# Text Driven Knowledge Graphs

Thesis submitted in partial fulfillment
of the requirements for the degree of

*Master of Science in* ***Computer Science and Engineering*** *by Research*

by

AMAN MEHTA
201502106
AMAN.MEHTA@RESEARCH.IIIT.AC.IN

International Institute of Information Technology
Hyderabad - 500 032, INDIA
March 2023

International Institute of Information Technology
Hyderabad, India

# CERTIFICATE

It is certified that the work contained in this thesis, titled "Text Driven Knowledge Graphs" by Aman Mehta, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. Kamalakar Karlapalem

To all of my *three* parents - father, mother and elder brother

# Acknowledgments

As I sit down to right this section, I feel nostalgic about all the time that I spent, in hostel corridors, canteens, classrooms and labs to discuss variety of research-things. I still cannot believe it's time.

A rush of memories goes past by, starting from first year when all the "dullas" are made fun of and joked about how hard it's going to be to get past the research. They weren't wrong, but it wasn't impossible. Second year comes by and I step on this with the aim of making a significant difference to whichever subject of research that I do, I get most excited by Dr. Kamal sir's work and as I fill the form, I already fill anxious about which advisor I am going to get, because that was probably going to shape how many next three+ years are going to be. I see that I get matched with Kamal sir, and felt lucky, but I had no idea how immensely lucky I got.

It has been an absolute pleasure working with you, Kamal sir. I'm forever grateful for you being extremely available, even reviewing my work in late hours, early mornings, you name it. I appreciate all of your ideas and guidance, which has helped me through my way here and it would *not* have been possible without your support. Again, forever grateful to have an opportunity to work with you. Cheers!

Well, I have come across many shortcomings and hardships, but if you're reading this, trust me when I say that they were really tough to past them. I'm grateful for all the time that my parents and my elder brother have checked on me and made sure that I get all the support that I need to excel. I feel extremely grateful for their support to get me out of my "shortcomings", as I look back and see what an insurmountable task they've accomplished for me. I owe all of my life accomplishments to my family. I'm super lucky to have them in my life and I hope to find ways to give back as I grow older.

To all my friends, who genuinely had my back and wished well for me - I'm thankful for your presence in my college life. Thank you for always motivating me when I felt low. I cherish all the wonderful experiences that you guys have made me a part of. Thank you for teaching me valuable lessons in life, which has helped me through life. I'm thankful for all the friendships that I've made and all the laughter that we've shared. Thanks for making me a better person.

Thanks to FYA, the smartest and loveli-est person I've ever met for helping me out and showing me a path whenever it felt impossible. Thanks for all your contributions to this thesis, I wouldn't have come to writing this page without you, that's for sure.

Thanks to all the people, for even the smallest of help or support that you might have provided, I'm sure it things wouldn't be same without those - butterfly effect :)

Lastly, thank you IIIT-H for providing me the opportunity to build a life I can enjoy and be proud of!

Grateful. Forever Grateful.

# Abstract

Although the concept of a knowledge graph has been discussed since at least 1972 [93] it wasn't until Google [96] unveiled their version in 2012 that it truly took off. Since then, several companies have started working on their own knowledge graphs, including Google, Amazon [54], eBay [82], Twitter, IBM [21], LinkedIn [37], Microsoft [95], and Uber. There has been a rise in the number of academic publications devoted to the topic of knowledge graphs in recent years, reflecting the growing interest in this idea. There are several books [79, 85, 52] and papers [84, 24] that detail knowledge graphs, as well as unique techniques to generating and analysing knowledge graphs and assessments of various knowledge graph aspects [103].

Apart from these *enterprise* Knowledge Graphs which are private and not accessible to public, there are a number of a number of public knowledge graphs published where the content is accessible for users of the web. The publicly available Knowledge Graphs are called *open* KGs. DBpedia [58], YAGO [40], Freebase [101], Wikidata [8] are the top few examples of such open KGs which are multi-domain and are built using the data from Wikipedia.

Automatic extraction of information from text and its transformation into a structured format is an important goal in both Semantic Web Research and computational linguistics. Knowledge Graphs (KG) serve as an intuitive way to provide structure to unstructured text. A fact in a KG is expressed in the form of a triple which captures entities and their interrelationships (predicates). Multiple triples extracted from text can be semantically identical but they may have a vocabulary gap which could lead to an explosion in the number of redundant triples. Hence, to get rid of this vocabulary gap, there is a need to map triples to a homogeneous namespace. In this work, we present an end-to-end KG construction system, which identifies and extracts entities and relationships from text and maps them to the homogenous DBpedia namespace. For Predicate Mapping, we propose a Deep Learning architecture to model semantic similarity. This mapping step is computation heavy, owing to the large number of triples in DBpedia. We identify and prune unnecessary comparisons to make this step scalable. Our experiments show that the proposed approach is able to construct a richer KG at a significantly lower computation cost with respect to previous work.

Over recent years, document similarity has grown to become the foundation of various natural language processing activities, which are crucial to information retrieval, automatic question answering, machine translation, dialogue systems, and document matching. For document or topic similarity, focusing on the semantics within the text has been the most common and pursued direction of effort.

Our novel KG-based similarity classifier works on the limitations of previous approaches and also provides reasoning behind the classification. Our results show that we're able to score similarity between Wikipedia documents accurately. Furthermore, the accuracy of our approach is able to withstand against noise and paraphrasing. We also see that our classifier can be used for category outlier detection in DBpedia.

In this thesis, we will focus on Knowledge Graph construction from unstructured text and document similarity. Our knowledge graph construction approach performs better $> 25\%$ better than Cosine & Rule-based approaches, and is also computationally cheaper than state-of-the-art T2KG [53] by a magnitude of atleast $10^6$. We use these constructed knowledge graphs to classify similarity between two documents, which is used to detect category outliers in DBpedia and find highly similar documents.

# Contents

# List of Figures

# List of Tables

*Chapter 1*

# Introduction

## 1.1 Motivation

The correspondence of data is all text driven. Specialists produce papers and articles that are made accessible in on the web and disconnected from distribution media as text reports. The whole library, innovation, administration, and exploration scene is based on unstructured text articles, blogs, research papers, and so on. With all this large amount of unstructured information stored in form of text documents, there is a rising need to organize and structure them inorder to being able to use information effectively. One of the most effective and popular ways to extract knowledge from text is *Knowlege Graphs*.

A KG built over any unstructured text document helps in making the information in the document queryable. This makes the previously inoperable information usable for tasks such as search result ranking, recommendation, question answering, etc [77, 32, 97, 78, 3]. Another interesting application of constructing KG over text is to add new or missing information to an existing KG like DBpedia [3], Freebase [7].

## 1.2 Knowledge Graph

In any industry, text documents that contain important data are a common occurrence. Hence, improving the ability of machines to understand the intent and context of information to the level of humans is one of biggest challenges faced today. Achieving this would involve transformation of unstructured text to a structured format. Knowledge graphs provide an intuitive way of giving shape and structure to the initially unstructured information. A fact in a KG is represented by a triple of the form $<S;\ P;\ O>$. $S$, $O$ denote the *subject* and *object*, respectively and $P$ is the *predicate* which describes the relationship between $S$ and $O$.

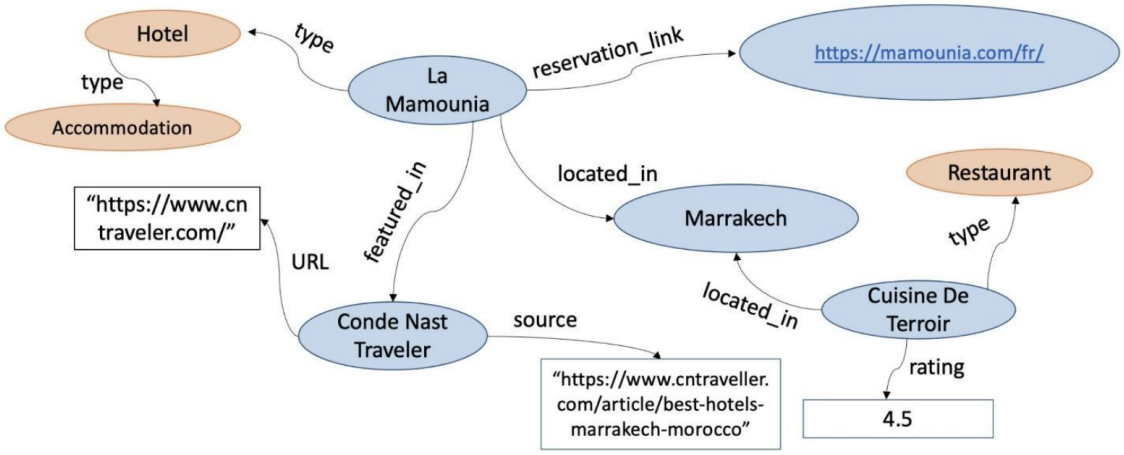


**Figure 1.1** Graphical extraction of a webpage's information

## 1.3 Brief study of a KG: Chilean tourism

Chilean Tourism KG [42] is a made-up knowledge network based on Chilean tourism to illustrate the notion of a KG. Government agencies are responsible for maintaining an up-to-date knowledge graph so that the KG can be used to advertise both new and established tourism attractions. To name only a few examples of what the knowledge graph will ultimately cover: cities, intercity lines, tourist destinations, cultural events, services, and businesses. Among the many potential applications for this data, one is the development of a multilingual website where tourists can search for attractions, events, and related services; the gathering of more information about the visitors' demographics popular season, nationality, etc.; and the investigation of tourists' attitudes toward various topics such as reviews, event and service summaries, reports of crime, etc. The KG could also help with common itineraries that detail the order and sequence of attractions to see on a vacation. We can also have bus, train and airlines network graphs which can be used for not only planning trips but also to offer more strategically sound alternate routes.

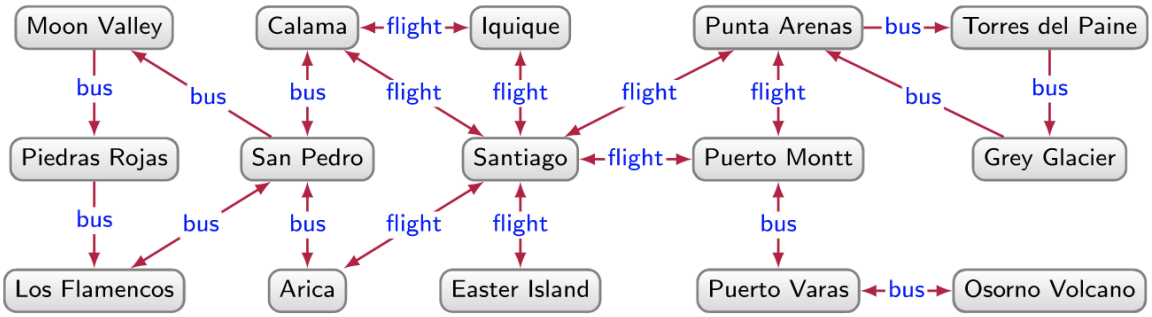


**Figure 1.2** Tourism in Chile illustrated as a knowledge graph of transport connections [42]

## 1.4 Problem 1: Text to KG

Owing to a large amount of information in form of text, it becomes important to construct Knowledge Graph from unstructured text. In chapter 3, we introduce a novel end-to-end text-to-KG construction system, which is mapped to DBpedia namespace in-order to reduce the redundancy due to vocabulary gap in unstructured text. We introduce an effective pruning strategy which reduces the Predicate Mapping step to be highly efficient (refer section 3.2.3 for details). We perform automatic evaluation using Wikipedia articles which shows that our system is able to generate knowledge graph with an F1-score of 0.678 (as shown in table 3.1). Due to a lack of standard evaluation metric for the problem statement, we also perform manual evaluation which showcases that we're able to generate a Knowledge Graph with a precision of 0.77, which is better than the state of the art. We also analyse at which step and why we're getting errors. Apart from this, we evaluate the significance of the sentence simplification module, as seen in section 3.3.3 we not only get higher number of KG triplets, but also improve the quantity of accurate triplets and reduce the number of misleading and incorrect triplets. We also see that our system is able to get rid of redundancy due to vocabulary gap, details are in section 3.3.4.

## 1.5 Problem 2: Document Similarity

Determining similarity between two texts has been a challenging problem in Natural Language Processing world. In chapter 4, we use our KG generation system and introduce a novel KG-based document similarity classifier. Our multi-level system focuses on unique set of structural qualities to determine whether the similarity between any pair of documents, either $Low$, $Medium$ or $High$. Not only that, our approach provides meaningful reasoning to explain the similarities and differences between the documents pair. We perform evaluations on Wikipedia articles and analyse the strengths and limitations of both the levels of our approach. Our approach can be used to determine category outliers, we perform

an experiment to see that we're able to detect outliers in DBpedia categories with high accuracy. Our approach is also tested to be robust to noise, as we evaluate documents after paraphrasing and noise, more details in section. More details on experiments and results in section 4.6.

## 1.6 Contributions of the thesis

We focus on two main problem statements. First, text to knowledge graph construction. Second, finding similarity between two text documents. In this thesis we propose an end-to-end pipeline of knowledge graph construction from text and a knowledge-graph based document similarity classifier with explanation. Our main contributions are as follows:

- We introduce a system to construct KG from text which is consistent with the DBpedia namespace.
- We introduce a novel Deep Learning based model to calculate similarity between two predicates for the Predicate Mapping step.
- Our approach is able to reduce complexity of Predicate Mapping by a factor of $10^7$.
- Our approach outperforms T2KG [53] with improved Precision and F1-score
- We evaluate the sentence simplification's effectiveness in improving the quality of triplet extraction.
- We evaluate the Redundancy Reduction of our approach for Science and History articles
- We survey the different methodologies of text document similarity
- We introduce SimX: a novel KG driven document similarity classification. SimX also provides explanations and we evaluate our approach on Wikipedia articles, for DBpedia category outliers and against paraphrasing + noise.

The organization of thesis is as follows:

- Chapter 2 presents the related work that has been done on Knowledge Graph construction from scratch and different approaches for Document similarity
- Chapter 3 presents our novel end-to-end pipeline of Knowledge Graph construction from unstructured text
- Chapter 4 presents our Knowledge Graph based Document similarity approach with explanations, SimX.
- Chapter 5 will conclude our contributions and discuss future work

Chapter 3 and 4 are self-contained, with experiments, evaluations and results.

*Chapter 2*

# Related Work

A knowledge graph is a comprehensive receptacle to exemplify perception, founded on the calculation of beings and connections. Knowledge graphs contribute an instinctive means of lending pattern and hierarchy to originally undeveloped data.

## 2.1 Text to Knowledge Graph

Traditional methods ([25]) focus on building KGs from infobox templates and categorization information in the Wikipedia articles. However, the unstructured text of these articles is left unprocessed. In order to make this natural language text structured and usable, generating a KG over text becomes an important task. Previous works such as [11, 27, 92, 12, 4, 55] use information extraction system to extract facts from NL text. But these facts do not necessarily follow the paradigm of a KG (such as DBpedia), making the KG construction task challenging. For instance, consider the output of sample sentence *(a)* from (table 2.1) using OLLIE information extractor - (Barack Obama, was born in, Honolulu). This fact is identical to (Barack Obama, birthPlace, Honolulu) in DBpedia. Although these two facts are identical, there is a vocabulary gap between them. In the fact extracted from text, *"was born in"* is a natural language phrase, whereas *"birthPlace"* is a formatted predicate in DBpedia. These two relational phrases have different surface forms and hence they cannot be mapped just on the basis of string similarity.

Furthermore, same DBpedia entities and predicate could be used in multiple NL excerpts. For instance, *"He", "Barack Obama"* and *"Barack"* in the facts extracted for *(a), (b)* and *(c)* (from 2.1) correspond to *"Barack_Obama"* in DBpedia. Similarly, the predicates such as *"was born in"*, *"grew up in"*, *"belongs to"* are related to the same predicate *"birthPlace"* in DBpedia. As shown in fig 2.6, this

| | |
|---|---|
| *(a)* | *Barack Obama was born in Honolulu.* |
| *(b)* | *Obama was elected in 2009 as the president of the United states. He belongs to Honolulu.* |
| *(c)* | *Barack served as the 44th president of the United States and grew up in Honolulu.* |

**Table 2.1** Examples of few sentence which hold information

leads to a high amount of redundancy in the constructed KG, giving rise to an unnecessarily large sized KG.

This redundancy leads to wastage of space for its storage and wastage of time in execution of graph retrieval algorithms [62, 64, 104]. Therefore, it is important to efficiently resolve entities and their relationships in triples to facilitate a queryable Knowledge Graph. The most common approach to resolve them is by mapping them to a single homogeneous namespace, such as DBpedia namespace.

A number of studies have proposed ideas for Entity Mapping [70]. However, only a few studies have worked on the task of Predicate Mapping. A few rule-based [26] and similarity-based approaches [53] have been proposed in recent years, but both of them have their own limitations. Simple rule-based approaches cannot generate rules efficiently, especially when text sources are sparse. This is due to the fact that these rules are manually generated and hence it cannot get rid of all redundancies. On the other hand, similarity-based solutions for Predicate mapping are challenging in two aspects:
a) To map a predicate to another namespace, they need to capture the accurate semantics for calculating similarity scores
b) Comparing a predicate to each of the predicate in DBpedia is highly time consuming owing to the number of *candidates* (i.e. the number predicates in DBPedia namespace to which a particular text predicate can map to)

## 2.2 Document Similarity

### 2.2.1 Overview

In the coming sections, we will be building context of current work on text similarity from scratch and in increasing order of sophistication. The aim of this section is to develop fundamentals around the topic. According to information theory [63], similarity is defined as the shared characteristics of two text samples. The degree of resemblance increases with increasing commonality and vice versa. In many NLP (Natural Language Processing)-based activities, including information retrieval [61], topic detection, topic tracking, automatic question answering [49], machine translation [102], dialogue systems [94], and document matching [81], text similarity is quickly emerging as a significant tool.

Traditional document similarity measurements distinguish between similar and dissimilar documents in a coarse manner. They usually overlook the perspective in which two documents are identical. This restricts the level of detail that programees like recommendation systems, which depend on document similarity, can reach.

Over the past thirty years, many semantic similarity methodologies have been measured in various ways. On the basis of statistics, corpora, and knowledge sources like Wikipedia, most academics categorise text similarity measurement techniques [34]. This categorization disregards the text distance computation method and just takes into account the text representation. Text similarity not only takes

into consideration the semantic similarity between texts, but also takes a more comprehensive approach by examining the common semantic characteristics of two terms. In contrast to the words *"King"* and *"Queen"*, which are semantically comparable, *"King"* and *"Man"*, for instance, are closely related but not semantically identical. Therefore, one of the aspects of semantic relatedness that can be considered is semantic similarity. The similarity of the semantic connection.

The text similarity measures differ fundamentally, in the following sections we'll take a close look on the different algorithms on ranking text similarity. First we'll take a close look at the different methods of calculating

### 2.2.2 Text Distances

In this section, we will examine numerous distances defined between two pieces of text, to determine the semantic closeness between them.

#### 2.2.2.1 Block Distance

Block Distance calculates the distance needed to travel along a grid-like path to go from one data point to the next. The sum of the discrepancies between two strings' respective components is known as the Block distance [69]. It is also referred to as Manhattan Distance, Boxcar Distance, Absolute Value Distance, L1 Distance, City Block Distance, and Manhattan Distance. Consider two strings - *A* and *B*, represented as a combination of two components *x* and *y*, then the block distance between *A* and *B* for example would be,

#### 2.2.2.2 Euclidean Distance

Euclidean distance or *L2* distance is the square root of the sum of squared differences between corresponding elements of the two vectors.

Formula is stated in equation 2.1

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{2.1}$$

$p, q$ = two points in Euclidean n-space
$q_i, p_i$ = Euclidean vectors, starting from the origin of space (initial point)
$n$ = n-space

#### 2.2.2.3 Manhattan Distance

The separation between two real-valued vectors is also determined using the Manhattan distance. The problem we are working on gives more importance to the distance between the points only along with

the grids, but not the geometric distance. Manhattan distance is calculated as the sum of the absolute differences between the two vectors, which typically only works if the points are arranged in the form of a grid. Following one-hot encoding, the Manhattan distance is used to determine how similar two documents are [22]. The Manhattan formula for two-dimensional space is as follows,

$$Sim(x, y) = |x_1 - x_2| + |y_1 - y_2| \tag{2.2}$$

#### 2.2.2.4 Cosine Similarity

A comparison of two vectors in an inner product space using a cosine-based measure of similarity is known as "cosine similarity." It helps determine whether or not two vectors are pointing in roughly the same direction. Cosine Similarity can be used to rank documents in relation to a given vector of search terms or to compare documents. Let's compare two vectors, *x* and *y*.

$$Sim(S_a, S_b) = cos\theta = \frac{\vec{S_a}.\vec{S_b}}{||S_a||.||S_b||} \tag{2.3}$$

An illustration of cosine similarity is shown below in Fig 2.4

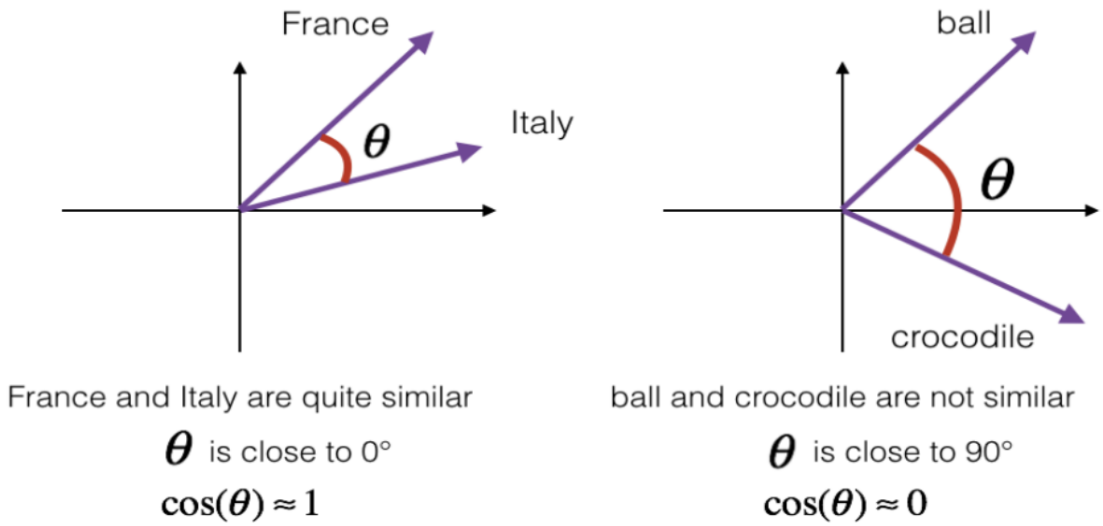


**Figure 2.1** Cosine similarity running example

#### 2.2.2.5 JS Divergence

The Jensen-Shannon divergence [66] is a tool for evaluating how similar two probability distributions are. It is often referred to as the total divergence from the average or the information radius (IRAD). In

order to compare the topic distributions of new documents with all topic distributions of documents in the corpus and to identify which documents are more similar in distribution by comparing their distribution differences, JS divergence is typically used in conjunction with LDA (latent dirichlet allocation). The distribution of the two documents is more comparable when the Jensen-Shannon distance is less [76].

$$JS(P_1||P_2) = \frac{1}{2}KL(P_1||\frac{(P_1+P_2)}{2}) + \frac{1}{2}KL(P_2||\frac{(P_1+P_2)}{2}) \tag{2.4}$$

#### 2.2.2.6 KL Divergence

Kullback-Leibler divergence is a measure of the differences between one probability distribution's degrees and a second base or reference probability distribution. KL divergence is also known as relative entropy or I-divergence.

$$KL(p||q) = \sum_{i=1}^{n} p(x)\log(\frac{p(x)}{q(x)}) \tag{2.5}$$

#### 2.2.2.7 Word Mover's Distance

Word mover's distance measures the smallest distance needed for a word in one text to reach a word in another text in the semantic space on the basis of expressing the text as a vector space [107] in order to reduce the cost of transferring text from text to text. It is based on earth's mover distance [1].

Word Mover's Distance" is based on a word vector and uses linear programming [56] at its foundation.

#### 2.2.2.8 Word Mover's Distance Extension

Mahalanobis distance [65] is an improvement of the Word Mover's distance. The similarity in word mover's distance is determined using the Euclidean distance. According to Euclidean distance, each dimension in a space has the same weight, meaning that they are all equally important. However, the relationship between the various dimensions is not considered. Instead of using the Euclidean distance, you can use the improved Mahalanobis distance to account for this information. To do this, first perform a linear transformation on the sample in the original space before calculating the Euclidean distance in the modified space.

The unsupervised word mover's distance is changed into the supervised word mover's distance by the addition of matrix loss, making it more effective for the purpose of text categorization [43].

### 2.2.3 String-based similarity measures

In this section, we'll look at the text representation, which displays text as directly calculable numerical properties. Measures of string similarity focus on character composition and string sequences. When two text strings are compared or approximate string matching is performed, a string metric is used to determine how similar or dissimilar (distance) the two text excerpts are. There are two ways that texts might be similar: **lexically** and **semantically**.

**Lexical Similarity** If the character sequences of the words that make up the text are similar, then the words are lexically similar. Formally, by combining word sets from the same or other languages, lexical similarity offers a way to compare two texts' similarities. A score of 0 indicates that there are no shared words between the two texts, while a score of 1 indicates that the vocabularies completely overlap. Lexical similarity can be measured in a variety of ways, including Jaccard Similarity, Cosine Similarity, Levenshtein Distance.

**Semantic Similarity** If two words have the same meaning, contrast one another, are used in the same way, are used in the same context, or are types of one another, then they are semantically similar. Formally, Semantic similarity gauges the resemblance between two texts based on their meaning as opposed to their lexical similarity. In order to summarise texts and extract important characteristics from lengthy papers or document collections, semantic similarity is a very valuable tool. Salient Semantic Analysis (SSA), Normalized Google Distance (NGD), and other techniques can be used to assess semantic similarity.

#### 2.2.3.1 Longest Common Substring

Consider two strings $S_a$ and $S_b$ from two different documents. LCS, when viewed as a string representation of the text, denotes the length of the longest substring that matches in strings $S_a$ and $S_b$. The length of the contiguous chain of characters that are present in both strings is what the algorithm uses to determine how similar two strings are. More information is shared among the lengthy strings ([98], [44], [45]).
The method of LCS [46] matching is frequently employed to assess how similar two strings (Sa, Sb) are to one another. LCS, when viewed as a string representation of the text, denotes the length of the longest substring that is identical to the strings Sa and Sb. Algorithm for finding LCS similarity between two strings $S_a$ and $S_b$ is described below:

$$LCS(S_a, S_b) = \begin{cases} 0,\ if\ S_a = 0\ or\ S_a = 0 \\ 1 + LCS(S_a - 1, S_b - 1), if\ x[S_a] == y[S_b] \\ max = \begin{cases} LCS(S_a, S_b - 1) \\ LCS(S_a - 1, S_b) \end{cases} if\ x[S_a] \neq y[S_b] \end{cases} \tag{2.6}$$

#### 2.2.3.2 Edit Distance - Damerau-Levenshtein

The least amount of changes needed to change the string from $S_a$ to $S_b$ is represented by the edit distance. The terms Levenshtein-distance [60] and Damerau-distance [18] are two different ways that editing distance is defined.

The atomic operations of D-distance and L-distance are different in that the former only includes delete, insert, and replace operations, while the latter also includes adjacent exchange activities. D-distance can only handle a single editing error because the definition of D-distance includes one additional adjacent action, whereas L-distance can handle numerous editing errors.

The lower the edit distance between $S_a$ and $S_b$, higher the similarity.

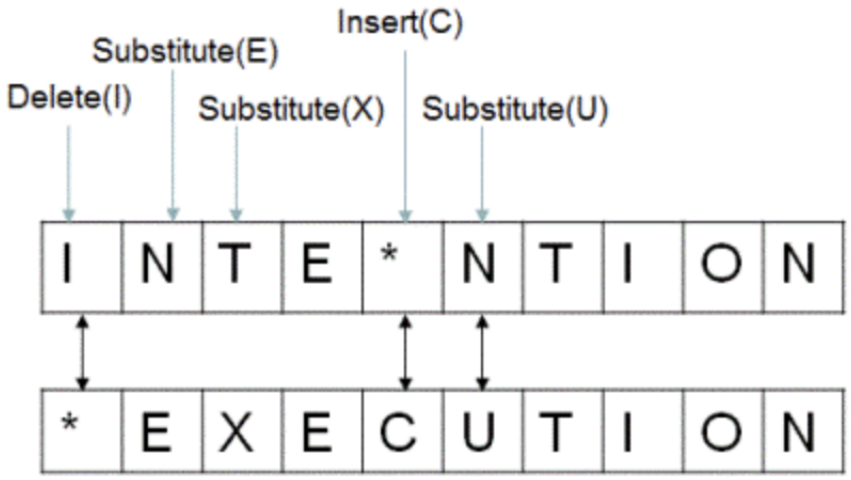


**Figure 2.2** Edit distance = 5, with one delete, one insert and 3 substitute operations

An example of a D-L edit distance is shown in fig 2.2

#### 2.2.3.3 Jaro Similarity

Jaro [48] is based on the quantity and arrangement of the shared characters between two strings; it accounts for common spelling mistakes and is mostly applied to record linking. The formula is as shown in Equation 2.7.

$$Sim = \begin{cases} 0, \ if \ m = 0 \\ \frac{1}{3}(\frac{m}{|S_a|} + \frac{m}{|S_b|} + \frac{m-t}{m}) \end{cases} \tag{2.7}$$

**Jaro-Winkler** [106] is an extension of Jaro distance that use a prefix scale to reward strings that match from the start for a predetermined prefix length.

#### 2.2.3.4 Needleman-Wunsch

Needleman-Wunsch method, which was the first to be used to compare biological sequences uses Dynamic Programming for it's computation. To determine the optimal alignment across both sequences, a global alignment is conducted. It works best when the two sequences are roughly the same length and share a lot of similarities overall [75].

#### 2.2.3.5 N-gram

A subsequence of *n* items from a particular text sequence is known as a "n-gram." The n-grams from each character or word in two strings are compared by n-gram similarity algorithms. By dividing the number of similar n-grams by the maximum number of n-grams, distance is calculated [5].

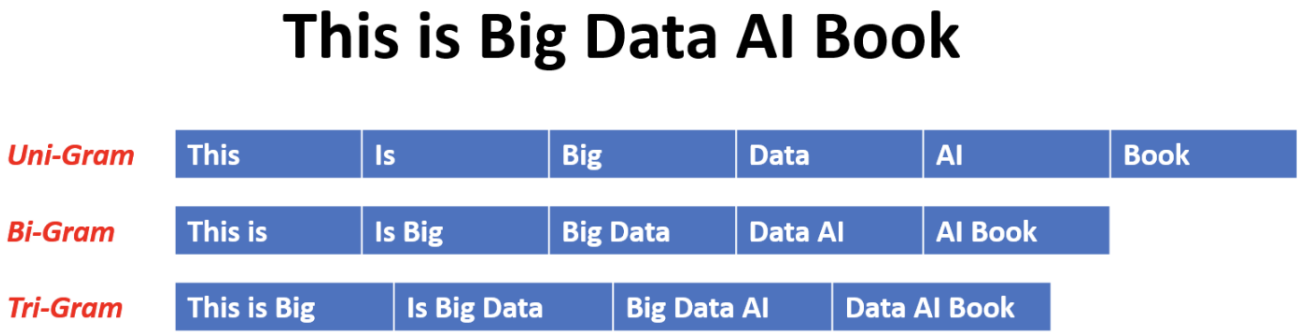


**Figure 2.3** N-gram example with uni-gram, bi-gram and tri-gram for a sentence

#### 2.2.3.6 Dice's Similarity Coefficient

The Sørensen–Dice coefficient [23] is a statistical approach used to gauge the similarity of two sample strings, $S_a$, $S_b$. It is defined by the following formulae, where *comm* ($S_a$, $S_b$) is the count of common characters in the two strings, known as collinear phrases.

$$Dice(S_a, S_b) = \frac{2 * comm(S_a, S_b)}{len(S_a) + len(S_b)} \tag{2.8}$$

#### 2.2.3.7 Jaccard Similarity

The size of the intersection divided by the size of the union of two sets is how Jaccard similarity is calculated [47]. When the text is somewhat long, the similarity will be smaller, so Jaccard must resolve the similarity through the set. Therefore, Jaccard is typically first normalised before computing similarity. Chinese words can be reduced to synonyms, whereas English terms can be reduced to the same root.

Formula is stated in equation 2.9

$$S(S_a, S_b) = \frac{S_a \cap S_b}{S_a \cup S_b} \tag{2.9}$$

The range of the Jaccard Similarity score is 0 to 1. Jaccard Similarity is 1 if the two documents are identical. In cases when there are no shared terms between two papers, the Jaccard similarity score is 0.

Consider two documents,
$doc_1$ = *"Data is the new oil of the digital economy"*
$doc_2$ = *"Data is a new oil"*

The Jaccard similarity formula would apply for the example as follows,

$$\begin{aligned} J(doc_1, doc_2) &= \frac{(data, is, the, new, oil, of, digital, economy) \cap (data, is, a, new, oil)}{(data, is, the, new, oil, of, digital, economy) \cup (data, is, a, new, oil)} \\ &= \frac{(data, is, new, oil)}{(data, a, of, is, economy, the, new, digital, oil)} \\ &= \frac{4}{9} = 0.444 \end{aligned} \tag{2.10}$$

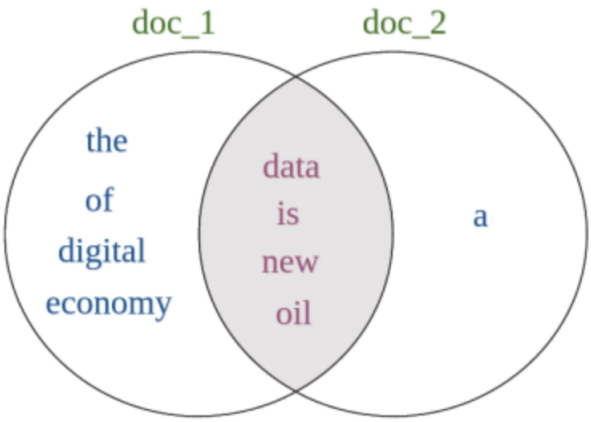


**Figure 2.4** Jaccard similarity between doc_1 and doc_2 using Pie chart

### 2.2.4 Corpus-based similarity

Corpus-Based Similarity is a semantic similarity metric that assesses word similarity using data gleaned from sizable corpora. A vast collection of written or spoken materials called a corpus is utilised in linguistic studies.

A significant distinction between corpus-based and string-based approaches is as follows: The corpus-based method calculates text similarity using data from the corpus, which might either be a textual characteristic or a co-occurrence probability. The string-based method, however, compares texts at the literal level.

#### 2.2.4.1 Pointwise Mutual Information (PMI-IR)

Turney [99] proposed the pointwise mutual information as a non-supervised method for assessing the semantic similarity of words utilising data gathered by information retrieval 776 (PMI-IR).

The PMI-IR similarity score of two words increases with how frequently they appear together nearby on a web page. Given two words $w_1$ and $w_2$, their PMI-IR is measured as

$$PMI - IR(w_1, w_2) = log_2 \frac{p(w_1 \& w_2)}{p(w_1) * p(w_2)} \tag{2.11}$$

The above equation serves as a gauge of the semantic similarity between $w_1$ and $w_2$, indicating the level of statistical reliance between them. We are utilising the NEAR query (co-occurrence within a ten-word frame), which strikes a balance between accuracy (results from synonymy tests) and efficiency (number of queries to be executed against a search engine), out of the four different types of queries provided by Turney [99]. The search engine AltaVista is specifically used to gather counts using the following query.

#### 2.2.4.2 Latent Semantic Analysis (LSA)

LSA [57] makes the assumption that words with similar meanings will appear in texts with similar structures. A large piece of text is converted into a matrix with word counts per paragraph (rows for unique words and columns for each paragraph), and a mathematical method known as singular value decomposition (SVD) is used to decrease the number of columns while maintaining the similarity structure among rows. The cosine of the angle generated by any two rows' two vectors is then used to compare words.

One method to get around the sparseness and large dimensionality of the traditional vector space model is by using LSA. In reality, the LSA similarity is calculated in a lower-dimensional space that takes use of second-order relationships between terms and texts. The conventional cosine similarity is then used to calculate the similarity in the resulting vector space. Also take note of the fact that LSA produces a vector space model that enables uniform representation and comparison of words, word sets, and texts.

By using singular value decomposition, LSA (latent semantic analysis) [19] maps the text from a sparse high-dimensional vocabulary space to a low-dimensional latent semantic space to determine how comparable the possible semantic space is. Values near 1 indicate documents that are quite similar, whereas values near 0 indicate documents that are very distinct [35]. Then, Hofmann proposes the subject layer based on LSA, training the topic with the improved PLSA (probabilistic latent semantic analysis) algorithm [41] utilising the expectation maximisation algorithm (EM).

#### 2.2.4.3 Generalized Latent Semantic Analysis (GLSA)

The Generalized Latent Semantic Analysis (GLSA) [67] framework is used to compute term and document vectors that are motivated by semantics. By concentrating on term vectors rather than the dual document-term representation, it expands the LSA technique. A dimensionality reduction technique and a measure of semantic association between concepts are needed for GLSA. Any form of similarity measure on the space of words and any useful dimensionality reduction technique can be combined with the GLSA methodology.

#### 2.2.4.4 Normalized Google Distance

The number of results provided by the Google search engine for a certain set of keywords is used to calculate the Normalized Google Distance (NGD) [16], a semantic similarity metric. In terms of Google distance, terms having the same or comparable meanings in a natural language sense tend to be "near," and terms with different meanings tend to be further apart.

The normalised Google distance between two search phrases is infinite if x and y never appear together on the same web page but do so separately. NGD is equal to zero if x and y always appear together.

#### 2.2.4.5 Bag-Of-Words

It is referred to as a "bag" of words because any details regarding the arrangement or structure of the words within the document are ignored. The model doesn't care where in the document recognised terms appear; it is only interested in whether they do. The key component of BOW is count vectorization, which measures similarity in applications like search, document categorization, and topic modelling by counting the amount of words that exist in a document to represent text. Refer [91] for more details.

#### 2.2.4.6 TF-IDF

TF-IDF [89], which stands for term frequency-inverse document frequency, is a metric that can be used to quantify the significance or relevance of string representations (words, phrases, lemmas) in a document among a group of documents (known as corpus). It is used in the fields of information retrieval (IR) and machine learning. The reason TF-IDF is effective is that a word is present in many papers and is used frequently in those documents. The words do not, despite the fact that they regularly appear in a document, have any particular significance.

$$tfidf(w, d, D) = tf(w, d) * idf(w, D) \tag{2.12}$$

where,

$$tf(w, d) = Freq(w, d)$$

$$idf(w, D) = log\frac{|D|}{N(w)}$$

- *Freq (w, d)* represents the frequency count of usage of word *w* in document *d*.
- $|D|$ indicates the total number of documents
- *N (w)* is the number of documents where the word *w* appears

#### 2.2.4.7 Latent Dirichlet Allocation

LDA (latent dirichlet allocation) presupposes that each document will have multiple themes, resulting in topic overlap in the document. Each document's words add to these topics/themes. Every document will be discretely distributed across all themes, and every word will be discretely distributed across all topics.

Latent Dirichlet Allocation (LDA), a generative statistical model used in natural language processing, explains a set of observations through unseen groups, with each group explaining why certain portions of the data are similar. LDA is used for topic modeling. This involves gathering words into documents and assigning each word's presence to one of the topics covered in the text. There will be a limited amount of topics in each paper.

In order to initialise the model, a topic is randomly assigned to each word in each document. Then, after iterating over each word, we cancel the assignment to its current topic, decrease the corpus scope of the topic count, and reassign the word to a new topic based on both the local and global (corpus scope) probabilities that the topic is assigned to the current document.

#### 2.2.4.8 Word2Vec

Word2vec [90] uses trained models of Bag-of-Words (BoW) and the model of word-skipping. The model fundamentals lies in predicting a middle word, based on the context and meaning of the surrounding words. A brief illustration of how it works is given in Figure 2.2

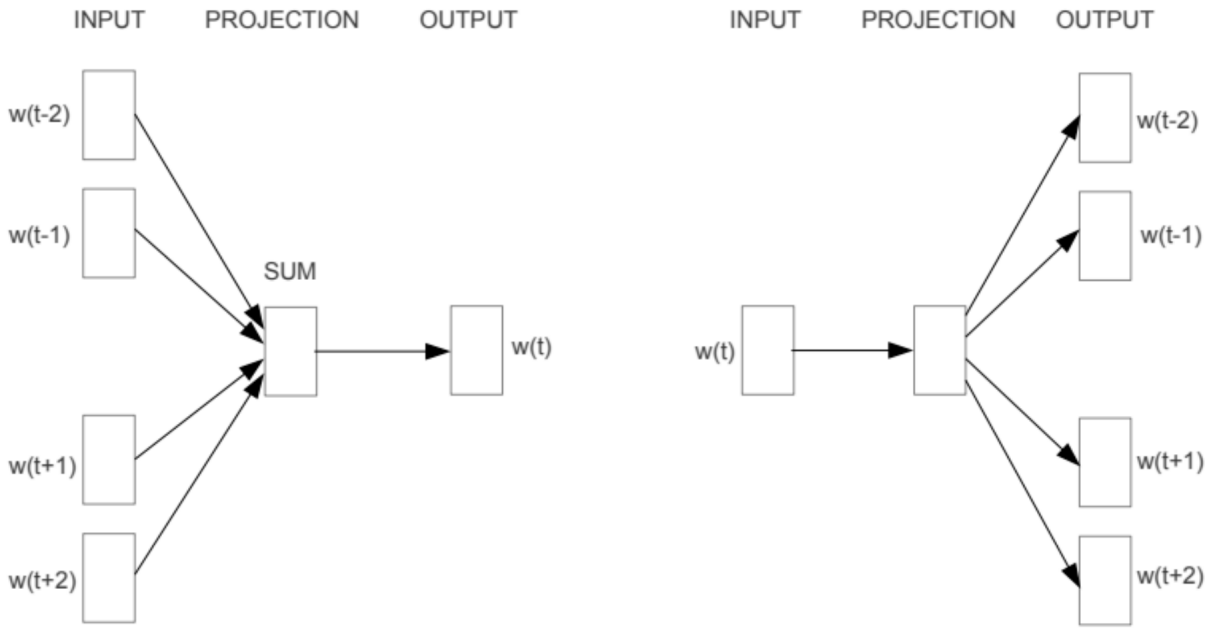


**Figure 2.5** Word2Vec illustration

## 2.3 Summary

T2KG [53] lacks in both of these aspects. Firstly, it uses a word2vec based model which is not able to capture all semantic features since it uses shallow neural networks[1]. Secondly, it scans all DBpedia triples in order to map a single predicate of KG to DBpedia namespace. So, when the text sources are large, it becomes a time consuming task to map all text predicates to DBpedia predicates. Even tougher when the size of the DBpedia itself is as large as 1.3 billion[2] triples, such as in the case of English version of DBpedia. We address both of the above challenges in the proposed approach.

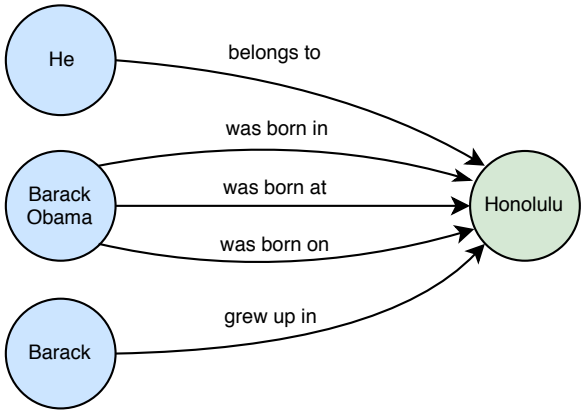


**Figure 2.6** Knowledge Graph obtained using information extraction (OLLIE [92])

The different document similarity approaches differ fundamentally from a Knowledge-Graph driven similarity, because

- We focus on structural based qualities in determining similarity, which are not considered by most of the above solutions,
- We provide explanation along with similarity classification whereas none of the above solutions provide explanation,
- We provide a classification of $Low, Medium, High$, not a numerical score.

Hence, due to these fundamental qualitative and quantitative differences, we should not compare our approach with current state of document similarity work.

[1] https://en.wikipedia.org/wiki/Word2vec
[2] https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10

*Chapter 3*

# Text to Knowledge Graph

In this chapter, we introduce an end-to-end KG construction system from unstructured text. In order to make this KG more useful, we map our KG to DBpedia namespace far more efficiently than the previous approaches. Under the Predicate Mapping step, we implement a pruning strategy to reduce a large number of unnecessary comparisons in order to make this step scalable. The end-to-end system is able to build a larger KG, with less redundancy while significantly reducing the computation cost in finding the Predicate Mapping step. Our main contributions are as follows:

- We developed a novel end-to-end pipeline to construct KG from text which is consistent with the DBpedia namespace.
- We proposed a Deep Learning model to calculate similarity between two predicates for the Predicate Mapping step.
- We developed optimization strategy to reduce the *Candidate Set* (described in section **??**) for each text predicate.
- We introduced a sentence simplification component to improve triple extraction.
- We performed multiple experiments to evaluate our KG construction system with other systems. The experiments show the effectiveness of our Predicate Mapping component and the benefits of the Sentence Simplification component.
- We conducted a separate study to quantify the extent of redundancy reduction in different domains of Wikipedia articles in the KGs constructed using our system.

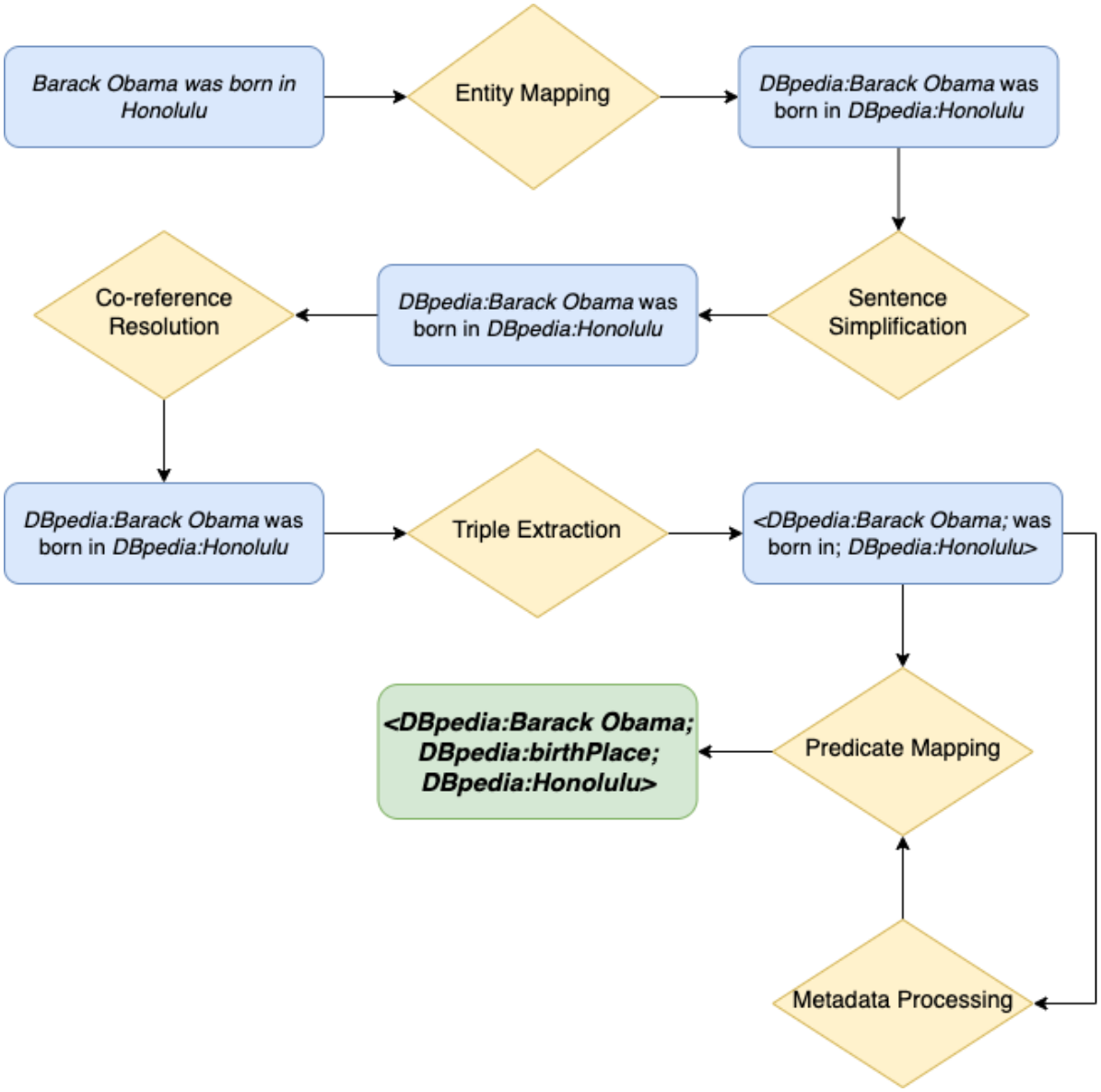


**Figure 3.1** Flow chart of our text to KG approach

In the following section, we describe in detail, the step-by-step procedure (as shown in Figure 3.1) to construct a Knowledge Graph from unstructured text. Our end-to-end pipeline has six components: The Entity Mapping (section 3.1.1) component maps the entities in the text triples to the DBpedia entities. The Sentence Simplifier (section 3.1.2) which simplifies sentences before triple extraction step to overcome limitations of triple extraction on complex sentences. The Co-reference Resolution component (section 3.1.3) which finds and replaces all the expressions that refer to the same entity in the text. The Triple Extractor (section 3.1.4) that uses information extraction techniques to extract relation triples from text (called text triples). The Metadata Processing (section 3.1.5) component prepares and stores the set of candidate DBpedia predicates for a given text predicate, that could possibly be the

mapping. The Predicate Mapping component (section 3.1.6 and details in section 3.2) maps a text triple's predicate to its matching predicate in DBpedia namespace.

## 3.1 End-to-end pipeline: Text to KG

### 3.1.1 Entity Mapping

An important step in KG construction is linking named entities to unique identifiers. In this component, we use a named entity recogniser (NER) [70] to mark all the named entities in the text. The NER algorithm uses a four-step procedure to find and mark named entities. First, it finds substrings of the input that may be entities. Second, it checks the these substrings if they are matching with any of the DBpedia resources. After that, it does disambiguation to determine the most probable named entities for the input string. Lastly, for any given user-configurations, it performs annotations and outputs the result.

Now, for an entity which has a possible mapping in DBpedia, we use the URI of such a DBpedia entity as the representation of the text entity. Otherwise, we define a custom namespace to create their URIs. The output from Entity Mapping step are the same set of sentences, but entities replaced by the URIs as defined by above conditions. For eg: "Barack Obama", "Obama" and "Barack" will be replaced by DBpedia:Barack_Obama, which is the URI of Barack Obama entity in DBpedia. Hence, output of this layer for above three instances will be,

(a) *DBpedia:Barack_Obama was born in DBpedia:Honolulu.*

(b) *DBpedia:Barack_Obama was elected in 2009 as the president of the DBpedia:United_States. He belongs to DBpedia:Honolulu.*

(c) *DBpedia:Barack_Obama served as the 44th president of the DBpedia:United_States and grew up in DBpedia:Honolulu.*

### 3.1.2 Sentence Simplification

English is a difficult language for machines to comprehend. As and when sentences grow complex in nature, extraction of relation triples from unstructured text becomes tougher. Due to complex structure of sentences, the frameworks which try to convert unstructured text into a relation triple, cease to identify correct relationships between the entities at hand. Hence, the aim of this component is to identify complex sentences and convert them into simpler sentence(s), such that the semantics is consistent even after simplification. We implement the approach proposed in [73] to achieve this. The sentence simplification framework works as follows:

1. It splits the sentences into simpler sentences based on semantics.

2. It uses a probabilistic model trained on semantic representations to delete that get rid of unnecessary words

3. It uses a substitution and reordering component based on translation and language model to simplify sentence

Here, (a) (table 2.1) remains unchanged, because it is not a complex English sentence whereas (b) and (c) change to:

(b) *DBpedia:Barack_Obama was elected in 2009. DBpedia: Barack_Obama was the president of the DBpedia:United_States. He belongs to DBpedia:Honolulu.*

(c) *DBpedia:Barack_Obama served as the 44th president of the DBpedia:United_States. DBpedia:Barack_Obama grew up in DBpedia: Honolulu.*

In section 3.3.3, we perform a separate study to evaluate the impact of this component in our system.

### 3.1.3 Co-reference Resolution

There are a lot of expressions for a given entity in unstructured text, such as pronouns and abbreviations, which act as a proxy for some real-world entity. And if these expressions are left unannotated, we may lose crucial information. Hence it is important to group all the mentions of an entity and link them to an URI. We use approach proposed by [86] to determine and replace all the linguistic expressions which refer to the same real-world entity and link them to its URI, as determined by the Entity Mapping component. The algorithm uses a multi-pass approach to do the job. In the first pass, it links two mentions if they match exactly to each other. In the second pass, it links two mentions if they match a set of syntactic rules. In the next three pass, it links two mentions if both of them have same *head matching* according to a set of rules, from high strict to low strict. In the final pass, it links the pronouns to their based on Named entity recognition and attributes like gender.

The output of this layer replaces all the co-reference chains by their URI. Only (b) changes, since it is the only sentence with a co-reference chain.

(b) *DBpedia:Barack_Obama was elected in 2009. DBpedia:Barack_Obama was the president of the DBpedia:United_States. DBpedia:Barack_Obama belongs to DBpedia:Honolulu.*

### 3.1.4 Triple Extraction

In this component, we extract relation triple from plain text. This is the primary step in extracting information from unstructured text. A relation triple is a data entity composed of a subject-predicate-object. Subject and object are real-world entities and predicate describes the relationship that the subject entity has with object entity. Below we show the triples extracted from the portion of text that talk about the birth place of Obama.

(a) $\{<$ *DBpedia:Barack_Obama; was born in; DBpedia:Honolulu* $>,$ $<$ *DBpedia:Barack_Obama; was born at; DBpedia:Honolulu,* $<$ *DBpedia:Barack_Obama; was born on; DBpedia:Honolulu* $>\}$

(b) $\{<$ *DBpedia:Barack_Obama; belongs to; DBpedia:Honolulu* $>\}$

(c) $\{<$ *DBpedia:Barack_Obama; grew up in; DBpedia:Honolulu* $>\}$

The task of extracting a relation triple from plain text can be carried out by any information extraction [2, 28, 105, 20, 68] technique. These techniques extract relation triples from text by identifying relation phrases and associated arguments in a sentence without requiring a pre-specified vocabulary. We use Open Language Learning for Information Extraction (OLLIE) [92] as the triple extractor in our implementation. The OLLIE algorithm takes a sentence as input an converts it a triplet of the form ¡*S;P;O*¿. It uses high precision seed tuples to bootstrap a large training dataset to learn learn pattern templates. These pattern templates are then used to do pattern matching on the sentences and after context analysis, triplets are generated.

### 3.1.5 Metadata Processing

This step is a preprocessing step for our main task of Predicate Mapping (discussed in section 3.2). The aim of this component is to store all possible DBpedia predicates to which a predicate from text triple can map, by using a pruning strategy. To find such a set of possible DBpedia predicates for a given text predicate, later defined as *Candidate Set*, we define two sub-tasks. Let our text triple be of the form $R_t$ =<*S;*$P_t$*;O*>. At first, we need to find the class to which $S$ and $O$ entities belong. If the $S$ and $O$ are mapped to some entities in DBpedia, we extract $S\_class$ and $O\_class$ using DBpedia class information. Otherwise, we use a Named Entity Recognizer (NER) [33] to predict it's class and map it to a DBpedia class using NER and Disambiguation (NERD) ontology [88], which is based on a bunch of NLP tools available.

For example, an NER class $Person$ maps to its equivalent DBpedia class, $DBpedia{:}Person$. Secondly, after the subject and object class of a text triple are extracted correctly, we prune the DBpedia predicates which can surely not be the mapping for $P_t$. The idea is, if the relation triple in the text $R_t$, is defined between a $Person$ class entity and a $Place$ class entity, clearly the map of $P_t$ in DBpedia namespace will also be only among the predicates which have domain as $DBpedia{:}Person$ and range as $DBpedia{:}Place$. So, instead of evaluating its similarity score with all the predicates of DBpedia, we only do it for relationships between a $DBpedia{:}Person$ and a $DBpedia{:}Place$ class. This remaining set, after pruning the unimportant predicates, is defined as the *Candidate Set (CS)* (described in section **??**) for $P_t$. We explain in detail (section 3.2.2) the difference in cost of computation of previous similarity-based approach and our approach.

### 3.1.6 Predicate Mapping

The aim of this component is to map predicate of a text triple to its matching predicate in DBpedia. From the previous component we obtained the Candidate Set ($CS$) for $P_t$. Here, we find the similarity of the text predicate with every candidate in the $CS$ using the model explained in next section. We finally map $P_t$ to the most similar candidate predicate in DBpedia. Our model is able to map "was born in", "belongs to", "was born at", "was born on" and "grew up in" correctly to DBpedia:birthPlace. Hence, the output for the examples becomes:

(a) *<DBpedia:Barack_ Obama; DBpedia:birthPlace; DBpedia:Honolulu>*

(b) *<DBpedia:Barack_ Obama; DBpedia:birthPlace; DBpedia:Honolulu>*

(c) *<DBpedia:Barack_ Obama; DBpedia:birthPlace; DBpedia:Honolulu>*

Simply extracting relation triples using OLLIE (as shown in fig 2.6) will give us a graph with four nodes (entities: "Barack Obama", "He", "Barack", "Honolulu") and five edges (predicates: "was born at", "was born in", "was born on", "belongs to") which clearly holds redundant information. Our approach is able to resolve all the identical mentions of entities and predicates hence forming a more precise graph with two nodes and one edge between them, i.e. *<DBpedia: Barack_ Obama;DBpedia: birthPlace;DBpedia: Honolulu>*.

The output of the Predicate Mapping step for a particular text triple is a triple that has its entities and predicate linked to DBpedia. The collection of such triples is our constructed KG. Thus, our system has constructed a KG over a specific text article or document. This KG makes the raw text information available for document specific tasks such as question answering and information retrieval. Furthermore, if there is some information present in this text, but not in DBpedia then we could add the missing information to DBpedia, by merging it with our constructed KG.

## 3.2 Predicate Mapping Model

Going back to the Figure 2.6, we can see that that Barack Obama has been linked to Honolulu using multiple relationships, "was born in", "was born at", "was born on", "grew up in". Owing to to synonymity in Natural Language, this becomes the root-cause for an explosion in number of triples with redundant information. This causes the applications of Knowledge Graphs such as summarization, question-answering, querying, etc that much in-efficient. Hence in order to remove this redundancy, it is crucial to map all the above relationships to a URI of a publicly accepted namespace. In our approach, we map all of our relationships (or predicates) to DBpedia namespace to tackle the explosion problem. In section 3.1.6 we had a brief overview of the Predicate Mapping component. In this section, we will cover details on how that is acheived.

Let $<S_{db}; P_t; O_{db}>$ denote a relation triple extracted from the text using the above pipeline. Our aim is to find a DBpedia predicate (say $P_{db}$) which is semantically similar to $P_t$ and map $P_t$ to $P_{db}$. We use a deep learning based similarity approach to find this $P_{db}$.

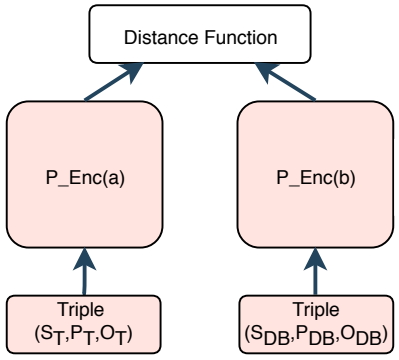


**Figure 3.2** Overview of Predicate Mapping Model

The idea is to get a vector representation of the text predicate and compare its similarity with possible DBpedia candidates. The representation of the predicate is formed using the Predicate Encoder ($P\_Enc(a)$ and $P\_Enc(b)$ as shown in fig.3.2). The predicate encoder is a multi-stage architecture (described in Section 3.2.1) which uses subject and object class as initial context to generate the representation since they store valuable information about the predicate.

Let the final encoded vector representation of the two predicates obtained from the predicate encoder be $H_t = [x_1, .., x_k]$ and $H_{db} = [y_1, .., y_k]$. The similarity between the two vector representations $H_t$ and $H_{db}$ is given by the following equation:

$$f(H_t, H_{db}) = exp(-||H_t - H_{db}||) \in [0, 1] \tag{3.1}$$

In the next section, we define our Predicate Mapping model.

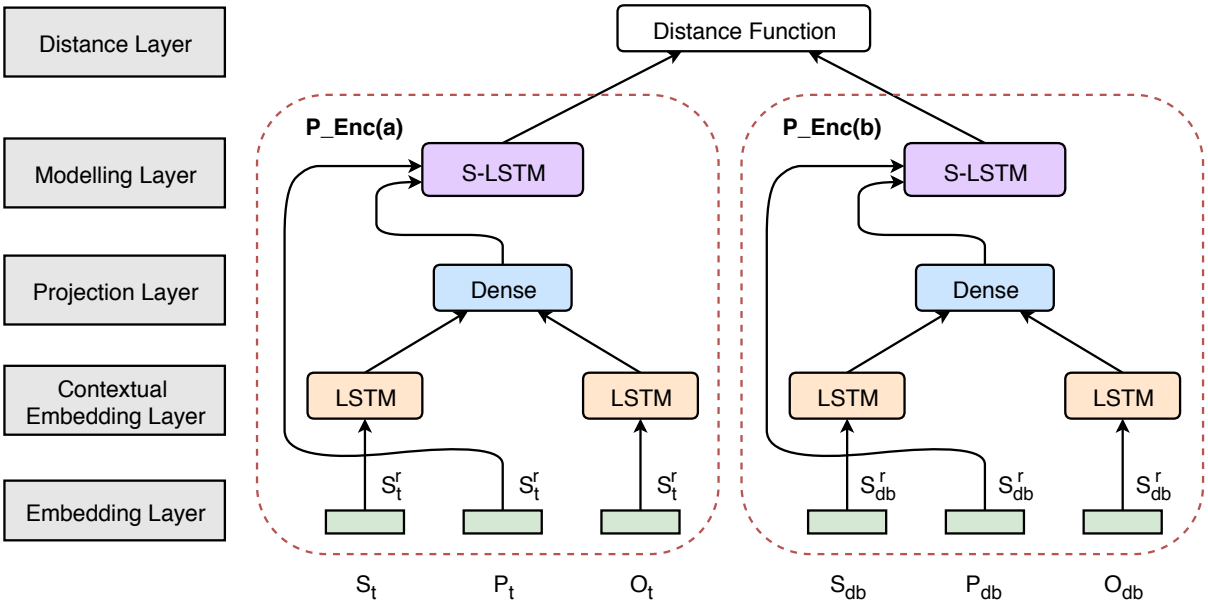

**Figure 3.3** End-to-end Predicate Mapping model

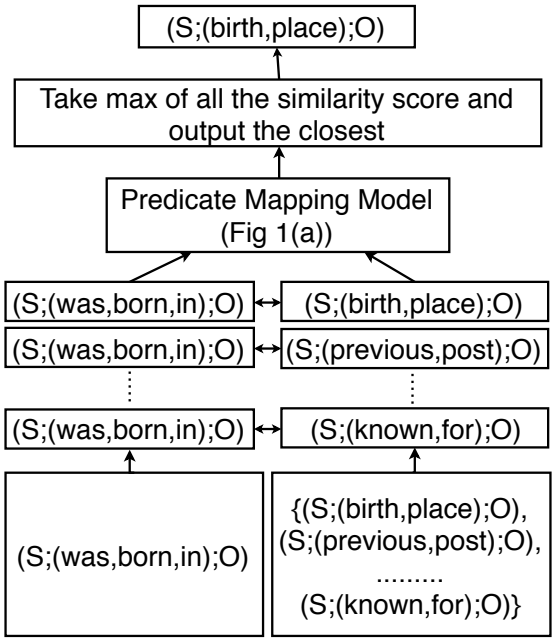


**Figure 3.4** An illustrative example of Predicate Mapping, where S and O are Subject Class and Object Class respectively

### 3.2.1 Model

Our Predicate Mapping model (shown in fig 3.3) is a hierarchical multi-stage Siamese Network, inspired by [72]. A Siamese Neural Nxetwork is an artificial neural network that uses the same weights while working in tandem on two different input vectors to compute comparable output vectors. In our use case, we are want to compare two sentences having synonymity between text and DBpedia. Hence, the idea of running two identical neural networks on two different inputs fits well for our usecase. Therefore, we picked Siamese neural network. The model consists of five layers. The *Embedding Layer* maps each word to a vector space using a pre-trained embedding model. The *Contextual Embedding Layer* utilizes the vector representations of surrounding words to refine the embeddings of the words. The *Projection Layer* forms the initial state for our Modeling Layer by densely combining subject class and object class embeddings. The *Modeling Layer* employs a Siamese Recurrent Network to create final embedding of the predicate using the combined embedding of subject class and object class as context. The *Distance Layer* computes the final similarity score.

**Embedding Layer** This is a word embedding layer which maps each word to a high-dimensional vector space. In our model, we use $d$-dimensional pre-trained GloVe word embeddings [80]. Let

$\{s_1, s_2, ...s_{k_t}\}$, $\{p_1, p_2, ...p_{l_t}\}$ and $\{o_1, o_2, ...o_{m_t}\}$ denote the sequence of words in subject, predicate and object of a text triple and similarly for a DBpedia triple. The output of this layer consists of three matrices for text triple: $S_t^r \in \mathbb{R}^{d \times k_t}$, $P_t^r \in \mathbb{R}^{d \times l_t}$, $O_t^r \in \mathbb{R}^{d \times m_t}$.

$$
\begin{aligned}
S_t^r &= [s_1^r s_2^r ... s_{k_t}^r] \\
P_t^r &= [p_1^r p_2^r ... p_{l_t}^r] \\
O_t^r &= [o_1^r o_2^r ... o_{m_t}^r]
\end{aligned}
$$

where $s_i^r \in \mathbb{R}^{d \times 1}$, $p_i^r \in \mathbb{R}^{d \times 1}$ and $o_i^r \in \mathbb{R}^{d \times 1}$ represent the embedding of $s_i$, $p_i$ and $o_i$. Similarly for DBpedia triple, we obtain: $S_{db}^r$, $P_{db}^r$ and $O_{db}^r$.

**Contextual Embedding Layer** In this layer, we use LSTM [39] to output the encoded representation of a given sequence. We use four LSTMs, two for $S_t$ and $S_{db}$ to form subject class encodings and the other two for $O_t$ and $O_{db}$ to form object class encodings.

LSTMs adapt feedforward neural networks for sequence data where at each time-step (say $x$), updates to a hidden state vector $h_x$ are performed. In our case, the input sequence data is, $[s_1^r s_2^r ... s_{k_t}^r]$. These updates also rely on a memory cell containing four components (which are real-valued vectors): a memory state $c_x$, an output gate $q_x$ (that determines how the memory state affects other units), an input gate $i_x$ (that controls what gets stored in memory) and a forget gate $f_x$ (that controls what does not get stored in memory). Below are the updates performed (for $S_t$) at each time-step $x \in 1, ..., k_t$:

$$
\begin{aligned}
i_x &= sigmoid(W_i s_x^r + U_i h_{x-1} + b_i) \\
f_x &= sigmoid(W_f s_x^r + U_f h_{x-1} + b_f) \\
\widetilde{c_x} &= tanh(W_c s_x^r + U_c h_{x-1} + b_c) \\
c_x &= i_x \cdot \widetilde{c_x} + f_x \cdot c_{x-1} \\
q_x &= sigmoid(W_q s_x^r + U_q h_{x-1} + b_q) \\
h_x &= q_x \cdot tanh(c_x)
\end{aligned} \tag{3.2}
$$

where $W_i, W_f, W_c, W_q, U_i, U_f, U_c, U_q$ are the trainable weight matrices and $b_i, b_f, b_c, b_q$ are the trainable bias-vectors. We take the last hidden state of the LSTM (i.e. $h_{kt}$) as our final representation. Hence, we obtain $G_t \in \mathbb{R}^{d \times 1}$, $U_t \in \mathbb{R}^{d \times 1}$ for subject classs ($S_t$) and object class ($O_t$) of text triple and $G_{db} \in \mathbb{R}^{d \times 1}$, $U_{db} \in \mathbb{R}^{d \times 1}$ for subject class ($S_{db}$) and object class ($O_{db}$) of DBpedia triple.

**Projection Layer:** This layer aims to prepare context for Modeling Layer using G and U vectors. We define

$$J_t = \{G_t; U_t\}$$
$$J_{db} = \{G_{db}; U_{db}\}$$

as the concatenation of subject class and object class representation of text triple and DBpedia triple respectively. This layer maps $J_t \in \mathbb{R}^{2d\times 1}$ to $Z_t \in \mathbb{R}^{d\times 1}$ by applying a Dense Layer and similarly $J_{db} \in \mathbb{R}^{2d\times 1}$ to $Z_{db} \in \mathbb{R}^{d\times 1}$. $Z_t$ and $Z_{db}$ store the combined representation of subject-object class of text and DBpedia triple and serve as the context for finding the representation of their respective predicates.

**Modeling Layer:** In this layer, we use another pair of LSTMs to create encoded representation of text predicate and candidate predicate. We use the output of the previous layer as context, i.e., $Z_t$ and $Z_{db}$ become the first hidden states ($h_0$ in eq. (3.2)) for the two LSTMs respectively. The input to this LSTM is $P_t$. We obtain $H_t \in \mathbb{R}^{d\times 1}$ as the last hidden state of this *S-LSTM* (as shown in fig 3.3). Similary, we obtain $H_{db} \in \mathbb{R}^{d\times 1}$. $H_t$ and $H_{db}$ represent the final text and DBpedia triple encoding, respectively.

**Distance Layer:** This layer computes the similarity score between $H_t$ and $H_{db}$ using the equation 3.1. This score is used to choose the most similar predicate amongst a set of predicates called *Candidate Set* (discussed in detail in the next section).

### 3.2.2 Candidate Selection

In the previous work, the approach to map predicate of a text triple to DBpedia predicate requires high computation cost. The high computation is because it requires an entire scan of DBpedia. Thus making the computation cost in the order of total number of triples in DBpedia ($1.3 \times 10^9$). This step has many unnecessary comparisons, for example, an extracted predicate "*was born in*" is being compared with DBpedia predicates such as *DBpedia:biggestCity* and *DBpedia:altitude*. To overcome this, we identify the DBpedia predicates which can be the possible mapping for the text triple's predicate.

The idea is as follows: each predicate, say $P_{db}$ in DBpedia has a unique subject class and object class, defined in the RDF-schema as *rdfs:domain* and *rdfs:range* respectively. This defines that, in any relation triple which has $P_{db}$ as it's predicate, say $<S; P_{db}; O>$, all such $S$'s and $O$'s belong to the class defined by *rdfs:domain* and *rdfs:range*, respectively. We exploit this information about a text triple, to narrow down our search space significantly. DBpedia ontology forms a subsumption hierarchy [59], hence our

new search space does not lose any potential candidates. Thus, we introduce a notion of *Candidate Set (CS)* for a text triple.

Formally, a $CS$ for a text triple, say $R_t = <S; P_t; O>$, is a set of all the DBpedia predicates (stored as $<S\_class; P_{db}; O\_class>$ in the $CS$) whose predicates occur between $S$'s class and $O$'s class. In our experiments, we find that the average size of the *Candidate Set* for a predicate is approximately 200, which reduces the search space by an order of $10^6$ for each text predicate. Now that we have defined $CS$ for each extracted predicate, we describe the steps to output the *closest* (most similar) DBpedia predicate for a particular $R_t$. Consider $C_t = <S\_class; P_t; O\_class>$ and the *Candidate Set* for $P_t$,

$$CS = \{<S\_class; P^1_{db}; O\_class>, \\ ...., <S\_class; P^K_{db}; O\_class>\} \tag{3.3}$$

where K is the size of the *Candidate Set*.
Let us do a running example for *Candidate Selection*. Consider a $R_t$ and its corresponding $C_t$ and *Candidate Set (CS)* be

$R_t$ = *<DBpedia:Barack_Obama; was born in; DBpedia:Honolulu>*

$C_t$ = *<DBpedia:Person; was born in; DBpedia:Place>*

$CS = \{$*<DBpedia:Person; DBpedia: birthPlace; DBpedia:Place>*,

*<DBpedia:Person; DBpedia: previousPost; DBpedia:Place>*,

*... ,<DBpedia:Person; DBpedia:knownFor; DBpedia:Place>*$\}$

where K = 255 since there are 255 different predicates between *DBpedia:Person* and *DBpedia:Place*[1]. The steps are:

1. We calculate the similarity score for $C_t$ and $i^{th}$ element of the *Candidate Set* ($CS[i]$), $\forall i \in [1, K]$ by feeding $C_t$ and $CS[i]$ to our Predicate Mapping model. That is, we calculate similarity score of *<DBpedia:Person; was born in; DBpedia:Place>* with *<DBpedia:Person; DBpedia:birthPlace; DBpedia:Place>* and so on with all the elements of *CS*.

2. We find the candidate from the *CS* which gives maximum similarity score with $C_t$, say $CS[j]$. In our example, we find that *<DBpedia:Person; DBpedia:birthPlace; DBpedia:Place>* has the maximum similarity with *<DBpedia:Person; was born in; DBpedia:Place>*.

3. Finally, we map $P_t$ with the predicate of $CS[j]$ by replacing all occurrences of $P_t$ with $P^j_{db}$. That is, we update our $R_t$ to *<DBpedia:Barack_Obama; DBpedia:birthPlace; DBpedia:Honolulu>*.

So effectively, we compare *"was born in"* with only 255 predicates in DBpedia. We input each pair $C_t$ and $CS[i]$ to our model (as shown in 3.4) and get similarity scores for each of them. The vector

[1]We use SPARQL endpoint (http://yasgui.org/) to query DBpedia

representation of *"was born in"* using *Person* and *Place* as context comes out to be closest (using eq. 3.1) to the vector representation *DBpedia:*$birthPlace$ calculated using *Person* and *Place* as context. If the size of *CS* for any text predicate is zero, we discard that text triple.

### 3.2.3 Scalability

Consider a text triple $R_t = <S_t; P_t; O_t>$ and let its mapping be $R_{db} = <S_{db}; P_{db}; O_{db}>$. In the previous similarity based approach [26], it does an entire scan of DBpedia to find this $P_{db}$. We find that it makes a lot of unnecessary computations. In our approach, we get rid of them by defining our *Candidate Set* using SPARQL queries. *As per the latest release of DBpedia*[2] *, in order to map a single* $P_t$ *to* $P_{db}$*, our system requires approximately 200 computations while T2KG[53] essentially requires computation in the order of size of DBpedia, that is,* $1.3 \times 10^9$.

To train this model in an end-to-end fashion, one would require a dataset consisting of pairs of similar triples. But, to the best of our knowledge, there is no such dataset available. Therefore, we need to train the components independently. We train two different models on two different datasets: SNLI[9] corpus and SemEval STS [51]. The SNLI dataset is a set of human-written English sentence pairs manually labeled for the task of Recognizing Textual Entailment (RTE). While the SemEval STS dataset consists of labeled cross-level semantically similar pairs of a paragraph and a sentence. The Predicate Mapping model is divided into three parts for training.

### 3.2.4 Contextual Embedding Layer Training

This part aims at obtaining the weights ($W_i, W_f, W_c, W_q, U_i, U_f, U_c, U_q$ in eq. (3.2)) for the Contextual Embedding Layer. We train the LSTM in this layer similar to a Language Model (LM)[71]. The goal of an LM is to predict the $(k+1)^{th}$ word of sequence, given the previous $k$ words. A huge chunk of text is given as input to the LSTM to train it as an LM. This way, it learns to encode a sequence of words and hence we use this trained LSTM as a sequence encoder in the Contextual Embedding Layer. We transform the sentences in the dataset into a sequence of tokens or words that is used for training the model. The weights obtained by training this model are used in all the four LSTMs in the Contextual Embedding Layer.

### 3.2.5 Projection Layer Training

This part aims at obtaining the weights for the Projection Layer. Since this layer, at its core, aims to reduce the dimension of the output of the Contextual Embedding Layer from $2d$x1 to $d$x1, we use an autoencoder[38] for this task. The input to the autoencoder is the concatenation of the two $d$x1 outputs of the LSTMs of the previous layer, which gives us a $2d$x1 vector. We store such $2d$x1 vectors for multiple triples and use them to train the autoencoder. The input is first transformed into a $d$x1 vector

[2] *https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10*

using a weight matrix $W_1 \in \mathbb{R}^{2d \times d}$, and then this intermediate vector is converted back to a $2d$x1 vector using another weight matrix $W_2 \in \mathbb{R}^{d \times 2d}$. The first part of the autoencoder reduces the input to $d$x1 vector. Hence, we use this $W_1$ matrix, as our weight matrix for this layer.

### 3.2.6 Modeling and Distance Layer Training

This part aims at obtaining the weights of the Modeling Layer. We train the LSTMs in Modeling Layer and the Distance Layer as an end-to-end Siamese Network [74]. For a given pair of sentences in the dataset, first and second sentence are respectively fed to the two *S-LSTM*s. In the Distance Layer, similarity score between the last hidden states of the two *S-LSTM*s ($H_t$ and $H_{db}$) is calculated using eq. 3.1. With similarity label known, the corresponding contrastive loss[36] is back propagated to the *S-LSTM*s according to the equation:

$$L(Y, H_t, H_{db}) = (Y)\frac{1}{2}(D)^2 + (1-Y)\frac{1}{2}\{max(0, 1-D)\} \tag{3.4}$$

where $D$ is the similarity score obtained using 3.1 and Y is the similarity label.

## 3.3 Experiments and Results

The aim of these experiments is to evaluate the performance and impact of different components in our pipeline. We also show how the Sentence Simplification component improves the quality and quantity of triples extracted. We perform following four experiments:

### 3.3.1 Automatic Evaluation

The aim of this experiment is to evaluate our Predicate Mapping model. Since there is no gold standard dataset for evaluation, we build a testable dataset by setting a ground truth. Consider a text triple, say $R_t = <S_t; P_t; O_t>$, which after performing Entity Mapping would become $R_t = <S_{db}; P_t; O_{db}>$ where $S_{db}$ and $O_{db}$ are the corresponding DBpedia URIs. Now, if there is only one predicate, say $P_{db}$, between $S_{db}$ and $O_{db}$ in DBpedia then $P_{db}$ is the ground truth for $P$. Such $R_t$ constitute our test dataset.

This experiment is performed using 140,000 randomly selected Wikipedia articles. A total of 3,271,660 triples were extracted and the ground truth was established for 58,842 triples. We compare our system's results with a baseline model defined below[3].

In the baseline model, we use cosine distance metric to calculate the similarity score between a text predicate and a DBpedia predicate. Let a text predicate ($P_t$) be a sequence of $\{t_1, t_2, ..., t_n\}$ words. We encode $P_t$ as the sum average of GLoVE vector embeddings (which are generated based on neural networks) of the $t_i's$. Similarly a DBpedia predicate ($P_{db}$) is encoded as a sum average of GLoVE

[3]We could not compare our results with the state-of-the-art paper ([53]) due to ambiguities in the implementation details of their paper

| | Recall | Precision | F1-score |
|---|---|---|---|
| Baseline - GLoVE + Cosine | 0.2561 | 0.3478 | 0.2949 |
| Rule-based | 0.3514 | 0.4627 | 0.3994 |
| SNLI trained - Our approach | **0.6069** | **0.7684** | **0.6781** |
| STS trained - Our approach | 0.5382 | 0.6762 | 0.5994 |

**Table 3.1** Automatic Evaluation results

embeddings of $\{d_1, d_2, ..., d_m\}$ words. The cosine similarity score between these two encodings $P_t$ and $P_{db}$ is defined as:

$$CosineSimilarityScore = \frac{P_t \cdot P_{db}}{||P_t|| \cdot ||P_{db}||}$$

Table 3.1 shows comparative results using cosine metric baseline, the rule-based approach [26], and our model trained on SNLI dataset [9] and SemEval STS dataset [51]. We observe that training our model on SNLI dataset gives the high *F1 score of 0.678*. It is interesting to note that the model is trained on text similarity dataset to identify the similar DBpedia triples (essentially verb phrases).

It took approximately *35 hours* to complete the steps before Predicate Mapping for 140,000 Wikipedia articles, a majority of which is network latency time due to large number of SPARQL [83] queries in defining *Candidate Set* for each triple. For the final step of Predicate Mapping, we were able to generate the mappings in approximately *six hours using the pre-trained model for 58,842 triples*.

### 3.3.2 Manual Evaluation

Since there is no standard evaluation metric to evaluate the quality of the Predicate Mapping step, it is necessary to perform a manual evaluation since in Experiment 1, we are able to consider a special case when only one predicate exists between a particular subject and object. In this experiment, we randomly select 200 sentences and were able to extract 416 triples using our triple extraction step. We then map the predicates of these triples manually to DBpedia predicates. We feed this set of triples to our Predicate Mapping model and observe that we are able to map predicates with a *recall of 0.633* and with a *precision of 0.771*.

Based on our error analysis of the extracted triples, results show that the most errors i.e. *35.2% are caused due to triple extractor*. While *30.7% errors are caused from co-reference resolution step*, *18.3% due entities* and *15.8% due predicates*. The rule-based approach [26] indicates that the most errors (*46%* approximately) were caused due to wrong Predicate Mapping, making this step a severe drawback. *Our Predicate Mapping along with the Sentence Simplifier is able to reduce this bottleneck leading to improvement of 0.22 in F1 score.*

| Sentence Type | Correct | Incorrect | Misleading | Total |
|---|---|---|---|---|
| Complex | 109 | 40 | 64 | 213 |
| Simple | 171 | 33 | 54 | 258 |

**Table 3.2** Sentence Simplification Evaluation results

### 3.3.3 Sentence Simplification Evaluation

Previous studies show that significant amount of errors are due to limitations of triple extraction, especially, when dealing with complex sentences. Hence in this experiment, we aim to quantify the extent to which the Sentence Simplification model improves the triple extraction performance, when dealt with complex sentences. We analyze OLLIE information extractor performance when it is fed with i) complex sentences only ii) simplified sentences using the Sentence-Simplification idea [73]. We randomly choose 100 complex sentences from Zhu dataset [110] and obtain their corresponding simple sentences and run OLLIE triple extractor on these simple sentences. We then manually classify each triple in three of the following categories: (a) *Correct triple* is a relation triple that can be justified as true, given the sentence, (b) *Incorrect triple* is a relation triple that can not be justified as true, given the sentence, and (c) *Misleading triple* is a triple which is neither correct nor incorrect but possess incomplete information, and may lead to incorrect relation if added in the KG.

As shown in 3.2, there is not only increase in total number of relation triples extracted, but also a $36.25\%$ *increase in the number of correct triples* obtained and a $17.5\%$ *decrease in the number of incorrect triples* from OLLIE when fed with simplified sentences vs complex sentences. This entails that the simplification component helps construct larger KG with less ambiguity and less incorrect relationships.

### 3.3.4 Redundancy Reduction Evaluation

The aim of this experiment is to evaluate how well our system is able to match similar relationships and entities into a homogeneous set of URIs in order to reduce redundancies in the KG. We take a Wikipedia article, say $A$, and paraphrase it to $A'$. We combine $A$ and $A'$ into a single document, say $B$. The idea is, this $B$ will give out redundant triples since same information is paraphrased and added to it (in the form of $A$ and $A'$) and we want to evaluate how well these redundant triples are resolved by our system. Ideally, the triples generated from $A$ should be identical to the triples generated from $B$. We perform our end-to-end KG generation system over $A$ and $B$ independently and study the (a) number of *accurate triples* - triples which are identical across $A$ and $B$, and (b) number of *additional triples* - which are present in $B$ but are absent in $A$. We use an online paraphrasing tool called spinbot[4] to paraphrase an article. We collect $1,000$ such $A$'s and build their corresponding $B$'s separately for two different topics of Wikipedia Science and History. Some examples of Science topic are Chlorophyll,

[4]www.spinbot.com

| | Accurate | Additional | Total Triples | Accuracy |
|---|---|---|---|---|
| History | 1895 | 278 | 2173 | 0.872 |
| Science | 948 | 217 | 1165 | 0.813 |

**Table 3.3** Redundancy Reduction Evaluation results

Photosynthesis, etc and History topic are French Revolution, World War, etc. We obtain a total of 2173 triples from History articles and 1165 triples from Science articles. We define accuracy for this experiment in the following manner:

$$Accuracy \quad = \quad \frac{n(a)}{n(a) + n(b)}$$

where $n(a)$ and $n(b)$ represent the number of *accurate triples* and *additional triples*, respectively. This accuracy will evaluate to $1.0$ when the paraphrasing module and our system perform perfectly. The results show that, the average accuracy over Science and History articles is 84.3%. As shown in table 3.3, we observe that the *accuracy for History articles is* $87.24\%$, slightly higher than $81.37\%$ *for Science articles*. We randomly sample 50 articles each of Science and History to qualitatively analyze the difference in accuracy and following are the observations:

- Science articles have a lesser share of facts but a lot of complex definitions which our Sentence Simplifier is often not able to correctly simplify. This could be improved by using a labeled training dataset consisting of complex scientific definitions and their corresponding simplified definitions for learning.
- History articles are almost entirely fact driven. This leads to high number of total triples as well as less ambiguity due to paraphrasing.

## 3.4 Cleaning KGs

For knowledge-driven applications, the correctness and quality of the content of Knowledge Graphs becomes very important. For that matter, it becomes important to clean noisy or possibly incorrect facts from Knowledge Graph.

One of the constant issues in Knowledge Graph based applications is identifying and fixing corrupt data, and failing to do so can lead to erroneous analytics and unreliable conclusions. The interest in data cleaning issues, including new abstractions, interfaces, scalability approaches, and statistical methodologies, has increased recently in both industry and academics. We will first describe a taxonomy of the data cleaning literature in which we highlight the recent interest in methods that use constraints, rules, or patterns to find errors, which we refer to as qualitative data cleaning, in order to better understand the

recent developments in the field. With a number of illustrated examples, we will present the most recent techniques and also point out their shortcomings.

Large volumes of data are become easier for businesses to store and collect. These data sets can help with better decision-making, more in-depth analytics, and a rising amount of machine learning training data. However, poor data quality continues to be a significant problem, as it can result in erroneous judgments and inaccurate analyses. Missing values, typos, mismatched formats, duplicate entries for the same real-world item, and infractions of business rules are a few examples of frequent errors. Data cleaning has been a crucial field of database study since analysts must take the impact of unclean data into account before making any conclusions - see Johnson and Dasu [50] and Rahm and Do [87].

Data cleaning has gained increased attention from both industry and academics [14]. Four categories can be used to group existing data cleaning techniques and prototypes:

- KG powered cleaning techniques;
- Statistical cleaning techniques
- User (specialist or public) interaction cleaning techniques

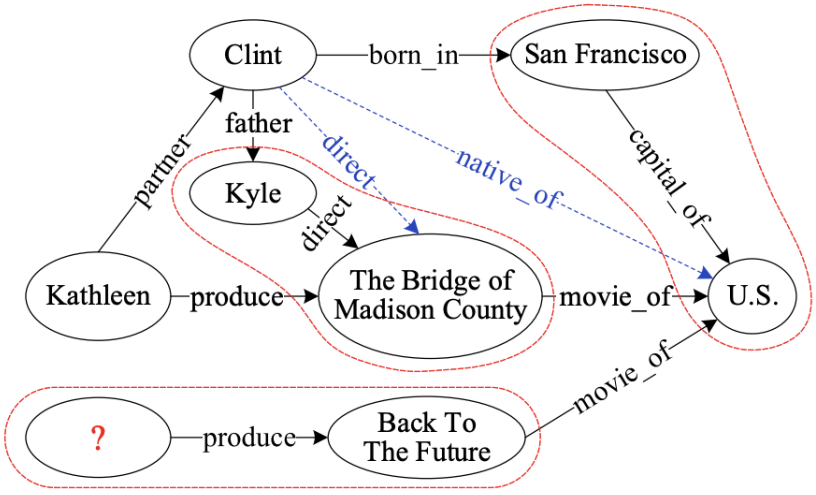


**Figure 3.5** An illustration of KG that needs cleaning

Let's discuss an example of a KG cleaning scenario (Figure 3.5). There are total three problems in the given Figure:

1. $<$ *Kyle ; direct ; The Bride of Madison County* $>$ - "Kyle" is not the director of the movie "The Bridge of Madison County",
2. $<$ *San Francisco ; capital_of ; U.S* $>$ - San Francisco is not the capital of the US
3. $<$ *? ; produce ; U.S* $>$ - '?' is not defined

Items 1 and 2 are wrong factually whereas item 3 is incomplete. As an example, let's talk about triplet 2: $<$ *San Francisco ; capital_of ; U.S* $>$. Inorder to fix this, we can either modify the relationship to *city_of* or change *San Francisco* to *Washington D.C*. Unfortunately, current approaches for knowledge graph construction cannot detect incorrect values for entities or relationships in these triples, let alone indicate which adjustment is more appropriate nor rectify the errors.

Cleaning dirty data, or identification and rectification of incorrect values, is a crucial and efficient technique to further enhance the data accuracy of KGs. The ability to recognise incorrect values is referred to in the "identification", while the ability to fix incorrect values is referred to in the "rectification".

For general graph networks, there are many error detecting methods that have been known to identify ([10], [17], [29], [31], [109]) and rectify [30] errors. Most of these approaches are rule-based. However, gathering enough data quality rules requires a lot of work, which reduces the efficiency of such cleaning techniques. It is absurd to believe that all necessary rules might be found when a graph's data is complex. As a result, it is challenging to fix mistakes that are not covered by any of the rules mentioned and often many errors are still left out.

Knowledge graph embedding techniques, whose goal is to convert words from a corpus into various values in vector spaces while preserving their semantic content, are becoming more widely available. For knowledge graph embedding, each triplet of a KG must follow the causality rules (can be treated as a weighted rule). A factually correct triplet must have a strong causality, whereas, a triplet is more likely to be incorrect if the causation is weak.

Furthermore, knowledge graph embedding is able to learn the causalities that possible triplets should obey in addition to automatically assessing the magnitude of the causalities followed by the given triplets.

### 3.4.1 Error Detection

Finding anomalies or errors is the first step when dealing with a dirty database instance. Our taxonomy of qualitative error detection is shown in Figure 1. Every technique must solve the following three issues: The first three are a) "What to Detect?", b) "How to Detect?", and c) "Where to Detect".

**Type of Error: What to Look for?** The type of errors that are captured can be used to categorise qualitative error detection approaches. What languages are used to define the patterns or limitations of a legal data instance, in other words. Integrity constraints (ICs), a subset of first order logic, are widely used to encapsulate the data quality standards that the database should adhere to, including functional dependencies (FDs [6])) and Denial Constraints (DCs [15]).

Despite the fact that record duplication might be seen as a breach of an integrity constraint (key constraint), we acknowledge the substantial body of research that focuses on this issue. For the exact scoping, we treat duplication separately from other types of integrity constraints. Automatic discovery strategies are crucial and have been proposed for numerous ICs because manual developing such ICs [15] or patterns requires extensive domain knowledge and takes time. We categorise IC discovery

methods into schema-driven and instance-driven methods, and we will talk about and contrast these two methods.

**Automation: How to Detect?** Some approaches on error detection are fully automatic such as Holistic data cleaning [15], DBRx [13], while other approaches, such as CrowdER [100], Scorpion [109], etc involve human intervention.

### 3.4.2 Error Repairing

KG Error Repairing is a process of repairing errors in triplets as shown in Figure 3.5. Every repairing technique must answer the following three questions, a) "What to Repair?", b) "How to Repair?" and c) "Where to repair?". In the following sections we will discuss the impacts on the techniques' efficiency and design.

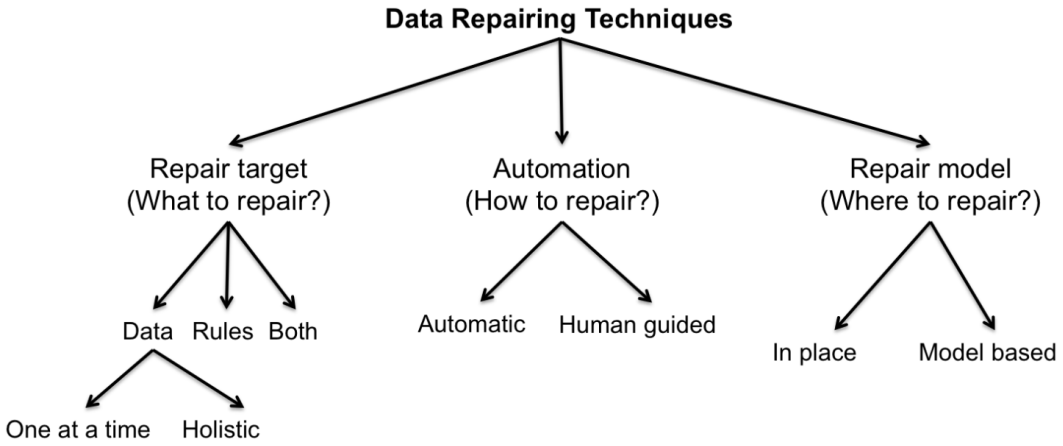


**Figure 3.6** Error Detection Approaches

**Repair Target: What to Repair?** Different assumptions are made about the data and the quality rules by different repair algorithms:

1. trust the Integrity Constraints - Meaning that at no point can the integrity constraints be altered, only the triplet data can be modified.

2. trust the data completely and allowing relaxation to constraints - this can mean that the schema can be evolved and obselete and old rules can be updated

3. looking at the prospect of altering the limits as well as the data [11].

The driver of the repairing activity, or the kinds of faults they are focusing on, can further categorise strategies by either trusting the rules and altering only the fact triplets. The vast majority of procedures

just correct one form of error at a time while fixing data, whereas newly developed solutions take interactions between many types of faults into account and correct data holistically.

**Automation: How to Repair?** We categorise suggested methods in terms of the equipment utilised throughout the repair process. We focus on the level of automation in the repair procedure and the level of human intervention needed. Some approaches for repair involve human intervention. People are involved either to confirm the modifications, to make suggestions for improvements, or to teach machine learning models used for automatic repair. However, some techniques work along the lines of minimizing the cost between original KG *k* and the repaired KG *k'* using a cost function model and hence these approaches are fully automatic [108].

**Repair Model: Where to Repair?** We categorise suggested methods according to whether they alter the database in-place or create a model to outline potential fixes. The majority of suggested ways fix the database while it is still in use, destroying the originality. Generally, a model is constructed to describe the various approaches to repair the underlying database for non on-site repairs. These models respond to queries using various probabilistic strategies, such as sampling from all possible repairs [10].

A sample of data repair methods employing the taxonomy is shown in Table 3.6.

## 3.5 Summary

Most organizations have high number of textual documents that need to be processed. But domain specific document processing has not seen a lot of work and different kinds of metric are needed to evaluate such a system. In this work, we present an end-to-end system for construction of Knowledge Graph from unstructured text. Our major focus lies in:

1. Improving the Predicate Mapping step for greater searchability in applications such as question answering and information retrieval.

2. Developing a pruning strategy to make our KG generation system scalable.

3. Studying the impact of the Sentence Simplification component in improving the quality of open Information Extraction techniques and evaluating the redundancy reduction in different domains of Wikipedia. Furthermore, these two experiments are novel and reproducible.

4. Evaluating the quality of constructed KG by performing automatic and manual experiments

Even though text data from different sources have vocabulary gap, our system is able to construct a Knowledge Graph with a *F1-score of* $0.678$ as shown in table 3.1.

*Chapter 4*

# SimX: KG-based Document Similarity Classification with Explanation

## 4.1 Overview

In 2.2, we established fundamentals on the importance of document similarity in Natural Language Processing and different approaches to do so. In this chapter, we will study the limitations of those approaches and propose our novel multi-level Knowledge Graph based similarity classifier, which can be coupled with ***any*** of the semantic approaches to enhance the similarity scoring system.

## 4.2 Limitations

We saw in 2.2 various number of document similarity approaches. In 2.2.2 we discuss different distance formulae between two pieces of text to determine similarity between them, such as Euclidean distance, Manhattan distance, Cosine similarity, and more sophisticted semantics based Word Mover's distance. In 2.2.3 we focussed on similarity approaches which used string literals to compute distance between two texts. In those, some focused on character-level differences like Longest Common Sub-string, DL-edit distance, etc. while some focussed on phrase-level differences, like Dice's coefficient, Jaccard similarity. Then we discuss corpus based similarity approaches like, Latent Semantic Analysis, Normalized Google distance, TF-IDF, and so on. All these approaches lack the ability to encapsulate structure and the connected-ness of the text. On top of that, similarity can mean differently for different people and contexts. Hence it is sometimes difficult to interpret what a similarity score of 0.8 can mean. Not only that, if two pairs of topics have same similarity score, it can sometimes become challenging to make an explanation so as to why they have exactly the same similarity scores. None of the current document similariity solutions provide explanation for the similarity scores.

Keeping in mind the limitations of current state of work, we propose a novel Knowledge Graph based Document similarity modeling. Given two documents and their topics, we propose a *multi-level KG driven comparative study of documents with explanations*.

## 4.3 Our Approach

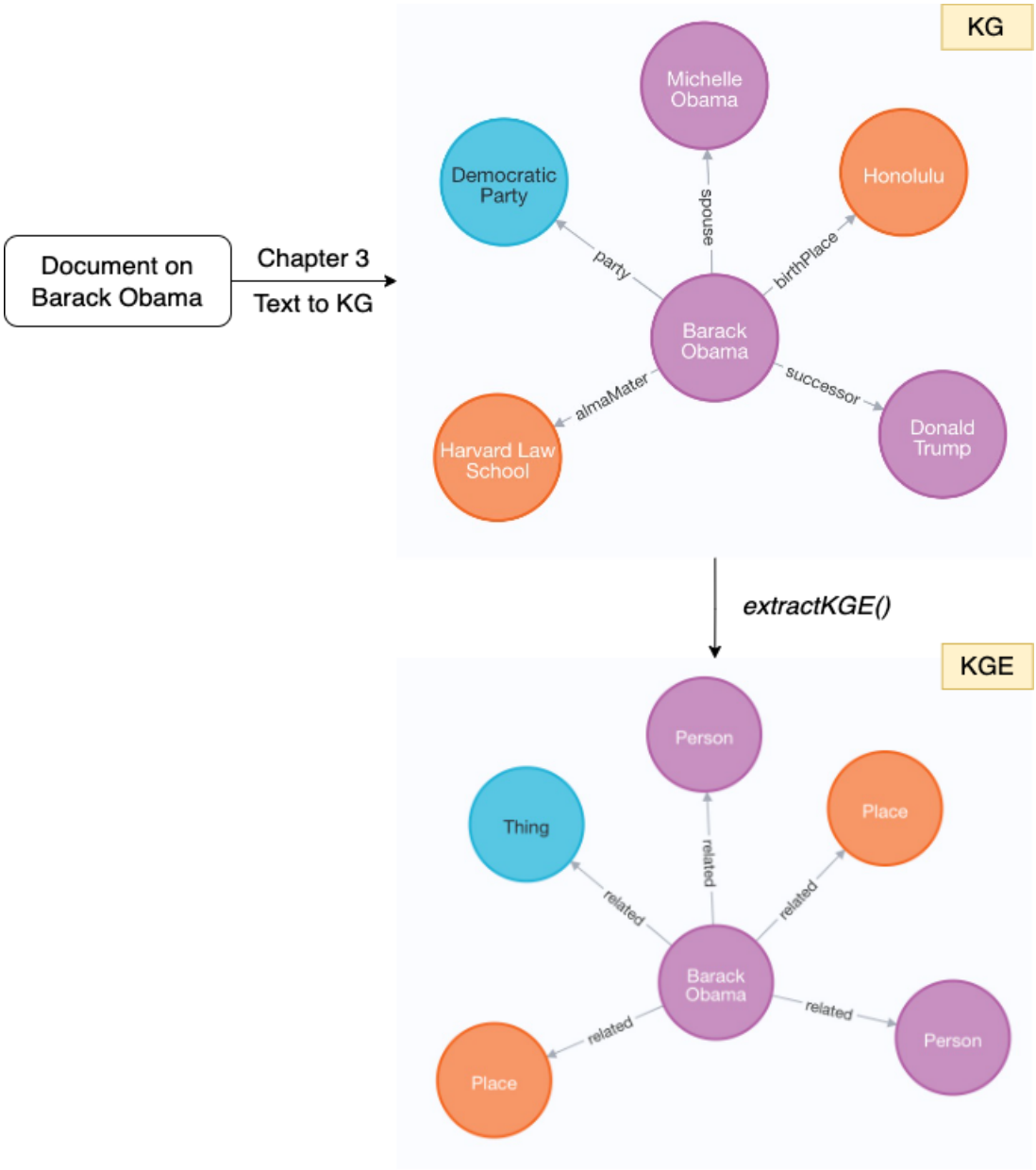


**Figure 4.1** Overview of Document to KGE construction

Given two text documents $d_a$ and $d_b$, we use our novel KG generation pipeline to generate Knowledge Graphs (Chapter 3), $kg_a$ and $kg_b$ respectively. From these KGs since we want to focus on the structural aspect and the kinds of connectivity, we will convert the entities of the kgs into *entity types* of DBpedia namespace, let us call them $kge_a$ and $kge_b$. We will define certain measurable qualities of $kge$'s that can be used to evaluate the similarity between them. The overview of Document to kge construction is shown in Figure 4.1.

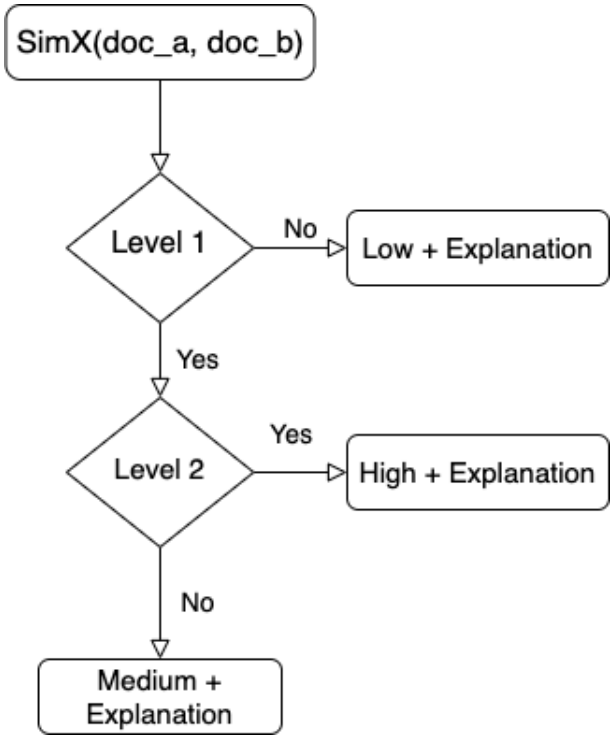


**Figure 4.2** SimX flow diagram

Lets look at the **Algorithm 1** to convert a $kg$ to $kge$. Formally, given a $kg$ as a list of $< Subject, Predicate, Object >$ triplets, like $kg$ = [$< s_1, p_1, o_1 >$, $< s_2, p_2, o_2 >$, ..], the algorithm $extractKGE(kg)$ will convert it into $kge$. Let number of triplets be, $N = len(kg)$. Let the topic of the document be defined by variable $topic$.

After we have got $kge$s for both the documents, we will use our multi-level algorithm to determine the level of similarity between the two topics, either - $Low, Medium, High$. Along with this classification, our approach will provide a explanations so as to why we have given the specific classification. See figure 4.2 for understanding the flow of our approach.

## 4.4 Level 1 - High level classifier with explanation

### 4.4.1 Theory

In this section, we will establish our Level 1 classifier based on High Level (HL) structural features of the two document kges. Once we have $kge_a$ and $kge_b$ for $d_a$ and $d_b$, we will proceed on defining measurable qualities that will help classify the similarity between two documents.

1. **Type of topic** - We can use the class types of the topics of the documents $d_a$ and $d_b$ to rule out the comparisons between two topics of different class types.

**Algorithm 1**: $extractKGE(KG)$ - Getting $kge$ triples from $kg$ triples

**Initialise:**
$topic \leftarrow$ Topic of the document
$kg \leftarrow [[s_1, p_1, o_1], [s_2, p_2, o_2], ..]$;
$N \leftarrow len(kg)$;
$kge \leftarrow kg$;
**for** $i = 0, N$ **do**
    $currentSubject \leftarrow kg[i][0]$;
    $currentObject \leftarrow kg[i][2]$;
    **if** $(topic == currentSubject)$ **then**
        $typeOfObject \leftarrow findHLType(currentObject)$ ;
        $kge_i \leftarrow [topic, "related", typeOfObject]$;

    **else if** $(topic == currentObject)$ **then**
        $typeOfObject \leftarrow findHLType(currentSubject)$;
        $kge_i \leftarrow [topic, "related", typeOfObject]$;

**endfor**
**return** $kge$;

2. **Count/Percent of connections of document's topic with entities of type *DBpedia:Person*** - The count of number of connections of the topic to other human beings will quantify the direct influence of the topic on other significant human entities. Formally, let us call this $count\_person$ of a document.

3. **Count/Percent of connections of document's topic with entities of type *DBpedia:Place*** - The count of number of connections of the topic to other human beings will quantify the direct relatedness of the topic on other significant places Formally, let us call this $count\_place$ of a document.

4. **Count/Percent of connections of document's topic with entities of type *DBpedia:Thing*** - The count of number of connections of the topic to other human beings will quantify the direct relatedness of the topic on other significant places Formally, let us call this $count\_thing$ of a document.

5. **Count of total connections of document's topic with entities of type *DBpedia:Person* or *DBpedia:Place* or *owl:Thing*** - This will help us evaluate the difference in scope of the two documents. Formally, let us call it $scope$ of a document. $scope$ will also help us normalise qualities of one documents with respect to other, so you will see $scope$ used in algorithms. Consider a case where we want to calculate similarity between Wikipedia page of Umesh Yadav (one of the bowlers of the Indian cricket team) and Wikipedia page of Sachin Tendulkar. The $scope$ aspect would help us differentiate between them amount of significance that both of these *Person type* entities have on the world.

### 4.4.2 Algorithms - Quality Extraction and Classifier

**Algorithm 2**: $extractLevel1(kge)$ - Getting Level 1 qualities from a given $kge$

**Initialise:**
$topic \leftarrow$ Topic of the document
$kge \leftarrow extractKGE(kg)$;
$N \leftarrow len(kge)$;
$type \leftarrow findType(topic)$;
$scope \leftarrow 0$;
$count\_person \leftarrow 0.0$;
$count\_place \leftarrow 0.0$;
$count\_thing \leftarrow 0.0$;
**for** $i = 0, N$ **do**
    $TypeOfObject \leftarrow kge[i][2]$;
    **if** $(DBpedia : Person == TypeOfObject)$ **then**
        $scope \leftarrow scope + 1$;
        $count\_person \leftarrow count\_person + 1$;

    **if** $(DBpedia : Place == TypeOfObject)$ **then**
        $scope \leftarrow scope + 1$;
        $count\_place \leftarrow count\_place + 1$;

    **if** $(DBpedia : Thing == TypeOfObject)$ **then**
        $scope \leftarrow scope + 1$;
        $count\_thing \leftarrow count\_thing + 1$;

**endfor**
$percent\_person \leftarrow count\_person/scope$;
$percent\_place \leftarrow count\_place/scope$;
$percent\_thing \leftarrow count\_thing/scope$;
**return** $\{$"$type$" : $type$, "$scope$" : $scope$, "$percent\_person$" : $percent\_person$, "$percent\_place$" : $percent\_place$, "$percent\_thing$" : $percent\_thing\}$;

**Algorithm 2**, shows a pseudo-snippet of implementation of our high level quality extraction. For finding entity types, we use *https://dbpedia.org/sparql* endpoint on a python script. For example, the following SPARQL query can be used to determine the type of Cristiano Ronaldo,

*SELECT DISTINCT ?type*
*WHERE* {
$< http://dbpedia.org/resource/Cristiano\_Ronaldo >$ *rdf:type ?type*
}

Once we have the individual quality values for both the documents, we will have a classifier function to classify the pair of documents based on the quality comparison. The output of this is going to be a

**Algorithm 3**: $level1Classifier(kge_a, kge_b)$ - Level 1 similarity classifier between two documents A and B

**Initialise:**
$level1\_a \leftarrow extractLevel1(kge\_a)$;
$level1\_b \leftarrow extractLevel1(kge\_b)$;
$sim\_class \leftarrow 1$;
$explanation \leftarrow []$;
$type \leftarrow []$;
$scope\_ratio \leftarrow level1\_a.scope/level1\_b.scope$;
$threshold \leftarrow 0.2$;
$topicA \leftarrow$ Topic of Document A
$topicB \leftarrow$ Topic of Document B
**if** $(scope\_ratio * threshold < 1$ *or* $scope\_ratio > threshold)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$("scope_mismatch");
    $explanation.add$('Scope of the two documents is too varied.');

**if** $(level1\_a.type \neq level1\_b.type)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$('type_mismatch');
    $explanation.add$('The documents talks about two entities of very different types.');

**if** $(level1\_b.percent\_person - level1\_a.percent\_person > threshold)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$('percent_person_b');
    $explanation.add$('The {topicB}'s interactions with other
    human entities is too high than that of {topicA}.');

**if** $(level1\_a.percent\_person - level1\_b.percent\_person > threshold)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$('percent_person_a');
    $explanation.add$('The {topicA}'s interactions with other
    human entities is too high than that of {topicB}.');

**if** $(level1\_b.percent\_place - level1\_a.percent\_place > threshold)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$('percent_place_b');
    $explanation.add$('The {topicB}'s interactions with other
     places is too high than that of {topicA}.');

*continued..*

boolean, $Yes$ or $No$. A Yes signifies that these two entities can either be highly or mediumly similar, go ahead with Level 2 Similarity measures to find the exact similarity classification. However, with a $No$, we deterministically say that that these two documents differ based on fundamental connectedness, resulting in a $Low$ similarity score. This also confirms that there's no need to go further to Level 2. When we output a No, it will result to the classfication $Low$, with appropriate reasoning. So a $Yes$ also means that the similarity classification is *to-be-decided (TBD)* but it is either going to be $Medium$ or $High$. **Algorithm 3** provides pseudo-code for implementation details of our Level 1 classifier.

---

**if** $(level1\_a.percent\_place - level1\_b.percent\_place > threshold)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$('percent_place_a');
    $explanation.add$('The {topicA}'s interactions with other
     places is too high than that of {topic B}.');

**if** $(level1\_b.percent\_thing - level1\_a.percent\_thing > threshold)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$('percent_thing_b');
    $explanation.add$('The {topic B}'s interactions with other things, ie.,
    non-person/non-place type things is too high than that of {topicA}.');

**if** $(level1\_a.percent\_thing - level1\_b.percent\_thing > threshold)$ **then**
    $sim\_class \leftarrow 0$;
    $type.add$('percent_thing_a');
    $explanation.add$('The {topicA}'s interactions with other things, i.e,
    non-person/non-place type things is too high than that of {topicB}.');

**return** $\{"sim\_class" : sim\_class, "no\_type" : no\_type, "explanation" : explanation\}$;

---

## 4.5 Level 2 - Low Level classifier with explanation

### 4.5.1 Theory

After the high-level classifier gives a $Yes$, we will use our Level 2 classifier for futher similarity classificiation. This classifier will double-down on the structural aspects and go lower to compare niche connectedness qualities. A $No$ from Level 1 classifier will not be using Level 2.

In Level 1 logic, refer Algorithm 3, we define statistical rules based on connectnedness to claim that two documents are not similar. If a pair of document does not satisfy any of those rules, we will be going deeper to further score the similarity between high level qualities.

We will determine level 2 qualities which we will use in conjunction to deeper high-level qualities to further understand the kind of similarity/dissimilarity between two documents. Following are the low level qualities:

1. **Low level Topic types** - A given entity has a number of entities. For eg: Cristiano Ronaldo, as a dbpedia resource - *http://dbpedia.org/resource/Cristiano_Ronaldo* it has multiple types - *dbo:Person, dbo:Animal*, *dbo:Athelete*, *dbo:SoccerPlayer*. As you can see, some of them, like **Athelete** and **SoccerPlayer** are more niche and provide details about the topic

2. **Low level Connection types** - All the entities that the topic is connected to (as described above) will also have multiple type classes. We will use the categorization of DBpedia, to calculate the count of not just Person, Place and Thing type entities, but all the classes of connected entities

3. **Deeper analysis of the High level Qualities** - In level 2, we will use the high level qualities 3, 4 and 5 to further highlight and score the differences and similarities.

### 4.5.2 Algorithms - Quality Extraction and Classifier

In this section, we will define the implementation of low level quality extraction that will be used by our Level 2 Classifier. **Algorithm 4** shows a pseudocode of implementation of our low level quality extraction. As mentioned earlier, we only apply level 2 Classifier for *worthy* comparisons, i.e leave out the pair of documents which are of low similarity, along with explanation using Level 1 approach.

The low level classifier focuses at determining whether the similarity between the pair of documents is $High$ or $Medium$. Using the low level qualities, we determine a similarity score and find more sophisticated explanations. let us take a closer look on $level2Classifier(kge_a, kge_b)$ algorithm, as shown in Algorithm 5.

## 4.6 Experiments and Results

This is a unique problem, and there isn't a readily available labelled dataset for us to test results against. On top of that, there really isn't a well defined way to evaluate the explanations for the similarity of documents. Hence we will do best effort work to take handful of examples to build trust on our system and discuss the results.

### 4.6.1 Wikipedia Article Evaluation

$SimX(doc_a, doc_b)$ not only classifies the similarity into $Low, Medium, High$, but also provides explanation. For our analysis, we work with 5 Wikipedia pages to understand the strengths and limitations of our approach. Refer Table 4.1 for the topics that we choose for our experiments. We choose these set of articles so that we can showcase various different scenarios and outputs that our Level 1

**Algorithm 4**: $extractLevel2(kge)$ - Level 2 qualities between two documents A and B

**Initialise:**
$level1\_results \leftarrow level1Classifier(kge\_a, kge\_b)$;
$sim\_class \leftarrow level1\_results['sim\_class']$;
$dbpedia\_types \leftarrow$ All DBpedia class types;
$connection\_types \leftarrow \{\}$;
$t\_types \leftarrow getTopicTypes(topic)$;
**for** *type in dbpedia_types* **do**
    $connection\_types[type] = 0$;
    $topic\_types[type] = 0$;
**endfor**
**if** $sim\_class == 0$ **then**
    $print$("No need for Level 2 analysis");
    $break$;

**for** *type in t_types* : **do**
    $topic\_types[type] \leftarrow topic\_types[type] + 1$;
**endfor**
**for** *triplet in kge* **do**
    $types \leftarrow getTopicTypes(triplet[2])$;
    **for** *type in types* **do**
        $connection\_types \leftarrow connection\_types + 1$;
    **endfor**
**endfor**
**return** $\{$"$topic\_types$" : $topic\_types$, "$connection\_types$" : $connection\_types\}$;

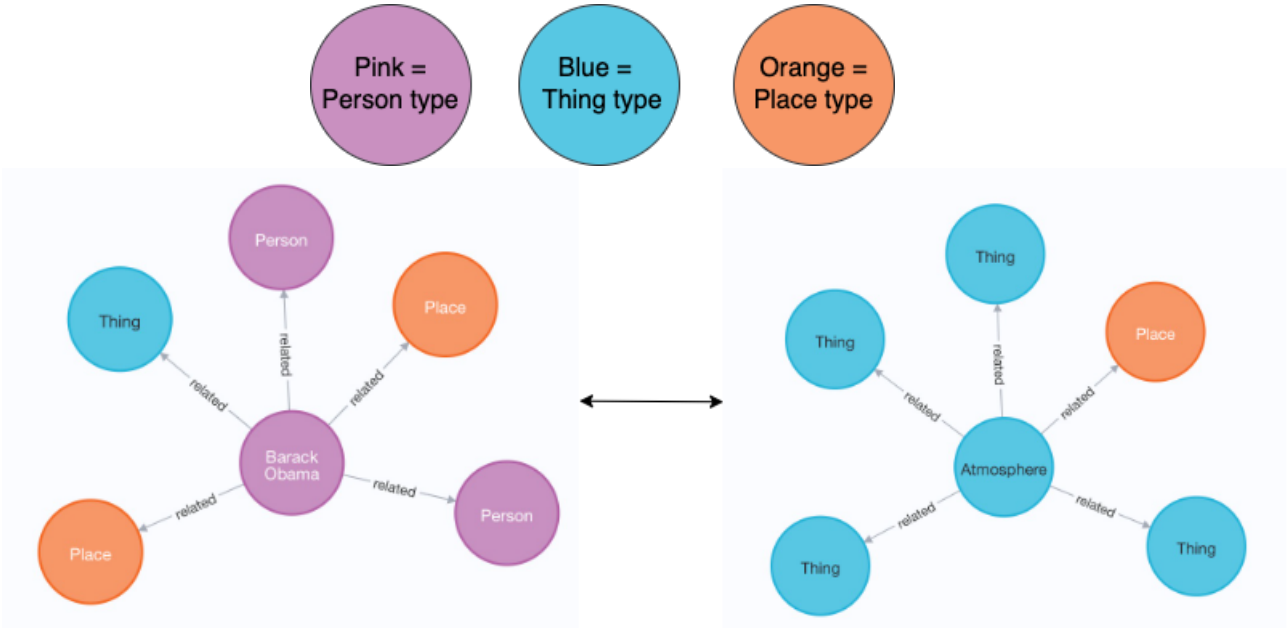


**Figure 4.3** Document similarity of Barack Obama with Atmosphere wikipedia article is LOW

**Algorithm 5**: $level2Classifier(kge_a, kge_b)$ - Determining similarity class with explanation

**Initialise:**
$level1\_a \leftarrow extractLevel1(kge\_a)$;
$level1\_b \leftarrow extractLevel1(kge\_b)$;
$topicA \leftarrow$ Topic of Document A
$topicB \leftarrow$ Topic of Document B
$level1\_result \leftarrow level1Classifier(kge\_a, kge\_b)$;
$explanation \leftarrow []$;
$ratio = level1\_result['scope\_ratio']$;
$level2\_a \leftarrow extractLevel2(kge\_a)$;
$level2\_b \leftarrow extractLevel2(kge\_b)$;
$topic\_diff \leftarrow 0.0$;
$connection\_diff \leftarrow 0.0$;
$dbpedia\_types \leftarrow$ All DBpedia class types;
$topic\_types\_a \leftarrow level2\_a["topic\_types"]$;
$topic\_types\_b \leftarrow level2\_b["topic\_types"]$;
$connection\_types\_a \leftarrow level2\_a["connection\_types"]$;
$connection\_types\_b \leftarrow level2\_b["connection\_types"]$;
$sim\_class \leftarrow 2$;
$threshold \leftarrow 3$;
$percent\_threshold \leftarrow 0.1$;
$topic\_total \leftarrow 0$;
$person\_diff \leftarrow level1\_a.percent\_person - level1\_b.percent\_person$;
$place\_diff \leftarrow level1\_a.percent\_place - level1\_b.percent\_place$;
$thing\_diff \leftarrow level1\_a.percent\_thing - level1\_b.percent\_thing$;
**if** $person\_diff > percent\_threshold$ **then**
    $sim\_class \leftarrow 1$;
    $explanation.add$('{topicA} has slightly more person type connections than {topicB}');

**if** $person\_diff < -1 * percent\_threshold$ **then**
    $sim\_class \leftarrow 1$;
    $explanation.add$('{topicB} has slightly more person type connections than {topicA}');

**if** $place\_diff > percent\_threshold$ **then**
    $sim\_class \leftarrow 1$;
    $explanation.add$('{topicA} has slightly more place type connections than {topicB}');

**if** $place\_diff < -1 * percent\_threshold$ **then**
    $sim\_class \leftarrow 1$;
    $explanation.add$('{topicB} has slightly more place type connections than {topicA}');

**if** $thing\_diff > percent\_threshold$ **then**
    $sim\_class \leftarrow 1$;
    $explanation.add$('{topicA} has slightly more thing type connections than {topicB}');

*part 2 in next page - continued..*

```
if thing_diff < −1 ∗ percent_threshold then
    sim_class ← 1;
    explanation.add('{topicB} has slightly more thing type connections than {topicA}');

for type in dbpedia_types do
    type_a = topic_types_a[type];
    type_b = topic_types_b[type];
    if type_a ≠ 0 and type_b == 0 then
        explanation.add('{topicA} is a type, whereas {topicB} is not.');
        topic_mismatch ← topic_mismatch + 1;

    if type_a == 0 and type_b ≠ 0 then
        explanation.add('{topicB} is a type, whereas {topicA} is not.');
        topic_mismatch ← topic_mismatch + 1;

    topic_total ← topic_total + 1;
endfor
topic_diff = (topic_mismatch/topic_total);
if topic_diff > 0.3 then
    sim_class ← 1;
    explanation.add('Overall: There's more than 30% topic mismatch between {topicA} and {topicB}');

part 3 in next page - continued..
```

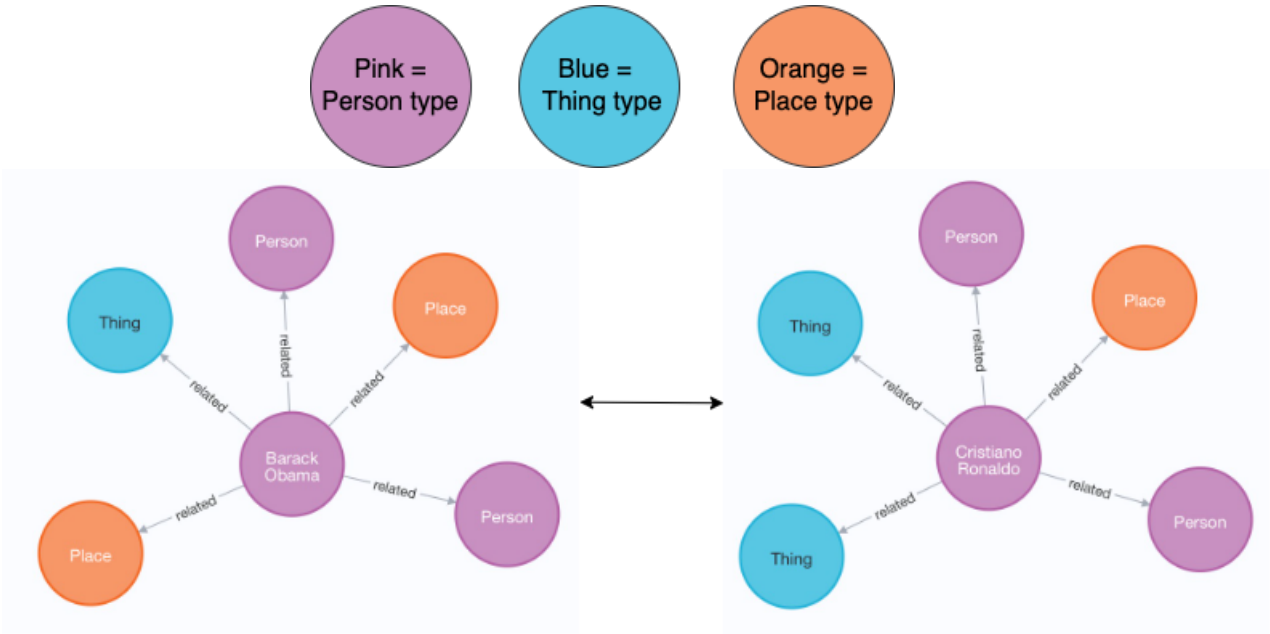


**Figure 4.4** Document similarity of Barack Obama with Cristiano Ronaldo wikipedia article is to be determined by Level 2

```
for type in dbpedia_types do
    type_a = connection_types_a[type];
    type_b = connection_types_b[type];
    type_diff_ab = (type_a/type_b) * ratio;
    if type_a == 0 or type_b == 0 then
        if type_a ≠ 0 and type_b == 0 then
            explanation.add('{topicA} has type_a connects with type type entities,
             whereas {topicB} has none.');

        if type_a == 0 and type_b‘ ≠ 0 then
            explanation.add('{topicB} has type_b connects with type type entities,
             whereas {topicA} has none.');


    else
        if type_diff_ab * threshold < 1 then
            explanation.add('{topic A} has significantly higher connection with
             type type entities than {topic B}. {topic A} has type_a connections
             whereas {topic B} has type_b.');

        if type_diff_ab > threshold then
            explanation.add('{topic B} has significantly higher connection with
             type type entities than {topic B}. {topic B} has type_b connections
             whereas {topic A} has type_a.');

    endif
    connection_diff = connection_diff + absolute(1 − type_diff);
endfor
connection_diff ← connection_diff/len(dbpedia_types);
if connection_diff > 0.1 then
    sim_class ← 1;
    explanation.add('Overall: There's more than 10% type mismatch
     between the connections of {topic A} and {topic B}');

return {"sim_class" : sim_class, "explanation" : explanation};
```

| Wikipedia | Person | Place | Thing | Total |
|---|---|---|---|---|
| Barack Obama | 278 | 60 | 143 | 481 |
| Percent→ | 0.58 | 0.12 | 0.3 | |
| Joe Biden | 270 | 42 | 150 | 462 |
| Percent→ | 0.58 | 0.09 | 0.32 | |
| Cristiano Ronaldo | 188 | 80 | 210 | 478 |
| Percent→ | 0.39 | 0.17 | 0.44 | |
| Atmosphere | 36 | 92 | 330 | 458 |
| Percent→ | 0.08 | 0.2 | 0.72 | |
| Methyl Formate | 3 | 5 | 36 | 44 |
| Percent→ | 0.07 | 0.11 | 0.82 | |

**Table 4.1** Level 1 values

classifier produces. We have taken similar sized excerpts from Wikipedia articles so that our comparison is not biased. You will see that, wikipedia document on Methyl Formate is also taken because of its small scope and size. Take a look at Figure 4.3 and 4.4 as an illustrative of connectivity comparison. Different colors represent different entity types.

#### 4.6.1.1 Level 1 Classifier Results

let us analyse our document comparison results, as shown in Table 4.2.

**Strengths**:

1. Our approach is correctly able to identify major differences between two document topics which makes their similarity classification to be $Low$. For instance, we can see that Barack Obama vs Atmosphere comparison results in Low classification. The explanation is crisp and accurate - That these two documents differ in their types, that the interaction of Obama are mostly with other human whereas Atmosphere has barely any. On contrary, the interactions of Atmosphere topic with other things is much higher than that of Obama. Hence we are able to catch all *true negatives*

2. Our approach is able to correctly able to differ wrt Scope of the documents. Atmosphere-MethylFormate both being similar entity types, are correctly identified to be different in their scopes.

3. Our approach is showcase no additional/untrue explanations. Wikipedia document of Ronaldo and Methly Formate differ in their entity types connections and scope which is exactly what our approach is able to establish as well.

| Wikipedia A vs B | Similarity | Explanation |
|---|---|---|
| Obama-Biden | Med/High | They are similar, forward to level 2 |
| Obama-Atmosphere | Low | The document talks about two entities of very different types. Obama's interactions with other human entities is too high than that of Atmosphere. Atmosphere's interactions with other things, ie., non-person/non-place type things is too high than that of Obama. |
| Obama-Ronaldo | Med/High | They are similar, forward to level 2 |
| Obama-Methyl Formate | Low | Scope of the two documents is too varied. The document talks about two entities of very different types. The Obama's interactions with other human entities is too high than that of Methyl Formate. Methyl Formate's interactions with other things, ie., non-person/non-place type things is too high than that of Obama. |
| Biden-Ronaldo | Med/High | They are similar, forward to level 2 |
| Biden-Atmosphere | Low | The document talks about two entities of very different types. Biden's interactions with other human entities is too high than that of Atmosphere. Atmosphere's interactions with other things, ie., non-person/non-place type things is too high than that of Biden. |
| Biden-Methyl Formate | Low | Scope of the two documents is too varied. The document talks about two entities of very different types. The Biden's interactions with other human entities is too high than that of Methyl Formate. Methyl Formate's interactions with other things, ie., non-person/non-place type things is too high than that of Biden. |
| Ronaldo-Atmosphere | Low | The document talks about two entities of very different types. Ronaldo's interactions with other human entities is too high than that of Atmosphere. |
| Ronaldo-Methyl Formate | Low | The document talks about two entities of very different types. Ronaldo's interactions with other human entities is too high than that of Methyl Formate. Methyl Formate's interactions with other things, ie., non-person/non-place type things is too high than that of Methyl Formate. |
| Atmosphere-MethylFormate | Low | Scope of the two documents is too varied. |

**Table 4.2** SimX: Level 1 classifier results

4. Our approach is able to correctly identify when it cannot decide if the Similarity class is either Medium or High. Our system is able to correctly conclude that Wikipedia article of Obama and Ronaldo are not significantly different, so as to label them as $Low$.

5. Our approach does not have any false negatives because of strictness of our qualitative rules.

**Limitations**:

1. Our approach lacks insights when comparing two similar documents. When we compare Wikipedia article on Joe Biden with Barack Obama, our approach is able to not conclude that they are highly related.

2. Due to the above, Level 1 system cannot deterministically differentiate between highly similar documents vs medium similarity documents. As shown in Figure 4.4 the similarity classification is $TBD : Medium$ or $High$.

#### 4.6.1.2 Level 2 Classifier Results

From table 4.2, only Obama-Biden and Obama-Ronaldo are sent to Level 2 classifier. We pick additional samples to test Level 2 classifier, take a look at Table 4.3 to check the results. Take a look at Figure 4.5 and Figure 4.6 to focus on the difference in structural connected-ness of these the two wikipedia documents - one of Barack Obama and other of Cristiano Ronaldo.

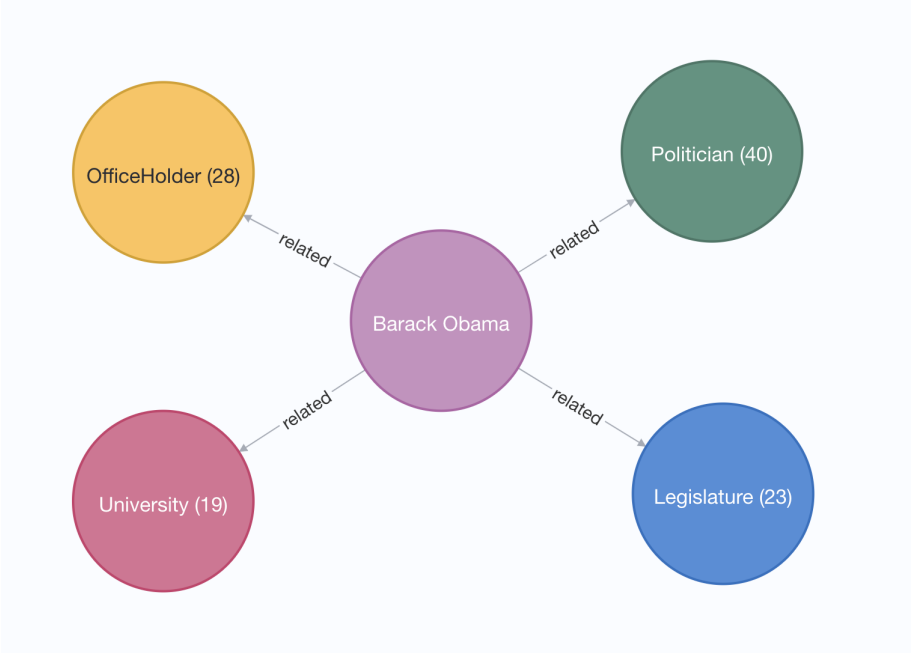


**Figure 4.5** Connected-ness of Barack Obama with different entity types. Numbers is bracket represent the count of the relationships with given entity types.

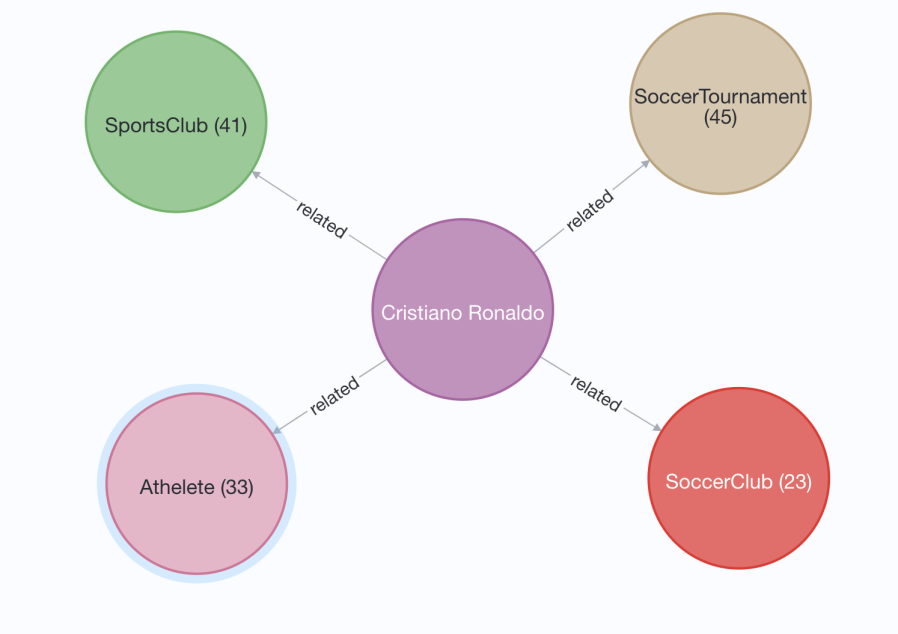


**Figure 4.6** Connected-ness of Cristiano Ronaldo with different entity types. Numbers is bracket represent the count of the relationships with given entity types.

**Strengths**:

1. Level 2 approach is correctly able to identify major differences between Barack Obama and Cristiano Ronaldo, which our Level 1 wasn't able to achieve.
2. Low level classifier is effectively able to separate Highly similar pair of documents with that of medium similarity. Wikipedia article of Barack Obama and Joe Biden are highly similar to each other, which is also the output of our approach. Hence we're able to identify True Negatives
3. Our approach is providing a suitable explanation, showing significant differences that the two documents might have based on the low-level qualities based on individual connectedness of the documents. Take a look at Obama-Ronaldo from Table 4.3.
4. Our approach is sophisticated, for example it is able to obtain difference between two atheletes who play different sports, take a look Tendulkar-Ronaldo.

**Limitations**:

1. Our approach lacks insights when comparing Joe Biden with Barack Obama and not able to spot all the minute differences.
2. Our approach can be extended to use actual node and relationship labels to get even more detailed comparison of two Documents.

| Wikipedia A vs B | Similarity | Explanation |
|---|---|---|
| Obama-Biden | High | Both Obama and Biden are Politician type entities. Both of them are connected with similar number of OfficeHolder, Politician, Legislature and University type entities. |
| Obama-Ronaldo | Medium | Obama has more person type connections than Ronaldo. Ronaldo has slightly more thing type connections than Obama. Ronaldo is a SoccerPlayer and Athelete whereas Obama is not. Obama is a Politician and OfficeHolder whereas Ronaldo is not. Obama has higher connections with Organization, Legislature, PoliticalParty and University type entities. Ronaldo has higher connections with SportsClub, SoccerClub, SoccerTournament, SoccerPlayer type entities. |
| Tendulkar-Ronaldo | Medium | Ronaldo is a SoccerPlayer whereas Tendulkar is not. Tendulkar is a Cricketer whereas Ronaldo is not. Tendulkar has higher connections with CricketTeam and Cricketer type entities. Ronaldo has higher connections with SoccerClub, SoccerTournament, SoccerPlayer type entities. |
| Tendulkar-Kohli | High | Both Tendulkar and Kohli are Athelete and Cricketer type entities. Both of them are connected with similar number of CricketTeam, Cricketer and Organization type entities. |
| Messi-Ronaldo | High | Both Messi and Ronaldo are Athelete and SoccerPlayer type entities. Both of them are connected with similar number of SoccerTournament, SoccerPlayer, SportsClub and SoccerClub type entities. |

**Table 4.3** SimX: Level 2 classifier results

3. Our approach might have some false positives, wherein the structural aspects of two entities are very similar, but the documents are in reality different. But, based on our experiments, we find the quantity of false positives to be as low as $6\%$, even with noise introduction, as shown in section 4.6.3.3.

### 4.6.2 Miscategorized DBpedia entities

DBpedia is one of the largest hubs of Linked data on web. Given the large scope of knowledge that DBpedia holds, it is not surprising that it contains many different kinds of errors. In this experiment, we focus on error in categorization of entities. Our experiment setup is as follows:

- We collect a set $S$ of DBPedia entities which are marked as $dbo : Cricketer$. That is, all the people in this set are cricketers or were cricketers in the past.

- We choose a baseline cricketer to be Sachin Tendulkar. We use our approach to compare a Wikipedia document on each entity in $S$ with the Wikipedia document of Sachin Tendulkar.

- If our classifier computes their similarity class to be $Low$, we claim that our chosen entity is wrongly categorized.

| Wikipedia | Difference Score | SimX miscategorized? | Actually miscategorized? |
|---|---|---|---|
| A-001 | 0.37 | Yes | Yes |
| A-002 | 0.3 | Yes | Yes |
| A-003 | 0.31 | Yes | Yes |
| A-004 | 0.43 | Yes | Yes |
| M. Amir | 0.07 | No | No |
| AB De Villiers | 0.05 | No | No |
| ACTH | 0.4 | Yes | Yes |
| Bracken School Readiness Assessment | 0.37 | Yes | Yes |

**Table 4.4** Outlier Detection of $dbo : Cricketer$ entities

| Wikipedia | Diff Score | | | SimX Outlier? | Actual Outlier? |
|---|---|---|---|---|---|
| | London | Bombay | Sydney | | |
| James W. Everington | 0.29 | 0.23 | 0.3 | Yes | Yes |
| Aurangabad | 0.11 | 0.1 | 0.09 | No | No |
| Mobile, Alabama | 0.14 | 0.12 | 0.13 | No | No |
| Nelle Peters | 0.33 | 0.26 | 0.27 | Yes | Yes |
| Delhi | 0.09 | 0.04 | 0.08 | No | No |
| Wardell Milan | 0.41 | 0.35 | 0.41 | Yes | Yes |
| Blaise Siwula | 0.33 | 0.35 | 0.39 | Yes | Yes |
| Dick Nourse | 0.37 | 0.36 | 0.39 | Yes | Yes |
| Slough | 0.08 | 0.10 | 0.09 | No | No |
| Peter Calandra | 0.4 | 0.41 | 0.29 | Yes | Yes |

**Table 4.5** Outlier Detection of $dbo : City$ entities

- We repeat this for $dbo : City$ category, where we take three different baselines - Wikipedia page of London, Bombay and Sydney.

We calculate the difference between the connectned-ness of baseline vs the entity. Our score will be between 0 and 1. All the difference scores greater than $0.2$ will be determined as Outlier by our approach SimX.

We run category outlier detection for 100 Cricketer type entities and find that 13 of them are outliers in reality and all the 13 are recognized accurately by our approach. Some results are shown in 4.4.

Similarly, for City type entities (see table 4.5), we find that 11 out of 100 entities are real outliers and again all of them are deterministically classified as outliers by SimX. We also take a special (ambiguous city) known as Mobile, which is a city in Alabama and find that Mobile, Alabama is also accurately determined to be a city and not an outlier. By taking multiple baselines for $dbo : City$, we also confirm that *any* (true) city can be selected as a baseline for outlier detection.

Overall, based on our analysis, we neither spot any false negatives nor any false positives, i.e., cases when there's a mismatch between SimX's output vs actual outlier or not. This is due to two reasons:

1. One, the stanford NER we use has an F-1 score of $> 92\%$ for English language, hence we get all the named Entities accurately

2. Second, the type of these entities is determined by DBpedia using a SPARQL query (as shown in section 4.4.2) which has no cause for errors.

Hence, owing to the high accuracy of individual components, and setting meaningful thresholds we do not observe any false negatives or false positives.

### 4.6.3 Same Topic Evaluation with noise

#### 4.6.3.1 Part 1

Our KG-based document similarity should be able to highly score similarity between exactly same document. The aim of this section is to evaluate the robustness of our approach against a pair of same documents. We took 100 wikipedia pages, and our algorithm is consistently able to score all the pairs $(doc_a, doc_a)$ with "$High$" score. Not only that, there were no contradicting explanations which confirmed that our similarity evaluation algorithms are robust.

#### 4.6.3.2 Part 2 - Paraphrasing

To make this more challenging for our system, we use paraphrasing tool to generate two not-exactly same, but similar documents. The procedure is as follows. We take a wikipedia article $A$ on topic $T$, and paraphrase it using online tools to $B$. Ideally, the similarly score of $(A, B)$ should be $High$, since these two documents have nothing but same content.

We use an online paraphrasing tool called spinbot[1] to paraphrase an article. We collect $10$ such $A$'s and build their corresponding $B$'s separately for two different categories of Wikipedia Person and Place. Some examples of Person topic are Barack Obama, Joe Biden, etc and Place topic are London, Mumbai, etc. These documents are all large with $50+$ sentences in each so that we make sure that we allow the algorithm to make some mistakes. We define accuracy for this experiment in the following manner,

$$Accuracy \quad = \quad \frac{count(SimX(A, B).class == High)}{Total}$$

Our Approach had a brilliant accuracy of $98\%$, meaning that almost all the paraphrased documents $(A')$ were evaluated to be highly similar to their original counterparts. This also strengthen's that our fundamentals are based on structural connected-ness of the KGs, which are robust to paraphrasing.

[1] www.spinbot.com

#### 4.6.3.3 Part 3 - Paraphrasing with noise

To make the experiment setup we check performance of SimX against noise. The setup is as follows - we take 20 sentences from a wikipedia topic $A$ and 10 sentences from a random document $N$ and combine them to make a document $B$. The motive behind adding 10 sentences of $N$ is to introduce *noise* to $A$. We then paraphrase $B$ to become $A'$. Now we use our approach to test whether similarity between $A$ and $B$ is still high or not. And that our approach does not show high similarity between $B$ and $N$, because $N$ is just acting as noise here. So we count two kinds of accuracies as,

- First for evaluating that we maintain high simialrity score between $A$ and $A'$ even after introduction of noise.

$$Accuracy_1 = \frac{count(SimX(A, A').class == High)}{Total}$$

- Second, that our approach doesn't output high similarity score between $A'$ and the noise $N$.

$$Accuracy_2 = \frac{count(SimX(N, A').class \neq High)}{Total}$$

We choose 10 different candidates for $A$ against 5 different versions of noise $N$, so we evaluate a total of 50 noisy documents paraphrased against original documents and the noise. We get $Accuracy_1$ of $88\%$, showcasing that we're consistently able to reduce the affect of noise in most cases. Of the remaining $12\%$ cases, we observe that we get a $Medium$ similarity, confirming that the effect of noise is non-zero. We get an even better $Accuracy_2$ of $94\%$, meaning that only 3 out of 50 times we get a $High$ similarity score between $N$ and $A'$.

## 4.7 Summary

- Our level 1 (super-fast) classifier is effectively and efficiently able to rule out all pairs of document comparisons with $Low$ scores along with crisp explanation, as shown in Table 4.2.
- As shown in Table 4.3, our level 2 classifier is accurately able to determine $Medium$ or $High$ classification, along with appropriate reasoning based on deeper structural aspects of the document KG.
- Our approach can be used for Category Outlier detections in DBpedia with excellent accuracy, refer Section 4.6.3.
- Our approach is able to effectively reduce the importance of noise or paraphrasing or both, giving strong results as shown in Section 4.6.3.
- Similarity can take different meanings in different contexts, which makes this problem space twisted. By providing explanations along with document similarity classification, we not only

explain reasoning behind our classification, but also lower down the ambiguity around the *"similarity"* meaning.

- Our approach is tunable to different settings depending on the requirements. We have a set of thresholds defined which are used to do the classification and explanation. With different applications, if one wishes the classifier to be more strict or lenient, we could tune the thresholds to adapt to the requirements.
- Our approach focuses on unique set of structural aspects of documents, which makes them usable alongside any other text similarity approaches, as discussed in Related Work.

*Chapter 5*

# Conclusion

Automatic extraction of information from text and its transformation into a structured format is an important goal in both Semantic Web Research and computational linguistics. Knowledge Graphs (KG) serve as an intuitive way to provide structure to unstructured text. Simple rule-based approaches cannot generate rules efficiently, especially when text sources are sparse. This is due to the fact that these rules are manually generated and hence it cannot get rid of all redundancies. On the other hand, similarity-based solutions for Predicate mapping are challenging in two aspects:

1. To map a predicate to another namespace, they need to capture the accurate semantics for calculating similarity scores

2. Comparing a predicate to each of the predicate in DBpedia is highly time consuming owing to the number of candidates (i.e. the number predicates in DBPedia namespace to which a particular text predicate can map to

In Chapter 3, we present our novel end-to-end KG construction system, where we improve the Predicate Mapping computationally, by introducing an effective pruning strategy. Low redundancy of our approach enhances the KG's searchability in applications such as question answering and information retrieval. We evaluate to see that our approach generates a Knowlege-graph, which has more number of correct triplets, less number of misleading or incorrect triplets and has lower redundant triplets as compared to other approaches.

In chapter 4, we introduce SimX: a novel multi-level knowledge-graph based document similarity classifier by using chapter 3's system to generate KGs from the documents. SimX has a unique ability to spot structural similarities and differences in the documents and use that to provide an explanation along with a $Low/Medium/High$ classification. Since our approach uses non-conflicting and unique set of statistical qualities, it can be used along with any of the existing text similarity solutions to enhance the results. Our level 1 fast classifier accurately spots articles which have $Low$ score along with providing crisp explanations. Level 2 classifier uses low level qualities to differentiate highly similar documents with medium level of similarity consistently. We observe that our approach is robust to noise and paraphrasing. Our approach also spots category outliers in DBpedia with high accuracy.

In the future, we continue to explore the space of problems where knowledge graph can be useful. We plan to work on constructing domain specific Knowledge Graphs for Health, History and Science. We will also explore Automatic Question Answering using these domain specific knowledge graphs. With respect to document similarity, we focussed on structural aspects of knowledge graphs. In order to enhance SimX, we plan to use node and edge labels to determine more sophisticated pieces of similarities and differences between documents.

# Related Publications

- **Aman Mehta**, Aashay Singhal, and Kamalakar Karlapalem. 2019. **Scalable Knowledge Graph Construction over Text using Deep Learning based Predicate Mapping**. In Companion Proceedings of The 2019 World Wide Web Conference (WWW '19).

# Bibliography


[1] A. Andoni, P. Indyk, and R. Krauthgamer. Earth mover distance over high-dimensional spaces. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, page 343–352, USA, 2008. Society for Industrial and Applied Mathematics.

[2] G. Angeli, M. J. J. Premkumar, and C. D. Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 344–354, 2015.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[4] I. Augenstein, S. Padó, and S. Rudolph. Lodifier: Generating linked data from unstructured text. In *Extended Semantic Web Conference*, pages 210–224. Springer, 2012.

[5] A. Barrón-Cedeño, P. Rosso, E. Agirre, and G. Labaka. Plagiarism detection across distant language pairs. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 37–45, Beijing, China, Aug. 2010. Coling 2010 Organizing Committee.

[6] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, page 143–154, New York, NY, USA, 2005. Association for Computing Machinery.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[8] P. A. Bonatti and S. Kirrane. Big Data and Analytics in the Age of the GDPR. pages 7–16.

[9] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[10] D. Calvanese, W. Fischl, R. Pichler, E. Sallinger, and M. Simkus. Capturing relational schemas and functional dependencies in rdfs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), Jun. 2014.

[11] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3. Atlanta, 2010.

[12] R. Cattoni, F. Corcoglioniti, C. Girardi, B. Magnini, L. Serafini, and R. Zanoli. The knowledgestore: an entity-based storage system. In *LREC*, pages 3639–3646. Citeseer, 2012.

[13] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 445–456, New York, NY, USA, 2014. Association for Computing Machinery.

[14] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 2201–2206, New York, NY, USA, 2016. Association for Computing Machinery.

[15] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 458–469, 2013.

[16] R. L. Cilibrasi and P. M. Vitanyi. The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.

[17] a. Corts-Calabuig and J. Paredaens. Semantics of constraints in rdfs. *CEUR Workshop Proceedings*, 866:75–90, 01 2012.

[18] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, mar 1964.

[19] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, pages 391–407, 1990.

[20] L. Del Corro and R. Gemulla. Clausie: clause-based open information extraction. In *Proceedings of the 22nd international conference on World Wide Web*, pages 355–366. ACM, 2013.

[21] D. Devarajan. Happy Birthday Watson Discovery. IBM Cloud Blog, December 21 2017.

[22] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009.

[23] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, July 1945.

[24] L. Ehrlinger and W. Wöß. Towards a Definition of Knowledge Graphs.

[25] A. A. El-atey and S. El-etriby. Semantic data extraction from infobox wikipedia template. *Coordinates*, 30:261, 1911.

[26] P. Exner and P. Nugues. Entity extraction: From unstructured text to dbpedia rdf triples. In *The Web of Linked Entities Workshop (WoLE 2012)*, pages 58–69. CEUR, 2012.

[27] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1535–1545. Association for Computational Linguistics, 2011.

[28] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference of Empirical Methods in Natural Language Processing (EMNLP '11)*, Edinburgh, Scotland, UK, July 27-31 2011.

[29] W. Fan and P. Lu. Dependencies for graphs. *ACM Trans. Database Syst.*, 44(2), feb 2019.

[30] W. Fan, P. Lu, C. Tian, and J. Zhou. Deducing certain fixes to graphs. *Proc. VLDB Endow.*, 12(7):752–765, mar 2019.

[31] W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 1843–1857, New York, NY, USA, 2016. Association for Computing Machinery.

[32] J. Fan et al. Prismatic: Inducing knowledge from a large scale lexicalized relation resource. In *Proceedings of the NAACL HLT 2010*, pages 122–127. ACL, 2010.

[33] J. R. Finkel et al. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. ACL, 2005.

[34] W. Gomaa and A. Fahmy. A survey of text similarity approaches. *international journal of Computer Applications*, 68, 04 2013.

[35] D. A. Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics*. The Kluwer International Series of Information Retrieval. Springer, second edition, 2004.

[36] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *null*, pages 1735–1742. IEEE, 2006.

[37] Q. He, B.-C. Chen, and D. Agarwal. Building The LinkedIn Knowledge Graph. LinkedIn Blog, October 6 2016.

[38] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[39] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum. YAGO2: Exploring and Querying World Knowledge in Time, Space, Context, and Many Languages. pages 229–232.

[41] T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, page 289–296, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[42] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J. E. Labra Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, and A. Zimmermann. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool, 2021.

[43] G. Huang, C. Guo, M. J. Kusner, Y. Sun, F. Sha, and K. Q. Weinberger. Supervised word mover's distance. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *NIPS*, pages 4862–4870, 2016.

[44] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Commun. ACM*, 20(5):350–353, may 1977.

[45] C. S. Iliopoulos and M. S. Rahman. New efficient algorithms for the lcs and constrained lcs problems. *Information Processing Letters*, 106(1):13–18, 2008.

[46] R. W. Irving and C. B. Fraser. Two algorithms for the longest common subsequence of three (or more) strings. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching*, pages 214–229, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[47] P. Jaccard. The distribution of the flora in the alpine zone. *The New Phytologist*, 11(2):37–50, 1912.

[48] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[49] N. Jiang and M.-C. de Marneffe. Do you know that florence is packed with visitors? evaluating state-of-the-art models of speaker commitment. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4208–4213, Florence, Italy, July 2019. Association for Computational Linguistics.

[50] T. Johnson and T. Dasu. Data quality and data cleaning: An overview. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, page 681, New York, NY, USA, 2003. Association for Computing Machinery.

[51] D. Jurgens, M. T. Pilehvar, and R. Navigli. Semeval-2014 task 3: Cross-level semantic similarity. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 17–26, 2014.

[52] M. Kejriwal, C. A. Knoblock, and P. Szekely, editors. *Knowledge Graphs: Fundamentals, Techniques, and Applications*. The MIT Press, 2021.

[53] N. Kertkeidkachorn and R. Ichise. T2kg: An end-to-end system for creating knowledge graph from unstructured text. In *Proceedings of AAAI Workshop on Knowledge-based Techniques for Problem Solving and Reasoning*, pages 743–749, 2017.

[54] A. Krishnan. Making search easier: How Amazon's Product Graph is helping customers find products more easily. Amazon Blog, August 17 2018.

[55] V. Kríž, B. Hladká, M. Nečaskỳ, and T. Knap. Data extraction using nlp techniques and its transformation to linked data. In *Mexican International Conference on Artificial Intelligence*, pages 113–124. Springer, 2014.

[56] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 957–966. JMLR.org, 2015.

[57] T. K. Landauer and S. T. Dumais. A solution to Plato's problem: The latent semantic analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240, 1997.

[58] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. 6(2):167–195, 2015.

[59] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[60] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[61] H. Li and J. Xu. Semantic matching in search. *Found. Trends Inf. Retr.*, 7(5):343–469, jun 2014.

[62] Y. Li. Research and analysis of semantic search technology based on knowledge graph. In *Computational Science and Engineering (CSE) and Embedded and Ubiquitous Computing (EUC), 2017 IEEE International Conference on*, volume 1, pages 887–890. IEEE, 2017.

[63] D. Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[64] Z. Liu, C. Xiong, M. Sun, and Z. Liu. Entity-duet neural ranking: Understanding the role of knowledge graph semantics in neural information retrieval. *arXiv preprint arXiv:1805.07591*, 2018.

[65] P. C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[66] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[67] I. Matveeva, G.-A. Levow, A. Farahat, and C. Royer. Term representation with generalized latent semantic analysis. *Recent Advances in Natural Language Processing IV: Selected Papers from RANLP 2005*, 292, 01 2007.

[68] Mausam, M. Schmitz, R. Bart, S. Soderland, and O. Etzioni. Open language learning for information extraction. In *Proceedings of Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CONLL)*, 2012.

[69] R. A. Melter. Some characterizations of city block distance. *Pattern Recognition Letters*, 6(4):235–240, 1987.

[70] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, pages 1–8. ACM, 2011.

[71] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[72] J. Mueller and A. Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *AAAI*, volume 16, pages 2786–2792, 2016.

[73] S. Narayan and C. Gardent. Hybrid simplification using deep semantics and machine translation. In *Proceedings of the 52nd Annual Meeting of the ACL (Volume 1: Long Papers)*, volume 1, pages 435–445, 2014.

[74] P. Neculoiu, M. Versteegh, and M. Rotaru. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157, 2016.

[75] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[76] F. Nielsen. A family of statistical symmetric divergences based on jensen's inequality, 2010.

[77] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. *VLDS*, 12:25–28, 2012.

[78] R. Padhye, D. Mukherjee, and V. S. Sinha. Api as a social glue. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 516–519. ACM, 2014.

[79] J. Z. Pan, G. Vetere, J. M. Gómez-Pérez, and H. Wu, editors. *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. Springer, 2017.

[80] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[81] H. Pham, T. Luong, and C. Manning. Learning distributed representations for multilingual text sequences. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 88–94, Denver, Colorado, June 2015. Association for Computational Linguistics.

[82] R. J. Pittman, A. Srivastava, S. Hewavitharana, A. Kale, and S. Mansour. Cracking the Code on Conversational Commerce. eBay Blog, April 6 2017.

[83] E. Prud, A. Seaborne, et al. Sparql query language for rdf. 2006.

[84] J. Pujara, H. Miao, L. Getoor, and W. W. Cohen. Knowledge Graph Identification. pages 542–557.

[85] G. Qi, H. Chen, K. Liu, H. Wang, Q. Ji, and T. Wu. *Knowledge Graph*. Database Management & Information Retrieval. Springer, 2021.

[86] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning. A multi-pass sieve for coreference resolution. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2010.

[87] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23:2000, 2000.

[88] G. Rizzo and R. Troncy. Nerd: a framework for unifying named entity recognition and disambiguation extraction tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the ACL*, pages 73–76. ACL, 2012.

[89] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, page 232–241, Berlin, Heidelberg, 1994. Springer-Verlag.

[90] X. Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[91] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[92] M. Schmitz et al. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534. ACL, 2012.

[93] E. W. Schneider. Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis. Technical Report HumRRO-PP-10-73, Human Resources Research Organization, Alexandria, VA, 300, North Washington Street Alexandria, Virginia 22314, November 1973. Paper presented at the Association for the Development of Instructional Systems (Chicago, Illinois, April 1972).

[94] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models, 2015.

[95] S. Shrivastava. Bring rich knowledge of people, places, things and local businesses to your apps. Bing Blogs, July 12 2017.

[96] A. Singhal. Introducing the Knowledge Graph: things, not strings. Google Blog, May 16 2012.

[97] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[98] Y.-T. Tsai. The constrained longest common subsequence problem. *Information Processing Letters*, 88(4):173–176, 2003.

[99] P. D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In L. De Raedt and P. Flach, editors, *Machine Learning: ECML 2001*, pages 491–502, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[100] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *2014 IEEE 30th International Conference on Data Engineering*, pages 244–255, 2014.

[101] D. Vrandečić and M. Krötzsch. Wikidata: A Free Collaborative Knowledgebase. 57(10):78–85, 2014.

[102] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao. Learning deep transformer models for machine translation, 2019.

[103] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. 29(12):2724–2743, December 2017.

[104] Z. Wang, T. Chen, J. Ren, W. Yu, H. Cheng, and L. Lin. Deep reasoning with knowledge graph for social relationship understanding. *arXiv preprint arXiv:1807.00504*, 2018.

[105] A. S. White and Reisinger. Universal Decompositional Semantics on Universal Dependencies. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1713–1723, Austin, Texas, November 2016. Association for Computational Linguistics.

[106] W. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. *Proceedings of the Section on Survey Research Methods*, 01 1990.

[107] L. Wu, I. E. H. Yen, K. Xu, F. Xu, A. Balakrishnan, P.-Y. Chen, P. Ravikumar, and M. J. Witbrock. Word mover's embedding: From word2vec to document embedding, 2018.

[108] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. 2011.

[109] Y. Yu and J. Heflin. Extending functional dependency to detect abnormal data in rdf graphs. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, and E. Blomqvist, editors, *The Semantic Web – ISWC 2011*, pages 794–809, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[110] Z. Zhu, D. Bernhard, and I. Gurevych. A monolingual tree-based translation model for sentence simplification. In *Proceedings of the 23rd international conference on computational linguistics*, pages 1353–1361. Association for Computational Linguistics, 2010.