats and thus require annotation for each format to learn the transformations. Moreover, the transformation result must be validated to guarantee accuracy before being used in other applications. The process of providing input

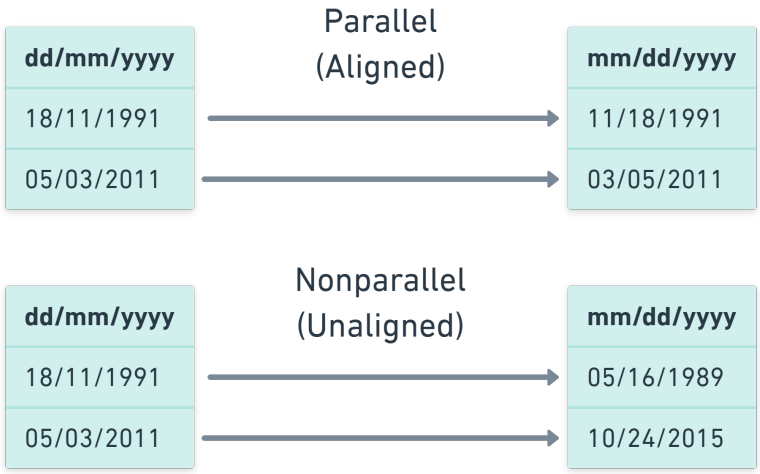


Figure 4.1: Parallel and nonparallel input-output data

examples for each format and verifying the transformed results eventually slows with thousands or millions of records. In this chapter, we introduce UDATA, a novel unsupervised system to solve the data transformation problem with minimal human interaction. UDATA takes as input a set of string values to be transformed, along with output examples, in a target format. However, unlike prior work, UDATA does not require output examples to align with the input data. Examples of parallel and nonparallel data are shown in Figure 4.1. Since target output examples have no connection with the transforming data, they can be specified before data collection: thus, no human interaction is required to learn the transformation programs. In this chapter, we call the problem of nonparallel data transformation *unsupervised* data transformation, in contrast to the *supervised* problem where alignments between input and output data are provided.

The main idea of UDATA is that data contained in formats usually follow a set of common syntactic patterns. Therefore, with enough data, UDATA can infer the underlying patterns and thus leverage these patterns to learn the transformations. The problem then goes from the traditional value-to-value to the new pattern-to-pattern transformation. Since each pattern usually contains

multiple string values, the similarity between the values in the pattern can provide the necessary information for UDATA to find the mappings between tokens. In the example of nonparallel data in Figure 4.1, if we have sufficient data, UDATA can conclude that the second set of digits in the input data should be mapped to the first set of digits in the output data, which shares the same range of values.

To summarize, we make the following key contributions:

- We formulate the problem of unsupervised data transformation, which has not been considered in previous research.
- We present a learning-based system to solve unsupervised data transformation.
- We propose a validation algorithm to validate our transformation result with high accuracy.
- We evaluate our system in five different data sets, where the system achieves an average accuracy close to the previous best PBE systems without any labeled data.

In our experiments, we evaluated the performance of our transformation and validation approaches. In the transformation evaluation, our system achieves comparable performance to IPBE, one of the state-of-the-art PBE systems with no parallel input-output data. In the validation experiments, our approach has an accuracy of 0.93, which makes it possible to identify incorrect transformation results automatically so they can be reviewed.

4.1 Overview

In data sources, data values are usually stored in string format. In addition, special values such as numeric (e.g., a phone number or SSN without punctuation) and special characters can also be

interpreted as string values. In this section, we define syntactic patterns, our main representation for string values. We then define the problem of unsupervised data transformation and give an overview of our approach.

4.1.1 Syntactic Patterns

A syntactic pattern represents the syntactic structure in a string value. In data transformation, Jin et al. [39] find that groups of adjacent characters from the same character class (e.g., digit, alphabet, alpha-numeric) usually share the same role in the transformation program. These groups of characters also have their meanings. For example, in the telephone number "(213) 775-2123," the three-digit "213" should be grouped and transformed together since the digits represent an area code, which is a meaningful concept. We call these character groups *tokens*.

A *token* is a constrained regular expression that contains a regex type and a quantifier to indicate the length of the token. Table 4.1 shows the list of regex types supported in UDATA. In addition, we support different types of punctuation and symbols (e.g., ".", ";", "/"), and each forms a separate regex type of its own. For example, a question mark "?" has its own regex type "?". A quantifier can be either a natural number or "+", which indicates that the length of the token is greater than one.

Example 4.1 *A token of 3 digits is represented as "$\langle D3 \rangle$." A token of 1 alphanum character is shown as "$\langle AD \rangle$."*

A *syntactic pattern* is a sequence of n tokens $\mathcal{S} = \{T_1, T_2, ..., T_n\}$ representing a string value's syntactic structure. A token T is denoted as "$\langle rq \rangle$," where r is the regex type, and q is its quantifier. If the token length is one, we can remove the quantifier to simplify the representation. A set of

Table 4.1: Supported regex types

Regex Types	Regex	Symbol
Uppercase	r_U = [A-Z]+	U
Lowercase	r_l = [a-z]+	L
Alphabet	r_a = [A-Za-z]+	A
Digit	r_0 = [0-9]+	D
Whitespace	r_{ws} = \s+	(S)
Alphanum	r_{a0} = [A-Za-z0-9]+	AD
Alnumspace	r_{a0} = [A-Za-z0-9\s]+	ADS

string values can be described by many syntactic patterns since some token types are supersets of other token types. For instance, Alphabet is the superset of Uppercase and Lowercase. Therefore, determining the most suitable pattern for a set of string values is also a challenge that affects our transformation.

Example 4.2 *"12/11/2017" can fit into several patterns such as "$\langle D2\rangle/\langle D2\rangle/\langle D4\rangle$" or "$\langle D+\rangle/\langle D+\rangle/\langle D+\rangle$"*

4.1.2 Data Transformation Problem

Based on the aforementioned definitions, the nonparallel data transformation problem can be defined as follows:

Definition 4.1 (Data Transformation) *Given a set of n strings to be transformed $\mathcal{S} = \{s_1, s_2, s_3, ..., s_n\}$ represented in m different patterns $\mathcal{P} = \{p_1, p_2, p_3, ..., p_m\}$ and a set of target examples $\mathcal{T} = \{t_1, t_2, t_3, ..., t_n\}$ represented in k target patterns $\mathcal{P}' = \{p'_1, p'_2, p'_3, ..., p'_k\}$, generate a transformation program $\mathcal{L}$ to transform each string s_i in $\mathcal{S}$ to its corresponding string in any of the target patterns contained in $\mathcal{P}'$.*

Our problem contains two sets of input data: the set of data to be transformed $\mathcal{S}$ and the set of target patterns $\mathcal{P}'$. Both $\mathcal{S}$ and $\mathcal{P}'$ are meant to be extracted from a set of string values.

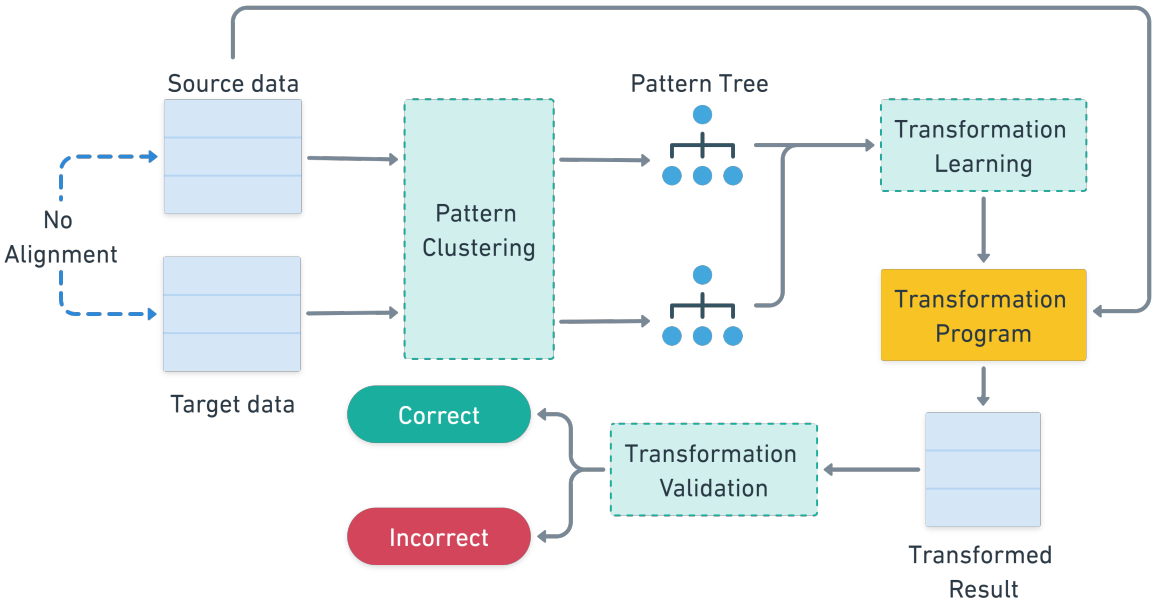


Figure 4.2: Overall workflow of UDATA system

The reason for supporting multiple target patterns is that it is difficult to represent semantically similar data in one pattern. A typical example is that of people's names. Full names with two, three, or even more words require different patterns to represent them.

4.1.3 Overall Approach

We propose a data transformation approach with three phases: clustering, transformation, and validation. Figure 4.2 shows the overall model of our approach.

Pattern clustering — First, both input and output strings are processed by the pattern clustering module. In UDATA, we use the pattern clustering model from Jin et al.'s work [39].

Transformation learning — The learning module synthesizes the transformation programs automatically using the syntactic patterns generated from the clustering model. From the input

and output patterns, tokens with similar string values are matched, and string manipulation functions can be applied when necessary. Since there are multiple source and target patterns, pattern matching must be done after the token matching. We explain our transformation learning model in Section 4.3.

Transformation validation — The validation module verifies the transformation output and then identifies transformation failures for users' curation. Details of our validation algorithm can be found in Section 4.4.

4.2 Pattern clustering

In the work of Jin et al. [39], the pattern clustering model is separated into two different phases: tokenization and agglomerative clustering.

Tokenization — In the tokenization phase, string values are first tokenized into tokens following the rules below:

- Special characters including punctuation mark (e.g., ";" "." "," ...) and symbols (e.g., "@", "$") are identified separately, and each type of punctuation mark is a token type on its own.
- Tokens are described using the most specific token types. For example, for "paper," we choose `Lowercase` type over `Alphabet`, `Alphanum` or `Alnumspace` as its token type.
- Token quantifiers are always natural numbers.

In order to perform the tokenization, the regular expression (regex) of each token type is applied to the string values. For each regex match, a token is created together with the match length.

Example 4.3 *The string "Dec 18th, 2018" is tokenized into: "$\langle U \rangle$" "$\langle L2 \rangle$" "$\langle S \rangle$" "$\langle D2 \rangle$" "$\langle L2 \rangle$" "," "$\langle S \rangle$" "$\langle D4 \rangle$"*

Agglomerative clustering — After tokenization, for each string value we obtain a sequence of tokens and their numeric quantifiers. These sequences are our string syntactic patterns at the lowest level.

Example 4.4 *The lowest level pattern of "Alice1811@gmail .com" is "$\langle U \rangle \langle L4 \rangle \langle D4 \rangle @ \langle L5 \rangle . \langle L3 \rangle$"*

Since the initial patterns are precise, the number of patterns generated is approximately equal to the number of examples. Thus, it makes the problem of transformation learning very computationally expensive. Transformation programs are also usually applicable for a set of strings, instead of one independent string. Therefore, it is necessary to find the common patterns among the string.

Jin et al. [39] proposed an agglomerative clustering algorithm to solve the problem. In their model the patterns generated from tokenization are leaves of the pattern tree. The patterns are then generalized to create their parent patterns.

Definition 4.2 *A parent pattern of pattern p is the lowest level pattern that can cover all of the strings covered by p yet is not p.*

Example 4.5 *The parent pattern of "$\langle U \rangle \langle L4 \rangle \langle D4 \rangle @ \langle L5 \rangle . \langle L3 \rangle$" is "$\langle U+ \rangle \langle L+ \rangle \langle D+ \rangle @ \langle L+ \rangle . \langle L+ \rangle$" and the parent pattern of "$\langle U+ \rangle \langle L+ \rangle \langle D+ \rangle @ \langle L+ \rangle . \langle L+ \rangle$" is "$\langle A+ \rangle \langle D+ \rangle @ \langle A+ \rangle . \langle A+ \rangle$"*

The generalization algorithm runs recursively through the list of existing patterns. Patterns that have the same parent are clustered together, and the algorithm continues until there is no

parent to be generated. The final result of agglomerative clustering is called a *pattern tree* in this paper and all of the patterns with same depth in the tree form a *pattern level*.

Example 4.6 *An example pattern tree built from four different string values is shown in Figure 4.3.*

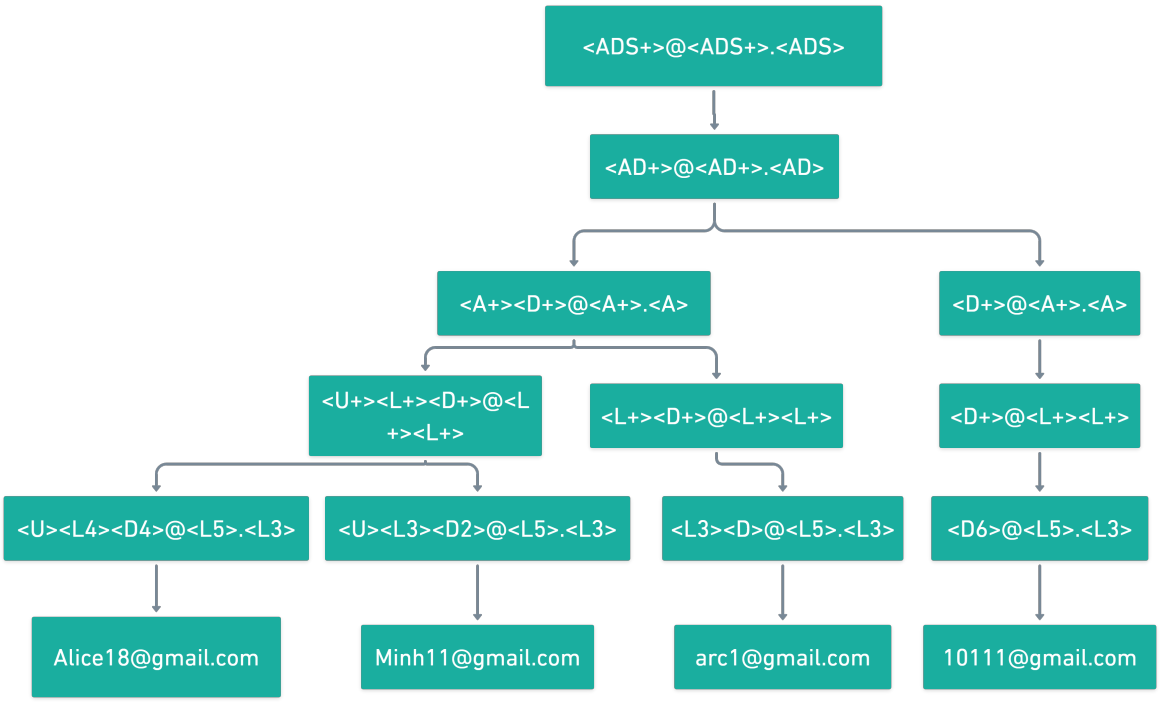


Figure 4.3: Visualization of pattern tree in Example 7

4.3 Transformation Learning

In this section, we describe our overall transformation program structure and the transformation-learning algorithm.

4.3.1 Transformation Program

In order to model the transformation program, we use a hierarchical structure, as shown in Figure 4.4. The top level in the program $\mathcal{P}$ is a pattern-to-function mapping $\mathcal{M}$. $\mathcal{M}$ maps a specific

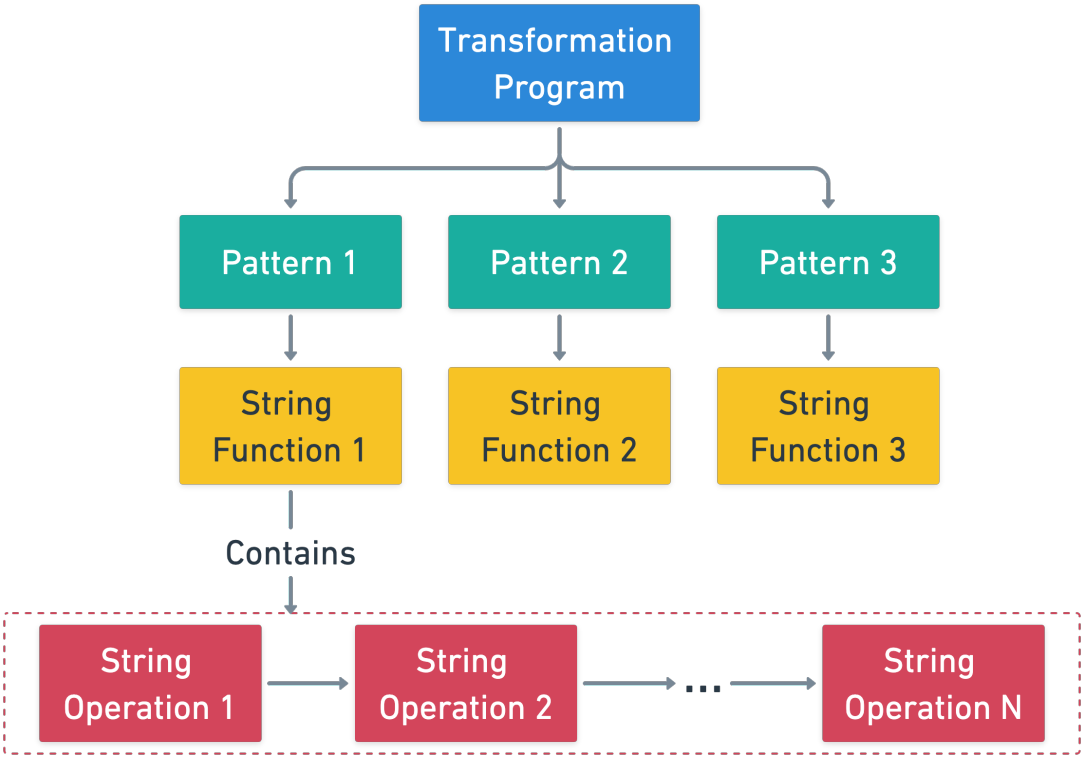


Figure 4.4: Hierarchical structure of transformation programs

original pattern p to a string function F, which can transform p to the target format. A transformation function F is a sequence of string operations O, where each operation can apply to one token in the source pattern.

String operations — A string operation O transforms the source token t to the target token $O(t)$ and has the form of $O = Op(t, a_1, a_2, ..., a_n)$. Op is the operation name, t is the source token, and a_1 to a_n are the required arguments. UDATA currently supports five string operations:

- `ConstStr`$(None, s)$ is a special operation that reuires no source token and returns the constant s.
- `Keep`(t) returns string values of token t.
- `ToLower`(t) returns the lowercase of values in token T.

- `ToUpper`(t'): returns the uppercase of values in token T.
- `Substr`(t, s, e) returns the substrings in range $(s..e)$ of values in token T. Negative values of s and e refer to backward direction.

Table 4.2 provides example outputs of four string operations (all except `Constant`). In the remainder of the chapter, we will refer to the token at position i as t_i

4.3.2 Transformation Program Synthesis

We use a bottom-up approach to learn the transformation program. The input of our transformation algorithm is the source pattern tree T and the target pattern tree T'. First, the algorithm iterates through all the levels in the pattern trees. At each level, we find the best transformation function between each source pattern p and all target patterns p' (Section 4.3.3). The mapping in each level is built upon the source patterns and their corresponding transformation functions. Each level is then ranked by the average score of its transformation functions, and the mapping from the best level is chosen as the final transformation program (Section 4.3.4).

Table 4.2: Output of atomic string operations

Operation	Output
`ToUpper`(“Messi”)	MESSI
`ToLower`(“Messi”)	messi
`Substr`(“Messi”, 0, 3)	Mes
`Substr`(“Messi”, -5, -2)	Mes
`Keep`(“Messi”)	Messi

4.3.3 String Function Generation

As we see in Figure 4.4, string functions are the key elements in a transformation program since the whole program structure is built on these functions. A valid string function should be able to transform values in a source pattern p to a target pattern p'. Therefore, a string function F is valid if the number of operations in F is equal to the number of tokens in the target pattern p'. Since the transformed strings must follow the target pattern p', they will have the same number of tokens n, and n operations are necessary to create n tokens.

Example 4.7 *Figure 4.5 shows an example string function that contains four operations:* `Substr`(t_1, *0, 1),* `ConstStr`*(".")*, `ConstStr`*("(space)")*, `Keep`(t_3)*.*

To generate valid string functions for each source pattern, our string function generation method has three steps: candidate generation, operation scoring, and function synthesis. **Candidate generation** — We generate a set of candidate operations for each pair of source and target tokens in the candidate generation step. An operation is a *candidate operation* if the source and target tokens satisfy the predefined conditions of that operation. Conditions of atomic string operations are:

- If the quantifier on source token is "+", the quantifier on the target token needs to be "+" as well.
- `ConstStr`(s) is a valid candidate if all the values in the target token are s.
- `ToUpper`(t)/`ToLower`(t) is a valid candidate if the t's regex type is `Lowercase`/`Uppercase` and the target regex type is `Uppercase`/`Lowercase`.

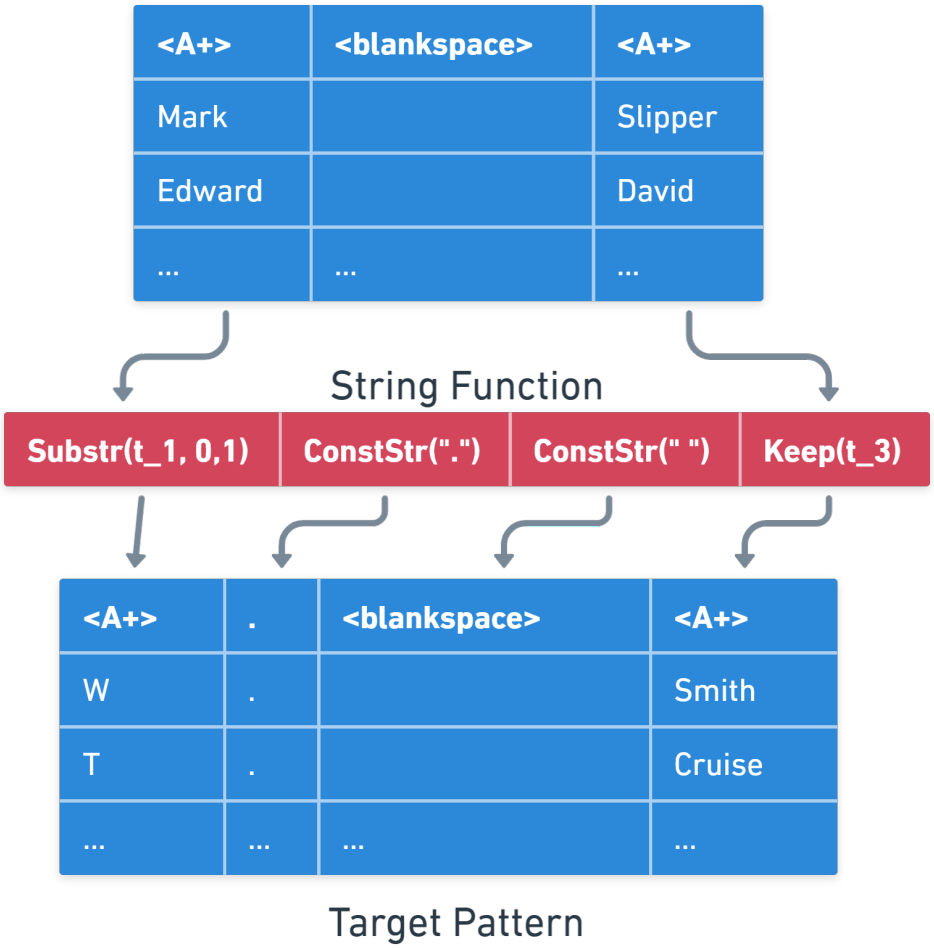


Figure 4.5: String function with source and target patterns

- `Substr`(t, s, e) operations are valid candidates if all strings in the target token have the same length of $|e - s|$ and all string lengths of t are longer than e.

Example 4.8 *Some candidate string functions between the two first tokens in Figure 4.5 are:* `Keep(1)`, `ToUpper(1)`, `SubStr(1, 0, 1)`, `SubStr(1, 1, 2)`, `SubStr(1, -2, -1)`.

Based on the conditions above, UDATA iterates through the list of available operations for each pair of tokens. The operation will be added to the candidate set if the source and target tokens satisfy the operation's condition.

String operation scoring — The generated candidate operations need to be scored and ranked to find the best operation for every source token. Figure 4.6 shows how the scoring works in our system. Our scoring method follows the process of our semantic labeling method, DSL, as described in Chapter 3. First, we apply the string operation to transform the source token. The similarity feature vector is then calculated between the transformed token and the target token. The feature vector $\mathfrak{f}(O(v_t), v_{t'})$ consists of two different feature types:

- Semantic metrics: illustrate the overlap between values in two different subsets. Higher similarity values indicate that they are more likely to contain the same specific parts of information, such as months, years, first names, or last names. We use Jaccard and tokenized Jaccard similarity as our features.
- Syntactic metrics: denote the degree of similarity in syntactic structures between the transformed source data and the target data. We use Jaccard similarity over the set of regex types between two patterns as our syntactic metrics.

The score of a string function F to transform a source token t to a target token t' is the similarity score between the transformed results $O(v_t)$ and the target values $v_{t'}$ and can be calculated using DSL model as follows:

$$\begin{aligned} \text{score}(O_{t,t'}) &= \text{sim}\left(O(v_t), v_{t'}\right) \\ &= \text{sigmoid}\left(w^T \mathfrak{f}\left(O(v_t), v_{t'}\right)\right) \end{aligned} \tag{4.1}$$

The score of an operation `ConstStr`(s) is 1 because the transformed values, which are all s, are the same as the values in the target token.

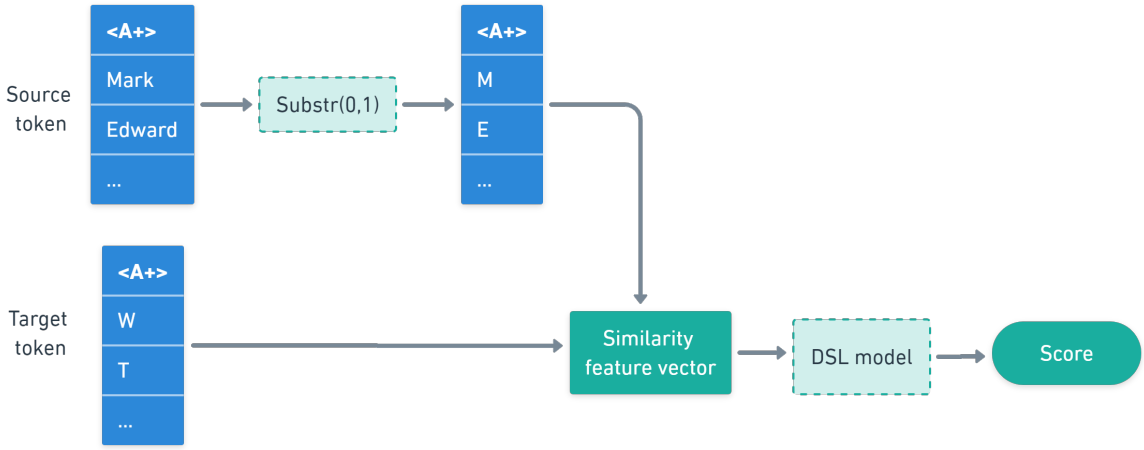


Figure 4.6: String operation scoring

The training data for our logistic regression model is generated exclusively from the source and target data given as the problem input. For each token in the source and target patterns, we split their values into three equal sets, which can be considered training sets. Training samples are created by randomly sampling a pair of string sets. Using the assumption that there is no pair of tokens in the same pattern that contain the same piece of information, if the pair of string sets come from the same token, the training label is True and vice versa. After training, the learning model can be used for string function scoring, as shown in Equation 4.1.

String function synthesis — After scoring all the candidate operations between source and target tokens, we create a score matrix. The matrix rows indicate tokens in the source pattern, the matrix columns denote tokens in the target pattern, and the matrix values are tuples of the best operations and their scores. `ConstStr` operations are excluded for simplification since they do not affect the problem of finding the optimal mapping. Table 4.3 shows the score matrix generated from Figure 4.5.

Table 4.3: Score and operation matrix from Figure 4.5

	$\langle A+\rangle(1^{st}$ token)	$\langle A+\rangle(4^{th}$ token)
$\langle A+\rangle(1^{st}$ token)	0.11 `Substr`(1, 0, 1)	0.07 `Substr`(3, 0, 1)
$\langle A+\rangle(3^{rd}$ token)	0.07 `Keep`(1)	0.12 `Keep`(2)

Table 4.4: Score and string function matrix

	p'_1	p'_2
p_1	0.17 {`Substr`(1,0,1)},`Keep`(2)	0.03 {`Substr`(2,0,1)}
p_2	0.22 {`Keep`(1), `Keep`(2)}	0.08 {`Keep`(2)}

Finding the best alignment between tokens of the source and target patterns is a maximum assignment problem and can be solved using the Hungarian algorithm [47]. After determining the best mappings between source and target tokens and the corresponding string functions, the string function F is formed as a sequence of operations ordered by target token positions. For example, the final string function in Table 4.3 after including `ConstStr` operations is the string function shown in Figure 4.5.

4.3.4 Pattern Mapping

Same-level pattern mapping — In every pattern level, UDATA generates the best string functions between two patterns based on results from previous steps. The input of UDATA's pattern mapping module is also a mapping between a pair of source and target patterns to the corresponding string functions. An example of pattern mapping is shown in Table 4.4.

As we see from Table 4.4, each source pattern can be mapped to different target patterns using different string functions. UDATA selects the best string function for each source pattern

to build the final pattern-to-function mappings. For instance, in Table 4.4, both p_1 and p_2 should be matched to p'_1 using the corresponding string functions.

Pattern-level ranking — Since the pattern mappings are restricted to only one pattern level, it is necessary to find, score, and rank the transformation program among the pattern levels to select the final transformation program. In this step, we apply the same idea from our string function scoring to score the pattern level. In our pattern-level scoring model, the mapping score between two pattern levels is the predicted similarity score between the transformed and target data. The transformed data can be obtained by applying the string function F of every pattern p to the set of source strings that fit in p. The pattern level with the highest score is chosen to build the transformation program.

4.3.5 Scalable Transformation

As the volume of data increases, the computational cost of our system increases because of the complexity of our pattern clustering and transformation learning modules. Therefore, we develop an adaptive transformation approach to improve the scalability of our system.

Our adaptive transformation works as follows:

- First, sample a subset A from the set of input strings $\mathcal{S}$.
- Run UDATA on A to learn the program P.
- Apply P to the rest of the data and create a subset S' of S for the values that P cannot transform.
- Repeat the process with S' as the input set of strings.

- Iterate until all of the strings in S are transformed.

The adaptive transformation method introduces a sample-build-test cycle that prevents repeatedly building pattern trees and mapping the patterns. Therefore, it prevents the computation cost of the UDATA from increasing exponentially.

4.4 Transformation validation

Using unsupervised techniques to transform data comes with the usual caveat that mistakes will be overlooked. To verify the accuracy of the transformation system for real-world applications, we implement a validation algorithm that can alert a user as to whether the transformation program is likely to produce correct results. In an application where a handful of mistransformed data elements could lead to catastrophic loss, UDATA can still be used as the first module of the pipeline to reduce the workload for users in later stages. There are two main reasons for transformation failures: syntactic and semantic mismatches, and we propose an approach that can identify these types of failures separately.

Syntactic mismatches — The syntactic validation algorithm builds the pattern tree from our transformed data and compares it with the target pattern tree. If the transformed pattern tree is a subtree of the target pattern tree, which means they have similar syntactic structures, we can conclude that our transformation program is syntactically correct.

This method captures mistakes in syntactic structure and identifies structural mismatches between transformed strings and desired string values. However, it cannot provide perfect accuracy

Table 4.5: Semantic ambiguity in the transformation

Source	Target
308-916-545	504
623-599-749	843
118-980-214	749

since semantic ambiguities also cause errors. In real-world problems, there are cases where finding the similarity between different sets of string values is ambiguous, causing our transformation learning to fail.

Semantic mismatches — To solve the issue above, we developed a semantic validation method that can report potential failures due to semantic mappings. The main reason for semantic mismatches in our system stems from the scoring model. Although we provide similarity features with different aspects to cover the semantic meanings contained in the string values, in many cases, the amount of data is insufficient to distinguish between different tokens.

Example 4.9 *In Table 4.5, the source data can be represented as "$\langle D3 \rangle$-$\langle D3 \rangle$-$\langle D3 \rangle$" and the target data can be represented as "$\langle D3 \rangle$". The transformation task is to extract the last three digits from a nine-digit phone number.*

As we see from Table 4.5, there is no overlap between the target data with any three-digit chunk in the source data. Moreover, the target data is syntactically similar to all the three-digit chunks in the source data. Therefore, there is not enough discrepancy between the source tokens for the system to infer the correct token mapping. Thus, the semantic validation module reports to users if top transformation programs have comparable scores. On the other hand, we also report scenarios where the highest similarity scores are zero. Our classification finds no overlap

in semantic and syntactic features between the target and source data in these cases, and thus the transformation programs cannot be learned.

4.5 Related Work

Data transformation — PBE systems are often used because they can automatically perform transformations given a small number of output samples. FlashFill [29] designed a string expression language and a set of transformation programs to learn the transformations based on input-output pair examples. Wu et al. [92, 93] extended FlashFill by providing a partitioning method to partition string values as well as introducing an incremental process to eliminate incorrect transformation programs while users are inputting the examples. Recently, Singh [80] and Jin [39] proposed a semi-supervised method to build syntactic patterns of input data to support transformation learning. Another approach for data transformation that has also received attention in recent years is machine learning. Shu [78] proposed a method that uses deep neural networks to learn the transformation programs. Wang et al. [91] used a probabilistic approach to model and learn the string transformations, while Devlin et al. [21] take advantage of deep learning models. Since both PBE and machine learning systems take advantage of a small set of labeled data combined with a large amount of unlabeled data, these systems can be considered semi-supervised systems, compared to our unsupervised system.

Another direction of data transformation focuses on transforming strings based on their semantic meaning. Singh et al. [81] learn the semantic transformations by using the information available within a given data source. DataXFormer [3] learns the semantic transformations from the set of web tables and forms available on the web. Singh et al. [83] learned semantic

transformations of different data types based on a set of operations predefined by developers. To date, our system cannot learn semantic transformations, but it is one of the goals for future work.

Data cleaning — Interactive data cleaning is another common approach since it can analyze the data and provide useful tools to create transform rules faster. Raman et al. [73] proposed an interactive data cleaning system that analyzes data and suggests transformation rules for users to select. Wrangler [41] provides a user-friendly interface for creating scripts that can be used to clean and transform data. ActiveClean [46] is another interactive data cleaning system that combines adaptive and rule-based models to suggest dirty data for users' curation based on previous dirty and cleaned data. Although interactive cleaning systems reduce users' time and effort to create the transformations, they do not address the problem of human interaction, as the UDATA system does.

Data profiling — There is also research in the field of data profiling. Saswat et al. [65] implemented FLASHPROFILER, which is a syntactic profiling system that discovers syntactic patterns from a set of string values. Andrew et al. [38] proposed a method that extracts syntactic patterns from databases in three different layers: branch, token, and symbol layers. LEARNPADS [23] proposed a learning algorithm using statistics over symbols and tokenized data chunks to discover pattern structure. However, these systems assume that attribute values follow the same pattern structure, which is hard to ensure for noisy datasets collected from multiple sources. In addition, there is other work focusing on analyzing and profiling data attributes (i.e., columns) from relational databases [15, 33], RDF triple stores [2, 5], and XML files [7].

4.6 Evaluation

In this section, we evaluate the transformation and validation methods in UDATA. Our system, datasets, and results are available online.[*]

4.6.1 Experimental Setup

Baseline systems — In the transformation evaluation, we compare UDATA performance with two state-of-the-art PBE systems: IPBE [92] and FLASHFILL [29]. Since IPBE and FLASHFILL can leverage parallel input-output examples, they serve as the upper-bound systems in our evaluation.

Datasets — We use five different datasets in the evaluation. Details of the five datasets are shown below:

- AAC: contains 173 transformation problems collected from 14 American art museums [45].
- SYGUS: contains 27 transformation problems from the Syntax-guided Synthesis Competition over the years [39].
- PREGPROG: contains six transformation problems from previous papers on program synthesis [82].
- IJCAI: contains 36 transformation problems synthesized by Wu et al. [93].
- NYC: contains five transformations that we collected and labeled from NYC Open Data.[†]

Data organization — In our transformation accuracy experiments, we only use the first 1000 examples for evaluation since PBE systems are not designed to handle extensive datasets. Since

[*]https://github.com/minhptx/ieee-bigdata2019-transformation
[†]https://opendata.cityofnewyork.us/

Table 4.6: Performance of transformation systems

System	AAC	IJCAI	Sygus	Prog	NYC	Mean
IPBE	0.99	0.83	0.93	0.99	0.97	0.94
FlashFill	0.91	0.62	0.88	0.99	0.90	0.86
UData	0.93	0.71	0.56	0.93	0.88	0.80

all the data in our datasets are parallel, we need to organize the data so that the three systems can use them properly. For each scenario, the parallel examples are divided into three partitions: $P1$, $P2$, and $P3$. For UData, we use $P1$ and $P2$ as our data to be transformed and $P3$ as example data in the target format. The output examples in $P1$ and $P2$ and the input examples in $P3$ are removed from our system. For FlashFill, the input-output examples of $P2$ are provided as parallel data, while input examples of $P1$ serve as the transforming data. In IPBE, we limit the set of active learning examples to be the same size as $P3$.

4.6.2 Transformation Result

In the transformation experiment, we evaluate the transformation accuracy of UData, IPBE, and FlashFill. The transformation accuracy of each scenario is reported as the ratio between the number of correctly transformed values and the size of the input data.

As we see in Table 4.6, UData's accuracy is about 10% lower than IPBE and comparable with FlashFill in AAC, NYC, and PregProg datasets. These three datasets are the real-world datasets in our evaluation data. Real-world scenarios usually contain little semantic ambiguity in their tasks since data are stored in a way that is easy for humans to understand. There are cases where real-world data contains multiple syntactic patterns due to data entry errors or conflicts in data specifications. However, the number of syntactic patterns is usually low. These two reasons make UData perform better on real-world datasets.

In the Sygus and IJCAI datasets, there is a bigger difference between the performance of other systems and UDATA. The main reason for our low accuracy is that both the Sygus and IJCAI datasets are synthesized for supervised program synthesis evaluation. The IJCAI and Sygus datasets contain complicated transformation scenarios with many syntactic patterns in the input data. For example, there are ten scenarios in Sygus related to extracting or reordering phone numbers (Example 4.9). Not only are the digits in phone numbers distributed randomly, but there is almost no overlap between sets of three digits in different phone numbers. Therefore, our string function scoring models cannot determine the mapping between source and target tokens. On the other hand, supervised systems such as IPBE and FLASHFILL have full information about the alignments and can handle these scenarios better.

To summarize, we see that UDATA has comparable performance on real-world datasets with state-of-the-art systems but has difficulty with performance on synthesized scenarios.

4.6.3 Validation Result

We evaluate our validation method and report the failure recall on the same datasets. We consider a transformation is successful if all the string values are transformed correctly. Otherwise, the scenario is a failure. The reason for this strict condition is that even if only one string value is transformed incorrectly, user interaction is required to make corrections; thus, such failures should be reported to the users. As shown in Table 4.7, the UDATA system achieves a high failure recall of 0.99 across different datasets, which guarantees that UDATA can identify almost every mistranformed output value for later curation.

Table 4.7: Validation recall of UDATA

AAC	IJCAI	Sygus	Prog	NYC	Mean
1.0	1.0	0.95	1.0	1.0	0.99

It is clear from the transformation evaluation that UDATA still has a performance gap compared with other supervised systems. However, by allowing fully unsupervised transformations, UDATA provides a tradeoff between the need for training data and higher accuracy. With a transformation accuracy of 0.80 and a validation recall of 0.99, in normal transformation applications, UDATA can be used as the first stage in a pipeline to transform a large portion of the data and reduce the workload for users in later phases.

4.6.4 Running Time

We report the running time of the UDATA, IPBE, and FLASHFILL in Table 4.8. As we see from the table, UDATA learns the transformations within two seconds in simple datasets. In more complicated ones, such as IJCAI and Sygus, the source input data can have a high number of source patterns in some scenarios, which increases the generated pattern tree's size and affects the system's performance. IPBE uses an adaptive active learning method to iteratively learn and suggest ideal examples for annotation, which is time-consuming and results in its highest running time in the evaluation. FLASHFILL does not have a built-in pattern clustering module, and the transformation program must be learned from all input/output examples at once. As a result, FLASHFILL achieves lower performance in datasets with various formats but has the lowest running time across all the systems.

Table 4.8: Average running time of UDATA

System	AAC	IJCAI	Sygus	Prog	NYC	Mean
IPBE	3.9s	36.8s	1.2s	4.2s	36s	16.4s
FLASHFILL	2.9s	1.6s	1.5s	1.4s	1.3s	1.7s
UDATA	7.6s	17.4s	0.6s	0.4s	1.8s	5.6s

4.7 Summary

In this chapter, we presented a novel unsupervised approach to infer expressive syntactic structures from data, learn the transformation between source and target data using these syntactic structures, and validate the transformed results. Without human interaction, our system can transform attribute data from different formats into one desired format. Therefore, our system can be applied to correct syntactic data in tables. For example, we have used UDATA for automatically correcting format inconsistencies in the problem of automatic spatial and temporal indexing from many data sources [44].

Chapter 5

Semantic Error Detection and Correction

A major challenge in detecting semantic errors lies in requiring external knowledge sources to validate the tables' values. Structured knowledge sources such as Wikidata have been used to extract the essential information for detecting semantic errors [16]. However, structured data sources require extensive manual effort to build and maintain, and thus they contain quite limited information compared to unstructured data. Table 5.1 shows a Wikipedia table about "Iain Glen," an actor. By referring to the "Iain Glen" entity in Wikidata, the current most extensive knowledge graph, we still cannot verify the information specified in the table since there are not enough details on the Wikidata page. To address the limited scope of data, we explore the open-domain web text as our source for evidential information in this thesis. Textual data have been used to retrieve information for relevant research problems such as fact verification [63, 87, 94] and open-domain question answering [26, 48].

Unlike structural knowledge bases that are constructed through more expensive knowledge acquisition and curation processes in a close domain, free text corpora represent a richer resource to cover information and details that structured data do not cover. For example, by checking the Wikipedia page of "Iain Glen", we can verify the information in Table 5.1 since it is written as

Table 5.1: Table in Wikipedia about "Iain Glen"'s awards

Year	Award	Category	Work	Result
1990	Silver Bear	Best Actor	Silent Scream	Won

"Other notable roles include [...] Larry Winters in Silent Scream (1990) for which he won the Silver Bear for Best Actor from the Berlin International Film Festival."

In constrast to structured data, where relationships and properties of instances are well-defined, relationships between different entities embedded in text are more implicit and difficult to recognize. Moreover, using textual information to validate tabular data requires additional processing steps for the two heterogeneous forms of data, textual and tabular data, to work in the same model. However, significant progress has been made in the field of natural language processing (NLP) in recent years and using textual data as knowledge sources allows us to leverage pre-trained NLP models and machine learning architectures such as BERT [52], RoBERTa or BART [50]. Tables can be converted into sequence forms by leveraging table linearization [14], and pre-trained NLP models can be explored to solve table-to-text problems such as semantic error detection.

In this chapter, we present a novel approach for semantic error detection using textual evidence called SEED (SEmantic Error Detector). SEED introduces an open-domain semantic error detection process that includes three main modules: document retrieval, sentence selection and table verification, similar to fact verification systems. In the document retrieval phase, the system selects a set of relevant documents using keyword search and Dense Phrase Retrieval (DPR) reranking. In sentence selection module, all relevant documents are segmented into sentences. We then leverage table linearization and BART [50] models to train a classification model to select evidential sentences. The same neural architecture is also used in the table verification step

to predict the correctness of table cell values. To obtain training data for all three modules, we leverage the ToTTo dataset [69] as our positive examples and perform multiple data augmentation strategies to generate negative examples.

The contribution of this work is an end-to-end open-domain semantic error detection and correction approach for tables that contains: (i) multiple strategies to obtain positive and negative training examples from available datasets in relevant problems and (ii) a novel method that leverages table linearization and existing NLP neural architecture to solve table-to-text problems (NLI).

5.1 Preliminaries

In this section, we first define the problem of semantic error detection and correction. Secondly, we introduce ToTTo and explain how we use ToTTo as our training dataset. Then, we also present our linearization approach, which follows the ToTTo preprocessing method.

5.1.1 Problem Definition

Tables collected from the web have different layouts and thus store data differently. Braunschweig [9] categorizes tables into horizontal tables, where the entities are represented in rows, and vertical tables, where entities are described in columns. In this chapter, we target horizontal tables and assume that rows in tables refer to instances and columns in tables refer to different attributes of the instance. Table understanding approaches [9, 10, 84] can be used to identify the layout and orientation of the tables before tables can be converted to the necessary formats.

Since rows in horizontal tables refer to multiple attributes of the same entity, we focus on identifying the erroneous rows in the thesis. The problem of entity-based semantic error detection can be defined as follows:

Definition 5.1 (Row-based Semantic Error Detection and Correction) *Suppose $\mathcal{T}$ denotes a table with m rows and n columns, where c_{ij} denotes the table cell in row i and column j. For each row r_i in $\mathcal{T}$,*

- *Error Detection: predict 1 if r_i is correct and 0 if r_i is incorrect.*
- *Error Correction: if r_i is incorrect, provide the corrected value r'_i*

Since we are using a text corpus as the reference source in this thesis, the problem can be modified as follows:

Definition 5.2 (Row-based Semantic Error Detection using Textual Evidence) *Suppose $P_i \in \mathcal{P}$ denotes an individual text document and $\mathcal{P} = \{P_1, P_2, ...\}$ is the text corpus. P_i contains a set of sentences $\{s_i^1, s_i^2, ...\}$. Suppose T denotes a table of n rows $\{r_1, r_2, ..., r_n\}$. The problem's input is the table $\mathcal{T}$ and $\bigcup P_i$, the union of all documents available in the text corpus. The output of the problem is*

- *Error Detection: a list $\mathcal{R}$ of size n where $\mathcal{R}_i$ indicates if a table row r_i is correct/incorrect.*
- *Error Correction: a list $\mathcal{R}'$ of size n where $\mathcal{R}'_i$ denotes the corrected version of table row r_i*

The main focus of this chapter is semantic error detection. The semantic error correction approach is presented in Section 5.2.4. The semantic error detection problem can be considered an extension of the fact verification problem where table cell values are verified instead of textual

claims. Semantic error detection is also closely related to the *table-based fact verification* problem proposed by Chen et al. [14]. In table-based verification, tables are used as the knowledge source to verify textual information, which is the opposite of semantic error detection, where text corpus is the knowledge source.

5.1.2 ToTTo dataset

ToTTo is a dataset published by Google* for the controlled table-to-text task and contains more than 120,000 training examples divided into train, dev and test sets. To create the dataset, tables from English Wikipedia are collected and matched with description sentences in the same articles. The original sentences may have been rewritten to correct the grammar.

An example of a ToTTo instance is shown in Figure 5.1. Each sample in ToTTo dataset contains a Wikipedia table, its corresponding description sentence, its metadata, and a set of highlighted cells. The table's metadata includes the page title, section title, and section text. The highlighted cells are a set of (row_index, column_index) tuples where each tuple specifies a pair of row and column indices. Each indices tuple refers to a table cell mentioned in the description sentence. For example, in Figure 5.1, the yellow cells are the highlighted cells corresponding to the given sentence.

The task designed for ToTTo can be defined formally as:

Definition 5.3 *Given a Wikipedia table $\mathcal{T}$ and a set of highlighted table cells $\mathcal{C} = \{(r_1, c_1), (r_2, c_2), ..., (r_n, c_n)\}$, produce a sentence s that describes these highlighted cells.*

This has a corresponding classification task:

*https://github.com/google-research-datasets/ToTTo

Tee Grizzley

Section Title: Awards and nominations
Table Section Text: *None*

Year	Award	Category	Work	Result
2017	2017 BET Hip Hop Awards	Best New Hip Hop Artist	Himself	Nominated
	2017 BET Hip Hop Awards	Best Mixtape	My Moment	Nominated
2018	MTV Video Music Awards	Push Artist of the Year	Himself	Nominated

Original sentence(s)

He received two 2017 BET Hip Hop Award nominations for Best New Hip-Hop Artist and Best Mixtape for My Moment.

Final sentence(s)

Tee Grizzley received two 2017 BET Hip Hop Award nominations for Best New Hip-Hop Artist, and for Best Mixtape for My Moment.

Figure 5.1: An example in ToTTo dataset

Definition 5.4 *Given a table $\mathcal{T}$, a set of highlighted table cells $\mathcal{C}$, and a sentence s, predict if sentence s is the description of the highlighted cells in table $\mathcal{T}$.*

The input and output of the above classification task are equivalent to the semantic error detection problem. Therefore, we use ToTTo as our training dataset in this method. As ToTTo involves manual curation to correct the sentences describing the cells while we require the sentences to be original, we remove noisy examples from the datasets as follows:

- Remove tables with no highlighted cells
- Remove tables with all numeric highlighted cells
- Remove tables with all numeric headers
- Remove tables where highlighted cells span across more than ten columns (95^{th}-percentile)
- Remove tables where highlighted cells span across more than three rows (95^{th}-percentile)
- Remove tables where the description sentences do not have a verb.

Table 5.2: Statistics of cleaning criteria for ToTTo

Criterion	Percentage
No highlighted cells	10.3%
All empty cells	0.06 %
All numeric highlighted cells	26.1%
All numeric headers	10.7%
Highlighted cells in more than ten columns	0.02%
Highlighted cells in more than three rows	2.8%
Description sentences with no verb	11.4 %
Tables in articles with “List” in the titles	18.0%

- Remove tables in articles with “List” in the titles.

Table 5.2 shows the statistics of each criterion when applying on ToTTo. Tables with empty highlighted cells and empty table cells are removed as empty training examples. We remove Wikipedia tables with numeric values and headers since they are mostly statistical and ranking tables and usually contain little factual details. Tables with highlighted cells that span many columns and cells are usually matched with summary sentences and are not in the scope of this work. Sentences without verbs need to be entirely rewritten by curators; thus, the original sentences from Wikipedia cannot be used for inference. Also, 47% of the articles with “List"" in titles contain no useful textual information and should be removed.

5.1.3 Wikipedia

Wikipedia is the largest online encyclopedia, and its English version contains more than six million articles. As ToTTo is being used as our training and evaluation data and its tables and sentences are extracted from Wikipedia, we use Wikipedia as the text corpus for factual reference in this thesis.

5.1.4 Table Linearization

As our framework involves many natural language models, we linearize tables into sequences so they can be fed directly into the model. In this work, we use the linearization method provided by ToTTo. By following a predefined template, a flattened table T with m rows $\{r_1, r_2, ..., r_m\}$, n columns $\{c_1, c_2, ..., c_n\}$ and $m \times n$ cells $\{T_{11}, T_{12}, ..., T_{mn}\}$ can be represented in regular expression form as "[table] ([row] r_i [/row])+ [/table]" and a row r_i is represented as r_i = "([cell] [col_header] c_i [/col_header] T_{ij} [/cell])+". Here *[row]*, *[cell]*, *[col_header]*, *[table]* are special tags used to separate between different elements in the tables and *[/table]*, *[/row]*, *[/cell]*, *[/col_header]* are their corresponding end tag. Throughout the chapter, when we refer to tables as an input of a model, we refer to their linearization. For ToTTo dataset, the linearization of the tables is the linearization of the highlighted cells in the tables.

5.2 Approach

As described in Section 5.1.1, our focus is to detect and correct erroneous rows in the table. The problem of table error detection can be divided into subproblems, where each row in the table can be considered a sub-table. For the rest of the chapter, when we refer to tables, we consider them one-row tables.

Figures 5.2a and 5.2b provide an overview of our approach implemented in SEED. Suppose $\mathcal{T}$ denotes a one-row table with m cells $\{c_1, c_2, ..., c_n\}$. Suppose a text corpus $\mathcal{K} = \{d_0, d_1, \dots\}$ where d_i denotes a text document and d_i includes a set of sentences $d_i = \{s_i^0, s_i^1, \dots, s_i^m\}$, the overall process of SEED contains four steps:

- Document retrieval: SEED retrieves a set of relevant documents $\hat{D} \subset \mathcal{K}$ for each table $\mathcal{T}$.

- Sentence selection: SEED extracts the evidential sentences $\hat{E} \subset \hat{D}$ from all sentences in the retrieved documents. If there are no evidential sentences found, the table cannot be verified.

- Table verification: For each evidential sentence $e_i \in \hat{E}$, SEED outputs True/False that indicates if the sentence entails contents of the table. The table is correct if there is any entailed sentence in the corpus. On the other hand, if there are no entailed sentences in the corpus, the table is considered an error.

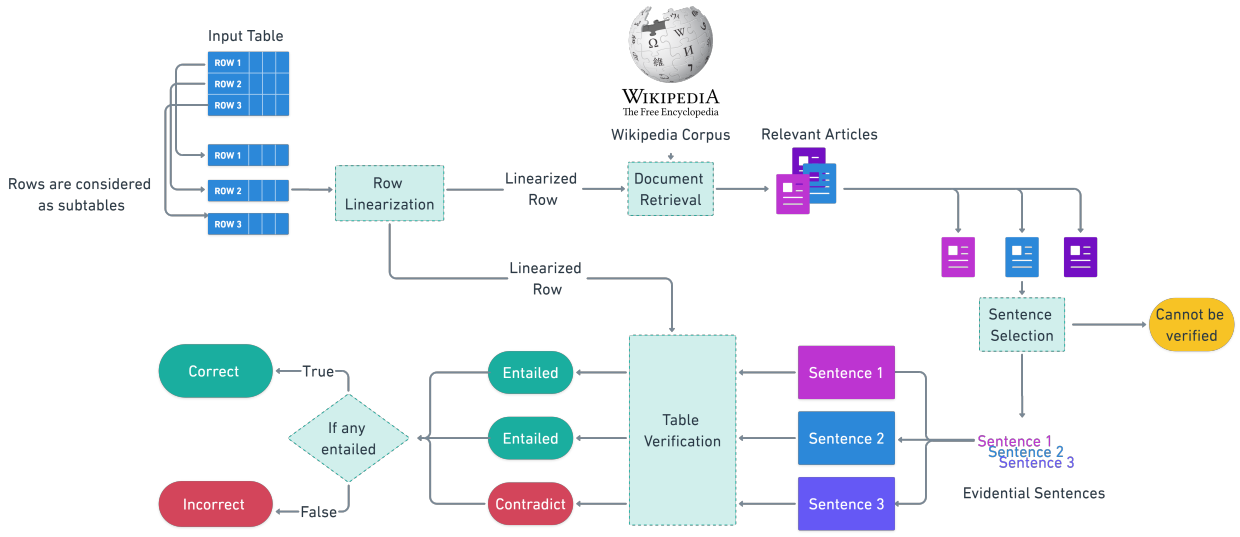
- Error cell correction: For each cell c_i in the erroneous table, we extract the predicted value c_i' for that cell from the context sentence s. A cell is considered an error if the extracted value differs from the cell value.

The details of the document retrieval module are explained in Section 5.2.1. Sentence selection is discussed in Section 5.2.2, while Section 5.2.3 illustrates the table verification step. Finally, Section 5.2.4 shows the error cell correction algorithm.

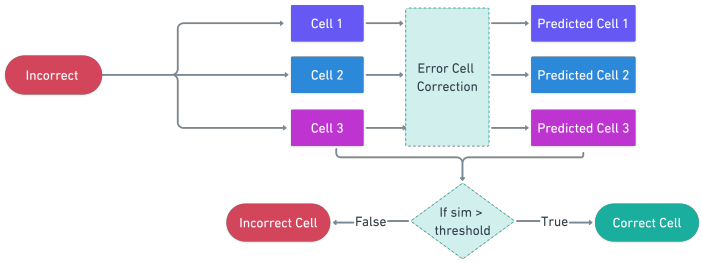
Example 5.1 *Tables 5.3 and 5.4 show two examples of correct and incorrect tables, with their outputs being processed by SEED.*

5.2.1 Document Retrieval

The goal of the document retrieval phase is to retrieve a set of relevant documents $\mathcal{P}'$ for a table $\mathcal{T}$. This module takes a table $\mathcal{T}$ and the text corpus $\mathcal{P}$ as input. It outputs a subset of documents $\mathcal{P}' \in \mathcal{P}$ where $\mathcal{P}'$ consists of relevant documents to the table row r as follows:



(a) Document retrieval, sentence selection and table verification



(b) Error cell correction

Figure 5.2: Overview of SEED

$$f_{DR}(\mathcal{T}, \mathcal{P}) = \mathcal{P}'$$

In this work, we present a hybrid approach that combines the strength of keyword search and Dense Passage Retrieval (DPR) [42], two popular methods for document retrieval.

Figure 5.3 shows the training process of DPR. First, we use the segmented Wikipedia provided in KILT [68] and index the segments in KILT using a traditional Lucene search engine, Pyserini [54]. As shown in Section 5.1.2, a sample in ToTTo is a tuple $\langle T, s, \mathcal{M} \rangle$ where T is the table, s is the description sentence, and $\mathcal{M}$ is the table metadata. For training examples, DPR

<table>
<tr><th>Phase</th><th>Output</th></tr>
<tr><td>Data input</td><td>Title: Kiichi Kunimoto
<table><tr><th>Opponent</th><th>Event</th><th>Date</th></tr><tr><td>Daniel Sarafian</td><td>UFC 174</td><td>June 14, 2014</td></tr></table></td></tr>
<tr><td>Document retrieval</td><td>“Kiichi Kunimoto’
“Daniel Sarafian”
“Neil Magny”</td></tr>
<tr><td>Sentence selection</td><td>“Kunimoto [...] UFC 174 on June 14, 2014, facing Daniel Sarafian.”
“Sarafian faced Kiichi Kunimoto [...] on June 14, 2014, at UFC 174.”</td></tr>
<tr><td>Table verification</td><td>“Kunimoto [...] UFC 174 on June 14, 2014, facing Daniel Sarafian.” ⇒ Entailed
“Sarafian faced Kiichi Kunimoto [...] on June 14, 2014, at UFC 174.” ⇒ Entailed</td></tr>
</table>

Table 5.3: Example of a correct table

requires triplets $\langle q, p^+, p^- \rangle$ of queries, and positive and negative passages. The linearized form of ToTTo table T is used as the query q for DPR training. The positive passages are the KILT passages that contain the sentence s in ToTTo. The negative passages are sampled from the search result when querying T cell values against the KILT knowledge source using BM25 [76].

A DPR model contains two different encoders: a query encoder E_q and a context encoder E_c, two independent BERT models [20] previously trained on five different QA datasets [42]. The queries are encoded by E_q, while E_c encodes the passages. The DPR model calculates the inner product between all encoded queries and all passages. For each query, the score vector is fed to a

<table>
<tr><th>Phase</th><th>Output</th></tr>
<tr><td>Data input</td><td>Title: Sam Underwood
<table>
<tr><th>Years</th><th>Title</th><th>Role</th><th>Location</th></tr>
<tr><td>2010</td><td>Equus</td><td>Alan Strang</td><td>HERE Arts Center</td></tr>
</table></td></tr>
<tr><td>Document retrieval</td><td>“Sam Underwood’
“James Rado”</td></tr>
<tr><td>Sentence selection</td><td>“[...], Underwood was asked to play the part of Alan Strang in a production of “Equus” at the John Drew Theatre [...]”</td></tr>
<tr><td>Table verification</td><td>“[...], Underwood was asked to play the part of Alan Strang in a production of “Equus” at the John Drew Theatre [...]” ⇒ Not entailed</td></tr>
<tr><td>Error cell correction</td><td>Location ⇒ John Drew Theatre</td></tr>
</table>

Table 5.4: Example of an incorrect table

softmax layer, and the negative log-likelihood for the positive passages is the objective function. The overall architecture is shown as follows:

$$E_{p^+} = \text{BERT}_{passage}(p^+)$$

$$E_{p^-} = \text{BERT}_{passage}(p^-)$$

$$E_q = \text{BERT}_{query}(q)$$

$$L = -\log \frac{e^{sim(E, E_{p^+})}}{e^{sim(E_q, E_{p^+})} + e^{sim(E_q, E_{p^-})}}$$

In the document retrieval phase, we retrieve the candidate documents from Wikipedia using the keyword search algorithm BM25 [77]. The queries used in BM25 are the concatenation of all

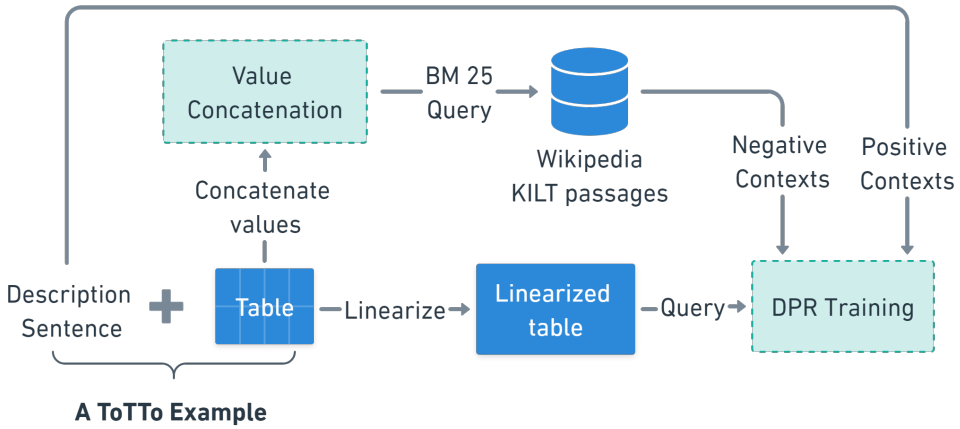


Figure 5.3: DPR training overview

cell values in tables. The candidate documents are then scored and reranked using DPR. We encoded the linearized tables and the candidate documents using DPR encoders. Then DPR assigns a retrieval score for each pair of query Q and passage P using Cosine similarity:

$$Score(Q, P) = CosineSim\Big(\text{BERT}_{query}(Q), \text{BERT}_{passage}(P)\Big)$$

Even though we index Wikipedia using KILT passages, SEED retrieves documents instead of passages. For every retrieved passage, we return the document containing the passage to increase the retrieval recall. Top-k documents with the highest scores are returned as our relevant documents for sentence selection.

5.2.2 Sentence Selection

Sentence selection is the process of extracting relevant sentences from the retrieved documents relevant to a table, formalized as follows:

$$g(\mathcal{T}, \bigcup_{s \in \mathcal{P}'} \mathcal{P}') = \mathcal{S}' \tag{5.1}$$

which takes a table $\mathcal{T}$ and the union of the sentence set of each retrieved document as inputs and outputs a subset of sentences $\mathcal{S}' \subseteq \mathcal{P}'$ as the evidence set. The sentence selection problem is equivalent to binary classification between each sentence $s \in \mathcal{P}'$ and the table $\mathcal{T}$. Positive indicates that the sentence is evidential, while negative shows that the sentence is not evidential. To generate the training data for sentence selection, we use the following strategy:

- For each table $\mathcal{T}$, query Wikipedia to retrieve relevant passages using BM25, similar to document retrieval.
- Segment relevant passages into sentences.
- The description sentences from ToTTo are the positive examples, and other sentences from the relevant passages are the negative examples.

We use the BART architecture for our sentence selection model. For each linearized table $\mathcal{L}$, we have one positive sentence s^+ from ToTTo dataset and one generated negative sentence s^-. The linearized table is paired with the positive/negative sentence to create a positive/negative training example. The encoded vectors are fed into a dense layer with softmax activation, and

the sentence selection model is then optimized using the negative log-likelihood of the positive sentences as follows:

$$I = \text{Concatenate}(d, s)$$

$$E = \text{BART}(I)$$

$$O = \text{softmax}(Dense(E_I))$$

$$L = \text{CrossEntropy}(O_{s^+,\mathcal{L}}, O_{s^-,\mathcal{L}})$$

The trained sentence selection is used to predict the probability that a sentence is relevant to a given table, and sentences with scores lower than a threshold are discarded.

5.2.3 Table Verification

This third sub-task requires logical inference from the sentences to the table $\mathcal{T}$, which is defined as:

$$h(s, \mathcal{T}) = y \tag{5.2}$$

where $s \in \mathcal{P}'$ an evidential sentence and $y \in \{\text{True}, \text{False}\}$ is the output label. To train the verification model, a tuple $\langle s, \mathcal{T} \rangle$ in ToTTo where $\mathcal{T}$ is the table, and s is the description sentence can be used as a positive example. To generate the negative examples for table verification training, we follow the following strategy:

- For each table $\mathcal{T}$, sample a cell from the set of highlighted cells as the replaced cell c_r
- Sample a replacing cell c_r' in the same column as the replaced cell

- Replace the value in the replaced cell with the value in replacing cell to create the negative table $\mathcal{T}^-$ and use the tuple $\langle s, \mathcal{T}^- \rangle$ as a negative example.

The table verification model has the same neural architecture as our sentence selection model.

$$I = \text{Concatenate}[s, l]$$

$$E = \text{BART}(I)$$

$$P = \text{softmax}(U \cdot E)$$

$$L = \text{CrossEntropy}(P_{s,\mathcal{T}^+}, P_{s,\mathcal{T}^-})$$

All evidential sentences are encoded together with the table linearization using BART before a binary classifier runs the prediction. The labels in our prediction are True/False, which indicates if the sentence entails/contradicts the table. The table is correct if there is any entailed sentence in the list of evidential sentences. Otherwise, if there is no entailed sentence, the table is considered erroneous and needs to be fixed in the next step.

5.2.4 Error Cell Correction

In the final subtask, our goal is to identify and correct the incorrect cell in the erroneous table $\mathcal{T}$ as follows:

$$p(S^-, \mathcal{T}) = \mathcal{T}' \tag{5.3}$$

where $S^- \in \mathcal{P}'$ is the list of all contradicting sentences and $\mathcal{T}'$ is the corrected version of table $\mathcal{T}$.

For each erroneous table detected in the table verification phase, we input the list of contradicting sentences S^- and table $\mathcal{T}$ into our error cell detection module. We use each contradicting sentence as the context and leverage UnifiedQA [43]to predict the correct value for all cells in the table. The questions for UnifiedQA are generated as follows:

- For columns related to date/time, the question is "When does this happen?". Columns with "month", "year", "date", and "time" in headers are considered date/time columns.
- For other columns, the question is "What is the X?" where "X" is the column's header.

The generated strings from UnifiedQA are the predicted correct cell values. We then use *Jaro-Winkler* string similarity to compare all predictions with the current value. If the similarity score is greater than a threshold, we consider the two values similar, and the cell value is correct. On the other hand, if the similarity value is less than the threshold, the cell value is identified as an error and will be replaced by the generated value. In this thesis, our threshold is set to 0.8 based on experimental results.

Since each contradicting sentence produces one answer for the question, we need to rank the answers and return only the most confident extractions. Because UnifiedQA produces no scoring for its generation, we rank the generated answers based on the verification probability scores of the context sentence.

5.3 Related Work

This section reviews three lines of relevant studies on document retrieval, table-based fact verification, and table pre-trained models.

5.3.1 Document Retrieval

Multi-task DPR [42] exploits multi-task learning by training both DPR passage and query encoder on all KILT tasks. DensePhrases [48] targets the area of knowledge retrieval tasks with short answers. DensePhrases leverages SpanBERT architecture from SpanBERT [40] to index the corpus that has potential answers. RAG [51] combines retrieval and generation problems. RAG retrieves a set of candidate documents, pass them to a seq2seq model for generation, and then marginalizes all the outputs. The retriever is initialized using pre-trained DPR models, the seq2seq module leverages the BART architecture, and the model is trained jointly, which allows both retrieval and generation modules to be finetuned for downstream tasks. Knowledge Graph Induction (KGI) [25] combines DPR and RAG models. However, instead of training jointly as RAG, KGI first trains the DPR model using KILT ground truth and then finetunes the RAG generation model, which yields significant improvements in retrieval performance and thus increases the performance of the later generation process. Re2G [26] follows the architecture of KGI and leverages the retrieval model to generate hard negative examples for generation training. In our work, we combine DPR with the traditional keyword search to exploit the strengths of both methods.

5.3.2 Table-based Fact Verification

The problem of Table-based Fact Verification verifies whether a textual hypothesis holds based on the given tables as evidence. TabFact [14] releases the TabFact dataset, which is designed for the problem and provides two different baseline systems using Table-BERT and Latent Program Algorithm. Wang et al. [89] estimate token-level salience and apply salience-aware data augmentation to train the fact, verification models. Eisenschlos, Krichene, and Müller [22] introduce two

intermediate pre-training tasks by generating synthetic and counterfactual sentences to enrich the original dataset. Eisenschlos, Krichene, and Müller [22] also show that column pruning lowers the computational cost of the model in a small cost of accuracy. In this thesis, we solve the opposite problem, which verifies the factual tables using textual evidence.

5.3.3 Table Pretraining

TAPAS [34] leverages the BERT architecture to encode tables as input using weakly-supervised training. The TAPAS model is optimized to select a set of related cells and then predict the optimal aggregation operation to infer the final results. TAPEX [55] leverages the BART architecture for table-related tasks. TAPAS is trained to generate correct answers for synthesized SQL queries from the tables and thus requires much less training data to achieve better performance when compared to TAPEX. Our sentence selection and table verification modules are built on the BART architecture, similar to TAPEX. However, by adopting ToTTo's linearization algorithm, where headers are grouped with cell values, we outperform both TAPAS and TAPEX, whose linearization algorithms consider headers as extra rows.

5.4 Evaluation

We evaluate our model on the ToTTo [66] dataset. ToTTo is an English dataset created for controlled table-to-text generation. It consists of 83,141 Wikipedia tables and 120,761/7,700/7,700 examples for the train/dev/test set. Target sentences in the test set are not publicly available.

Therefore, we use the dev set as our evaluation dataset. Our system, datasets, and results are available online.[†]

We evaluate three modules: document retrieval, sentence selection, and table verification separately first and then perform an end-to-end evaluation. We train and validate the models for each task using datasets generated from the train and dev set of ToTTo. The train and dev datasets are generated or augmented using the strategies we explained in each corresponding module in Section 5.2. In our end-to-end evaluation, we also show the performance of each individual module and explain if there is any difference between the end-to-end and the standalone evaluation. Since ToTTo is designed for a generation task, and thus the dataset contains only positive examples. To generate test examples for our end-to-end evaluation, we use the same strategy in table verification training to create factually incorrect tables.

5.4.1 Document Retrieval

For the document retrieval task, we evaluate our system with 2 other baselines: BM25 and DPR. We called our method "DPR reranking" since we use DPR encoders and cosine similarity to rerank the BM25 search results. All models are trained and evaluated using the train and dev set of ToTTo. For each table in ToTTo, we perform a query on the KILT knowledge source and collect the top-k passages. We then calculate the retrieval recall at the document level instead of the passage level, where a passage is considered a hit if it belongs to the correct document.

Table 5.5 compares document retrieval methods on ToTTo's dev set, using the top-k retrieval recall with $k \in \{1, 5, 10\}$. DPR reranking is consistently better than DPR and BM25 on all metrics and gains nearly 0.2 in retrieval recall compared with the other methods. Document retrieval

[†]https://github.com/minhptx/seed/

Table 5.5: Document retrieval evaluation

Method	Recall@1	Recall@5	Recall@10
BM25	0.36	0.65	0.80
DPR	0.18	0.56	0.78
DPR reranking (SEED)	0.54	0.93	0.95

recall is the hard upper bound of the system performance since tables cannot be verified without relevant textual evidence. Thus, a high recall of 0.95 when taking top-10 Wikipedia documents allows us to achieve much better performance in the end-to-end evaluation.

Our experiments show that DPR has the lowest performance in document retrieval for tables, contradicting the results in the original DPR paper Karpukhin et al. [42]. One possible explanation is that tables usually contain many numeric values (36% of cell values in ToTTo) and named entities (40% of cell values in ToTTo), which dense representations have difficulties distinguishing. On the other hand, keyword search, where exact matches are preferred, excels in these cases. Using DPR encoders to rerank the BM25 search results allows us to retrieve a set of candidates by exact matching and then take advantage of the contexts of the correct passages and a trained similarity metric to yield the best result.

5.4.2 Sentence Selection

We evaluate our system with two baselines for the sentence selection task: InfoTab [31] and TAPAS [34]:

- InfoTab: A RoBERTa-based model that leverages sentence templates and table pruning for table linearization.

Table 5.6: Sentence selection evaluation

Method	Precision	Recall	F1
InfoTab	0.844	0.931	0.886
TAPAS	0.973	0.951	0.962
SEED	0.976	0.985	0.981

- TAPAS: A BERT-based model designed specifically for tabular data. TAPAS can be fine-tuned to answer questions about tabular data.

InfoTab's original model is designed for three-label classification. We keep the whole architecture in InfoTab but modify the output layer for binary classification. The training and validation sets for sentence selection are generated using BM25 queries. For each example in ToTTo's dev set, we use the BM25 method to query the top 5 passages from the KILT knowledge source. All passages are then segmented into sentences, and the sentence selection task is binary classification on these sentences, where True indicates the correct description sentence and False denotes all other sentences.

As we can see from Table 5.6, SEED outperforms both InfoTab and TAPAS in our sentence selection validation by 10% and 2%, respectively. The main reason for the small performance gaps is that our training and validation sets are generated using BM25 queries for faster retrieval time. Therefore, the documents are collected using exact matching and thus contain little semantic ambiguity. In the end-to-end evaluation, sentences are reranked based on their semantic relevance, which results in harder predictions and bigger performance gaps (Section 5.4.4).

Table 5.7: Table verification evaluation

Method	Precision	Recall	F1
InfoTab	N/A	N/A	N/A
TAPAS	0.926	0.916	0.921
SEED	0.965	0.957	0.961

5.4.3 Table Verification

We evaluate our system using the same two baselines for table verification task: InfoTab and TAPAS. The training and validation sets for table verification are generated using error generation, as shown in Section 5.2.3. An example in table verification is a pair of the ToTTo description sentence paired with the original positive table or the generated negative table.

In our validation, InfoTab training does not converge after ten epochs and keeps outputting only one class in prediction. InfoTab leverages templates for text generation and then applies normal natural language inference. Since InfoTab converts each cell value into a sentence, it has difficulties in bigger tables as in our dataset. Moreover, InfoTab's multiple-sentence templates perform worse in comparing facts and details, which is the main focus of our problem. Similar to the sentence selection task, Table 5.7 shows that SEED outperforms TAPAS in the verification task with an improvement of 4%.

5.4.4 End-to-end Evaluation

In this evaluation, we evaluate the end-to-end system, which includes: document retrieval, sentence selector, and table verification. We evaluate SEED against both InfoTab and TAPAS. Since all the other systems are designed for only inference (table verification) task, we combine InfoTab and TAPAS with the document retrieval module of SEED and evaluate the pipeline together.

Table 5.8: End-to-end evaluation on ToTTo

Method	Document Retrieval			Sentence Selection			Table Verification		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
INFOTAB (w/ SEED Document Retrieval)	0.25	0.88	0.39	0.05	0.56	0.09	N/A	N/A	N/A
TAPAS (w/ SEED Document Retrieval)	0.25	0.88	0.39	0.23	0.27	0.25	0.51	0.43	0.47
SEED	0.25	0.88	0.39	0.67	0.82	0.74	0.88	0.83	0.86

As we can see in Table 5.8, SEED significantly outperforms both INFOTAB and TAPAS. The gaps in performance in sentence selection and table verification widen compared to the individual evaluation. Sentences in the end-to-end evaluation are retrieved using DPR reranking and are difficult to identify the relevance because of the similarity in semantic meanings.

Moreover, low performance in sentence selection affects table verification performance and thus results in low performance for the end-to-end pipelines. For example, INFOTAB's sentence selection module has good recall but very low precision. Even though good recall means that the correct sentences are collected and forwarded to table verification, having very low precision means the TAPAS's verification module will receive a bigger set of sentences, which makes verification predictions much harder.

On the other hand, INFOTAB has a lower recall in sentence selection when compared to TAPAS and SEED. Therefore, the description sentences cannot be found, and the tables cannot be verified correctly. Theoretically, the performance in the table verification phase is upper-bounded by the recall of the sentence selection phase since correct sentences are required to verify the tables correctly. However, all three systems, SEED, INFOTAB, and TAPAS, have an increase in F1 scores between the sentence selection and table verification phases. Since all three systems have excellent performance in table verification validation when the correct sentences are provided, the performance gains can be attributed to random guesses when the correct sentences cannot be identified.

Table 5.9: Table cell correction evaluation

Method	Recall	MRR
SEED	0.65	0.46

5.4.5 Table Cell Correction

In this evaluation, we report the performance of our cell correction method. From the errors reported by our table verification pipeline, we calculate the recall and mean reciprocal rank (MRR) of the correct answers from the list of report answers. Table 5.9 shows that our approach achieved a recall of 0.65 and MRR of 0.46. Since MRR is calculated as $MRR = \frac{1}{rank}$, the result indicates that our correct answers are approximately in the top 2 in all erroneous cases.

5.5 Summary

We presented SEED, a novel end-to-end solution for semantic error detection that can effectively identify semantic errors using textual information. SEED introduces multiple strategies to address the necessity of training data by generating negative examples in all three phases: document retrieval, sentence selection, and table verification. Moreover, we leverage table linearization and the power of massive pre-trained language models to solve the problem of semantic error detection as a sequence-to-sequence problem. Finally, SEED provides a novel solution for identifying erroneous cells and correcting semantic errors by leveraging question-answering methods.

Chapter 6

Discussion

In this dissertation, we presented a novel approach to detecting and correcting errors in tabular data. In this chapter, I summarize the contributions of my thesis. Then, I discuss the main limitations of the presented approach and how we plan to address these limitations in future work.

6.1 Contributions

In this thesis, we presented three main contributions in syntactic and semantic error detection and correction together with a sub-contribution on semantic labeling.

In Chapter 2, we presented SPADE, a novel learning-based system for error detection that can effectively reduce the number of labels required for training while maintaining excellent performance in error detection. SPADE incorporates a probabilistic active learning model that allows signals to capture both internal and external information. During the active learning process, SPADE flexibly updates the active learning model using user labels. Therefore, SPADE achieves

better performance in all five experimental datasets than previous state-of-the-art systems. Furthermore, SPADE utilizes a data augmentation process where we enrich our training datasets with synthetic data. The process generates additional errors and helps SPADE generalize better to unseen errors. As a result, SPADE has a better recall in the evaluation.

In Chapter 3, we introduce a novel domain-independent approach to semantic labeling called DSL that leverages similarity measures and machine learning techniques. In our system, we capture the patterns of matching decisions given the similarity scores between unlabeled attributes and labeled data to find the correct semantic types. The approach allows us to train the machine learning model only once and use it in multiple different domains. Moreover, our similarity features are independent within and across semantic types. We can compute feature vectors using a parallel and distributed implementation to reduce the running time. DSL is used as the semantic alignment algorithm in Chapter 4

In Chapter 4, we presented UDATA, a novel unsupervised approach to infer expressive syntactic structures from data, learn the transformation between source and target data using these syntactic structures, and validate the transformed results. UDATA can transform attribute data from different formats into one desired format without human interaction. Therefore, UDATA can be applied to problems that require fully automatic processing. For example, we have used UDATA for automatically normalizing data in the problem of automatic spatial and temporal indexing for many data sources [44].

In Chapter 5, we presented SEED, a novel end-to-end solution for semantic error detection that can effectively identify semantic errors using textual information. Because of limitations in training data, SEED introduces multiple strategies to generate negative examples in all three main phases of semantic error detection: document retrieval, sentence selection, and table verification.

Table 6.1: Correlation in errors across multiple columns

Id	Name	Rating	Year
tt2131532	Hidden	6.4	2015
tt0084516	Poltergeist	7.4	1982
immortals_2011	Immortals	510,510	2010 2011 2012
tt0026261	Dangerous	7.2	1935

Moreover, we leverage table linearization and the power of massive pre-trained language models to solve the problem of semantic error detection as a sequence-to-sequence problem.

In conclusion, this thesis presents multiple approaches for syntactic and semantic error detection and correction in tables. All the approaches aim to reduce human interaction in data cleaning.

6.2 Limitations

In Chapter 2, SPADE explores indicative signals from attributes in tables to detect potential errors. However, one of the main limitations of SPADE is that it treats attributes in tables separately. In practice, errors or even normal values in different columns can give extra information about errors in another column and thus can help improve the prediction accuracy. Table 6.1 shows an example from Movie dataset where records collected from different sources have rating values stored in different formats and thus need to be fixed. Records with 'ID" that do not follow the format "ttXXXXXXX" are collected from different sources and have two different ratings in the "Rating" attribute. Examining all the columns together gives a stronger signal for detecting erroneous values since they appear in the same records, and learning these correlations can be beneficial for more complex data.

Table 6.2: A complex transformation that requires external knowledge

Raw data	Target data
01/11/2011	Jan 11th, 2011
11/17/2013	Nov 13th, 2013

Chapter 4 provides a novel approach for learning syntactic transformations from tabular data. The work exploits the patterns within table data to cluster string values and learn transformations between these clusters. However, the approach struggles to learn complex transformations where alignments between tokens are insufficient. Table 6.2 shows an example that UDATA cannot handle. The transformation from the numeric format of months to the abbreviations requires external knowledge and is out of the coverage of UDATA.

A limitation of our semantic error detection and correction approach (Chapter 5) is its modular design. SEED contains three main modules: document retrieval, sentence selection, and table verification. Dividing the problem into three sub-problems simplifies the process and allows us to train and optimize each module separately. However, this design allows errors made in previous phases to be carried on to the later phases. Therefore, the performances of the later stages are upper-bounded by the performance of the previous stages. Recent research in fact verification and question answering focuses on training end-to-end model [24, 26, 51]. End-to-end training optimizes all modules simultaneously and allows self-correction between different phases. For example, RAG [51] marginalizes the documents retrieved from document retrieval for question generation and thus minimizes the effect of incorrect retrieval.

Table 6.3: Table in Wikipedia about "Iain Glen"'s awards

Year	Award	Category	Work	Result
1990	Silver Bear	Best Actor	Silent Scream	Nominated

6.3 Future Work

A direction for future work is to improve the response time of different modules in our thesis. For example, currently, SPADE has an average running time of 62 seconds across five different datasets, which makes it hard to apply SPADE as an interactive approach. In our method, the PSL model is currently the bottleneck because of its inference time. We plan to reduce the inference time by replacing the PSL inference with simpler probabilistic models, such as Logistic Regression and Random Forests. Moreover, we also want to explore knowledge distillation to simplify our semantic error detection and correction models to obtain faster and more lightweight models. Since this thesis contains approaches for multiple sub-problems in data cleaning, combining all approaches into one complete solution is ideal. We believe that open-domain web text can also be exploited to help syntactic error detection and correction. Similarly, users' feedback can provide additional knowledge for semantic error detection and correction. For example, Table 6.3 shows the example from Chapter 1 but the "Result" is modified to "Nominated". Currently, SEED cannot detect this as an error since "Nominated" is semantically close to "Won" in meaning; thus, most natural language inference systems cannot distinguish between them. Having user curation, in this case, is valuable, and SEED can incrementally learn from its mistake to make better predictions later.

We plan to support complex transformations requiring external knowledge for error correction. Existing research has studied the uses of web tables and web forms to extract potential

transformations. Our plan is to exploit publicly-available source code from code-sharing platforms, such as GitHub[*] and GitLab[†]. We then can analyze the source code, extract transformation functions and their parameters, and then build a database of transformation programs. We can synthesize training input/output using the transformation programs and train a machine model to suggest the best transformation programs given certain scenarios. In addition, we can also exploit open-domain web tables and knowledge bases to learn value mappings. Transformation programs can then be learned from these mappings.

Finally, we plan to implement an interactive system that combines all the approaches presented in this thesis. The system will allow users to quickly detect and correct errors in tabular data. Our plan is to design an user interface where the system can suggest a set of potential errors and then incrementally update the results based on user feedback.

[*]https://github.com/
[†]https://gitlab.com

Bibliography

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. "Detecting data errors: Where are we and what needs to be done?" In: *Proceedings of the VLDB Endowment* 9.12 (2016), pp. 993–1004.

[2] Ziawasch Abedjan, Toni Grütze, Anja Jentzsch, and Felix Naumann. "Mining and profiling RDF data with ProLOD++". In: *Proceedings of the International Conference on Data Engineering (ICDE)*. 2014, pp. 1198–1201.

[3] Ziawasch Abedjan, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. "DataXFormer: A robust transformation discovery system". In: *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE. 2016, pp. 1134–1145.

[4] José Luis Ambite, Sirish Darbha, Aman Goel, Craig A. Knoblock, Kristina Lerman, Rahul Parundekar, and Thomas Russ. "Automatically Constructing Semantic Web Services from Online Sources". In: *Proceedings of the 8th International Semantic Web Conference*. Springer-Verlag, 2009, pp. 17–32. ISBN: 978-3-642-04929-3. DOI: 10.1007/978-3-642-04930-9_2. (Visited on 04/30/2016).

[5] Sören Auer, Jan Demter, Michael Martin, and Jens Lehmann. "LODStats–an extensible framework for high-performance dataset analytics". In: *International Conference on Knowledge Engineering and Knowledge Management*. Springer. 2012, pp. 353–362.

[6] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. "Hinge-loss markov random fields and probabilistic soft logic". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 3846–3912.

[7] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. "Inferring XML schema definitions from XML data". In: *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment. 2007, pp. 998–1009.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146. ISSN: 2307-387X.

[9] Katrin Braunschweig. “Recovering the semantics of tabular web data”. In: (2015).

[10] Katrin Braunschweig, Maik Thiele, Julian Eberius, and Wolfgang Lehner. “Column-specific context extraction for web tables”. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015, pp. 1072–1077.

[11] Leo Breiman. “Random Forests”. en. In: *Machine Learning* 45 (2001), pp. 5–32. ISSN: 0885-6125, 1573-0565. DOI: 10.1023/A:1010933404324. (Visited on 01/24/2016).

[12] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[13] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. 2016, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785.

[14] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. “Tabfact: A large-scale dataset for table-based fact verification”. In: *arXiv preprint arXiv:1909.02164* (2019).

[15] Xu Chu, Ihab F Ilyas, Paolo Papotti, and Yin Ye. “RuleMiner: Data quality rules discovery”. In: *2014 IEEE 30th International Conference on Data Engineering*. IEEE. 2014, pp. 1222–1225.

[16] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. “Katara: A data cleaning system powered by knowledge bases and crowdsourcing”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 2015, pp. 1247–1261.

[17] Nick Craswell. “Mean reciprocal rank”. In: *Encyclopedia of Database Systems*. Springer, 2009, pp. 1703–1703. (Visited on 01/31/2016).

[18] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. “NADEEF: a commodity data cleaning system”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM. 2013, pp. 541–552.

[19] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. *The Magellan Data Repository*. https://sites.google.com/site/anhaidgroup/projects/data. 2015.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[21] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. “Robustfill: Neural program learning under noisy I/O”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 990–998.

[22] Julian Martin Eisenschlos, Syrine Krichene, and Thomas Müller. “Understanding tables with intermediate pre-training”. In: *arXiv preprint arXiv:2010.00571* (2020).

[23] Kathleen Fisher, David Walker, and Kenny Q Zhu. “LearnPADS: automatic tool generation from ad hoc data”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pp. 1299–1302.

[24] Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, and Alfio Gliozzo. “Robust Retrieval Augmented Generation for Zero-shot Slot Filling”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 1939–1949. DOI: 10.18653/v1/2021.emnlp-main.148.

[25] Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, and Alfio Gliozzo. “Robust retrieval augmented generation for zero-shot slot filling”. In: *arXiv preprint arXiv:2108.13934* (2021).

[26] Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Rajaram Naik, Pengshan Cai, and Alfio Gliozzo. “Re2G: Retrieve, Rerank, Generate”. In: *arXiv preprint arXiv:2207.06300* (2022).

[27] Aman Goel, Craig A. Knoblock, and Kristina Lerman. “Exploiting structure within data for accurate labeling using conditional random fields”. In: *Proceedings of the 14th International Conference on Artificial Intelligence (ICAI)*. Vol. 69. 2012. (Visited on 01/16/2016).

[28] Sumit Gulwani. “Automating string processing in spreadsheets using input-output examples”. In: *ACM SIGPLAN Notices*. Vol. 46. 1. ACM. 2011, pp. 317–330.

[29] Sumit Gulwani, William R Harris, and Rishabh Singh. “Spreadsheet data manipulation using examples”. In: *Communications of the ACM* 55.8 (2012), pp. 97–105.

[30] Kalpa Gunaratna, Krishnaprasad Thirunarayan, Amit Sheth, and Gong Cheng. “Gleaning types for literals in rdf triples with application to entity summarization”. In: *European Semantic Web Conference*. Springer. 2016, pp. 85–100.

[31] Vivek Gupta, Maitrey Mehta, Pegah Nokhiz, and Vivek Srikumar. “INFOTABS: Inference on tables as semi-structured data”. In: *arXiv preprint arXiv:2005.06117* (2020).

[32] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. “Holodetect: Few-shot learning for error detection”. In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 829–846.

[33] Joseph M Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, et al. “The MADlib analytics library: or MAD skills, the SQL”. In: *Proceedings of the VLDB Endowment* 5.12 (2012), pp. 1700–1711.

[34] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. “TaPas: Weakly supervised table parsing via pre-training”. In: *arXiv preprint arXiv:2004.02349* (2020).

[35] Kevin Hu, Snehalkumar’Neil’S Gaikwad, Madelon Hulsebos, Michiel A Bakker, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. “Viznet: Towards a large-scale visualization learning and benchmarking repository”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–12.

[36] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. “Active learning by querying informative and representative examples”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.10 (2014), pp. 1936–1949.

[37] Zhipeng Huang and Yeye He. “Auto-detect: Data-driven error detection in tables”. In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 1377–1392.

[38] Andrew Ilyas, Joana MF da Trindade, Raul Castro Fernandez, and Samuel Madden. “Extracting Syntactical Patterns from Databases”. In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE. 2018, pp. 41–52.

[39] Zhongjun Jin, Michael J. Cafarella, H. V. Jagadish, Sean Kandel, and Michael Minar. “Unifacta: Profiling-driven String Pattern Standardization”. In: *CoRR* abs/1803.00701 (2018). arXiv: 1803.00701. URL: http://arxiv.org/abs/1803.00701.

[40] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. “Spanbert: Improving pre-training by representing and predicting spans”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 64–77.

[41] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. “Wrangler: Interactive visual specification of data transformation scripts”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 3363–3372.

[42] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. "Dense Passage Retrieval for Open-Domain Question Answering". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550.

[43] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. "Unifiedqa: Crossing format boundaries with a single qa system". In: *arXiv preprint arXiv:2005.00700* (2020).

[44] Craig A Knoblock, Aparna R Joshi, Abhishek Megotia, Minh Pham, and Chelsea Ursaner. "Automatic Spatio-temporal Indexing to Integrate and Analyze the Data of an Organization". In: *Proceedings of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. ACM. 2017, p. 7.

[45] Craig A Knoblock, Pedro Szekely, Eleanor Fink, Duane Degler, David Newbury, Robert Sanderson, Kate Blanch, Sara Snyder, Nilay Chheda, Nimesh Jain, et al. "Lessons learned in building linked data for the American art collaborative". In: *International Semantic Web Conference*. Springer. 2017, pp. 263–279.

[46] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. "Activeclean: Interactive data cleaning for statistical modeling". In: *Proceedings of the VLDB Endowment* 9.12 (2016), pp. 948–959.

[47] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics (NRL)* 2.1-2 (1955), pp. 83–97.

[48] Jinhyuk Lee, Mujeen Sung, Jaewoo Kang, and Danqi Chen. "Learning Dense Representations of Phrases at Scale". In: *Association for Computational Linguistics (ACL)*. 2021.

[49] Erich L Lehmann and Joseph P Romano. "Testing Statistical Hypotheses (Springer Texts in Statistics)". In: (2005).

[50] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *arXiv preprint arXiv:1910.13461* (2019).

[51] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[52] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. “Truth Finding on the Deep Web: Is the Problem Solved?” In: *Proceedings of the VLDB Endowment* 6.2 (2012).

[53] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. “Annotating and Searching Web Tables Using Entities, Types and Relationships”. In: *Proc. VLDB Endow.* 3 (2010), pp. 1338–1347. ISSN: 2150-8097. DOI: 10.14778/1920841.1921005. (Visited on 01/16/2016).

[54] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. “Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations”. In: *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*. 2021, pp. 2356–2362.

[55] Qian Liu, Bei Chen, Jiaqi Guo, Zeqi Lin, and Jian-guang Lou. “Tapex: Table pre-training via learning a neural sql executor”. In: *arXiv preprint arXiv:2107.07653* (2021).

[56] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[57] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. “Raha: A configuration-free error detection system”. In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 865–882.

[58] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 978-0-521-86571-5.

[59] Zelda Mariet, Rachael Harding, Sam Madden, et al. “Outlier detection in heterogeneous datasets using automatic tuple expansion”. In: (2016).

[60] Varish Mulwad, Tim Finin, and Anupam Joshi. “Semantic message passing for generating linked data from tables”. In: *The Semantic Web–ISWC 2013*. Springer, 2013, pp. 363–378.

[61] Varish Vyankatesh Mulwad. “TABEL - A Domain Independent and Extensible Framework for Inferring the Semantics of Tables”. PhD thesis. University of Maryland, Baltimore County, 2015. (Visited on 07/08/2016).

[62] Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. “Ed2: A case for active learning in error detection”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 2249–2252.

[63] Yixin Nie, Haonan Chen, and Mohit Bansal. “Combining fact extraction and verification with neural semantic matching networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 6859–6866.

[64] Mourad Ouzzani, Hossam Hammady, Zbys Fedorowicz, and Ahmed Elmagarmid. “Rayyan–a web and mobile app for systematic reviews”. In: *Systematic reviews* 5.1 (2016), p. 210.

[65] Saswat Padhi, Prateek Jain, Daniel Perelman, Oleksandr Polozov, Sumit Gulwani, and Todd Millstein. “Flashprofile: Interactive synthesis of syntactic profiles”. In: *arXiv preprint arXiv:1709.05725* (2017).

[66] Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. “ToTTo: A Controlled Table-To-Text Generation Dataset”. In: *Proceedings of EMNLP*. 2020.

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[68] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. “KILT: a Benchmark for Knowledge Intensive Language Tasks”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 2523–2544. DOI: 10.18653/v1/2021.naacl-main.200.

[69] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. “Language models as knowledge bases?” In: *arXiv preprint arXiv:1909.01066* (2019).

[70] Minh Pham, Suresh Alse, Craig A Knoblock, and Pedro Szekely. “Semantic labeling: a domain-independent approach”. In: *International Semantic Web Conference*. Springer. 2016, pp. 446–462.

[71] Minh Pham, Craig A Knoblock, and Jay Pujara. “Learning Data Transformations with Minimal User Effort”. In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 657–664.

[72] Minh Pham, Craig A. Knoblock, Muhao Chen, Binh Vu, and Jay Pujara. “SPADE: A Semi-supervised Probabilistic Approach for Detecting Errors in Tables”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 3543–3551. URL: https://doi.org/10.24963/ijcai.2021/488.

[73] Vijayshankar Raman and Joseph M Hellerstein. “Potter’s wheel: An interactive data cleaning system”. In: *VLDB*. Vol. 1. 2001, pp. 381–390.

[74] S. K. Ramnandan, Amol Mittal, Craig A. Knoblock, and Pedro Szekely. “Assigning Semantic Labels to Data Sources”. In: *The Semantic Web. Latest Advances and New Domains*. Springer International Publishing, 2015, pp. 403–417. (Visited on 01/16/2016).

[75] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. “Matching HTML Tables to DBpedia”. In: *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*. WIMS ’15. New York, NY, USA: ACM, 2015, 10:1–10:6. ISBN: 978-1-4503-3293-4. (Visited on 04/20/2016).

[76] Stephen Robertson, Hugo Zaragoza, et al. “The probabilistic relevance framework: BM25 and beyond”. In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389.

[77] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. “Okapi at TREC-3”. In: *Nist Special Publication Sp* 109 (1995), p. 109.

[78] Chengxun Shu and Hongyu Zhang. “Neural Programming by Example.” In: *AAAI*. 2017, pp. 1539–1545.

[79] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. “Discriminative learning of deep convolutional feature point descriptors”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 118–126.

[80] Rishabh Singh. “Blinkfill: Semi-supervised programming by example for syntactic string transformations”. In: *Proceedings of the VLDB Endowment* 9.10 (2016), pp. 816–827.

[81] Rishabh Singh and Sumit Gulwani. “Learning semantic string transformations from examples”. In: *Proceedings of the VLDB Endowment* 5.8 (2012), pp. 740–751.

[82] Rishabh Singh and Sumit Gulwani. “Predicting a correct program in programming by example”. In: *International Conference on Computer Aided Verification*. Springer. 2015, pp. 398–414.

[83] Rishabh Singh and Sumit Gulwani. “Transforming spreadsheet data types using examples”. In: *ACM SIGPLAN Notices*. Vol. 51. 1. ACM. 2016, pp. 343–356.

[84] Kexuan Sun, Harsha Rayudu, and Jay Pujara. “A Hybrid Probabilistic Approach for Table Understanding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 5. 2021, pp. 4366–4374.

[85] Zareen Syed, Tim Finin, Varish Mulwad, and Anupam Joshi. “Exploiting a web of semantic data for interpreting tables”. In: *Proceedings of the Second Web Science Conference*. 2010. (Visited on 01/16/2016).

[86] Mohsen Taheriyan, Craig A. Knoblock, Pedro Szekely, and José Luis Ambite. “Learning the semantics of structured data sources”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* (2016). (Visited on 01/26/2016).

[87] James Thorne and Andreas Vlachos. “Evidence-based factual error correction”. In: *arXiv preprint arXiv:2012.15788* (2020).

[88] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. “Recovering semantics of tables on the web”. In: *Proceedings of the VLDB Endowment*. 2011, pp. 528–538. (Visited on 01/16/2016).

[89] Fei Wang, Kexuan Sun, Jay Pujara, Pedro Szekely, and Muhao Chen. “Table-based fact verification with salience-aware learning”. In: *EMNLP - Findings*. 2021.

[90] Pei Wang and Yeye He. “Uni-detect: A unified approach to automated error detection in tables”. In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pp. 811–828.

[91] Ziqi Wang, Gu Xu, Hang Li, and Ming Zhang. “A Probabilistic Approach to String Transformation”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.5 (2014), pp. 1063–1075.

[92] Bo Wu and Craig A Knoblock. “Maximizing Correctness with Minimal User Effort to Learn Data Transformations”. In: *Proceedings of the 21st International Conference on Intelligent User Interfaces*. ACM. 2016, pp. 375–384.

[93] Bo Wu and Craig A. Knoblock. “An Iterative Approach to Synthesize Data Transformation Programs”. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. 2015.

[94] Jie Zhou, Xu Han, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. “GEAR: Graph-based evidence aggregating and reasoning for fact verification”. In: *arXiv preprint arXiv:1908.01843* (2019).