LeaREX: LEARNING RELATIONSHIP EXTRACTION PATTERNS FROM TEXT BASED ON TYPED DEPENDENCIES

by

CHATALI PATEL

(Under the Direction of Krzysztof J. Kochut)

ABSTRACT

An immense number of articles containing important information are being published every day. We have developed a generalized text mining system which automatically extracts relationships between concepts from free text and presents them in user desired format. The system requires example sentences with entities of interest annotated by the user as an input to train the system. The system uses the SPARQL query language as an interface to identify grammatical patterns existing in the sentence, which helps in extracting relationships. A curatorial system can be used to verify extracted relationships. To improve the performance, an additional module was developed that generates SPARQL query patterns using expert feedback from the curatorial system; this module adds patterns to the extraction patterns set. Similar patterns are combined to reduce the overall numbers of distinct patterns to speed up extraction process. Additionally, the module improves system accuracy over time.

INDEX WORDS: Text Mining, Natural Language Processing, Relationship Extraction, Parsing, Pattern Recognition, Ontology, Jena, SPARQL

LeaREX: LEARNING RELATIONSHIP EXTRACTION PATTERNS FROM TEXT BASED ON TYPED DEPENDENCIES

by

CHATALI PATEL

B.E., GUJARAT TECHNOLOGICAL UNIVERSITY, INDIA, 2013

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2017

LeaREX: LEARNING RELATIONSHIP EXTRACTION PATTERNS FROM TEXT BASED ON TYPED DEPENDENCIES

by

CHATALI PATEL

DEDICATION

Dedicated to my family and friends who supported me throughout this work.

ACKNOWLEDGEMENTS

I am very much grateful to Dr. Krzysztof J. Kochut for his motivation and consistent guidance throughout my research. His office door was always open whenever I had any doubt or questions regarding my work. I could not have imagined completing my work without his supervision, patience, and immense knowledge. I would also like to thank my committee members, Dr. Hamid A. Arabnia and Dr. Ismailcem Budak Arpinar, for their support and agreeing to be committee members.

I would like to thank my labmates for their guidance, comments and suggestions. I would also thank my friends who supported me, especially Ankita Joshi, Amitabh Priyadarshi and Sanath Bhat, master's students in The Computer Science Department at The University of Georgia, for helping me work through my issues and clarifying my understanding.

Finally, I would like to thank The University of Georgia, especially The Computer Science Department, for providing me a stimulating environment to study and serving as a gateway for better opportunities.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this chapter, we briefly discuss Text Mining and its applications in different areas. Also, this chapter includes different facets of Text Mining. We have also provided an overview of our approach and how it could be helpful in extracting information from the data contained in a natural language text.

1.1 Introduction

A large amount of textual data has been generated during the past several years. Every single day, about 10 million web pages are being added to the Internet, and this trend is expected to keep accelerating [11]. Most of this data is in an unstructured form; in other words, it is represented in natural language as text data. For scientists, it is virtually impossible to process this data and manually discover needed elements. Much of this data contains very useful information but it is not easy to read. This situation can be affectively addressed by Text Mining. Text Mining [1] is popular technology that strives to derive meaningful information from the large amount of unprocessed natural language data, which may be ambiguous, unstructured or semi-structured and difficult to process automatically. In simple terms, it is the ability to process unstructured text from thousands or even millions of articles, interpret the meaning and automatically identify and extract useful information, such as the relationships exist of texting between concepts of interest. To mine the information, a text mining process usually involves a series of tasks to be performed, such as document collection, document pre-processing [2], Part-of-Speech (POS) tagging [3], and application of other Natural Language Processing techniques. These tasks are explained in detail in Chapter 2 of this thesis. Text mining has many different facets, depending on the characteristics and applications of the area. The following are examples:

- Search and Information Retrieval (IR)
- Document Classification
- Data Mining
- Natural Language Processing (NLP)
- Information Extraction (IE)

Search and Information Retrieval [4] helps in searching documents relevant to the problem, which significantly speeds up the analysis process. Document Classification [5] helps in managing and sorting documents by assigning them different categories or classes. Data mining [6] is the process of extracting useful information from data and converting it into an easily readable form. Natural Language Processing (NLP) [7] enables computers to glean meaning from natural language text, which is not easily understandable by machines. Often, NLP works as an input to the Information Extraction (IE) phase. Many NLP applications exist and use word dictionaries, ontologies, word-lists, rules etc. In the Information Extraction [8] phase, structured information is extracted from semi-structured or unstructured machine readable data, which is often received from the Natural Language Processing phase.

1.2 Major Contribution

In this work, we mainly focused on Information Extraction and Natural Language Processing to automatically extract relationship from text using grammatical dependence patterns.

A wide range of Text Mining Systems have been developed according to the area of application. One of the most common areas where text mining is applied the most is the biomedical domain. The majority of the text mining systems use a bag of words and some set of predefined rules or patterns to train the system, and this trained system is eventually used to identify/predict possible outcomes.

We have developed a generalized text mining system that can be applied to arbitrary natural language text with minimum human effort. It is domain independent and builds its own knowledge base in any new corpus of text automatically using provided training data. It requires the user to provide a set of example sentences with the index of critical words present in the sentence. The user also needs to provide a set of words from different categories which can exist in other testing sentences. The system utilizes the grammatical structure of the sentence for analysis, transforms it into graph structure using a graph library and, ultimately, takes advantage of **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage (SPARQL) [9] expressiveness to encode it into a SPARQL query. Later, these SPARQL queries are used for pattern matching to predict the relationship between concepts in the text. To test our system, we used a system called Ki-MIner [68], created and implemented by Bhargabi Chakrabarti, a student who graduated from The Computer Science Department at the University of Georgia, as part of her Master’s Thesis project. She used articles from PubMed Central [14] for her system. To assure the correctness of the predictions generated by the system, we used a system called CURAMI [69], created and implemented by Reshmi De, a student who graduated from The Computer Science Department at the University of Georgia, as part of her Master’s Thesis project. It is a curatorial system that provides a significant level of assistance to human experts, implements a multilevel verification and provides a feedback mechanism. It presents the extraction outcome along with all relevant information to the human curators. Curators assess the predictions and give feedback on them. We also added an additional feature to improve our system using curatorial knowledge. The system generates the SPARQL patterns using expert feedback and adds them to the training dataset automatically rather than requiring manual observation. This makes the system robust and more reliable over time. To increase speed and efficiency, we attempted to optimize the system by minimizing the number of patterns generated automatically. The measures taken to achieve this are described in the subsequent chapters.

In Chapter 2, we discuss background information regarding Text Mining, Ontology and SPARQL, Parsing and the Stanford Parser, Natural Language Processing and Information Extraction from Text. Chapter 3 discusses related work that has been done in this research area. Chapter 4 states our approach to extracting information from text using dependence patterns. Chapter 5 explains the implementation of our approach. Evaluations are described in Chapter 6. Finally, Chapter 7 presents the conclusion and a discussion of future work.

CHAPTER 2

BACKGROUND

2.1 Text Mining

Text Mining is an increasingly popular area of research in computer science that strives to derive meaningful information from the large amount of unprocessed natural language data, which may be ambiguous, unstructured or semi-structured and difficult to process. This unprocessed data is in the form of newspaper or web articles, email, blog entries, internal reports, research papers, transcripts of phone calls and many more [10]. Text Mining generally attempts to discover the semantics of the data by deducing rules or non-trivial patterns from natural language text documents, which helps in automatically discovering knowledge from other unprocessed documents. The challenge here is to transform unstructured data into rules or patterns. They can be domain-specific or general rules that can be applied to any domain. Also, they can be built by hand or automatically with minimal human involvement.

Machine Learning, Data Mining, Information Extraction and Natural Language Processing are some of the most frequently used techniques for discovering knowledge using Text Mining [12]. Though Text Mining and Data Mining tend to have a similar purpose, Text Mining is regarded as having higher potential than Data Mining. The reason is that Text Mining discovers knowledge from free text, while Data Mining attempts to discover knowledge from structured data like a database. Thus, Text Mining can be viewed as an extension of Data Mining. In other words, it can be called Text Data Mining. As mentioned in the Introduction chapter, mining text involves a series of tasks to mine the text such as Information Retrieval (document collection), document pre-processing, Part-of-Speech (POS) tagging, Natural Language Processing and others which are described in detail below.

2.1.1 Information Retrieval

Information Retrieval (IR) is also called Document Collection. It is the first key step in Text Mining. IR is the activity of acquiring static or dynamic documents that are relevant to the domain of research. In static document collection, documents remain the same, while in dynamic document collection, documents are updated over time [12]. IR starts with the user query containing relevant strings describing the kind of documents needed. It returns documents (or images, audio, video, etc.), which may or may not match the user search. PubMed [13] is one of the largest platforms for text data collections. It has been maintained by The United States National Library of Medicine. It contains millions of documents on biomedical topics.

2.1.2 Document Pre-Processing

Once the relevant documents are collected, the next step is to process them. Since the text in the documents is not structured data, analyzing it becomes a difficult task. This text might also have redundant and irrelevant data. In this step, the syntactic and semantic structure of the text is extracted, and characters, words or sentences are identified. In the later processing stages of Chakrabarti’s thesis, Part-of-Speech tagging was applied to the resulting sentences to give the sentence a better structure.

2.1.3 Part-of-Speech (POS) tagging

Tagging means annotating a sentence and Part-of-Speech references grammatical category of words (POS tags includes the verb, noun, adjective, preposition, etc.) [3]. This means that words having the same POS belong to the same category and are similar in some way. Generally, they tend to have similar syntax. Thus, POS tagging annotates words in the text by appropriate category based on their grammatical relationship with other words in the phrase, sentence or paragraph. POS tagging is usually a sentence-based process that labels the word by its correct part-of-speech, which helps in adding more structure to the sentence. Most of the English taggers use The Penn Treebank [15] tag set. The example sentence and its part-of-speech tagging is shown below:

"I like watching movies" [16]

- I-PRP like-VBP watching-VBG movies-NNS

Here, we see that *I* is PRP (Personal Pronoun), *like* is VBP (Verb, non-3rd person singular present), *watching* is VBG (Verb, gerund or present participle) and *movies* is NNS (Noun, plural). These tags are taken from The Penn Treebank Project.

POS taggers perform morphological analysis of words and produce stems for the input words. For the purpose of stemming, we apply The Porter Stemmer algorithm [17]. Many part-of-speech taggers are available nowadays. In our approach, we used the Stanford Dependency Parser [18], which first performs POS tagging of the sentence and then generates the parse tree and the grammatical typed dependencies present between the words in the sentence. The Stanford Parser is described in greater detail later in this chapter.

2.2 Natural Language Processing

Natural Language Processing enables a computer to process natural language text or human speech by extracting the semantics from it. To extract the semantics from a sentence, NLP performs grammatical analysis, which helps in reading the text. Some of the most common NLP tasks are: Part-of-speech (POS) tagging, Parsing, Named Entity Recognition (NER) and Relationship extraction. As mentioned above, Part-of-speech tagging determines the part of speech (such as noun, adjective, verb, etc.) for each word in the sentence. Parsing [19] determines the syntax of the sentence. It tokenizes the sentence and converts it into a parse tree [20] structure presenting its syntactic representation. Named Entity Recognition [21] finds named entities from text and annotates them according to some pre-defined categories (e.g., names of persons, locations, organizations, expressions of times, monetary values, percentages, quantities, etc.). Relationship Extraction [22] identifies semantic relations between those named entities in text. One uncommon way to represent those relationship is by using RDF (Resource Description Framework) [23] triples in domain ontologies [24]. RDF triples are in the form of (subject, relationship, object).

NLP techniques can be employed in many applications. One of the popular applications is sentiment analysis of social media posts to determine public opinion or trends for marketing, customer service, or other purposes. Question-answering systems are also one of the most used applications, in which a user query is taken as an input and a relevant result is produced as output. A few other applications are machine translation, automatic summarization, spam email filtering, etc. We have developed an information extraction system that uses natural language processing techniques like POS tags, parse tree and typed dependencies using the Stanford NLP parser library to retrieve information as per user requirements.

The difference between Text Mining and Natural Language Processing is that Text Mining transforms unstructured data into a structured form to discover new information from text, while Natural Language Processing is one of the analysis methodologies to achieve that goal. Thus, NLP can be regarded as a component of Text Mining, which performs linguistic analysis to extract meaning.

2.3 Parsing and Stanford Parser

Parsing refers to the process of examining or analyzing natural language text. The Stanford Parser is a probabilistic natural language parser which uses a highly optimized PCFG (Probabilistic Context-Free Grammar) [18].

A context-free grammar, abbreviated as CFG, consists of 4-tuple [25]:

$G = (N, \Sigma, R, S)$,

where

- N is a finite set of non-terminal symbols
- Σ is a finite set of terminal symbols
- R is a finite set of rules of the form $X \rightarrow Y_1 Y_2 \ldots Y_n$, where $X \in N, n \geq 0$, and $Y_i \in (N \cup \Sigma)$ for $i = 1 \ldots n$
- $S \in N$ is a distinguished start symbol.

Probabilistic Context-Free Grammar is an enhancement of CFG in which rules are associated with a parameter called probability. Each rule is assigned a probability that lies between 0 and 1. Rules can be in the forms shown below:

S → NP VP

VP → Vi

NP → DT NN

NP → NP PP

PP → IN NP,

where S=sentence, NP=noun phrase, VP=verb phrase, Vi=intransitive verb, DT=determiner, NN=noun, PP=prepositional phrase, IN=preposition [25]. These are the POS tags taken from

Penn Treebank. A parse tree can be constructed using these rules. Let us take an example sentence:

"Keith saw the man with the telescope" [26]

The parse tree for this sentence is shown in Figure 1.

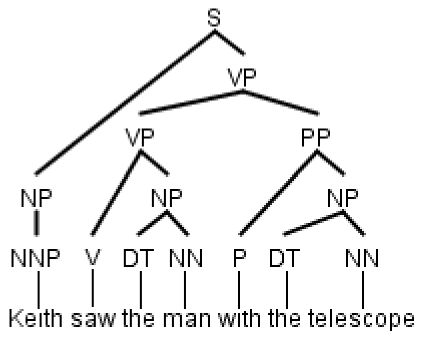


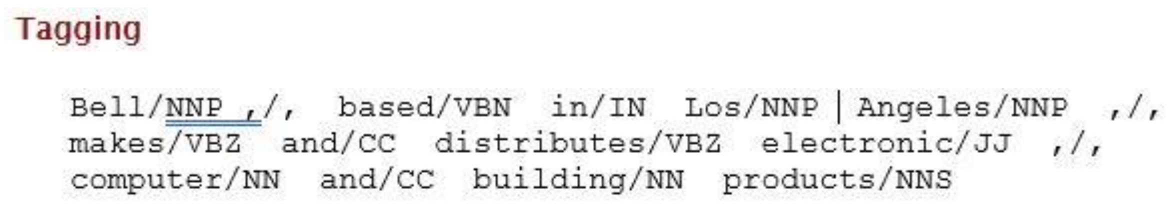
Figure 1: Parse tree representation of the sentence "Keith saw the man with the telescope"

Since the existence of ambiguity is obvious in natural language, an NLP parser sometimes generates more than one parse tree for the same sentence. Here, PCFG takes the advantage of augmented probabilities and generates the most probable parse tree or a ranking of the parse trees. Thus, PCFG improves the accuracy of parsing.

In addition to parsing, the Stanford parser also provides grammatical relationships between words in the sentence in the form of typed dependencies.

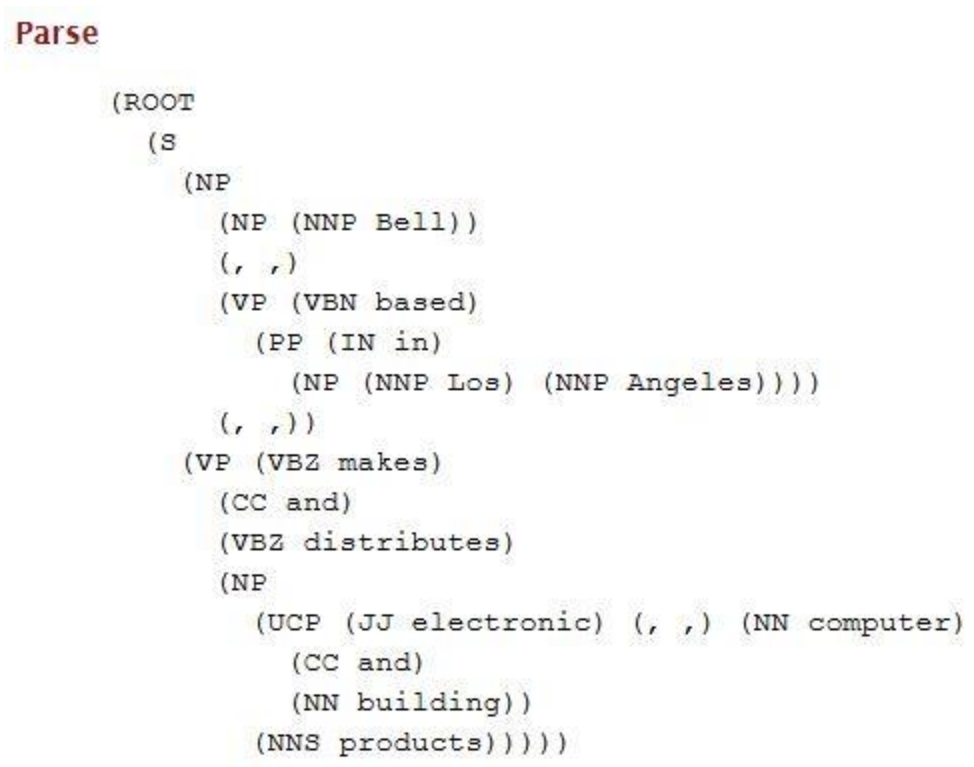
Below is an example sentence with POS tagging, parse tree and typed dependencies generated by The Stanford Parser:

> "Bell, based in Los Angeles, makes and distributes electronic, computer and building products" [27]


Tagging

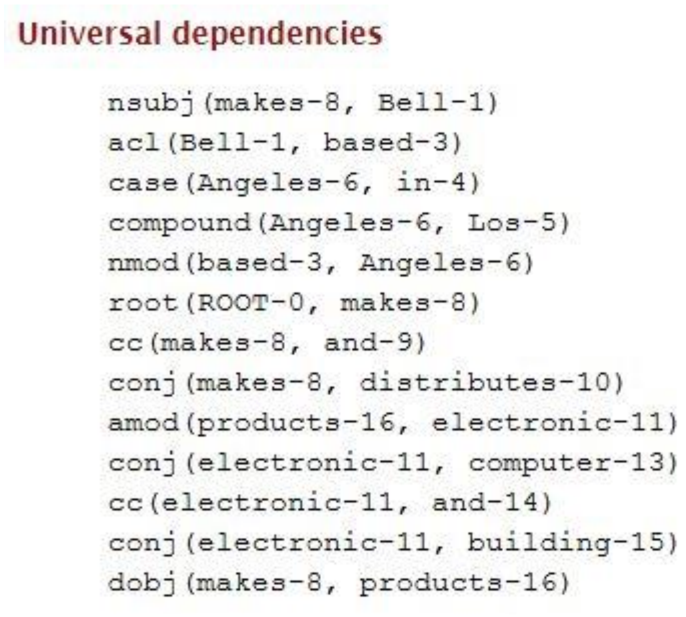
```
Bell/NNP ,/,  based/VBN  in/IN  Los/NNP | Angeles/NNP  ,/,
makes/VBZ  and/CC  distributes/VBZ  electronic/JJ  ,/,
computer/NN  and/CC  building/NN  products/NNS
```

Figure 2: POS Tagging of the sentence “Bell, based in Los Angeles, makes and distributes electronic, computer and building products” generated by The Stanford Parser [28]


Parse

```
(ROOT
  (S
    (NP
      (NP (NNP Bell))
      (, ,)
      (VP (VBN based)
        (PP (IN in)
          (NP (NNP Los) (NNP Angeles))))
      (, ,))
    (VP (VBZ makes)
      (CC and)
      (VBZ distributes)
      (NP
        (UCP (JJ electronic) (, ,) (NN computer)
          (CC and)
          (NN building))
        (NNS products)))))
```

Figure 3: Parse of the sentence “Bell, based in Los Angeles, makes and distributes electronic, computer and building products” generated by The Stanford Parser [28]


Universal dependencies

```
nsubj(makes-8, Bell-1)
acl(Bell-1, based-3)
case(Angeles-6, in-4)
compound(Angeles-6, Los-5)
nmod(based-3, Angeles-6)
root(ROOT-0, makes-8)
cc(makes-8, and-9)
conj(makes-8, distributes-10)
amod(products-16, electronic-11)
conj(electronic-11, computer-13)
cc(electronic-11, and-14)
conj(electronic-11, building-15)
dobj(makes-8, products-16)
```

Figure 4: Typed dependencies of the sentence “Bell, based in Los Angeles, makes and distributes electronic, computer and building products” generated by The Stanford Parser [28]

The Stanford parser first tokenizes the sentence and assigns an index to each word in the sentence. Then, it uses those indexes in typed dependency generation. Thus, the number shown in the typed dependencies indicates the index of that word. Typed dependencies represent relationships between governor (head) and dependent. Definitions of the typed dependencies generated in the example above are explained below:

- **nsubj:** nominal subject

  E.g. “Clinton defeated Dole” [29]

  nsubj(defeated, Clinton)

- **acl:** clausal modifier of noun

  E.g. “I don’t have anything to say to you” [29] acl(anything, say)

- **case:** case marking

  E.g. “I saw a cat in a hat” [29]

  case(hat, in)

- **compound:** compound

  E.g. “I have four thousand sheep” [29]

  compound(thousand, four)

- **nmod:** nominal modifier

  E.g. “the office of the Chair” [29]

  nmod(office, Chair)

- **root:** root

  E.g. “Bill is an honest man” [29]

  root(ROOT, man)

- **cc:** coordination

  E.g. “And then we left.” [29]

  cc(left, And)

- **conj:** conjunct

E.g. “Bill is big and honest” [29]

conj(big, honest)

- **dobj:** direct object

  E.g. “She gave me a raise” [29]

  dobj(gave, raise)

There are many more typed dependencies used by the Stanford Parser, such as advcl, advmod, xcomp, ccomp, etc. [29], which are not further explained here.

Typed dependencies can be represented in many different forms. The basic type and the collapsed types are the most used types of representation.

The basic type forms a tree structure using dependencies. The graph generated using basic typed dependencies does not contain cycles. An example of basic dependency representation for the phrase “based in LA” is shown below [27]:

prep(based-7, in-8)

pobj(in-8, LA-9)

In the collapsed type representation, basic dependencies containing conjuncts or prepositions collapse to obtain direct dependencies between main words in the sentence. In a collapsed dependency representation, the example of basic dependencies shown above will be collapsed as follows:

prep_in(based-7, LA-9)

There are other variants of collapsed typed dependency representation:

1. Collapsed dependencies with propagation of conjunct dependencies
2. Collapsed dependencies preserving a tree structure

The dependencies which contain conjuncts in it are propagated in Collapsed dependencies with propagation of conjunct dependencies. With this form, preservation of a tree structure is not assured.

Collapsed dependencies preserving a tree structure are the same as Collapsed dependencies with propagation of conjunct dependencies with removal of those dependencies that do not preserve tree structure.

2.4 Ontology and SPARQL

An Ontology is a theory about what kind of objects exist in the universe and how they are related to each other. Ontology defines concepts (objects/entities) in the form of classes and relationships (interactions) between them in terms of properties. Ontologies can be used to create a knowledge base in any domain using a representation vocabulary. A Representation vocabulary provides a list of concepts in a particular domain. The W3C states that: “An ontology defines the terms used to describe and represent an area of knowledge.”

Relational databases are also being used as a knowledge base for many applications, but there are some features that make ontologies special. In ontologies, we can infer implicit knowledge that we cannot with a relational database. The main advantage of using ontologies is that they represent the intended meaning of vocabularies, while a relational database only represents data [31]. For any domain, its ontology builds a core of knowledge that clarifies the structure and constraints of the domain knowledge.

When should we use an ontology? When the schema is very large and/or complex, and schema information is used while querying, we should consider using an ontology. On the other hand, when the schema is small and simple, and querying does not use the schema structure, a database should be used. One of the most popular real world ontologies is WordNet [30], which models the domain of the English language. It is comprised of concepts, concept types and relations between concepts to provide a “terminological knowledge base” in the English language domain [12].

Thus, a major application of ontologies is the Semantic Web [32]. The contents on the web are not structured. They can be made useful by converting complex website content and the large number of applications on semantic web into ontologies to include the semantically related information and exclude irrelevant information [31].

The semantics in the ontology rely on the representation language. Many languages have been developed to represent an ontology [33]. Some of the semantic markup ontology languages are Resource Description Framework (RDF) [23], RDF Schema (RDFS) [34], Web Ontology Language (OWL) [35], etc., and traditional syntax ontology languages are LOOM (ontology) [36], KIF (Knowledge Interchange Format) [37], F-Logic (Frame Logic) [38], etc.

Resource Description Framework is a framework that models metadata information on web resources in graph format. It enables reuse and interchange of the data on the web using XML (eXtensible Markup Language) syntax. XML syntax makes data user-readable and machine-processable. RDF represents information in the form of triple (subject, predicate/property, object) expressions. These triples collectively can generate a graph. In RDF, URIs can also be used as resources. Thus, RDF is a method for making statements on web resources (like URI) in the form of triples, and the semantic web’s structure is constructed using the RDF concept. RDF is a vocabulary of terms which can be rendered in number of different formats such as RDF/XML, N3, Turtle, etc.

RDF Schema (RDFS) provides a schema to RDF terms. It is a language to represent RDF vocabularies. In simple terms, we can say that RDF defines instances, while RDFS provides a way to define classes and properties for those instances. RDFS adds semantics to the RDF terms. For example, Car and Vehicle are two instances defined in RDF, RDFS gives us the flexibility to say that Car is a class and Car is a sub-class of the Vehicle class. In RDF vocabulary, the term “rdf:type” is defined, stating that the subject which comes before “rdf:type” is an instance of an object that comes after “rdf:type”. For example [39],

```
PREFIX rdf:<https://www.w3.org/ 1999/02/22 -rdf-syntax-ns>
<http://example.com/Car> rdf:type <http://example.com/Vehicle>
```

Here, PREFIX defines the document it refers to. Car is the subject, and Vehicle is the object.

On the other hand, in RDFS, “rdfs:subclassOf” is one of the example terms in its vocabulary. For the example above, it can be defined as [39]:

```
PREFIX rdfs:<https://www.w3.org/2000/01/rdf-schema#>
<http://example.com/Car> rdfs:subClassOf  <http://example.com/Vehicle>
```

Along with classes and sub-classes, RDFS allows the definitions of properties, sub-properties and the typing of properties. Although RDFS is expressive, it lacks many expressions, some of which are as follows:

- range restrictions to particular classes
- inverse of property

- negation of an expression
- cardinalities
- metadata of the schema
- disjoint classes

Web Ontology Language (OWL) addresses the limitations of RDF Schema and adds more semantics to data. OWL also provides a much larger vocabulary and metadata than RDF or RDFS. Hence, OWL can be used as a robust data modeling language that provides the ability of automatic reasoning.

OWL defines following three species which are used as per the needs of users or implementers to allow layering within OWL:

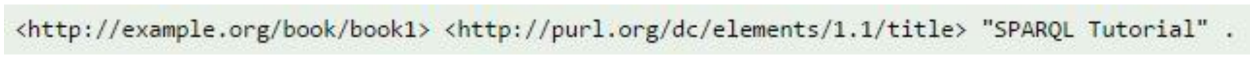
1. OWL Lite
2. OWL DL
3. OWL Full

OWL Lite is the simplest of the three. It provides simple constraints such as cardinality, but it allows only 0 and 1 as a value. OWL DL, as its name suggests, supports description logic capabilities. It guarantees computational completeness, decidability and maximum expressiveness [40]. OWL Full provides maximum expressiveness but does not guarantee computational completeness. Also, it does not have any syntactic constraints on RDF [40]. Since RDF documents cannot be assumed to be compatible with OWL Lite or OWL DL documents, RDF constructs cannot be mixed with OWL Lite or OWL DL constructs. On the other hand, OWL Full documents are compatible with RDF documents; thus, OWL Full and RDF documents can be mixed, augmented or redefined. This shows that RDFS is more expressive than RDF, and OWL is more expressive than RDFS.

Once the ontologies are created with a set of RDF triples, they can be queried using SPARQL (Simple Protocol and RDF Query Language) [9]. Just as SQL is used as a query language to query relational databases, SPARQL is used to query the semantic web. It was created by the W3C and is comprised of a PREFIX declaration to abbreviate URIs, a FROM clause to define which RDF graph to query, a SELECT clause defining the information to extract from the ontology in the form of variables starting with a Question mark (?), and a WHERE clause stating conditions to be satisfied while querying an ontology in the form of triple patterns. Triple patterns are just like triples except that they can contain variables.

A simple example of a RDF dataset, SPARQL query and result set is given below:

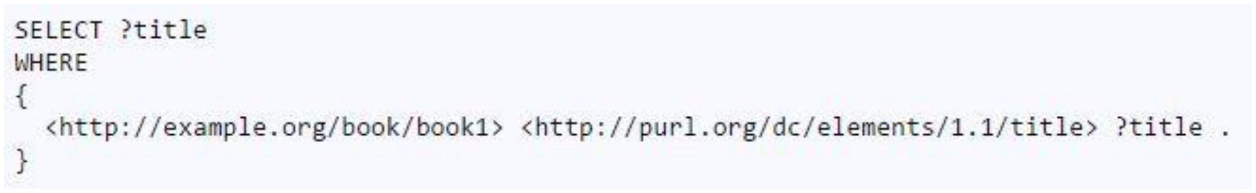
Data [9]:



```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Figure 5: Example RDF data

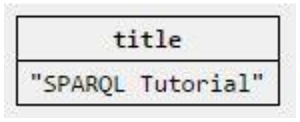
SPARQL Query [9]:



```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Figure 6: Example SPARQL query

Result [9]:



title
"SPARQL Tutorial"

Figure 7: Example SPARQL query result

No PREFIX is used in this example. The example variable “?title” in the SELECT clause contains variables to be returned as a result. The WHERE clause consists of triple pattern which instructs the system to return the title of a book from the given dataset. SPARQL also allows placing constraints on string or numerical data using FILTER functions [9].

2.5 Information Extraction from Text

Owing to the complexity of natural language text, it is very difficult to understand the meaning of the text. The amount of electronic content is increasing tremendously day by day. This leads to a need for building a system that identifies entities and relationships between those entities and extracts structured data from unstructured text. Information extraction works on two key jobs which are Named Entity Recognition and Relationship Extraction which identify entities and extract relations between them, respectively. Thus, Information Extraction helps in transforming textual unstructured or semi-structured data into structured form and extracts useful/meaningful information. Information Extraction can be applied to a wide range of applications in different domains. Depending on the particular requirement of an application, information is extracted in different type of structured format.

Following is an example sentence and the extracted result by applying information extraction task:

“*In 1998, Larry Page and Sergey Brin founded Google Inc.*” [41]

Extracted information:

“*FounderOf(Larry Page, Google Inc.),*

FounderOf(Sergey Brin, Google Inc.),

FoundedIn(Google Inc., 1998).” [41]

In this work, we have developed a Text Mining system which uses an information extraction task as an initial step to generate relationship extraction patterns. Along with the information extraction, it also uses a parsing technique and some other steps to achieve the goal which is described in detail in Chapter 4 and Chapter 5. The original Information Extraction systems and many other currently developed systems work on hand-coded rules or patterns to extract needed information. Also, the majority of them are domain specific. We have developed a system which automatically generates relationship extraction patterns with the least amount of human effort. In addition, it is domain independent and can be used for any area of interest.

CHAPTER 3

RELATED WORK

The aim in developing a system (text mining system) that extracts structured information from unstructured text can be achieved using different approaches. This section discusses the implementation of six different systems which are related to our work. These systems usually aim to create a set of rules or patterns to extract useful information from natural language text. Also, the strengths and weaknesses of these systems are discussed in terms of comparative analysis.

3.1 "Learning Information Extraction Patterns from Examples"

This paper introduces a system called LIEP (Learning Information Extraction Patterns) [42], which learns a set of patterns from user-provided example sentences and events to be extracted based on local syntax. Those patterns are later used by the extraction system, called ODIE (On Demand Information Extractor), to extract relationships between key events in the sentence. ODIE takes text as an input and breaks it into sentences. If a sentence contains any event of interest, its words were tagged using Eric Brill's part-of-speech tagger [43]. Next, it applies patterns generated by LIEP to identify events and check the syntactic relationships between them. ODIE does not parse the whole sentence but checks for the possible availability of syntactic relationships between events. It makes a strong assumption about what part of the sentence could have an event and parses that part of the sentence.

Before building a new pattern for any sentence, LIEP checks if any known pattern can be matched by this sentence. If that is not the case, it tries to produce a generalized pattern that can cover that example. If it fails to do so, it generates a new pattern. To extract a pattern, LIEP finds a path between three (two role-filling constituents and the relationships between them) pairs and combines those paths to create a set of relationships and creates a pattern from the relationships. ODIE's average values of precision, recall, and F-measure [44] for hand built patterns were 93.2%, 85.9%, 89.4%, respectively and for LIEP-built patterns, the values were 89.4%, 81.6%, 85.2%, respectively.

Hence, LIEP is a tool which does not require any specialized programmer to extract patterns from free text. Also, patterns learned by this system have performance very close to the level of a hand-built pattern dictionary. The general task of extracting relationships using example sentences is similar to our approach. The difference is in creating patterns and identifying matching patterns using ODIE. In LeaREX, patterns are encoded as SPARQL queries to leverage SPARQL's expressivity and reasoning power. The Jena [66] library is used to identify matching patterns. Also, LIEP makes some assumptions while learning new patterns, which adds some uncertainty to the system, which is not the case with LeaREX.

3.2 "Semanticizing Syntactic Patterns in NLP Processing using SPARQL-DL"

The main purpose of this project [45] is to use the SPARQL-DL [46] query language as a rule engine to identify syntactic patterns in a sentence. Also, the authors have used FrameNet [47] elements and valance patterns to add semantics to a sentence along with the syntactic tagging obtained from the NLP parser (The Stanford Parser), which together helps in obtaining a semantic parse of a sentence. FrameNet carries about 1,200 semantic frames, 13,000 lexical units, and over 190,000 example sentences. *Lexical Units* (*LUs*) in a FrameNet play a key role.

They are usually in a verb form and can be used in a sentence in various *Syntactic Realizations* (*SRs*) depending on its role and form (e.g. I like sleeping and I like iPhone). A valance pattern is used to present those realizations. It consists of *frame elements* (*FEs*) like Experiencer, Reason, Parameter, etc.

FrameNet's example valance pattern for the sentence "*I like him as a fellow*" given in the paper is:

$NP.Obj_{Content}\ NP.Ext_{Experiencer}\ PP[as].Dep_{Parameter}$

$[I]_{Experiencer}$ *like* $[him]_{Content}$ $[as\ a\ fellow]_{Parameter}$

where the subscripts are the *FEs* and the words separated by dot are *phrase type* (*PT*) and *grammatical function* (*GF*), respectively. By using valance pattern's <PT, GF> pair and typed dependencies, SPARQL-DL queries were formulated by hand. Here, SPARQL-DL queries are used to detect PhraseType Dependency patterns only in the sentences.

Object properties containing PP[as].Dep are:

ObjectProperty(VBP, prep, IN)

ObjectProperty(IN, pobj, NN)

SPARQL-DL query for matching the pattern can be represented as:

```
PREFIX : <ontology prefix>
SELECT ?n1 ?n2 ?n3
WHERE { Type(?n1, :VB), Type(?n2, :IN), Type(?n3, :NN),
                    PropertyValue(?n1, :prep, ?n2),
                    PropertyValue(?n2, :pobj, ?n3),
                    PropertyValue(?n2, :lemma, "as") }
```

A predefined ontology with POS tags as classes, grammatical dependencies as object properties, and a data type property ‘lemma’, which relates a word with its base form, was populated for each sentence with words as individuals of the defined classes and object properties relating the words. To leverage the capabilities of an ontology, some rules or constrains can be applied to the ontology for pattern matching. It is also possible to loosen some of the constraints in case no matching sentence is found.

After constructing the temporary ontology for the sentence, all the queries were executed against it, and the resulting <PT, GF> pairs were collected. Using those pairs returned as a result, a suitable matching valance pattern (consisting of frame elements associated with PhraseType.Dependency pairs defined in FrameNet) was chosen to semantically tag the sentence using the valance pattern and Frame Entities. As this work is intended to provide a domain independent system using SPARQL, it is much related to our work, but in this work, SPARQL queries were formulated by hand, which is automatic in our system.

3.3 “RelEx—Relation extraction using dependency parse trees”

The authors of this paper [48] proposed a system called RelEx to extract relationships from free biomedical texts. This system was evaluated on one million MEDLINE abstracts to extract protein-gene relationships, and about 150,000 relationships were extracted with a performance of approximately 80% for both precision and recall. It was also applied to other datasets like the Learning Language in Logic (LLL) [49] dataset and the Human Protein Reference Database (HPRD) [50] dataset to evaluate system performance. In this approach, they used MedPost [51] for part-of-speech tagging, fnTBL [52] to identify noun-phrase chunks, and The Stanford Lexicalized Parser to generate dependency trees for a sentence and to set word positions for each word in the sentence. These dependency trees are later enriched with gene and protein words by

ProMiner [53] based on matching to a synonym dictionary [54]. Based on dependency parsed trees, a set of three simple rules were manually crafted to extract relationships from sentences of a text. RelEx checks for the presence of a protein pair and then looks for a relationship term on the path between two protein terms created by the dependency parse tree. They use the following three rules to describe relations:

1. effector-relation-effectee (‘A activates B’)
2. relation-of-effectee-by-effector (‘Activation of A by B’)
3. relation-between-effector-and-effectee (‘Interaction between A and B’).

These rules were applied to extract candidate relations which were later given as an input to the filtering module for negation check, effector/effectee detection, enumeration detection, and restriction to focus the domain of interest for screening the relations. These patterns of rules can be adapted or expanded to any other domain to extract different types of relations by using corresponding relation terms and entity terms including synonyms.

Thus, RelEx is a simple and straightforward approach based on publicly available tools with higher performance noted on public datasets. This system is also related to our system as it extracts relationships from a text using NLP techniques but it is rule-based rather than pattern-based.

3.4 “A Framework for Schema-Driven Relationship Discovery from Unstructured Text”

A schema-driven approach is described in this paper [55] to extract implicit and explicit relationships between known entities in biomedical texts. It uses a combination of vocabulary from the Medical Subject Headings (MeSH) [56] and domain knowledge in the form of the Unified Medical Language System (UMLS) [57]. Since MeSH also contains synonyms of entities, it eliminates the need for Named Entity Identification and Named Entity Disambiguation/Reference Reconciliation. Later, NLP techniques were applied on the domain knowledge to extract relationships. An empirical rule-based method was used to extract entities and relationships between them and to convert them in RDF form. In the methodology, first they split the PubMed [13] text into sentences, tag parts-of-speech (using SS-Tagger [58]) and generate a parse tree (using SS-parser [59]). The parse tree obtained was enriched by known entities (defined in MeSH terms) and relationships (defined in the UMLS). In any biomedical domain, entities are not always in a simple form. They might be combined with other entities or modifiers. They used rules to identify simple, modified, or composite entities and developed an algorithm to apply those rules. Eventually, they developed an algorithm to extract relationships between them. A relationship extraction algorithm checks the parse tree to determine if children under node S (sentence - root node) contain an entity followed by the relationship term, which in turn is followed by another entity. If that is true, it indicates the presence of a relationship between two entities. They validate those relations using UMLS schema information. After having all relationships between entities identified, they converted them into RDF form. These RDF resources can be utilized in many applications for experimental analysis of large datasets and to obtain useful information by querying.

To evaluate the scalability of the system, they used two datasets from PubMed. Another objective behind this experiment was to check the effectiveness of the rules they developed to extract different types of entities and relationships between them and to understand the usefulness of extracted RDF triples. In the first dataset, they processed about 1.6 million candidate sentences, which resulted in 200,000 triples. In the second dataset, out of 798 candidate sentences, 122 relationships were extracted. More modified entities were extracted than composite or simple entities. The number of simple, composite, and modified entities extracted were 752, 377 and 4762, respectively. From 122 relationships identified, 5 were incorrect, yielding a precision of 95%. They did not measure the recall because of the need for an expert to read all sentences to verify extracted relations. They planned to do that in the future. Hence, results clearly show the ability of generated RDF resources in discovering knowledge from large texts by using analytical path queries. Though this approach focuses on general relationships rather than any specific type of relationships, the overall system concentrates on the biomedical domain.

3.5 “Learning User-Defined, Domain-Specific Relations: A Situated Case Study and Evaluation in Plant Science”

The goal of this paper [60] was to develop a system that predicts the nature of the relation that exists between two entities automatically by using a lightly supervised machine learning approach and giving the user the flexibility to define domain specific relations.

The first step in the process of achieving the goal was to define domain specific relations between a plant and a location from the plant science articles with the help of domain experts and other research group members. Relations identified were neither primitive nor domain specific nor general. They can be said to be the subtype of a more general ‘*location-of*’ or ‘*spatially-related-to*’ relation. The four relations identified were between:

1. plant and the manufacturer of the tool or instrument that was used
2. plant and the location of a seed donor
3. plant and seed origin
4. plant and location of the field experiment

The system provides user-facilitated indexing that requires the user to provide the sentences with relations of interest extracted from articles. For the case study, the author considered 662 candidate sentences out of 2595 sentences annotated with a plant and location. Out of 662 sentences, 110 were used for testing, 518 for training, and 34 were removed because of ambiguous relations.

They used entity, lexical, and syntactic features that were helpful in using training sentences. In the entity feature, plant names and locations were identified using the Unified Medical Language System (UMLS) and Stanford Named Entity Recognizer, respectively. While identifying lexical features for each sentence, highest frequency and inverse document frequency weighted words that occur between plant and location were extracted and ranked to obtain words with highest rank in each category. The Stanford syntactic parse of the sentence was used to identify syntactic features like root of the sentence, and syntactic path from root to plant and from root to location. It was also used to check if location appeared under the same branch of plant. If not, root was used to connect the path between plant and location.

Using annotated sentences and features, binary classifiers were created and were run on 518 training sentences to learn domain specific relations using the Oracle Data Miner (ODM) [61] with the default settings of Generalized Linear Model (GLM), Decision Tree, Support Vector Machines (SVM), and Naïve Bayes (NB).

To evaluate the performance of the system, the SVM model created using training sentences was applied to the remaining 1967 sentences, and 200 randomly selected sentences (with 50 instances in each category) were manually evaluated. This resulted in the highest precision of 94% for *manufacturer_location* relation, which was consistent with training set results. The precision values for *seed_bank-donor_location*, *seed_origin_location* and *field_experiment_location* categories were lower from what was achieved in training sets. Thus, the results demonstrate that one of the four categories achieved good performance for test data, but user can change the default settings of ODM to change precision and recall for other categories. Also, adding more information about main verb in features improves the performance.

Therefore, a method developed to identify user-defined domain-specific relations facilitates user indexing and provided assistance to bio-curators, data curators and metadata librarians to assign metadata to articles. The method relies on a high precision syntactic parser and named entity recognizer used to identify features. This system is related to our system because it focuses on user-defined domain-specific relation extraction.

3.6 "Semantic Tool for Analysing Unstructured Data"

In this paper [62], semantic and NLP techniques were used to convert unstructured data into a semantically structured form, and it also presents an approach to visualizing those structured data using a graph mechanism. The authors implemented this approach as a web application that can extract and visualize data. To extract information, the system used static ontology [63] that was developed by domain experts for an agriculture and technology application to model information about Organization, Company, Country/Region, Person, and Products. The data repository from which information was extracted contains full news articles, web sites URLs and RSS feeds. The structural analysis was performed on those documents to make them noise free and added metadata information for additional processing. Once the documents are ready for use, linguistic and semantic analyses were performed using static domain ontology to extract information constructs.

In the linguistic analysis, Onto Root Gazetteer [64] was used to annotate documents with entities by using static ontology. This annotated corpus was used as an input to the semantic analysis phase. In this phase, they defined the set of JAPE rules to be executed by the JAPE transducer [65] to extract relevant entities and identify relationships between them.

The method to validate semantic relation constructs generated by semantic analysis was also implemented. To do so, paths connecting pairs of entities and the relationship term between them were identified from the dependency parse tree. Those validated constructs were stored and presented as triplets using the RDF framework [23]. The reason behind using RDF representation was its capability of complex querying on extracted data. Jena [66] was used to store ontology and RDF data. Jena's Reasoning Engine was used to acquire required result for the search query of the user. Finally, the Spring Graph Generator [67] was used to generate spring graph for the visualization of the resultant set.

Hence, the web application developed here uses three algorithms to extract and visualize structured information from free text, and it also provides dynamic and interactive graphs as an output using publicly available tools. The system also gives the user the flexibility to choose options for the information to make visible on the graph.

Thus, the approach presented here was straightforward to implement and achieved competitive performance. Though the intent of developing this system was visualization of extracted information, which not the case for LeaREX, the algorithm used to achieve it is similar to LeaREX.

3.7 Comparative Evaluation

All the systems described above extract relationships from free text. Though all of them use some syntactic or semantic rules or NLP techniques to identify entities and relationships between them, they have several significant differences and similarities among them. Abbreviations used to address systems:

System 1: “Learning Information Extraction Patterns from Examples”

System 2: “Semanticizing Syntactic Patterns in NLP Processing using SPARQL-DL”

System 3: “RelEx—Relation extraction using dependency parse trees”

System 4: “A Framework for Schema-Driven Relationship Discovery from Unstructured Text”

System 5: “Learning User-Defined, Domain-Specific Relations: A Situated Case Study and Evaluation in Plant Science”

System 6: “Semantic Tool for Analysing Unstructured Data”

	Multi-slot extraction	**Approach Used**	**Domain Independent**	**Entities Allowed**	**Validation Mechanism**	**Ontology-based**	**Pattern Formulation**
System 1	Yes	Pattern-based	Yes	Exact Word	No	No	Automatic
System 2	Yes	Pattern-based	Yes	Not Specified	No	Yes	Manual
System 3	Yes	Rule-based	Yes	Exact word, Synonyms	Yes (Filtering)	No	Manual
System 4	Yes	Rule-based	Yes	Simple, Modified, Composite	Yes	No	Manual
System 5	Yes	Lightly supervised machine learning	No	Exact Word	No	No	N/A
System 6	Yes	Rule-based	No	Not Specified	Yes	Yes	Manual

Table 1: Comparison of related systems

Table 1 presents a brief comparative analysis of all systems. All six systems support multi-slot extractions. Multi-slot extraction systems generate a single rule/pattern for all items of interest in a sentence, rather than generating one for each item of interest. System 1 and System 2 followed a pattern-based approach, while System 3, System 4 and System 6 followed a rule-based approach. In contrast, System 5 uses a lightly supervised machine learning approach. Each of these approaches performs better in different scenarios. A pattern-based approach has an advantage over a machine learning approach in that patterns are easy to customize to deal with errors. However, a pattern-based approach requires human involvement for manual inspection of extracted output, which is time consuming. System 5 and System 6 are domain-specific systems, while the other four systems can be applied to any domain with little or no changes in extraction mechanism. Though the intents of System 2 and System 6 are different, they use Ontology to extract relationships. In System 2, ontology was populated for each sentence, and queries were executed against it to match the pattern. In System 6, static domain ontology was used to obtain the results for user queries using the JENA library.

The Entity Identification mechanism is different for many of these systems. System 1 and System 5 allows only an exact word as an entity, while System 4 allows simple (exact word), modified, and composite entities. System 3 considers synonyms of entities along with exact entity words. Thus, granularity of the extraction is not the same. System 4 and System 6 have implemented mechanisms to validate entities and relationships, while System 3 has implemented a filtering mechanism to remove unwanted entities or relationships. System 1 assumes that all needed information is between two entities, and it generates a parse tree for that part of the sentence only. It uses an “On-demand” parser that does not consider semantics that sometimes result in incorrect pattern matches. All other systems parse the whole sentence. System 1 automatically generates patterns, while System 2, System 3, System 4 and System 6 uses a hand-coded set of rules or patterns.

CHAPTER 4

LeaREX IMPLEMENTATION

4.1 Motivation

Since a number of text mining systems have been developed to extract information in a specific domain, we have developed a generalized (domain independent) text mining system. In simple terms, the system can be applied to a new corpora without much human effort to train the system. It automatically builds its knowledge base by using user provided example sentences and a number of sets of words from different categories of user interest. It can also use curatorial feedback and add new knowledge to the system's training set (knowledge base).

In this work, we used Chakrabarti's system [68] as a case study to evaluate our system. Her system takes a full text article as an input, processes it, and uses text mining methods to output the mutation impacts present in the articles. The output mutation impacts were of 4 different types: Positive, Negative, Neutral and Unknown. Positive impact means the mutation has a positive effect on biological function or protein property. Negative impact means it affects the function negatively. Neutral impact represents no effect or a neutral effect on biological function or protein property. In the case of sentences for which the system cannot detect the impact, or whether the sentence is negated, her system could not predict the impact and set the mutation impact as Unknown. To ensure the correctness of the predicted impact, Chakrabarti used the curatorial system CURAMI [69] (developed by her fellow student De). CURAMI provides a user interface to the human curators providing prediction generated by Chakrabarti's system along with some other useful information (such as a whole sentence, the paragraph to which the sentence belongs, other critical words identified by the system, etc.). Curators can agree or disagree with the predicted output, change the predicted output, and also specify the impacts for sentences which were predicted as Unknown. In her work, Chakrabarti manually analyzed curated results and added patterns to her system for corrected (changed) predictions and for newly specified impacts by curators.

In our work, we followed a similar approach for mining the text (explained in detail in later sections), but Chakrabarti's system can be used to predict the impact of protein mutations only, while our system can be used to predict any type of output, in any domain, as per user requirements. Also, the output can be generated in the format of the user given templates. Along with this generalized text mining system, we developed an additional module which can make Chakrabarti's system much faster than before. Since Chakrabarti manually added new knowledge to the training dataset based on curatorial feedback, we created a module which can perform the same task automatically.

4.2 Implementation

The LeaREX system has been implemented in the Java programming language. The flow of how the system works is explained in Chapter 5. We used open source libraries in implementing a part of the system. A PostgreSQL database was used to store the data.

4.2.1 System Architecture

The two main modules of LeaREX are the training and testing modules. Apart from them, an additional module was developed to automatically generate patterns from curatorial feedback of

KiMIner (Chakrabarti's system) generated predictions. The architecture of the system is shown in Figure 8.

The training module and automatic pattern generation module follows the same algorithm. Both uses The Stanford Parser [18] [27] to generate parse tree and typed dependencies for sentences. Next, to generate the smallest subgraph interconnecting critical words, The Grph 2.0.0 library [70] was used. It is a Java library to manipulate graphs. It is easy to use and offers high performance. Once the subgraph was ready, we used the Jena library version 2.11.2 to generate RDF triples followed by the RDF graph, and finally the pattern was generated.

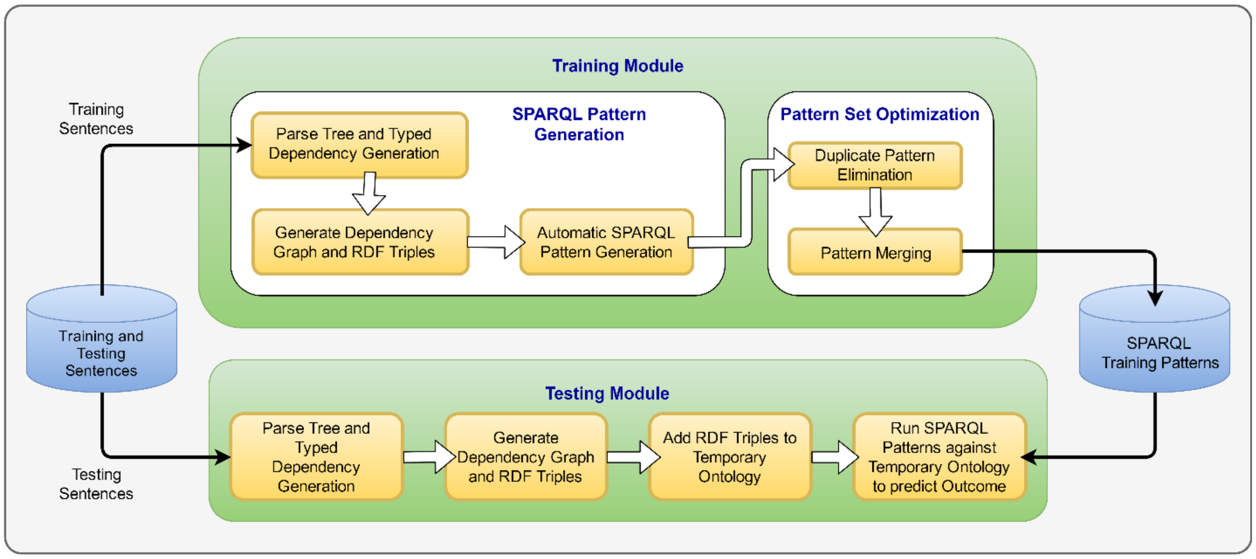

Training Module
Training Sentences
SPARQL Pattern Generation
Parse Tree and Typed Dependency Generation
Generate Dependency Graph and RDF Triples
Automatic SPARQL Pattern Generation
Pattern Set Optimization
Duplicate Pattern Elimination
Pattern Merging
Training and Testing Sentences
SPARQL Training Patterns
Testing Module
Testing Sentences
Parse Tree and Typed Dependency Generation
Generate Dependency Graph and RDF Triples
Add RDF Triples to Temporary Ontology
Run SPARQL Patterns against Temporary Ontology to predict Outcome

Figure 8: LeaREX Architecture

In the testing module, one sentence at a time is taken as an input. The parse tree and typed dependencies generated for the sentence are used to generate RDF triples. Apart from these triples, one triple per critical word existing in the sentence is added to the set of triples in the form of

?Word textMine:type textMine:Category

Critical words are identified using a user provided set of words of different categories. The temporary ontology is created using the Jena library for each sentence, and all the triples generated for that sentence are added to it. Once we have an ontology filled with all the triples, we run all the SPARQL queries (training patterns) against the ontology. If any query matches the pattern existing in the sentence, it will return the result. The result will contain the value of all variables in the *select* clause of the query. The system has the ability to provide the result in the form of user requests. It uses Apache FreeMarker [71], a Java library which generates text output based on an input template(s). Users can provide FreeMarker template as an input.

LeaREX is designed to take example sentences and lists of words per category from the database. We also created a small web interface which allows user to provide that information in a set of Excel files. The interface first accepts excel files, one file per category, containing lists of words belonging to that category. Next, it accepts Excel file containing example sentences with indexes of critical words for each sentence. These data are used to train the system. Once the system is trained, it takes Excel file containing testing sentences and produces the outcome in the form of a template provided by the user.

4.2.2 Database Design

The UML diagram in Figure 9 depicts the design of the database used by LeaREX. The system uses the tables Sentence, Word, WordOccurance, Category, and Pattern. Other tables are used by Chakrabarti's or De's systems. Example sentences were stored in the Sentence table, the index of critical words in each sentence was stored in the WordOccurance table, categories were stored in the Category table, and words from each category were stored in the Word table. Final generated patterns were stored in the Pattern table.

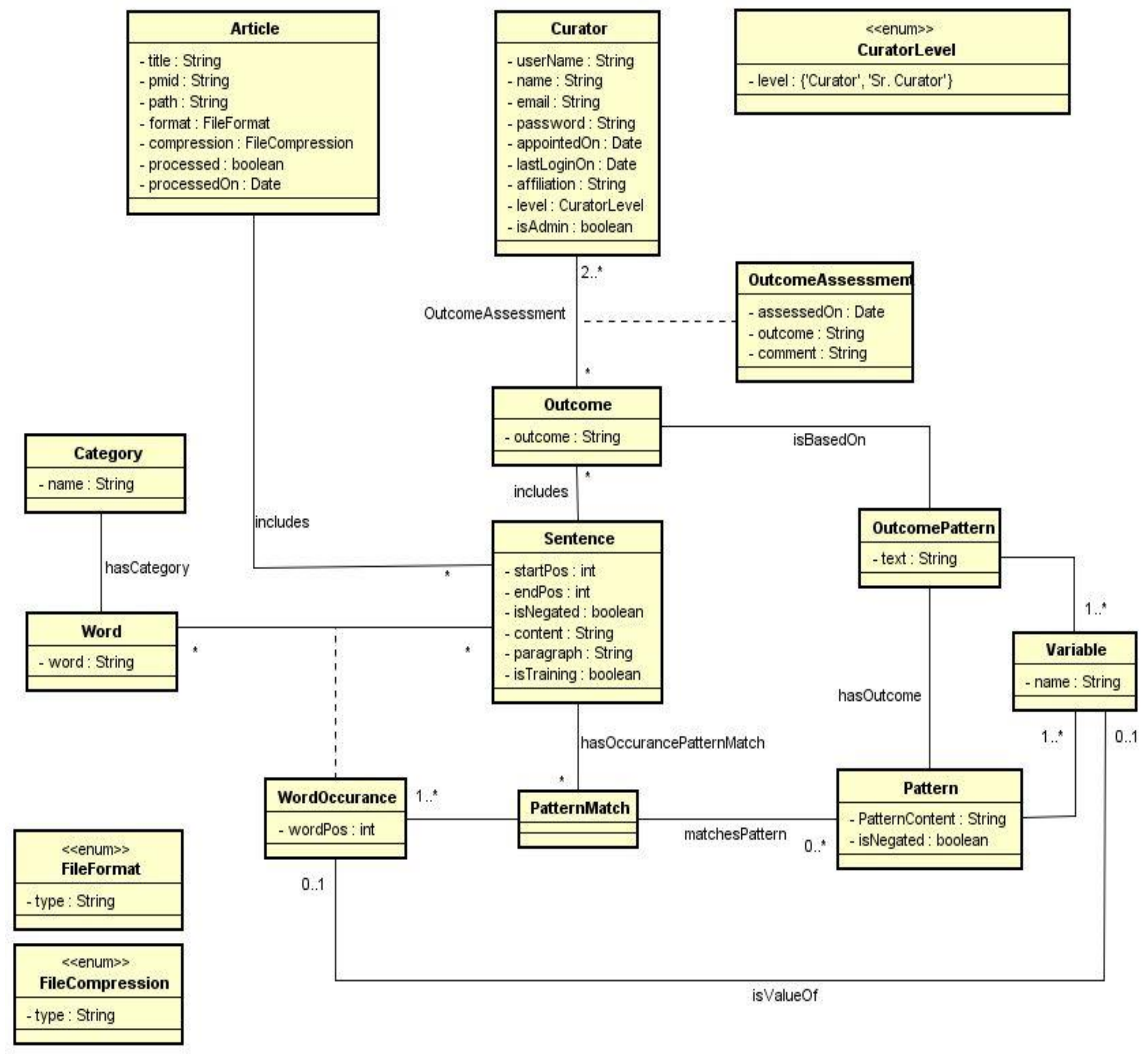


Figure 9: UML diagram of the system's database

CHAPTER 5

RELATIONSHIP EXTRACTION USING LeaREX

The flow of the system is explained in this chapter.

5.1 Training the system

The main task of our system is to extract relationships between concepts existing in a natural language text. As an input to the system, a set of example sentences with the indexes and categories of critical words for each sentence is given. By *Critical word*, we mean that a word that needs to be identified as concept in the sentence. As there might be other critical words which can belong to those categories but is not present in any of the example sentences, the user also provides a list of words for each category. As mentioned above, we used Chakrabarti's system as a case study, we took some number of sentences with indexes of critical words (in this case, Function, Mutation and Impact words) and a list of other Function, Mutation and Impact words (along with the types of impact words which can be positive, negative, neutral or unknown) as an input to the system.

Once we have the input dataset ready, we use it to train the system first. After the system is trained, we can pass any sentence (not present in the training set) to the system, and the system will predict the outcome for that sentence based on example sentences given by the user. We can check the correctness of our system by using curatorial knowledge.

5.1.1 Parse Tree and Typed Dependencies Generation

As the initial task to train the system, sentence level analysis is performed by determining the semantics of the sentences. To determine the semantics, the system generates a parse tree and collapsed typed dependencies for each sentence. The system uses The Stanford Parser to perform these tasks.

The parser first tokenizes the sentence into words and then performs the Part-of-Speech (POS) tagging for each word. Once the sentence is tagged by POS tags, the parser generates a parse tree that shows syntactic structure of the sentence followed by typed dependencies generation. An example sentence with parse tree and typed dependencies is given below.

Sentence:

"*Under restrictive media conditions, only S252W fibroblasts showed enhanced migration*"

Parse Tree:

```
(ROOT
    (S
        (PP (IN Under)
            (NP (JJ restrictive) (NNS media) (NNS conditions)))
        (, ,)
        (NP (RB only) (CD S252W) (NNS fibroblasts))
        (VP (VBD showed)
            (NP (JJ enhanced) (NN migration)))))
```

Typed Dependencies:

amod(conditions-4, restrictive-2)

nn(conditions-4, media-3)

prep_under(showed-9, conditions-4)

advmod(fibroblasts-8, only-6)

num(fibroblasts-8, S252W-7)

nsubj(showed-9, fibroblasts-8)

root(ROOT-0, showed-9)

amod(migration-11, enhanced-10)

dobj(showed-9, migration-11)

We can visualize the sentence based on a parse tree and typed dependencies generated as shown in Figure 10 below.

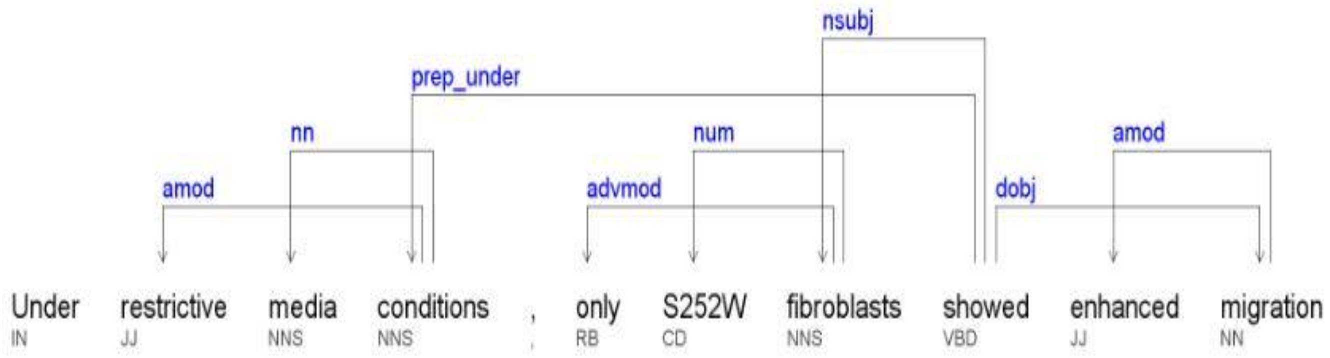


Figure 10: Visualization of the grammatical structure of the sentence "*Under restrictive media conditions, only S252W fibroblasts showed enhanced migration*" using POS tagging, parse tree and typed dependencies.

Since typed dependencies provide grammatical relationships between pairs of words in a sentence, we can represent them in the form of RDF triples [40] and encode them as an RDF graph structure with words as nodes and grammatical relationships between them as edges between nodes.

5.1.2 Pattern Generation

We analyzed example sentences in order to discover repetitive patterns connecting critical words in the sentences. The key task in the training phase is to create a pattern for each example sentence provided by the user. As we can generate an RDF graph for a sentence and the user has provided indexes of critical words in the sentence, we can find the shortest path between each unique pair of critical words and create the smallest subgraph interconnecting all critical words in the sentence.

The critical words with categories for the example sentence "*Under restrictive media conditions, only S252W fibroblasts showed enhanced migration*" are Mutation: S252W, Function: migration, Positive Impact: enhanced.

Figure 11 shows an example sentence with the smallest subgraph interconnecting these three critical words highlighted with thicker lines.

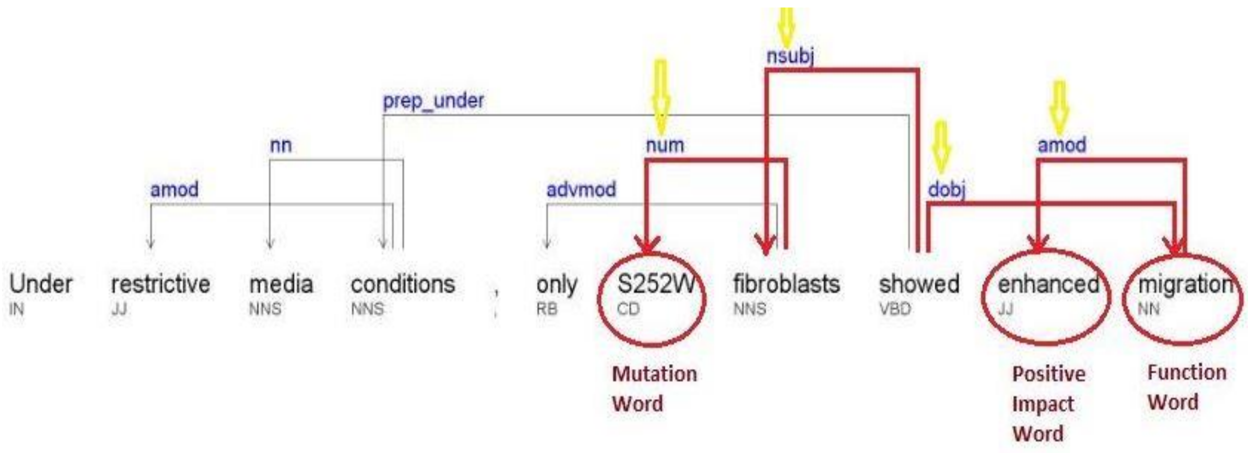


Figure 11: Visualization of the grammatical structure of the sentence "*Under restrictive media conditions, only S252W fibroblasts showed enhanced migration*" with the smallest subgraph interconnecting critical words highlighted.

Based on this smallest subgraph, we can extract the triples which played a role in forming their connection. For the example sentence, those triples will be as follows:

(showed, dobj, migration)

(migration, amod, enhanced)

(showed, nsubj, fibroblasts)

(fibroblasts, num, S252W)

We can also represent these triples as a graphical structure by replacing the intermediate nodes with interconnecting variables.

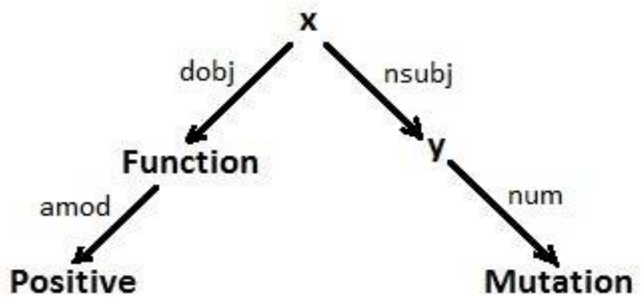
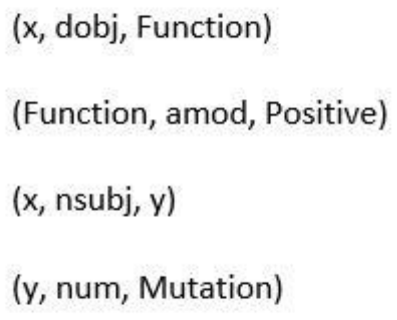


Figure 12: Graphical representation of triples used to interconnect the critical nodes of the sentence "*Under restrictive media conditions, only S252W fibroblasts showed enhanced migration*" with intermediate nodes replaced by variables and critical words by the categories they belong to.

As we are interested in how critical words are related to each other, we can replace intermediate nodes by variables and critical words by the categories they belong to as shown in Figure 12. The new triples representation is shown in Figure 13:



(x, dobj, Function)

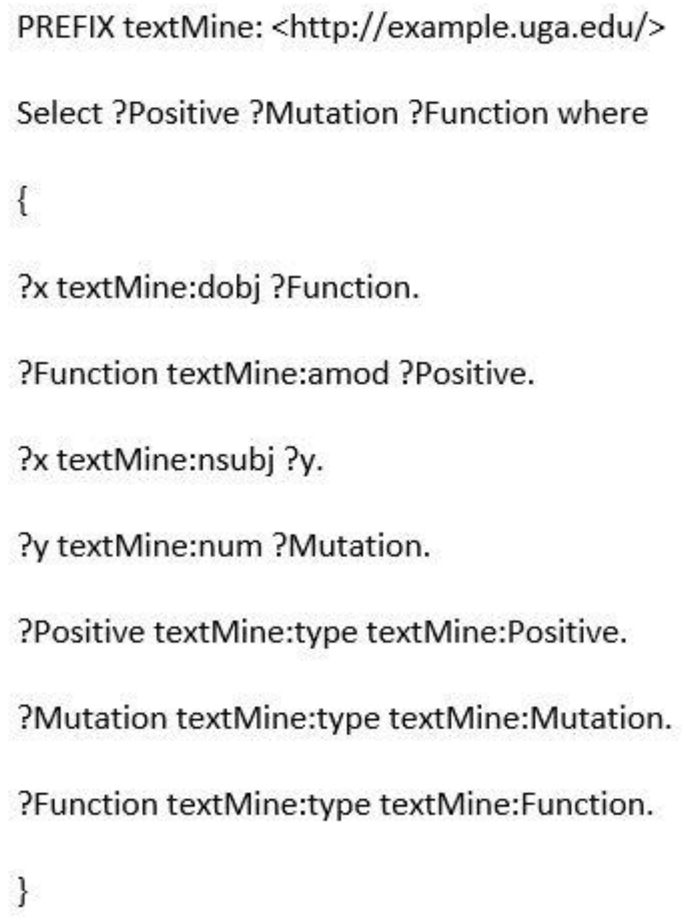
(Function, amod, Positive)

(x, nsubj, y)

(y, num, Mutation)

Figure 13: Triples representation of the sentence "*Under restrictive media conditions, only S252W fibroblasts showed enhanced migration*" after replacing intermediate nodes by variables and critical words by the categories they belong to.

To generate the final pattern, we encode these triples into a SPARQL query as shown in Figure 14 below. We add additional triples to the pattern with the property *textMine:type* to indicate which critical word belongs to which category.



```
PREFIX textMine: <http://example.uga.edu/>

Select ?Positive ?Mutation ?Function where

{

?x textMine:dobj ?Function.

?Function textMine:amod ?Positive.

?x textMine:nsubj ?y.

?y textMine:num ?Mutation.

?Positive textMine:type textMine:Positive.

?Mutation textMine:type textMine:Mutation.

?Function textMine:type textMine:Function.

}
```

Figure 14: Final SPARQL query representation of the sentence "*Under restrictive media conditions, only S252W fibroblasts showed enhanced migration*"

We generate patterns for all user provided example sentences by keeping the number of the patterns generated at a minimum. To minimize the number of patterns generated, we applied two different optimization steps, as explained in the next section. The final patterns are saved in the database as learned patterns which are later used to discover relationships from other testing or unknown sentences.

5.1.3 Pattern Set Optimization

As discussed, our LeaREX system generates a set of patterns, based on a set of training sentences. For some realistic applications, the system may generate a high number of patterns, which will slow down the final extraction task. We have developed a framework for optimizing the final set of patterns by reducing their number and so decreasing the extraction execution time. We applied two approaches as explained below.

Approach 1: Elimination of duplicate patterns

Our analysis of patterns (generated from example sentences) frequently showed three types of similarities, and they were not added to the training pattern set.

1. Exact patterns
2. Patterns with the same set of triples but arranged in different order. These types of patterns have the exact same meaning; the only difference is in the representation of triples. The program generates triples depending on the formation of the sentence. When the same sentence is formed differently, it generates patterns with the same triples arranged in different order. For example,
    i. ?Function.textMine:dobj ?x.
       ?Function textMine:amod ?Positive.
       ?x textMine:nsubj ?Mutation.
    ii. ?x textMine:nsubj ?Mutation.
       ?Function textMine:amod ?Positive.
       ?Function.textMine:dobj ?x.

Below are the two example sentences for which system generates SPARQL patterns with the same set of triples arranged in different order.

Sentence 1: "The mutation Y253F experiences increased electrostatic interactions, Y253H experiences increased electrostatic interactions and decreased vdW when binding to ponatinib"

SPARQL Pattern:

```
PREFIX textMine: <http://example.uga.edu/>
SELECT ?Positive ?Mutation ?Function WHERE     {
?C textMine:nn ?Mutation.
?Positive textMine:nsubj ?C.
?Positive textMine:dobj ?Function.
?Positive textMine:type textMine:Positive.   ?Mutation textMine:type textMine:Mutation.
?Function textMine:type textMine:Function.              }
```

Sentence 2: "The EphA3 G518L lung cancer mutation enhances cis interaction with ephrin A3"

SPARQL Pattern:

```
PREFIX textMine: <http://example.uga.edu/>
SELECT ?Positive ?Mutation ?Function WHERE     {
?C textMine:nn ?Mutation.
?Positive textMine:dobj ?Function.
?Positive textMine:nsubj ?C.
?Positive textMine:type textMine:Positive.   ?Mutation textMine:type textMine:Mutation.
```

?Function textMine:type textMine:Function.   }

The triples in the Blue font in both the sentences' patterns are re-organized. In SPARQL, the order of triples does not matter. They mean the same. So, these types of patterns are not added to the training pattern set again.

3. Patterns with Structurally Identical triples. These types of patterns have the same graphical structure that connects critical nodes, but the intermediate nodes are different. For example,

   i. ?Function.textMine:dobj ?x.

      ?Function textMine:amod ?Positive.

      ?x textMine:nsubj ?Mutation.

   ii. ?Function.textMine:dobj ?y.

      ?Function textMine:amod ?Positive.

      ?y textMine:nsubj ?Mutation.

   Two example sentences for which SPARQL patterns contain structurally identical triples are shown below.

   Sentence 1: "Immune complexes kinase assays showed, as expected, high BRAF activity in YULAC BRAF V600E and YUMAC BRAF V600K cells that was suppressed after treatment with PLX4032"

   SPARQL Pattern:

   PREFIX textMine: <http://example.uga.edu/>

   SELECT ?Positive ?Mutation ?Function WHERE   {

   ?E textMine:nn ?Mutation.

?Function textMine:amod ?Positive.

?D textMine:prep ?E.

?D textMine:prep ?Function.

?Positive textMine:type textMine:Positive. ?Mutation textMine:type textMine:Mutation. ?Function textMine:type textMine:Function. }

Sentence 2: "FDG could furthermore be used to distinguish between BRAF V600E mutant melanomas with high or low sensitivity to PLX4032"

SPARQL Pattern

PREFIX textMine: <http://example.uga.edu/>

SELECT ?Positive ?Mutation ?Function WHERE {

?Function textMine:amod ?Positive.

?A textMine:prep ?Function.

?A textMine:prep ?B.

?B textMine:nn ?Mutation.

?Positive textMine:type textMine:Positive. ?Mutation textMine:type textMine:Mutation. ?Function textMine:type textMine:Function. }

Both the SPARQL patterns show same graphical structure. The only difference is in the organization of triples (shown in Blue font) and the variables used to represent intermediate nodes. As the name of variables does not make any difference in the meaning of SPARQL query, we consider them duplicates.

In this approach, we check the similarities of patterns by comparing their triples. Every time a new query is generated, we compare it with all previously generated queries, and we add it to the training pattern set only if it is not a duplication of any of the previously generated queries. If triples in any pattern are exactly the same as any already generated pattern, no new pattern is added. There is also a possibility that the triples are exactly the same but are not in the exact same order. Since these types of queries have the same meaning in the SPARQL query language, we do not add it. Also, if the structure of the graph generated from the pattern triples has the same structure as any other pattern, these patterns are not added.

Approach 2: Merging similar patterns

Some patterns were not duplicates, but they were sufficiently similar to be merged and still retain the same meaning and structure. Out of all generated patterns, the majority of them have a linear or star structure. Figure 15 and Figure 16 show how the graphical structure appear for the linear and star patterns.

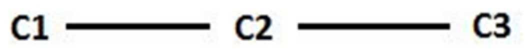


Figure 15: Linear pattern graphical structure

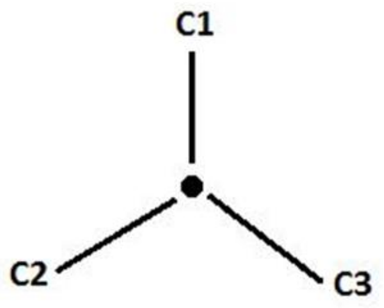


Figure 16: Star pattern graphical structure

Here, C1, C2 and C3 are assumed to be critical words of the sentence. There can be more than three critical words in the sentence. In a linear pattern, all critical words are connected to each other in a linear manner. There might be some intermediate node connecting them. In a star pattern, all critical words are connected to each other via one common intermediate node.

The example sentence along with its dependence graph, and SPARQL pattern for both the linear and star pattern is shown below.

Linear Pattern Example Sentence:

"Since residue V617 is located in the pseudokinase domain of JAK2, this lack of information has hindered a detailed understanding of the mechanism of activation of JAK2 V617F"

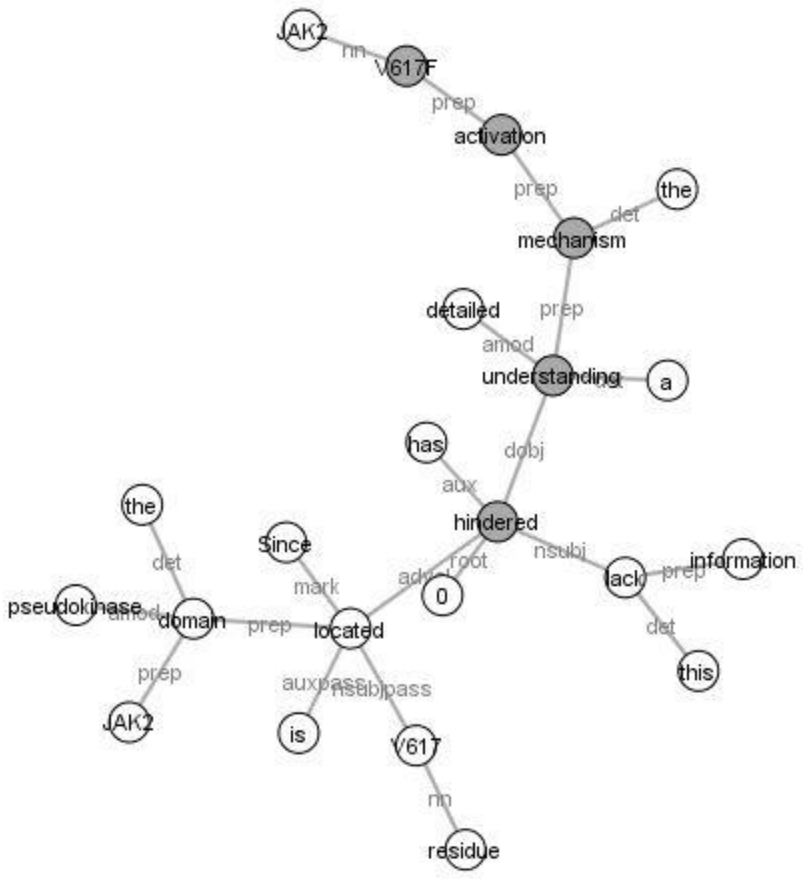


Figure 17: The graph showing grammatical interconnection between words of the sentence "*Since residue V617 is located in the pseudokinase domain of JAK2, this lack of information has hindered a detailed understanding of the mechanism of activation of JAK2 V617F*".

The highlighted subgraph displaying linear interconnection between critical words *V617F* (Mutation word), *activation* (Function word), and *hindered* (Negative Impact word) in the sentence is shown in Figure 17. Here, *understanding* and *mechanism* are the intermediate nodes. The graph is generated using the typed dependencies of the sentence that shows the grammatical relationship between words of the sentence. Each node in the graph is word of the sentence and edges show the grammatical relationship between those words. The simple graphical representation of critical words interconnection in the sentence is shown in Figure 18 below.

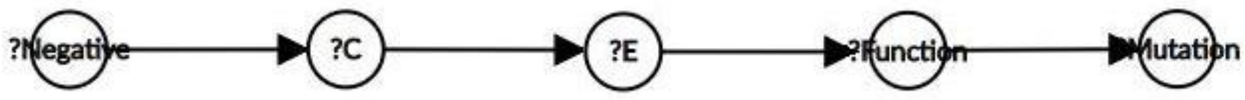


Figure 18: Graphical representation of linear interconnection between critical words of the sentence "*Since residue V617 is located in the pseudokinase domain of JAK2, this lack of information has hindered a detailed understanding of the mechanism of activation of JAK2 V617F*".

The SPARQL pattern generated from the graph is shown below. The Blue font triples shows the interconnection.

```
PREFIX textMine: <http://example.uga.edu/>
SELECT ?Negative ?Mutation ?Function WHERE {
?Negative textMine:dobj ?C.
?C textMine:prep ?E.
```

?Function textMine:prep ?Mutation.

?E textMine:prep ?Function.

?Negative textMine:type textMine:Negative. ?Mutation textMine:type textMine:Mutation.

?Function textMine:type textMine:Function. }

The example sentence to demonstrate star pattern:

"The AKT1 (E17K) mutant can cause increases in the protein levels of Bcl 2 and the phosphorylation of the pro apoptotic protein BAD, resulting in enhanced resistance to apoptosis"

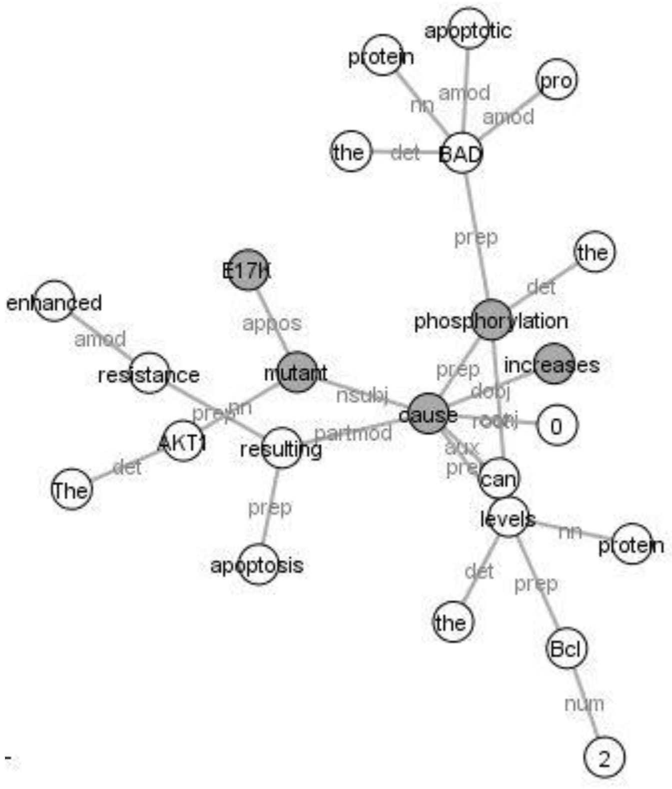


Figure 19: The graph showing grammatical interconnection between words of the sentence "*The AKT1 (E17K) mutant can cause increases in the protein levels of Bcl 2 and the phosphorylation of the pro apoptotic protein BAD, resulting in enhanced resistance to apoptosis*".

The highlighted subgraph in Figure 19 shows that critical words *E17K* (Mutation word), *phosphorylation* (Function word), and *increases* (Positive Impact word) are connected via common intermediate node *cause* in a Star pattern. The graphical representation is shown in the Figure 20.

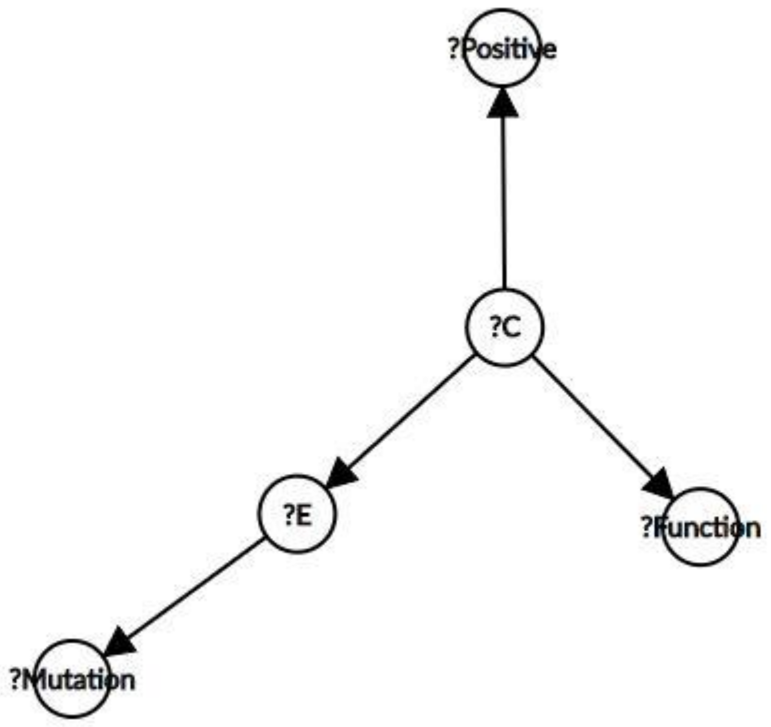


Figure 20: Graphical representation of star interconnection between critical words of the sentence "*The AKT1 (E17K) mutant can cause increases in the protein levels of Bcl 2 and the phosphorylation of the pro apoptotic protein BAD, resulting in enhanced resistance to apoptosis*"

The SPARQL pattern (with interconnection triples in Blue font) is generated as below:

PREFIX textMine: <http://example.uga.edu/>

SELECT ?Positive ?Mutation ?Function WHERE {

?C textMine:prep ?Function.

?C textMine:dobj ?Positive.

?C textMine:nsubj ?E.

?E textMine:appos ?Mutation

?Positive textMine:type textMine:Positive. ?Mutation textMine:type textMine:Mutation.

?Function textMine:type textMine:Function. }

In this approach, we are considering only a linear structured patterns for merging. Before applying this approach, we changed the representation of patterns generated, as shown in the example below. Initially, for ease of understanding, only triples are shown as only representation of triples is changed. The PREFIX clause, select clause and additional triples indicating the category of critical words remain the same. Later, the example sentence with the complete SPARQL query is provided.

Triples in the original pattern generated:

?Function.**textMine:dobj ?x**.

?x textMine:nsubj ?Mutation.

?Function textMine:amod ?Positive.

In the new representation of the query, the triples will appear as:

?Function **textMine:dobj/textMine:nsubj** ?Mutation

?Function textMine:amod ?Positive.

We combine the paths (connecting critical words) by sequence path operator "/". Sequence operator shows that one path is followed by another path in the graph. The paths that contain intermediate node(s) to connect critical words get combined. This combination of paths retains the meaning of SPARQL query and ease the merging process. We added this new form of query to the training pattern set which are used to see if it is eligible for merging with any previously generated pattern. For merging, we first ensure that both patterns are linear, then we check if the categories of the critical words in both are the same. Next, we check if the path between any two categories (critical words) in both the patterns are same, and we then merge the path between the other two categories. The example below will assist in understanding this explanation. First, triples of three different linear type queries are given, and how they are merged in new pattern is shown.

1. ?Function textMine:prep ?Negative. ?Function textMine:amod ?Mutation.
2. ?Function textMine:prep ?Negative. ?Function textMine:nsubj ?Mutation.
3. ?Function textMine:prep ?Negative. ?Function textMine:prep/textMine:prep ?Mutation.

Triples in Merged Pattern:

?Function textMine:prep ?Negative.

?Function (textMine:amod) | (textMine:nsubj) | (textMine:prep/textMine:prep) ?Mutation.

In the triples of three linear patterns, first we check if critical words in the patterns belong to the same three categories (Negative, Function and Mutation). Then, we see if the path between Function and Negative category words is the same (shown as triple in blue font). Next, the path between other triples (shown in red font) are merged by the "|" operator, which signals an alternative path expression. By this merging, we represented three different patterns by one that can match all three sentence patterns. Shown below is how one sentence (used to demonstrate linear pattern structure) is merged with another sentence's pattern.

Sentence 1: *"Since residue V617 is located in the pseudokinase domain of JAK2, this lack of information has hindered a detailed understanding of the mechanism of activation of JAK2 V617F"*

SPARQL Pattern:

```
PREFIX textMine: <http://example.uga.edu/>
SELECT ?Negative ?Mutation ?Function WHERE {
?Negative textMine:dobj ?C.
?C textMine:prep ?E.
?Function textMine:prep ?Mutation.
?E textMine:prep ?Function.
?Negative textMine:type textMine:Negative. ?Mutation textMine:type textMine:Mutation.
?Function textMine:type textMine:Function. }
```

New representation of SPARQL pattern is:

```
PREFIX textMine: <http://example.uga.edu/>
SELECT ?Negative ?Mutation ?Function WHERE {
?Negative textMine:dobj/textMine:prep/textMine:prep ?Function.
?Function textMine:prep ?Mutation.
?Negative textMine:type textMine:Negative. ?Mutation textMine:type textMine:Mutation.
?Function textMine:type textMine:Function. }
```

The triples in Blue font shows how the representation is changed from actual one. In this sentence, Negative and Function nodes are connected via two intermediate nodes, and their paths get combined. In the example sentence 2 below, all three critical nodes are connected directly to each other without any intermediate node. So, their representation does not get changed.

Sentence 2: "*In pre clinical studies, JAK2 inhibitors reduced the proliferation of JAK2 V617F and MPL W515L mutant cells and attenuated disease development in murine models of MPN*"

SPARQL Pattern:

```
PREFIX textMine: <http://example.uga.edu/>
SELECT ?Negative ?Mutation ?Function WHERE {
?Negative textMine:iobj ?Function.
?Function textMine:prep ?Mutation.
?Negative textMine:type textMine:Negative. ?Mutation textMine:type textMine:Mutation.
?Function textMine:type textMine:Function. }
```

As the critical words' categories (Function, Mutation, and Negative) in both the patterns are same, and the path between Function and Mutation words is same, these patterns are eligible for merging. New merged pattern is shown below. The blue font triples get changed in new pattern.

```
PREFIX textMine: <http://example.uga.edu/>
SELECT ?Negative ?Mutation ?Function WHERE {
?Negative (textMine:dobj/textMine:prep/textMine:prep) | (textMine:iobj) ?Function.
?Function textMine:prep ?Mutation.
```

?Negative textMine:type textMine:Negative. ?Mutation textMine:type textMine:Mutation. ?Function textMine:type textMine:Function. }

5.2 Testing the system

Once the system is trained, the set of SPARQL query patterns is placed in the database. Next, the system is tested with other sentences. In the testing phase, we first identify the critical words in the sentence based on the list of user provided words (with category) as an input. Then, we find the relationships between each of these critical words by finding the shortest path between each of them as explained in section 4.2.2. We obtain the set of triples from the shortest path list. We create a temporary ontology, add these triples to it, and run all the training SPARQL patterns against that ontology. If any pattern matches the testing sentence's pattern, it will return the result. The variables in the SPARQL query are returned in the result.

5.3 Automatic Pattern Generation from curatorial knowledge

Apart from the Generalized text mining system, we also developed an additional module to automatically generate SPARQL query patterns from curation of predictions generated by Chakrabarti's system. Patterns were generated only for those predictions which were changed from "Unknown" to some known (Positive, Negative or Neutral) impact by curators. As curators provided the indexes of critical words for each of those sentences, we used the same approach that we used to train our system to generate the pattern using a parse tree and typed dependencies (explained in detail in Section 5.1) and which were later added to the training patterns. This will make the system more accurate over time.

CHAPTER 6

EXPERIMENTS AND EVALUATION

The main goal of our system is to generate a pattern correctly for each sentence. To verify the correctness of patterns generated, we took 262 sentences which were predicted to have *unknown* mutation impact by Chakrabarti's system and were changed to some known impact (Positive, Negative or Neutral) by curators. Patterns were generated for those 262 sentences and used as the training dataset. Those sentences were passed through the LeaREX to check if they successfully matched the pattern for each sentence, and the correct output (mutation impact) was predicted. As a result, each sentence was matched to a correct pattern. Furthermore, we randomly took 1100 sentences which were correctly predicted by Chakrabarti's system and curated by curators and applied those 262 patterns on them. It matched patterns for 329 sentences correctly out of 1100 sentences. Along with 262 patterns generated by LeaREX, we added 32 more patterns used by Chakrabarti to predict the mutation impact on the training dataset. Next, with a total of 294 (262 + 32) patterns in the training dataset, 1100 sentences were tested, and all the sentences were matched to one or more patterns. These results are summarized in Table 2 below.

No. of Sentences	**No. of Patterns used to predict Outcome**	**No. of sentences that matched Pattern correctly**
262	262	262
1100	262	329
1100	294	1100

Table 2: Experimental results of Sentence to Pattern Matching

A flowchart illustrating the evaluation of LeaREX is shown in Figure 21 below.

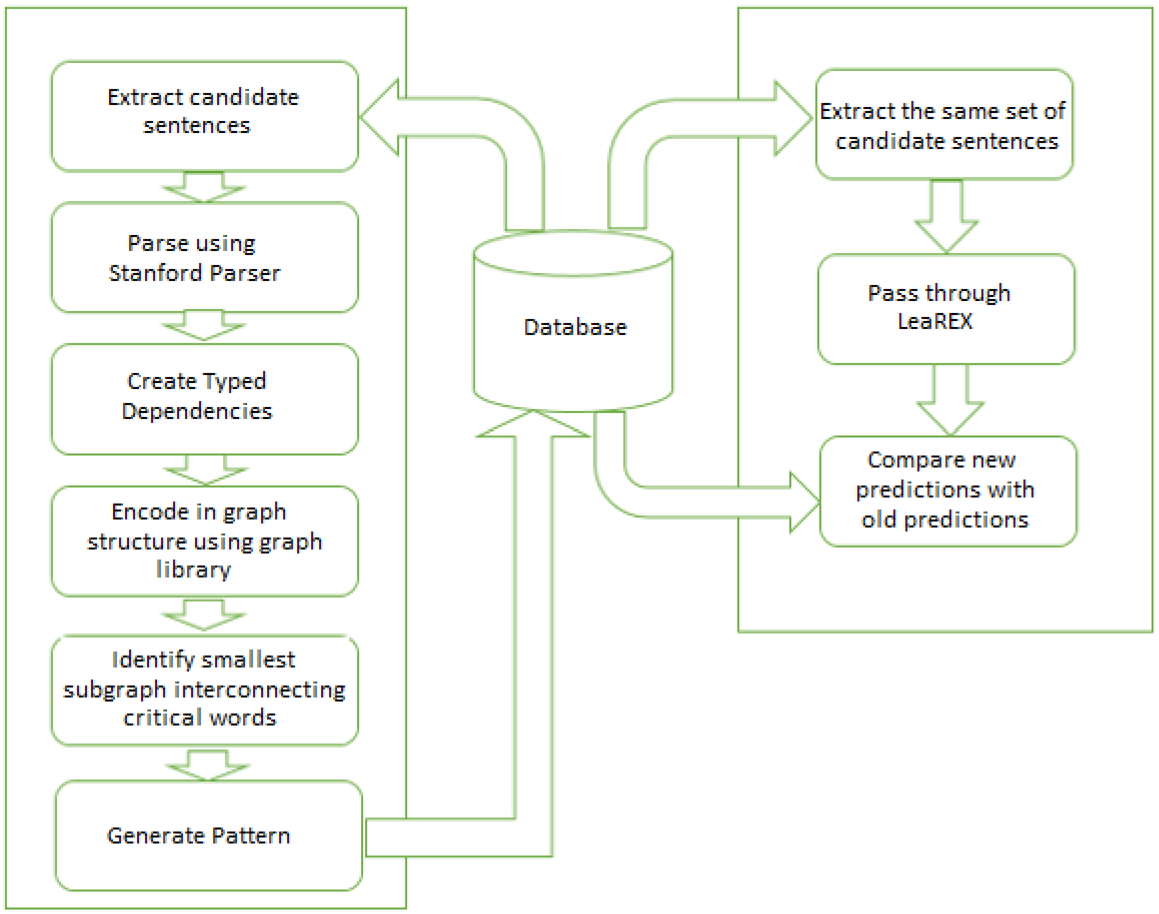

Extract candidate sentences
Parse using Stanford Parser
Create Typed Dependencies
Encode in graph structure using graph library
Identify smallest subgraph interconnecting critical words
Generate Pattern
Database
Extract the same set of candidate sentences
Pass through LeaREX
Compare new predictions with old predictions

Figure 21: LeaREX evaluation flowchart

Along with this testing, we did another experiment that used those same 1100 sentences to see how system improves accuracy as it learns more patterns. Out of 1100 sentences, we used 1000 sentences for testing and remaining 100 sentences for testing. First, we used only 100 sentences (of 1000 training sentences) to train the system and checked the accuracy against 100 testing sentences. We kept adding 100 sentences in each experiment and checked how many testing sentences matched the pattern(s). In the graph below, X-axis shows the number of sentences used for training and Y-axis shows how many testing sentences matched the pattern(s).

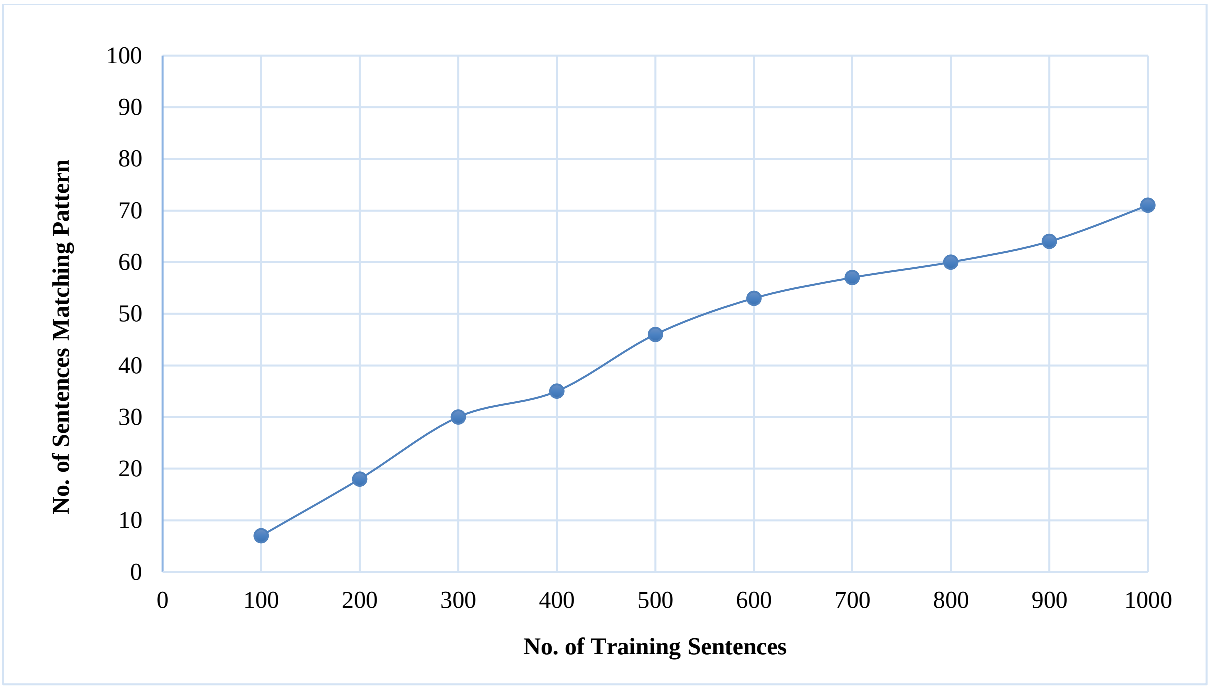


Figure 22: Graph representing System Accuracy based on 1000 Training and 100 Testing Sentences

When system is trained with all 1000 training sentences, it got 71% accuracy. It shows that as we add more sentences to train system, it gives better results.

After evaluating the training and testing modules of LeaREX, we applied two approaches discussed in Chapter 5 (Section 5.1.3) to optimize patterns. We took 2 sets of 500 and 1 set of 1000 sentences (Union of those two sets) and applied both the approaches on them. The results of applying these approaches are shown in Table 3.

No. of Sentences	Approach 1 – Duplicate Pattern Elimination			Approach 2 - Merging	Reduction in Patterns (%)
	Exact Patterns	Re-Organized Patterns	Similar Structured Patterns	Merged Patterns	
500	22	10	30	102	33%
500	20	13	27	108	34%
1000	67	46	75	221	41%

Table 3: Results of pattern optimization approaches

By applying both the approaches, we removed 164 (22+10+30+102), 168 (20+13+27+108), and 409 (67+46+75+221) patterns out of first (500 sentences), second (500 sentences) and third set (1000 sentences), respectively.

This shows that more than 1/3 of the total patterns were removed, which sped up the pattern identification and relationship extraction process in this instance. Future runs should display similar increases.

We used the same 1000 sentences and checked how gradually system can reduce the number of patterns. The same way that we did to test system's accuracy, we first took 100 sentences and checked the reduction in patterns and kept adding 100 more sentences in each run and observed the reduction in number of patterns after applying both the optimization approaches. The X-axis in the graph below shows the number of training sentences used to train the system and Y-axis shows how much percentage of the total training patterns are actually distinct which are used to predict the outcome.

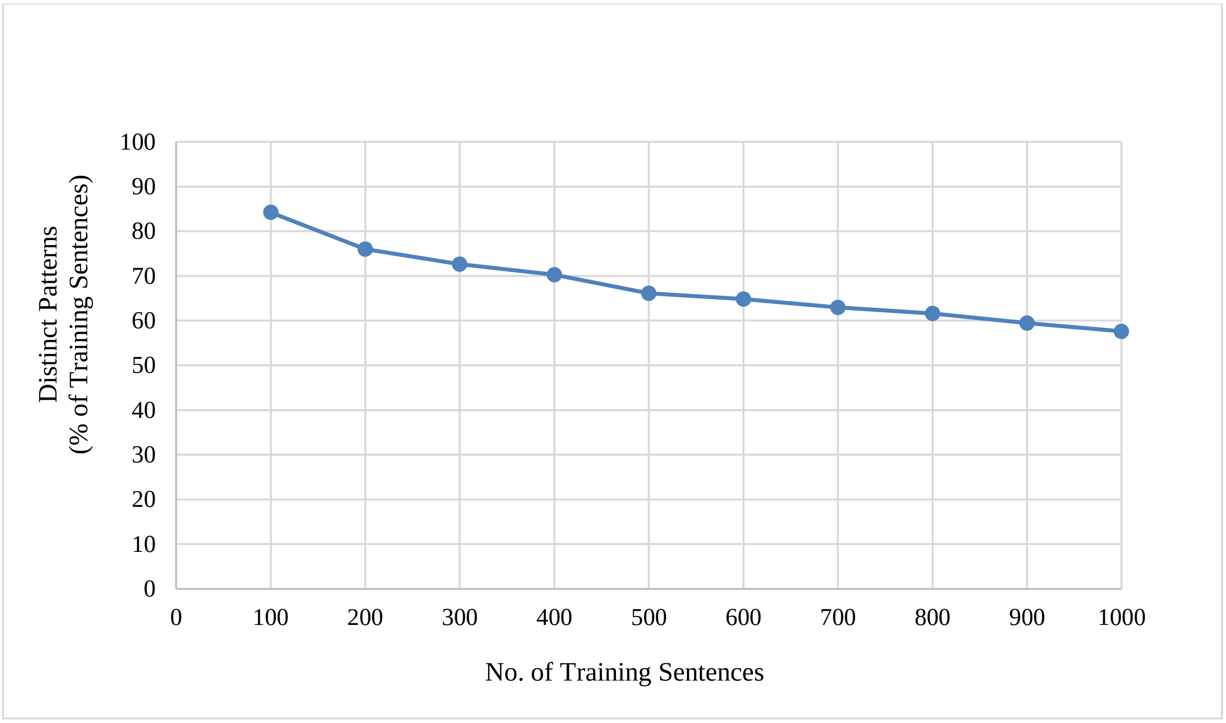


Figure 23: Graph representing Reduction in Patterns based on 1000 Training Sentences

This shows that as we increase the number of training sentences, reduction in number of patterns also increases. In other words, number of distinct patterns decreases as we add more data to train the system.

CHAPTER 7

CONCLUSION AND FUTURE WORK

We implemented a generalized text mining system which uses Natural Language Processing techniques to extract relationships from text. The system can be used to extract relationships in any domain, but as a case study, we used data from Chakrabarti's system to extract the impact of mutations. Our system automatically generates patterns to identify relationships. Many similar text mining systems exist, but the majority of them have been developed for specific domains and very few generate patterns automatically. Also, the approaches used to generate patterns are different from ours. We use sentence-level grammatical analysis and SPARQL to detect and store grammatical patterns, which is similar to Chakrabarti's system, but the novelty here is that the system is domain independent and generates patterns automatically without any human intervention except proving example sentences and lexical lists of words for different categories. Also, the pattern optimization is automatic. The system also gives users the flexibility to provide a freemarker template to obtain the output in the desired format. A small web interface was also created which allows user to provide example sentences and list of words in excel files. The only restriction is that system takes the complete sentence only as an input. It does not predict the output for sentences which cannot be parsed by The Stanford Parser.

As we do not consider non-linear patterns in the second approach to merging patterns, we plan to discover other frequently occurring pattern structures (such as star) and optimize more patterns. Also, new pattern optimization approaches can be explored, which will improve pattern identification process. A PostgreSQL database is used to store patterns (SPARQL queries). SPARQL Inferencing Notation (SPIN) [72], a SPARQL based constraint language for Semantic Web, combines ideas from rule based systems and object oriented languages, allows the linking of OWL and RDFS classes with reusable SPARQL queries. Thus, rather than storing patterns in the database, we can use SPIN and link patterns in the ontology itself, which will probably reduce the pattern matching time. Also, we can introduce the concept of constraint/rule matching using SPIN, which helps to filter out results and adds another feature to the system.

REFERENCES

1. "Text Mining", https://en.wikipedia.org/wiki/Text_mining
2. "Document pre-processing", https://en.wikipedia.org/wiki/Document_processing
3. Toutanova, Kristina, et al. "Feature-rich part-of-speech tagging with a cyclic dependency network." Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1. Association for Computational Linguistics, 2003.
4. "Information Retrieval" https://en.wikipedia.org/wiki/Information_retrieval
5. "Document Classification" https://en.wikipedia.org/wiki/Document_classification
6. "Data Mining" https://en.wikipedia.org/wiki/Data_mining
7. "Natural Language Processing", https://en.wikipedia.org/wiki/Natural_language_processing
8. "Information Extraction", https://en.wikipedia.org/wiki/Information_extraction
9. "SPARQL", http://www.w3.org/TR/rdf-sparql-query/
10. "What is text mining?", https://ischool.syr.edu/infospace/2013/04/23/what-is-text-mining/
11. "How Big Is The Internet", https://metamend.com/archive/education/internet-growth/
12. "Text Mining Handbook", https://wtlab.um.ac.ir/images/e-library/text_mining/The%20Text%20Mining%20HandBook.pdf
13. "PubMed", http://www.ncbi.nlm.nih.gov/pubmed
14. "PubMed Central", http://www.ncbi.nlm.nih.gov/pmc/

15. “Penn Tree Bank Project”, https://www.cis.upenn.edu/~treebank/

16. “Java example for using Stanford POSTagger”, http://www.programcreek.com/2012/07/java-example-for-using-stanford-postagger/

17. “Porter Stemmer implementation”, http://svn.apache.org/repos/asf/lucene/java/branches/flex_1458/src/java/org/apache/lucene/analysis /PorterStemmer.java

18. “Stanford Dependency Parser”, http://nlp.stanford.edu/software/stanford-dependencies.shtml#About

19. “Parsing”, https://en.wikipedia.org/wiki/Parsing

20. “Parse Tree”, https://en.wikipedia.org/wiki/Parse_tree

21. “Named Entity Recognition”, https://en.wikipedia.org/wiki/Named-entity_recognition

22. “Relationship Extraction”, https://en.wikipedia.org/wiki/Relationship_extraction

23. “Resource Description Framework (RDF)”, https://en.wikipedia.org/wiki/Resource_Description_Framework

24. “Ontology”, https://en.wikipedia.org/wiki/Ontology_(information_science)

25. Michael Collins, Probabilistic Context Free Grammar, http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf

26. “Parse trees visualization”, https://kwtrnka.wordpress.com/2011/01/14/parse-trees-visualization/

27. Marie-Catherine de Marneffe and Christopher D. Manning, Stanford Typed Dependency Manual, http://nlp.stanford.edu/software/dependencies_manual.pdf

28. Online Stanford Parser, http://nlp.stanford.edu:8080/parser/

29. Universal typed dependencies, http://universaldependencies.org/docs/en/dep/

30. Fellbaum, Christiane. WordNet. Blackwell Publishing Ltd, 1998.

31. Martinez-Cruz, Carmen, Ignacio J. Blanco, and M. Amparo Vila. "Ontologies versus relational databases: are they so different? A comparison." Artificial Intelligence Review 38.4 (2012): 271-290.

32. Berners-Lee, Tim, James Hendler, and Ora Lassila. "The semantic web." Scientific american 284.5 (2001): 28-37.

33. “Ontology language”, https://en.wikipedia.org/wiki/Ontology_language

34. “RDF Schema”, https://en.wikipedia.org/wiki/RDF_Schema

35. “Web Ontology Language”, https://en.wikipedia.org/wiki/Web_Ontology_Language

36. “LOOM (ontology)”, https://en.wikipedia.org/wiki/LOOM_(ontology)

37. “Knowledge Interchange Format”, https://en.wikipedia.org/wiki/Knowledge_Interchange_Format

38. “F-logic”, https://en.wikipedia.org/wiki/F-logic

39. “RDF Triples”, https://www.w3.org/TR/n-triples/

40. “OWL Web Ontology Language Guide”, https://www.w3.org/TR/2004/REC-owl-guide-20040210/#Introduction

41. Jiang, Jing. "Information extraction from text." Mining text data. Springer US, 2012. 11-41.

42. Huffman, Scott B. "Learning information extraction patterns from examples." International Joint Conference on Artificial Intelligence. Springer Berlin Heidelberg, 1995.

43. Brill, Eric. "Some advances in transformation-based part of speech tagging." arXiv preprint cmp-lg/9406010 (1994).

44. Chinchor, Nancy, and Beth Sundheim. "MUC-5 evaluation metrics." Proceedings of the 5th conference on Message understanding. Association for Computational Linguistics, 1993.

45. Vitucci, Nicola, et al. "Semanticizing Syntactic Patterns in NLP Processing Using SPARQL-DL Queries." OWLED. 2012.

46. Sirin, Evren, and Bijan Parsia. "SPARQL-DL: SPARQL Query for OWL-DL." OWLED. Vol. 258. 2007.

47. Baker, Collin F., Charles J. Fillmore, and John B. Lowe. "The berkeley framenet project." Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1. Association for Computational Linguistics, 1998.

48. Fundel, Katrin, Robert Küffner, and Ralf Zimmer. "RelEx—Relation extraction using dependency parse trees." Bioinformatics 23.3 (2007): 365-371.

49. Nédellec, Claire. "Learning language in logic-genic interaction extraction challenge." Proceedings of the 4th Learning Language in Logic Workshop (LLL05). Vol. 7. 2005.

50. Peri, Suraj, et al. "Human protein reference database as a discovery resource for proteomics." Nucleic acids research 32.suppl 1 (2004): D497-D501.

51. Smith, L., Thomas Rindflesch, and W. John Wilbur. "MedPost: a part-of-speech tagger for bioMedical text." Bioinformatics 20.14 (2004): 2320-2321.

52. Ngai, Grace, and Radu Florian. "Transformation-based learning in the fast lane." Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies. Association for Computational Linguistics, 2001.

53. Hanisch, Daniel, et al. "ProMiner: rule-based protein and gene entity recognition." BMC bioinformatics 6.Suppl 1 (2005): S14.
54. Fundel, Katrin, and Ralf Zimmer. "Gene and protein nomenclature in public databases." Bmc Bioinformatics 7.1 (2006): 1.
55. Ramakrishnan, Cartic, Krys J. Kochut, and Amit P. Sheth. "A framework for schema-driven relationship discovery from unstructured text." International Semantic Web Conference. Springer Berlin Heidelberg, 2006.
56. NLM, Medical Subject Heading (MeSH), The National Library Of Medicine, Bethesda, MD.
57. NLM, Unified Medical Language System (UMLS), The National Library Of Medicine, Bethesda, MD.
58. Tsuruoka, Y. and J.i. Tsujii, Bidirectional Inference with the Easiest-First Strategy for Tagging Sequence Data, in Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing. 2005, Association. p. 467-474.
59. Tsuruoka, Y. and J.i. Tsujii, Chunk Parsing Revisited, in Proceedings of the 9th International Workshop on Parsing Technologies (IWPT 2005). 2005. p. 133-140. 19
60. Lucic, Ana, and Catherine Blake. "Learning user-defined, domain-specific relations: A situated case study and evaluation in plant science." Proceedings of the Association for Information Science and Technology 52.1 (2015): 1-12.
61. Tamayo, Pablo, et al. "Oracle data mining." Data mining and knowledge discovery handbook. Springer US, 2005. 1315-1329.

62. Jadhao, Harish, Dr Jagannath Aghav, and Anil Vegiraju. "Semantic Tool for Analysing Unstructured Data." International Journal of Scientific & Engineering Research 3.8 (2012).

63. Heiner Stuckenschmidt, Frank van Harmelen, "Information Sharing on the Semantic Web," Springer, 2005.

64. Damljanovic, Danica, Valentin Tablan, and Kalina Bontcheva. "A Text-based Query Interface to OWL Ontologies." LREC. 2008.

65. Cunningham, Hamish, Diana Maynard, and Valentin Tablan. "JAPE: a Java annotation patterns engine." (1999).

66. "An Introduction to Jena RDF API", http://jena.sourceforge.net/tutorial/RDF API/index.html.

67. M. Mudholkar, S.Peswani, S. Kela, J. Aghav, H. Jadhao, P. Gaikwad, "Generating Spring Graph To Infer RDF Data", ICITCS, India, 2012

68. Bhargabi Chakrabarti (2015). KiMIner:TEXT MINING THE IMPACT OF PROTEIN KINASE MUTATIONS (Master's thesis). Retrieved from https://getd.libs.uga.edu/pdfs/chakrabarti_bhargabi_201505_ms.pdf

69. Reshmi De (2015). CURAMI : A SYSTEM FOR CURATION OF MUTATION IMPACTS FROM TEXT MINING (Master's thesis). Retrieved from https://getd.libs.uga.edu/pdfs/de_reshmi_201505_ms.pdf

70. Grph Package, http://www.i3s.unice.fr/~hogie/grph/

71. "Apache Freemarker", https://en.wikipedia.org/wiki/FreeMarker

72. "SPARQL Inferencing Notation (SPIN)", http://spinrdf.org/