Semantic Annotation of Numerical Data in Web Tables

by

Yuchen Su

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

A large portion of quantitative information about entities mentioned in Web pages is expressed as Web tables, and these tables often lack proper schema and annotation, which introduces challenges for the purpose of querying and further analysis. In this thesis, we study the problem of annotating the numerical columns of Web tables by linking them to properties in a knowledge graph.

Unlike some approaches in the literature that use contextual information (such as column headers and captions), which can be missing or not reliable, or labeled data for model training, which can be difficult to obtain, our approach relies only on the semantic information readily available in knowledge graphs. We show that our approach can reliably detect both semantic types (e.g., height) and unit labels (e.g., centimeters) when the semantic type is present in the knowledge graph.

Our evaluation on real-world web tables data shows that our method outperforms, in terms of precision and F1 score, some of the state-of-the-art approaches on semantic labeling. Our evaluation also gives an insight of precision on unit detection given that no previous works have explored the similar problem to the best of our knowledge.

In memory of Nancy Fairs-Natland

For all the help and support when I first came to the land of Canada

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Davood Rafiei, for guiding me through the research work and providing constant support. It would be impossible to complete this work without his help. He has been patient at all times even though I do not meet all of his requirements sometime. He introduces me to the research world and I have learned much more than what is presented here. Working with Dr. Rafiei is a valuable experience for me which I will benefit from for the rest of my life. I would like also to thank Dr. Di Niu and Dr. Mario Nascimento, for being my commitee members. They have spent valuable time in reading and providing comments to my work. They help me to present the best of this work.

On a personal level, I have to thank my parents for raising me up, encouraging me when I experienced down times and providing invaluable supports. I am also grateful to my wife Yourui, who have always been a good listener, and was always there when I needed help. She is also one of the main contributors of the datasets used in this work, which require large amount of manual labeling.

Finally, I would like to thank everyone in Dr. Rafiei's research group. I am grateful to be able to finish grad school with the amazing friends. Thanks to Arif Hasnat and Arash Dargahi for all the discussions and for sharing their lives with me. The quarantine year has been a hard time for all of us and we manage to go through it together.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

There are hundreds of millions of relational tables that are embedded in the HTML content of webpages [5], and many of these tables contain highly valuable data. To leverage this data in downstream tasks (*e.g.* question answering and information extraction), one requires some knowledge of the semantics of the rows and the columns. For example, consider a few industrialized countries and some quantitative information about each country, as shown in Table 1.1. Without knowing what the numerical columns are describing about the countries in the table, one cannot use the data for further analysis.

China	1,373,541	4,566	8,147.94
United States	323,995	3,602	57,951.58
Japan	126,702	1,368	38,761.82
Germany	80,722	1,050	42,107.52

Table 1.1: A few industrialized countries

1.1.1 Knowledge Graph

Knowledge graph (KG) is a way of storing and representing massive unstructured data that may be used by computer systems for possible annotations. Specifically, knowledge graphs describe real-world entities and their relationships to one another [33], and are frequently used in many research and business applications [25], [35]. The term "knowledge graph" starts gaining pop-

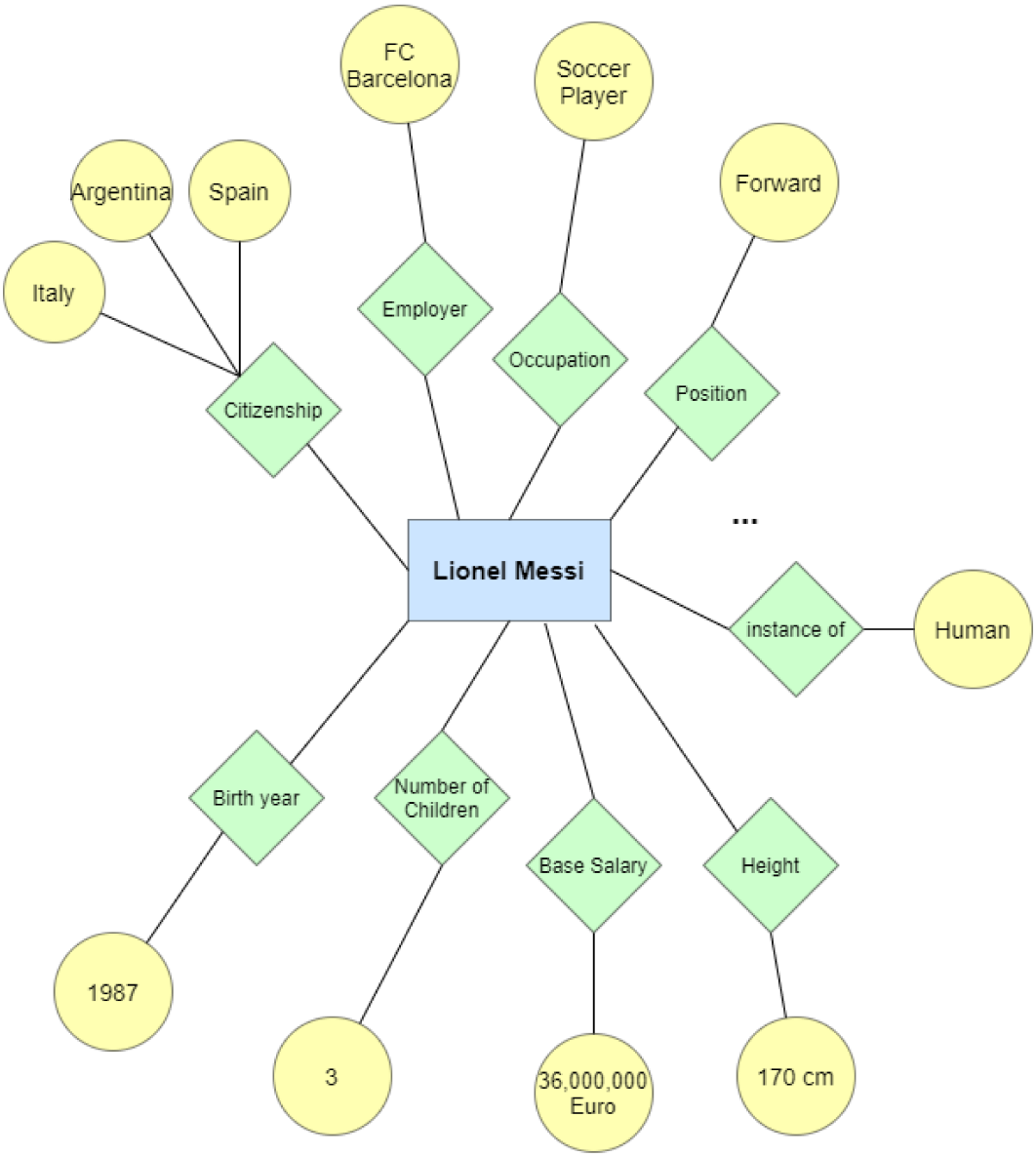

Figure 1.1: Illustration of knowledge representation for Lionel Messi(Q615) in WikiData

ulation when Google first introduced its knowledge graph in 2012, with the goal of enhancing its search function and enabling search for real-world objects instead of strings [11]. The term has been frequently used since then, with no official or clear formal definition given. Recently, researchers have tried to define a knowledge graph to be a graph structure describing real-world entities and their interrelations, as well as possible classes and relations of entities in a schema [27].

There are many existing knowledge graphs that are publicly available such as DBpedia [1], WikiData[2] and YAGO[3]. These knowledge graphs extract and derive their knowledge about entities from different sources. DBpedia extracts structured information from Wikipedia, and keeps track of any changes in Wikipedia articles. It also derives its own set of cross-domain ontology, which is constructed by mapping the most commonly used infoboxes[4]. WikiData leverages and stores structured data from wiki projects such as Wikipedia, Wikitionary, Wikisource, *etc*. It also accepts human contribution and is maintained by Wikimedia community. WikiData focuses mainly on *items* or *entities*, where each entity is assigned a unique ID. YAGO is another knowledge graph where they extract data from a variety of sources such as Wikipedia, WordNet[5], WikiData, GeoNames[6] and others. All data in YAGO is manually curated in order to maximize accuracy. In this thesis we mainly use WikiData as our knowledge graph, but other knowledge graphs may also be used instead. An example representation of knowledge about an entity is shown in Figure 1.1.

One common property of the knowledge graphs listed above is that the data can be are modeled as RDF graphs. It is a model for linked data and is commonly used for knowledge graphs. The model consists of a finite set of RDF triples (s, p, o) where s, p and o in the triple represent subject, predicate and object in that order. As an example, in Figure 1.1 the corresponding

[1] `http://dbpedia.org/`
[2] `https://www.wikidata.org/`
[3] `https://yago-knowledge.org/`
[4] `https://www.dbpedia.org/about/`
[5] `https://wordnet.princeton.edu/`
[6] `https://www.geonames.org/`

RDF triple for the information "Messi works for FC Barcelona" is *(Lionel Messi, Employer, FC Barcelona)*. In this thesis, we use the term "property" for predicates in an RDF triple. One advantage of modeling data as RDF is that the data can be accessed and leveraged using SPARQL queries, which allows for complex query searches that suit specific information needs.

We observe that many properties of an entity is expressed in numeric values such as "birth year" and "base salary" in Figure 1.1. In our problem, if we treat the knowledge graph and its numeric properties as sources of semantic information, the columns of a table may be annotated with semantic types from a knowledge graph, and those annotations can provide a basic understanding of the table semantics for many applications that are either trained on or work with knowledge graphs.

1.1.2 Challenges

There are some challenges in mapping the columns of a table to a semantic type in a knowledge graph. First, many values in web tables are less likely to be present in a knowledge graph. This is a common problem for both textual and numerical columns. Second, numerical values often provide a quantitative description of an entity or a relationship and there can be slight differences between sources. For example, the height of Gheorghe Muresan, a basketball player, is listed 231cm in Wikipedia [7] and 234cm in Wikidata [8]. This makes the problem of matching and annotation for numerical columns more challenging. Third, numerical quantities often change over time. For example, it is less likely for a country name to change but a country population, GDP, import and export values can change from one year to next. Finally, numerical quantities can be expressed in different units and scales, and this introduces another twist in matching numerical columns.

Annotating tabular data is mostly studied in the context of entity types and for textual columns [6], [10], [37]. Despite the fact that numerical data makes

[7] `https://en.wikipedia.org/wiki/List_of_tallest_players_in_National_Basketball_Association_history`

[8] `https://www.wikidata.org/wiki/Q460116`

up over 40% of the content of Web tables [31], the annotation of numeric columns has not been much explored in the literature. While some of the approaches developed for textual columns may be applied to numerical data (*e.g.* [34], [36]), a common approach for annotating numerical columns has been based on a comparison of the column statistics (*e.g.* mean and standard deviation) to detect if column values follow the same distribution as a known distribution such as a semantic type in a knowledge graph [17], [22], [29], [30].

The two underlying assumptions here are that the values of each semantic type follows a known distribution (*e.g.* normal, exponential, gamma, etc.) and that the query column is a random sample from the same distribution. These assumptions are easily violated in real world settings, and the query column may not be a random sample of the target population. For example, the query column may be associated to a selected set of entities (*e.g.* the GDP of a few industrialized countries) whereas the target semantic type may represent a larger population (*e.g.* the GDP of all countries). Also a fundamental problem here is using the statistical test to accept the hypothesis that the sample is taken from a distribution; we can reject the null hypothesis when the means are apart but we cannot accept it when the means are close. Other approaches (*e.g.* [34]) adopt supervised learning where the performance heavily depends on the choice of labeled data and the alignment of query columns with the labeled data. There is also the cost of labelling, which can be enormous for preparing sufficient data for training. There are also approaches for unit labeling based on table metadata such as column headers [31]. These approaches are not applicable when such data do not exist or are not reliable.

1.2 Problem Definition

The problem studied in this thesis is given a table with one or more numerical columns, we want to assign to each numerical column the most likely semantic types and unit labels. We refer to a column being tagged or annotated as a *query column*. The set of potential candidate semantic types is provided by knowledge graph properties.

Definition 1 (Query column)**.** *A query column is a set of numerical values of arbitrary length that the user is interested in finding the semantic meaning of.*

We further define "KG property" to be a collection of values in a knowledge graph that describe the same predicate (*e.g.* height), and the name of the predicate may provide an annotation for the values in the collection.

Definition 2 (Knowledge graph property)**.** *A knowledge graph property, or KG property, is a set of numeric values extracted from the object field in RDF triples of a knowledge graph that have the same predicate. The name of the predicate is set to be the label of the set of extracted values.*

Column annotation tasks are often transformed into matching between a query column and KG properties. In this thesis, we consider matching as a *mapping* construction between the query column and KG properties. In mathematics, the term *mapping* generally refers to value assignments from one set to another. In our context we adapt the following definition:

Definition 3 (Cost)**.** *A Cost value is a measurement of degree of difference between two numeric values. It represents how far apart two numeric values are. The cost value is smaller when two numeric values are close, and is larger when two numeric values appears to be different. In this work, we define the Cost value to be the absolute difference between two numeric values, where* $Cost(a, b) = |a - b|$ *and* $a, b \in \mathbb{R}$.

Definition 4 (Mapping)**.** *Given a query column and a KG property, a mapping is a set of value assignments that prescribes for each value in the query column a value in the KG property. Each mapping is associated with a total Cost representing the distance between the query column and the KG property.*

Here we set the constraint that a mapping must be constructed from a query column to a KG property. Based on the definition of mapping, each value in the domain needs to be mapped. For knowledge graph in general, we expect the size of a candidate KG property to be much larger than the

size of a query column. Thus a mapping from KG property to query column will mostly contain irrelevant value assignments, therefore less meaningful. Figure 1.2 shows a graphical representation of possible mappings from query column to KG property. We will review this definition and explain mapping in more detail in Chapter 3.

Definition 5 (Injective Property)**.** *Given a mapping between a query column and a KG property, the mapping is injective where each value in the query column is mapped to a unique value in the KG property. In other words, no two values in the query column are mapped to the same value in the KG property.*

It is desirable to have the injective property because it ensures that the cost of a mapping represents the actual distance between two numeric columns. For mappings without the injective property, values from the query column may be mapped to an outlier by chance. This can happen if the query column and KG property are sampled from different distributions and an outlier exists in the KG property which is closer to the query column, all values in the query column may map to the outlier and have a relatively small cost. As the size of KG property increases, the chance of matching to an outlier also increases, and the algorithm may be biased towards larger KG properties.

However, the injective property may not be applicable for all cases. When the size of the query column is greater than the size of KG property, an injective mapping is not possible. In order to allow mapping construction in such cases, we relax the injective property and let values in the KG property to be mapped more than once.

Our objectives is: (1) given a numeric column q and a candidate column c, find a mapping $f : q \to c$ such that $\sum_{(u,v)\in f} Cost(u, v)$ is minimized; (2) given a collection of candidate numeric columns C, for each $c \in C$, construct a mapping f_c as described in (1), and return the numeric column c subject to:

$$\min_{c\in C} \sum_{(u,v)\in f_c} Cost(u, v). \tag{1.1}$$

Figure 1.3 gives an illustration of finding the desired KG property.

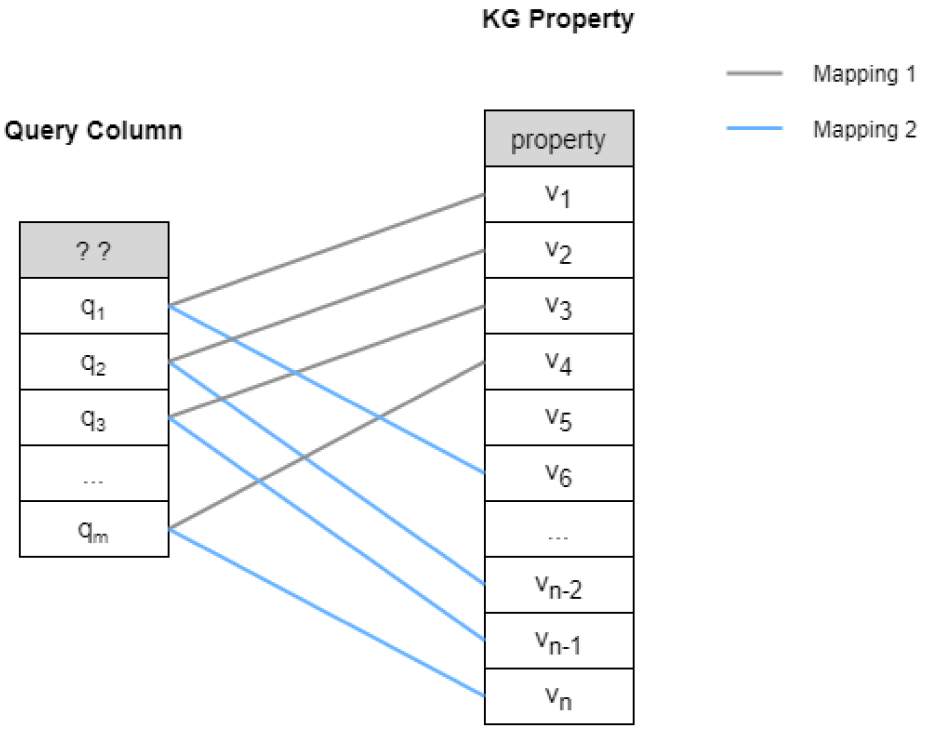

Figure 1.2: Illustration of query column mapping

In real situations, a query column may be annotated with multiple property labels when it is relevant or similar to more than one semantic type in a knowledge graph such as "length" and "distance". As a result our goal may be restated as finding top k nearest matches.

1.3 Contribution

In this work, we propose a column annotation method which assigns both semantic and unit labels to a given numeric column, referred to as the query column. This is done by mapping the query column to a property in a knowledge graph. We show that our method overcomes some of the limitations of existing approaches by directly mapping query column entries and not relying on some aggregate statistics. We formulate the problem as an optimization problem where a cost is associated to each entry mapping and the goal is finding a mapping of the query column with the least cost. Our evaluation on annotating the columns of real web tables reveals that this approach significantly outperforms the existing approaches.

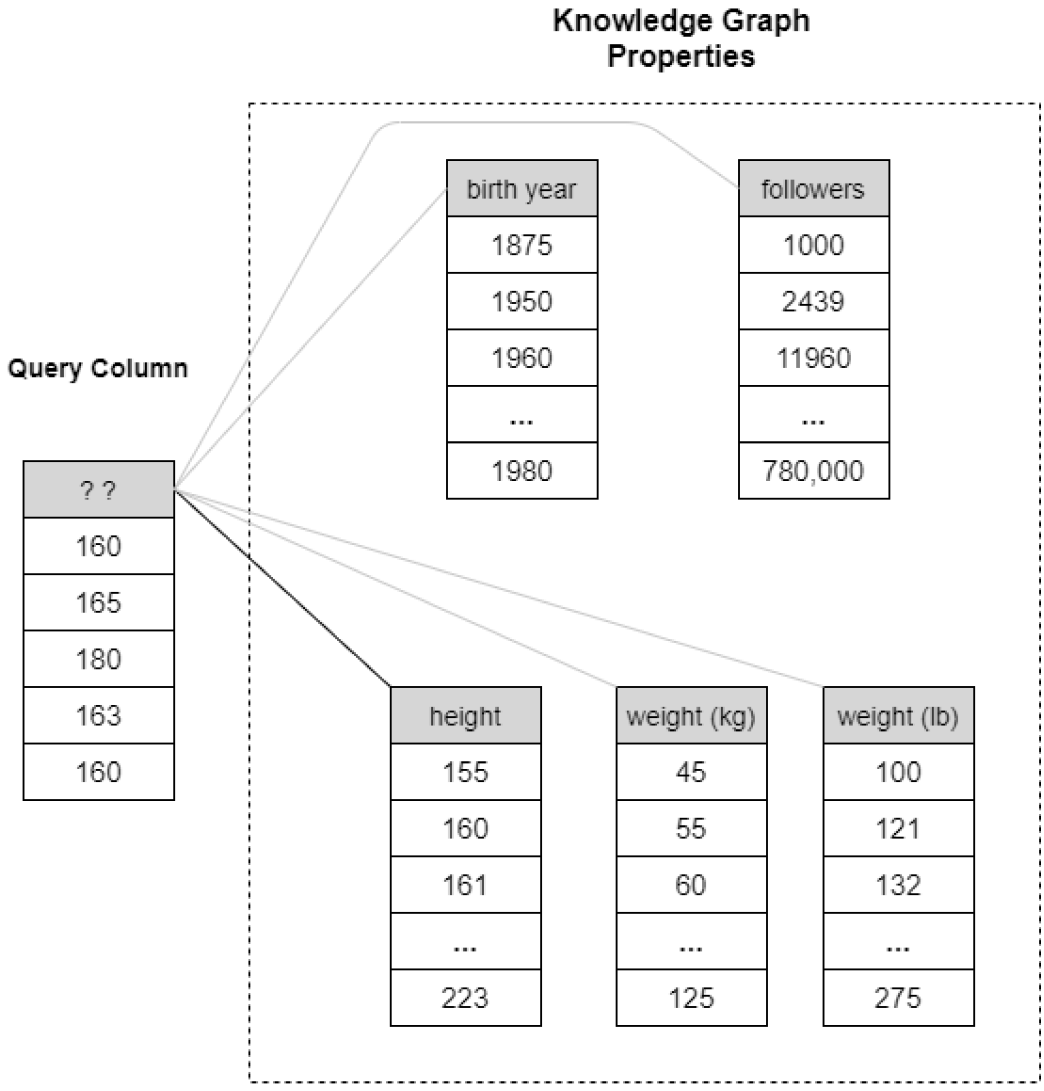

Figure 1.3: Illustration of searching the most similar KG property

Our work makes two main contributions in annotating numerical column. The first is that we propose a framework for mapping query columns to a knowledge graph and show that such mappings can provide reliable semantic types to the query columns. In addition, We develop some pruning strategies to filter mappings that cannot produce "good" solutions hence saving the optimization time. The second contribution is that our proposed framework is able to detect the associated unit label of a numerical column if one exists. Particularly, our framework relies solely on the content in a column, and does not require additional sources such as column headers. To the best of our knowledge, no previous works have tried tackling a similar problem. Our work provides novel insight on how much we can achieve on the task of unit labeling. Finally, we evaluate our work on real data and show that our method significantly improves upon the state-of-the-art in terms of the precision of predicting the semantic labels.

1.4 Outline

In the rest of this thesis, we first start with introducing related works in Chapter 2 along with brief explanations of the techniques they use. We present our method in Chapter 3 in two steps: Section 3.2 explains the process of compiling necessary knowledge from a knowledge graph, and Section 3.3 discusses the problem of mapping construction between a query column and knowledge graph properties. We present our evaluation results in Chapter 4. This includes a performance comparison on the task of semantic and unit labeling with baseline methods, and a comprehensive study of the impact of all parameters used in our method. Finally, we show our conclusion and discuss limitations and future work directions in Chapter 5.

Chapter 2

Related Works

There has been a growing research interest in annotating the columns of tables collected from the Web and open data sources. The goal has been at predicting a semantic label to each column of a table that can help downstream tasks such as web searching and schema completion that use those tables. This chapter introduces existing works closely related to the goal listed above. The related works can be grouped into (1) knowledge graph matching-based approaches and (2) a variety of supervised approaches. Since our method integrates knowledge graph matching in the annotation process, we first present related works that are built on the same principle. Next, we introduce a few works which utilizes different supervised techniques including deep learning and probabilistic graphical model. Then, we briefly review a few works that target columns in general without treating numerical columns differently. After that, we also review one previous work on unit detection which is used as one baseline in our evaluation. Finally, we summarize the limitations of the existing related works.

2.1 Property Matching in Knowledge Graph

One of the key challenges in KG property matching is how to recognize similar properties given a query column. In previous works, the measurement of similarity is often viewed as a distance measure. Thus, the problem becomes finding KG properties that have the least distance to a given query column. We discuss a few techniques in property matching as well as the distance

measurements they have used.

2.1.1 Similarity-based Scoring

Hignette *et al.* [13] propose a scoring function that calculates a score for each ontology t given a query column col. The score is based on two components: (1) the matching of the column title and (2) the matching of the unit information in the column content. A sub-score is computed for each component and the sub-scores are combined by a product. The ontology that has the highest final score is returned as a label. The scoring function can be written as:

$$Score(t, col) = 1 - (1 - score_{\text{title}}(t, col)) * (1 - score_{\text{unit}}(t, col)), \qquad (2.1)$$

where $score_{\text{title}}(t, col)$ is the term similarity between the column title and the ontology name. The authors do not specify what similarity measurement is used in this paper. Based on the equation, any similarity function that returns a similarity value between 0 and 1 is valid. Possible similarity functions include Jaccard Similarity and normalized Cosine Similarity. The actual result may vary depending on if constraints such as order and stop words are considered.

$score_{\text{unit}}(t, col)$ is the unit similarity between a unit u listed in the column and the set of units T_u associated with t. $score_{\text{unit}}(t, col)$ equals to $\frac{1}{|T_u|}$ if $t \in T_u$, and 0 otherwise. If the unit information is missing in the query column, u is set to a special value equivalent to "no unit" value in ontology. The term $score_{\text{unit}}(t, col)$ is set in a way that matching a unit in an ontology with less units is preferred over matching to ontology with more units. The goal of the final scoring function is to maximize the impact of sub-scores when the component shows strong evidence of matching. An exact match of either component will result in a final score of 1. On the other hand, if one component does not match, the final score depends solely on the other.

2.1.2 Statistical Tests

There are a number of works that treat each numerical column as a sample and use statistical tests to detect if two samples are drawn from the same

distributions [17], [22], [29], [30]. With one sample taken from a column in a knowledge graph, its label is known and can be assigned to columns that are not in the knowledge graph but have similar distributions. The process of statistical test often sets the null hypothesis to be that the two samples are drawn from the same distribution, then calculates a p-value that indicates how likely one can actually observe the given samples under the assumption that the null hypothesis is true. Finally the decision of whether the null hypothesis can be rejected is made based on the p-value. Kacprzak *et al.* [17] use Kolmogorov–Smirnov(KS) test to compare a query column with knowledge graph columns. Neumaier *et al.* [22] attempt to improve this by clustering the knowledge graph properties and breaking them into more refined "content". By having smaller knowledge graph columns, the size difference between a query column and a knowledge graph column may be reduced, leading to a more precise comparison. The KS test is shown to have better performance than other tests such as Welch's t-test and Mann-Whiteney U-test in the literature [22], [30]. It is a nonparametric test which compares the distance of two empirical distribution functions. Unlike Welch's t-test and Mann-Whiteney U-test, which assume the data to be normally distributed, the KS test does not make assumptions on the distribution of the data other than assuming the distribution is continuous. This is an advantage when applying the model to more general cases. The KS statistics D with respect to two distribution functions F_1 and F_2 is given as:

$$D = \underset{x}{sup}|F_1(x) - F_2(x)|, \tag{2.2}$$

where *sup* is the supremum of the set of distances between F_1 and F_2 across all x. Intuitively, it can be seen as the greatest value among all distances in the set. An illustration of KS statistics of two empirical distribution functions is shown in Figure 2.1. Not surprisingly, if two sets of numeric values are equally distributed, the statistics D converges to 0.

Since we are interested in finding the distance between a query column and KG properties, there are two approaches in the literature for obtaining a

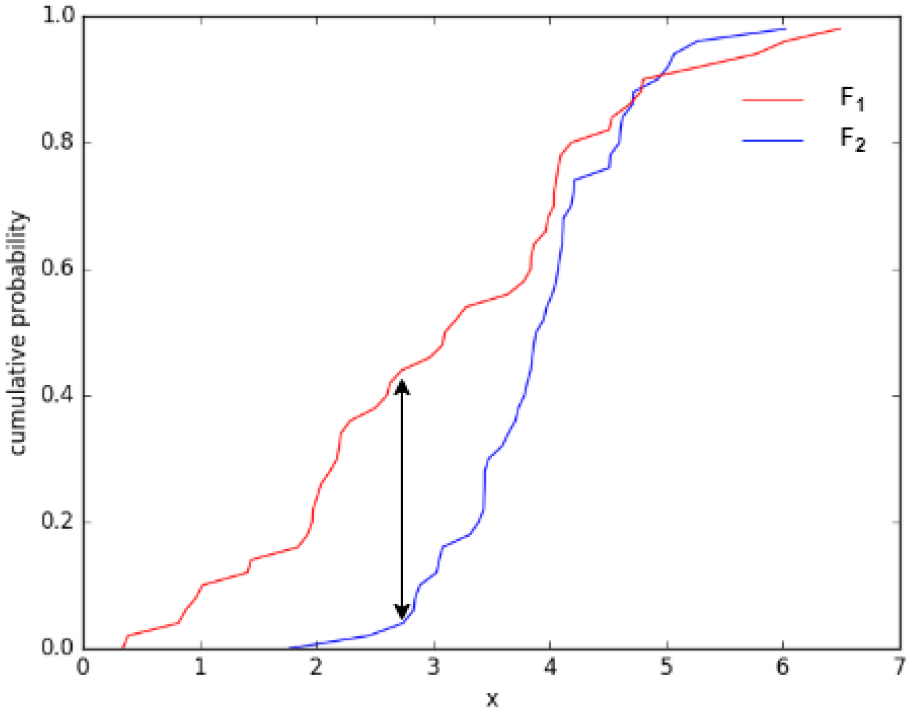

Figure 2.1: Illustration of KS statistics

distance measure from a statistical test result: one approach directly taking the p-value of the test as a normalized distance [17], [22], [30], whereas another approach converting the test result to a binary distance based on if the null hypothesis is rejected or not [29].

2.1.3 Clustering and Classification

The problem of finding a matched KG property can also be formulated into a classification problem, where we want to assign a class label (*i.e.* a property name) to the given query column. Since values in KG are associated with property names, an obvious approach would be treating each KG property as a cluster, under the assumption that values in different properties form separable clusters. Then a classifier can be applied to the query column to find the matched cluster. However, given that knowledge graphs are non-ideal in general, KG properties may not always form clusters with good quality. Another idea is to perform a clustering on the knowledge graph values in an

unsupervised manner in order to create clusters that are more effective in a classification. There are existing methods that first cluster the knowledge graph and then predict a matching cluster using a classifier. As briefly mentioned in Section 2.1.2, Neumaier *et al.* [22] cluster the knowledge graph using a rule-based hierarchical clustering algorithm. Values in the knowledge graph are clustered as raw data, where the label of KG property is not used in the clustering process. The clustering stops when the distances between all possible sub-clusters and the parent cluster are less than a threshold. They define the finest clusters as *"contents"*. Note that each *content* may contain values from multiple KG properties, and may be assigned multiple property labels. In addition to clustering, the algorithm also builds a type hierarchy on top of the clusters. The types of KG properties are extracted from sub-type relations in the knowledge graph. This will be the objects of the predicates, for example, "rdfs:subClassOf" in DBpedia and "P279" in WikiData. An example of the type hierarchy is shown in Figure 2.2. Given a query column, the algorithm determines the matching *content* using a KNN classifier, where the distance measurement used in KNN is the p-value returned by the statistical test. At the end a matched KG property is assigned to the query column by majority voting among all properties in the top k nearest *contents*.

Similarly, Alobaid *et al.* [1], [2] also cluster the knowledge graph, but they extract the typology features such as ordinal, nominal and interval-ratio, from each knowledge graph property as well as from the query column. Then they treat each KG property as a separate cluster and extract numerical features according to its topology. For the topology type of "categorical", the number of categories and percentage of each category are extracted as feature. For all other types, they compute *trimean* and *t-std* as features. The trimean of a cluster is computed as:

$$Trimean = \frac{Q1 + 2 * Q2 + Q3}{4}, \tag{2.3}$$

where $Q1$, $Q2$ and $Q3$ represent 25%, 50% and 75% quartiles respectively. The t-std refers to the standard deviation with trimean, which replaces mean with

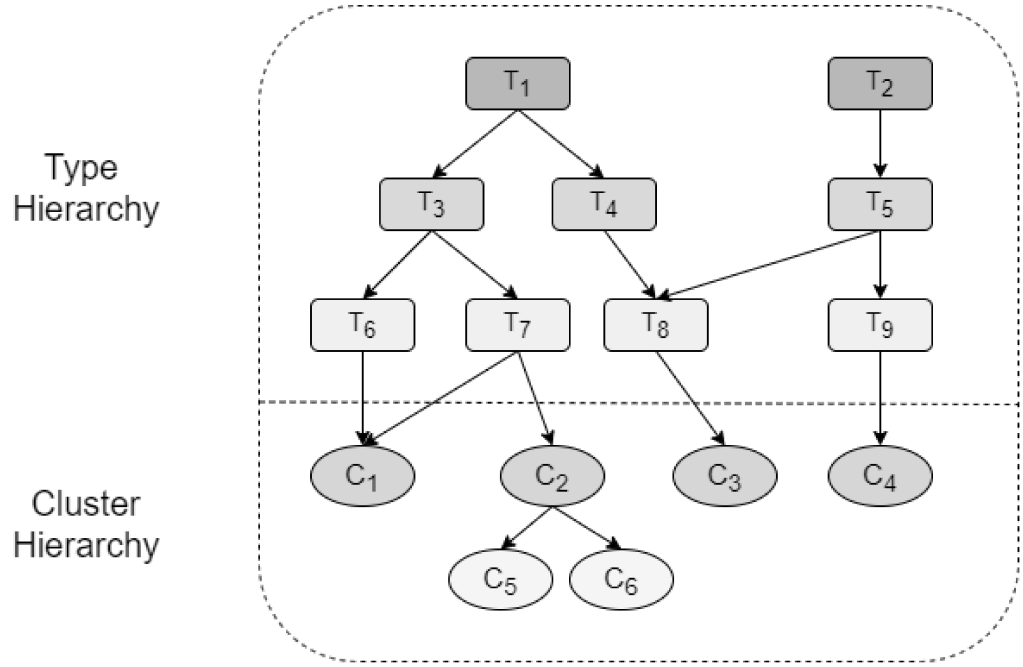

Figure 2.2: Structure of the complete hierarchy in Neumaier's method

trimean in the standard deviation formula:

$$t\text{-}std = \sqrt{\frac{\sum(x_i - trimean)^2}{N}} \tag{2.4}$$

Based on the extracted features it is possible to compute a two dimensional centroid for each cluster. Then the method looks for matched KG properties to the query column using the nearest neighbour search. The distance is set to be the Euclidean distance between the centroid of the query column and the centroid of KG properties.

An illustration of the label assignment in Neumaier *et al.* [22] and Alobaid *et al.* [2] in 2D space is shown in Figure 2.3.

2.2 Supervised Approaches

There are also supervised approaches for the task of type prediction [23], [24], [29], [34], [36]. Earlier works tried to create hand crafted features to train traditional supervised classifiers [29]. Recently, researchers have focused more on exploring the power of neural network and deep learning techniques [23], [24], [36].

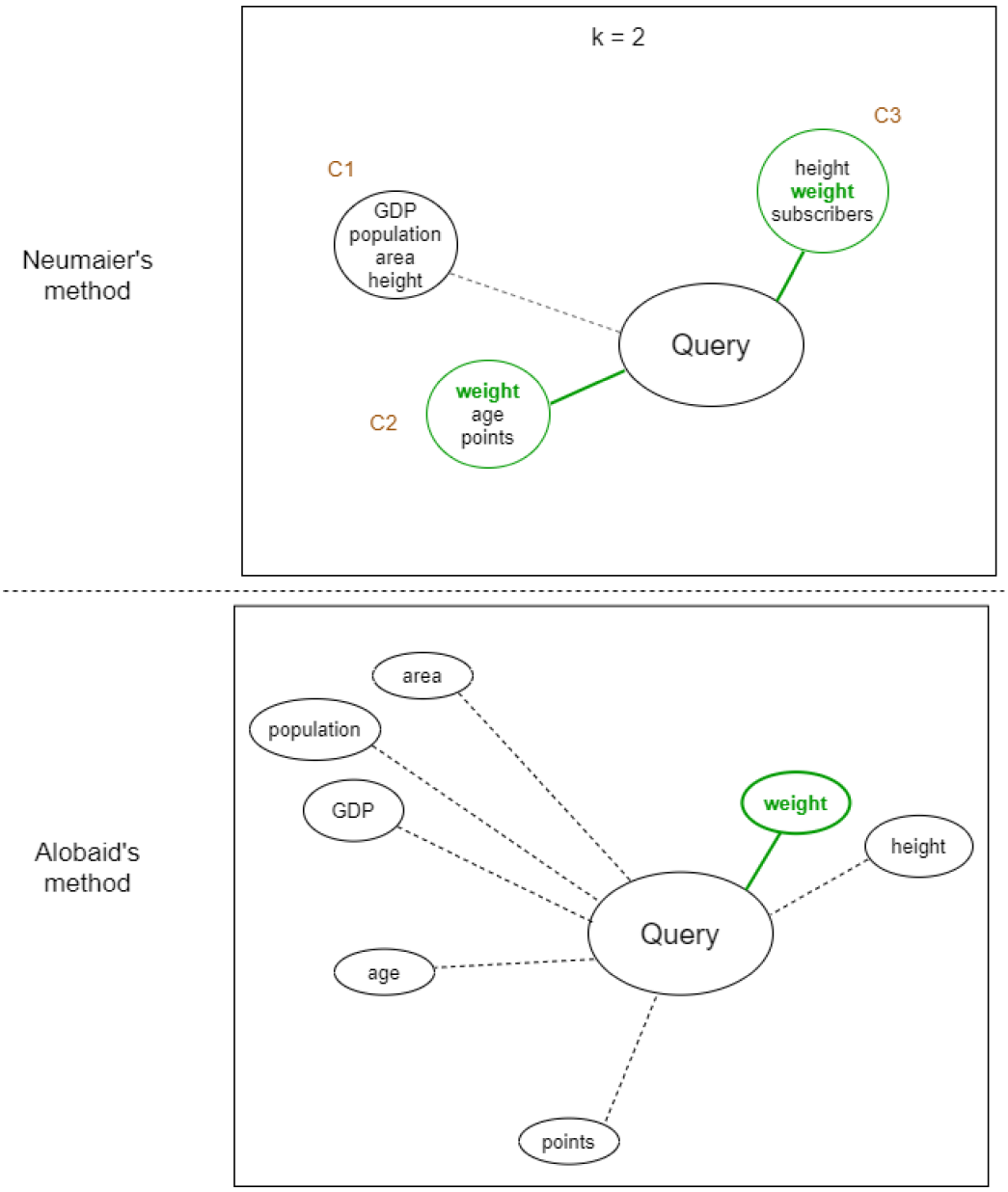

Figure 2.3: Illustration of prediction classifiers used in related works

Pham *et al.* [29] train binary classifiers including logistic regression and random forest based on the similarity between the query column and labeled columns. The features for classifiers include a variation of the Jaccard Similarity that captures the overlap range of two sets of numerical values, as well as a binary KS distance. The Jaccard Similarity is computed as follows:

$$NumJaccard(s_1, s_2) = \frac{min(max(s_1), max(s_2)) - max(min(s_1), min(s_2))}{max(max(s_1), max(s_2)) - min(min(s_1), min(s_2))}. \tag{2.5}$$

This score gives rough estimate of similarity based on the minimum and the maximum values of two sets of numbers. It is easy to compute but does not capture enough information and may under-estimate the similarity in some cases. Thus, the KS distance is used as a complementary input of the classifiers.

Prior machine learning algorithms such as Logistic Regression and SVM, suffer from curse of dimensionality due to their shallow structure, thus only work on relatively small feature sets [15], [18]. Recently a number of works adapt the deep learning framework which supports a significantly larger number of features and samples [15]. Deep learning is developed from artificial neural networks, where it adopts the hierarchical structures in layer connection and have multiple levels of representation [9], [20]. Each level of representation is transformed to the next level using a non-linear transformation, and a composition of enough such transformations is able to approximate very complex functions [20]. Deep learning has shown to have improvement in the processing of natural data and extracted features comparing to prior machine learning techniques [9], [20]. Next, we review some works that take a deep learning approach. Nguyen *et al.* [23] train a CNN network that uses various sampling techniques to construct features. They also utilize representation learning to generate column embedding which is then used as the input of the network. Recent work introduces Sherlock [15], which is a multi-layer feedforward neural network built for single column semantic type detection. It formulates the semantic annotation problem into a multi-class prediction where each se-

mantic type is represented as a class. It extracts three levels of embedding from column content including character embeddings, word embeddings and paragraph embeddings. These features are designed for textual columns. The character embeddings include for example, "fraction of values with numerical characters" and the count of special characters such as "-" and "/". The Word embedding is designed for characterizing the semantic content of the words. Words are mapped to high-dimensional fixed-length numeric vectors where the words with similar semantic content are "closer" in the multi-dimensional space. In Sherlock, the authors use pre-trained GloVe dictionary [28] where it contains 50-dimensional representations of 400,000 English words. Paragraph embeddings is able to convert a piece of text into a numeric vector, and is commonly used for document similarity [8]. In Sherlock, the authors use a pre-trained paragraph embedding proposed in [19], and they treat each column as a "paragraph" where values in each cell are "words". Global statistics such as mean and standard deviation are extracted as features for numeric columns. It also includes features like "column entropy", which determines how uniformly values are distributed in a column. The size of the numeric features is 27 in total. Zhang *et al.* [36] adopt the structure from Sherlock, and improve it by adding cross-column influence in their prediction model. They use the same feature set as Sherlock, but the three levels of embedding features are compressed to dense feature vectors with 1560 dimensions, and the global statistic features with 27 dimensions are concatenated to the embedding features. This produce a vector of 1587 dimensions in total. They have also used the batch normalization technique to process the features. It is proposed by Ioffe *et al.* [16] and is used for accelerating the training of the deep networks. An overview of features and the network structure used in [15], [36] is shown in Figure 2.4.

2.3 Annotating Entity Columns

The large body of work on annotating web tables has focused on textual and categorical columns, and a popular approach has been utilizing the power of a

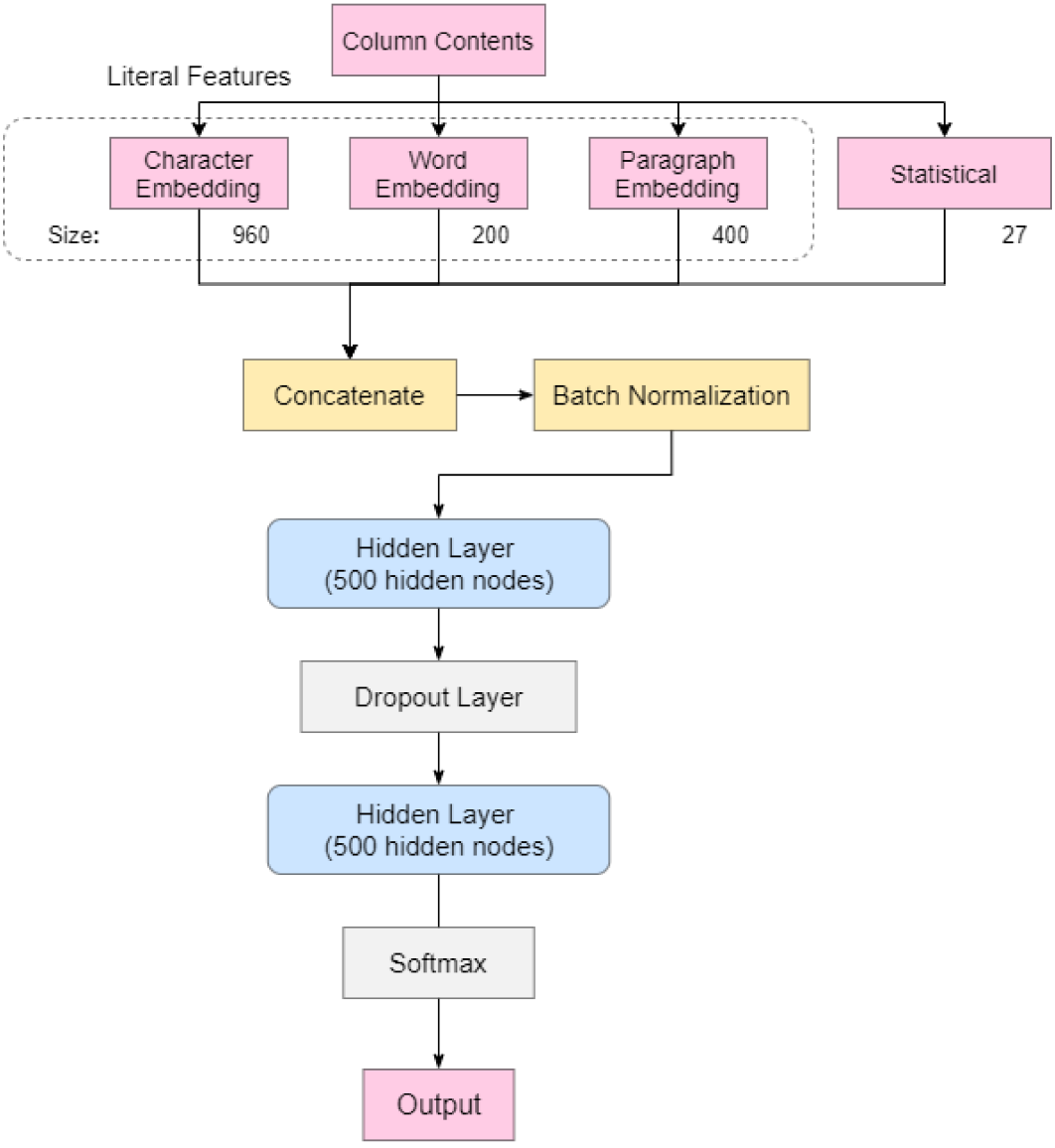

Figure 2.4: Architecture of the network used in Sherlock and Sato

knowledge graph for the annotation. These works are not directly related to our problem, but they provide valuable ideas and insights that may be adapted to our problem.

A general approach has been mapping a column to an entry in the knowledge graph [6], [10], [37]. Chen *et al.* [6] do this by finding an embedding of each entity and classifying the query column to a knowledge graph class. The embedding of a column or a class here is a matrix which is fed into a CNN network for classification. Alternatively, Efthymiou *et al.* [10] consider cell level matching, which assigns a property label to each cell, and use the majority vote to predict a semantic label for the query column.

To better model the relationship between different columns with the same or similar semantic labels, several works have modeled a corpus of tables using a probabilistic graph model [21], [34]. Limaye *et al.* [21] build a probabilistic graphical model that describes the type similarity between columns in an annotated dataset and assign a label to a query column that maximizes the joint probability of matching each cell text to an entity label and each column header to an entity type.

2.4 Unit Extractor

There is also some work on unit labeling, which is relevant to ours. Sarawagi *et al.* [31] propose a pattern-based approach to extract units from column headers. They compile a set of units from a knowledge graph, and train a rule based unit extractor that is based on a Context Free Grammar (CFG). A subset of their grammar rules are shown in Figure 2.5. As each header can be parsed using multiple grammar rules, a discriminator scoring function is learned. They also consider relative-frequency and co-occurrence statistics in the corpus of tables to disambiguate the extracted units.

2.5 Limitations

Many of the KG matching-based approaches use adhoc features and heuristics (e.g. topology features in Alobaid *et al.* [2] and the clustering method

```
Multiplier      :=      multipliers extracted from KG
AtomUnit        :=      atom units extracted from KG
Operator        :=      Empty  OR  'per'  OR  '/'  OR  '*'
Separator       :=      Empty  OR  'of'  OR  'in'
SimpleUnit      :=      AtomUnit  OR  Multiplier + AtomUnit
CompoundUnit    :=
    SimpleUnit  OR
    SimpleUnit + Operator + SimpleUnit
ComplexUnit     :=
    CompoundUnit  OR
    Multiplier + Separator + CompoundUnit OR
    CompoundUnit + Separator + Multiplier
```

Figure 2.5: Example grammar used by CFG

in in [22]), which are subjective and may not generalize well. In addition, Alobaid *et al.* [2] classifies all columns with decimal value as *"others"* without further refinement. As a result these columns cannot benefit from the topology features and could potentially become too general. Moreover, the accuracy of statistical tests generally drops when the values in a query column do not represent the entire distribution. This will happen for example, when the query column maps to only a small region in a knowledge graph column.

The score-based measurement proposed in [13] also has limitations. This method ignores the numeric values in a column and only focuses on the column metadata, which means the method assumes the metadata is reliable. The column metadata such as header is often unreliable or missing. Also if the column header can be assumed to be reliable, there is not much need for annotating the column. Additionally, the scoring function does not penalize well for mis-matches. For example, if column title is completely different than an ontology, having a matched "no unit" could still dominate the final score. However, only matching "no unit" may not be a strong enough evidence.

As for the unit extractor, there are two main issues. First, the unit string is expected to appear in column headers whereas our analyzes of a real web table dataset shows that only less than 20% of the numeric columns contain unit strings, which means that this method is not applicable for more than 80% of the columns. Second, the method relies on identifiers to detect compound units. For example, this method fails to extract the unit *"Newton-metre"*, written as *"Nm"*, which is a compound unit with a hidden multiplication symbol in the middle. This method recognizes it as *"Nanometre"*.

Chapter 3

Methodology

Our approach for annotating numerical columns makes use of a knowledge graph (KG) for semantic labels. Under *the closed world assumption*, the knowledge graph is treated as "complete," and any semantic type that does not exist in the KG is deemed not to exist. The same or a similar assumption is made in many approaches on linking and disambiguating entities mentioned in text [7], [14], [32] and web tables [3] as well as in approaches for annotating tabular data [6], [10], [37].

3.1 System Architecture

As shown in Figure 3.1, our system architecture mainly consists of two components: (1) compiling candidate labels, and (2) mapping query columns. Compiling candidate labels can be done offline as it is usually a one-time process, and it can be done either at the beginning or after the knowledge graph has gone through a major change. We discuss these components in more details next.

3.2 Compiling Candidate Labels

A source for compiling candidate labels is public knowledge graphs (such as Wikidata, Freebase, Dbpedia), with their structured content about many entities. Many of these knowledge graphs keep various quantitative information about entities. Information about each entity is often stored as a set of state-

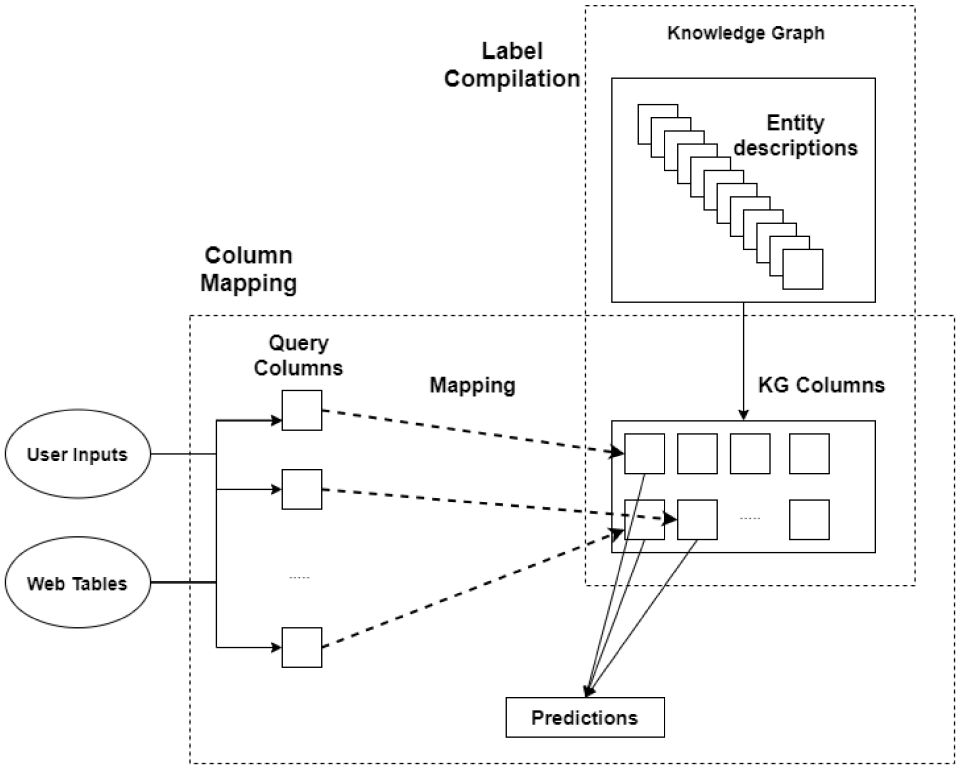

Figure 3.1: System architecture

ments, also known as triples. Each triple is typically in the form of $<s$, pd, $o>$ where s, pd and o denote subject, predicate and object respectively. Each entity is an instance of one or more *semantic type* (also known as *class*); for example Canada is an instance of *country* and Barack Obama is an instance of *human*. Each entity can also have a set of *properties* or *attributes*. For example, Canada's *population* is $35,158,300$ and Canada's *median income* is $70,336$ Canadian dollars.

Each numerical quantity in a knowledge graph may be labelled with all attributes or properties it represent. For example, $35,158,300$ may be tagged with population. However, an attribute name alone does not always provide a descriptive label since different entity types can have the same attribute name. For example, height may describe the height of a person, a building, or a car, and the valid ranges can be different depending on the entity type they are describing. Also many quantities are associated with a unit. For example, 178 centimeters may represent a person height but 178 meters will not. With these

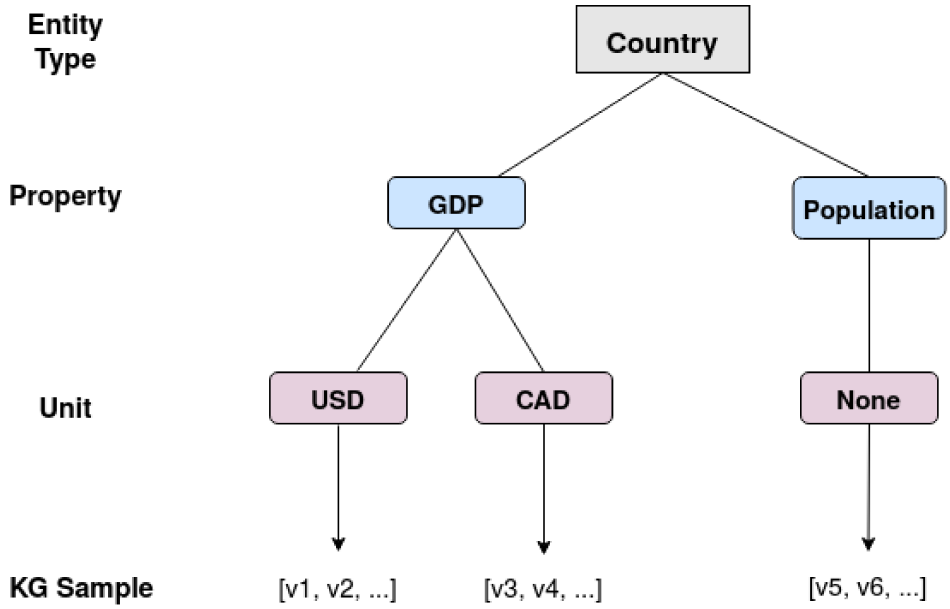

Figure 3.2: Examples of semantic labels (properties) for *country*

observations, a semantic type for numerical columns is defined as follows.

Definition 6 (Semantic numeric type)**.** *Given a set of entity types T, properties P and units U, the semantic type of a numerical column is a triple $<t, p, u>$ where $t \in T$, $p \in P$ and $u \in U$.*

The algorithm for compiling the candidate types from a knowledge graph may look like this. First, any entity-attribute pair that takes a numerical value is a candidate for our type system and is extracted. Second, since numerical attributes are usually assigned to entities, whereas our type system assigns attributes to entity types, we push the attributes of each entity to its entity types. For example, from the statements "Canada *median income* is $70,336$ CAD" and "Canada is an instance of country," we know that the entity type country can have an attribute called *median income* and *CAD* is a possible unit. Hence (country,"median income","CAD") can be added to our semantic type [1].

We also aim at collecting a representative sample for each semantic type, and a knowledge graph may provide one such sample. For example, 178 may be

[1]In Wikidata, the predicates for "is instance of" and "recommended unit of measurement" are P31 and P8111 respectively.

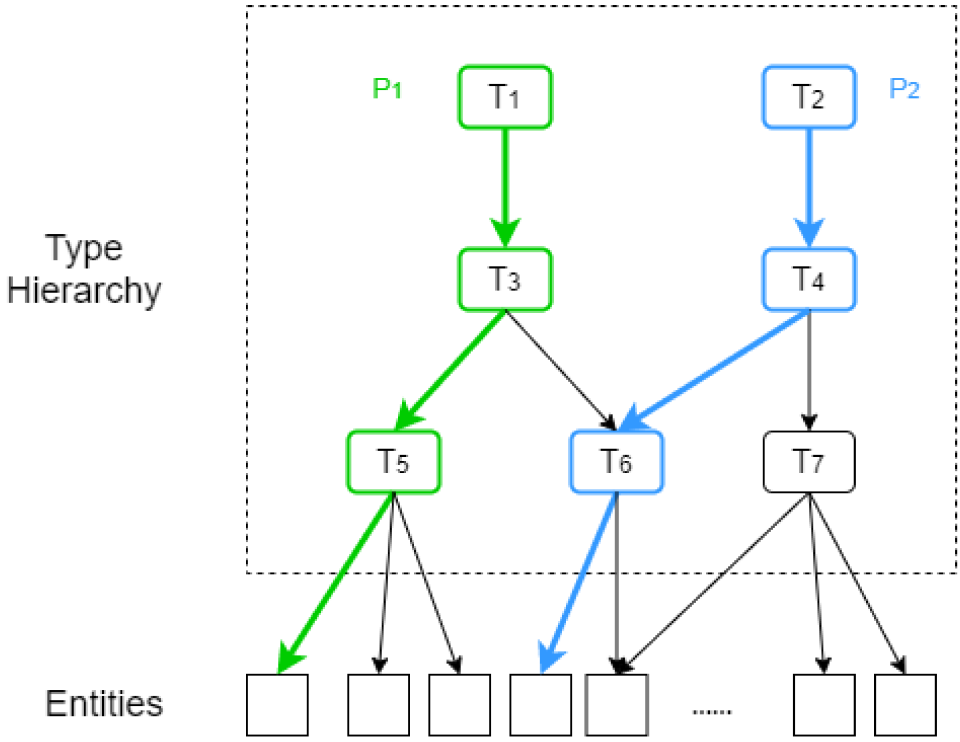

Figure 3.3: Entity type hierarchy

treated as a sample data point for semantic type (human,height,centimeters) and $70,336$ may be a sample point for type (country,"median income","CAD"). Figure 3.2 shows some semantic candidate labels or properties for entity type country and a sample for each candidate.

Sometimes an entity inherits properties from a super type, and knowing the type hierarchy may help with resolving the query columns. Hence, the set of semantic types may be expanded by considering the subclass relationships between entities. For example, Microsoft is an instance of "software company" and the latter is a subclass of "business," "company" and "technology company." The subclass relationship may be described in terms of a directed graph (as shown in Figures 3.3 and 3.4) where each entity type is a node, and a directed edge from node u to node v indicates that v is a sub-type of u. Our column mapping, as discussed in the next section, makes use of this hierarchy in assigning candidate labels.

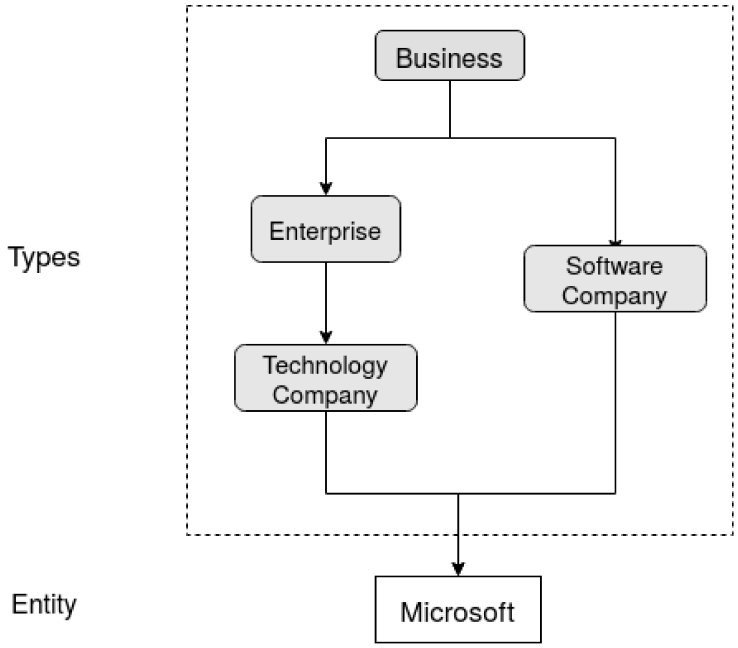

Figure 3.4: Type hierarchy for entity "Microsoft"

3.3 Column Mapping

Given a query table with some numeric columns and a set of candidate property labels, for example, from a knowledge graph, we want to assign each numeric column in the query table a property label that best describes the column. If we treat each property or a column a random variable that follows a distribution, we want to detect if two random variables are providing the same measurements hence they are expected to have the same distributions.

When two random variables approximately follow a normal distribution, the null hypothesis that they have the same distribution may be rejected using a statistical t-test. The assumption on the normal distribution may be relaxed if we replace the t-test with a comparison of cumulative distributions, known as KS test. This is also a test that is used in the literature [17], [22] and is also reported as a baseline in our experimental evaluation. However, a statistical test is neither sufficient nor necessary to detect if two random variables are providing the same measurements and should have the same labels. When the null hypothesis is not rejected, it does not mean that the two variables can be assigned the same labels. Two columns can have the same distributions

but can describe totally different properties. It is not also necessary for two columns that describe the same property to have the same distributions. For example, age can be different for people living in different countries or even the same country and in different times. Hence the null hypothesis may be rejected but the two columns may still have the same labels.

We treat column mapping as a cost optimization, where each mapping of a candidate column is associated with a cost and we want to find a mapping with the least cost. Assuming that the true label of the query column has a corresponding property in the knowledge graph, we expect one such property can be identified through a cost optimization. However, false mappings are also expected especially when there is a large number of candidates. Since each property is usually associated with an entity type (e.g. the population of a *country* or the age of a *person*), one way to avoid or reduce the number of false mappings is to detect the type of the query entity column first. For example, knowing that a query column describes countries, the set of candidate labels is limited to properties that can be assigned to a country.

3.3.1 Entity Type Detection

If the given numeric column is associated with an entity column, we want to detect the semantic type of that entity column. This can be seen as another type annotation task, which is outside the scope of this work. However, the problem is studied in the literature [6], [10], [21], and several methods have been proposed; any of those methods can be readily used here. In this paper, we apply a variation of majority voting, where each cell content of the entity column in the query table is mapped to an entity in the knowledge graph, the entity type(s) of the matching entities in the knowledge graph are looked up, and the entity type that covers the largest number of entities in the query column is selected as the entity type.

Bounding the type of an entity column in a query table reduces the number of candidate mappings of other columns but it does not limit the set of candidates only to the direct properties of the entity type. For example, a query table describing *basketball player* may have attributes that belong to the entity

type *person*, and limiting the candidates to the direct properties of basketball player will miss those properties. To avoid such cases, we want to include, as candidate semantic types, those properties that are associated with an entity through the type hierarchy. For this, we want to check for each pair of entity types S_i and S_j in the type hierarchy if there exists a sub-type relationship in the form of a path from S_i to S_j. If such path exists, we know S_j is the more specific type and will inherit the properties of its super-type S_i. All such paths in the type hierarchy from the root to an entity (as shown in Figure 3.3) are candidate types that can be assigned to the entity. The set of candidate paths for all query column entities can be ranked based on their frequency and the most frequent path can be selected as the query column type.

A caveat in detecting the entity type is that entity mentions in a query table can belong to many different types, and the most frequent candidate type may not have the majority vote. As a solution, we may set a threshold τ based on the size of the column, and if a candidate type has a frequency less than τ, we may conclude that entities in this particular column do not agree on a type label.

3.3.2 Cost Optimization

We want to find out if a given query column is sampled from the domain of a knowledge graph property (i.e. the set of values it takes), but we cannot always expect overlaps between numerical sets. More often, numbers that describe the same property are close in terms of the absolute value or statistical measures. Statistical measures look at the overall distribution and are more accurate when the sample size is sufficiently large and the knowledge graph is near-complete (i.e., it include the set of all values it can take). Under more realistic settings where query columns are small and knowledge graph properties are incomplete, a more effective measurement is desired. We formulate the problem as a bipartite matching between two bags representing the query column q and a knowledge graph property c, and focus on the micro-level similarity, which is the differences in element-wise matching. With the distance between mapped values considered as a dis-similarity measure, our goal is to

minimize the sum of the distances in the mapping. If we represent every element in q and c with a vertex, let V_q denote the vertices of q and V_c denotes the vertices of c. Consider a complete edge-weighted bipartite graph formed between V_q and V_c, and let $Cost(u, v)$ be the cost of mapping u to v in terms of their distance. We want to find a mapping $M \subseteq (V_q \times V_c)$ where the total cost $\sum_{(u,v)\in M} Cost(u, v)$ is minimized and M covers all vertices of V_q.

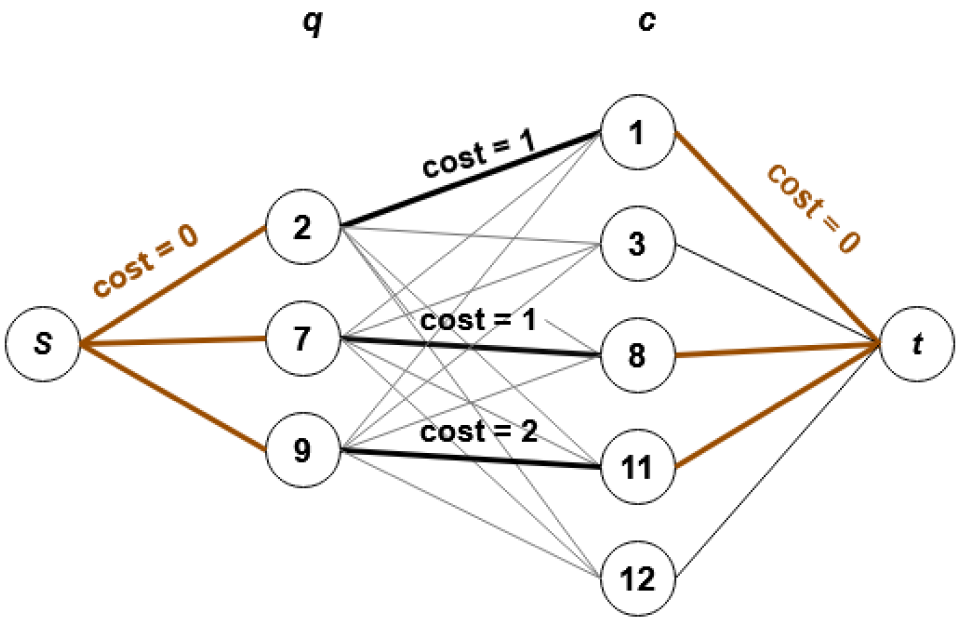

Figure 3.5: Illustration of a mapping between q and c

The matching problem can be cast as a minimum cost flow, for which efficient algorithms exist [12], [26]. We want to transfer $|V_q|$ units from a source s to a target t. As shown in Figure 3.5, there are edges between s and all nodes of V_q, between t and all nodes of V_c, and between every node of V_q and V_c. Each edge is assigned both a capacity and a cost. The capacity of all edges are set to one and the cost is defined as follows:

$$Cost(u, v) = \begin{cases} dist(val(u), val(v)), & \text{for } u \in V_q \text{ and } v \in V_c \\ 0, & \text{for } u = s \text{ or } v = t, \end{cases} \tag{3.1}$$

where $val(u)$ and $val(v)$ are the numerical values represented by nodes u and v respectively, and $dist$ is a function that can be set to any valid distance

measure between two numbers. In this paper, the absolute difference is used as the distance function.

If E denotes the set of all edges, we want to find a flow $f : E \to \{0, 1\}$ where the total cost

$$Cost(q, c) = \sum_{(u,v)\in f} Cost(u, v) \tag{3.2}$$

is minimized subject to $\sum_{e\in E} f(e) = |V_q|$.

3.4 Label Prediction

A label predication algorithm will need to compare the query column q to all candidate columns C in the knowledge graph, following the cost optimization as discussed above, and will return top-k nearest neighbours, in terms of $Cost(q, c)$ for $c \in C$, as possible types. Each such type assigns a *property* and a *unit* to the query column. If the query column is associated with an entity type, the number of candidate types may be reduced. For example, if an entity has type *"human"*, it is likely to have properties such as birth year, height and weight. On the other hand, it is unlikely that it has property *GDP*. Hence, for a given query table, the entity type may first be detected (as discussed in Section 3.3.1), and only properties of the detected entity type are considered as candidate types.

As comparing a query column to all semantic types in a knowledge graph or the semantic types that are associated with an entity type can be costly, we next discuss a few strategies to reduce the cost in Section 3.5.

3.5 Running Time Improvements

An efficient algorithm for the minimum cost flow problem is the network simplex algorithm of Orlin [26], which runs in $O(min(n^2m^2logn,\ n^2mlog(nC)))$ where n is the number of nodes in the network, m is the number of edges and C is the maximum edge cost. For small integer edge costs, which is usually the case for numerical columns, the algorithm runs in $O(n^2m)$. The algorithm can

be expensive for large query and knowledge graph columns, hence one wants to effectively reduce the running time. We next introduce a few techniques to reduce both the search space and the running time.

3.5.1 Reducing the Size of the Query Column

Under an injective mapping, every element of the query column q is mapped to a distinct element of c, hence it is expected that $|V_q| \leq |V_c|$. In more realistic settings, the knowledge graph is rarely complete and an injective mapping may not be possible for large query columns. At the same time, mapping large query columns can be costly since the time increases quadratically with the number of nodes. One strategy to reduce the cost of mapping is to reduce the size of V_q. For example, if the size of V_q exceeds a limit l, one may select a random sample of size l for the mapping. Our hypothesis is that the query column can be sampled and that mapping a sample can provide a good estimate of the semantic label if the sample is large enough. In our experiments, we evaluate different sample sizes and how both the accuracy and the running time are affected.

3.5.2 Pruning Candidate Columns

Finding a mapping between a query column and every candidate column can be costly when there are a large number of candidates, and a question is if this cost can be reduced without affecting the correctness of the algorithm. The idea here is that a full costly mapping may be avoided when the mapping does not provide a better solution than what is already computed. Suppose we are are interested in top k mappings with the least cost. If there is evidence that a candidate column cannot have a cost less than what we have already seen, there is no need to perform a full mapping. For our pruning, we introduce two lower bounds.

Lemma 1 (LB_1)**.** *Consider two columns q and c with non-overlapping ranges, and let q_{min} and q_{max} be respectively the largest and smallest elements in q and c_{min} and c_{max} be respectively the largest and smallest elements in c. There are*

two cases (as shown in Figure 3.6).
(a) If $q_{max} \leq c_{min}$, then $dist(q_{max}, c_{min}) * |q|$ is a lower bound on the cost of the mapping, i.e. $Cost(q, c) \geq dist(q_{max}, c_{min}) * |q|$.
(b) If $q_{min} \geq c_{max}$, then $dist(c_{max}, q_{min}) * |q|$ is a lower bound on the cost of the mapping, i.e. $Cost(q, c) \geq dist(c_{max}, q_{min}) * |q|$.

Proof. Assume $q_{max} \leq c_{min}$, then

$$Cost(q, c) \leq \sum_{q_i \in q} |q_i - c_{min}|, \text{ under a non-injective mapping and}$$
$$Cost(q, c) = \sum_{q_i \in q} |q_i - c_{min}| \text{ under a injective mapping.}$$
$$\text{Since } q_i \leq q_{max},$$
$$|q_i - c_{min}| \geq |q_{max} - c_{min}|,$$
$$Cost(q, c) = \sum_{q_i \in q} |q_i - c_{min}| \geq |q| * |q_{max} - c_{min}| = LB_1,$$

which implies LB_1 is a lower bound of $Cost(q, c)$.
The proof is similar when $q_{min} \geq c_{max}$ in Case (b). □

The lower bound is pretty easy to compute especially if the minimum and the maximum values of each column are computed in advance. Our next lower bound provides a tighter bound and is more accurate.

Lemma 2 (LB_2)**.** *Consider two columns q and c and let M_c be a mapping between q and c where each element of q is mapped to its closest element in c with no constraint on the number of query elements that can be mapped to c. Then $Cost(q, c) \geq \sum_{e \in M_c} Cost(e)$.*

Proof. We need to show:

(1) LB_2 is a lower bound. This is equivalent to show that $LB_2 \leq Cost(q, c)$;

(2) LB_2 is a tighter lower bound than LB_1. In other words, $LB_2 \geq LB_1$.

Proof of (1):

Denote with M_1 the actual mapping between q and c and M_2 the mapping under LB_2. Then $cost(M_2) = LB_2$. Suppose the claim of the lemma does not hold and this means that $LB_2 > Cost(M_1)$ and $\sum_{e_i \in M_2} Cost(e_i) >$

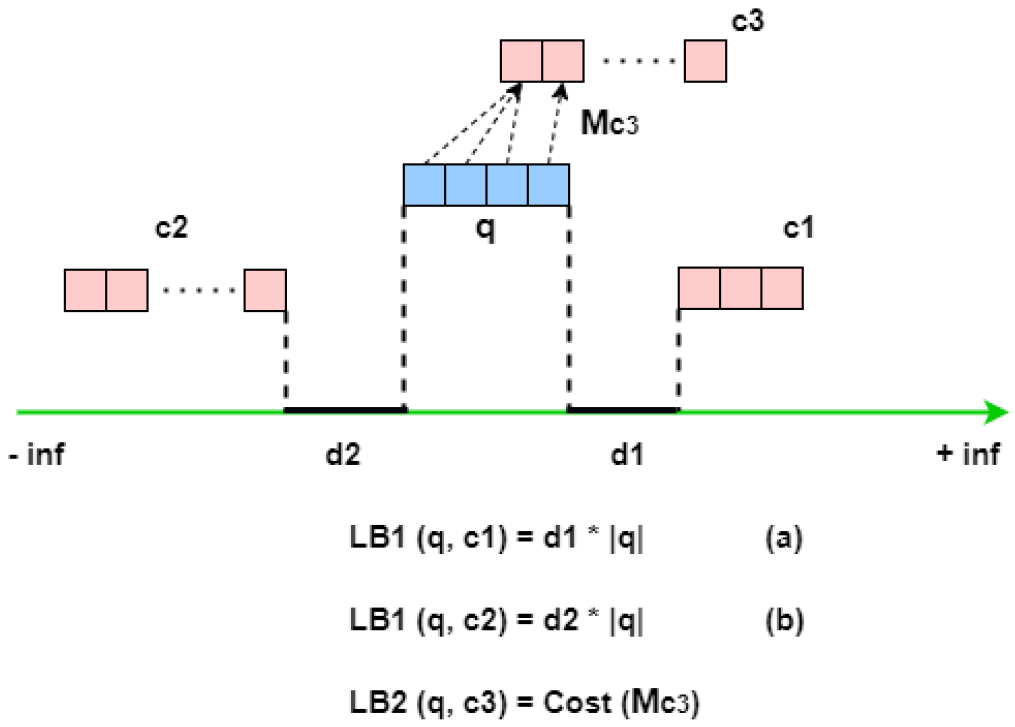

Figure 3.6: Illustration of the lower bounds LB_1 and LB_2

$\sum_{e_j \in M_1} Cost(e_j)$. Consider mapping an element $u \in q$. Based on the definition of LB_2, we have $Cost(e_i) = \min_{v \in c} Cost(u, v)$, whereas the actual cost can be larger since not all values in q can be mapped to the same value in c. This implies that $Cost(e_i) \leq Cost(e_j)$, and that $\sum_{e_i \in M_2} Cost(e_i) \leq \sum_{e_j \in M_1} Cost(e_j)$.

(2) It is easy to see that a comparison between LB_1 and LB_2 is valid only when LB_1 can be computed, which are Cases (a) and (b) in Lemma 1. Assuming $q_{max} \leq c_{min}$, it is obvious that $LB_2 = \sum_{q_i \in q} |q_i - c_{min}|$. In Lemma 1 we already proved that $\sum_{q_i \in q} |q_i - c_{min}| \geq LB_1$, therefore $LB_2 \geq LB_1$. □

The lower bound can be efficiently computed by searching for every element in q its closest element in c. It is easy to see that LB_2 provides a tighter lower bound than LB_1, and LB_2 is applicable in many cases where LB_1 may not not be applicable.

To apply these lower bounds, one may sort the candidate columns based on LB_1. Since we are interested in top k mappings, we may construct a priority queue pq that holds top k candidates so far. To process each new candidate

column in the ordered list of candidates, we compute LB_2 for the candidate column and prune the column if LB_2 is greater than or equal to the largest cost in pq. Otherwise we compute the exact cost of the column and push the column to pq if the exact cost is less than the cost of a column in the queue. The algorithm stops when the next column has a lower bound that is greater than the cost of all columns in the queue. Algorithm 1 gives the steps of our column matching including the steps of pruning.

Algorithm 1 Column Matching with Pruning

```
1: q ← query column
2: C ← sorted candidate columns based on LB_1
3: topK ← initialized priority queue
4: i ← 0
5: while LB_1(C[i]) < max(topK) do
6:     if LB_2(C[i]) < max(topK) then
7:         M ← cost of mapping (q, C[i])
8:         if M < max(topK) then
9:             push C[i] to topK
10:    i ← i + 1
11: return topK
```

Chapter 4

Evaluation

This section presents an experimental evaluation of our algorithms and pruning strategies under different parameter settings and in comparison with competitors from the literature. After discussing our evaluation setup, Section 4.2 compares the performance of our proposed method with the existing approaches [2], [31], [36] that are considered state-of-the-art in semantic and unit labeling, Section 4.3 investigates the choice of parameters and how the performance of our method is affected under different settings, and Section 4.4 evaluates the effectiveness of our pruning strategies in reducing the search space.

Dataset	Numerical columns	Entity types	Semantic labels	Unit labels	Average query size
WDC	133	27	22	3	27.49
Wikitables	262	96	77	43	20.97

Table 4.1: Statistics of the testing corpus

4.1 Evaluation Setup

The corpus of web tables used in our evaluation was extracted from two sources: (1) wikiTables [4], which contains 1.9 million tables extracted from Wikipedia, and WDC table corpus 2015 where tables are extracted from Common Crawl [1].

[1] `http://data.dws.informatik.uni-mannheim.de/webtables/2015-07/relationalCorpus/compressed/`

We randomly selected 300 numeric columns from each source, and for each numeric column, we also extracted the entity or subject column from the same table (when present) and the column headers (if any) [2]. This resulted in 300 pairs of entity-numeric column pairs, which were manually annotated with entity type for the entity column and the semantic and unit labels for the numeric column. The annotation was done by two individuals, in a peer review manner, and the column pairs that received the same labels by both annotators were selected as our testing corpus. The two annotators were responsible for providing an entity type, a semantic label as well as a unit label for each query pair given, and they could use column headers and/or search the knowledge graph and other online sources to assist them in their annotations. Some statistics of our testing corpus is given in Table 4.1. For example, our annotators agreed on the labels of 262 columns from Wikitables, which belonged to 96 different entity types, and 133 columns from WDC, which belonged to 27 different entity types. The rest of the columns either could not be annotated or an agreement was not reached. Also not many unit labels could be assigned by our annotators to the columns from the WDC corpus. For the same reson, we evaluate the task of unit annotation on WikiTables dataset only, whereas semantic annotation is evaluated on both datasets.

We used Wikidata as our knowledge graph, and extracted *type relation* from the database dump as of June 2020 [3]. We excluded knowledge graph properties that only contained a single numerical value. Such properties may not construct useful mappings since values in the query column all maps to that single value. Besides, removing those properties reduces the number of KG properties by 50%, which effectively reduce the search space. The statistics of the extracted types are shown in Table 4.2.

The performance of our algorithms was measured in terms of precision, i.e. the fraction of annotated query columns that are correctly labeled, and recall, i.e. the fraction of all query columns that are correctly labeled. We report

[2]The column header was not used in our method but it was used by some of our competitors as well as by our annotators when assigning the labels.

[3]`https://www.wikidata.org/wiki/Wikidata:Database_download`

Entity types	Properties	Units	(Entity,Property,Unit) triples
473	13149	640	17617

Table 4.2: Statistics of the extracted types from Wikidata

the precision at k and the F1 score, consistent with the reported results of our competitors [2], [31].

For our performance comparison on semantic labelling, we compare our algorithm with two baseline methods: TTLA [2] and the Kolmogorov Smirnov (KS) test which is used in a few related work [17], [22], [29]. We rank the candidate columns based on the p-value returned by the KS test and select top k results. The algorithm of Alobaid et al. [2] requires the entity type as input, which is not the case for our method. In our comparison with Alobaid et al., we provide all methods with the entity type label for a fair comparison. However, we also separately evaluate the performance of our method without providing the entity type label. For a performance comparison on unit labeling, we compare our algorithm with UnitTagger [31]. Since our method and UnitTagger rely on different column components (with UnitTagger using the column header and our approach using the column content), a comparison under an exact same setting is not possible. On the other hand, to the best of our knowledge, there is no prior work that annotate the unit label based on the content of a column. Therefore we propose a setting that aims at finding out how much our method can achieve. For this, we report the number of columns that have the unit string in the header, and for which UnitTagger is applicable. We test our method on the same data without using the column headers, and report the precision of both methods. Our method is clearly orthogonal to UnitTagger and the two methods can be combined for a better performance.

4.2 Performance Comparison

Tables 4.3 and 4.4 show the performance of our semantic labeling, in terms of precision and the F1 score, compared to the state-of-the-art baselines. For

the first three methods (i.e., TTLA, KS-Test, Proposed), the performance is reported with the entity type provided. The performance of our method is also reported without providing the entity type; this is denoted as Proposed-w/o. In our entity type detection, we set the parameter τ to $50\% * |q|$, to indicate that a majority vote is needed for an entity type to be selected.

4.2.1 Results on Semantic Labeling

	Method	Annotated	Precision@k	F1@k
k = 1	TTLA	168	0.107	0.135
	KS-Test	131	0.099	0.094
	Proposed	131	**0.221**	**0.199**
	Proposed-w/o	95	0.137	0.094
k = 3	TTLA	168	0.155	0.180
	KS-Test	131	0.160	0.128
	Proposed	131	**0.351**	**0.298**
	Proposed-w/o	95	0.284	0.186
k = 5	TTLA	168	0.179	0.205
	KS-Test	131	0.244	0.217
	Proposed	131	**0.397**	**0.331**
	Proposed-w/o	95	0.357	0.230

Table 4.3: Performance of semantic labeling on Wikitables with 262 columns compared to our baselines

The results in Tables 4.3 and 4.4 show that our approach achieves a higher precision and F1 score than TTLA and KS-test on both datasets. For top three labels (k=3), the precision of our algorithm on Wikitables is more than twice that of our baselines, and a similar pattern is seen in terms of the F1 score as well. On WDC, our algorithm also performs better than our baselines but the gap is not as wide as the gap on Wikitables. The difference in performance between the two datasets may be due to the use of Wikidata as our knowledge graph, which is expected to have more overlapping content with wikitables than WDC. With a big gap between our approach and KS-test, it is clear that a statistical testing for real web table columns that are not large (the average column size for Wikitables was 21 and for WDC was 27) is less effective, whereas our mapping-based approach works much better on the same datasets.

	Method	Annotated	Precision@k	F1@k
k = 1	TTLA	67	0.074	0.072
	KS-Test	101	0.069	0.100
	Proposed	101	**0.119**	**0.165**
	Proposed-w/o	72	0.083	0.086
k = 3	TTLA	67	0.209	0.194
	KS-Test	101	0.129	0.178
	Proposed	101	**0.218**	**0.284**
	Proposed-w/o	72	0.139	0.140
k = 5	TTLA	67	0.209	0.194
	KS-Test	101	**0.277**	**0.347**
	Proposed	101	**0.277**	**0.347**
	Proposed-w/o	72	0.263	0.250

Table 4.4: Performance of semantic labeling on WDC with 133 columns compared to our baselines

The performance of our method drops with an entity type detection module (denoted as proposed-w/o) due to any possible noise that may be introduced, but the results are still comparable to those of our baselines despite the fact that our approach has to detect the entity type whereas the baselines are provided with a correct entity type.

dataset	Total	Annotated	Precision@1
WikiTables	262	258	0.159
WDC	131	127	0.300

Table 4.5: Performance of SATO on semantic labeling

We also try to compare our method with recent supervised approach. The result in Table 4.5 shows the performance of SATO [36] on our testing dataset. We use their pre-trained model in this experiment. SATO does not match columns to knowledge graph properties, thus exact matches to the annotated label are not possible. We manually check the label returned by SATO and compare with our annotated label. Results show that under this setting, the performance of SATO is not satisfactory. It reaches a precision of 0.159 in WikiTable dataset and 0.3 on WDC dataset. By looking further into the results, we found that for columns only containing integer values, SATO predict all of them as "rank". We also observed that SATO does excellent in predict-

ing rank and year columns. This explains why performance on WDC dataset is better: in WDC dataset 13.9% of the columns are rank or year, and the proportion is higher than what is in for WikiTables. In our setting each query column is independent from others, where SATO cannot take advantage of cross-column relations. This could be another reason it is under-performing.

4.2.2 Results on Unit Labeling

In our Wikitables collection of 262 columns, our annotators found only 110 columns with a clear unit indicator in the column headers, and the rest of the columns had no units. The no-unit columns included properties with no associated unit (e.g. "population") as well as cases where the unit was missing due to incomplete headers. We did not distinguish between the two cases and both were removed from our unit labelling evaluation.

Our algorithm applied to the remaining 110 columns with known unit labels resulted in 25 correct labels, which translates to a precision of 0.227. This labelling was solely based on mapping query columns to knowledge graph properties and without using query column headers. There are clearly some challenges for this method of labelling, especially when there are unit mismatches. For example, the query column can be in "hectare" but the corresponding KG property may only be in "square metre."

For a comparison, we also ran UnitTagger [31] on the same set of 110 columns with the unit given in the header. UnitTagger could pull a correct unit for 79 columns (a precision of 0.718) and missing 31 columns even though the unit was in the header. For example, it was not able to recognize some compounded units like " $ million." Given that our method is orthogonal to UnitTagger, one using the header and another using the content, it is possible to combine the two approaches to achieve a better performance; this is outside the scope of this paper and is not further pursued. An idea of combining the two methods is listed in Section 5.3.

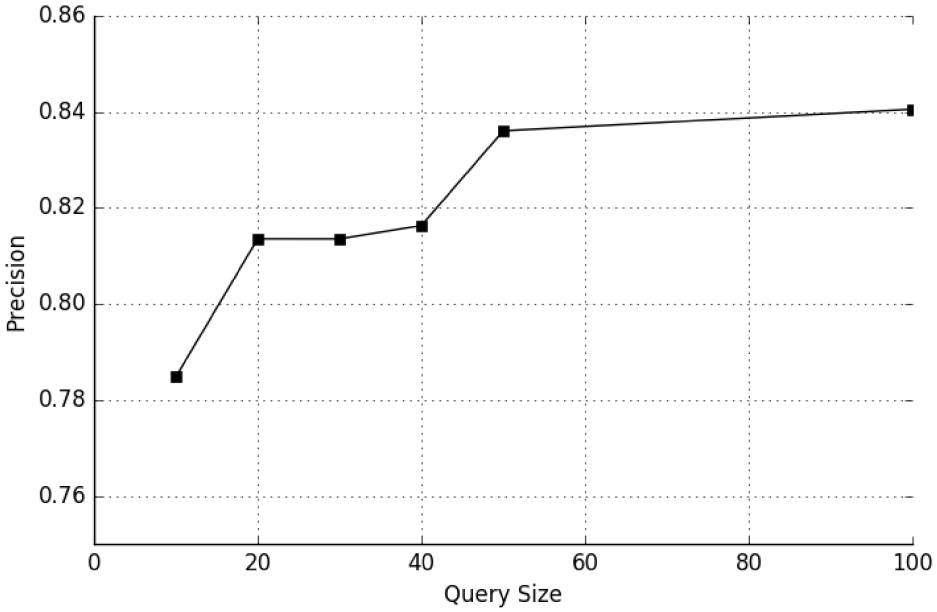

Figure 4.1: Comparing average precision@1 of different sample sizes.

4.3 Choice of Sampling Parameters

There are two parameters related to sampling from query and knowledge graph columns that can affect the performance. Here we evaluate different choices of these parameters and the impacts on the performance in terms of the running time and precision.

4.3.1 Query Sample Size

As discussed in Section 3.5, one way to reduce the size of a query column, and as a result the cost of a mapping, is to use a sample of the query column instead of the whole column. To understand the impacts of the sample size on the performance and the trade-offs between running time and the quality of the mapping, we partition our KG columns into 40% as query columns and 60% as KG columns. We then randomly select 200 queries and construct query samples of sizes ranging from 10 to 100 [4]. All experiments in this subsection are run on a machine with i5-7200U and 8GB of RAM.

[4]The selected queries all had at least 100 elements to make this sampling possible. No limit was set on the maximum size.

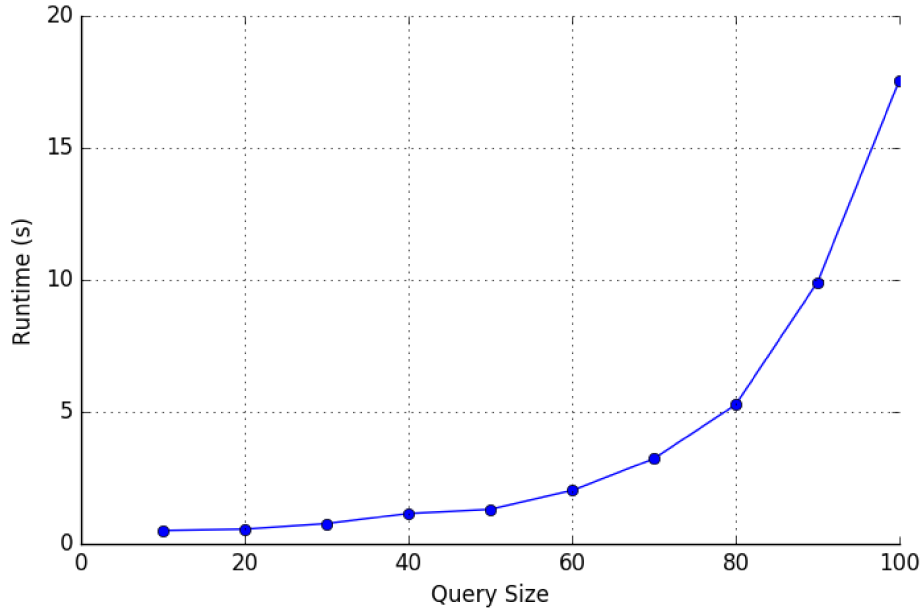

Figure 4.2: Comparing average runtime of different sample sizes

Figure 4.1 and 4.2 shows both the running time and precision for different sample sizes, with the results averaged over 200 queries. As expected, the running time increases quadratically with the query sample size and smaller samples are preferred. At the same time, the precision does not change much as the sample size increases. Given that the running time increases quadratically, whereas the precision does not change much for sample sizes larger than 50, a preferred range for the sample size is between 10 and 50. In addition, we observe that a leap of precision occurs where sample size increases from 10 to 20, and from 40 to 50. Selecting sample sizes larger than 50 has minor effects on the precision. A selection that balances precision and running time can be made at sample size of 50.

4.3.2 Number of KG Replacements

Sometimes the size of a query column is larger than the size of a matching property in a knowledge graph, hence an injective mapping cannot be constructed. This happens more often for large query columns. To avoid false negatives (i.e., not missing any qualifying KG properties), the injective map-

ping constraint may be relaxed, allowing multiple values of the query column to be mapped to the same value of a KG property. We refer to this as sampling KG with replacements, on the basis that the query is sampled from the domain of a KG column allowing replacements. Table 4.6 shows the effect of replacements on the precision for 200 randomly selected queries. In our experiment, we partition the knowledge graph columns into query and KG columns with ratios (60%, 40%) and (70%, 30%), to ensure query columns are larger than the matching KG columns. The minimum number of replacements is set to $\lceil \frac{|q|}{|c|} \rceil - 1$ for constructing a valid mapping.

query size	1	1.5	2	2.5	unlimited
60%	**0.760**	0.735	0.720	0.730	0.660
70%	**0.730**	0.710	0.710	0.705	0.590

Table 4.6: Precision varying the number of replacements

As expected, the best mapping, in terms of precision, is when there is the least number of replacements. As the number of replacements increases, the precision drops but the drop is not significant for small numbers of replacements. We also show the result when there is no limit on the number of replacements, which is equivalent to ranking the candidate columns based on LB_2. Clearly we want to keep the number of replacements close to the minimum $\lceil \frac{|q|}{|c|} \rceil - 1$ to reduce the impact on precision as well as the running time, which increases with the number of candidate KG columns.

4.4 Effectiveness of Pruning

Tables 4.7 and 4.8 show the effectiveness of our pruning under the two lower bounds, LB_1 and LB_2, introduced in Section 3.5.2. For a set of 100 query columns randomly selected from our Wikitables dataset, we report the number of candidate columns pruned by each of LB_1 and LB_2. Our experiment is conducted under two settings: (1) mapping query columns to the entire candidate columns set (shown in Table 4.7), and (2) mapping query columns to only candidate columns of a matching entity type (shown in Table 4.8). Note

that LB_1 is a looser lower bound but more efficient, and it is applied prior to LB_2 in our pruning steps in Algorithm 1. k is set to 3 in this experiment.

Lower bound	Columns	Pruned	Left
LB_1	17617	13096.8 (74.3%)	4520.2 (25.7%)
LB_2	4520.2	4222.9 (93.4%)	297.3 (6.6%)
Overall	17617	17319.7 (98.3%)	297.3 (1.7%)

Table 4.7: Columns pruned by our lower bounds (without bounding the entity types), averaged over 100 query columns

Lower bound	Columns	Pruned	Left
LB_1	40.73	25.84 (63.4%)	14.89 (36.6%)
LB_2	14.89	11.12 (74.7%)	3.76 (25.3%)
Overall	40.73	36.97 (90.7%)	3.76 (9.3%)

Table 4.8: Columns pruned by lower our bounds (with entity types bounded), averaged over 100 query columns

We find that without bounding the entity type to reduce the search space, most of the columns can be pruned with an average of 2% of the KG columns left for the mapping construction. When the entity type is bounded, there is less number of candidate columns to be pruned. This results in pruning 91% of the columns in our experiment, and leaving less than, on average, 4 columns for a cost optimization.

Chapter 5

Conclusion

5.1 Summary

In this thesis we have studied the problem of semantic and unit annotation of numerical columns in web tables, and proposed a KG matching based method for column annotation. We have formulated the problem as an optimization where a least cost mapping to a KG column is sought.

We evaluated our method on manually labeled dataset from two real-world web table sources, and compared our method with some of the state-of-the-art methods with a similar setting to ours, as well as a recent supervised approach. The results show that our proposed method performs better in terms of precision and the F1 score under our evaluation settings comparing to the baselines.

5.2 Limitations

Our work faces some limitation where the main limitation is the incompleteness of knowledge graphs. Our work is under the assumption of having a complete knowledge graph, which is not possible in realistic settings. As a result, many columns are not annotatable since they are not recorded in the knowledge graph. For cases where knowledge graph provides sufficient information, our method performs reasonably well. Another limitation is running time. As shown in Figure 4.2, the running time increases quadratically as the size of query column increases in the stage of finding optimal mapping, which

may be infeasible for large columns. This may force us to abandon mapping construction for the entire column, and instead we take samples of smaller sizes as a workaround. Having smaller query size may reduce precision as shown in Figure 4.1.

5.3 Future Work Directions

Our work leads to a few interesting directions. First, as our work matches entity types and not the actual entities, a question is if the presence of query entities in the knowledge graph can help with the annotations. Second, two columns with the same semantic labels may not match if they are given in different units (e.g., pounds vs kilograms). Detecting possible transformations between different units and incorporating this in retrievals is another direction. Finally, numeric values in a column may vastly differ from knowledge graph entries if a scalar multiplier is applied such as "million". As mentioned in Section 4.2, one idea to combine our method with existing methods is to first detecting scalar in the query column using UnitTagger, and then apply the scalar multiplier to the query column. This will reduce errors in mapping construction and may improve the performance.

References

[1] A. Alobaid and O. Corcho, "Fuzzy semantic labeling of semi-structured numerical datasets," in *Knowledge Engineering and Knowledge Management*, C. Faron Zucker, C. Ghidini, A. Napoli, and Y. Toussaint, Eds., Cham: Springer International Publishing, 2018, pp. 19–33, ISBN: 978-3-030-03667-6.

[2] A. Alobaid, E. Kacprzak, and Ó. Corcho, "Typology-based semantic labeling of numeric tabular data," *Semantic Web*, vol. 12, no. 1, pp. 5–20, 2021.

[3] C. Bhagavatula, T. Noraset, and D. Downey, "Tabel: Entity linking in web tables," in *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, Eds., ser. Lecture Notes in Computer Science, vol. 9366, Springer, 2015, pp. 425–441.

[4] C. S. Bhagavatula, T. Noraset, and D. Downey, "Methods for exploring and mining tables on wikipedia," in *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics*, ser. IDEA '13, Chicago, Illinois: Association for Computing Machinery, 2013, pp. 18–26, ISBN: 9781450323291. DOI: 10.1145/2501511.2501516. [Online]. Available: https://doi.org/10.1145/2501511.2501516.

[5] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "Webtables: Exploring the power of tables on the web," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 538–549, 2008.

[6] J. Chen, E. Jiménez-Ruiz, I. Horrocks, and C. Sutton, "Colnet: Embedding the semantics of web tables for column type prediction," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 29–36, Jul. 2019. DOI: 10.1609/aaai.v33i01.330129. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/3765.

[7] S. Cucerzan, "Large-scale named entity disambiguation based on wikipedia data," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 708–716.

[8] A. M. Dai, C. Olah, and Q. V. Le, "Document embedding with paragraph vectors," *CoRR*, vol. abs/1507.07998, 2015. arXiv: 1507.07998. [Online]. Available: http://arxiv.org/abs/1507.07998.

[9] X. Du, Y. Cai, S. Wang, and L. Zhang, "Overview of deep learning," in *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 2016, pp. 159–164. DOI: 10.1109/YAC.2016.7804882.

[10] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, and V. Christophides, "Matching web tables with knowledge base entities: From entity lookups to entity embeddings," in *The Semantic Web – ISWC 2017*, C. d'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange, and J. Heflin, Eds., Cham: Springer International Publishing, 2017, pp. 260–277, ISBN: 978-3-319-68288-4.

[11] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs," in *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*, M. Martin, M. Cuquet, and E. Folmer, Eds., ser. CEUR Workshop Proceedings, vol. 1695, CEUR-WS.org, 2016. [Online]. Available: http://ceur-ws.org/Vol-1695/paper4.pdf.

[12] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by successive approximation," *Mathematics of Operations Research*, vol. 15, no. 3, pp. 430–466, 1990.

[13] G. Hignette, P. Buche, J. Dibie-Barthélemy, and O. Haemmerlé, "Fuzzy annotation of web data tables driven by a domain ontology," in *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, and E. P. B. Simperl, Eds., ser. Lecture Notes in Computer Science, vol. 5554, Springer, 2009, pp. 638–653. DOI: 10.1007/978-3-642-02121-3_47. [Online]. Available: https://doi.org/10.1007/978-3-642-02121-3%5C_47.

[14] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum, "Robust disambiguation of named entities in text," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 2011, pp. 782–792.

[15] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, Ç. Demiralp, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining, ser. KDD '19, Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1500–1508, ISBN: 9781450362016. DOI: 10.1145/3292500.3330993. [Online]. Available: https://doi.org/10.1145/3292500.3330993.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, F. R. Bach and D. M. Blei, Eds., ser. JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, 2015, pp. 448–456.

[17] E. Kacprzak, J. M. Giménez-García, A. Piscopo, L. Koesten, L.-D. Ibáñez, J. Tennison, and E. Simperl, "Making sense of numerical data - semantic labelling of web tables," in *Knowledge Engineering and Knowledge Management*, C. Faron Zucker, C. Ghidini, A. Napoli, and Y. Toussaint, Eds., Cham: Springer International Publishing, 2018, pp. 163–178.

[18] F. Q. Lauzon, "An introduction to deep learning," in *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, 2012, pp. 1438–1439. DOI: 10.1109/ISSPA.2012.6310529.

[19] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, ser. JMLR Workshop and Conference Proceedings, vol. 32, JMLR.org, 2014, pp. 1188–1196. [Online]. Available: http://proceedings.mlr.press/v32/le14.html.

[20] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nat.*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[21] G. Limaye, S. Sarawagi, and S. Chakrabarti, "Annotating and searching web tables using entities, types and relationships," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 1338–1347, Sep. 2010, ISSN: 2150-8097. DOI: 10.14778/1920841.1921005. [Online]. Available: https://doi.org/10.14778/1920841.1921005.

[22] S. Neumaier, J. Umbrich, J. X. Parreira, and A. Polleres, "Multi-level semantic labelling of numerical values," in *The Semantic Web – ISWC 2016*, P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil, Eds., Cham: Springer International Publishing, 2016, pp. 428–445.

[23] P. Nguyen, K. Nguyen, R. Ichise, and H. Takeda, "Embnum+: Effective, efficient, and robust semantic labeling for numerical values," *New Gener. Comput.*, vol. 37, no. 4, pp. 393–427, 2019. DOI: 10.1007/s00354-019-00076-w.

[24] K. Nishida, K. Sadamitsu, R. Higashinaka, and Y. Matsuo, "Understanding the semantic structures of tables with a hybrid deep neural network architecture," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, Feb. 2017.

[25] N. F. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: Lessons and challenges," *Commun. ACM*, vol. 62, no. 8, pp. 36–43, 2019. DOI: 10.1145/3331166. [Online]. Available: https://doi.org/10.1145/3331166.

[26] J. B. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," *Math. Program.*, vol. 77, pp. 109–129, 1997. DOI: 10.1007/BF02614365. [Online]. Available: https://doi.org/10.1007/BF02614365.

[27] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic Web*, vol. 8, no. 3, pp. 489–508, 2017. DOI: 10.3233/SW-160218. [Online]. Available: https://doi.org/10.3233/SW-160218.

[28] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, A. Moschitti, B. Pang, and W. Daelemans, Eds., ACL, 2014, pp. 1532–1543. DOI: 10.3115/v1/d14-1162.

[29] M. Pham, S. Alse, C. A. Knoblock, and P. Szekely, "Semantic labeling: A domain-independent approach," in *The Semantic Web – ISWC 2016*, P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil, Eds., Cham: Springer International Publishing, 2016, pp. 446–462.

[30] S. Ramnandan, A. Mittal, C. A. Knoblock, and P. Szekely, "Assigning semantic labels to data sources," in *The Semantic Web. Latest Advances and New Domains*, F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, Eds., Cham: Springer International Publishing, 2015, pp. 403–417, ISBN: 978-3-319-18818-8.

[31] S. Sarawagi and S. Chakrabarti, "Open-domain quantity queries on web tables: Annotation, response, and consensus models," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14, New York, New York, USA: Association for Computing Machinery, 2014, pp. 711–720, ISBN: 9781450329569. DOI: 10.1145/2623330.2623749. [Online]. Available: https://doi.org/10.1145/2623330.2623749.

[32] W. Shen, J. Wang, and J. Han, "Entity linking with a knowledge base: Issues, techniques, and solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 2, pp. 443–460, 2014.

[33] T. Steiner, R. Verborgh, R. Troncy, J. Gabarró, and R. V. de Walle, "Adding realtime coverage to the google knowledge graph," in *Proceedings of the ISWC 2012 Posters & Demonstrations Track, Boston, USA, November 11-15, 2012*, B. Glimm and D. Huynh, Eds., ser. CEUR Workshop Proceedings, vol. 914, CEUR-WS.org, 2012. [Online]. Available: http://ceur-ws.org/Vol-914/paper%5C_2.pdf.

[34] K. Takeoka, M. Oyamada, S. Nakadai, and T. Okadome, "Meimei: An efficient probabilistic approach for semantically annotating tables," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 281–288, Jul. 2019. DOI: 10.1609/aaai.v33i01.3301281. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/3796.

[35] A. Uyar and F. M. Aliyu, "Evaluating search features of google knowledge graph and bing satori: Entity types, list searches and query interfaces," *Online Inf. Rev.*, vol. 39, no. 2, pp. 197–213, 2015. DOI: 10.1108/OIR-10-2014-0257. [Online]. Available: https://doi.org/10.1108/OIR-10-2014-0257.

[36] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Ç. Demiralp, and W.-C. Tan, "Sato: Contextual semantic type detection in tables," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 1835–1848, 2020.

[37] Z. Zhang, "Effective and efficient semantic table interpretation using tableminer^{+}," *Semantic Web*, vol. 8, no. 6, pp. 921–957, 2017. DOI: 10.3233/SW-160242. [Online]. Available: https://doi.org/10.3233/SW-160242.