UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science

Master's Thesis
in Partial Fullfilment of the Requirements for the Degree of
Master of Science

Relation Extraction using Pattern Generation and Semantic Embeddings

by
Danish Ahmed
Matriculation No.: 6826347

submitted to:
Prof. Dr. Axel-Cyrille Ngonga Ngomo
Dr. Ricardo Usbeck

Thesis Supervisor:
Dr. Ricardo Usbeck

Paderborn, April 30, 2019

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Ort, Datum	Unterschrift

Abstract. RAPID - (Relation extrAction using Pattern generation and semantIc embeDding) tries to bridge the gap between DBpedia (structured data) and Wikipedia by extracting relations from any given natural language context. RAPID follows the approach of pattern generation by creating semantic subgraphs between entities using Stanford CoreNLP. Each of the generated patterns is semantically compared with existing patterns using our Forskipo-Semantic Similarity (FSS) algorithm. Also, the word features present in the generated pattern are classified semantically against our Property Embedding Model (PEM). A PEM is word embedding representation of properties, generated using property labels and WordNet vocabulary expansion. The combined use of PEM and FSS provides a state of the art approach towards relation extraction by avoiding semantic drift at its best. Along with the extended pattern representation used for semantic analysis, we have a generalized pattern representation which contributes to our confidence value by means of support and specificity pattern-features. Our approach makes RAPID highly scalable by allowing to extract relations independent of a corpus, and can also work with less pre-generated patterns. To determine property against a pattern, we used unsupervised learning by varying α and β parameters that affect patterns and semantic embedding factors respectively. A maximum f-measure of 0.5698 was achieved using an empirical threshold. The results obtained show property dependency on specific parameters; allowing properties to perform best under certain parameter combination.

Contents

List of Figures

List of Tables

1 Introduction

In most cases, meaningful information usually exist whenever a sentence has at least two entities in it [RYM10]. The information is only useful/meaningful if these entities are related to each other. The relation between entities is the *property* that holds between them. RAPID (Relation extrAction using Pattern generation and semantIc embeDding) aims to extract meaningful data from a raw text by identifying the properties that could possibly exist between two entities. In this chapter, we will discuss why is it necessary to address this problem and state a formal definition of RAPID.

Later in chapter 2, we list a few existing systems that serve the purpose of relation extraction, along with their drawbacks. Chapter 3 overviews through all the necessary prerequisites, and the tools that were used for developing RAPID. RAPID is an undocked system having several components that work together to identify relations from a given sentence. Therefore, it is necessary to understand the architecture4 of our system that reflects the linking and flow within the components. All the essential algorithms and internal working of components are described in chapter 5. Finally, we discuss our evaluation results 7 performed under specific environment 6, and discuss the limitations 8 of RAPID along with providing future possibilities 9. We conclude 10 that RAPID provides a state-of-art approach by using a combination of Froskipo-Semantic Similarity, and Property Embedding Model that helps to avoid semantic drift and makes RAPID work with any corpus.

1.1 Motivation

Wikipedia is a hub of collaboratively edited knowledge articles [VKV+06] that reflect resource entities. Almost every article has an infobox which helps a user to filter meaningful information without reading the entire article. An infobox is a machine-harvestable object-attribute-value triple collected mostly through human efforts [WW08]. DBpedia triples reflect these infoboxes [LIJ+15] where a subject resource is the article's title, having another entity/concrete value connected to it via a predicate that describes the relationship between them. The following figure 1.1 shows a sub-section of an infobox taken from Wikipedia's Albert Einstein article [1]:

[1] `https://en.wikipedia.org/wiki/Albert_Einstein`

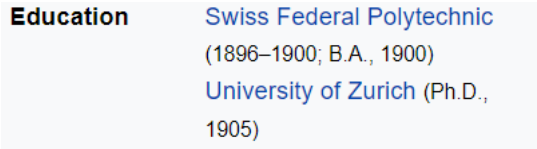

Figure 1.1: Education subsection of infobox for Albert Einstein

The infobox above suggests that Albert Einstein studied only at the above-specified institutes. But, the following sub-statement from Wikipedia article validates the existence of missing triple:

Sentence:	*Albert attended a Catholic elementary school in Munich.*
Triple:	dbr:Albert_Einstein dbo:education dbr:Catholic_school

Having a system or a framework that is able to extract relations from natural language documents can help to extend existing knowledgebases in a structured form. Also, a new knowledgebase can be generated independent of corpus type and crawled across the web to transform data into a machine readable format. It also makes a lot easier for users to analyze a document without proofreading it. Many approaches have been tried in the past to extract relations, but all the approaches are either not scalable, introduces semantic drift, or works on retrieving phrases used against relations. Our RAPID system is highly scalable and flexible that also avoids semantic drift.

1.2 Problem Statement

Given a natural language input, our aim is to identify relations that hold true between entities and to disambiguate between relations. Let D be a set of input sentence(s) se, and E is the set having entities e present in sentence(s). Then our aim is to find all the relations that can exist between entity combinations.

$$RAPID(D) = \bigcup_{i=1}^{T} (e_x, p, e_y) : e_x \neq e_y \mid D = \{se_1, se_2, ..., se_n\} \wedge E = \{e_1, e_2, ..., e_i\} \tag{1.1}$$

Where e_x is the subject reflecting predicate p by existence of object e_y. It is possible that a single entity can possess multiple properties. The total number of relations extracted by RAPID is denoted by T.

2 Related Work

A good idea for relation extraction is to harvest words and phrases that reflect a property. Bootstrapping Linked Data(BOA) [GN11], and PATTY [NWS12] are such examples. In the past few years, the most focus lies on extracting such phrases or extending the existing dataset like RELLY [GWP+15]. On the other hand, recent research like Ocelot [SN18] relies on dependency parse tree and uses tree generalization technique. While Heist and Paulheim introduced a language-agnostic approach [HP17] that is independent of language features.

2.1 Bootstrapping Linked Data (BOA)

BOA first transforms XML data of WikiDump to UTF-8 encoded text [Yoi12], and applies further filtration to get desired garbage-free context [BHQR07]. BOA patterns are the natural language patterns extracted against a property. For a given predicate p, The framework uses the triples from DBpedia knowledgebase holding property p, and use pattern search approach for collecting candidate phrases that could possibly reflect the property. Consider a triple $X = (s, p, o)$ where s is the subject connected to object o via property p, then knowledge acquired are the statements that have presence of s and o along valid *rdf:type C*. BOA formulates a pattern based on the knowledge possessed by replacing the resource entities of the subject and object by *?D?* and *?R?* respectively. While the sub-knowledge expressing predicate remains in natural language and becomes part of the BOA pattern. Hence, BOA has a set of natural language representations against every predicate p; and every pattern is a pair $P = (\mu(p), \theta)$ where $\mu(p)$ is the URI of predicate p, and θ is the natural language representation. Following example shows the pattern search approach:

Triple:	*dbr:Albert_Einstein dbo:birthPlace dbr:Germany*
Phrase:	*Albert Einstein was born in Germany.*
BOA Pattern:	*($\mu(p)$, ?D?was born in ?R?)*

BOA then makes use of language independent features to scores a pattern using support, typicity, and specificity. Support gives priority if the same pattern occurs several times with respect to a property. And specificity helps to remove patterns that are highly used in other properties too. While high typicity ensures the mapping restriction of a property with respect to its domain and range.

2.1.1 Limitations

Since BOA entirely depends on exact phrase matching, therefore it can never handle words that are semantically similar but are not present in their vocabulary dataset for a property p.

In many cases, subject s and object o occurs before the *act* that expresses a property p. In such scenarios, either the phrase between entities are meaningless with respect to the property or are empty phrases. Consider subject $s = Bruce$ and object $o = Jane$; the sentence *"Bruce and Jane got married."* reflects property $dbo : spouce$ that can be identified using Dependency Parse Tree as *married* is the root node. But the phrase between entities is *and* which would lead BOA to generate pattern:

BOA Pattern:	$(\mu(p), ?D?and\ ?R?)$

Moreover, if word *awarded* persist in vocabulary of property p but input sentence contains word *award*, BOA will not be able to categorize it under p.

2.2 Inferring Hypernym Relationships Between Relational Phrases (RELLY)

RELLY aims to harvest phrases expressing relations by means of probabilistic soft logic (PSL) [BBHG15]. It formulates hypernymy graphs based on phrases generalization by using PSL models combining semantic and statistical signals. The semantical rules utilize existing vocabulary datasets of WordNet [Mil98] and YAGO [SKW07] hyponyms, while RELLY uses closed relation extraction [WLX+04] libraries/frameworks of PATTY and HARPY [GW14] as statistical features via *argument overlap*. An example of RELLY's hypernymy generalization is mentioned below:

$\{< person >$ moves to $< country >\} \leftarrow hypernym- \{< musician >$ emigrates to $< country >\}$
$\{< person >$ created a $< artifact >\} \leftarrow hypernym- \{< person >$ wrote a poem $< artifact >\}$

The author formulated rules that are processed by PSL predicates being either statistical, semantic, or output type. Out of nine rules, R1 and R2 uses *pattySubsumption* and *weedsInclusion* PSL predicates for determining if rules are synonyms or hypernyms. R3 is responsible for WordNet verb senses and relational phrases *alignment*. Rule (4-6) explicitly serves the purpose of *type compatibility* by enabling *hypernymy restriction* and *transitive closures*. These compatibility rules ensures useage of indirect hyponyms. R7 (asymmetry), R8 (transitivity), and R9 (acyclicity) confirms retaining *structure* of output graph.

2.2.1 Limitations

For capturing semantics, author entirely relies on existing framework/libraries of WordNet synset and Yago hyponyms that has limited vocabulary to reflect a relation. Using embeddings would be more beneficial since they are vector based, and can overcome vocabulary limitation if embedding model is trained on a large corpus. Our property embedding model (PEM) outperforms; as the vectors in PEM represents combined semantics of WordNet synset, hyponyms, and pre-trained source embedding model. Moreover, RAPID's forskipo-semantic similarity (FSS) algorithm ensures semantic matching with existing pre-generated patterns (that lacks in RELLY).

2.3 On Extracting Relations using Distributional Semantics and a Tree Generalization (OCELOT)

Ocelot closely matches our approach as it makes use of both dependency parse tree and distributional semantics. But the utilization and algorithm development strategy completely differs from RAPID. Ocelot emphasis on tree generalization using dependency tree from linguistic annotations, and tree filtration to avoid semantic drift [CMS07, RJ$^+$99] introduced due to distant supervision.

The authors created a word-level embedding and trained continuous skip-gram model [MSC$^+$13] implemented in word2vec. They made use of DBpedia and Wikidata knowledgebases to acquire labels against relations. These seed labels were then used to gather similar words using Oxford-Dictionary[1], Wordnik[2], and Wordnet[3]. Resultant similar words become part of the label set to reflect a relation in vector representation.

The generalization step takes candidate trees into account along with the output of the embedded semantic module for filtration. The generalization algorithm is distributed over two functions which aim to return a generalized tree T, or an empty tree (provided two input trees $\{T_1, T_2\}$). Function 1 preserves the root from both candidate trees and passes it to function 2 for vertices generalization and stores the resultant tree in T. The candidate trees are iterated from root node twice to maintain a set of common *edge-labels* and *node-labels*. Function 1 is called recursively for each combination of edge and node labels. After recursion, all the edges that are common in both trees are removed, and only distinct edges are added to T.

Function 2 serves vertex generalization and focuses on vertex's attributes of *{label, lemma, {POS, NER, domain, range}}* by obeying the constraints of attributes in *left-to-right* priority-order. If the label is not matched then lemma of both vertices are verified. In case of no match, the remaining attributes are compared. Finally, false candidate tree patterns are filtered by eliminating candidates if they do not contain at least one label that is present in the label set representing relation in embedding semantic module.

2.3.1 Limitations

Lacking coreference resolution [LPC$^+$11] is a major drawback of Ocelot. In almost every article, pronouns are used instead of the entity's label once their mention has been used in the previous sentence. All such sentences that actually reflects a property but uses a pronoun, are not considered by Ocelot; causing it to be less scalable and having limited sentence consideration. RAPID overcomes this issue by applying coreference resolution and sentence collection technique 5.1.3 that gathers n next sentences until it finds the existence of any entity's mention (subject or object).

The tree generalization strategy makes Ocelot unique but has 2 major flaws. First, the vertex generalization only considers root's label/lemma for semantic purpose[4], and ignores all other valuable verbs, adjectives, and nouns that are encountered during iteration from root node.

[1] `https://www.oxforddictionaries.com`

[2] `https://www.wordnik.com`

[3] `https://wordnet.princeton.edu`

[4] As v_1 and v_2 in function 1 are always assigned by roots of their respective tree; that are passed to $generalizeVertices$ function.

Another flaw is to use the semantic comparison on a later step instead of utilizing it during vertex creation. Consider roots with labels *deny* and *refuse*, where both label and lemma differs from each other but have a high semantic similarity of 0.816 (using word2vec pre-trained model). In all such cases, Ocelot will *over-generalize* by comparing vertex with only {*POS, NER, domain, range*} attributes. RAPID overcome these issues by means of Forskipo Semantic Similarity (FSS) algorithm.

Using embedding semantics module is a worthy approach to avoid semantic drift but has a huge limitation due to the utilization strategy of Ocelot. The filtration step will eliminate abundant tree patterns since language is broad, and their approach requires the presence of at least one label that exists in the embedding semantic label set (reflecting a relation). Instead of *contains* validation, introducing an intermediate source embedding model to use vectors directly will resolve this issue. Our RAPID system dodges the problem by generating property embedding model (PEM).

2.4 Language-agnostic Relation Extraction from Wikipedia Abstracts

The author presents a machine learning approach for relation extraction that is independent of language features [Ben09] and utilizes background knowledge from existing knowledgebases. The aim is to identify patterns and train model by extracting features from Wikipedia abstract.

The core idea is to use DBpedia and identify links present in the abstract of an article and to use these links (reflecting entities) for determining relations which satisfy domain and range of DBpedia ontologies. Their approach makes use of *partial completeness assumption* [GTHS13] or *local closed world assumption* [DGH$^+$14] where relations acquired are considered correct. Therefore, all the relations that exist within the abstract are considered as positive examples; whilst the remaining relations are marked as negative training examples. Their features are entirely based on *entities* present in the abstract, further specifying *candidates* as the one that satisfies ontology constraint on domain and range. Naive Bayes, RIPPER [Coh95], Random Forest [Bre01], Neural Networks [Hay94] and Support Vector Machine [CST$^+$00] models were trained using features that considers the total number of candidates present in an abstract $F00$, the sentence in which candidate is used $F01$, position of candidate's sentence in an abstract $F07$ and, positional distance of candidate with other candidates $F02$ and their sentences $F03$. The same processing is done for all the entities (regardless of being a candidate or not) and features $\{F04, F05, F06\}$ are extracted. The best F-measure was obtained by Random Forest using ten-fold cross-validation.

2.4.1 Limitations

Even though the paper presents a unique approach that claims to be language-agnostic (LA), but is not entirely true. Their pattern formation is dependent on the structure of abstract, meaning that any document as an input that does not obey *abstract-writing* rules might result in no/false positive results. For example, *"the first city mentioned in the abstract about a person is that person's birthplace"* stated in [HP17] will give a false positive result for input sentence discussing only *death place*.

Rules and training based on abstract bounds the system to work only with the documents that follow the grammatical tone of an abstract. Hence, leaving the system with a narrow

input domain and reducing its scalability. Whereas RAPID is independent of corpus and can work even with a single input sentence. Moreover, their LA approach doesn't compare words semantically at all; leaving the system vulnerable to semantic drift. In addition to that, their training features *{F02, F07}* entirely depends on *abstract-writing* skills.

3 Background

In this chapter, we will briefly highlight the necessary tools/libraries that were used to develop RAPID.

3.1 WikiDump

The Wikimedia Foundation project distributes the archive of Wikipedia articles in the form of XML containing metadata of current and previous revision, and article text [FZG11]. These archives are known as *WikiDump* [KRPA10]. The text is in markup format and needs to be converted into plain text so it can be used for annotation purpose. We used WikiDump 2018 [1] (63.4 GB), and processed it using *WikiClean* [2] to transform it into a plain text. After processing, all the articles were indexed multithreadedly using Elasticsearch.

3.2 Elasticsearch

Elasticsearch is a highly scalable and distributed search engine based on Lucene [MHG10], that allows a user to index their documents. It supports a fully compatible text search over REST API and schema-free JSON documents [DG13]. We used *Elasticsearch 6.4.0* [3] and indexed every Wikipedia article's text against their URI. RAPID makes use of Elasticsearch for retrieving triple's article during the data collection step.

3.3 DBpedia

DBpedia is a knowledgebase having information stored in structured RDF format extracted from Wikipedia by the combined efforts of community. The transformed RDF structured data can be queried via SPARQL [ABK+07]. Over 1 billion triples have been extracted using Wikipedia content. RDF triple consists of a subject, predicate, and an object; where a predicate defines the relationship that holds between subject and object. For collecting triples against a property,

[1] `https://dumps.wikimedia.org/enwiki/20180701/`
[2] `https://github.com/lintool/wikiclean`
[3] `https://www.elastic.co/products/elasticsearch`

we used AKSW DBpedia SPARQL endpoint [4], and used *Apache Jena 3.10.0* [5] for executing SPARQL queries.

3.4 Stanford CoreNLP

Stanford CoreNLP [6] is a linguistic tool that allows to process natural language text. It annotates a document by the annotators passed in the pipeline. Along with the grammatical structure of a sentence, Stanford CoreNLP can determine the base form of a word, it's part of speech (POS), named entity recognition, and coreference resolution [MSB+14]. We used Stanford CoreNLP for lemmatization, entity recognition, POS, coreference resolution, and parsing through a semantic graph to generate new subgraphs against our patterns.

3.5 WordNet

WordNet is a semantic network database where words are interconnected by means of labeled arcs expressing relational characteristics. Synset and synonyms are its highlighted contributions. A synset is the lexical expression of a concept, and also incorporates gloss for the concept. These concepts are mapped together through synonyms. Synonyms also include mapping of word forms [Fel10].

3.6 Word Embeddings

The idea is to have a vector representation of a word in such a way that semantically similar words are more like have similar vector representation. Words represented in a learned distributed representation having dense, low-dimensional, real-values are called word embeddings. Where each dimension of the embedding corresponds to a latent feature of the word allowing it to capture semantic properties [MYZ13]. We used existing embedding models trained using Word2vec, Global Vectors for Word Representation (Glove) and fastText; and generated property embedding models using their available dataset. We will discuss briefly about the different embeddings below.

3.6.1 Word2Vec

Word2Vec is an approach to learn word embeddings based on vector-oriented reasoning. The relationships can be learned based on vector offset between words. For example, word *Queen* can be derived from $King - Man + Woman$; hence allowing to learn relationship of *male / female* [MYZ13]. We used the pre-trained model of Google News [7] having 300-dimensional vector consisting of three million words and phrases.

3.6.2 Global Vectors for Word Representation (Glove)

Glove is an extension of Word2Vec and is an algorithm for obtaining vector representation of a word using unsupervised learning. It combines the idea of merging matrix factorization like latent semantic analysis (LSA) [LFL98] and the local context window. Training a Glove model is

[4] `http://akswnc9.aksw.uni-leipzig.de/dbpedia/sparql`
[5] `https://jena.apache.org/`
[6] `https://stanfordnlp.github.io/CoreNLP/`
[7] `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/`

based on global corpus statistics (word-word co-occurrence) resulting in log-bilinear prediction-based model [PSM14]. We generated our PEM using existing Stanford Glove model [8] having 300-dimensional vector consisting of 2.2 million vocabulary.

3.6.3 FastText

FastText is like Word2Vec but is a time-efficient approach. It can train one billion words in less than 10 minutes. Instead of Bag of word approach (which is more expensive), fastText uses a bag of n-grams to capture partial information about word order. FastText makes use of softmax function for computing probability distribution over predefined classes [JGBM16]. Due to the use of a bag of n-grams, this approach can work with unseen words. We used the existing fastText model [9] provided by Facebook as a source model, and generated PEM using it.

[8] `http://nlp.stanford.edu/data/glove.840B.300d.zip`
[9] `https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip`

4

RAPID Architecture

RAPID architecture can be categorized into three main components of data collection, pattern generation, and property classification. The figure 4.1 below describes the overview of components present within RAPID architecture. In the following section, we will briefly discuss all the modular components.

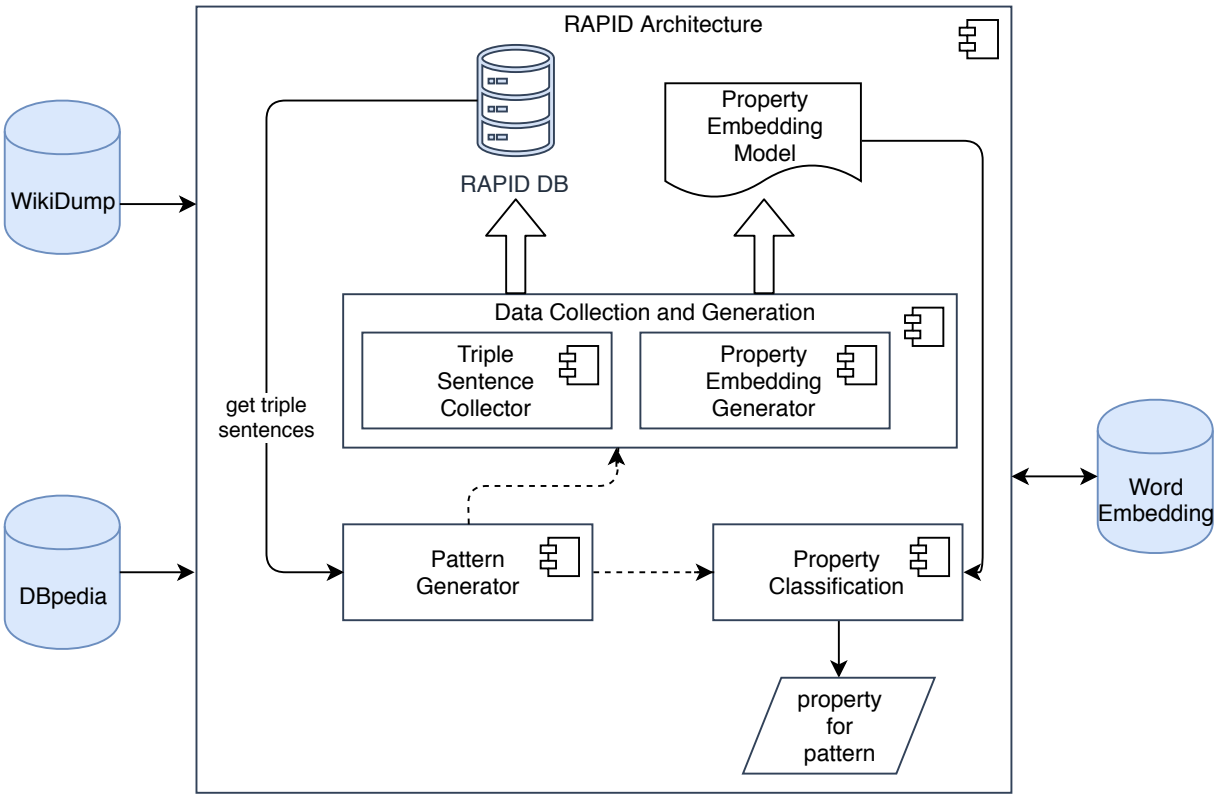

Figure 4.1: Overview of RAPID Architecture

4.1 Approach

The elasticsearch indexing created from processed WikiDump is used for collecting sentences that reflect a triple obtained from DBpedia. Patterns are generated from the triple entities

and the sentence reflecting it. We store this data into MySQL based RAPID database. Also, we gather word features for our properties during subgraph iteration. Moreover, a Property Embedding Model (PEM) is generated using WordNet. This PEM, source embedding model and existing patterns for property help us to classify pattern based on input sentence.

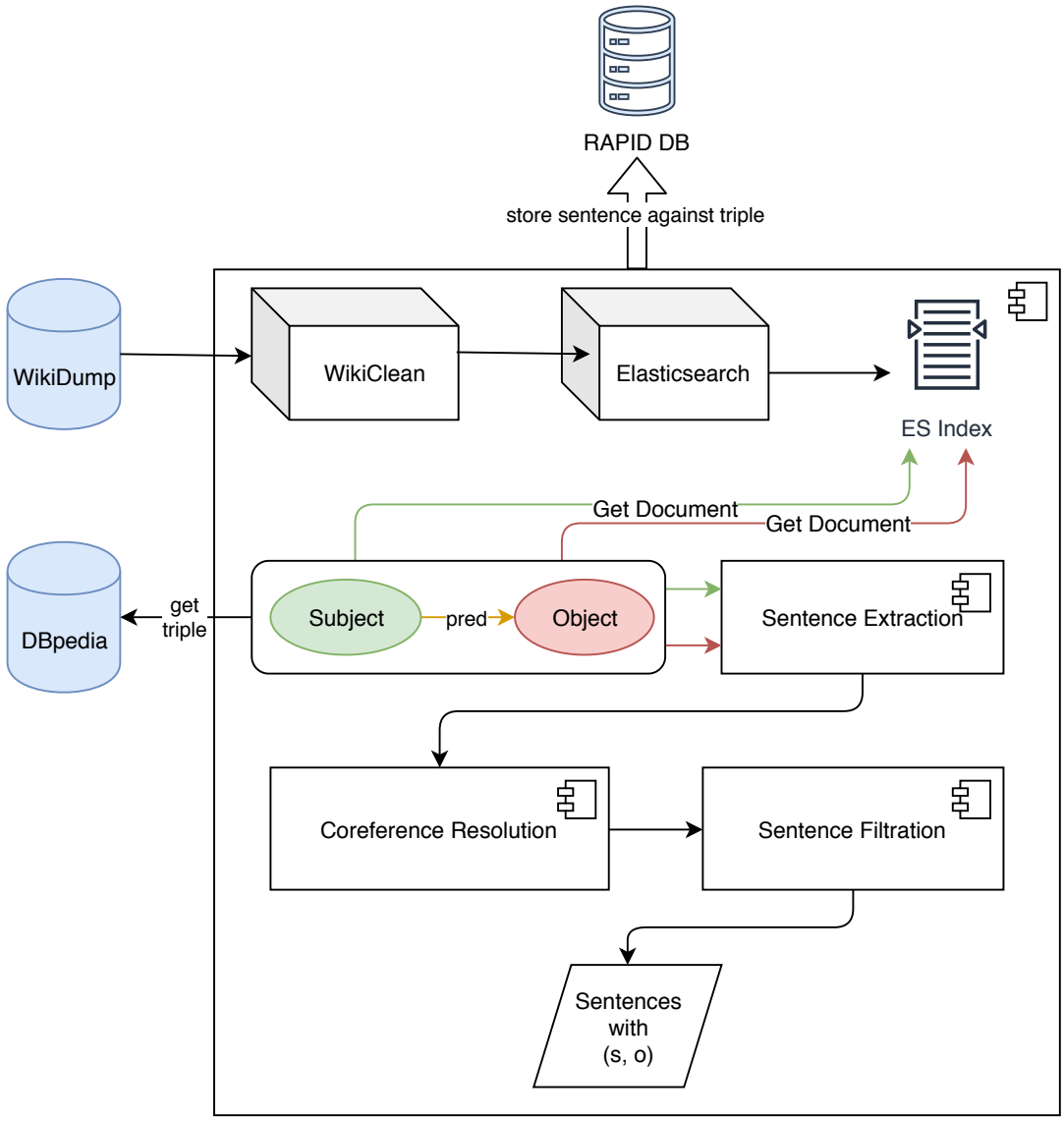

Figure 4.2: Overview of data-flow and sub-modules utilized by *Triple Sentence Collector*

4.1.1 Data Collection

Data collection process consist of two modules namely *Triple Sentence Collector*, and *Property Embedding Generator*. Both modules do not interact with each other during the collection step but are dependent on a few external systems.

Triple Sentence Collector

The data from WikiDump is transformed into plain text and is then indexed using elastic-search. We use DBpedia to retrieve triples against a property. These triples are passed to the

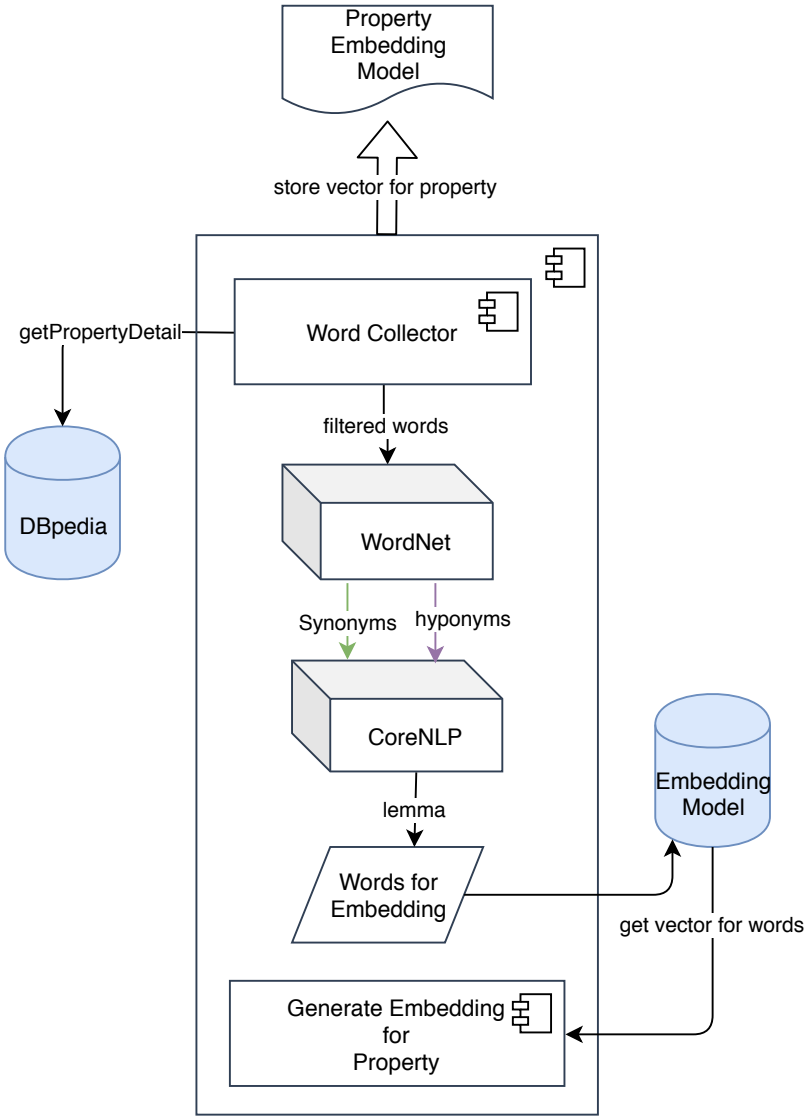

Figure 4.3: The components and sequence of steps used for generating *Property Embedding Model (PEM)*

sentence-extraction component which extracts sentences from elasticsearch indexing. The extracted sentences for a triple are passed to coreference resolution module that returns modified sentences with respect to the entity's mention. Finally, these modified sentences are filtered and stored in our RAPID database. Figure 4.2 represents the data-flow of discussed approach. Where section 5.1 shows how these modules work internally.

Property Embedding Generator

For each property, useful informations are collected from DBpedia and split into words. These words are further filtered and then passed to WordNet for getting synonyms and hyponyms. All the words retrieved from WordNet, and from DBpedia goes through lemmatization step performed using CoreNLP. Once the words are in their surface form, we get its vector representation and generate Property Embedding Model for each property. Figure4.3 represents the necessary steps needed for generating a Property Embedding Model.

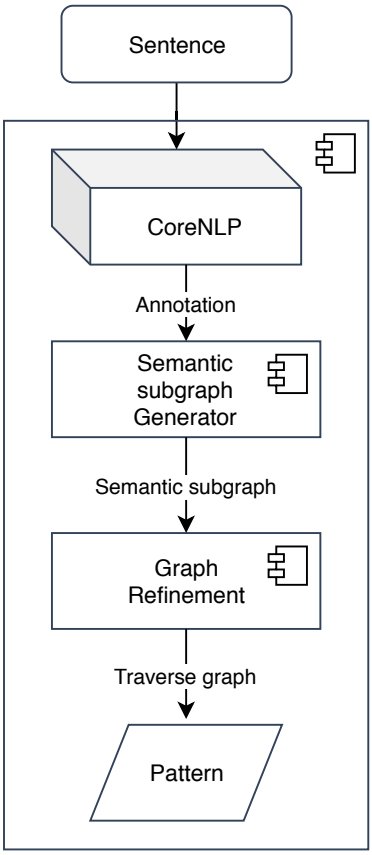

Figure 4.4: Representation of dependency flow from input to pattern generation

4.1.2 Pattern Generator

The pattern generator component 4.4 allows to generate pattern given an input sentence. Means, this module is also related to *Data Collection* as we are storing patterns that were generated using sentences collected against triples satisfying a property.

The given sentence is annotated using CoreNLP, and n number of semantic subgraphs are generated against the entities. These graphs go through refinement operations for graph pruning and modifications. Once, the subgraph is in its final form after replacing domain and range. We traverse our semantic subgraph to generate a pattern in our decided format.

4.1.3 Property Classification

Property classification is done with respect to a pattern, and their possible candidate properties that could possess a relation between entities present. Patterns from the candidate properties are extracted for comparison, along with their features. The features, embedding similarity for input pattern, and Forskipo-Semantic Similarity (FSS) against candidate patterns helps us to calculate confidence value. Property is decided based on the maximum score achieved. Figure 4.5 shows the fundamental modules that are called for gathering the necessary information to compute confidence value.

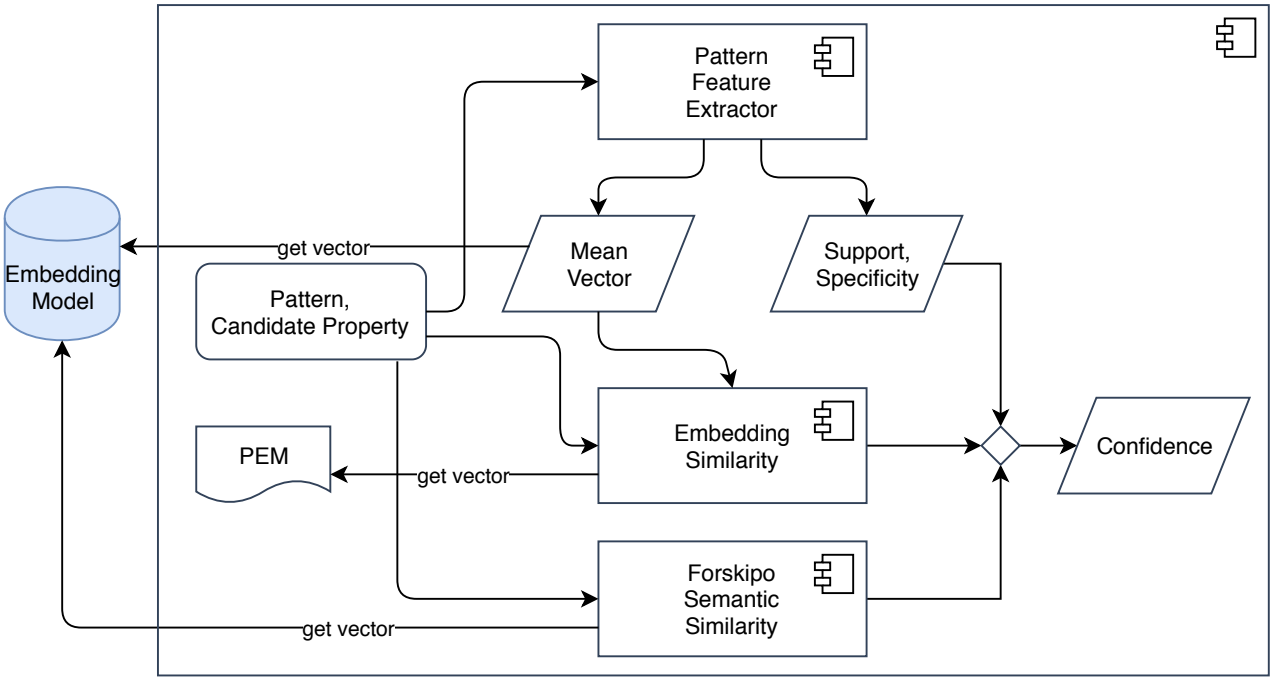

Figure 4.5: The modules required for calculating confidence value against a property

5 Implementation

This chapter describes the technical and theoretical approaches opted for each module, and how these modules together serve to identify relation from a pattern. The chapter is divided into four major sections of preprocessing, pattern generation, property embedding model generation, and property recognition.

5.1 Preprocessing

Preprocessing consist of all the steps needed to be achieved before pattern generation. Since we are relying on DBpedia triples that were extracted using Wikipedia. Therefore, all the sentences for a property against which patterns will be generated are a subset of Wikipedia's article sentences. The articles were indexed asynchronously using elasticsearch after converting each article from markdown to a plain format using WikiClean [1].

5.1.1 Index Creation

We created an index dataset *wikiclean* and indexed the text of each article present in WikiDump against it's URI using elasticsearch. The dataset is clustered over 5 fragments, and each indexed document contains properties of *title*, and *text* (article's content). The documents in the sentence extraction phase are retrieved using REST call [2]. Indexes were generated asynchronously using multi-threading.

Alternative approach could be to create indexes *sentence by sentence* and later use intersection for obtaining sentences having subject and object (during sentence extraction phase). This approach is easy to be implemented but have limitations while performing XOR operations on sentences, as indexing is done in parallel, therefore, sentences will lose its order.

5.1.2 Triples Selection

Humans keep on learning consciously and unconsciously through their surroundings [GP10]. Learnings are based on any means of communication; one such example is language. Reading influences a lot on language skills, therefore we decided to choose Wikipedia articles based on

[1] `https://github.com/lintool/wikiclean`
[2] `http://solide.cs.upb.de:9200/wikiclean/title_uri/Wikipedia_Article_URI`

their page-rank. It can be assumed that people would usually encounter similar grammatical tone (as of these articles) over any unstructured data.

For each of our selected DBpedia ontology, we queried 1000 triples based on subject's PageRank. The query uses two datasets of *DBpedia* [3] and *KIT PageRank* [4]; and sort the retrieved results in descending order of their *vrank:rankValue*. The SPARQL query in listing 5.1 is used for getting property triples:

Listing 5.1: SPARQL query to retrieve triples based on page-rank against properties

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vrank:<http://purl.org/voc/vrank#>
PREFIX dbo:<http://dbpedia.org/ontology/>

SELECT distinct ?s ?subLabel ?o ?objLabel
FROM <http://dbpedia.org>
FROM <http://people.aifb.kit.edu/ath/#DBpedia_PageRank>
WHERE {
?s dbo:PROPERTY ?o .
?s rdfs:label ?subLabel .
?o rdfs:label ?objLabel .
{
SELECT distinct ?s ?v WHERE {
?s dbo:PROPERTY ?o .
?s vrank:hasRank/vrank:rankValue ?v.
}
ORDER BY DESC(?v)
LIMIT 1000
}
FILTER (lang(?subLabel) = 'en' && lang(?objLabel) = 'en')
}
ORDER BY DESC(?v)
```

5.1.3 Sentence Extraction

RAPID makes use of *local closed world assumption* [DGH+14] where sentences extracted are considered correct. To get the sentences that satisfy a property p, we use the triples collected in the previous phase. Since the triples are from DBpedia knowledgebase, it must be extracted from some Wikipedia article. Therefore, we used the subject and object resource URI as our candidate documents. Let d be a document (representing Wikipedia article) retrieved from elasticsearch consisting of n total sentences. Then we can express a document by:

$$d = \bigcup_{i=1}^{n} se_i \tag{5.1}$$

Where a sentence se_i is created by a combination of words. Subject l_s and object l_o labels are composed of a word or more. Therefore, we can say that $l_s \subseteq se_x$ and $l_o \subseteq se_y$. We are

[3] `http://dbpedia.org`
[4] `http://people.aifb.kit.edu/ath/#DBpedia_PageRank`

interested in sentences from x to y, or y to x. In some cases, x can be equal to y. This criteria for our candidate sentences can be represented as:

$$CSE = \left\{ \bigcup_{i=x}^{y} se_i : (\text{se}_x \supset l_s) \wedge (\text{se}_y \supset l_o) \right\} \vee \left\{ \bigcup_{i=y}^{x} se_i : (\text{se}_y \supset l_s) \wedge (\text{se}_x \supset l_o) \right\} \tag{5.2}$$

But it is not true to always find a relation where there are at least two entities. Consider the following sentence: "Richard met Prof. Heinz in a conference. Later, Heinz supervised him for Ph.D." The first sentence which comprises of both entities lack a meaningful relational property, whereas the second sentence having only one entity leads to *dbo:doctoralAdvisor* and *dbo:doctoralStudent* ontologies.

In natural language, it's common to encounter sentences where entities are described in a previous sentence and next sentence(s) uses pronouns. To solve this scenario, after we have identified our candidate sentence, we keep appending next k sentences which have mentioned either subject or an object label. We stop appending if none or both labels are discovered.

$$CSE = CSE \cup \left\{ \bigcup_{k=y+1} se_k : (se_k \supset l_s) \oplus (se_k \supset l_o) \right\} \tag{5.3}$$

After we have extracted our sentences that could possess relationships between entities. We need to refine our sentences and remove unnecessary sentences from our candidate sentence set.

Sentence Refinement using Coreference Resolution

Considering triple $X = (s, p, o)$ for which CSE sentences were extracted, we need to replace all the mentions with the labels of triple's subject l_s and object l_o respectively. Replacing a mention that is a noun will not have any effect on grammatical structure. But the typed dependencies are changed while replacing a pronoun with a noun-based entity if the next words have a part-of-speech (POS) in $\{Noun, Adjective\}$. To retain the grammatical structure in such cases, we append $'s$ after replacing a mention with resource label. We used the approach described in Algorithm 1 for the sentence refinement. If an empty set of labels is passed as an input, then the mentions are replaced with the very first mention present in their respective coref-cluster. The algorithm keeps the sentences unchanged if there doesn't exist any coref-cluster for an annotated document d.

The following example makes the working of algorithm 1 easier to understand.

LabelSet:	{*Albert Einstein, Germany, Nobel Prize in Physics*}
CSE:	{*Albert Einstein was born in Germany.*, *He got Nobel Prize in Physics.*}
corefSentences:	{*Albert Einstein was born in Germany.*, *Albert Einstein got Nobel Prize in Physics.*}

Filtration of Candidate Sentences

The output of algorithm 1 is again processed through sentence extraction phase with a modification that both subject and object label must occur in the same sentence. All the other sentences

Algorithm 1: Sentences Coreference Resolution Transformation

```
input  : Annotation document, Set<String> corefLabelSet
output: List<String> corefSentences
Let DTreeUtils be a helper class for annotated document and
Let CCMap represent coreference chain mapping against cluster id;
HashMap<Integer, String> cIdResourceMap :=
  DTreeUtils.setClusterLabelMap(CCMap, corefLabelSet);
foreach CoreMap sentence : getSentences(document) do
    List<String> sentenceWords; List<CoreLabel> tokens := sentence.getTokens();
    for int i := 0; i < token.size(); i++ do
        CoreLabel token := tokens.get(i);
        Integer corefClustId := token.getClusterId;
        CorefChain corefChain := CCMap.get(corefClustId);
        if corefChain is null then
            sentenceWords.add(token.word());
            continue;
        end
        String firstMention;
        if corefClustId ∈ cIdResourceMap then
            firstMention = cIdResourceMap.get(corefClustId);
        end
        else
            firstMention = corefChain.getFirstMention.mentionSpan;
        end
        if token.POS.contains("PRP") then
            if validateSize() then
                CoreLabel nextToken := tokens.get(i + 1);
                if nextToken.POS.contains("NN") ∨ nextToken.POS.contains("JJ")
                  then
                    sentenceWords.add(firstMention + "'s");
                end
                else
                    sentenceWords.add(firstMention);
                end
            end
        end
        else
            sentenceWords.add(getMentionOrToken(sentId));
        end
    end
    String corefSentence = String.join(sentenceWords).append(".");
     corefSentences.add(corefSentence);
end
return corefSentences;
```

are removed. Our final candidate sentence set having n sentences can be represented as:

$$CSE = \bigcup_{i=1}^{n} se_i : (\text{se}_i \supset l_s) \wedge (\text{se}_i \supset l_o) \tag{5.4}$$

The following example shows how sentences are filtered based on the provided triple.

CSE:	{*Albert Einstein was born in Germany.*, *Albert Einstein got Nobel Prize in Physics.*}
Triple X:	dbr:*Albert_Einstein* dbo:*award* dbr:*Nobel_Prize_in_Physics*
label(*subject*):	*Albert Einstein*
label(*object*):	*Nobel Prize in Physics*
sentenceFiltration(CSE):	{*Albert Einstein got Nobel Prize in Physics.*}

5.2 Property Embedding Model Generation

To classify a sentence against a property p, there must exist certain words w that semantically reflects the obedience of that property. Most meaningful words resembling the property can be obtained via it's label L_p, and the comment L_c against it (if exist any). Let L be our vocabulary such that:

$$L = L_p \cup L_c \tag{5.5}$$

In many cases, the only label for a property exist and the comment is missing. Since language is a broader way of expressing *things*, we need to extend our vocabulary somehow. We made use of WordNet for this purpose. This vocabulary will be then used for generating vectors that represent a property.

5.2.1 Property Vocabulary Expansion

For vocabulary expansion, there should be at least one word for which we need more words that have the exact same meaning or are nearly the same semantically. *Synonyms* are one way to get such words. We made use of WordNet Synset that provides a set of cognitive synonyms having nouns, verbs, adjectives, and adverbs mapped against a word. Our aim is to include these meaningful words in our vocabulary. First, we removed all the stop-words from the comments $L_c = L_c - SP$ (where SP is a set of stop words) and then retrieved synset against the property label and remaining words.

Considering property *dbo:spouse* having label *spouse*, WordNet synset provides us these cognitive synonyms:*{partner, mate, better half, spouse, married person}*. But these synonyms aren't enough to express the property (near to) their completeness. Potential candidate words like *husband* and *wife* cannot be retrieved from synonyms as they differ from the *gloss* of word *spouse*. To overcome this issue and further extend our vocabulary, we used *hyponyms* [CG90] which expresses *type/kind of* relationship. If every Y is a *kind of* X, then Y is a hyponym of X. The hyponym of word spouse are listed here: *{married man, wife, honeymooner, polygamist, monogamist, husband, married woman, bigamist, helpmeet, consort, monogynist, newlywed, hubby, helpmate}*. After getting our resultant set, we did manual filtration and removed a few words.

$$L = L \cup wordNet(L) : wordNet(L) = \{synonyms(L) \cup hyponyms(L)\} \tag{5.6}$$

5.2.2 Vocabulary Normalization

The English language consist of several grammatical forms and tense just like all other languages. Instead of including all the words in our vocabulary, we should convert the word into its base form. We performed lemmatization [MRS10] using Stanford CoreNLP and achieved the surface form of a word. For example, all the following words *{employ, employed, employing}* are transformed into *employ*. But using only lemmatization is not sufficient if the word in the vocabulary is a noun. For instance, the lemma of *{employee , employees}* is *employee*, whereas *employer* has a lemma *employer* and the lemma of *employment* also remains the unchanged.

Since relations are also a way to define characterization of an act, therefore we need to convert these nouns into verbs to have same lemmatization effect. For the input word (that belongs to our vocabulary), we retrieved all the lexical-related words and pruned the return set by keeping only the words with POS *verb*. Using this approach returns *employ* against the above-mentioned nouns. As a result, now our vocabulary contains only word *employ* against words: *{employ, employed, employing, employee, employees, employer, employment}*. Let l_w be the resultant word in our vocabulary, then:

$$l_w = lemma(nn2vb(word)) \tag{5.7}$$

5.2.3 Property Vector Generation

Now that we have a vocabulary for each property, we would need to classify our sentence against these properties if vocabulary words are encountered in our input sentence (after lemmatization and lexical-relation). One could do this by setting a HashMap of vocabulary set against each property, and then select the property with maximum vocabulary match. The problem with this approach is its limitation of handling words that are not part of the vocabulary but are still semantically related to the property. Since our vocabulary is also a semantic representation of the property, therefore all the words that are semantically related to the vocabulary words should also be considered. We need a way to represent our vocabulary in such a way that it can be compared with the words that are not part of it.

One way to compare these words is to use word embeddings where every word is represented as a vector of dimension d having real number values. These d values against a word is a semantic feature for it [MYZ13]. Hence, the words that are not in our vocabulary but are semantically related, should be near to each other in the vector space. Since our vocabulary is a collection of words that are mapped against a property p, therefore we need to represent all the words representing the property p in form of a single vector having d dimension.

Transforming Vocabulary to Property Vector Representation

Let $P = \{p_1, p_2, ..., p_i\}$ be a set of properties having property p_i, then the vocabulary against each property can be expressed as l_{p_i}. Each vocabulary consist of j words where $j \geq 1$. j could be different for each vocabulary. Let $l_{p_i} = \{w_1, w_2, ..., w_j\}$, then there exist a vector representation v_{w_j} for each word. The vector has dimension d which remains consistent for all words regardless of the vocabulary it belongs to.

Our aim is to generate a single vector u_i of dimension d against property p_i. Which means that our property embedding model would have exactly i rows having column of size d with real

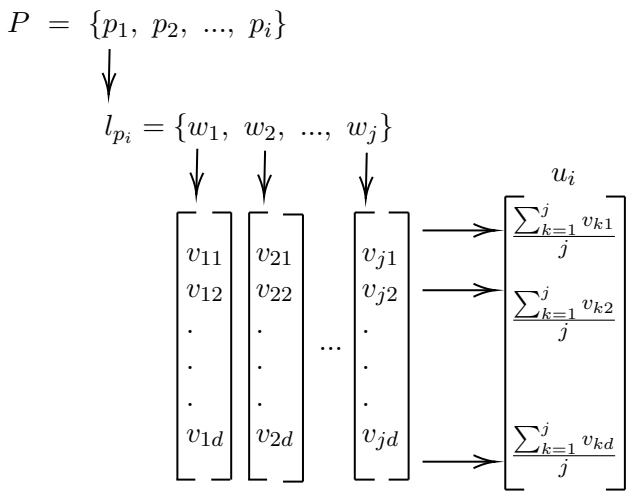

Figure 5.1: Property vector representation by taking average of words that exist in its vocabulary.

number values.

$$p_i \implies u_i : u_i = \{x_1, x_2, ..., x_d\} \tag{5.8}$$

We could do it by either summing the vectors, concatenation of word vectors, or taking an average of all the word vectors in a vocabulary. In our case, we cannot concatenate vectors due to the variant size of words in the property's vocabulary. Also, the input sentence for which relation has to be extracted would not necessarily have the same number of words in it. We decided to go for the average vector approach as it is known to provide the most optimal results for merging vectors [Soc] with respect to word embeddings.

All of our words in vocabulary can be represented as a vector matrix, where each row is a word vector having d columns of real-number values. The number of rows is depended on the size of vocabulary. Since in word embedding each dimensional value is a semantic feature, therefore we need to make sure that the feature conceptualization should remain intact after merging vectors. For this reason, first we compute the sum of each column c values present in row j; $\sum_{k=1}^{j} v_{kc}$. And then take an average by dividing the summation value by the size of vocabulary. Figure 5.1 shows the conceptual flow for property vector generation. Our final vector u_i against a property p_i would have values:

$$u_i = [\frac{\sum_{k=1}^{j} v_{k1}}{j}, \frac{\sum_{k=1}^{j} v_{k2}}{j}, ..., \frac{\sum_{k=1}^{j} v_{kd}}{j}] \tag{5.9}$$

5.3 Pattern Generation

Pattern generation depends on the semantic graph acquired by annotating input sentence using Stanford CoreNLP. The core idea is to find the shortest semantic subgraphs between two entity nodes. In a semantic graph generated by CoreNLP, each token (word) is represented as an IndexedWord (possessing grammatical and positional attributes) connected via typed dependency edges. The following subsections explain our aim and the approach for semantic subgraph generation.

5.3.1 Semantic Subgraph Generation

Let $G = (V, E)$ be the original semantic graph obtained from an annotated document d, where V is a set of IndexedWord having each v connected to typed dependency edge $e \in E$. Our aim is to extract n subgraphs where each subgraph is equivalent to our pattern pt. Since the subgraph is always extracted between two entities x_1 and x_2, we can denote our subgraph as:

$$\mathrm{G' = (V', E') : (V' \subset V \wedge E' \subset E) \wedge (\exists x_1 \in V' \wedge \exists x_2 \in V')} \tag{5.10}$$

Entity Nodes Identification

The very first step for semantic subgraph generation is to identify IndexWord node(s) representing subject and object labels. If the labels are single-word, then we return all the nodes having the same *CoreLabel* value as of our labels. The same approach will not work for nodes identification if the label consists of multiple words. Consider the following sentence: "University of Saarland is a member of the University of Paderborn.". Having the subject "University of Saarland" shares the words "University of" of the object "University of Paderborn". If we consider all the nodes based on words patterns only, then we might end-up having exact same node (IndexedWord with same vertex position) for both subject and object. Let IWS represent the set of indexedWord nodes iw obtained by matching of sub-label words of the subject, and IWO be a set of indexedWord nodes against object sub-label. Then, our aim is to detect nodes where the intersection between subject and object indexedWord nodes should be null.

$$IWS \cap IWO = \varnothing \tag{5.11}$$

$$\Big\{IWS = \{\mathrm{iw_i} + \mathrm{iw_{i+1}} + \cdots + \ \mathrm{iw}_{\ \mathrm{len(subjsplit)}}\}, IWO = \{\mathrm{iw_j} + \mathrm{jw_{j+1}} + \cdots + \ \mathrm{iw}_{\ \mathrm{len(objsplit)}}\}\Big\}$$
$$: \{subjSplit[0] = getWord(iw_i)\} \wedge \{objSplit[0] = getWord(iw_j)\}$$

If the label consists of multiple tokens (words), we select the first token and find all the nodes matching that token. The matching nodes are then filtered based on criteria that the next consecutive nodes must match the sub-label of their corresponding split.

Subgraph Generation

All the nodes that were identified using the approach in previous subsection 5.3.1, are used for subgraph generation. We form combinations of entities from both set of IWS and IWO. A combination is never made within the same set. For each combination of entities, we use the Dijkstra algorithm [PRV09] to determine the shortest path of semantic graph edges that connects these two nodes regardless of edge direction. We specifically chose the undirected edge as it is not necessary to have a consecutive out-edges from the source node (subject node) to the target node (object node).

Once we have obtained the shortest path, we have a list of semantic graph edges. These edges contain information regarding the governor and dependent nodes. A governor *gov* is a node which has an out-edge e, while dependent *dep* node is connected to *gov* via the same incoming edge. Using these edges, we generate a new semantic subgraph $G\prime$ having governor $gov\prime$ and dependent $dep\prime$ nodes attached to each edge $e\prime$. But further extend our subgraph using typed dependency merging set.

Extending Semantic Subgraph We formulated typed dependency merging set $MTYPD$ based on grammatical relations categorization. This set contains all the typed dependencies expressing grammatical knowledge of word's *possession*, and *part-of* information. Possession typed dependencies includes all the noun, nominal and predicate dependents along with word modifiers. Whereas part-of includes all the compounding and unanalyzed typed dependencies[5].

From our generated semantic subgraph $G\prime$, we iterate through all the existing edges $e\prime$ and get it's governor $gov\prime$ and dependent $dep\prime$ nodes. Each of the $gov\prime$ and $dep\prime$ is compared against original semantic graph G. For each of these nodes, we retrieve it's out-edges e from original graph and check if the typed dependency of e is in our merge set $MTYPD$. If $e \in MTYPD$, then we create an edge in our semantic subgraph $G\prime$ and include the dependent node dep of e in $G\prime$. Algorithm 2 describes how a semantic subgraph is generated.

Determining Root Node The generated subgraph $G\prime$ is only a subgraph having nodes connected via edges but without grammatical semantics. A semantic graph must have a root to drive semantics out of it. For this purpose, we need to identify the root amongst all the nodes present in a subgraph. This is done by iterating across nodes and collecting the number of incoming (in-degree d_{in}) and outgoing edges (out-degree d_{out}). A root node is the one which has no incoming edges, but must have at least one out-going edge i.e. $root = r_n : d_{in} = 0 \wedge d_{out} \geq 1$.

Since we have used Dijkstra regardless of edges direction, therefore in most cases, the path from source to target contains both kind of edges (incoming and outgoing). Hence, giving us the root node. But if all the edges in the path contain either incoming or outgoing edge, then root cannot be determined. In this scenario, we discard our semantic subgraph and construct a new one by utilizing the root node of the original semantic graph. The subgraph is generated based on paths from *source-to-root*, and *root-to-source*.

5.3.2 Semantic Subgraphs Filtration

Our aim is to have each subgraph as specific as possible. This allows us to avoid semantic drifts by eliminating subgraphs that could reflect more than one property. If a single sentence contains multiple properties, then in most cases, a subgraph (that should reflect a single property) would include nodes that are semantically part of another subgraph. It is not possible to remove such subgraphs if coreference resolution (CR) has not been performed over the sentences.

Consider the following sentence: *"Ahmed studied at University of Paderborn, and his thesis was under the supervision of Dr. Ricardo."*. A path from subject *Ahmed*, to object *Dr. Ricardo* would give us the subgraph containing both nodes of label *studied* and *supervision* while traversing from the root. Since we have implemented coreference resolution, therefore the sentence has already been transformed into *"Ahmed studied at University of Paderborn, and Ahmed's thesis was under the supervision of Dr. Ricardo."*. Implementing CR would always let us generate an extra subgraph that reflects only one property. Following are the serialized representation of two semantic subgraphs generated using subject *Ahmed* and object *Dr. Ricardo*:

Subgraph G_1 generated using subject *Ahmed* at index 0, and object *Dr. Ricardo* at index 15.

[5] `http://universaldependencies.org/docs/en/dep/`

Algorithm 2: Semantic Graph Generation

input : Original semantic graph G, Source IndexedWord iw_s, Target IndexedWord iw_t
output: Semantic Graph $G\prime$

```
1  Let MERGE_TYPED_DEPENDENCIES contains possession and part-of
     dependencies
2  G' := new SemanticGraph();
3  edgeList := G.getShortestUndirectedPathEdges(iw_s, iw_t);
4  foreach SemanticGraphEdge edge : edgeList do
5  |  G'.addEdge(edge);
6  end
7  foreach SemanticGraphEdge edge : edgeList do
8  |  IndexedWord gov : = edge.getGovernor();
9  |  IndexedWord dep := edge.getDependent();
10 |  origGovOutEdges := G.getOutEdgesSorted(gov);
11 |  origDepOutEdges := G.getOutEdgesSorted(dep);
12 |  foreach SemanticGraphEdge govEdge : origGovOutEdges do
13 |  |  IndexedWord outNode = govEdge.getDependent();
14 |  |  if (outNode ≠ dep)
15 |  |      ∧ (govEdge.getRelation ∈ MERGE_TYPED_DEPENDENCIES)
16 |  |      ∧ (edge(govEdge.getGovernor(), outNode) ∉ G') then
17 |  |  |  G'.addEdge(G.getEdge(govEdge.getGovernor(), outNode));
18 |  |  end
19 |  end
20 |  foreach SemanticGraphEdge depEdge : origDepOutEdges do
21 |  |  IndexedWord outNode = depEdge.getDependent();
22 |  |  if (depEdge.getRelation ∈ MERGE_TYPED_DEPENDENCIES)
23 |  |      ∧ (edge(depEdge.getGovernor(), outNode) ∉ G') then
24 |  |  |  G'.addEdge(G.getEdge(depEdge.getGovernor(), outNode));
25 |  |  end
26 |  end
27 end
28 IndexedWord root' := extractRoot(G');
29 if root' = null then
30 |  G' := addRootEdges(G);
31 |  root' := getRoot(G);
32 |  G'.setSemanticGraphRoot(root');
33 end
34 return G';
```

```
[studied/VBD
    nsubj>Ahmed/NNP
    conj:and>[supervision/NN
                cop>was/VBD
                case>under/IN
                nmod:of>[Ricardo/NNP case>of/IN compound>Dr./NNP]]]
```

Subgraph G_2 generated using subject *Ahmed* at Index 7, and object *Dr. Ricardo* at index 15.

```
[supervision/NN
    nsubj>[thesis/NN nmod:poss>[Ahmed/NNP case>'s/POS]]
    cop>was/VBD
    case>under/IN
    nmod:of>[Ricardo/NNP case>of/IN compound>Dr./NNP]]
```

Our aim is to get rid of subgraph G_1 since the node *studied* has no relation with our object term, and will only introduce semantic diversion form our desired context. This problem can be solved by taking the roots (*studied*, *supervision*) of both subgraphs into account. Let $R = \{r_1, r_2\}$ be a set containing the roots of generated subgraphs G_1 and G_2. If any subgraph G contains node iw that is the root of any other subgraph, then we delete the current subgraph G.

$$DelSubG(G) = true \iff iw \in G \wedge iw \in \{R - r_G\} \tag{5.12}$$

Since G_1 has $l_{iw} = supervision$ which is the root of another subgraph, therefore we will remove G_1 from our generated subgraph set. This strategy would never delete any potential subgraphs since the graphs were generated using coreference resolution. We cannot simply select the subgraph that has the least summation distance from the subject and objects to the root node. Doing this would restrict us to have only one subgraph per subject and object. Once we have our desired subgraphs, the next step is to apply refinement operations to formulate patterns against each subgraph.

5.3.3 Subgraph Refinement

Once we have collected our desired semantic subgraphs, we need to prune our graphs (if possible) by removing certain nodes that could be present in subgraphs generated using other entity combinations. Also, we need to keep track of the subject and object nodes position that would be necessary for our patterns formation.

Node Removal and Inclusion

In English grammar, a nominal refers to the grouping of nouns and adjectives that share the same characteristics [Mac83]. And, modifiers [Niv05] are the grammatical elements that serve the change of contextual meaning. In Stanford CoreNLP, nominal modifiers are represented by *nmod* typed dependency. The following example 5.2 would help us in the understanding of nominals and the possible pruning we can make in our existing subgraphs:

The sentence in figure 5.2 have entities *{Albert Einstein, Ulm, Kingdom of Wurttemberg, German Empire}*, where *Albert Einstein* is the subject, and all other entities are object. Since the property *birthPlace* remains the same for all the objects, therefore the graph filtration strategy discussed 5.3.2 would not work in this scenario. Retrieving a semantic subgraph from *Albert Einstein* to *German Empire* will include other objects too. It is not a problem semantically since

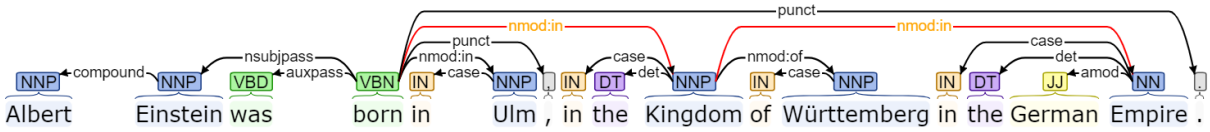

Figure 5.2: Inclusion of Object node *Kingdom* while considering *German Empire* as object.

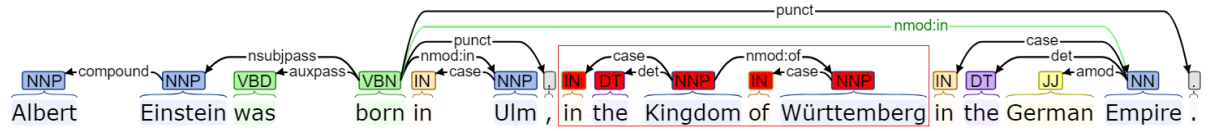

Figure 5.3: Creating new semantic edge from previous node. And removing nodes from semantic graph that are isolated.s

the context remains intact, but including these nodes would have an impact on our patterns that are further used for property classification.

We solved this issue by formulating a rule that *"If a nominal modifier is followed by the same nominal modifier, then the semantic graphs can be split into two"* – as both dependencies reflect a change in context from each other, but have similar context with respect to the node iw_{i-k} present before *nmod*. In the case of object *German Empire* iw_i, the node iw_{i-1} with label *Kingdom* will be removed (along with the nodes that are connected via out-degree). This will make the *Empire* node iw_i (and its out-nodes) isolated from the subgraph. A *SemanticGraphEdge* is created from node iw_{i-k} to node iw_i with the same nominal modifier typed dependency. Following 5.3 figure shows the transformed subgraph.

Domain and Range Replacement

Our patterns that will be generated using these semantic subgraphs has to be classified with the patterns generated from the input sentence. We need a way to distinguish the presence of our subject and object in it. Keeping the triple resources as it is in the subgraph would produce ambiguity since classification will not be able to determine the respective positions of subject and object. Therefore, we iterate across our subgraphs and check each node if it is present in IWS (subject nodes identified 5.3.1) or IWO (object nodes identified 5.3.1). The *orginalText* of the nodes present in IWS are replaced with $\%D\%$, while we replace the *originalText* with $\%R\%$ in case of IWO. Please note that we are not losing any information since the previous label of the node can still be retrieved using *CoreLabel* functionality.

5.3.4 Pattern Features

To extract relations from the sentence(s), we need to gather the words that make up the context and reflects the property. We traversed our subgraph and extracted *nouns* (other than proper singular nouns), *verb*, and *adjectives* that are not part of subject or object entity. All the nouns are looked for their verbs using WordNet lexical relations 5.2.2, assuming that properties are meant to represent an act which is best described by a verb. Finally, we replace all the verbs by their lemma and store it against our patterns.

5.3.5 Pattern Representation

For our data collection, all the patterns are generated using semantic graphs from annotated sentences. Classifying an input sentence against P candidate properties will bound us to read

thousands of annotation files and create semantic graph instances. Another way around is to use serialized tree representation and create an instance after parsing the pattern. All these approaches are computation expensive, and since we have already performed semantic subgraphs filtration and node reduction, therefore we do not need a tree structure to further generalize our subgraphs. An appropriate way is needed to represent our patterns in such a way that graph structure and semantic still remains intact. We created a generalized, and extended pattern representation for each of the subgraph.

Generalized Pattern

The generalized pattern does not contain any label in it. Patterns are solely composed of *{typed dependencies, POS, %D%, %R%}*. We formulated our patterns in such a way that the path from the root to the subject will always reside on the left; root to object path on the right; whereas root's lemma remains in the middle of the pattern.

$$\{rootToSubjectPath\}rootLemma\{rootToObjectPath\} \tag{5.13}$$

The sentence *"Ahmed studied at University of Paderborn, and his thesis was under the supervision of Dr. Ricardo."* with subject label *Ahmed* and object *Dr. Ricardo* will lead to following generalized format. Note that coreference resolution, subgraph filtration and refinement has already been performed before pattern generation.

Pattern:*{(NN)-nsubj>(NN)-nmod:poss>%D%(NNP)}supervise{(NN)-nmod:of>%R%(NNP)}*

We first traverse the subgraph from root to the subject and merge all the typed dependencies and POS if the following node shares the same edge type along with the POS type. This makes our pattern generalized by handling the sentences that differ due to existence of extra nodes connected via *MTYPD* 5.3.1.

Extended Patterns

Extended patterns follow the same representation format 5.13 but also include labels. These patterns are the actual representation of the semantic graph path from the root to their respective subject and object. We do not merge any nodes, therefore extended patterns differ a lot from other patterns.

Our pattern format does not lose the graph and grammatical structure since the traversal path remains consistent, and also allows us to distinguish subject and object directly. This format helps us to easily make comparisons while classifying a property with respect to entity pairs.

5.4 Property Recognition

Given an input sentence, our aim is to identify the properties that could exist between the pair of entities present in it. First, we perform an entity recognition [ZS02] step that is a prerequisite for pattern generation. Since we are dependent on Stanford CoreNLP for entity recognition, and if CoreNLP cannot identify any entity then no pattern would be generated. Hence, no property would be identified against the input sentence. To make RAPID flexible, it allows the user to pass entities explicitly which can help RAPID's integration with any other *Entity Recognition System*.

After identifying entities, we make combination of entity pairs $ep(e_1, e_2)$ considering them as *subject* and *object* for to-be-classified property p. These pairs are passed to the pattern generation module $pg(ep(e_1, e_2))$ that returns a set of patterns $PT = \{pt_1, pt_2, ..., pt_k\}$ against it. These patterns have to be classified against all the possible properties that share the same domain and range as of our entity pair. Therefore, our candidate property set CP is a subset of all properties P for which we generated patterns in preprocessing step of *Data Collection*; $(CP \subset P)$. Let ϱ be our confidence score for a pattern pt against property $cp_i \in CP$. Then property classification for a pattern can be expressed as:

$$prop(pt) = p : p = cp_i, arg_{max} \{\varrho(pt, cp_i) \geq \theta(cp_i)\} \tag{5.14}$$

For a given sentence having at least two entities, the generated patterns could be greater than, or equal to one. Which means, even though we have gathered sentences by considering 1000 triples for each property, it is not always true to have the number of patterns equivalent to the number of sentences (also that there could be multiple sentences against a triple). Therefore, the features that we will use for our scoring has to be normalized. Following are the features we used to calculate our confidence value.

5.4.1 Support

Given a pattern pt being classified against candidate property cp_i, support is the number of times pt occurs in PT_i. Where PT_i is a *bag* containing all the patterns that were generated (using triple sentences) during data collection process against property cp_i.

$$sup(pt, cp_i) = \frac{freq(pt) : pt \in PT_i}{totalPatternInProperty(cp_i)} \tag{5.15}$$

5.4.2 Specificity

Given a pattern pt being classified against candidate property cp_i, specificity is the number of times pt occurs in properties OP other than cp_i. Where $OP \subseteq \{P - cp_i\}$ having properties op_k.

$$spc(pt, cp_i) = \frac{freq(pt) : pt \in getPatterns(OP)}{\sum_{k=1}^{sizeOf(OP)} totalPatternsInProperty(op_k)} \tag{5.16}$$

5.4.3 Pattern Similarity

Our patterns were generated using the CoreNLP semantic graphs having typed dependency edges, and nodes with several attributes like POS, NER, node label, and so on. Our patterns have a specific format where the root node in the pattern will always own the center position, while the subject path shares the left side to the root and object path to its right. Consider a sentence *"Albert Einstein was born in Germany"* with only two entities e_1 =*Albert Einstein* and e_2 =*Germany*, then to identify a property, we make pair of entities $\{(e_1, e_2), (e_2, e_1)\}$ and generate patterns for it. These entity pair patterns are then classified against the candidate property set. The particular format we are using for our patterns provides a higher similarity score for the entity combination that closely matches with the patterns obtained during data collection of our triple sentences.

Time-Efficient Pattern Similarity

Since the generalized pattern format only consist of *{typed dependencies, POS, %D%, %R%}*, and is free from any contextual word (as we already capture the context by keep track of

nouns, verbs and adjectives encountered while traversing our sub-semantic graph), therefore one can use string similarity algorithms likes *n-gram* (by means of intersection) [Kon05], *cosine* [Hua08], *Jaro-Winkler* [GES14] and so on for computing the similarity between input pattern, with the patterns generated during data collection for a property. And then combine the score with embedding similarity 5.4.4 to capture the semantics. This approach will work for relation extraction but is not the best way of comparing similarity, as abundant patterns from different properties might result in approximately same similarity due to pattern generalization. It only compares the structure of two patterns, ignoring the semantics that is shared between them. As the embedding similarity is the score with respect to the property embedding model and not the semantic similarity between two patterns.

Alternative Approaches

Our pattern is a narrow and specific representation of sub-semantic graph that is in a tree format having leaf nodes as entities. One possible way to calculate the similarity between two trees is to compute the number of steps (node-edge removal/addition) needed to perform for making both trees similar. But doing this will still not allow us to compare semantics unless we use word embeddings. Moreover, making two trees similar is not an efficient way for semantic similarity as the words could still be the same, but in a different tense (which will also change the typed dependency edge).

Using Tree-LSTM [TSM15] is another alternative approach that calculates the similarity between two trees along with their semantics. This approach also relies on word embedding model for determining semantics but needs to be trained due to the artificial recurrent neural network architecture of LSTM (long short-term memory). We devised a new algorithm *Forskipo-Semantic Similarity (FSS)* to capture the similarity of patterns along with their semantics which doesn't require any training and can be used with existing word embedding models.

Forskipo-Semantic Similarity (FSS)

The algorithm uses an approach to formulate sequences from a given pattern, and align it with respect to a comparison sequence. The comparison sequence belongs to another pattern, making it possible for patterns to be compared. Sequences are the path from root node to the subject seq_{subj} and object seq_{obj}. Means for each pattern, two sequences are generated. Our aim is to determine semantic similarity between input pattern $pt_{in} \in generatePattern(InputDocument)$, and comparison pattern $pt_{comp} \in getPatternForProperty(p) : p \in CP$.

$$ps(pt_i, pt_{comp}) = \frac{FSS(seq_{in_{subj}}, seq_{comp_{subj}}) + FSS(seq_{in_{obj}}, seq_{comp_{obj}})}{2} \tag{5.17}$$

Matrix Formation Since every pattern provides us with two sequences of subject and object respectively, therefore input patterns can be compared with pre-generated patterns. The input pattern's subject sequence $seq_{in_{subj}}$ is compared with the subject sequence of comparison pattern $seq_{comp_{subj}}$. Likewise, the object's sequences of both patterns do not have any impact of the subject's sequences on similarity. Hence, two matrices are formulated for their respective comparisons. These matrix representations are then used to calculate the semantic similarity between each comparison node.

We align the sequences using matrix by defining the larger sequence amongst the two over column j in matrix space, whereas another sequence is specified with row i space in that matrix

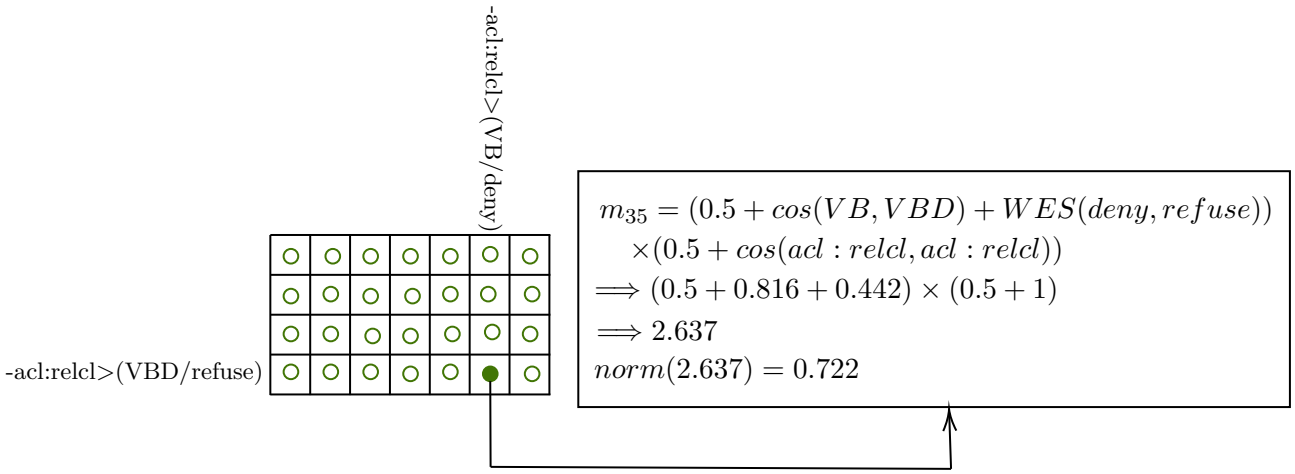

Figure 5.4: Matrix Block Calculation

m. This representation allows us to compare patterns even if the number of nodes varies since the size of the column/row is equivalent to the number of nodes (POS) present in it. Each matrix block posses a real number value reflecting the semantic and grammatical similarity between each node. The next paragraph highlights on how similarity is calculated.

Matrix Population Each block m_{ij} of the matrix posses combined calculated value using POS, typed dependency, and semantic similarity of node labels. Part-of-speech (POS) in Stanford CoreNLP is designed in such a way that their POS tags are already categorized[6] with respect to their surface tag. For example, {noun-plural (*NNS*), noun-proper-singular (*NNP*), noun-proper-plural (*NNPS*)} shares the surface form of {noun-singular (*NN*)}. This categorization proves a lot beneficial and allows us to use string similarity matrices for its comparison. We transform our tags into vector so that cosine similarity can be applied. The same categorization is also implemented over typed dependencies.

Since the patterns are based on out-edges from the root node, therefore it is always safe to assume that each node in our sequence has an incoming typed dependency edge. Our similarity is a combinational approach where the sum of POS cosine similarity and label's embedding similarity are multiplied with an immediate cosine similarity score of typed dependency. The following equation is used to populate each matrix block m_{ij}.

$$m_{ij} = \Big\{0.5 + cos(POS_{seq_{1j}}, POS_{seq_{2i}}) + WESim(label(seq_{1j}), label(seq_{2i}))\Big\} \times \Big\{0.5 + cos(typD_{seq_{1j}}, typD_{seq_{2i}})\Big\} \tag{5.18}$$

Where *WESim* is calculated using word embedding model (Word2Vec/Glove/FastText). The additional score of 0.5 on both sides helps to avoid having zero similarity if any of the sub-formula has no similarity at all. We then normalize each matrix value between $(0, 1)$ by using the highest and lowest possible values of each sub-formula. Figure 5.4 represents the subpart of a pattern, and how similarity is calculated after parsing it.

[6] `https://www.clips.uantwerpen.be/pages/mbsp-tags`

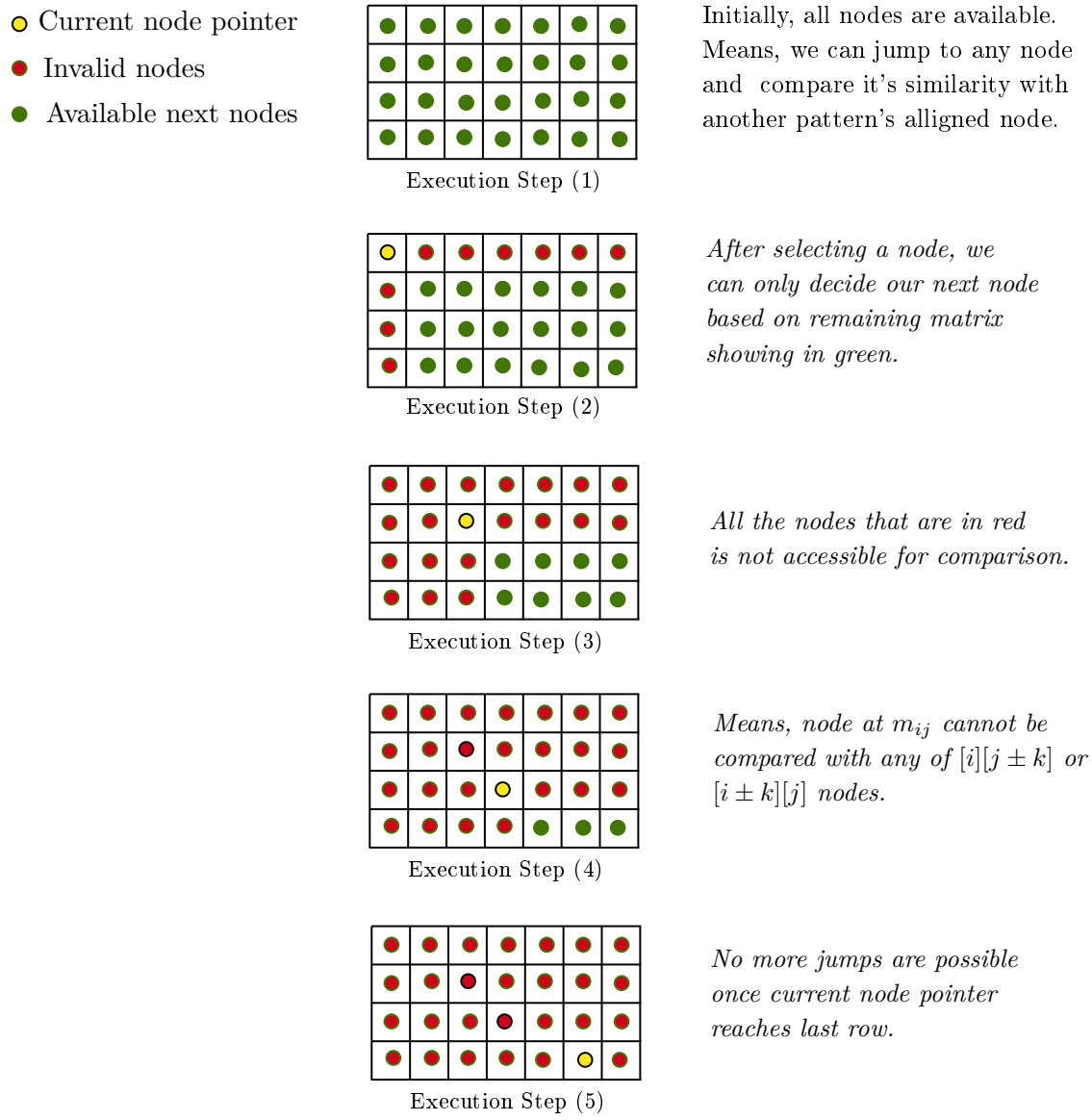

Figure 5.5: Forward skipping sample execution

Forward Skipping Once we have populated our similarity values, now our task is to determine the combination of values that results in maximum similarity. We cannot simply select the maximum value from each row and sum it to obtain the maximum score. Doing this would impair the semantic subgraph structure. To retain the graph structure and calculate maximum similarity, we created some rules and named the approach *Forward skipping*. Figure 5.5 shows the possible execution steps of FSS.

The selection of next matrix-block state is dependent on the current block position (current node pointer - CNP); representing the nodes that were compared at last. Being on a current comparison node, next node possibility is the sub-matrix starting from its immediate diagonal position, that is, if CNP is at $m[i][j]$, then the sub-matrix will start from position $m[i+1][j+1]$. Nodes from the similar row i or column j are never available for the next comparison, since comparing these nodes will form the *sibling* relationship between them and impair the semantic subgraph. Likewise, we can never select the nodes for comparison from previous rows or columns.

Doing this reverses the edge direction resulting in either cyclic graph, or changing *in-edges* to *out-edges*.

Initially we do not have any current node pointer, therefore we can pick any matrix block as a starting point. It is not true to assume that the element at $m[0][0]$ will always lead us to maximum similarity. Consider a matrix where $m[0][j]$ has a similarity score, higher than combined values of other matrix blocks (bounded by our rules). In such a case, we would consider the similarity of $m[0][j]$ and return the score after normalization.

Since we have to figure out a combination that will provide us the maximum score, and the fact that any next block can be selected from the sub-matrix makes the problem a lot complex. One possible solution is to create another matrix $sumM$ having summation score values for each block m_{ij} (the maximum score that can be attained starting from m_{ij}). We used a recursive approach and designed algorithm 3 to populate our summation matrix.

Root Similarity The similarity matrix we computed earlier represented the path from the root node to its leaf node (entity). Since root has no incoming edge, therefore we need to calculate the root similarity $rootSim$ of two patterns explicitly. We make use of POS and WES sub-formula for calculating it, and normalize the score accordingly within $(0, 1)$. We also need to determine the maximum value $maxSim$ from our $sumM$ matrix and combine it with $rootSim$ to get the final similarity score of the two comparing sequences.

$$FSS(seq_j, seq_i) = \frac{rootSim + maxSim}{i + 1} \tag{5.19}$$

5.4.4 Embedding Similarity

After that we have calculated the pattern similarity, we need to determine how much the pattern is semantically related with respect to our candidate property. For this purpose, we make use of the nouns, verbs, and adjectives identified while traversing semantic subgraph and compare these words against our generated property embedding model (PEM). Instead of comparing each word (and to be consistent), we lookup the vector representation of each word from our source embedding model (SEM). SEM is the model through which PEM was generated. For example, if PEM was generated using word2vec, then we extract the vector representation of pattern-words from word2vec. We then calculate the mean vector v_{pmean} of pattern words and determine it's cosine similarity from PEM against each candidate property vector v_{cp_i}. Let pattern-words $PW = \{w_1, w_2, ..., w_n\}$, then embedding similarity can be calculated using:

$$ems(v_{pmean}, cp_i) = getSimilarity(v_{pmean}, v_{cp_i}) : v_{pmean} = [\frac{\sum_{k=1}^{n} v_{k1}}{n}, \frac{\sum_{k=1}^{n} v_{k2}}{n}, ..., \frac{\sum_{k=1}^{n} v_{kd}}{n}] \tag{5.20}$$

5.4.5 Parameter Variation

We used two parameters of α and β in our confidence calculation and ranged the values from *0.1 to 0.9* to figure out the parameter-values combination under which it performs the relation extraction task best, and gives highest F-measure.

Algorithm 3: Generating summation matrix having each block representation of maximum similarity path

```
   input  : Similarity Matrix simM[][]
   output: Similarity Summation Matrix sumM[][]
 1 double[][] sumM := new double[simM.length][simM[0].length];
 2 for int i := 0; i < simM.length; i++ do
 3 |   for int j := 0; j < simM[i].length; j++ do
 4 |   |   sumM[i][j] := getMaxSumSubMatrix(i, j, 0, -1);
 5 |   end
 6 end
 7 Function getMaxSumSubMatrix(int row, int col, double[][] simM, double sum, double
     max):
 8 |   if row = simM.length ∨ col = simM[0].length - 1 then
 9 |   |   double localMax := -1;
10 |   |   for int j := col; j < simM[0].length; j++ do
11 |   |   |   if simM[rowI][j] > localMax then
12 |   |   |   |   localMax := simM[rowI][j];
13 |   |   |   end
14 |   |   end
15 |   |   double localSum := sum + localMax;
16 |   |   if localMax > max then
17 |   |   |   max := localSum;
18 |   |   end
19 |   |   return max;
20 |   end
21 |   for int i := row + 1; i < simM.length; i++ do
22 |   |   for int j := col + 1; j < simM[0].length; j++ do
23 |   |   |   double localSum :=
24 |   |   |   getMaxSumSubMatrix(i, j, simM, sum + simM[row][col], max);
25 |   |   |   if localMax > max then
26 |   |   |   |   max := localSum;
27 |   |   |   end
28 |   |   end
29 |   end
30 return max;
```

The factor of embedding similarity has no impact of α and is only affected by β. Higher β value gives more priority to the semantics of words in a pattern with respect to PEM, where as α is responsible to saturate the effect of patterns with respect to candidate property, and other properties that share the same pattern. We finally use these parameters along with the features mentioned above to calculate our confidence value.

5.4.6 Calculating Confidence Value

Given a candidate property cp_i, a comparison pattern pt (belonging to the PT_i bag consisting of all the patterns that were stored during data collection), and pattern pt_{in} generated from input sentence, then confidence for pt_{in} against cp_i can be calculated by:

$$\begin{aligned} \varrho(pt_{in}, cp_i) = \{\alpha(sup(pt_{in}, cp_i)) + (1-\alpha)(spc(pt_{in}, cp_i))\} \times ps(pt_{in}, pt) \\ +\beta(ems(v_{pt_{iMean}}, cp_i)) \end{aligned} \tag{5.21}$$

Where α, β, and pt are the values from it's execution environment.

5.4.7 Environment for Confidence

For a given value of α and β, and a set of our candidate properties CP; we calculate the similarity of each pattern pt_{cp_i} present in property $cp_i \in CP$ against our input pattern pt_{in}. We keep track of confidence values and replace it with another value if a greater value is found during iteration. Finally, the candidate property which resulted in maximum confidence score is classified as the property for our input pattern.

Algorithm 4: Pattern Property Decision

input : Parameter α, Parameter β, Input Pattern pt_{in}, Candidate Properties CP
output: Property pt

```
1  double maxConfidence := Double.MIN;
2  foreach Property cp : CP do
3      double PEMSimilarity := patternPropertynEmbeddingSimilarity(pt_in, cp);
4      List<Pattern> trainPatterns := getCollectedPatterns(pt_cp);
5      foreach Pattern pt_cp : trainPatterns do
6          double score := getConfidence(α, β, pt_in, pt_cp, PEMSimilarity);
7          if score ≥ maxConfidence then
8              maxConfidence := score;
9              pt := cp;
10         end
11     end
12 end
13 return pt;
```

6

Experimental Setup And Usage

Since we are comparing the input patterns against our pre-generated patterns for a property, therefore, we will also divide this chapter into the sections of preprocessing, embedding setup, and preliminary requirements. An optional demo setup is also described.

6.1 Preprocessing

If a user wishes to train extra properties, then the following subsections describe the necessary requirements. Otherwise, to work with existing properties, the user can skip to requirements mentioned in next subsections.

6.1.1 Triples and Sentence Selection

Our system RAPID currently works on 29 DBpedia ontologies A. Instead of selecting random triples for a property, we collected 1000 triples from DBpedia [1] based on the subject's Wikipedia' page rank[2]. These triples were then used for collecting sentences (using elasticsearch 6.4.0 [3]) described in section 5.1.3. Sentences extracted were then annotated using Stanford CoreNLP 3.8.0 having pipeline parameters of *{tokenize, ssplit, pos, lemma, ner, parse, mention, coref}*. All the resultant information is saved to our RAPID database based on MySQL 5.1.39.

6.1.2 Patterns

The pattern generation uses Stanford CoreNLP for parsing and creation of semantic graphs. We do not use the previously mentioned pipeline since *mention*-argument makes the computation extensive, and we do not need it in pattern generation step. We create two different pipelines, one for parsing - having arguments *{tokenize, ssplit, pos, lemma, ner, parse}*, and another pipeline serves the purpose of POS tagging and lemmatization - having arguments *{tokenize, ssplit, pos, lemma, ner}*. We use WordNet 3.0 for determining verb against a noun using approach described in section 5.2.2.

[1] `http://dbpedia.org`
[2] `http://people.aifb.kit.edu/ath/#DBpedia_PageRank`
[3] `https://www.elastic.co/products/elasticsearch`

6.1.3 Features for Confidence

To pre-compute the statistics of support and specificity, we formulated a MySQL query that can do the work for us. Using the query in listing 6.1 creates a new table *pattern_stats* and populate it with the required data needed for confidence.

Listing 6.1: Query used for populating *pattern_stats* table with confidence feature statistics.

```
1 INSERT INTO pattern_stats (pattern, prop_uri, sg_pretty,
2 support, specificity, occur_prop, occur_pattern_freq)
3 SELECT pp.pattern, prop.prop_uri,
4 (SELECT pa.sg_pretty FROM property_pattern pa
5 WHERE pa.prop_uri = pp.prop_uri
6 AND pa.pattern = pp.pattern
7 LIMIT 1) AS sg_pretty,
8 COUNT(pp.pattern) AS support,
9 (SELECT COUNT(ps.pattern) FROM property_pattern ps
10 WHERE ps.prop_uri != pp.prop_uri
11 AND ps.pattern = pp.pattern) AS specificity,
12 (SELECT COUNT(DISTINCT(ppa.prop_uri)) FROM property_pattern ppa
13 WHERE ppa.prop_uri != pp.prop_uri
14 AND ppa.pattern = pp.pattern) AS occur_prop,
15 (SELECT COUNT(id_prop_pattern) FROM property_pattern sp
16 WHERE sp.prop_uri IN (SELECT DISTINCT(prop_uri)
17 FROM property_pattern dp
18 WHERE dp.pattern = pp.pattern)
19 AND sp.prop_uri != pp.prop_uri) AS occur_pattern_freq
20 FROM property prop
21 INNER JOIN property_pattern pp ON pp.prop_uri = prop.prop_uri
22 GROUP BY pattern, pp.prop_uri
23 ORDER BY support DESC;
```

6.2 Embedding Setup

Since embedding relies on the surface form of the word, therefore embedding also requires CoreNLP; and the vocabulary expansion was done using WordNet. Generating a Property Embedding Model requires a source model. For this purpose, we used three different embedding models of Word2Vec[4], Glove[5], and fastText[6]. Hence, generating three PEM using each source model.

6.3 Preliminary Requirements

MySQL To run RAPID, a user should have *RAPID* database which contains all the information regarding pre-generated patterns against a property. The database can be loaded using MySQL 5.1.39[7].

[4] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/
[5] http://nlp.stanford.edu/data/glove.840B.300d.zip
[6] https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip
[7] https://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.39

Source Embedding Models If a user decides to choose our PEM that was generated using Word2Vec, then it is required to have the same source embedding model (i.e. Word2Vec); as the mean vector for input-sentence-pattern would be calculated using this source model, and then later classified against respective PEM.

CoreNLP Furthermore, CoreNLP 3.8.0[8] - along with *{Default, English, kbp}*[9] models should be used for being consistent with our database. It is not a mandatory requirement to use exact same models and library version as far as it supports all the features being used by RAPID.

WordNet Along with using a dictionary from WordNet 3.0[10], we made use of JWI 2.4.0[11] library for accessing it. Different versions from the same library can be used as far as it has supporting features being utilized by RAPID. But, a user cannot benefit from an alternative library (*like Rita*) for accessing WordNet; due to the difference in class implementations.

6.4 Demo

After everything has been set up for RAPID to work, it can directly be executed as a java program - providing the context/sentence(s) a user wish to extract relations for. Another option is to use the web service that we created.

REST API The REST API uses Apache Tomcat 8.0[12] and is based on spring boot 2.0.0[13]. A request can be sent to the following URL:

```
http://danish-thesis.cs.upb.de:8181/rapid/api/re/
```

The request requires *context, alpha, beta, embeddingClassifier* headers to work properly. REST API doesn't cover all the features present in RAPID. A user cannot specify entities explicitly; means entities will be identified using CoreNLP only. But the web service can later be extended to integrate external entities, as the feature is completely supported by RAPID.

Web-App Web-App allows the user with a non-technical background to benefit from our system. The demo is based on NodeJs 8.10.0[14]. Executing the node server will get the demo live. But, in order for RAPID to work properly, it is a prerequisite to have REST API running. Following 6.1 is the screenshot taken from RAPID Web-App showing relations extracted from random Input sentences.

[8] https://mvnrepository.com/artifact/edu.stanford.nlp/stanford-corenlp/3.8.0
[9] https://stanfordnlp.github.io/CoreNLP/download.html
[10] https://wordnet.princeton.edu/download
[11] https://projects.csail.mit.edu/jwi/
[12] https://tomcat.apache.org/download-80.cgi
[13] https://mvnrepository.com/artifact/org.springframework.boot
[14] https://nodejs.org/en/download/releases/

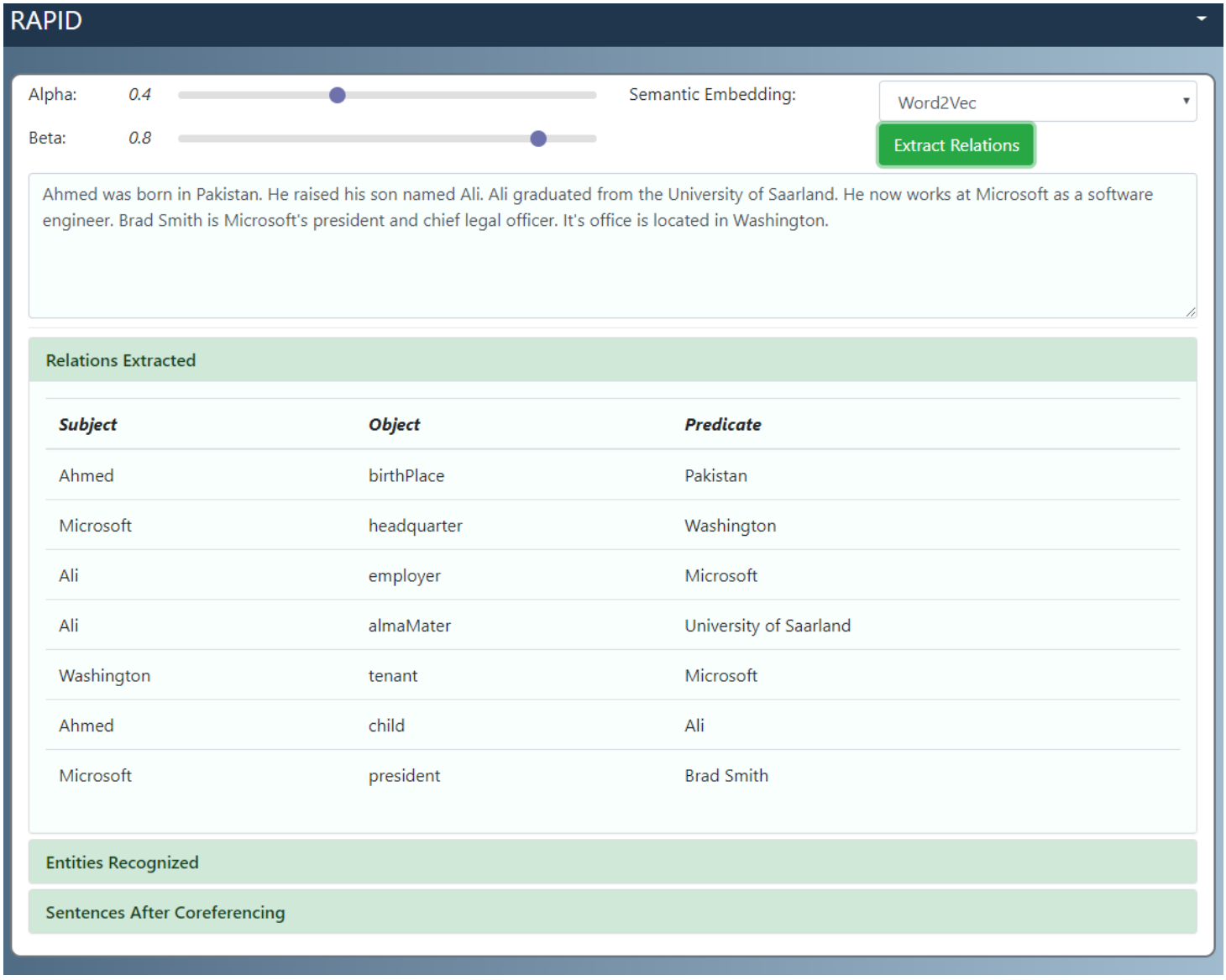

Figure 6.1: RAPID Web-App Demo

7 Evaluation

The evaluation of RAPID is based on 29 DBpedia ontologies P that were used for data collection and pre-generation of patterns. We used OKE train dataset[1] of 96 turtle files having RDF data possessing triples X with predicates $p \in P$. Each file can have one or more sentences that reflect some relational properties between the resource entities. RAPID was able to secure a maximum f-measure of 0.4378 without filtering false positive results; and achieved 0.5698 F-measure after applying the empirical threshold strategy. The evaluation was performed using source embedding models of Word2Vec, Glove, and fastText with their respective property embedding models PEM. Word2Vec outperformed all other models. Detailed evaluation results can be viewed at our open-source GitHub repository [2].

7.1 Results

This section highlights on how varying α and β parameters impact on precision and recall. We evaluated using three embedding source models along with our generated PEM, and calculated the f-measure on 81 (α, β) combinations in total.

7.1.1 Word2Vec

Word2Vec outperformed other embedding models of Glove and fastText and gave a maximum f-measure of 0.4378. The highest f-measure is obtained by using parameters $\{(0.4, 0.7), (0.4, 0.8), (0.5, 0.9)\}$. Figure 7.1 represents the f-measure obtained against each parameter combination.

7.1.2 Glove

A maximum f-measure of 0.3867 was achieved using Glove as source model and PEM generated using Glove. It performed 11.67% inferior to word2vec but 1.35% better than FastText. The maximum f-measure was obtained using $\alpha = 0.7$ and $\beta = 0.5$. Figure 7.1.2 shows the f-measure comparison using different α and β variants.

7.1.3 FastText

Using fastText as source embedding model did not give credible results and had the lowest f-measure score. We got a maximum f-measure of 0.3814 which is 1.35% lower than Glove, and

[1] `https://project-hobbit.eu/challenges/oke2018-challenge-eswc-2018`
[2] `https://github.com/danishahmed92/RAPID/tree/master/evaluation`

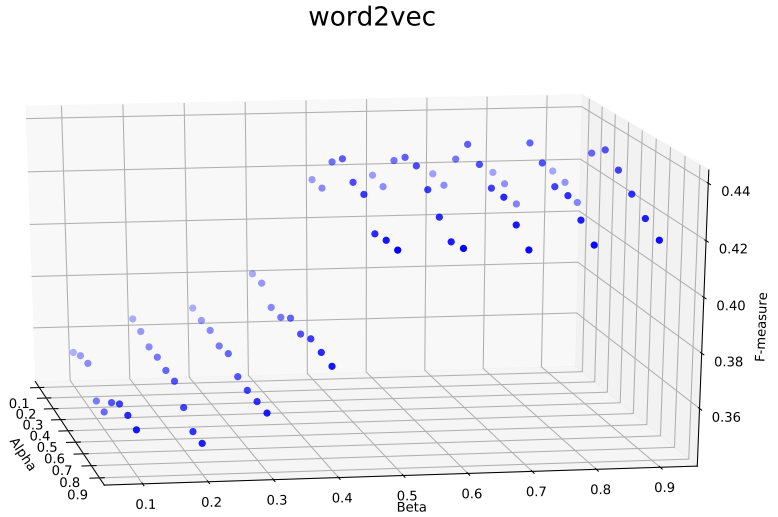

Figure 7.1: F-measure obtained using word2vec model by varying α and β.

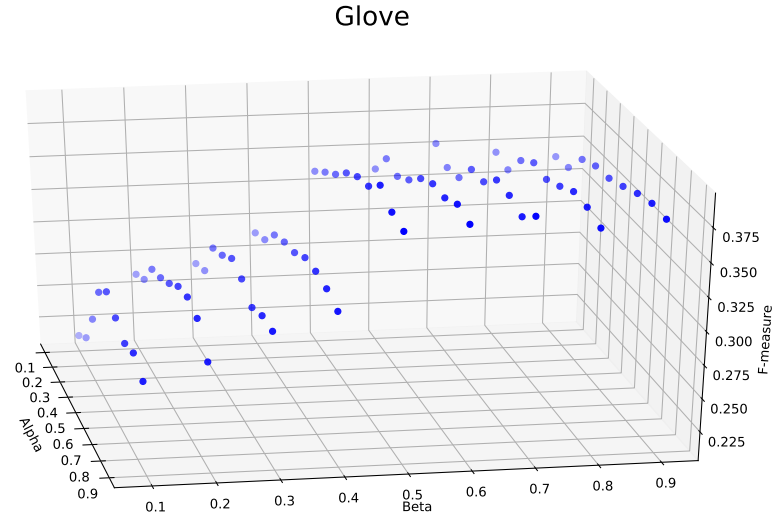

Figure 7.2: F-measure obtained using Glove model by varying α and β.

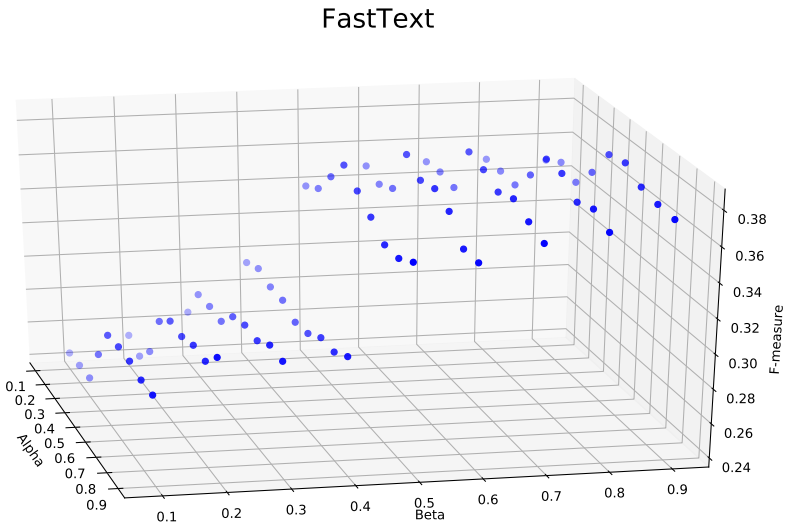

Figure 7.3: F-measure obtained using FastText model by varying α and β.

12.86% lower than Word2Vec embedding model. The combination of α and β parameters at 0.6 and 0.9 respectively gave maximum f-measure. Figure 7.1.3 shows the impact on f-measure by varying α and β parameters using FastText.

7.2 Discussion

We have a higher recall than precision. The reason behind low precision is a fair amount of false positive results that RAPID predicts; since it always returns the property with maximum confidence, no matter how lower the score is. Whereas high recall is due to the coreference resolution, that allows RAPID to return most of the relevant results.

It is quite visible from the results discussed in section 7.1, and the graph in figure 7.4 that all the parameter combinations having β less than 0.5 did not give convincing results, and a huge f-measure difference can be seen where $\beta \geq 0.5$. The graphs also prove that our property embedding model (PEM) that makes use of synonyms, and hyponyms; is a great contribution towards resolving relations since PEM is dependent on β while calculating confidence.

Having multiple parameter combinations resulting in similar f-measure indicates that the properties can be parameter dependent, and can perform better with specific combinations. Section 7.4 briefly describes about property parameter dependency.

A question might rise up that our confidence equation 5.21 has no impact of β on pattern statistics sub-calculation. The equation has intentionally kept this way to allow α as a boosting parameter for existing patterns in RAPID, and improve the confidence score by including more pre-generated patterns to RAPID in future. While β solely serves the purpose to enhance the semantic feature (with respect to property) of RAPID.

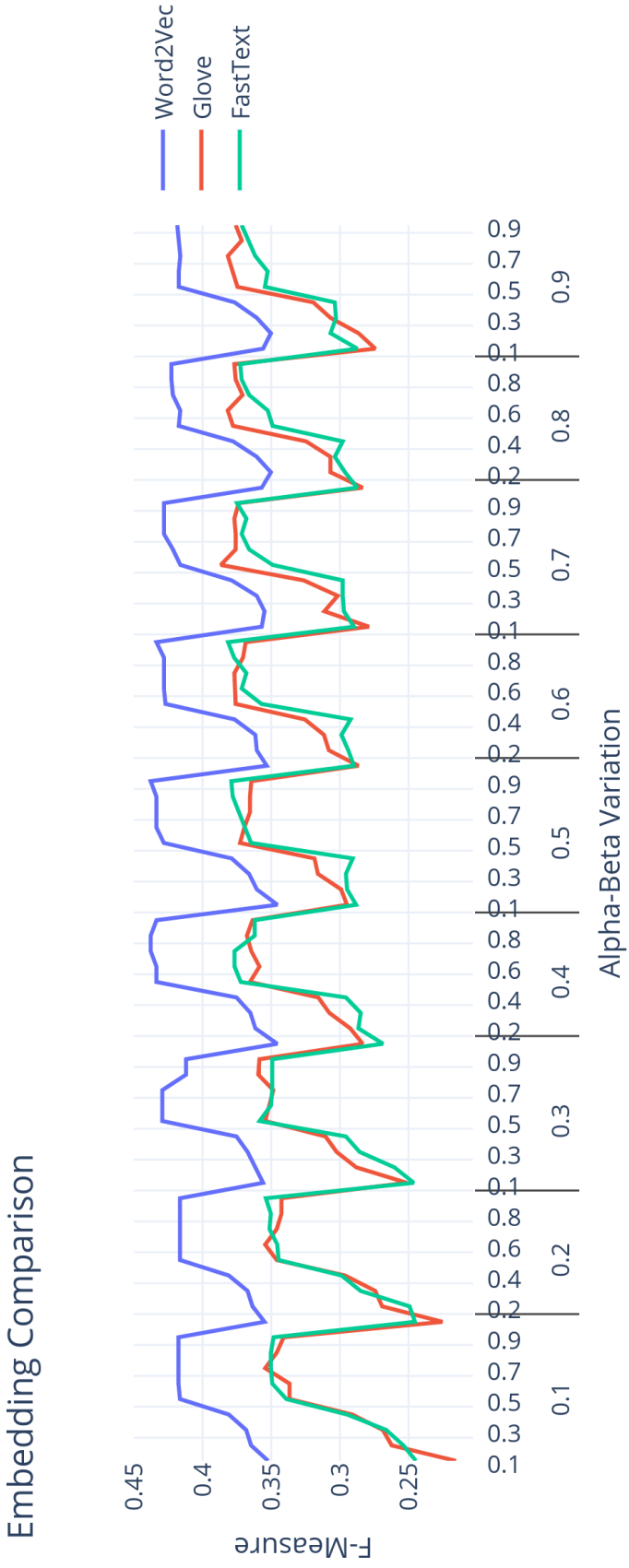

Figure 7.4: A comparison of different embedding model, and f-measure achieved by varying α and β parameters.

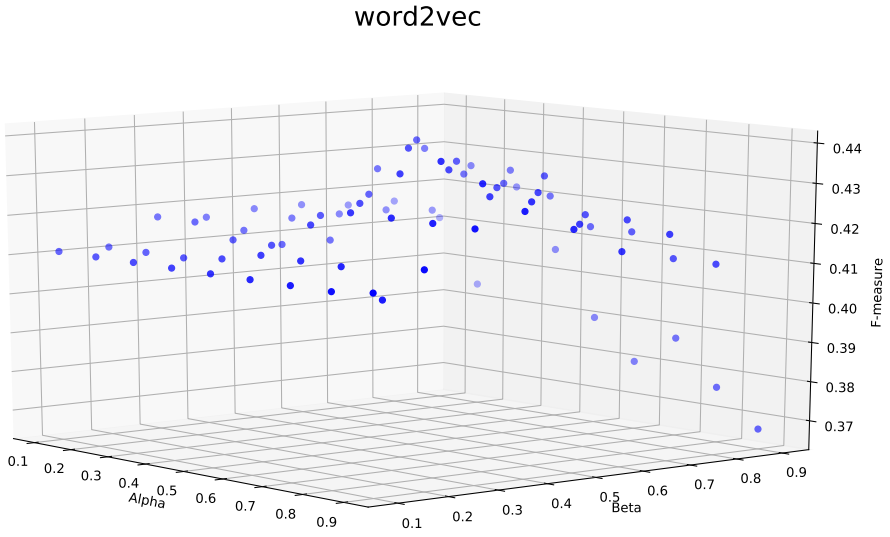

Figure 7.5: F-measure obtained using Word2Vec model by varying α and β; having β dependency on pattern statistics subcalculation.

We also modified the confidence equation with the following and performed evaluation again to determine the parameter influence.

$$\begin{aligned} \varrho(pt_{in}, cp_i) = \beta\left\{\alpha(sup(pt_{in}, cp_i)) + (1-\alpha)(spc(pt_{in}, cp_i))\right\} \times ps(pt_i, pt) \\ +(1-\beta)(ems(v_{pt_{iMean}}, cp_i)) \end{aligned} \tag{7.1}$$

Figure 7.5 shows the f-measure results obtained using equation 7.1. The graph can be seen as normal distribution where the peak is at $(0.5, 0.5)$, and all other combinations below and above will result in a lower f-measure. Increasing the value of α limits the effect of β; and the other way around. Making complete equation dependent on all parameters will nullify their effect, and will always provide a maximum score at the middle value of parameters. Therefore, it is better to keep parameters independent in sub-calculation and let parameters act as boosting feature.

7.3 Empirical Threshold

The sentences that we extracted from Wikipedia against a triple (s, p, o) is always assumed to reflect property p [DGH+14]. But it is not necessarily true, since properties that share the same subject s, and object o would have common sentences during sentence collection step. Doing manual filtration of sentences will reduce RAPID's scalability (as our aim is to target all DBpedia ontologies), and would require a lot of computational resources for calculating the threshold using every pre-generated patterns.

The sentences were collected using Wikipedia articles whereas the evaluation was performed on sentence(s) that belongs to some news articles / blogs [SRC+18]; proves the corpus independency of RAPID. Therefore, the confidence obtained for a property will have a similar score range; that is based on α and β parameters. Learning thresholds directly from evaluation statistics

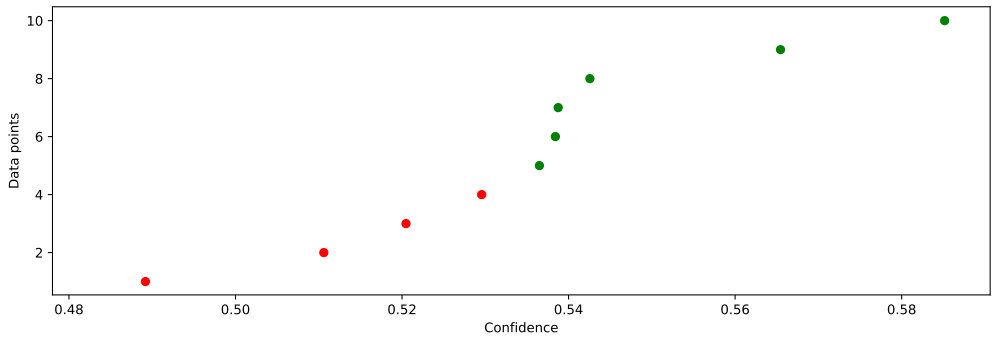

Figure 7.6: True positive and false positive values obtained after evaluation for property *birthPlace*.

allows us to use fewer computation resources, and have an adaptive threshold mechanism that will support the inclusion of new properties into the RAPID system.

7.3.1 True Positive and False Positive Clustering

Every property has a different vector representation in property embedding model (PEM), therefore we cannot use the same threshold value for every property. We used the values of true positive tp, and false positive fp from our evaluation results to compute threshold θ_p for each property p. The following graph reflects tp and fp values obtained using $\alpha(0.4)$ and $\beta(0.8)$ for *birthPlace* ontology.

It is prominent that the confidence values for false positive $fp \in FP$ are concentrated in a cluster that is distant from true positive values $tp \in TP$. Same goes with all other properties, and a margin can be easily drawn between two clusters of TP and FP. Since RAPID is not a perfect system, therefore a few true positive results can be eliminated due to having a confidence value less than our decided threshold.

7.3.2 Threshold Decision

A large number of data points will be residing in their correct clusters (TP or FP), but there would a few data points that have a lower or higher value than the cluster mean. We cannot simply neglect these outliers by taking mean μ of both clusters and deciding their average as our threshold. Using variance σ^2 would help us to decide the threshold accordingly. We calculated the variance from all data points of TP and FP separately and allowed a tolerance of variance from mean in case of TP, whereas we increased the false positive cap by adding variance to the mean of FP. This provides us two marginal values from each cluster which are then aggregated to determine the threshold value.

$$\theta = \frac{(\mu_{FP} + \sigma^2_{FP}) + (\mu_{TP} - \sigma^2_{TP})}{2} \tag{7.2}$$

The threshold for data points present in figure 7.6 can be calculated as shown in table 7.1.

$Margin(FP) = \mu_{FP} + \sigma^2_{FP}$	$Margin(TP) = \mu_{TP} - \sigma^2_{TP}$
$\implies (2.049839/4) + 0.00022546$	$\implies (3.306841/4) - 0.00032762$
$\implies 0.51245975 + 0.00022546$	$\implies 0.55114016 - 0.00032762$
$\implies 0.51268521$	$\implies 0.55048492$
Threshold θ	$(0.51268521 + 0.55048492)/2 \implies 0.531585065$

Table 7.1: Calculation of empirical threshold based on figure 7.6

7.3.3 Revaluation With Threshold

Our approach always gives us a resultant property based on the maximum confidence obtained; no matter how lower the score is. This leads us to have a lot of false positive resultant properties. Having a threshold will allow us to eliminate those results and increase our precision. But since RAPID emphasis more on semantics, therefore it could be true to have some of the actual properties eliminated (after applying threshold) due to lower confidence value. We implemented a threshold strategy over the results obtained using Word2Vec source model.

7.3.4 Word2Vec

Figure 7.7 shows the difference of F-measure before and after applying the threshold on results obtained using Word2Vec source model.

Having a threshold gives RAPID approximately 1.3× better results by eliminating false positive properties that are below the specified threshold.

7.4 Property Parameter Dependency

To determine if the resultant properties are dependent on parameters or not, we need to look at the evaluation statistics of *true positive*, *false positive*, and *false negative* against a property p. Calculating f-measure already provided us the desired values. We executed the following MySQL query 7.1 on RAPID database to analyze the property parameter dependency.

Listing 7.1: Query to analyze property parameter dependency

```
1 SELECT DISTINCT subquery.prop_uri,
2 subquery.alpha,
3 subquery.beta,
4 (subquery.tpCount
5 - (IFNULL(subquery.fnCount, 0))
6 - subquery.fpCount) AS stats
7 FROM (SELECT DISTINCT tp.prop_uri,
8 tp.alpha, tp.beta,
9 (SELECT COUNT(stp.id_tp) FROM oke_prop_true_positive stp
10 WHERE stp.prop_uri = tp.prop_uri
11 AND stp.alpha = tp.alpha
12 AND stp.beta = tp.beta) AS tpCount,
13 (SELECT COUNT(sfp.id_fp) FROM oke_prop_false_positive sfp
14 WHERE sfp.prop_uri = tp.prop_uri
15 AND sfp.alpha = tp.alpha
```

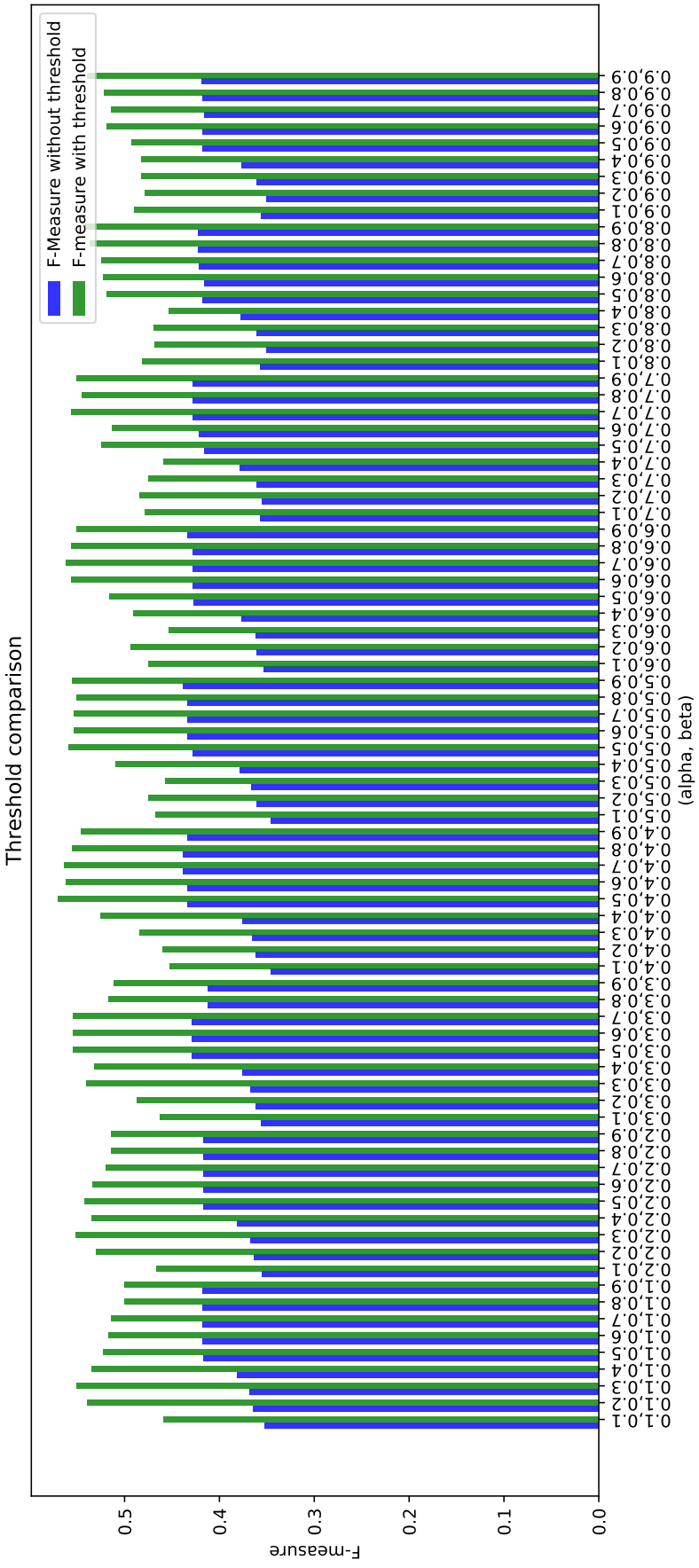

Figure 7.7: F-measure obtained using word2vec by varying α and β and eliminating properties by applying threshold with respect to parameter combination.

```
16 AND sfp.beta = tp.beta) AS fpCount,
17 (SELECT fn_count FROM oke_prop_false_negative sfn
18 WHERE sfn.prop_uri = tp.prop_uri
19 AND sfn.alpha = tp.alpha
20 AND sfn.beta = tp.beta) AS fnCount
21 FROM oke_prop_true_positive tp) AS subquery
22 ORDER BY prop_uri ASC,
23 stats DESC;
```

The query above returns the parameter combination sorted in descending order that gave best results with respect to properties during evaluation. Parameter combination is considered best for a property when it returns maximum number of *true positive* results along with no (or a few) *false positive* and *false negative* results. A huge variation in the results is prominent, which proves that the properties perform best with certain parameter combinations. If the *stats* column from above query would have given almost the same values for a similar property, then the properties would have been independent. The reason for parameter dependency is only due the limited number of pre-generated patterns that affect support and specificity. It is visible from the query results, that β remains consistent for each combination.

7.5 Auto Parameter Adjustment

Since the above query returns all the combinations of parameters for each property, therefore it can be modified with *LIMIT* clause to return only the combination that results in maximum *stats*. Table 7.2 are the best parameter combinations acquired after limiting rows to return 1 combination per property.

DBpedia Ontology	α	β
affiliation	0.9	0.9
almaMater	0.6	0.9
bandMember	0.7	0.9
birthPlace	0.5	0.9
ceo	0.1	0.9
child	0.9	0.9
club	0.9	0.9
deathPlace	0.9	0.4
district	0.9	0.8
doctoralAdvisor	0.9	0.1
employer	0.9	0.2
formerBandMember	0.2	0.9
formerTeam	0.9	0.1
foundationPlace	0.9	0.9
headquarter	0.9	0.9
hometown	0.4	0.5
leaderName	0.9	0.9
locatedInArea	0.1	0.9
nationality	0.8	0.9
parent	0.9	0.6
president	0.9	0.4
spouse	0.9	0.9
subsidiary	0.9	0.9
tenant	0.9	0.9
trainer	0.9	0.9

Table 7.2: Best parameter combination against each property

8 Limitations

In this chapter, we will discuss the limitations of RAPID, which would allow us to view our system from a broader perspective; leading towards future possibilities.

8.1 Property against Pattern

RAPID compares the input-sentence-pattern against every pre-generated patterns for property and uses embedding to avoid semantic drift by the means Forskipo-Semantic Similarity (FSS) and Property Embedding Model (PEM). Due to this comparison approach, the identified property is the one who wins against other properties by having a maximum score. This scoring methodology limits RAPID to return only one property for an input pattern.

On a large scale after training a lot of properties, it is possible that a single pattern can have multiple properties. This case will occur when many properties share the same domain and range. In DBpedia, there are 145 ontologies in total that have same domain and range. SPARQL query in listing 8.1 makes our concern valid.

Listing 8.1: Ontologies sharing same domain and range

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 SELECT distinct ?onto WHERE {
4 ?onto rdfs:domain ?domain .
5 ?onto rdfs:range ?range .
6 FILTER (?domain = ?range) .
7 FILTER (regex(?onto, "ontology" ))
8 }
```

8.2 Limited Pre-generated Patterns

Since we used only 1000 triples for each property, we have a good amount of patterns. But since we are using *support* and *specificity* for calculating confidence, therefore the numbers of pre-generated patterns are still not sufficient enough. 1000 triples for each property was decided due to high time complexity for sentence annotation (as we are using coreference resolution).

8.3 False Sentimental Properties

RAPID extracts the property *employee* for entities *Google* and *James Damore* from the following sentence:

"Google CEO Sundar Pichai responded today to the firing of employee James Damore"

The use of word *firing* expresses that James Damore doesn't serve Google anymore, and is not an employee. But since we are not doing sentiment analysis, therefore, RAPID produces false positive results for such cases.

9 Future Work

RAPID is an entire System with several components. Being a system itself allows RAPID to have features and modules extension possibilities. In this chapter, we will discuss a few improvements that are easier to implement and can make RAPID perform better.

9.1 Multiple Properties per Pattern

Instead of selecting the property that gives a maximum score, one can modify the implementation to store the maximum score achieved against each candidate property. If the resulting score for each property is greater than the threshold, then we can include these properties in our resultant set.

9.2 Term Frequency-Inverse Document Frequency

There is a pretty good amount of pre-generated patterns for each property. These patterns store some useful information of verbs, nouns, and adjectives encountered during semantic subgraph traversing. For each of these words in a property p, one can calculate tf-idf, and utilize these values in our final confidence formula if input-sentence-pattern encounters the same word.

9.3 Pre-generated Pattern Expansion

Even though we generated patterns for comparison using only 1000 triples, but still these are not sufficient. To have a better value for *support* and *specificity*, abundant amount of pre-generated patterns are required. RAPID will perform better if it has more patterns in the RAPID database. *Note: Developer will have to perform all three processes of triple collection, sentence filtration, and annotation.*

9.4 Named Entity Recognition Enhancement

Instead of relying completely on Staford CoreNLP for entity recognition, we made use of our written SPARQL query that returns superclass of input words (containing POS=*NN*) based on label's frequency with respect to resource URI. This query only executes for the words that are not part of entities recognized by CoreNLP. The problem with this approach is; there

could be many words in a sentence containing POS tag *NN*. And since we are always returning result against entities, therefore we have a bunch of false positive results. Which later generate false patterns. Hence, we discarded this implementation and now RAPID relies on Standford CoreNLP for entity recognition, or, external entities can be passed to it (making system flexible, and allowing to solely make evaluations with respect to relation extraction).

A possibility is to couple Stanford CoreNLP with DBpedia Spotlight [MJGSB11]. All the entities that were missed from CoreNLP and got retrieved by spotlight can be merged into the entity set.

9.5 DBpedia Properties

Since we generated patterns only for 29 properties, the RAPID database can be further extended to generate patterns against all DBpedia properties. Along with this, a property embedding model can be created once the necessary data (at least presence of property's label) for a property is available.

9.6 Sentiment Analysis

As we are already using CoreNLP, the library allows sentiment analysis feature by marking a sentence as *positive*, *negative*, or *neutral*. But to perform sentiment analysis, necessary pipeline attributes should be passed while annotating a document. After generating pattern from input sentence, if it contains the word that causes *negative* sentiment response, then we can skip the pattern instead of classifying it against properties.

9.7 Forskipo-Semantic Similarity Analysis

Thorough and extensive experimenting can be performed on our devised FSS algorithm. And comparison can be made against LSTM. Along with F-measure, execution runtime and performance can also be compared.

9.8 Supervised Learning

Since we have already computed patterns against properties, and tf-idf can be calculated from word gathered through semantic subgraph iteration, therefore one can analyze and extract features from our patterns to train a supervised machine learning model. Moreover, on increasing the number of patterns for a property (abundant patterns for each property), a recurrent neural network can be trained and used for property identification without the need of deciding feature selection.

10 Conclusion

Content in the world of web is massively occupied by unstructured data and hard to be read by machines. Transforming this data into a structured form will not only help us to process our desired information but will also help to originate a graph that is linked across other webs. Our thesis introduces RAPID, a highly scalable and extensible system that aims to bridge this gap by extracting relations from the natural language input text. RAPID uses a combinational approach of pattern generation and property embeddings to avoid semantic drift at its best.

The initial preprocessing involves indexing of WikiDump using elasticsearch and collecting sentences based on triples (from DBpedia) satisfying a property. The harvested sentences go through filtration step to eliminate sentences lacking the presence of both entities (subject and object) after they are transformed using coreference resolution. These sentences are stored in RAPID database and are used for pattern generation.

Patterns are generated using dependency parse tree of generated subgraph created using Dijkstra algorithm between two entity nodes, and applying node filtration and refinement operations. These subgraphs are traversed and organized as generalized and extended patterns. The generalized-patterns serve to assure maximum matching of the generated pattern from input sentence; whereas extended-patterns promises to incorporate semantics along with keeping graph structure intact.

Beside patterns, property embedding models (PEM) are used for scoring and classification of sentences against properties. PEM relies on property's label from DBpedia; and makes use of it to gather synonyms and hyponyms using WordNet. The labels and all the resultant words from WordNet become part of vocabulary V for a property p. The words in V are looked up in the source embedding model (word2vec, glove, or fastText) to retrieve their vectors. We calculate the mean of these vectors to have a single vector that represents our property. PEM is a word embedding having mean vector representation of all the properties that RAPID can classify.

Once we have our pre-generated patterns and property embedding model; RAPID makes use of support and specificity features by comparing generalized-pattern obtained from input sentence, with the pre-generated generalized-patterns. The extended-patterns are utilized by Forskipo-Semantic Similarity (FSS) algorithm where given two patterns (*inputSentencePattern*,

existingPattern) are aligned in a matrix and compared semantically using forward skipping approach. Each matrix block is initialized with a double value representing the semantic similarity of nodes[*pattern1*][*pattern2*]'s attributes. The aim is to determine the overall maximum similarity and retain the graph structure. RAPID uses a recursive approach of forward skipping such that inclusion of matrix-blocks-score will not lead to any siblings, cyclic graph, or reversing edge direction. Support, specificity, and FSS combinedly serves the purpose of pattern statistics (with respect to existing patterns) in our confidence equation; and uses the α parameter to boost it.

Since the pre-generated patterns are derived from sentences based on *local closed world assumption* therefore we also need to compare the semantics of input pattern with respect to the properties. The *domain* and *range* present in input sentence narrows our comparison against properties. We traverse the semantic graph of input pattern and collect context-based word features (verb, nouns, adjectives). These words are looked up in the source embedding model and their vectors are retrieved. The mean μ of these vectors are used to identify the vector from PEM against which μ closely matches. The β parameter is used in our confidence value to give extra priority to property semantics. Property statistics and property semantics, along with parameters help us to determine the similarity of input sentence pattern against candidate properties. The property that gives maximum score is accounted as a result of that input pattern if the score is greater than the empirical threshold.

RAPID is a state of art approach towards relation extraction that contributes FSS algorithm and property embedding model to ensure it's corpus independent execution and promises to avoid semantic drift at its best. Moreover, our pre-generated patterns is a unique data set that can be used to extract pattern-features, and transform RAPID to work with both supervised and unsupervised learning. Also, the developed web service can help our fellow researchers to benefit from RAPID.

Bibliography

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[BBHG15] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *arXiv preprint arXiv:1505.04406*, 2015.

[Ben09] Emily M Bender. Linguistically naïve!= language independent: why nlp needs linguistic typology. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 26–32, 2009.

[BHQR07] Chris Biemann, Gerhard Heyer, Uwe Quasthoff, and Matthias Richter. The leipzig corpora collection-monolingual corpora of standard size. *Proceedings of Corpus Linguistic*, 2007, 2007.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[CG90] Roger Chaffin and Arnold Glass. A comparison of hyponym and synonym decisions. *Journal of Psycholinguistic Research*, 19(4):265–280, 1990.

[CMS07] James R Curran, Tara Murphy, and Bernhard Scholz. Minimising semantic drift with mutual exclusion bootstrapping. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, volume 6, pages 172–180. Citeseer, 2007.

[Coh95] William W Cohen. Fast e ective rule induction. In *Twelfth International Conference on Machine Learning*, page 115, 1995.

[CST+00] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[DG13] Manda Sai Divya and Shiv Kumar Goyal. Elasticsearch: An advanced and quick search technique to handle voluminous data. *Compusoft*, 2(6):171, 2013.

[DGH+14] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[Fel10] Christiane Fellbaum. Wordnet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer, 2010.

[FZG11] Oliver Ferschke, Torsten Zesch, and Iryna Gurevych. Wikipedia revision toolkit: Efficiently accessing wikipedia's edit history. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Systems Demonstrations*, HLT '11, pages 97–102, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[GES14] KS Gayathri, Susan Elias, and S Shivashankar. An ontology and pattern clustering approach for activity recognition in smart environments. In *Proceedings of the Third International Conference on Soft Computing for Problem Solving*, pages 833–843. Springer, 2014.

[GN11] Daniel Gerber and A-C Ngonga Ngomo. Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction@ ISWC*, volume 2011, 2011.

[GP10] Suzanne Gaskins and Ruth Paradise. Chapter five: Learning through observation in daily life. *The anthropology of learning in childhood*, 85:85–110, 2010.

[GTHS13] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422. ACM, 2013.

[GW14] Adam Grycner and Gerhard Weikum. Harpy: Hypernyms and alignment of relational paraphrases. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2195–2204, 2014.

[GWP+15] Adam Grycner, Gerhard Weikum, Jay Pujara, James Foulds, and Lise Getoor. Relly: Inferring hypernym relationships between relational phrases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 971–981, 2015.

[Hay94] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[HP17] Nicolas Heist and Heiko Paulheim. Language-agnostic relation extraction from wikipedia abstracts. In *International Semantic Web Conference*, pages 383–399. Springer, 2017.

[Hua08] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, volume 4, pages 9–56, 2008.

[JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[Kon05] Grzegorz Kondrak. N-gram similarity and distance. In *International symposium on string processing and information retrieval*, pages 115–126. Springer, 2005.

[KRPA10] Adam Kilgarriff, Siva Reddy, Jan Pomikálek, and PVS Avinesh. A corpus factory for many languages. In *LREC*, 2010.

[LFL98] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[LIJ+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[LPC+11] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the fifteenth conference on computational natural language learning: Shared task*, pages 28–34. Association for Computational Linguistics, 2011.

[Mac83] J Lachlan Mackenzie. Nominal predicates in a functional grammar of english. *Advances in Functional Grammar*, 11:31, 1983.

[MHG10] Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in action: covers Apache Lucene 3.0*. Manning Publications Co., 2010.

[Mil98] George Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[MJGSB11] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian 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.

[MRS10] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[MSB+14] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[Niv05] Joakim Nivre. Dependency grammar and dependency parsing. *MSI report*, 5133(1959):1–32, 2005.

[NWS12] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. Patty: a taxonomy of relational patterns with semantic types. In *Proceedings of the 2012 Joint*

Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 1135–1145. Association for Computational Linguistics, 2012.

[PRV09] Sven Peyer, Dieter Rautenbach, and Jens Vygen. A generalization of dijkstra's shortest path algorithm with applications to vlsi routing. *Journal of Discrete Algorithms*, 7(4):377–390, 2009.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher 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.

[RJ+99] Ellen Riloff, Rosie Jones, et al. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479, 1999.

[RYM10] Sebastian Riedel, Limin Yao, and Andrew McCallum. Modeling relations and their mentions without labeled text. In *ECML/PKDD*, 2010.

[SKW07] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[SN18] René Speck and Axel-Cyrille Ngomo Ngonga. On extracting relations using distributional semantics and a tree generalization. In *European Knowledge Acquisition Workshop*, pages 424–438. Springer, 2018.

[Soc] Richard Socher. Cs224d lecture 2 - deep learning for natural language processing.

[SRC+18] René Speck, Michael Röder, Felix Conrads, Hyndavi Rebba, Catherine Romiyo, Gurudevi Salakki, Rutuja Suryawanshi, Danish Ahmed, Nikit Srivastava, Mohit Mahajan, and Axel-Cyrille Ngonga Ngomo. Open knowledge extraction challenge 2018. In *Semantic Web Evaluation Challenge*. AKSW, 2018.

[TSM15] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[VKV+06] Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web*, pages 585–594. ACM, 2006.

[WLX+04] Sheng-Tang Wu, Yuefeng Li, Yue Xu, Binh Pham, and Phoebe Chen. Automatic pattern-taxonomy extraction for web mining. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 242–248. IEEE Computer Society, 2004.

[WW08] Fei Wu and Daniel S Weld. Automatically refining the wikipedia infobox ontology. In *Proceedings of the 17th international conference on World Wide Web*, pages 635–644. ACM, 2008.

[Yoi12] Hasebe Yoichiro. Method for using wikipedia as japanese corpus. In *Doshisha studies in language and culture*, pages 9(2):373–403. Springer, 2006-12.

[ZS02] GuoDong Zhou and Jian Su. Named entity recognition using an hmm-based chunk tagger. In *proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 473–480. Association for Computational Linguistics, 2002.

Appendices

A

DBpedia Ontologies Considered

DBpedia Ontologies
affiliation
almaMater
bandMember
birthPlace
ceo
child
club
deathPlace
debutTeam
department
district
doctoralAdvisor
doctoralStudent
employer
formerBandMember
formerTeam
foundationPlace
headquarter
hometown
leaderName
locatedInArea
nationality
parent
president
relative
spouse
subsidiary
tenant
trainer

Table A.1: DBpedia ontologies against which triples were retrieved, and patterns were generated from the collected sentences.

B

Evaluation Results - Word2Vec

α	β	**Precision**	**Recall**	**f-measure**
0.1	0.1	0.22545454545454546	0.8051948051948052	0.35227272727272724
0.1	0.2	0.23443223443223443	0.8205128205128205	0.36467236467236464
0.1	0.3	0.23809523809523808	0.8125	0.3682719546742209
0.1	0.4	0.24817518248175183	0.8192771084337349	0.380952380952381
0.1	0.5	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.1	0.6	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.1	0.7	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.1	0.8	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.1	0.9	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.2	0.1	0.22826086956521738	0.7974683544303798	0.35492957746478876
0.2	0.2	0.23443223443223443	0.810126582278481	0.36363636363636365
0.2	0.3	0.23809523809523808	0.8024691358024691	0.3672316384180791
0.2	0.4	0.24817518248175183	0.8192771084337349	0.380952380952381
0.2	0.5	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.2	0.6	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.2	0.7	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.2	0.8	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.2	0.9	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.3	0.1	0.22743682310469315	0.8181818181818182	0.35593220338983056
0.3	0.2	0.23443223443223443	0.7901234567901234	0.3615819209039548
0.3	0.3	0.23809523809523808	0.8024691358024691	0.3672316384180791
0.3	0.4	0.2454212454212454	0.7976190476190477	0.3753501400560224
0.3	0.5	0.2947761194029851	0.79	0.4293478260869566
0.3	0.6	0.2936802973977695	0.797979797979798	0.42934782608695654
0.3	0.7	0.2936802973977695	0.797979797979798	0.42934782608695654
0.3	0.8	0.2788104089219331	0.7894736842105263	0.4120879120879122
0.3	0.9	0.2788104089219331	0.7894736842105263	0.4120879120879122
0.4	0.1	0.2210144927536232	0.7922077922077922	0.34560906515580736
0.4	0.2	0.2318840579710145	0.8205128205128205	0.36158192090395486
0.4	0.3	0.23809523809523808	0.7831325301204819	0.3651685393258427
0.4	0.4	0.2454212454212454	0.7976190476190477	0.3753501400560224
0.4	0.5	0.29850746268656714	0.7920792079207921	0.4336043360433604

0.4	0.6	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.4	0.7	0.30223880597014924	0.7941176470588235	0.4378378378378378
0.4	0.8	0.30111524163568776	0.801980198019802	0.43783783783783786
0.4	0.9	0.29739776951672864	0.8	0.4336043360433604
0.5	0.1	0.2210144927536232	0.7922077922077922	0.34560906515580736
0.5	0.2	0.23104693140794225	0.8205128205128205	0.36056338028169016
0.5	0.3	0.23636363636363636	0.8125	0.36619718309859156
0.5	0.4	0.25	0.7816091954022989	0.37883008356545966
0.5	0.5	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.5	0.6	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.5	0.7	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.5	0.8	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.5	0.9	0.30223880597014924	0.7941176470588235	0.4378378378378378
0.6	0.1	0.22545454545454546	0.8157894736842105	0.35327635327635326
0.6	0.2	0.23104693140794225	0.8205128205128205	0.36056338028169016
0.6	0.3	0.2318840579710145	0.8205128205128205	0.36158192090395486
0.6	0.4	0.24817518248175183	0.7816091954022989	0.3767313019390582
0.6	0.5	0.2958801498127341	0.7669902912621359	0.42702702702702705
0.6	0.6	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.6	0.7	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.6	0.8	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.6	0.9	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.7	0.1	0.22743682310469315	0.8289473684210527	0.35694050991501414
0.7	0.2	0.22826086956521738	0.7974683544303798	0.35492957746478876
0.7	0.3	0.23104693140794225	0.8205128205128205	0.36056338028169016
0.7	0.4	0.24727272727272728	0.8095238095238095	0.37883008356545966
0.7	0.5	0.2873134328358209	0.7549019607843137	0.4162162162162162
0.7	0.6	0.29213483146067415	0.7572815533980582	0.4216216216216216
0.7	0.7	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.7	0.8	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.7	0.9	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.8	0.1	0.22743682310469315	0.8289473684210527	0.35694050991501414
0.8	0.2	0.2246376811594203	0.7948717948717948	0.3502824858757062
0.8	0.3	0.23104693140794225	0.8205128205128205	0.36056338028169016
0.8	0.4	0.2463768115942029	0.8095238095238095	0.3777777777777778
0.8	0.5	0.2851851851851852	0.7777777777777778	0.41734417344173447
0.8	0.6	0.2873134328358209	0.7549019607843137	0.4162162162162162
0.8	0.7	0.29213483146067415	0.7572815533980582	0.4216216216216216
0.8	0.8	0.291044776119403	0.7722772277227723	0.4227642276422764
0.8	0.9	0.291044776119403	0.7722772277227723	0.4227642276422764
0.9	0.1	0.22743682310469315	0.8181818181818182	0.35593220338983056
0.9	0.2	0.2246376811594203	0.7948717948717948	0.3502824858757062
0.9	0.3	0.23104693140794225	0.8205128205128205	0.36056338028169016
0.9	0.4	0.24548736462093862	0.8095238095238095	0.3767313019390582
0.9	0.5	0.2851851851851852	0.7777777777777778	0.41734417344173447
0.9	0.6	0.2851851851851852	0.7777777777777778	0.41734417344173447
0.9	0.7	0.2873134328358209	0.7549019607843137	0.4162162162162162
0.9	0.8	0.2883895131086142	0.7549019607843137	0.41734417344173436
0.9	0.9	0.2873134328358209	0.77	0.41847826086956524

Table B.1: Evaluation results obtained using Word2Vec as source mode; and PEM generated from Word2Vec.

C

Evaluation Results - Glove

α	β	**Precision**	**Recall**	**f-measure**
0.1	0.1	0.1262135922330097	0.7358490566037735	0.21546961325966849
0.1	0.2	0.15789473684210525	0.7741935483870968	0.2622950819672131
0.1	0.3	0.16393442622950818	0.746268656716418	0.26881720430107525
0.1	0.4	0.17857142857142858	0.7857142857142857	0.291005291005291
0.1	0.5	0.21639344262295082	0.7586206896551724	0.33673469387755106
0.1	0.6	0.2185430463576159	0.7333333333333333	0.33673469387755106
0.1	0.7	0.23333333333333334	0.7368421052631579	0.35443037974683544
0.1	0.8	0.22818791946308725	0.7157894736842105	0.3460559796437659
0.1	0.9	0.22408026755852842	0.7127659574468085	0.34096692111959287
0.2	0.1	0.13355048859934854	0.7192982456140351	0.22527472527472525
0.2	0.2	0.16225165562913907	0.7903225806451613	0.2692307692307692
0.2	0.3	0.16776315789473684	0.75	0.27419354838709675
0.2	0.4	0.18181818181818182	0.8	0.2962962962962963
0.2	0.5	0.22442244224422442	0.7555555555555555	0.34605597964376583
0.2	0.6	0.23255813953488372	0.7446808510638298	0.35443037974683544
0.2	0.7	0.22818791946308725	0.7157894736842105	0.3460559796437659
0.2	0.8	0.22666666666666666	0.7010309278350515	0.34256926952141054
0.2	0.9	0.22666666666666666	0.7010309278350515	0.34256926952141054
0.3	0.1	0.15333333333333332	0.696969696969697	0.25136612021857924
0.3	0.2	0.1760797342192691	0.7910447761194029	0.2880434782608695
0.3	0.3	0.18627450980392157	0.8028169014084507	0.3023872679045093
0.3	0.4	0.19218241042345277	0.8082191780821918	0.31052631578947365
0.3	0.5	0.23178807947019867	0.7526881720430108	0.35443037974683544
0.3	0.6	0.23076923076923078	0.7340425531914894	0.3511450381679389
0.3	0.7	0.2292358803986711	0.7263157894736842	0.34848484848484845
0.3	0.8	0.23509933774834438	0.7634408602150538	0.359493670886076
0.3	0.9	0.23432343234323433	0.7634408602150538	0.35858585858585856
0.4	0.1	0.174496644295302	0.7536231884057971	0.28337874659400547
0.4	0.2	0.18151815181518152	0.7534246575342466	0.29255319148936176
0.4	0.3	0.1901639344262295	0.8055555555555556	0.3076923076923077
0.4	0.4	0.19736842105263158	0.7894736842105263	0.3157894736842105
0.4	0.5	0.24242424242424243	0.7422680412371134	0.36548223350253806

0.4	0.6	0.23666666666666666	0.7395833333333334	0.35858585858585856
0.4	0.7	0.23841059602649006	0.7741935483870968	0.3645569620253164
0.4	0.8	0.24092409240924093	0.776595744680851	0.36775818639798485
0.4	0.9	0.2376237623762376	0.7741935483870968	0.3636363636363636
0.5	0.1	0.18211920529801323	0.7746478873239436	0.29490616621983906
0.5	0.2	0.18688524590163935	0.75	0.2992125984251969
0.5	0.3	0.2	0.7530864197530864	0.3160621761658031
0.5	0.4	0.20132013201320131	0.7625	0.3185378590078329
0.5	0.5	0.24503311258278146	0.7789473684210526	0.37279596977329976
0.5	0.6	0.2425249169435216	0.776595744680851	0.369620253164557
0.5	0.7	0.23841059602649006	0.782608695652174	0.3654822335025381
0.5	0.8	0.23841059602649006	0.782608695652174	0.3654822335025381
0.5	0.9	0.23841059602649006	0.7741935483870968	0.3645569620253164
0.6	0.1	0.17647058823529413	0.7605633802816901	0.2864721485411141
0.6	0.2	0.19218241042345277	0.7763157894736842	0.3080939947780679
0.6	0.3	0.19480519480519481	0.7792207792207793	0.31168831168831174
0.6	0.4	0.20454545454545456	0.7974683544303798	0.32558139534883723
0.6	0.5	0.24834437086092714	0.7731958762886598	0.37593984962406013
0.6	0.6	0.24834437086092714	0.7731958762886598	0.37593984962406013
0.6	0.7	0.24916943521594684	0.7731958762886598	0.37688442211055273
0.6	0.8	0.2425249169435216	0.7849462365591398	0.3705583756345178
0.6	0.9	0.24092409240924093	0.7849462365591398	0.36868686868686873
0.7	0.1	0.17049180327868851	0.7647058823529411	0.27882037533512066
0.7	0.2	0.19218241042345277	0.8194444444444444	0.31134564643799467
0.7	0.3	0.18627450980392157	0.7916666666666666	0.3015873015873016
0.7	0.4	0.20521172638436483	0.7974683544303798	0.32642487046632124
0.7	0.5	0.25	0.8539325842696629	0.38676844783715014
0.7	0.6	0.24834437086092714	0.7731958762886598	0.37593984962406013
0.7	0.7	0.24834437086092714	0.7731958762886598	0.37593984962406013
0.7	0.8	0.24916943521594684	0.7731958762886598	0.37688442211055273
0.7	0.9	0.24584717607973422	0.7789473684210526	0.3737373737373737
0.8	0.1	0.17263843648208468	0.7910447761194029	0.28342245989304815
0.8	0.2	0.18831168831168832	0.8285714285714286	0.3068783068783069
0.8	0.3	0.18892508143322476	0.8169014084507042	0.30687830687830686
0.8	0.4	0.2012987012987013	0.8378378378378378	0.324607329842932
0.8	0.5	0.24752475247524752	0.7978723404255319	0.3778337531486146
0.8	0.6	0.24671052631578946	0.8426966292134831	0.38167938931297707
0.8	0.7	0.24503311258278146	0.7628865979381443	0.3709273182957394
0.8	0.8	0.24834437086092714	0.7731958762886598	0.37593984962406013
0.8	0.9	0.24834437086092714	0.78125	0.37688442211055273
0.9	0.1	0.16612377850162866	0.7846153846153846	0.27419354838709675
0.9	0.2	0.1758957654723127	0.7714285714285715	0.2864721485411141
0.9	0.3	0.18892508143322476	0.8169014084507042	0.30687830687830686
0.9	0.4	0.19934640522875818	0.8026315789473685	0.3193717277486911
0.9	0.5	0.24422442244224424	0.8043478260869565	0.37468354430379747
0.9	0.6	0.24752475247524752	0.7978723404255319	0.3778337531486146
0.9	0.7	0.24671052631578946	0.8426966292134831	0.38167938931297707
0.9	0.8	0.24172185430463577	0.8021978021978022	0.3715012722646311
0.9	0.9	0.24834437086092714	0.7731958762886598	0.37593984962406013

Table C.1: Evaluation results obtained using Glove as source mode; and PEM generated from Glove.

D

Evaluation Results - fastText

α	β	**Precision**	**Recall**	**f-measure**
0.1	0.1	0.14545454545454545	0.7692307692307693	0.24464831804281345
0.1	0.2	0.15272727272727274	0.75	0.2537764350453173
0.1	0.3	0.16071428571428573	0.7758620689655172	0.26627218934911245
0.1	0.4	0.1795774647887324	0.8225806451612904	0.2947976878612717
0.1	0.5	0.2158273381294964	0.7894736842105263	0.3389830508474576
0.1	0.6	0.22382671480144403	0.7948717948717948	0.3492957746478873
0.1	0.7	0.2246376811594203	0.7948717948717948	0.3502824858757062
0.1	0.8	0.2246376811594203	0.7948717948717948	0.3502824858757062
0.1	0.9	0.22302158273381295	0.7948717948717948	0.34831460674157305
0.2	0.1	0.145985401459854	0.7692307692307693	0.24539877300613497
0.2	0.2	0.14909090909090908	0.7592592592592593	0.24924012158054712
0.2	0.3	0.17314487632508835	0.8032786885245902	0.28488372093023256
0.2	0.4	0.18309859154929578	0.8125	0.2988505747126437
0.2	0.5	0.22021660649819494	0.7922077922077922	0.34463276836158196
0.2	0.6	0.2210144927536232	0.7922077922077922	0.34560906515580736
0.2	0.7	0.2246376811594203	0.8051948051948052	0.35127478753541075
0.2	0.8	0.22382671480144403	0.8051948051948052	0.3502824858757062
0.2	0.9	0.22661870503597123	0.8076923076923077	0.35393258426966295
0.3	0.1	0.14705882352941177	0.7547169811320755	0.24615384615384617
0.3	0.2	0.15827338129496402	0.7333333333333333	0.26035502958579876
0.3	0.3	0.17314487632508835	0.8166666666666667	0.28571428571428575
0.3	0.4	0.18021201413427562	0.8225806451612904	0.2956521739130435
0.3	0.5	0.2318840579710145	0.7901234567901234	0.3585434173669468
0.3	0.6	0.22545454545454546	0.7848101265822784	0.35028248587570626
0.3	0.7	0.2246376811594203	0.7848101265822784	0.34929577464788736
0.3	0.8	0.22302158273381295	0.8051948051948052	0.34929577464788736
0.3	0.9	0.22302158273381295	0.8051948051948052	0.34929577464788736
0.4	0.1	0.16129032258064516	0.8035714285714286	0.26865671641791045
0.4	0.2	0.17375886524822695	0.8166666666666667	0.28654970760233917
0.4	0.3	0.17314487632508835	0.8032786885245902	0.28488372093023256
0.4	0.4	0.18021201413427562	0.8225806451612904	0.2956521739130435
0.4	0.5	0.24363636363636362	0.788235294117647	0.37222222222222223

0.4	0.6	0.2463768115942029	0.8	0.3767313019390582
0.4	0.7	0.24548736462093862	0.8095238095238095	0.3767313019390582
0.4	0.8	0.23381294964028776	0.8024691358024691	0.36211699164345407
0.4	0.9	0.23381294964028776	0.8024691358024691	0.36211699164345407
0.5	0.1	0.17253521126760563	0.875	0.2882352941176471
0.5	0.2	0.17894736842105263	0.8360655737704918	0.29479768786127175
0.5	0.3	0.1795774647887324	0.8360655737704918	0.29565217391304344
0.5	0.4	0.17667844522968199	0.819672131147541	0.29069767441860467
0.5	0.5	0.23655913978494625	0.7951807228915663	0.3646408839779006
0.5	0.6	0.24014336917562723	0.7976190476190477	0.3691460055096419
0.5	0.7	0.2446043165467626	0.7906976744186046	0.37362637362637363
0.5	0.8	0.24731182795698925	0.8023255813953488	0.3780821917808219
0.5	0.9	0.24731182795698925	0.8117647058823529	0.37912087912087916
0.6	0.1	0.17525773195876287	0.8360655737704918	0.28977272727272724
0.6	0.2	0.17832167832167833	0.8360655737704918	0.29394812680115273
0.6	0.3	0.18181818181818182	0.8387096774193549	0.2988505747126437
0.6	0.4	0.17770034843205576	0.8225806451612904	0.2922636103151863
0.6	0.5	0.23214285714285715	0.7738095238095238	0.35714285714285715
0.6	0.6	0.24199288256227758	0.8	0.371584699453552
0.6	0.7	0.2392857142857143	0.7976190476190477	0.3681318681318681
0.6	0.8	0.24731182795698925	0.7931034482758621	0.3770491803278688
0.6	0.9	0.25	0.8045977011494253	0.38147138964577654
0.7	0.1	0.17586206896551723	0.8225806451612904	0.28977272727272724
0.7	0.2	0.17993079584775087	0.8524590163934426	0.29714285714285715
0.7	0.3	0.18118466898954705	0.8387096774193549	0.2979942693409742
0.7	0.4	0.18118466898954705	0.8387096774193549	0.2979942693409742
0.7	0.5	0.225	0.7777777777777778	0.3490304709141275
0.7	0.6	0.2392857142857143	0.7790697674418605	0.366120218579235
0.7	0.7	0.24199288256227758	0.8	0.371584699453552
0.7	0.8	0.2392857142857143	0.7976190476190477	0.3681318681318681
0.7	0.9	0.24555160142348753	0.7931034482758621	0.375
0.8	0.1	0.17301038062283736	0.847457627118644	0.28735632183908044
0.8	0.2	0.1793103448275862	0.8524590163934426	0.2962962962962963
0.8	0.3	0.18466898954703834	0.8548387096774194	0.30372492836676224
0.8	0.4	0.18118466898954705	0.8387096774193549	0.2979942693409742
0.8	0.5	0.22580645161290322	0.7682926829268293	0.3490304709141274
0.8	0.6	0.22857142857142856	0.7710843373493976	0.3526170798898072
0.8	0.7	0.2392857142857143	0.7790697674418605	0.366120218579235
0.8	0.8	0.24199288256227758	0.8	0.371584699453552
0.8	0.9	0.24285714285714285	0.8	0.37260273972602737
0.9	0.1	0.17301038062283736	0.847457627118644	0.28735632183908044
0.9	0.2	0.18556701030927836	0.8852459016393442	0.3068181818181819
0.9	0.3	0.1840277777777778	0.8548387096774194	0.3028571428571429
0.9	0.4	0.18466898954703834	0.8548387096774194	0.30372492836676224
0.9	0.5	0.22857142857142856	0.7901234567901234	0.3545706371191135
0.9	0.6	0.22939068100358423	0.7619047619047619	0.3526170798898072
0.9	0.7	0.2357142857142857	0.7764705882352941	0.36164383561643837
0.9	0.8	0.2392857142857143	0.7790697674418605	0.366120218579235
0.9	0.9	0.24199288256227758	0.8	0.371584699453552

Table D.1: Evaluation results obtained using fastText as source mode; and PEM generated from fastText.

Evaluation of Equation7.1 - Word2Vec

E

α	β	**Precision**	**Recall**	**f-measure**
0.1	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.1	0.2	0.2808988764044944	0.7653061224489796	0.410958904109589
0.1	0.3	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.1	0.4	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.1	0.5	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.1	0.6	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.1	0.7	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.1	0.8	0.2835820895522388	0.7835051546391752	0.4164383561643835
0.1	0.9	0.2777777777777778	0.7894736842105263	0.410958904109589
0.2	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.2	0.2	0.2808988764044944	0.7653061224489796	0.410958904109589
0.2	0.3	0.2835820895522388	0.7916666666666666	0.41758241758241754
0.2	0.4	0.2808988764044944	0.7894736842105263	0.4143646408839779
0.2	0.5	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.2	0.6	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.2	0.7	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.2	0.8	0.2835820895522388	0.7755102040816326	0.41530054644808745
0.2	0.9	0.2656826568265683	0.7659574468085106	0.39452054794520547
0.3	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.3	0.2	0.2808988764044944	0.7653061224489796	0.410958904109589
0.3	0.3	0.2808988764044944	0.7894736842105263	0.4143646408839779
0.3	0.4	0.2788104089219331	0.7894736842105263	0.4120879120879122
0.3	0.5	0.2788104089219331	0.7894736842105263	0.4120879120879122
0.3	0.6	0.2936802973977695	0.797979797979798	0.42934782608695654
0.3	0.7	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.3	0.8	0.2958801498127341	0.7745098039215687	0.4281842818428184
0.3	0.9	0.2867647058823529	0.7959183673469388	0.42162162162162153
0.4	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.4	0.2	0.2819548872180451	0.7653061224489796	0.41208791208791207
0.4	0.3	0.2808988764044944	0.7894736842105263	0.4143646408839779
0.4	0.4	0.2862453531598513	0.7938144329896907	0.42076502732240434
0.4	0.5	0.2899628252788104	0.7959183673469388	0.4250681198910081

0.4	0.6	0.30223880597014924	0.7941176470588235	0.4378378378378378
0.4	0.7	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.4	0.8	0.29259259259259257	0.797979797979798	0.4281842818428184
0.4	0.9	0.2767527675276753	0.7653061224489796	0.4065040650406504
0.5	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.5	0.2	0.2808988764044944	0.7894736842105263	0.4143646408839779
0.5	0.3	0.2862453531598513	0.7938144329896907	0.42076502732240434
0.5	0.4	0.2899628252788104	0.7959183673469388	0.4250681198910081
0.5	0.5	0.30111524163568776	0.801980198019802	0.43783783783783786
0.5	0.6	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.5	0.7	0.2958801498127341	0.7669902912621359	0.42702702702702705
0.5	0.8	0.2878228782287823	0.7959183673469388	0.42276422764227645
0.5	0.9	0.26296296296296295	0.7553191489361702	0.3901098901098901
0.6	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.6	0.2	0.2808988764044944	0.7894736842105263	0.4143646408839779
0.6	0.3	0.2899628252788104	0.7959183673469388	0.4250681198910081
0.6	0.4	0.29739776951672864	0.8	0.4336043360433604
0.6	0.5	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.6	0.6	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.6	0.7	0.29850746268656714	0.7692307692307693	0.43010752688172044
0.6	0.8	0.28308823529411764	0.7857142857142857	0.4162162162162162
0.6	0.9	0.25555555555555554	0.7419354838709677	0.38016528925619836
0.7	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.7	0.2	0.2808988764044944	0.7894736842105263	0.4143646408839779
0.7	0.3	0.2899628252788104	0.7959183673469388	0.4250681198910081
0.7	0.4	0.30111524163568776	0.801980198019802	0.43783783783783786
0.7	0.5	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.7	0.6	0.29699248120300753	0.7669902912621359	0.4281842818428184
0.7	0.7	0.28888888888888886	0.78	0.4216216216216216
0.7	0.8	0.28308823529411764	0.7857142857142857	0.4162162162162162
0.7	0.9	0.25925925925925924	0.7692307692307693	0.3878116343490305
0.8	0.1	0.2808988764044944	0.7653061224489796	0.410958904109589
0.8	0.2	0.275092936802974	0.7872340425531915	0.4077134986225895
0.8	0.3	0.2899628252788104	0.7959183673469388	0.4250681198910081
0.8	0.4	0.29850746268656714	0.7920792079207921	0.4336043360433604
0.8	0.5	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.8	0.6	0.29213483146067415	0.7572815533980582	0.4216216216216216
0.8	0.7	0.28888888888888886	0.78	0.4216216216216216
0.8	0.8	0.28044280442804426	0.7676767676767676	0.41081081081081083
0.8	0.9	0.25092250922509224	0.7555555555555555	0.3767313019390582
0.9	0.1	0.2819548872180451	0.7653061224489796	0.41208791208791207
0.9	0.2	0.2825278810408922	0.7916666666666666	0.4164383561643836
0.9	0.3	0.2899628252788104	0.7959183673469388	0.4250681198910081
0.9	0.4	0.2947761194029851	0.7821782178217822	0.4281842818428184
0.9	0.5	0.291044776119403	0.7722772277227723	0.4227642276422764
0.9	0.6	0.2873134328358209	0.7549019607843137	0.4162162162162162
0.9	0.7	0.2867647058823529	0.78	0.4193548387096773
0.9	0.8	0.28044280442804426	0.7676767676767676	0.41081081081081083
0.9	0.9	0.24354243542435425	0.75	0.36768802228412256

Table E.1: Evaluation results obtained using equation 7.1 from section 7.2; The equation is bounded by having both parameters dependency on sub-calculation. Results shown are obtained using Word2Vec source model and it's respective PEM.

Results after Threshold - Word2Vec

F

α	β	**Precision**	**Recall**	**f-measure**
0.1	0.1	0.3391304347826087	0.7090909090909091	0.45882352941176474
0.1	0.2	0.40860215053763443	0.7916666666666666	0.5390070921985817
0.1	0.3	0.41836734693877553	0.803921568627451	0.5503355704697986
0.1	0.4	0.40384615384615385	0.7924528301886793	0.5350318471337578
0.1	0.5	0.4051724137931034	0.734375	0.5222222222222223
0.1	0.6	0.3983050847457627	0.734375	0.5164835164835164
0.1	0.7	0.3949579831932773	0.734375	0.5136612021857924
0.1	0.8	0.38333333333333336	0.71875	0.5
0.1	0.9	0.38333333333333336	0.71875	0.5
0.2	0.1	0.35185185185185186	0.6909090909090909	0.46625766871165647
0.2	0.2	0.3960396039606	0.8	0.5298013245033113
0.2	0.3	0.42105263157894735	0.8	0.5517241379310345
0.2	0.4	0.40384615384615385	0.7924528301886793	0.5350318471337578
0.2	0.5	0.42857142857142855	0.7384615384615385	0.5423728813559321
0.2	0.6	0.41739130434782606	0.7384615384615385	0.5333333333333333
0.2	0.7	0.4051724137931034	0.7230769230769231	0.5193370165745858
0.2	0.8	0.3983050847457627	0.7230769230769231	0.5136612021857924
0.2	0.9	0.3983050847457627	0.7230769230769231	0.5136612021857924
0.3	0.1	0.34782608695652173	0.6896551724137931	0.4624277456647399
0.3	0.2	0.38144329896907214	0.6727272727272727	0.4868421052631578
0.3	0.3	0.40816326530612246	0.8	0.5405405405405406
0.3	0.4	0.4	0.7924528301886793	0.5316455696202531
0.3	0.5	0.4396551724137931	0.75	0.5543478260869564
0.3	0.6	0.4396551724137931	0.75	0.5543478260869564
0.3	0.7	0.4396551724137931	0.75	0.5543478260869564
0.3	0.8	0.4017094017094017	0.7230769230769231	0.5164835164835165
0.3	0.9	0.3949579831932773	0.7230769230769231	0.5108695652173912
0.4	0.1	0.336283185840708	0.6909090909090909	0.4523809523809524
0.4	0.2	0.3592233009708738	0.6379310344827587	0.45962732919254656
0.4	0.3	0.37755102040816324	0.6727272727272727	0.48366013071895425
0.4	0.4	0.3925233644859813	0.7924528301886793	0.5249999999999999
0.4	0.5	0.4396551724137931	0.8095238095238095	0.5698324022346368

0.4	0.6	0.4482758620689655	0.7536231884057971	0.5621621621621621
0.4	0.7	0.4491525423728814	0.7571428571428571	0.5638297872340425
0.4	0.8	0.4380165289256198	0.7571428571428571	0.5549738219895287
0.4	0.9	0.4322033898305085	0.7391304347826086	0.5454545454545455
0.5	0.1	0.35135135135135137	0.6964285714285714	0.4670658682634731
0.5	0.2	0.36538461538461536	0.6785714285714286	0.475
0.5	0.3	0.3557692307692308	0.6379310344827587	0.45679012345679015
0.5	0.4	0.40816326530612246	0.6779661016949152	0.5095541401273885
0.5	0.5	0.4444444444444444	0.7536231884057971	0.5591397849462365
0.5	0.6	0.4369747899159664	0.7536231884057971	0.553191489361702
0.5	0.7	0.4369747899159664	0.7536231884057971	0.553191489361702
0.5	0.8	0.43333333333333335	0.7536231884057971	0.5502645502645502
0.5	0.9	0.4380165289256198	0.7571428571428571	0.5549738219895287
0.6	0.1	0.3584905660377358	0.7037037037037037	0.47500000000000003
0.6	0.2	0.38095238095238093	0.7017543859649122	0.49382716049382713
0.6	0.3	0.3564356435643564	0.6206896551724138	0.45283018867924524
0.6	0.4	0.38461538461538464	0.6779661016949152	0.49079754601227
0.6	0.5	0.4224137931034483	0.6621621621621622	0.5157894736842105
0.6	0.6	0.4406779661016949	0.7536231884057971	0.5561497326203209
0.6	0.7	0.4482758620689655	0.7536231884057971	0.5621621621621621
0.6	0.8	0.4406779661016949	0.7536231884057971	0.5561497326203209
0.6	0.9	0.43333333333333335	0.7536231884057971	0.5502645502645502
0.7	0.1	0.3611111111111111	0.7090909090909091	0.4785276073619632
0.7	0.2	0.36792452830188677	0.7090909090909091	0.48447204968944096
0.7	0.3	0.36538461538461536	0.6785714285714286	0.475
0.7	0.4	0.3611111111111111	0.6290322580645161	0.45882352941176474
0.7	0.5	0.41739130434782606	0.7058823529411765	0.5245901639344263
0.7	0.6	0.4188034188034188	0.6621621621621622	0.5130890052356022
0.7	0.7	0.4406779661016949	0.7536231884057971	0.5561497326203209
0.7	0.8	0.4262295081967213	0.7536231884057971	0.544502617801047
0.7	0.9	0.43333333333333335	0.7536231884057971	0.5502645502645502
0.8	0.1	0.3611111111111111	0.7222222222222222	0.4814814814814815
0.8	0.2	0.3523809523809524	0.6981132075471698	0.4683544303797469
0.8	0.3	0.3584905660377358	0.6785714285714286	0.46913580246913583
0.8	0.4	0.35454545454545455	0.6290322580645161	0.45348837209302323
0.8	0.5	0.4083333333333333	0.7101449275362319	0.5185185185185186
0.8	0.6	0.415929203539823	0.7014925373134329	0.5222222222222221
0.8	0.7	0.4117647058823529	0.7205882352941176	0.5240641711229946
0.8	0.8	0.4180327868852459	0.75	0.5368421052631578
0.8	0.9	0.42276422764227645	0.7536231884057971	0.5416666666666666
0.9	0.1	0.3627450980392157	0.7551020408163265	0.49006622516556286
0.9	0.2	0.3644859813084112	0.6964285714285714	0.4785276073619632
0.9	0.3	0.3669724770642202	0.7017543859649122	0.48192771084337355
0.9	0.4	0.37272727272727274	0.6833333333333333	0.48235294117647054
0.9	0.5	0.39166666666666666	0.6619718309859155	0.49214659685863876
0.9	0.6	0.41025641025641024	0.7058823529411765	0.518918918918919
0.9	0.7	0.4051724137931034	0.7014925373134329	0.5136612021857924
0.9	0.8	0.41025641025641024	0.7164179104477612	0.5217391304347826
0.9	0.9	0.4214876033057851	0.75	0.5396825396825398

Table F.1: Evaluation results obtained after filtering false positive results by using empirical threshold over Word2Vec source mode; and PEM generated from Word2Vec.