KNOWLEDGE EXTRACTION FROM UNSTRUCTURED DATA FOR SEARCH DRIVEN ANALYTICS

PROJECT REPORT

Submitted by

Rohit Mapakshi

(Register Number: 15CS145)

Under the guidance of

Dr. J. Kumaran@Kumar

Assistant Professor

Department of Computer Science and Engineering

&

Mr. Sridhar K Sundaram

Member Leadership Staff, Zoho Search

Zoho Corporation

to the Pondicherry University, in partial fulfillment of the requirement for the award of degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PONDICHERRY ENGINEERING COLLEGE

PUDUCHERRY – 605 014

April 2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PONDICHERRY ENGINEERING COLLEGE

PUDUCHERRY – 605 014.

BONAFIDE CERTIFICATE

This is to certify that the Project work titled **"Knowledge Extraction from Unstructured Data for Search Driven Analytics"** is a bonafide work done by **Rohit Mapakshi (15CS145)** in partial fulfilment for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of the **Pondicherry University** and that this work has not been submitted for the award of any other degree of this/any other institution.

Project Guide **Head of the Department**

(Dr. J. Kumaran@Kumar) **(Dr. R. MANOHARAN)**

*Submitted for the University Examination held on*_____

Internal Examiner **External Examiner**

[Project Completion Certificate from ZOHO]


[Evaluation Sheet from
ZOHO]

ACKNOWLEDMENT

I am deeply indebted to **Dr. J. Kumaran@Kumar**, Assistant Professor, Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry, for his valuable guidance throughout this project.

I would like to express my sincere thanks to **Mr. Sridhar K Sundaram**, Member Leadership Staff, Zoho Corporation, for his valuable guidance throughout this project.

I also express my heart-felt gratitude to **Dr. R. Manoharan,** Professor & Head (CSE) for giving constant motivation in succeeding my goal.

With profoundness I would like to express my sincere thanks to **Dr. P. Dananjayan,** the Principal, Pondicherry Engineering College, for his kindness in extending the infrastructural facilities to carry out my project work successfully.

I would be failing in my duty if I do not acknowledge the efforts of the Project Coordinator, **Dr. J.Jayabharathy**, and the Project Review Panel members, viz. **Dr.K.Vivekanandan, Dr.R. Manoharan, Dr.E.Ilavarasan, Dr.S.Lakshmana Pandian, Dr.J. Jayabharathy, Dr. N. Sivakumar and Dr. J. Kumaran@Kumar** for shaping our ideas and constructive criticisms during project review.

I also express my thanks to all the **Faculty** and **Technical Staff** members of the CSE department for their timely help and the **Central Library** for facilitating useful reference materials.

I would be failing in my duty if I don't acknowledge the immense help extended by my friends, who have always been with me in all my trials and tribulations and encouraging me to complete.

Rohit Mapakshi

ABSTRACT

The project focuses on parsing unrestricted text, which is useful for business analytics applications but requires parsing methods that are both robust and efficient. The project aims at building a language-independent system for data-driven dependency parsing that can be used to induce a parser for from a treebank sample in a simple yet flexible manner. The developed system should handle the ambiguities in unstructured sentences and integrated into the system without affecting the current functionality of the system. It also aims to achieve robust, efficient and accurate parsing for a without language-specific enhancements and with rather limited amounts of training data. For training the parser model various treebanks available in the Universal Dependencies (UD) framework were used.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	BONAFIDE CERTIFICATE	***ii***
	PROJECT COMPLETION CERTIFICATE	***iii***
	PERFORMANCE REPORT	***iv***
	ACKNOWLEDGEMENT	***v***
	ABSTRACT	***vi***
	LIST OF FIGURES	***ix***
	LIST OF TABLES	***ix***
	LIST OF ABBREVIATIONS	***x***
1	**INTRODUCTION**	**1**
	1.1 OVERVIEW	1
	1.2 OBJECTIVE OF STUDY	2
	1.3 MOTIVATION	2
	1.4 ORGANIZATION OF CHAPTERS	2
2	**LITERATURE REVIEW**	**3**
	2.1 TECHNIQUES	3
	2.1.1 Tokenization	3
	2.1.2 Lemmatization And Stemming	3
	2.1.3 Named Entity Recognition	4
	2.1.4 Parts Of Speech Tagging	4
	2.1.5 Maximum Entropy	5
	2.1.6 Shallow Parsing Or Chunking	5
	2.1.7 Dependency Parsing	6
	2.1.8 Parsing Algorithms	9
	2.1.9 Feature Models	10
	2.1.10 Discriminative Machine Learning	11
	2.1.11 Support Vector Machines	12
3	**EXISTING WORK**	**13**
	3.1 EXISTING ARCHITECTURE	13
	3.2 AUTO-SUGGEST ENGINE	13
	3.3 NLP CORE ENGINE	14

	3.3.1 Sentence Detector And Tokenizer	14
	3.3.2 Named Entity Recognition	15
	3.3.3 Parts Of Speech Assignment	18
	3.4 INTERPRETATION ENGINE	19
	3.5 REPORT GENERATION ENGINE	20
	3.6 LIMITATIONS	20
4	**PROPOSED SYSTEM**	**21**
	4.1 PROPOSED ARCHITECTURE	21
	4.2 AGGREGATE NER	21
	4.3 DEPENDENCY PARSING	21
	4.3.1 Feature Models	23
	4.3.2 Learning Mode	24
	4.3.3 Parsing Mode	25
	4.3.4 Universal Dependencies	26
5	**EXPERIMENTAL RESULTS**	**28**
	5.1 PLATFORM AND TOOLS	28
	5.2 OUTPUT	28
	5.2.1 Dependency Graph for the given query	28
	5.2.2 Full Stack Output For The Query	29
	5.2.3 Accuracy for MaltParser Model	30
	5.2.4 Accuracy based on Projectivity	30
	5.2.5 Accuracy for each dependency relation	31
	5.2.6 Sample 1 from ZOHO Analytics	32
	5.2.7 Sample 2 from ZOHO Analytics	32
6	**CONCLUSION AND FUTURE ENHANCEMENTS**	**33**
	6.1 CONCLUSION	33
	6.2 FUTURE ENHANCEMENTS	33
	REFERENCES	**34**

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
Figure 2.1	DEPENDECY GRAPH FOR AN ENGLISH SENTENCE	7
Figure 2.2	MAXIMUM-MARGIN STRATEGY	11
Figure 3.1	ARCHITECTURE DIAGRAM OF SEARCH DRIVEN ANALYTICS	13
Figure 3.2	NLP PIPELINE IN EXISTING SYSTEM	14
Figure 4.1	MODIFIED NLP PIPELINE IN PROPOSED SYSTEM	21
Figure 5.1	DEPENDENCY GRAPH	28
Figure 5.2	FULL STACK OUTPUT	29
Figure 5.3	ACCURACY	29
Figure 5.4	PRECISION, RECALL AND FSCORE	30
Figure 5.5	PRECISION, RECALL AND FSCORE FOR EACH DEPENDENCY RELATION	30
Figure 5.6	WEB SAMPLE 1	31
Figure 5.7	WEB SAMPLE 2	31

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO
Table 3.1	LIST OF SUPPORTED NER CLASSES	17
Table 3.2	LIST OF POS TAGS	18
Table 4.1	CoNLL-X REPRESENTATION FOR A SAMPLE SENTENCE	24
Table 4.2	LIST OF UNIVERSAL DEPENDENCIES	26
Table 4.3	CLASSIFICATION OF DEPENDENCY RELATIONSHIPS	27

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
AI	ARTIFICAIL INTELLIGENCE
BI	BUSINESS INTELLIGENCE
DEPREL	DEPENDENCY RELATION
IT	INFORMATION TECHNOLOGY
NDFA	NON-DETERMINISTIC FINATE AUOMATA
NER	NAMED ENTITY RECOGNITION
NLP	NATURAL LANGUAGE PROCESSING
POS	PARTS OF SPEECH
SQL	STRUCTURED QUERY LANGUAGE
SVM	SUPPORT VECTOR MACHINE
UD	UNIVERSAL DEPENDENCIES

CHAPTER I

INRODUCTION

Natural Language Processing has been on the rise. From simple text to speech conversion to powering intelligent voice assistants, we have seen various applications of NLP. Voice assistants like Siri and Alexa have become widely popular among users. Today, NLP has become a core component of all search engines. In the BI domain, NLP helps by improving the user experience in drawing insights from a huge collection of data. Business Intelligence tools are becoming increasingly popular and are being relied upon to understand data and NLP provides a simpler user interface and ease with BI tools. For example, businesses rely on IT experts for report creation and modification of existing reports, with NLP this task flow can be directly between the BI tool and its user. Search engines make use of NLP to transform natural language into queries which fetches information from data warehouses. The same workflow is applied by BI tools. Users can speak to tools which then uses NLP to convert spoken words into SQL or any technical language queries that can fetch the required information from a database. What happens behind the scene is a complex flow of queries fetching data from different sources. NLP makes the process as simple as asking your virtual assistant app for the weather report. Zoho's AI assistant Zia has ongoing research on these technologies.

1.1 OVERVIEW

In English, some words such as adverbs can freely be moved within a sentence without affecting its correctness or meaning. The project addresses the problem of "free word order". This is the important reason due to which a dependency parser should be preferred over classical phrase-based methods. Free word order has a negative impact on the accuracy of conventional parsing methods. This project tries to address these problems by creating a dependency parser with the help of MaltParser. The training data which is used consists of sentences annotated with dependency trees. MaltParser supports several input formats, but in this project, data is in the CoNLL representation format. It is currently the de facto standard for data-driven dependency parsing. LIBSVM is used to induce a classifier for predicting the next transition given a feature representation of the current parser configuration. LIBSVM makes feature optimization a little easier due to the kernel that implicitly add conjoined features. The feature model essentially consists of three groups of features, all of which are centered around the two target tokens, i.e., the tokens that are being considered for a dependency in the current

configuration. These are Part-of-speech tags in a wide window around the target tokens, Word forms in a narrower window around the target tokens and Dependency labels on outgoing arcs of the target tokens.

1.2 OBJECTIVE OF STUDY

Most of the challenges surrounding functional NLP are rooted in the complexity of human language. It is difficult to interpret the meaning of a text or email when subtle cues like body language, tone, inflection, facial expression, and volume are omitted from the equation. Simple word phrases can take on different meanings dependent upon the sentence structure. The main objective is to establish dependency links between words that are close to semantic relationships needed for the interpretation of user queries. In doing so the process of parsing should be made straight forward and overcome the challenges of shallow parsing that the current system has been built upon. Finally, the vast syntactical complexities that exist in languages is to be reduced by following this approach

1.3 MOTIVATION

Natural language processing (NLP) and search-driven analytics can connect the most potent business minds with the right data. Organizations have already started implementing NLP for a variety of tasks beyond just raw text or speech processing. Voice assistant apps and devices have become immensely popular with more and more popping up every now and then. In the coming years, we will see more apps and devices using NLP to take user interaction and experience to a whole new level. This cutting-edge technology has a huge potential to transform Business analytics.

1.4 ORGANIZATION OF THE CHAPTERS

Chapter 1 provides background, history and applications of NLP in the field of Business Intelligence. Chapter 2 provides an overview of available techniques in NLP and machine learning. Chapter 3 explains the existing work, the techniques used and the limitations in the work. Chapter 4 describes a proposed work model based on the better method. Chapter 5 discusses the results of the simulations of existing work. Chapter 6 gives a brief summary of the work done and presents some future enhancement work.

CHAPTER II

LITERATURE REVIEW

2.1 TECHNIQUES

2.1.1 Tokenization

The process of chopping the given sentence into smaller parts (tokens) is known as tokenization. Tokenization is a two-stage process: first, sentence boundaries are identified, then tokens within each sentence are identified.

Tokenizers can be implemented in three ways:

- Whitespace Tokenizer - A whitespace tokenizer, non-whitespace sequences are identified as tokens.
- Simple Tokenizer - A character class tokenizer, sequences of the same character class are tokens.
- Learnable Tokenizer - A maximum entropy tokenizer, detects token boundaries based on probability model.

Most part-of-speech taggers, parsers and so on, work with text tokenized in this manner. It is important to ensure that the tokenizer produces tokens of the type expected by your later text processing components.

2.1.2 Lemmatization and Stemming

Lemmatisation is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form. Lemmatisation depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighbouring sentences or even an entire document.

Stemming is the process of reducing inflected words to their word-stem base or root form, generally a written word form. The stem need not be identical to the morphological root

of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root.

2.1.3 Named Entity Recognition (NER)

In any text document, there are particular terms that represent specific entities that are more informative and have a unique context. These entities are known as named entities, which more specifically refer to terms that represent real-world objects like people, places, organizations, and so on, which are often denoted by proper names. A naive approach could be to find these by looking at the noun phrases in text documents. Named entity is a technique used in information extraction to identify and segment the named entities and classify or categorize them under various predefined classes. But NER systems aren't just doing a simple dictionary lookup. Instead, they are using the context of how a word appears in the sentence and a statistical model to guess which type of noun a word represents.

2.1.4 Parts of Speech Tagging

Parts of speech (POS) are specific lexical categories to which words are assigned, based on their syntactic context and role. Usually, words can fall into one of the following major categories.

- Noun: This usually denotes words that depict some object or entity.
- Verb: Verbs are words that are used to describe certain actions, states, or occurrences.
- Adjective: Adjectives are words used to describe or qualify other words, typically nouns and noun phrases.
- Adverb: Adverbs usually act as modifiers for other words including nouns, adjectives, verbs, or other adverbs.

Besides these four major categories of parts of speech, there are other categories that occur frequently in the English language. These include pronouns, prepositions, interjections, conjunctions, determiners, and many others. Furthermore, each POS tag like the noun (N) can be further subdivided into categories like singular nouns (NN), singular proper nouns (NNP), and plural nouns (NNS).

The process of classifying and labelling POS tags for words called parts of speech tagging or POS tagging. POS tags are used to annotate words and depict their POS, which is really

helpful to perform specific analysis, such as narrowing down upon nouns and seeing which ones are the most prominent, word sense disambiguation, and grammar analysis.

2.1.5 Maximum Entropy

Many problems in natural language processing can be re-formulated as classification problems [1] in which the task is to observe some linguistic" context" b∈B and predict the correct linguistic class a∈A. This involves constructing a classifier cl : B → A, which in turn can be implemented with a conditional probability distribution p, such that p(a|b) is the probability of "class" a given some "context" b. Contexts in NLP tasks usually include at least words and the exact context depends on the nature of the task for some tasks the context b may consist of just a single word, while for others b, may consist of several words and their associated syntactic labels. Large text corpora usually contain some information about the co-occurrence of a's and b's but never enough to reliably specify p(a|b) for all possible (a,b) pairs, since the words in b are typically sparse. The challenge is then to find a method for using the partial evidence about the a's and b's to reliably estimate the probability model p.

Maximum entropy probability model offers a clean way to combine diverse pieces of contextual evidence in order to estimate the probability of a certain linguistic class occurring with a certain linguistic context.

2.1.6 Shallow parsing or Chunking

It is an analysis of a sentence which first identifies constituent parts of sentences and then links them to higher order units that have discrete grammatical meanings (noun groups or phrases, verb groups, etc.). While the most elementary chunking algorithms simply link constituent parts on the basis of elementary search patterns, approaches that use machine learning techniques can take contextual information into account and thus compose chunks in such a way that they better reflect the semantic relations between the basic constituents. That is, these more advanced methods get around the problem that combinations of elementary constituents can have different higher level meanings depending on the context of the sentence. It is a technique widely used in natural language processing. However, chunking fails as the complexity of the language increases.

2.1.7 Dependency Parsing

The dependency grammars are quite important in contemporary speech and language processing systems. In dependency parsing, phrasal constituents and phrase-structure rules do not play a direct role. Instead, the syntactic structure of a sentence is described solely in terms of the words (or lemmas) in a sentence and an associated set of directed binary grammatical relations that hold among the words. A major advantage of dependency grammars is their ability to deal with languages that are morphologically rich and have a relatively free word order. For example, word order in Czech can be much more flexible than in English; a grammatical object might occur before or after a location adverbial. A phrase-structure grammar would need a separate rule for each possible place in the parse tree where such an adverbial phrase could occur. A dependency-based approach would just have one link type representing this particular adverbial relation. Thus, a dependency grammar approach abstracts away from word-order information, representing only the information that is necessary for the parse.

Dependency graphs

A dependency graph expresses links between individual words, rather than breaking the sentence up into smaller phrases as in a constituency tree. Figure 2.1 shows an example of a dependency graph. The links, or arcs, denote dependency relations between two words, called head and dependent. The arc labels signify the type of dependency relation; in Figure 2.1 below, for example, there is an arc labelled NMOD from the second to the first node. This arc itself denotes that the first node is the dependent of the second node, while the label indicates that the relation is that of a nominal modifier. For the sentence $x = (w0,w1,...,wn)$, the dependency graph is thus defined as $G = (V,A)$ [2] where:

- $V = \{0,1,...,n\}$ is the set of nodes, each corresponding to the linear position of a word in the sentence x. The node 0 is artificial and corresponds to an artificial ROOT word inserted at the beginning of the sentence in position 0. All nodes corresponding to actual words in the sentence x are called token nodes.
- $A \subseteq V \times L \times V$ is the set of labelled, directed arcs between the nodes. Each arc is a triple (i,l,j) where i and j are nodes, and $l \in L$ the arc label.

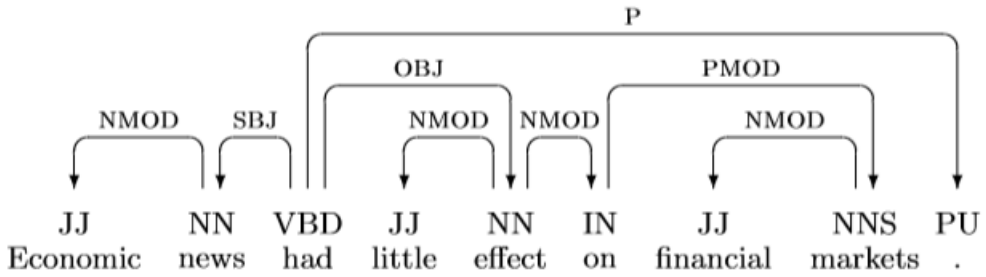

Figure 2.1 Dependency graph for an English sentence

In the dependency graph shown in Figure2.1, not only arcs have labels, but the words (including punctuation) making up the sentence are also marked. Here, the example sentence is taken from a treebank where each word is tagged; JJ, NN, VBD etc. are all examples of part-of-speech tags (POS tags). The tags, if any, are thus dependent on the treebank used. For the purpose of this project, a token is defined as a word and any associated tags. A dependency graph is defined as well-formed if it satisfies some basic requirements [2]:

- Each node is allowed at most one head (another node). This is also called the single-head constraint.
- The node 0 must be the root, which means that there can be no arc (i, l,0) where i is some token node and l is the arc label.

Projectivity

The notion of Projectivity imposes an additional constraint that is derived from the order of the words in the input, and is closely related to the context-free nature of human languages. An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence. A dependency tree is then said to be projective if all the arcs that make it up are projective. In order for the graph to be projective, for every arc $(i,l,j) \in A$ and every node k, if $i < k < j$ or $i > k > j$, there must also be some subset of arcs forming a path from i to k.

Data-Driven Models

Within text parsing in general, and here dependency parsing in particular, two different approaches can be distinguished [3]: grammar-driven and data-driven parsing. Grammar-driven parsing requires a formal grammar defining the well-formed analyses of the parsed

language. An appropriate parsing algorithm is applied to the input text, producing an analysis which is allowed by the grammar and therefore considered correct. Data-driven parsing is the focus of this project. It is a very versatile approach to syntactic parsing since it can be applied to any language as long as there is an annotated treebank available. Data-driven parsing requires no pre-existing grammar; in order to derive correct analyses of sentences, it applies to the treebank some method for inductive learning. Inductive learning is a form of 'learning by example, and as such, is dependent on the number of examples available for training. The two most widely used models for data-driven dependency parsing: graph-based and transition-based parsing.

- In graph-based dependency parsing, for each input sentence all possible, valid dependency graphs are assigned a score. Some method for inductive learning is used to determine how to score each dependency graph. The system then performs a search for the highest-scoring dependency graph. Depending on the parsing algorithm, projective and/or non-projective structures can be captured.
- In transition-based dependency parsing, the parser instead makes a series of transitions to get from an initial configuration to a terminal configuration, representing a finished dependency graph. Inductive learning is used to guide the parser in deciding what transition to choose in cases where more than one transition is possible.

According to [3], one of the potential advantages of data-driven approaches to natural language processing is that they can be ported to new languages, provided that the necessary linguistic data resources are available.

Inductive dependency parsing

Inductive learning is a form of 'learning by example, and as such, is dependent on the number of examples available for training. The framework of inductive dependency parsing can be viewed as based on three elements, all of which will be discussed further below:

(1) deterministic parsing algorithms,

(2) history-based feature models, and

(3) discriminative machine learning (here exemplified by support vector machines).

2.1.8 Parsing algorithms

There are two groups of parsing algorithms, [4] and [5]. Each group contains two parsing algorithms. They both process input from left to right, making use of a stack for partially processed nodes, and a queue for tokens remaining in the input. More so than the arc-eager algorithm, used in this project, the arc-standard version is similar to a basic shift-reduce algorithm.

Transition system

for transition-based parsing, a transition system consists of a set of parser configurations and the transitions between them. The transition system defines:

- a set of parser configurations C,
- an initialization function cs,
- a set of terminal configurations Ct $\subseteq$ C, and
- a set of transitions T.

Parser configuration

For Nivre's [4] algorithms, each of the parser configurations c $\in$ C for a given input sentence x = (w0,w1,...,wn) can be characterized as c = (σ,τ,A) [2]. The parser configuration is explained in the following way:

- σ is a stack of partially processed token nodes.
- τ is the list of nodes representing the remaining words in the input sentence.
- A is a set of dependency arcs, such that the dependency graph for the sentence x is defined as G = (V,A) where V = [0,1,...,n].

Each non-terminal parser configuration for the sentence x = (w0,w1,...,wn) thus defines a (partially-built) dependency graph.

Initialization function

The initialization function cs maps the sentence x = (w0,w1,...,wn) to a configuration with τ = [1,...,n] and where the stack σ is empty. Initially, all token nodes are also set as dependents of the root node, since they have not yet been attached to a head.

Terminal configurations

A parser configuration is called terminal if τ is empty, i.e. if the parser has reached the end of the sentence and there are no more words to be parsed. A sentence may have more than one terminal configuration.

Transitions

Nivre's arc-eager algorithm [4] provides the following four transitions, each available for every dependency type l [2]:

- Right-arc(l): Creates an arc (i,l,j) with the dependency type l from the node i on the top of the stack to the next node j in the input list, pushing j onto the stack. In order to satisfy the single-head constraint, the node j may not already have a head. The arc (i,l,j) is added to the set A.
- Left-arc(l): Creates an arc (j,l,i) with the dependency type l from the next input node j to the token node i on the top of the stack, popping i from the stack. In order to satisfy the single-head constraint, the node j may not already have a head. Since the root node 0 may not have a head, i cannot be the head. The arc (j,l,i) is added to the set A.
- Reduce: Pops the stack, removing the topmost node i. This is permitted only when the head of the topmost token node is another token node and not the root node. It is not allowed to pop the topmost node unless it has been assigned a head, because it cannot be attached later.
- Shift: Shifts (pushes) the next input token node onto the stack. This is permitted as long as there are token nodes remaining to be processed.

2.1.9 Feature models

When training a classifier, the goal is to have the classifier derive the correct transitions from the annotated sentences in the treebank. In order to do this, a one-to-one mapping is defined from an input string and its dependency graph to a sequence of parser transitions. Every transition is dependent on all previous transitions made by the parser, and the history contains complete information about these. The idea of a history-based feature model is to make the learning problem tractable by extracting only certain relevant parts of the history. A theoretically ideal history-based feature model may therefore be defined as one that includes all parts of the history that contribute to the parser accuracy, while at the same time excluding

all parts of the history that do not. The feature model can be defined as a set of feature functions, where each function corresponds to an attribute of the (current) parser state. When applying these feature functions to the parser state at any given time, a sequence of features is returned. This sequence of features is called a feature vector, and it is then used by the classifier for predicting the next transition. It is common for feature models to include both static and dynamic features. Dynamic features may be different depending on the parser state when the feature function was applied; static features remain the same throughout parsing. Examples of static features are the part-of-speech tag or word form of a token. Dynamic features, for example dependency types (arc labels) are determined at some point during the parsing process and thus are not the same for every parser state.

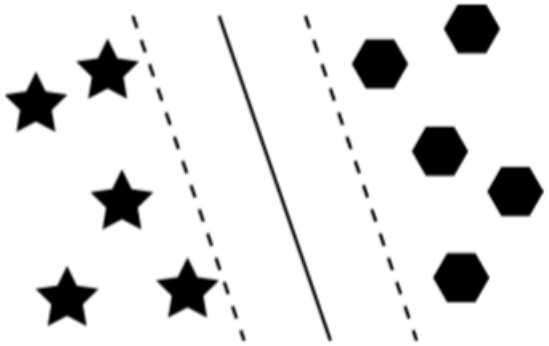

Figure 2.2 Maximum-margin strategy.

2.1.10 Discriminative machine learning

The task of classification generally means predicting a class y given an observation x, or in other words, deciding what class y that the observation x belongs to. In a very basic version of binary classification, for each x the classifier would simply return the value 0 or the value 1, representing the class y. In transition-based dependency parsing, classification is used to determine the parser decision whenever several transitions are possible out of a given parser configuration. The class y thus represents a parser decision, whereas the observation x is the parser history. For every observation x, the classifier assigns a score to each class y, and predicts the highest-scoring class as the class that x belongs to.

2.1.11 Support vector machines

Support vector machines are commonly used for binary classification. The basic idea of SVM, introduced by [6] is to separate the two classes by as wide a margin as possible, assuming that the data is linearly separable. Linear separation can be exemplified by drawing a line on a graph of the training data, separating the two classes, given that the input feature vector is of dimensionality 2. An example of this is seen in Figure 2.2. If the input is of a higher dimensionality, the two classes may be separated by a hyperplane. SVM can be used to classify data that is not linearly separable. A cost, or penalty parameter may be introduced in order to allow for some misclassification. This C parameter controls the trade-off between allowing for some data points to be misclassified and enforcing the margin between the classes. A higher C value enforces a harder margin between the classes. This can result in so-called overfitting, obtaining a more accurate model for that particular set of training data, but one that may not perform well when used with another data set. A lower C value means more misclassifications are allowed, perhaps resulting in a lower accuracy. It is thus important to select an appropriate value for this parameter in order to achieve good accuracy without overfitting the model. Further, the so-called kernel trick may be used to transform the linear SVM algorithm into one that is able to handle non-linearly separable data. The kernel trick means replacing every calculation of a dot product between two vectors with a kernel function.

There are several such functions; some examples are the polynomial, the radial basis, and the sigmoid kernels. The result is the input feature vectors mapped to a higher-dimensional space, where the data is linearly separable. Multi-class classification is also possible with SVM. Two methods for this, using many binary classifiers, are one-versus-one and one-versus-all. With the one-versus-all method, for n classes as many classifiers are trained in order to separate each class from the rest. With the one-versus-one method, one classifier is trained for each pair of classes. A voting system may be used to discriminate between the classes.

CHAPTER III

EXISTING WORK

3.1 EXISTING ARCHITECTURE

The workflow can be explained through the architecture diagram given below.

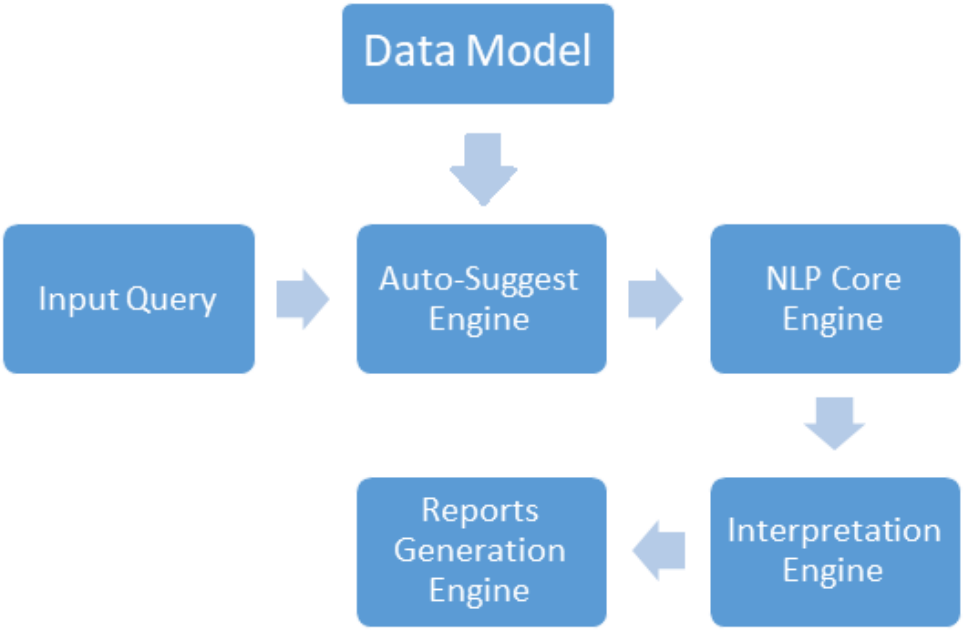

Figure 3.1 Architecture Diagram of Search Driven Analytics

3.2 AUTO-SUGGEST ENGINE

In the auto-suggest engine the data model of the database is loaded as the meta data. The meta data includes previous suggestions, history, column types etc. While the user is typing the query contextual suggestions are provided using the meta data and NDFA templates. The NDFA templates consists of directed graphs of some pre-defined sentence styles. The auto suggest engine also provides alternate entities for already typed entities.

3.3 NLP CORE ENGINE

The architecture of NLP core engine is given in the figure below.

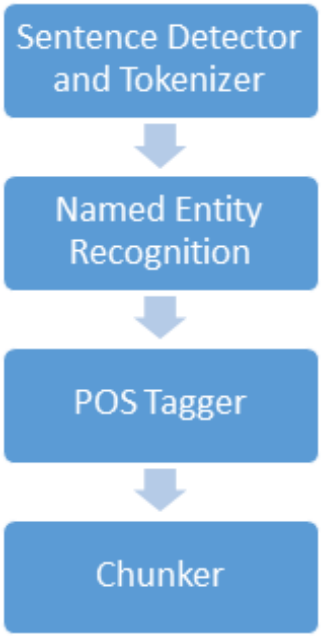

Figure 3.2 NLP Pipeline in existing system

3.3.1 Sentence Detector and Tokenizer

It may appear that postulating a sentence boundary for every occurrence of a potential sentence-final punctuation mark, such as . , ? ;and !, is sufficient to accurately divide text into sentences. However, these punctuation marks are not used exclusively to mark sentence breaks. For example, embedded quotations may contain any of the sentence-ending punctuation marks and . is used as a decimal point in e-mail addresses to indicate ellipsis and in abbreviations. Both ! And ? Are somewhat less ambiguous but appear in proper names and may be used multiple times for emphasis to mark a single sentence boundary.

Contextual Predicates The token containing the symbol which marks a putative sentence boundary is called as the Candidate. The portion of the Candidate preceding the potential sentence boundary is called the Prefix and the portion following it is called the Suffix. The templates used are:

- The Prefix

- The Suffix
- Whether the Prefix or Suffix is on the list of induced abbreviations
- The word left of the Candidate
- The word right of the Candidate
- Whether the word to the left or right of the Candidate is on the list of induced abbreviations

Features for sentence and token detection

For each potential sentence boundary token (., ?, and !) we wish to estimate a joint probability distribution p of it and its surrounding context occurring as an actual sentence boundary. The probability distribution used here is a maximum entropy model identical to the equation:

$$p(a|b) = \frac{1}{Z(b)} \prod_{j=1}^{k} a_j^{f_j(a,b)}$$

The contextual predicates deemed useful for sentence-boundary detection, which we described earlier, are encoded in the model using features. For example, a useful feature might be:

$$f_j(a,b) = \begin{cases} 1 & if\ Prefix(b) = Mr\ and\ a = no \\ 0 & otherwise \end{cases}$$

This feature will allow the model to discover that the period at the end of the word Mr. seldom occurs as a sentence boundary.

3.3.2 Named Entity Recognition

The named entity recognition (NER) task involves identifying noun phrases that are names, and assigning a class to each name. The maximum entropy classifier is used to classify each word as one of the following: the beginning of a NE (B tag), a word inside a NE (C tag), the last word of a NE (L tag), or the unique word in a NE (U tag).

The basic features used by the Maximum Entropy model can be divided into two classes: local and global [5]. Local features of a token *w* are those that are derived from the sentence containing *w*. Global features are derived by looking up other occurrences of *w* within the same document. The *w−i* refers to the ith word before *w*, and w+I refer to the ith word after *w*. The features used are similar to those used in (Chieu and Ng, 2002b). Local features include:

Local Features

First Word, Case, and Zone for English, each document is segmented by simple rules into 4 zones: head-line (HL), author (AU), dateline (DL), and text (TXT). To identify the zones, a DL sentence is first identified using a regular expression. The system then looks for an AU sentence that occurs before DL using another regular expression. All sentences other than AU that occur before the DL sentence are then taken to be in the HL zone. Sentences after the DL sentence are taken to be in the TXT zone. If no DL sentence can be found in a document, then the first sentence of the document is taken as HL, and the rest as TXT. If *w* starts with a capital letter (i.e., *initCaps*), and it is the first word of a sentence, a feature (*firstword-initCaps,zone*)is set to 1. If it is *initCaps* but not the first word, a feature (*initCaps*, zone) is set to 1. If it is the first word but not *initCaps*,(*firstword-notInitCaps, zone*) is set to 1. If it is made up of all capital letters, then (*allCaps, zone*) is set to 1. If it starts with a lower case letter, and contains both upper and lower case letters, then (*mixedCaps, zone*) is set to 1. A token that is all Caps will also be *initCaps*.

Case and Zone of w+1 and w−1 If w+1 (orw−1) is *initCaps*, a feature(*initCaps, zone*)NEXT (or*(initCaps, zone*)PREV) is set to 1, etc.

Case Sequence Suppose both *w−1* and *w+1* are *init-Caps*. Then if *w* is *initCaps*, a feature I is set to 1, else a feature N I is set to 1.

Token Information These features are based on the string *w*, such as contains-digits, contains-dollar-sign, etc [7].

Lexicon Feature The string of *w* is used as a feature. This group contains a large number of features (one for each token string present in the training data).

Lexicon Feature of Previous and Next Token The string of the previous token w−1 and the next token w+1 is used with the *initCaps* information of *w*. If *w* has *init-Caps*, then a feature (*initCaps,w+1)*NEXT is set to 1. If w is not *initCaps*, then (not-initCaps,w+1)NEXT is set to1. Same for *w−1*.

Hyphenated Words Hyphenated words w of the form s1-s2 have a feature U-U set to 1 if both s1 and s2 are initCaps. If s1 is initCaps but not s2, then the features U=s1,L=s2, and U-L are set to 1. If s2 is initCaps but not s1, then the features U=s2, L=s1, and L-U are set to1.

Within Quotes/Brackets

Sequences of tokens within quotes or brackets have a feature to indicate that they are within quotes. This feature useful for directly identifying tokens from data model

Global Features

Unigrams for each name class, words that precede the name class are ranked using correlation metric [7], and the top 20 are com-piled into a list called Useful Unigrams. If another occurrence of *w* in the same document has a previous word *wp* that can be found in Useful Unigrams, then these words are used as features *Other-occurrence-prev=wp*.

Bigrams Useful Bigrams is a list that consists of bigrams of words that precede a name class. Examples are "CITYOF", "ARRIVES IN", etc. The list is compiled by taking bigrams with higher probability to appear before a name class than the unigram itself (e.g., "CITY OF" has higher probability to appear before a location than "OF") If another occurrence of w has the feature BI-nc set to 1, then *w* will have the feature Other BI-nc set to 1.

Acronyms Words made up of all capitalized letters in the text zone will be stored as acronyms (e.g.,IBM). The system will then look for sequences of initial capitalized words that match the acronyms found in the whole document.

Name List

In additional to the above features used by both Max-Ent model, it uses additional features derived from name lists compiled from the text corpus. The list is a mapping of sequences of words to name classes. These are identified in the table below.

Table 3.1 List of supported NER Classes

Class	**Description**
Measure	Numerical values that mathematical functions work on.
Dimension	These are qualitative and do no total a sum. Ex. Region, names, locations, dates etc.
Function	Mathematical entities that assign unique output to given inputs. Ex. Total, average etc.
Sort	Special functions which arranges a group of inputs in a systematic order. Ex. Ascending, descending etc.
Negation	Words that implies a logical complement Ex. Not, except etc.

DateFunction	Special functions that can be applied on time-date data.
StringCriteria	The strings which from a part of data model or the meta.
NumericCriteria	Numbers and relational operators.
DateCriteria	Dates and times or timestamps.
ReportType	The type of graph or chart that is explicitly mentioned.

3.3.3 Part-of-Speech Assignment

The POS tagger used in the NLP core engine is implemented under the maximum entropy framework that learns a probability distribution for tagging from manually annotated data, namely, the . The probability p(a|b) represents the conditional probability of a tag a ∈A, given some context or history b ∈B, where A is the set of allowable tags, and where B is the set of possible word and tag contexts. The probability model is identical to equation 2.1. Where as usual, each parameter a_j corresponds to a feature f_j. Given a sequence of words $\{w_1,.....,w_n\}$ and tags $\{a_1,…..,w_n\}$as training data, we dene b_i as the context available when predicting a_i. The following table lists the POS tags in the Penn Treebank.

Table 3.2 List of POS Tags

Tag	**Description**
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular

NNPS	Proper noun, plural
PDT	Pre-determiner
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

3.4 INTERPRETATION ENGINE

The interpretation engine takes all the query details generated in the previous module and then finds the columns for each entity in the database. The data model loaded during the initial stages are used to identify the column ids. Then all the possible columns are mapped using the chunking data available and the mappings are ranked. The mappings are converted in to queries and fed into the Report Generation engine

3.5 REPORT GENERATION ENGINE

The queries generated in the previous model are used to generate appropriate graphs. The best graph is selected based on the type of query. If a chart type is explicitly mentioned in the query, then that chart type is overridden by the engine and graphs are produced.

3.6 LIMITATIONS

1. The NER is not able to tag the given sentences when they are unstructured.
2. The chunker is based on shallow parsing, and its accuracy drops when the sentences are semi-structured or unstructured.

CHAPTER IV

PROPOSED SYSTEM

4.1 PROPOSED ARCHITECTURE

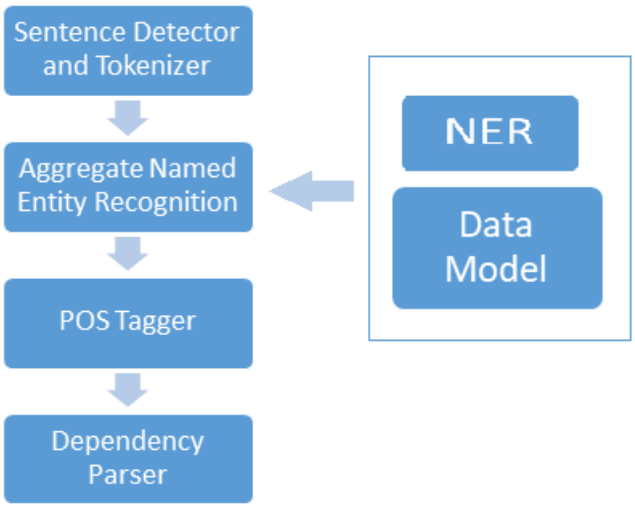

Figure 4.1 Modified NLP Pipeline in Proposed System

4.2 AGGREGATE NER

To increase the efficiency of the NER model. The entities are recognised by weighted probabilities from both the existing NER model and the Data model that is initially loaded. Then the Aggregate NER decides what to tag the token with depending on the probabilities provided by NER or Data Model. For example, if the NER recognises the token to be a Measure with probability of 0.75 and the Data Model recognises the token to be a dimension with 0.8 probability, then the aggregate NER tags the token to be a Dimension.

4.3 DEPENDENCY PARSING

One of the advantages of data-driven approaches to natural language processing is that development time can be much shorter than for systems that rely on hand-crafted resources in the form of lexicons or grammars. Here we use MaltParser. MaltParser can be characterized as

a data-driven parser-generator. While a traditional parser-generator constructs a parser given a grammar, a data-driven parser-generator constructs a parser given a treebank. MaltParser is an implementation of inductive dependency parsing [8], where the syntactic analysis of a sentence amounts to the derivation of a dependency structure, and where inductive machine learning is used to guide the parser at nondeterministic choice points. The Malt Parser operates with the following set of data structures, which also provides the interface to the feature model:

- A stack STACK of partially processed tokens, where STACK[i] is the i+1th token from the top of the stack, with the top being STACK[0]

- A list INPUT of remaining input tokens, where INPUT[i] is the i+1th token in the list, with the first token being INPUT[0].

- A stack CONTEXT of unattached tokens occurring between the token on top of the stack and the next input token, with the top CONTEXT[0] being the token closest to STACK[0] (farthest from INPUT[0]).

- A function HEAD defining the partially built dependency structure, where HEAD[i] is the syntactic head of the token i (with HEAD[i] = 0 if i is not yet attached to a head).

- A function DEP labelling the partially built dependency structure, where DEP[i] is the dependency type linking the token i to its syntactic head (with DEP[i] = ROOT if i is not yet attached to a head)

- A function LC defining the leftmost child of a token in the partially built dependency structure (with LC[i] = 0 if i has not left children)

- A function RC defining the rightmost child of a token in the partially built dependency structure (with RC[i] = 0 if i has not right children)

- A function LS defining the next left sibling of a token in the partially built dependency structure (with LS[i] = 0 if i has no left siblings)

- A function RS defining the next right sibling of a token in the partially built dependency structure (with RS[i] = 0 if i has no right siblings)

An algorithm builds dependency structures incrementally by updating HEAD and DEP, but it can only add a dependency arc between the top of the stack (STACK[0]) and the next input token (INPUT[0]) in the current configuration. We use [4] which is a linear-time algorithm limited to projective dependency structures.

4.3.1 Feature Models

We use history-based feature models for predicting the next action in the deterministic derivation of a dependency structure, which means that it uses features of the partially built dependency structure together with features of the (tagged) input string. More precisely, features are defined in terms of the word form (LEX), part-of-speech (POS) or dependency type (DEP) of a token defined relative to one of the data structures STACK, INPUT and CONTEXT, using the auxiliary functions HEAD, LC, RC, LS and RS.
A feature model is defined in an external feature specification with the following syntax:

```
<fspec>    ::=  <feat>+
<feat>     ::=  <lfeat>|<nlfeat>
<lfeat>    ::=  LEX\t<dstruc>\t<off>\t<suff>\n
<nlfeat>   ::=  (POS|DEP)\t<dstruc>\t<off>\n
<dstruc>   ::=  (STACK|INPUT|CONTEXT)
<off>      ::=  <nnint>\t<int>\t<nnint>\t<int>\t<int>
<suff>     ::=  <nnint>
<int>      ::=  (...|-2|-1|0|1|2|...)
<nnint>    ::=  (0|1|2|...)
```

As syntactic sugar, any <lfeat> or <nlfeat> can be truncated if all remaining integer values are zero. Each feature is specified on a single line, consisting of at least two tab-separated columns. The first column defines the feature type to be lexical (LEX), part-of-speech (POS), or dependency (DEP). The second column identifies one of the main data structures in the parser configuration, usually the stack (STACK) or the list of remaining input tokens (INPUT),

as the “base address” of the feature. The third alternative, CONTEXT, is relevant only together with Covington’s algorithm in non-projective mode.

4.3.2 Learning Mode

In learning mode, the system takes as input a dependency treebank and induces a classifier for predicting parser actions, given specifications of a parsing algorithm, a feature model and a learning algorithm. The input must be in the CoNLL-X format. For the sentence “total and average sales in 2010”, the following table gives the CoNLL-X representation that is required to train the classifier.

Table 4.1 CoNLL-X representation for a sample sentence

ID	FORM	LEMMA	CPOSTAG	POSTAG	FEATS	HEAD	DEPREL	PHEAD	PDEPREL
1	total	total	JJ	JJ	-	4	amod	-	-
2	and	and	CC	CC	-	1	cc	-	-
3	average	average	JJ	JJ	-	1	conj	-	-
4	sales	sales	NNS	NNS	-	0	Root	-	-
5	in	in	IN	IN	-	6	case	-	-
6	2010	2010	CD	CD	-	4	nmod	-	-

- **ID**: Token counter, starting at 1 for each new sentence.
- **FORM**: Word form or punctuation symbol.
- **LEMMA**: Lemma or stem (depending on the particular treebank) of word form, or an underscore if not available
- **CPOSTAG**: Coarse-grained part-of-speech tag, where the tag-set depends on the treebank.
- **POSTAG**: Fine-grained part-of-speech tag, where the tag-set depends on the treebank. It is identical to the CPOSTAG value if no POSTAG is available from the original treebank.
- **FEATS**: Unordered set of syntactic and/or morphological features, or an underscore if not available.
- **HEAD**: Head of the current token, which is either a value of ID, or zero (’0’) if the token links to the virtual root node of the sentence.

- **DEPREL**: Dependency relation to the HEAD. The set of dependency relations depends on the particular treebank. The dependency relation of a token with HEAD=0 may be meaningful or simply 'ROOT' (also depending on the treebank).
- **PHEAD**: Projective head of current token, which is either a value of ID or zero ('0'), or an underscore if not available. The dependency structure resulting from the PHEAD column is guaranteed to be projective (but is not available for all data sets).
- **PDEPREL**: Dependency relation to the PHEAD, or an underscore if not available.

4.3.3 Parsing Mode

In parsing mode, the system takes as input a set of sentences and constructs a projective dependency graph for each sentence, using a classifier induced in learning mode (and the same parsing algorithm and feature model that were used during learning). The raw sentence is taken and POS tags are identified using the POS tagger. Then the sentence is converted to CoNLL-X format by filling the columns, ID, FORM, LEMMA, CPOSTAGS and POSTAGS. All the other columns are filled with "-". Then the classifier identifies the rest of head-dependent relationships.

4.4.4 Universal Dependencies

The following table lists the 37 universal syntactic relations used in UD sorted in alphabetical order.

Table 4.2 List of Universal Dependencies

Tag	Dependency Relation
acl	clausal modifier of noun (adjectival clause)
advcl	adverbial clause modifier
advmod	adverbial modifier
amod	adjectival modifier
appos	appositional modifier
aux	auxiliary
case	case marking
cc	coordinating conjunction
ccomp	clausal complement
clf	classifier
compound	compound
conj	conjunct
cop	copula
csubj	clausal subject
dep	unspecified dependency
det	determiner
discourse	discourse element
dislocated	dislocated elements
expl	expletive
fixed	fixed multiword expression
flat	flat multiword expression
goeswith	goes with
iobj	indirect object
list	list
mark	marker
nmod	nominal modifier
nsubj	nominal subject
nummod	numeric modifier
obj	object
obl	oblique nominal
orphan	orphan
parataxis	parataxis
punct	punctuation
reparandum	overridden disfluency
root	root
vocative	vocative
xcomp	open clausal complement

Table 4.3 Classification of Dependency Relationships

	Nominals	**Clauses**	**Modifier words**	**Function Words**
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod* discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct root dep

CHAPTER V

EXPERIMENTAL RESULTS

5.1 PLATFORM AND TOOLS

OPERATING SYSTEM	UBUNTU LINUX
LANGUAGE	JAVA
LIBRARIES & TOOLS	APACHE OPENNLP, MALTPASER

5.2 OUTPUT

5.2.1 Dependency Graph for The Given Query

Query: total and average sales in 2010

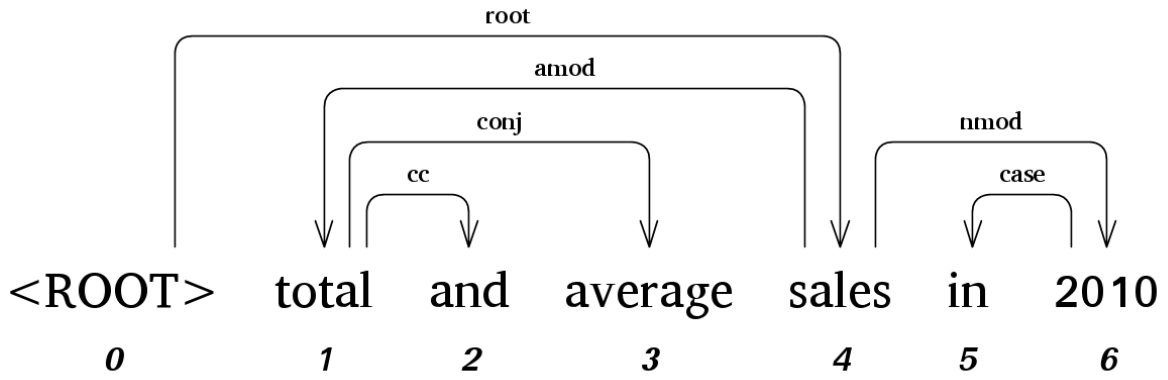

Figure 5.2 Dependency graph

5.2.2 Full Stack Output for the Query

Query: total and average sales in 2010

```
Enter The Query :
total and average sales in 2010

NER Output : [[0..1) function, [2..3) function, [3..4) measure, [5..6) datecriteria]
--------
Dependency Parsing

Index   Tokens  POS Tags
0       total   JJ
1       and     CC
2       average JJ
3       sales   NNS
4       in      IN
5       2010    CD

Dependency Relations:

amod(sales(4)->total(1))
cc(total(1)->and(2))
conj(total(1)->average(3))
ROOT(0)->sales(4)
case(2010(6)->in(5))
nmod(sales(4)->2010(6))

Conll-X Format :
1       total   total   JJ      JJ      _       4       amod    _       _
2       and     and     CC      CC      _       1       cc      _       _
3       average average JJ      JJ      _       1       conj    _       _
4       sales   sales   NNS     NNS     _       0       root    _       _
5       in      in      IN      IN      _       6       case    _       _
6       2010    2010    CD      CD      _       4       nmod    _       _


OUTPUT :
Visualziation type: UNKNOWN
Measure 1
Table Name: Order
Column Name: Sales
Function: SUM
isFormulaColumn: false
sortType: DEFAULT
sortTypeApplied: NO_SORT

Measure 1
Table Name: Order
Column Name: Sales
Function: AVERAGE
isFormulaColumn: false
sortType: DEFAULT
sortTypeApplied: NO_SORT

Criteria 1
Table Name: Order
Column Name: Order Date
Function: ACTUAL
isFormulaColumn: false
Criteria Type: YEAR
Standard Criteria Values: [2010:XX:XX:XX:XX:XX]
isCriteriaIncluded: true
```

Figure 5.1 Full Stack Output

5.2.3 Accuracy for MaltParser Model

```
============================================================
Gold:   ../../Demo/Testing/golden_standards.conll
Parsed: ../../Demo/Testing/test.conll
============================================================
GroupBy-> Token
Metric-> BothRight

============================================================

accuracy   Token
--------------------
0.899      Row mean
135683     Row count
--------------------
```

Figure 5.3 Accuracy

5.2.4 Accuracy Based On Projectivity

```
============================================================
Gold:   ../../Demo/Testing/golden_standards.conll
Parsed: ../../Demo/Testing/test.conll
============================================================
GroupBy-> ArcProjectivity
Metric-> ArcProjectivity

============================================================

precision   recall   fscore   ArcProjectivity
-----------------------------------------------
-           0        -        Non-proj
0.949       1        0.974    Proj
```

Figure 5.4 Precision, Recall and Fscore

5.2.5 Accuracy for Each Dependency Relation

```
================================================
Gold:   ../../Demo/Testing/golden_standards.conll
Parsed: ../../Demo/Testing/test.conll
================================================
GroupBy-> Deprel
Metric-> Deprel
================================================

precision   recall   fscore   Deprel
-----------------------------------
0           -        -        ROOT
0.835       0.764    0.798    acl
-           0        -        acl_to
0.78        0.61     0.684    advcl
0.333       0.125    0.182    advcl_on
0.969       0.961    0.965    advmod
0.954       0.972    0.963    amod
0.744       0.846    0.792    appos
0.943       0.975    0.958    aux
1           0.994    0.997    auxpass
0.992       0.997    0.995    case
0.998       0.999    0.999    cc
0.743       0.619    0.675    ccomp
0.96        0.944    0.952    compound
0.895       0.928    0.911    conj
0.991       1        0.996    cop
0.615       0.229    0.333    csubj
0.552       0.539    0.546    dep
0.999       0.999    0.999    det
1           1        1        discourse
0.917       0.902    0.909    dobj
1           1        1        expl
0.894       0.971    0.931    fixed
0.961       1        0.98     iobj
0.963       0.936    0.949    mark
0.993       0.986    0.989    mwe
0.974       0.98     0.977    neg
0.968       0.98     0.974    nmod
-           0        -        nmod_for
-           0        -        nmod_from
0           0        -        nmod_of
0.887       0.441    0.589    nmod_on
-           0        -        nmod_respect_to
0.75        0.167    0.273    nmod_with
0.96        0.796    0.87     nsubj
0.968       0.939    0.953    nsubjpass
0.966       0.971    0.969    nummod
0.489       0.512    0.5      parataxis
0.997       0.99     0.994    punct
0.994       0.987    0.991    ref
0.875       0.848    0.862    root
0.922       0.81     0.862    xcomp

------------------------------------------------
```

Figure 5.5 precision, recall and fscore for each dependency relation

5.2.6 Sample 1 from Website

Query: Sales by product name as pie chart

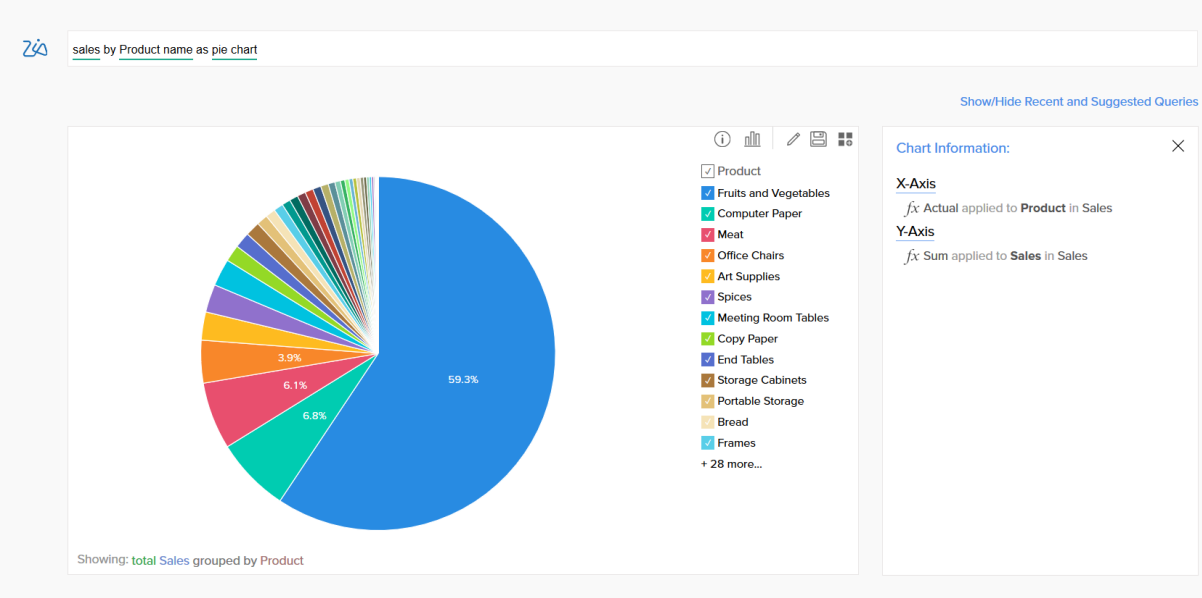

Figure 5.6 Web Sample 1

5.2.7 Sample 2 from Website

Query: Compare sales and cost for every first quarter

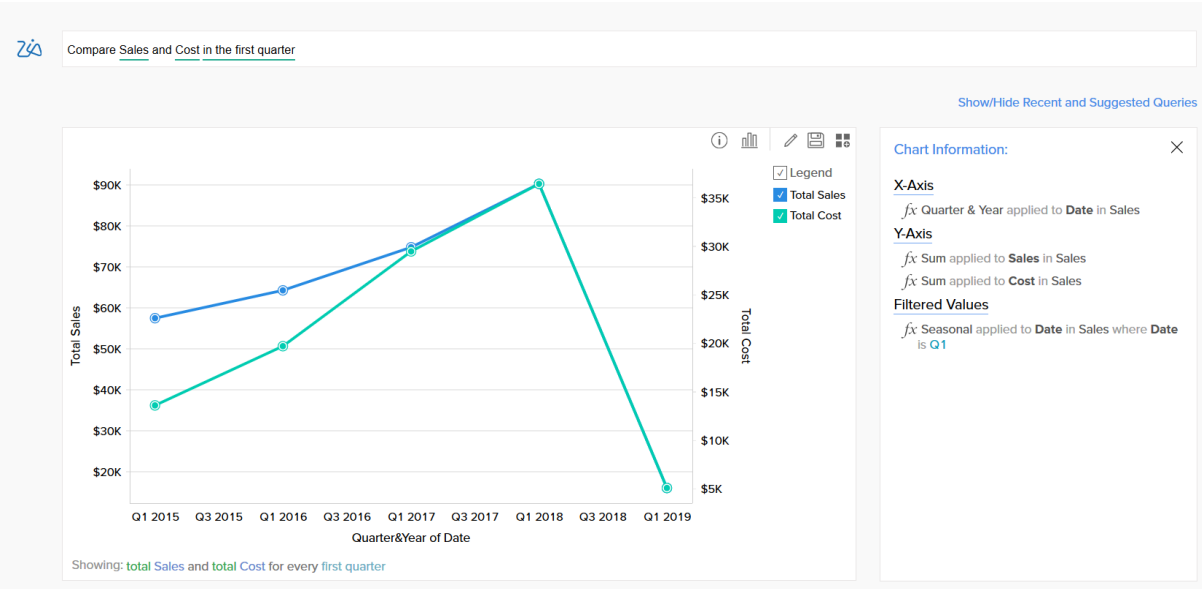

Figure 5.7 Web Sample 2

CHAPTER VI

CONCLUSION AND FUTURE ENHANCEMENTS

6.1 CONCLUSION

The project has presented a data-driven system for dependency parsing that can be used to construct syntactic parsers to use in Business Intelligence and more specifically in Search-Driven Analytics. Experimental evaluation using data shows that MaltParser generally achieves good parsing accuracy without language-specific enhancements and with fairly limited amounts of training data. The project has achieved an accuracy of 89.9% accuracy. This was possible because of researching and testing different parsing algorithms and learning algorithms and to define arbitrarily complex feature models in terms of lexical features, part-of-speech features and dependency type features.

6.2 FUTURE ENHANCEMENTS

The possibility of developing custom dependency relations to tweak and customize the system more towards Search-Driven Analytics has to be explored. The classifiers used at various staged can be updated to Deep Neural Networks to further boost the accuracy to reach state of the art accuracies.

REFERENCES

[1] Ratnaparkhi Adwait, “Maximum entropy models for natural language ambiguity resolution”, *in the proceedings of IRCS Technical Reports Series. 60. University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-98-15,* 1998

[2] Nivre, J, “Algorithms for Deterministic Incremental Dependency Parsing”, *in proceedings of the Computational Linguistics* 34(4), 513-553, 2008.

[3] Nivre, J., J. Hall and J. Nilsson “MaltParser: A Data-Driven Parser-Generator for Dependency Parsing”, *in proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006),* Genoa, Italy, pp. 2216-2219, May 2006.

[4] Nivre, J, “An Efficient Algorithm for Projective Dependency Parsing” in *proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03),* Nancy, France, pp. 149-160. 23-25 April 2003.

[5] Michael A. Covington, “A Fundamental Algorithm for Dependency Parsing” *in proceedings of the 39th Annual ACM Southeast Conference*, 2000.

[6] Cortes, Corinna; Vapnik, Vladimir N, "Support-vector networks" *in the proceedings Machine Learning.* 20 (3): 273–297, 1995.

[7] Hai Leong Chieu, Hwee Tou Ng, “Named entity recognition with a maximum entropy approach” *in the proceedings of the seventh conference on Natural language learning* at HLT-NAACL 2003-Volume 4, 31 May 2003.

[8] Nivre, J. and J. Nilsson, “Pseudo-Projective Dependency Parsing”, in *proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL),* pp. 99-106, 2005