Terascale Knowledge Acquisition

by

Deepak Ravichandran

Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2005

Dedication

To My Parents....

Acknowledgments

First and foremost, I would like to thank my advisor and chair of my committee, Dr. Eduard Hovy, for his inspiring guidance over the past 5 years. He was truly an exemplar role model for me. The following quote by Edward Bulwer-Lytton aptly describes him: *"The best teacher is the one who suggests rather than dogmatizes, and inspires his listener with the wish to teach himself."* I always felt that he had the perfect blend of micro-management and macro-management skills while guiding his students.

I would also like to specially thank another member of my committee, Dr. Patrick Pantel. I had the great opportunity to work with him very closely. He has been like an academic older-brother to me. I was always impressed by his impeccable attitude towards research, which helped shape a lot of my ideas.

I am also very thankful to other members of my committee: Dr. Kevin Knight, Dr. Daniel Marcu, Dr. Dennis Mcleod and Dr. Daniel O'Leary.

I was blessed to have two wonderful and outstanding colleagues and friends in Michael Fleischman and Alexander Fraser, with whom I have had many insightful conversations and arguments about research. These discussions were a source of inspiration for many of my research endeavors.

I was fortunate to be surrounded by many gifted and smart colleagues like Jafar Adibi, Rahul Bhagat, Michele Banko, Timothy Chklovski, Hal Daumé III, Donghui Feng, Michel Galley, Ulf Hermjakob, Abraham Ittycheriah, Soo-Min Kim, Philipp Köhn, Namhee Kwon, Chin-Yew Lin, Shou-de Lin, Lucian Vlad Lita, Jay Modi, Dragos Munteanu, Tom Murray, Franz Josef Och, Andrew Philpot, David Pynadath, Monica Rogati, Salim Roukos, Radu Soricut, Lara Taylor, and Liang Zhou.

I am also very thankful for the help and support I received from the members of ISI Natural Language Group , ISI ISD division and the "Venice beach happy-hour crowd" over the past several years. It made my stay in Los Angeles over the past five years to be truly memorable. I would especially like to thank Kathy Kurinsky and Erika Barragán-Nuñez for helping me with all the administrative work at ISI.

I am indebted to Michel Galley and Lara Taylor for helping me proof read parts of my thesis. I would also like to thank USC Center for High Performance Computing for helping me use their cluster computers. Finally, I would like to thank my parents and my sister who were a constant source of inspiration and always helped me to believe in myself.

Contents

List Of Tables

List Of Figures

Abstract

Performance of many Natural Language Processing (NLP) systems have reached a plateau using existing techniques. There seems to be a general consensus that systems have to integrate semantic knowledge or world knowledge in one form or another in order to provide additional information required to improve the quality of results. But building adequate and large enough semantic resources is a difficult unsolved problem. In my thesis, I attack the problem of very large scale acquisition of semantic knowledge by exploiting natural language text available on the Internet. In particular, I concentrate on one problem: extracting *is-a* relations from a very large corpus (70 million web pages, 26 billion word corpus) downloaded from the Internet. Since the amount of data involved is greater by two orders of magnitude than published before, the algorithms designed had to be highly scalable. This was achieved by:

1. Using a novel Pattern-based learning algorithm that exploits local features.

2. Using a Clustering algorithm that uses randomized techniques by exploiting co-occurrence (global) features in linear time.

Using these algorithms, I extract *is-a* relations from text to build a huge table. These extracted relations are then evaluated by using a host of different applications.

Chapter 1

Introduction

1.1 Motivation

Natural Language Processing is one of the core sub-fields of Artificial Intelligence. It deals with the ability of computers to understand human language. The definition of what constitutes "understanding human language" is very unclear and varies from person to person. The Turing test is one way of testing an artificial system's ability to act intelligibly. But it remains a well-known fact that computers of today are not even close to passing the Turing test. Hence, many natural language systems today are increasingly focusing on task-oriented applications such as Machine Translation (MT), Question Answering (QA), Summarization, and Information Retrieval (IR). Working on these types of systems have two-fold benefits:

1. Success in these systems directly translates to useful (commercially viable) technologies.

2. These systems have clear evaluation criteria that can be objectively evaluated using an array of measures.

According to classical NLP theory, an AI system needs to have lexical, syntactic, semantic, pragmatic, and discourse knowledge to understand language. Some NLP systems primarily use simple word (lexical) models. These include typical IR and Summarization systems. Other systems have transitioned from simple word models to bag of word models (e.g., some MT systems). Other NLP systems have started using sophisticated syntax based models. Discourse theories are being used by some Summarization systems.

However, NLP researchers are increasingly realizing that the performances of their system have reached a plateau using the above mentioned techniques. There seems to be a general consensus that systems have to integrate semantic knowledge or world-knowledge in one form or another. For example, to answer a question, *What is the color of cranberry juice?* it would be very useful for a system to have the knowledge about *What is a color?* and *What are the different colors?*. Working with semantic knowledge is considered to be very hard problem, for several reasons, including the following: How does one represent knowledge? How does one acquire knowledge? and How does one reason with knowledge?

There has been a lot of work on representing and reasoning with knowledge. Examples include logic, Knowledge Representation (KR) and AAAI conferences, domain ontologies, and expert systems. But while small-scale KR systems (domain nodes under 1000 nodes) have been used many times, no true large scale systems exists with the exception of CYC [46]. This is primarily due to the difficulty of acquiring and integrating lots of knowledge automatically. It would be perfect if a system could simply read and encode knowledge from an encyclopedia, topic by topic. Presently knowledge is built at a lexical level. It would be useful to develop an automatic process constructs such knowledge. Two of the best systems widely known for providing semantic knowledge are CYC [46] and WordNet [53]. CYC is a general knowledge base and commonsense reasoning engine. WordNet is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. Both these semantic sources are built manually by a painstakingly slow and tedious process and despite many attempts there are very few systems that seem to have leveraged their performance by use of WordNet and CYC. We argue that this is because of two reasons:

1. These systems have a very low coverage (recall).

2. These systems cannot be dynamically modified to handle new domains or events.

In this thesis, we attack the problem of large scale acquisition of semantic knowledge. We base our work on two facts:

1. We have seen an evolution of sophisticated statistical/empirical techniques being used by researchers of the NLP community.

2. There is an explosion in the amount of available digital text resources. It is estimated that the Internet contains approximately 7500 Terabytes of digital data. This data is available in structured, semi-structured and unstructured formats. Structured data is in the form of online databases. Semi-structured data is in the form of HTML tables and other HTML tags. But a vast majority of this data is in the form of raw text.

In the next section, we make a more detailed analysis of the challenges associated with working with large amounts of data.

1.2 Working with Terabytes of data

Statistical/Empirical techniques employed by NLP researchers till date operate on data in the order of Megabytes or Gigabytes. We argue that this is because of the following reasons:

1. Most of the work involves use of supervised learning algorithms which typically require data tagged by humans (e.g. FrameNet corpus, Penn Tree Bank). This is a time consuming and extremely costly process.

2. Many algorithms use NLP tools such as Syntactic Parsers, which require large amounts of processing time.

3. Many unsupervised algorithms (e.g. clustering algorithms) work well only on Gigabytes of data and do not scale to the Terabytes level.

4. Terabytes of text data is not made readily available to NLP researchers by organizations like LDC (Linguistic Data consortium).

5. Most of the places performing NLP research do not have the necessary infrastructure (e.g., disk space, computing power) to work with such kinds of data.

The amount of data available on the Internet is growing at an exponential rate. It is estimated that the Internet contains at least 100 Terabytes of text. The popular search engine Google indexes about 9 Terabyte of the Web to perform its search. One reasonable question to ask is what would happen if we extracted knowledge from a corpus which is in the order of Terabytes instead of Gigabytes. Banko and Brill [4], while working with large amounts of data empirically showed that:

1. Scaling data by orders of magnitude tremendously boosts the performance of a NLP system.

2. Simple algorithms have similar performance to more sophisticated algorithms when experiments are performed with huge amounts of data.

Working on Terabytes of data poses new challenges. These involves various engineering and algorithmic changes to the current approaches. Some of the basic challenges are:

1. **Algorithm**: Algorithms have to strictly be linear with respect to the size of the corpus *O(n)*. It is probably impossible to work with algorithms which is more than linear with the given computing power. Also the algorithms should involve only unsupervised or semi-supervised machine learning techniques. It is almost impossible to hand tag data which is in the order of Terabytes.

2. **Storage**: How would one store Terabytes of data? The answer to this question is straightforward – Hard Disks. It is estimated that data storage capacity doubles every year. (This statement holds true only after 1989. Between 1960 and 1989 data storage grew only at the rate of 30%). A Terabyte of data today costs less than $5,000. It is estimated that by the early 2010s we could buy a Petabyte of data for the same cost as a Terabyte today.

3. **Data access**: What is the rate at which one could access data? The Data access rate from Hard Drives has only been growing at a rate of 10% a year. The data storage rate is, thus, growing an order of magnitude faster that data access rate. This probably means that we need to consider with care the ways in which we

access data. As we learned in basic Computer Science Systems text books, accessing a random location on a disk involves an overhead in terms of disk head rotate and seek. This is a major source of delay. Disks allow roughly 200 accesses per second. So if one reads only a few Kilobytes in every disk access, it will take almost a year to read an entire 20 Terabyte disk [31]. To significantly simplify our data access problems, we may need to start using our disk as tapes by performing sequential access. If one reads and writes large chunks of data, data access speed can be increased 500 times.

4. **NLP tools**: Which NLP tools could one use? One of the biggest achievements in Natural Language Processing in the 1990s was the availability of free tools to perform various tasks such as Syntactic parsing, Dependency parsing, Discourse parsing, Names-Entity Identification, Part of Speech Taggers, etc. Almost all these tools work linearly in the number of sentences. This is because they treat each sentence independently of other sentences. (However, in the intra-sentence level these tools may perform differently as a function of the number of words in the sentences.) We study and apply various off-the-shelf tools to data sets and estimate the amount of time taken to process a Terabyte of corpus. We take Brill's Part of Speech Tagger[11], Noun Phrase Chunker CASS [8], Lin's dependency parser Minipar [47], and Charniak's syntactic parser [19]. Results are shown in

Table 1.1: Approximate processing time on a single Pentium-4 2.5 GHZ machine for a 1 Terabyte text corpus.

Tool	Processing time for 1 TB(approx.)
POS Tagger	125 days
NP Chunker	216 days
Dependency Parser	10.2 years!
Syntactic Parser	388.4 years!

Table 1.1. It is very clear that Terabyte-sized experiments cannot use any NLP tools in the current form.

5. **Computing Power**: What computer should one use? Computers have been following Moore's law: Computer processing speed doubles every 18 months. An exciting development over the past years has been the availability of cluster computers to NLP researchers. Cluster computers are relatively cheaper compared to Vector computers, because they are built from cheap and mass-produced Intel processors with the free Linux Operating System installed on them. Cluster computers also have a Gigabit switch between them, acting like a cheap context switch. Using a cluster computer, Part of speech tagging and noun phrase chunking becomes manageable to handle. However, Syntactic parsers and Dependency parsers cannot be used in the order of Terabytes.

1.3 Objective and Approach

In this work, we make use of unstructured text from the Internet to build a knowledge base of semantic information (called semantic repositories) on a very large scale.

The principle question answered in our thesis is:

"How can we build large semantic repositories by exploiting the Terabytes of unstructured text data available on the Internet?"

The approach taken to solve this question can be stated simply by the following steps:

1. Develop fast algorithms for extracting semantic knowledge from text.

2. Download text data in the order of Terabytes from the Internet.

3. Apply algorithms for extracting semantic knowledge on this data.

Which semantic relation should we use to acquire knowledge? It is beyond the scope of one thesis to acquire everything. We focus on a collection of basic and essential semantic relation: Hyponymy or *is-a* relation. Examples of Hyponymy relation are:

1. *Red* is a *color*.

2. *United States* is a *country*.

3. *Martin Luther King* was a *leader*.

Our algorithm to extract knowledge from text consists of two approaches:

1. Pattern learning to exploit local features.

2. Clustering to exploit global (co-occurrence) features.

Both these algorithms are designed so that they are scalabale for handling terabytes of data.

1.4 Major Contributions

The thesis has the following major contributions:

1. Most present-day NLP systems work on relatively smaller corpora. Their algorithms do not scale up to a corpus in the order of Terabytes. Our thesis is the first comprehensive effort to use both engineering and algorithmic solution in NLP to process data in the order of Terabytes.

2. This thesis is the first one in NLP to make use of randomized algorithms and high speed noun clustering.

3. This thesis uses a novel pattern based algorithm to exploit local features and a randomized clustering algorithm to exploit global features to build a scalable framework for extracting knowledge from text.

1.5 Thesis Outline

The remaining chapters are divided as follows:

- Chapter 2 discusses related work and how our work differs from others.
- Chapter 3 discusses various pattern learning algorithms and their advantages and disadvantages.
- Chapter 4 describes the use of fast randomized algorithms for performing high speed clustering.
- Chapter 5 discusses the application of a pattern learning algorithm and a high speed pattern learning algorithm to a specific domain of knowledge harvesting (viz. *is-a* relation) along with various refinements.
- Chapter 6 provides the evaluation of our system on precision and recall measures.
- Chapter 7 provides future work and conclusions.

Chapter 2

Related Work

2.1 Introduction

I broadly classify the previous work, associated with my research into two broad categories.

1. Text Mining

2. Working on Large Training data sets

2.2 Text Mining

2.2.1 Data Mining

The origin of Text Mining draws upon Data Mining. Data Mining is defined as the "the non trivial process of identifying valid, novel, potentially useful, previously unknown

and ultimately understandable patterns in large databases" [27]. Data Mining work was based solely on database or structured data. The motivation of data mining was captured tersely by the Adibi: "We are drowning in Data but Starving for Knowledge" [1]. A higher level knowledge representation was sought from raw lower level data. This goal spurred researchers to explore several techniques previously used in fields as diverse as AI/Machine Learning, Database, Statistics, Visualization, Information Retrieval, and High performance computing to perform Knowledge Discovery in Database (KDD). A famous KDD, "Diapers and Beers", was conducted by K. Heath at Terradata's Industry Consulting. He found by performing SQL (Structured Query Language) joins on sales of products related to baby items, that if one tries to buy diaper, they also tend to buy beer. This pattern was found true in over 50 stores over a 90 day period.

2.2.2 Information Extraction

In Information Extraction tasks certain kind of information was required to be searched from a given corpus. The Message Understanding Conference (MUC) [66, 67, 68] evaluations helped various Information Extraction systems participate in a common domain. These domains were:

1. Telegraphic messages about naval operations (MUC-1 and MUC-2).

2. News articles and transcripts of radio broadcast (some converted from Spanish) related to Latin American terrorism (MUC-3 [66] and MUC-4 [67]).

3. Joint-ventures from business news (MUC-5 [68]).

As given in [37] we describe the information extraction task for MUC-5. A typical text would contain:

Bridgetone Sports Co. said Friday that it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.

The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and "metal wood".

The information to be extracted is given by the following templates:

Relationship:	*TIE-UP*
Entities:	*"Bridgestone Sports Co."*
	"a local concern"
	"a Japanese trading house"
Joint Venture Company:	*"Bridgestone Sports Taiwan Co."*
Activity:	*ACTIVITY-1*
Amount:	*NT$20,000,000*
ACTIVITY-1:	
Activity:	*PRODUCTION*
Company:	*"Bridgestone Sports Taiwan Co."*
Product:	*"iron and 'metal wood' clubs"*
Start Date:	*DURING: January 1990*

Thus the problem of Information Extraction was reduced to the one of filling templates. Each template has a predefined structure. In the above example related to *business tie-ups*, a system had to fill information such as the *companies* involved in the tie-up, *name* of the new company, the *amount* of money involved in the business merger etc.

Most of the earlier Natural Language Information Extraction system relied on using deep semantic/syntactic processing for filling in such templates. These tasks were viewed as specific applications of the general problem of Natural Language Understanding (NLU). As it turned out these NLU based Information Extraction systems had good precision but low recall. They also needed a lot of computer-processing power. However, during the MUC-3 evaluations it was found that the best system [45] used simple text patterns to perform Information Extraction. The apparent power of such a simple system surprised many people. The patterns were of the following form: To identify companies that are involved in a *business merger* pattern such as *<COMPANY A> merges with <COMPANY B>* could be used. Initially several systems used such lexical patterns to build Information Extraction systems. Some other systems used lexico-syntactic patterns. These patterns were originally learned of a manually tagged corpus or hand-crafted by a human. The advantage of using hand-crafted patterns is that generally these patterns have high precision and good compact representation. But

these patterns are extremely time-consuming and laborious to build and they were not easily portable across different domains.

To overcome these problems, Riloff [63] suggested an automatic unsupervised way of learning Information Extraction patterns. Starting from a few seed examples, Riloff learn text based patterns. As reported in her paper, she used simple heuristics to collect patterns. Having collected some patterns corpus statistics to weed out the "bad" patterns from the "good" ones by using manual relevance judgment. Although, this method acquires patterns automatically, it still requires some human effort for using relevance judgment in selecting good patterns.

As given by Grishman [32], a typical Information Extraction system uses the following components for pattern (template) generation:

1. **Lexical Analysis**: This is where the basic text preprocessing is performed. The text is broken into tokens and sentence segmented.

2. **Named Entity recognition**: The named entity analysis is performed. All the phrases representing Names of Persons, Organizations, Dates, Time, Money, Location and various other important nouns are identified.

3. **Partial syntactic analysis**: The text is processed through a shallow syntactic parsers. Most of the parsers are finite state automata based for speed of processing. Here phrases like noun phrases, verb phrases, subject-object relationships are identified.

4. **Scenario pattern matching**: In this phase patterns are constructed using semi-supervised techniques.

5. **Co-reference resolution**: Here generally all the pronominal references are normalized to their respective mentions.

6. **Inference**: In this phase information from adjacent sentences from the same discourse are fused to perform some low-level inference specific to the task.

7. **Template generation**: In this phase, patterns are generated using the output from previous stages for information extraction.

Brin [9], and Agichtein and Gravano [2] propose pattern-based bootstrapping methods for extracting entities such as { *organization,headquarters* } relation from very large text collections. In [3], Agichtein and Gravano suggest techniques for speeding up the process, by training queries used in an Information Retrieval System.

2.2.3 Text Mining

According to Hearst [36] "Text Mining is the discovery by computer of new, previously unknown information, by automatically extracting information from different written resources. A key element is the linking together of the extracted information together to form new facts or new hypotheses to be explored further by more conventional means of experimentation."

Heart in [34] proposed an algorithm for the (semi-) automatic acquisition of Hyponym relations from unstructured text. Some examples of Hyponym relations (as defined by WordNet [53]) are :

1. *Paris* is a *city*.

2. *Red* is a *color*.

Starting with a few seed examples of Hyponym relations (e.g., *Paris - city*, *red - color*) she extracted all the sentences containing both the terms of the Hyponym from a corpus. From amongst these sentences useful patterns of interests were extracted. Once these patterns were discovered, more instances of the Hyponym relations were extracted and the same process was repeated whenever a new pattern of interest was found.

The following "useful" patterns of interest were discovered:

1. NP such as { NP ,}* { (or | and) } NP

2. such NP as { NP ,}* { (or | and) } NP

3. NP{, NP}* {,} or other NP

4. NP{, NP}* {,} and other NP

5. NP {,} including {NP ,}* {or | and} NP

6. NP {,} especially {NP,}*{or | and} NP

Hearst was able extract less than 400 relations from a corpus of about 20M words. The accuracy of the results was not reported quantitatively, but they were found to be of high quality.

Berland and Charniak [7] used techniques similar to Hearst [34] to extract part-whole relations. These relations are also called Meronymy in [53]. Some examples of Meronymy relations are:

1. *Basement* is a part of *building*.

2. *Wheel* is a part of *car*.

They used the following two patterns to extract meronymy relations:

1. whole-NN 's part-NN

2. part-NN of {the | a} whole-NN

Using these patterns together with other heuristic technique Berland and Charniak were able to extract relations with about 55% accuracy. They worked on a corpus of about 100,000 words. However, coverage is not clearly reported in their work.

Girju et al. [29] improved upon the work Berland and Charniak [7]. In their work, they used three patterns learnt semi-automatically:

1. whole-NP 's part-NP

2. part-NP of whole-NP

3. part-NP VERB while-NP

These patterns were used to extract about 100,000 sentences from the LA Times articles of TREC9 corpus. They then used a filter to weed out the bad examples from the legitimate ones using supervised learning (Decision Trees) algorithm. WordNet based features were used for the learning algorithm. They report a precision of 83% and a recall of 72%.

Mann [51] proposed the use of lexico-POS based patterns for constructing an ontology of *is-a* relations for proper nouns. He used the manually crafted pattern *CN PN*[1]. This pattern detects phrases of the type *[the] automaker Mercedes Benz*. He reported generating 200,000 unique descriptions of proper nouns with 60% accuracy.

[1] *CN= Common Noun PN=Proper Noun*

Fleischman et al. [28] improved upon the work of Mann [51] by using a larger set of manually crafted patterns to extract *is-a* relationship for people and their occupation/position. E.g. *Nadia Comaneci* is a *Romanian Gymnast*. *Lilian Thuran* is a *French defender*. To improve the accuracy of the extracted relations they used a Decision tree based machine learned filter based on a series of rich semantic and syntactic features. They report an accuracy of 93% extracted from 930,000 unique proper noun descriptions which was obtained from a corpus of 15GB.

A good overview of the various Text Mining techniques is given by Moldovan and Girju [54].

Paşca [58] describes a technique for extracting glosses present for nodes present in WordNet from a corpus of 500 million webpages. He also proposes a clustering technique to group together nuggets that have a very high overlap. The paper reports that 60% of the WordNet nodes had at least one gloss extracted from the Web corpus. In [57], Paşca uses a pattern-based technique to extract *is-a* relations from a corpus of 500 webpages. However, our work differs from [57] and [58] because we make use of rich co-occurrence features.

Etzioni et al. [24, 25, 26] extracts instance-concept relations from a huge web corpus. They use a combination of Pattern Learning, Subclass extraction and List Extraction to perform the task. However, in all their experiments the extraction category is known prior to extraction. They use Point-wise mutual information (PMI) measure

to exploit some co-occurrence features for outputs that are fired by multiple patterns. However, they also do not use clustering technique that are capable of extracting very rich global co-occurrence statistics.

Ciaramita and Johnson [21] use a fixed set of 26 semantic labels to perform *is-a* supersense tagging.

Caraballo [15] uses a clustering technique to extract hyponym relations from newspaper corpus. Pantel and Ravichandran [56]. also use similar technique. They use Clustering by Committee (CBC) algorithm [55] to extract clusters of nouns belonging to the same class. Thus, the fruit sense of orange would contain the following members obtained after clustering:

... peach, pear, apricot, strawberry, banana, mango, melon, apple, pineapple, cherry, plum, lemon, grapefruit, orange, berry, raspberry, blueberry, kiwi, ...

For each cluster, certain signature features are extracted which are known to signify Hyponym relations. Examples of such features include Appositives (e.g. ... *Oracle*, a *company* known for its progressive employment policies, ..) and Nominal subjects (e.g. ... *Apple* was a hot young *company*, with Steve Jobs in charge..). These signature features are used to extract the name of each cluster. Thus the top five ranking names for a cluster containing the following elements:

{*...Curtis Joseph, John Vanbiesbrouck, Mike Richter, Tommy Salo..*}

would be:

1) goalie 2) goaltender 3) goalkeeper 4) player 5) backup

In their work, they report a labeling precision of 77.1% from 1432 noun clusters obtained from a corpus of 3GB.

Cederberg and Widdows [17] use Latent Semantic Ananlysis and noun co-ordination (co-occurrence) technique to extract Hyponyms from a newspaper corpus. They report 58% accuracy using their approach.

Snow et al. [65] exploits both pattern-based and rich co-occurrence features to extract *is-a* relations from text. However, their technique is not easily web-scalable.

2.3 Working with very Large Corpora

Banko and Brill [4, 5] investigated the advantages of working with very large corpora. In particular, they worked on the problem of confusion set disambiguation. It is the problem of choosing the correct use of a word from a confusion set such as {*principle, principal*}, {*then, than*}, {*to, two, too*}, and {*weather, whether*}.

They empirically proved the following:

1. Learning curve is generally log-linear irrespective of the algorithm.

2. Simple and sophisticated algorithms have comparable performance with very large amount of data. In particular, the technique of voting by using different

classifiers trained on the same corpus seems to be ineffective in improving performance with large amounts of data.

3. One can achieve good performance by using supervised learning techniques by employing active learning and sample selection. In this way, one can obtain performances that are better than those of unsupervised methods by annotating only a fraction of the total corpus.

4. Weakly supervised techniques almost seem to have no effect on performance accuracy.

Curran and Moens [22] experimented with corpora of various sizes for automatic thesauri construction and made similar conclusions to that of Banko and Brill [4, 5]

2.4 Conclusion

Our work in this thesis closely resembles the previous work in Text Mining. However, this thesis extends previous work in text mining by proposing both algorithmic and engineering solution for web-scale knowledge harvesting.

Chapter 3

Pattern Learning

3.1 Introduction

In this chapter, we describe two pattern learning algorithms to extract knowledge from text. The first algorithm is a simple surface pattern learning algorithm which works only on the lexical (word) level. The second algorithm works on improving the drawbacks of the first algorithm and uses Part of Speech information. We call this the *multi-level pattern learning* algorithm. Both these algorithms are designed so that they are scalable at the terabyte level. For this reason, we avoid using Syntactic Parsers and Dependency Parsers since they are not generally web scalable. Pattern learning algorithms exploit local features. The patterns developed using the algorithms in this chapter will be later used in Chapter 5 to extract knowledge from text.

3.2 Simple Surface Patterns

Surface text patterns are the simplest patterns used in Text Mining technology. For example, to mine the "author-book" semantic relation, patterns such as *X is the author of Y* and *X's Y* (e.g., *Leo Tolstoy is the author of War and Peace*; *Leo Tolstoy's war and Peace*) could be used. We describe an algorithm [60] to learn such simple surface patterns automatically, and provide an example.

3.2.1 Algorithm 1: Collection of Patterns

A table of patterns is constructed for each individual semantic type by the following procedure.

1. Select an example for a given question type. For example, for the BIRTHYEAR[1] semantic relation, we select *Mozart 1756* (we refer to *Mozart* as the question term and *1756* as the answer term).

2. Submit the question and the answer term as queries to a search engine. Thus, we give the query +*"Mozart"* +*"1756"* to AltaVista [2] (http://www.altavista.com).

3. Download the top 1000 web documents provided by the search engine.

[1] BIRTHYEAR semantic relations are characterized by answers to the question: Which year was X born?

[2] We chose AltaVista to perform our experiments because it allowed automated querying and displayed at most 1000 links for each query

4. Apply a sentence breaker to the documents.

5. Retain only those sentences that contain both the question and the answer term. Tokenize the input text, smooth variations in white space characters, and remove HTML and other extraneous tags, to allow simple regular expression matching tools such as egrep to be used.

6. Pass each retained sentence through a suffix tree constructor. This finds all substrings, of all lengths, along with their counts. For example consider the sentences *The great composer Mozart (1756–1791) achieved fame at a young age*, *Mozart (1756–1791) was a genius*, and *The whole world would always be indebted to the great music of Mozart (1756–1791)*. The longest matching substring for all 3 sentences is *Mozart (1756–1791)*, which the suffix tree would extract as one of the outputs, along with the frequency count of 3.

7. Pass each phrase in the suffix tree through a filter to retain only those phrases that contain both the question and the answer term. For the example, we extract only those phrases from the suffix tree that contain the words *Mozart* and *1756*.

8. Replace the word for the question term by the tag "<NAME>" and the word for the answer term by the term "<ANSWER>".

This procedure is repeated for different examples of the same question type. For BIRTHDATE we also use *Gandhi 1869*, *Newton 1642*, etc. For BIRTHDATE, the above steps produce the following output:

a. born in <ANSWER> , <NAME>

b. <NAME> was born on <ANSWER> ,

c. <NAME> (<ANSWER> -

d. <NAME> (<ANSWER -)

...

These are some of the most common substrings of the extracted sentences that contain both <NAME> and <ANSWER>. Since the suffix tree records all substrings, partly overlapping strings such as **c.** and **d.** are separately saved, which allows us to obtain separate counts of their occurrence frequencies. As will be seen later, this allows us to differentiate patterns such as **d.** (which records a still living person, and is quite precise) from its more general substring **c.** (which is less precise).

3.2.2 Algorithm 2: Calculating the precision of each pattern

From among the above extracted patterns we need to extract good patterns. Therefore, we have to measure their utility. We perform this by using the precision measure. To measure precision, we apply the following algorithm:

1. Query the search engine by using only the question term (in the example, only *Mozart*).

2. Download the top 1000 web documents provided by the search engine.

3. As before, segment these documents into individual sentences.

4. Retain only those sentences that contain the question term.

5. For each pattern obtained from Algorithm1, check the presence of each pattern in the sentence obtained from above for two instances:

   i) Presence of the pattern with <ANSWER> tag matched by any word.

   ii) Presence of the pattern in the sentence with <ANSWER> tag matched by the correct answer term. In our example, for the pattern "<NAME> was born in <ANSWER>" we check the presence of the following strings in the answer sentence

   i) *Mozart* was born in *<ANY_WORD>*.

   ii) *Mozart* was born in *1756*.

6. Calculate the precision of each pattern by the formula P = Ca / Co where,

   Ca = total number of patterns with the answer term present

   Co = total number of patterns present with answer term replaced by any word

7. Retain only the patterns matching a sufficient number of examples (we experimentally choose the number of examples > 5). We obtain a table of regular expression patterns for a given question type, along with the precision of each pattern. This precision is the probability of each pattern containing the answer and follows directly from the principle of maximum likelihood estimation.

3.2.3 Discussion

For BIRTHDATE the following table is obtained:

PRECISION	PATTERNS
1.0	<NAME>(<ANSWER> –)
0.85	<NAME> was born on <ANSWER>,
0.6	<NAME> was born in <ANSWER>
0.59	<NAME> was born <ANSWER>
0.53	<ANSWER> <NAME> was born
0.50	<NAME> (<ANSWER>
0.36	<NAME> (<ANSWER> –
....	

For a given question type a good range of patterns was obtained by giving the system as few as 10 examples. The rather long list of patterns obtained would have been very difficult for any human to come up with manually. The question term could appear in the documents obtained from the web in various ways. Thus *Mozart* could be written as *Wolfgang Amadeus Mozart*, *Mozart, Wolfgang Amadeus*, *Amadeus Mozart* or *Mozart*.

This is part of a more general problem and hard problem of identifying synonyms. To tackle this problem we manually specify the various ways in which the question term could be specified in the text. The presence of any of these names would cause it to be tagged as the original question term *Mozart*. The same arrangement is also done for the answer term so that presence of any variant of the answer term would cause it to be treated exactly like the original answer term. However, it would very interesting to solve this problem using automatic techniques. While easy to do for BIRTHDATE, this step can be problematic for question types such as DEFINITION, which may contain various acceptable answers. In general, the input example terms have to be carefully selected so that the questions they represent do not have a long list of possible answers, as this would affect the confidence of the precision scores for each pattern. All the answers need to be enlisted to ensure a high confidence in the precision score of each pattern, in the present framework. The precision of the patterns obtained from one QA-pair example in Algorithm 1 is calculated from the documents obtained in Algorithm 2 for other examples of the same question type. In other words, the precision scores are calculated by cross-checking the patterns across various examples of the same type. This step proves to be very significant as it helps to eliminate dubious patterns, which may appear because the contents of two or more websites may be the same, or the same web document reappears in the search engine output for algorithms 1 and 2. Algorithm 1 does not explicitly specify any particular question type. Judicious choice of the QA

example pair therefore allows it to be used for many question types without change. We present more examples in the experiments subsection, 3.2.6.

3.2.4 Suffix Trees: Why Use Them?

Suffix Tree, originally introduced by Weiner [74], is a data structure that stores in linear space all suffixes for a given string. Suffix Trees are useful for a variety of problems wherein the structure of the string has to be analyzed. In particular, they provide an elegant solution to the problem of finding the longest common substring from multiple input strings. This solution is linear in time on the size of the input data. Such techniques have been used extensively in the field of Computational Biology for finding out the longest DNA sequences in genes for various purposes. Gusfield [33] gives a detailed description about Suffix Trees.

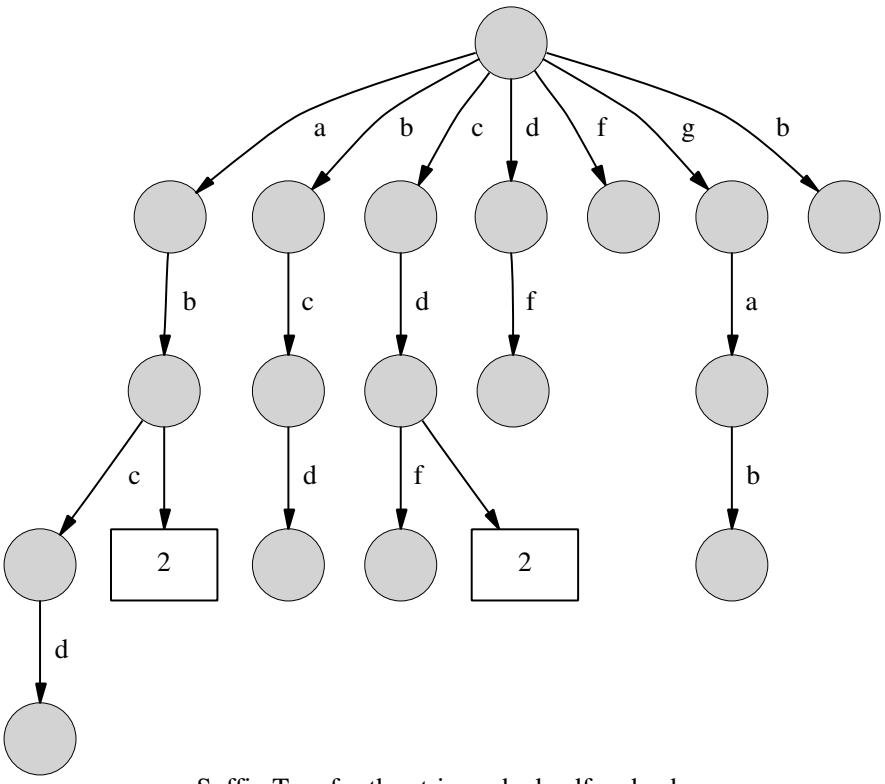
A Suffix Tree has the following properties [33].

1. Every edge of the tree represents a non-empty substring of S.

2. No two edges beginning from a node can have the same edge-labels.

3. Each path starting from the root to any leaf of the Suffix Tree represents a unique substring.

4. Information regarding the counts of the substrings is obtained by maintaining a count field on each edge of the Suffix Tree.



Suffix Tree for the strings abcd, cdf and gab.
Sub-strings with a minimum length of 2, and with a score greater than 1
are ab and cd as shown.

Figure 3.1: Suffix Tree

For Figure 3.1, consider the strings *abcd*, *cdf*, and *gab* The set of suffixes for the strings are:

1. { abcd , bcd , cd , d }

2. { cdf , df , f }

3. { gab , ab , b }

Each of the suffixes is passed to the Suffix Tree is shown in Figure 3.1. Only the leaves that represent a substring of length greater than 2 and a count greater than 2 are highlighted. Substrings *ab* and *cd* are the only ones that satisfy the given criteria. Similarly, Suffix Trees can be applied at the sentence level, where each individual word corresponds to a character in the above example.

3.2.5 Finding Answers for New Questions

Having acquired the extraction of patterns, we could simply use them to filter candidate answers in a factoid QA system and pinpoint the exact answers or perform text harvesting. Factoid Question Answering (QA) is defined as the task of answering fact-based questions phrased in Natural Language.We used them for a QA system by using the following algorithm:

1. Determine the question type of the new question. We use our existing QA system [38] [39] to do so.

2. Identify the question term from our existing system.

3. Create a query from the question term and perform IR (by using a given answer document corpus such as the TREC-10 collection or web search otherwise).

4. Segment the documents obtained into sentences and smooth out white space variations and HTML and other tags, as before.

5. Replace the question term in each sentence by the question tag ("<NAME>", in the case of BIRTHYEAR).

6. Using the pattern table developed for that particular question type, search for the presence of each pattern. Select words matching the tag "<ANSWER>" as the answer.

7. Sort these answers by their pattern's precision scores. Discard duplicates (by elementary string comparisons). Return the top 5 answers.

3.2.6 Experiments

From our Webclopedia QA Typology [39] we selected 6 different question types: BIRTHDATE, LOCATION, INVENTOR, DISCOVERER, DEFINITION, WHY-FAMOUS. The pattern table for each of these question types was constructed using Algorithm 1. Some of the patterns obtained along with their precision are as follows:

BIRTHYEAR

1.0	<NAME> (<ANSWER> -)
0.85	<NAME> was born on <ANSWER> ,
0.6	<NAME> was born in <ANSWER>
0.59	<NAME> was born <ANSWER>
0.53	<ANSWER> <NAME> was born
0.5	- <NAME> (<ANSWER>
0.36	<NAME> (<ANSWER> -
0.32	<NAME> (<ANSWER>) ,
0.28	born in <ANSWER> , <NAME>
0.2	of <NAME> (<ANSWER>

INVENTOR

1.0	<ANSWER> invents <NAME>
1.0	the <NAME> was invented by <ANSWER>
1.0	<ANSWER> invented the <NAME> in
1.0	<ANSWER> ' s invention of the <NAME>
1.0	<ANSWER> invents the <NAME> .
1.0	<ANSWER> ' s <NAME> was
1.0	<NAME> , invented by <ANSWER>
1.0	<ANSWER> ' s <NAME> and
1.0	that <ANSWER> ' s <NAME>
1.0	<NAME> was invented by <ANSWER> ,

DISCOVERER

1.0	when <ANSWER> discovered <NAME>
1.0	<ANSWER> ' s discovery of <NAME>
1.0	<ANSWER> , the discoverer of <NAME>
1.0	<ANSWER> discovers <NAME> .
1.0	<ANSWER> discover <NAME>
1.0	<ANSWER> discovered <NAME> , the
1.0	discovery of <NAME> by <ANSWER>.
0.95	<NAME> was discovered by <ANSWER>
0.91	of <ANSWER> ' s <NAME>
0.9	<NAME> was discovered by <ANSWER> in

DEFINITION

1.0	<NAME> and related <ANSWER>s
1.0	<ANSWER> (<NAME> ,

1.0 <ANSWER> , <NAME> .
1.0 , a <NAME> <ANSWER> ,
1.0 (<NAME> <ANSWER>) ,
1.0 form of <ANSWER> , <NAME>
1.0 for <NAME> , <ANSWER> and
1.0 cell <ANSWER> , <NAME>
1.0 and <ANSWER> > <ANSWER> > <NAME>
0.94 as <NAME> , <ANSWER> and

WHY-FAMOUS
1.0 <ANSWER> <NAME> called
1.0 laureate <ANSWER> <NAME>
1.0 by the <ANSWER> , <NAME> ,
1.0 <NAME> - the <ANSWER> of
1.0 <NAME> was the <ANSWER> of
0.84 by the <ANSWER> <NAME> ,
0.8 the famous <ANSWER> <NAME> ,
0.73 the famous <ANSWER> <NAME>
0.72 <ANSWER> > <NAME>
0.71 <NAME> is the <ANSWER> of

LOCATION
1.0 <ANSWER> ' s <NAME> .
1.0 regional : <ANSWER> : <NAME>
1.0 to <ANSWER> ' s <NAME> ,
1.0 <ANSWER> ' s <NAME> in
1.0 in <ANSWER> ' s <NAME> ,
1.0 of <ANSWER> ' s <NAME> ,
1.0 at the <NAME> in <ANSWER>
0.96 the <NAME> in <ANSWER> ,
0.92 from <ANSWER> ' s <NAME>
0.92 near <NAME> in <ANSWER>

For each question type, we extracted all the corresponding questions from the TREC-10 set. Two sets of experiments were performed. In the first case, the TREC corpus was used as the input source and IR was performed by the IR component of our QA system

Table 3.1: Performance of the system measured by MRR on TREC and Web Corpus.

Question type	Number of questions	TREC MRR	Web MRR
BIRTHYEAR	8	0.48	0.69
INVENTOR	6	0.17	0.58
DISCOVERER	4	0.13	0.88
DEFINITION	102	0.34	0.39
WHY-FAMOUS	3	0.33	0.00
LOCATION	16	0.75	0.86

[49]. In the second case, the web was the input source and the IR was performed by the AltaVista search engine.

Results of the experiments are measured by Mean Reciprocal Rank (MRR) score. The mean reciprocal rank (MRR) of a system is defined as the average of the inverse rank of correct answers. MRR is computed as

$$\text{MRR} = \frac{1}{N}\sum_{i=0}^{N}\begin{cases}\frac{1.0}{\text{rank}_i} & \text{if correct answer found}\\ 0 & \text{otherwise}\end{cases}$$

where N is the number of questions and $rank_i$ is the first rank at which a judge found a correct answer. Answers with ranks of 1–5 are considered. In the case that none of the five responses are correct, MRR is assigned 0 by definition.

The results Table 3.1 indicate that the system performs better on the Web data than on the TREC corpus. The abundance of data on the web makes it easier for the system to locate answers with high precision scores (the system finds many examples of correct answers among the top 20 when using the Web as the input source). A similar result

for QA was obtained by Brill [12]. The TREC corpus does not have enough candidate answers with high precision score and has to settle for answers extracted from sentences matched by low precision patterns. The WHY-FAMOUS question type is an exception and may be due to the fact that the system was tested on a small number of questions.

3.2.7 Drawbacks

While the surface pattern learning technique was very simple it had the following shortcomings:

1. It does not make use of several available off-the-shelf tools, such as Named-Entity and Part of Speech taggers. Almost all existing QA systems make use of such tools and find that they are key to answer pinpointing.

2. The text pattern method of identifying answers does not model the technique of obtain exact answers. Rather, it retrieves a text segment of 50 bytes dictated by an answer pattern. The following example illustrates this point:

   Question: *Who are the Aborigines?*

   Answer: *...time helping Aborigines, an ancient tribe whose plight has been similar to American Indians.*

   Pattern: *<Question_Term>, an <Answer>*

   Here the winning pattern has no information about the expected length of the answer.

3. Classification of a given question so that patterns of a particular group could be applied, is a very difficult problem. Since the patterns convey very specific information, question classification has to be very precise. E.g., each of the following question asking, about TIME would have different sets of patterns:

    1 When was A born?
    2 When did A become the President of B?
    3 When did C achieve statehood?
    4 When did A write Y?

    To obtain a very through classification of questions we need to have a large amount of training data and even then we may be presented with new unseen questions which cannot be easily classified into the pre-existing set of classes. Hence we need to devise a back-up framework that would take precedence, if the question fails to classify into one of the existing set of classes.

4. The text patterns are in essence not real regular expression patterns but would rather qualify as templates. The power of such text patterns could be further increased if one could also learn non-consecutive patterns.

3.3 Multilevel Patterns

To overcome the drawbacks of the basic surface text pattern matching algorithm, we explore other sequence aligning algorithms. One of the basic sequence aligning algorithm is the basic edit distance learning algorithm.

3.3.1 Basic Edit Distance

The basic edit distance algorithm, originally devised by Wagner and Fischer [73], is a dynamic programming algorithm. This algorithm finds the optimal alignment of two strings and calculates the minimal edit distance between them. The edit distance is defined as the number of edit operations required to change one string to another. Three types of edit operations are defined, viz. substitution, insertion, and deletion. Given two strings of length m and n, the edit distance algorithm calculates the minimal edit distance in $O(nm)$ time and the optimal path in $O(n+m)$ time. The algorithm consists of two parts. The first path of the algorithm calculates the minimum distance between two strings and the second part of the algorithm retrieves the optimal alignment.

The first part of the algorithm is given in Figure 3.2. D[i,j] is the minimal edit distance between two substrings $a(1,i)$ and $b(1,j)$, where $i<=m$ and $j<=n$. m and n are the lengths of strings $a(1,m)$ and $b(1,n)$ respectively. The variables cost(substitution),

```
(1)   D[0,0]=0
(2)   for i=1 to m do
(3)      D[i,0] = D[i-1,0] + cost(insertion)
(4)   for j = 1 to n do
(5)      D[0,j] = D[0,j-1] + cost(deletion)
(6)   for i = 1 to m do
(7)      for j = 1 to n do
(8)         D[i,j] = min(D[i-1,j-1] + cost(substitution),
                         D[i-1,j] + cost(insertion),
                         D[i,j-1] + cost(deletion) )
(9)   print(D[m,n])
```

Figure 3.2: Basic Edit Distance Calculation Algorithm.

cost(insertion), and cost(deletion) holds the cost values associated with substitution, insertion, and deletion respectively. This algorithm is a dynamic programming algorithm for the simple reason that D[i,j] is calculated in a recursive fashion using *D[i-1,j]*, *D[i, j-1]* or *D[i-1, j-1]* as given in line (8) of Figure 3.2.

Figure 3.3 represents the second part of the algorithm for optimal alignment. The algorithm basically uses the distance measures from the first path of the algorithm and traces its path from *D[n,m]* to *D[0,0]*. During this motion, the algorithm outputs certain strings based on the decision of insertion, substitution, or deletion decisions made by the previous part of the algorithm. These choices could be stored in an *n*m* matrix.

By assuming *cost(insertion)* to be 3; *cost(deletion)* to be 3; and *cost(substitution)* to be 0 for identical segments and 3 otherwise we get:

```
(1)   i = n , j = m
(2)   while i ≠ 0 and j ≠ 0
(3)      if D[i,j] = D[i-1,j] + cost(insertion)
(4)         print(a_i ,-)
(5)         i = i - 1
(6)      elseif D[i,j] = D[i,j-1] + cost(deletion)
(7)         print(-,b_j )
(8)         j = j - 1
(9)      else
(10)        print (a_i ,b_j )
(11)        i = i - 1 , j = j - 1
```

Figure 3.3: Optimal Alignment Retrieval Algorithm.

Input:

a	b	x	c	y	d	e
a	b	c	z	d	e	

Output:

a	b	x	c	y	d	e
a	b	_	c	z	d	e

3.3.2 Non Sequence Pattern Learning Algorithm

We wish to exploit the above algorithm to extract more sophisticated patterns. To accommodate near-miss patterns, in which two strings may be off by one or more words as in the sequence above, we follow regular expressions and introduce two wild-card

```
(1)     i = n , j = m
(2)     while i ≠ 0 and j ≠ 0
(3)        if D[i,j] = D[i-1,j] + cost(insertion)
(4)           print(*s*)
(5)           i = i - 1
(6)        elseif D[i,j] = D[i,j-1] + cost(deletion)
(7)           print(*s*)
(8)           j = j - 1
(9)        elseif a_i ≠ b_j
(10)          print (*g*)
(11)          i = i - 1 , j = j - 1
(12)       elseif a_i = b_j
(13)          print (a_i)
(14)          i = i - 1 , j = j - 1
```

Figure 3.4: Non sequence Pattern Learning.

operators, called skip *(*s*)* and wild-card *(*g*)*. The skip operator *(*s*)* represents 0 or 1 instance of any word (similar to $\backslash S*$ operator in Perl), while the wild-card operator *(*g*)* represents exactly 1 instance of any word (similar to $\backslash S+$ operator in Perl). Incorporating these changes into the algorithm for retrieving optimal pattern reduces to the one shown in Figure 3.4.

This modified algorithm would give the following:

Input:

a	b	x	c	y	d	e
a	b	c	z	d	e	

Output:

a	b	(*s*)	c	(*g*)	d	e

We call the algorithm in Figure 3.4 *Single level non sequence pattern learning algorithm*.

3.3.3 Multilevel Non Sequence Learning Algorithm

Above patterns work only on the lexical level (words only). But this means we need a new pattern for each word, even when words belonging to a particular class provide identical patterns. We would like to replace words with identical pattern behavior with their class name. Several classes that may be useful come to mind, including POS (Part of Speech), Syntactic, and Semantic information. POS and Syntactic patterns are frequently employed by NLP researchers. For example, to learn causal relations, Girju et al. [29] employ patterns such as *<NP1> is caused by <NP2>* , while Fleischman et al. [28] use POS patterns such as *CN PN*[3] and appositives to extract relations for questions such as *Who is X?* . The range and value of the results vary dramatically by the number and accuracy of the patterns used. However, no work has been done in learning such sophisticated multi-level patterns automatically in a principled way. We define

[3] *CN= Common Noun PN=Proper Noun*

multi-level patterns as a sequence of strings wherein information from different sources (level), such as POS tagger, syntactic parser, and named entity tagger are combined.

Given two sentences with their surface form and the part of speech form we need to find the optimal multi-level alignment. For example, consider the following two sentences:

1. Platinum is a precious metal.

2. Molybdenum is a metal.

These two sentences look very similar, the only two differences being:

1. The second sentence contains the word *Molybdenum* instead of the word *Platinum*.

2. The word *precious* is absent in the second sentence.

Applying a POS tagger would deliver us the following output:

Surface	Platinum	is	a	precious	metal	.
POS	NNP	VBZ	DT	JJ	NN	.

Surface	Molybdenum	is	a	metal	.
POS	NNP	VBZ	DT	NN	.

A very good pattern to generalize from the alignment of these two strings would be:

Surface		is	a		metal	.
POS	NNP	VBZ	DT		NN	.

We conveniently write such an alignment as *_NNP is a (<any_word>)?* metal . where *_NNP* represents the POS level and *(<any_word>)?* represents a wild-card word which can appear 0 or 1 time (similar to the Perl $\backslash S*$ operator). By using the previously introduced operators for skip and replacement, we can also represent the same pattern by *_NNP is a (*s*) metal* .

The algorithm for multi-level pattern learning also follows the basic edit distance algorithm with slight modifications. Consider two strings *a(1,n)* and *b(1,m)* of lengths n and m respectively. Let a_1*(1,n)* and a_2*(1,n)* be the level 1 and level 2 representations for the string *a(1,n)*. Similarly, let b_1*(1,m)* and b_2*(1,m)* be the level 1 and level 2 representations for the string *b(1,m)*. The algorithm consists of two parts: calculation of minimal edit distance and retrieval of optimal pattern.

The initial part of the algorithm for determining the optimal edit distance remains the same as in Figure 3.2. However, for the optimal alignment of patterns the algorithm is modified as shown in Figure 3.5.

For a pair of string of length *x* and *y* respectively, the algorithm has to fill up a grid of *x.y*. Hence, the complexity of the algorithm is *O(xy)*. This holds true for both the

```
(1)   i = n , j = m
(2)   while i ≠ 0 and j ≠ 0
(3)      if D[i,j] = D[i-1,j] + cost(insertion)
(4)         print(*s*)
(5)         i = i - 1
(6)      elseif D[i,j] = D[i,j-1] + cost(deletion)
(7)         print(*s*)
(8)         j = j - 1
(9)      elseif a2i = b2j
(10)        print (a2i)
(11)        i = i - 1 , j = j - 1
(12)     elseif a1i = b1j
(13)        print (a1i)
(14)        i = i - 1 , j = j - 1
(15)     else
(16)        print (*g*)
(17)        i = i - 1 , j = j - 1
```

Figure 3.5: Non sequence multi level pattern learning.

single level and the multi-level non-sequence pattern learning algorithm. However, the simple surface pattern learning algorithm has a time complexity of only $O(x+y)$.

3.4 When do Pattern-based methods fail?

It is very clear that bootstrapping pattern-based methods are generally very simple and powerful. But there are many cases where they fail. The following lists illustrates some of the shortcomings:

1. Patterns predominantly work only for binary relations. Thus, these patterns may work for learning birthyear relationships but may fail for relationships which have have more than two slots. For example, it is easier to learn the binary relation *George Bush:president* (*George Bush is the president*). However, it is going to be hard to learn the tertiary relation *George Bush:president:United States* (*Bush is the president of United States*).

2. Web redundancy is the key to achieve high accuracy and recall. However, patterns fail when the relations involved are very rare and cannot be captured by only a few patterns.

3. All these bootstrapping pattern based techniques work only in case of dominant relations, ie. the binary relation captured does not satisfy any other relation [6].

Thus, the relation BIRTHYEAR is easier to capture than the relation WHY-FAMOUS as demonstrated in Section 3.2.6. For the same reason, these kind of pattern-learning techniques fail to work for cases, where we need to identify relationships between two verbs.

3.5 Conclusion

In this chapter, we described a pattern learning approach that was designed to work on a web-scale corpus. Having learned these pattern learning algorithms, we are now equipped with one of the core technology to extract knowledge from text. Details are explained in Chapter 5. In the next chapter, we describe a co-occurrence based technique

Chapter 4

High Speed Clustering

4.1 Introduction

In the previous chapter, we described a pattern learning algorithm that exploits local features to extract relations from text (or answer questions). We then focus our attention on a different algorithm, viz. one that involves clustering. Clustering algorithms use global (co-occurrence) features to extract knowledge from text. In this chapter, we propose a web scalable solution to clustering nouns, which employs randomized algorithms. In doing so, we are going to explore the literature and techniques of randomized algorithms. All clustering algorithms make use of some distance similarity (e.g., cosine similarity) to measure pair wise distance between sets of vectors. Assume that we are given n points to cluster with a maximum of k features. Calculating the full similarity matrix would take time complexity $O(n^2k)$. With large amounts of data, say

n in the order of millions or even billions, having an $O(n^2k)$ algorithm would be very infeasible. To be scalable, we ideally want our algorithm to be proportional to $O(nk)$.

Fortunately, we can borrow some ideas from the Math and Theoretical Computer Science community to tackle this problem. The crux of our solution lies in defining Locality Sensitive Hash (LSH) functions. LSH functions involve the creation of short signatures (fingerprints) for each vector in space such that those vectors that are closer to each other are more likely to have similar fingerprints. LSH functions are generally based on randomized algorithms and are probabilistic. We present LSH algorithms that can help reduce the time complexity of calculating our distance similarity atrix to $O(nk)$.

Rabin [59] proposed the use of hash functions from random irreducible polynomials to create short fingerprint representations for very large strings. These hash function had the nice property that the fingerprint of two identical strings had the same fingerprints, while dissimilar strings had different fingerprints with a very small probability of collision. Broder [13] first introduced LSH. He proposed the use of Min-wise independent functions to create fingerprints that preserved the Jaccard similarity between every pair of vectors. These techniques are used today, for example, to eliminate duplicate web pages. Charikar [18] proposed the use of random hyperplanes to generate

an LSH function that preserves the cosine similarity between every pair of vectors. Interestingly, cosine similarity is widely used in NLP for various applications such as clustering.

In this chapter, we perform high speed similarity list creation for nouns collected from a huge web corpus. We linearize this step by using the LSH proposed by Charikar [18]. This reduction in complexity of similarity computation makes it possible to address vastly larger datasets, at the cost, as shown in Section 4.5, of only little reduction in accuracy. In our experiments, we generate a similarity list for each noun extracted from 70 million page web corpus. Although the NLP community has begun experimenting with the web, we know of no work in published literature that has applied complex language analysis beyond IR and simple surface-level pattern matching.

4.2 Theory

The core theory behind the implementation of fast cosine similarity calculation can be divided into two parts:

1. Developing LSH functions to create signatures;

2. Using fast search algorithm to find nearest neighbors.

We describe these two components in greater detail in the next subsections.

4.2.1 LSH Function Preserving Cosine Similarity

We first begin with the formal definition of cosine similarity.

Definition: Let u and v be two vectors in a k dimensional hyperplane. Cosine similarity is defined as the cosine of the angle between them: $cos(\theta(u, v))$. We can calculate $cos(\theta(u, v))$ by the following formula:

$$cos(\theta(u, v)) = \frac{|u.v|}{|u||v|} \tag{4.1}$$

Here $\theta(u, v)$ is the angle between the vectors u and v measured in radians. $|u.v|$ is the scalar (dot) product of u and v, and $|u|$ and $|v|$ represent the length of vectors u and v respectively.

If $\theta(u, v) = 0$, then $cos(\theta(u, v)) = 1$

and if $\theta(u, v) = \pi/2$, then $cos(\theta(u, v)) = 0$

The LSH function for cosine similarity as proposed by Charikar [18] is given by the following theorem:

Theorem: Suppose we are given a collection of vectors in a k dimensional vector space (as written as R^k). Choose a family of hash functions as follows: Generate a

spherically symmetric random vector r of unit length from this k dimensional space. We define a hash function, h_r, as:

$$h_r(u) = \begin{cases} 1 & : \quad r.u \geq 0 \\ 0 & : \quad r.u < 0 \end{cases} \tag{4.2}$$

Then for vectors u and v,

$$Pr[h_r(u) = h_r(v)] = 1 - \frac{\theta(u,v)}{\pi} \tag{4.3}$$

Proof of the above theorem is given by Goemans and Williamson [30]. We rewrite the proof here for clarity. The above theorem states that the probability that a random hyperplane separates two vectors is directly proportional to the angle between the two vectors (i,e., $\theta(u,v)$). By symmetry, we have $Pr[h_r(u) \neq h_r(v)] = 2Pr[u.r \geq 0, v.r < 0]$. This corresponds to the intersection of two half spaces, the dihedral angle between which is θ. Thus, we have $Pr[u.r \geq 0, v.r < 0] = \theta(u,v)/2\pi$. Proceeding we have $Pr[h_r(u) \neq h_r(v)] = \theta(u,v)/\pi$ and $Pr[h_r(u) = h_r(v)] = 1 - \theta(u,v)/\pi$. This completes the proof.

Hence from equation 4.3 we have,

$$cos(\theta(u,v)) = cos((1 - Pr[h_r(u) = h_r(v)])\pi) \tag{4.4}$$

This equation gives us an alternate method for finding cosine similarity. Note that the above equation is probabilistic in nature. Hence, we generate a large (d) number of random vectors to achieve the process. Having calculated $h_r(u)$ with d random vectors for each of the vectors u, we apply equation 4.4 to find the cosine distance between two vectors. As we generate more number of random vectors, we can estimate the cosine similarity between two vectors more accurately. However, in practice, the number (d) of random vectors required is highly domain dependent, i.e., it depends on the value of the total number of vectors (n), features (k) and the way the vectors are distributed. Using d random vectors, we can represent each vector by a bit stream of length d.

Carefully looking at equation 4.4, we can observe that $Pr[h_r(u) = h_r(v)] = 1 - (hamming\ distance)/d$[1] . Thus, the above theorem, converts the problem of finding cosine distance between two vectors to the problem of finding hamming distance between their bit streams (as given by equation 4.4). Finding hamming distance between two bit streams is faster and highly memory efficient. Also worth noting is that this step could be considered as dimensionality reduction wherein we reduce a vector in k dimensions to that of d bits while still preserving the cosine distance between them.

[1]Hamming distance is the number of bits which differ between two binary strings. Thus the hamming distance between two strings A and B is $\sum_i |A_i - B_i|$.

4.2.2 Fast Search Algorithm

To calculate the fast hamming distance, we use the search algorithm PLEB (Point Location in Equal Balls) first proposed by Indyk and Motwani [42]. This algorithm was further improved by Charikar [18]. This algorithm involves random permutations of the bit streams and their sorting to find the vector with the closest hamming distance. The algorithm given in Charikar [18] is described to find the nearest neighbor for a given vector. We modify it so that we are able to find the top B closest neighbor for each vector. We omit the math of this algorithm but we sketch its procedural details in the next section. Interested readers are further encouraged to read Theorem 2 from Charikar [18] and Section 3 from Indyk and Motwani [42].

4.3 Algorithmic Implementation

In the previous section, we introduced the theory for calculation of fast cosine similarity. We implement it as follows:

1. Initially we are given n vectors in a huge k dimensional space. Our goal is to find all pairs of vectors whose cosine similarity is greater than a particular threshold.

2. Choose d number of ($d << k$) unit random vectors $\{r_0, r_1, ..., r_d\}$ each of k dimensions.

A k dimensional unit random vector, in general, is generated by independently sampling a Gaussian function with mean 0 and variance 1, k number of times. Each of the k samples is used to assign one dimension to the random vector. We generate a random number from a Gaussian distribution by using Box-Muller transformation [10].

3. For every vector u, we determine its signature by using the function $h_r(u)$ (as given by equation 4.4). We can represent the signature of vector u as: $\bar{u} = \{h_{r1}(u), h_{r2}(u), ..., h_{rd}(u)\}$. Each vector is thus represented by a set of a bit streams of length d. Steps 2 and 3 takes $O(nk)$ time (We can assume d to be a constant since $d << k$).

4. The previous step gives n vectors, each of them represented by d bits. For calculation of fast hamming distance, we take the original bit index of all vectors and randomly permute them (see Appendix A for more details on random permutation functions). A random permutation can be considered as random jumbling of the bits of each vector[2]. A random permutation function can be approximated by the following function:

$$\pi(x) = (ax + b) \text{mod p} \tag{4.5}$$

[2]The jumbling is performed by a mapping of the bit index as directed by the random permutation function. For a given permutation, we reorder the bit indexes of all vectors in similar fashion. This process could be considered as column reording of bit vectors.

where, p is prime and $0 < a < p$, $0 \leq b < p$, and a and b are chosen at random.

We apply q different random permutation for every vector (by choosing random values for a and b, q number of times). Thus for every vector we have q different bit permutations for the original bit stream.

5. For each permutation function π, we lexicographically sort the list of n vectors (whose bit streams are permuted by the function π) to obtain a sorted list. This step takes $O(nlogn)$ time. (We can assume q to be a constant).

6. For each sorted list (performed after applying the random permutation function π), we calculate the hamming distance of every vector with B of its closest neighbors in the sorted list. If the hamming distance is below a certain predetermined threshold, we output the pair of vectors with their cosine similarity (as calculated by equation 4.4). Thus, B is the beam parameter of the search. This step takes $O(n)$, since we can assume B, q, d to be a constant.

Why does the fast hamming distance algorithm work? The intuition is that the number of bit streams, d, for each vector is generally smaller than the number of vectors n (ie. $d << n$). Thus, sorting the vectors lexicographically after jumbling the bits will likely bring vectors with lower hamming distance closer to each other in the sorted lists.

Overall, the algorithm takes $O(nk + nlogn)$ time. However, for noun clustering, we generally have the number of nouns, n, smaller than the number of features, k. (i.e.,

$n < k$). This implies $logn << k$ and $nlogn << nk$. Hence the time complexity of our algorithm is $O(nk + nlogn) \approx O(nk)$. This is a huge saving from the original $O(n^2k)$ algorithm. In the next section, we proceed to apply this technique for generating noun similarity lists.

4.4 Building Noun Similarity Lists

A lot of work has been done in the NLP community on clustering words according to their meaning in text [41, 48]. The basic intuition is that words that are similar to each other tend to occur in similar contexts, thus linking the semantics of words with their lexical usage in text. One may ask why is clustering of words necessary in the first place? There may be several reasons for clustering, but generally it boils down to one basic reason: if the words that occur rarely in a corpus are found to be distributionally similar to more frequently occurring words, then one may be able to make better inferences on rare words.

However, to unleash the real power of clustering one has to work with large amounts of text. The NLP community has started working on noun clustering on a few gigabytes of newspaper text. But with the rapidly growing amount of raw text available on the web, one could improve clustering performance by carefully harnessing its power. A core component of most clustering algorithms used in the NLP community is the creation of a similarity matrix. These algorithms are of complexity $O(n^2k)$, where n is the

number of unique nouns and k is the feature set length. These algorithms are thus not readily scalable, and limit the size of corpus manageable in practice to a few gigabytes. Clustering algorithms for words generally use the cosine distance for their similarity calculation [64]. Hence instead of using the usual naive cosine distance calculation between every pair of words we can use the algorithm described in Section 4.3 to make noun clustering web scalable.

To test our algorithm we conduct similarity based experiments on 2 different types of corpus:

1. Web Corpus (70 million web pages, 116GB),

2. Newspaper Corpus (6 GB newspaper corpus)

4.4.1 Web Corpus

We use a corpus of 31 million webpages downloaded from the Internet. These webpages are tokenized, language identified. Only English documents are retained. All duplicate and near duplicate documents are then eliminated. We identify all the nouns in the corpus by using a noun phrase identifier. For each noun phrase, we identify the context words surrounding it. Our context window length is restricted to two words to the left and right of each noun. We use the context words as features of the noun vector.

Table 4.1: Corpus description

Corpus	Newspaper	Web
Corpus Size	6GB	116GB
Unique Nouns	65,547	655,495
Feature size	940,154	1,306,482

4.4.2 Newspaper Corpus

We parse a 6 GB newspaper (TREC9 and TREC2002 collection) corpus using the dependency parser Minipar [47]. We identify all nouns. For each noun we take the grammatical context of the noun as identified by Minipar[3]. We do not use grammatical features in the web corpus since parsing is generally not easily web scalable. This kind of feature set does not seem to affect our results. Curran and Moens [22] also report comparable results for Minipar features and simple word based proximity features. Table 4.1 gives the characteristics of both corpora. Since we use grammatical context, the feature set is considerably larger than the simple word based proximity feature set for the newspaper corpus.

4.4.3 Calculating Feature Vectors

Having collected all nouns and their features, we now proceed to construct feature vectors (and values) for nouns from both corpora using mutual information [20]. We first

[3]We perform this operation so that we can compare the performance of our system to that of Pantel and Lin [55].

construct a frequency count vector $C(e) = (c_{e1}, c_{e2}, ..., c_{ek})$, where k is the total number of features and c_{ef} is the frequency count of feature f occurring in word e. Here, c_{ef} is the number of times word e occurred in context f. We then construct a mutual information vector $MI(e) = (mi_{e1}, mi_{e2}, ..., mi_{ek})$ for each word e, where mi_{ef} is the pointwise mutual information between word e and feature f, which is defined as:

$$mi_{ef} = log \frac{\frac{c_{ef}}{N}}{\sum_{i=1}^{n} \frac{c_{if}}{N} \times \sum_{j=1}^{k} \frac{c_{ej}}{N}} \tag{4.6}$$

where n is the number of words and $N = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}$ is the total frequency count of all features of all words. Mutual information is commonly used to measure the association strength between two words [20]. A well-known problem is that mutual information is biased towards infrequent elements/features. We therefore multiply mi_{ef} with the following discounting factor.

$$mi_{ef} = \frac{c_{ef}}{c_{ef} + 1} \times \frac{min(\sum_{i=1}^{n} c_{ei}, \sum_{j=1}^{k} c_{jf})}{min(\sum_{i=1}^{n} c_{ei}, \sum_{j=1}^{k} c_{jf}) + 1} \tag{4.7}$$

Having thus obtained the feature representation of each noun we can apply the algorithm described in Section 4.3 to discover similarity lists. We report results in the next section for both corpora.

4.5 Evaluation

Evaluating clustering systems is generally considered to be quite difficult. However, we are mainly concerned with evaluating the quality and speed of our high speed randomized algorithm. The web corpus is used to show that our framework is web-scalable, while the newspaper corpus is used to compare the output of our system with the similarity lists output by an existing system, which are calculated using the traditional formula as given in equation 4.1. For this base comparison system we use the one built by Pantel and Lin [55]. We perform 3 kinds of evaluation:

1. Performance of Locality Sensitive Hash Function

2. Performance of fast Hamming distance search algorithm

3. Quality of final similarity lists.

4.5.1 Evaluation of Locality sensitive Hash function

To perform this evaluation, we randomly choose 100 nouns (vectors) from the web collection. For each noun, we calculate the cosine distance using the traditional slow method (as given by equation 4.1), with all other nouns in the collection. This process creates similarity lists for each of the 100 vectors. These similarity lists are cut off at a threshold of 0.15. These lists are considered to be the gold standard test set for our evaluation.

For the above 100 chosen vectors, we also calculate the cosine similarity using the randomized approach as given by equation 4.4 and calculate the mean squared error with the gold standard test set using the following formula:

$$error_{av} = \sqrt{\sum_{i} \left(CS_{real,i} - CS_{calc,i}\right)^2}/total \tag{4.8}$$

where $CS_{real,i}$ and $CS_{calc,i}$ are the cosine similarity scores calculated using the traditional (equation 4.1) and randomized (equation 4.4) technique respectively. i is the index over all pairs of elements that have $CS_{real,i} >= 0.15$

We calculate the error ($error_{av}$) for various values of d, the total number of unit random vectors r used in the process. The results are reported in Table 4.2[4]. As we generate more random vectors, the error rate decreases. For example, generating 10 random vectors gives us a cosine error of 0.4432 (which is a large number since cosine similarity ranges from 0 to 1.) However, generation of more random vectors leads to reduction in error rate as seen by the values for 1000 (0.0493) and 10000 (0.0156). But as we generate more random vectors the time taken by the algorithm also increases. We choose $d = 3000$ random vectors as our optimal (time-accuracy) cut off. It is also very interesting to note that by using only 3000 bits for each of the 655,495 nouns, we are able to measure cosine similarity between every pair of them to within an average error

[4]The time is calculated for running the algorithm on a single Pentium IV processor with 4GB of memory

Table 4.2: Error in cosine similarity

# random vectors d	Average error	Time (hours)
1	1.0000	0.4
10	0.4432	0.5
100	0.1516	3
1000	0.0493	24
3000	0.0273	72
10000	0.0156	241

margin of 0.027. This algorithm is also highly memory efficient since we can represent every vector by only a few thousand bits. Also the randomization process makes the the algorithm easily parallelizable since each processor can independently contribute a few bits for every vector.

4.5.2 Evaluation of Fast Hamming Distance Search Algorithm

We initially obtain a list of bit streams for all the vectors (nouns) from our web corpus using the randomized algorithm described in Section 4.3 (Steps 1 to 3). The next step involves the calculation of hamming distance. To evaluate the quality of this search algorithm we again randomly choose 100 vectors (nouns) from our collection. For each of these 100 vectors we manually calculate the hamming distance with all other vectors in the collection. We only retain those pairs of vectors whose cosine distance (as manually calculated) is above 0.15. This similarity list is used as the gold standard test set for evaluating our fast hamming search.

We then apply the fast hamming distance search algorithm as described in Section 4.3. In particular, it involves steps 4 to 6 of the algorithm. We evaluate the hamming distance with respect to two criteria: **1.** Number of bit index random permutations functions q; **2.** Beam search parameter B.

For each vector in the test collection, we take the top N elements from the gold standard similarity list and calculate how many of these elements are actually discovered by the fast hamming distance algorithm. We report the results in Table 4.3 and Table 4.4 with beam parameters of ($B = 25$) and ($B = 100$) respectively. For each beam, we experiment with various values for q, the number of random permutation function used. In general, by increasing the value for beam B and number of random permutation q , the accuracy of the search algorithm increases. For example in Table 4.4 by using a beam $B = 100$ and using 1000 random bit permutations, we are able to discover 72.8% of the elements of the Top 100 list. However, increasing the values of q and B also increases search time. With a beam (B) of 100 and the number of random permutations equal to 100 (i.e., $q = 1000$) it takes 570 hours of processing time on a single Pentium IV machine, whereas with $B = 25$ and $q = 1000$, reduces processing time by more than 50% to 240 hours.

We could not calculate the total time taken to build noun similarity list using the traditional technique on the entire corpus. However, we estimate that its time taken would be at least 50,000 hours (and perhaps even more) with a few of Terabytes of

Table 4.3: Hamming search accuracy (Beam $B = 25$)

Random permutations q	Top 1	Top 5	Top 10	Top 25	Top 50	Top 100
25	6.1%	4.9%	4.2%	3.1%	2.4%	1.9%
50	6.1%	5.1%	4.3%	3.2%	2.5%	1.9%
100	11.3%	9.7%	8.2%	6.2%	5.7%	5.1%
500	44.3%	33.5%	30.4%	25.8%	23.0%	20.4%
1000	58.7%	50.6%	48.8%	45.0%	41.0%	37.2%

Table 4.4: Hamming search accuracy (Beam $B = 100$)

Random permutations q	Top 1	Top 5	Top 10	Top 25	Top 50	Top 100
25	9.2%	9.5%	7.9%	6.4%	5.8%	4.7%
50	15.4%	17.7%	14.6%	12.0%	10.9%	9.0%
100	27.8%	27.2%	23.5%	19.4%	17.9%	16.3%
500	73.1%	67.0%	60.7%	55.2%	53.0%	50.5%
1000	87.6%	84.4%	82.1%	78.9%	75.8%	72.8%

disk space needed. This is a very rough estimate. The experiment was infeasible. This estimate assumes the widely used reverse indexing technique, where in one compares only those vector pairs that have at least one feature in common.

4.5.3 Quality of Final Similarity Lists

For evaluating the quality of our final similarity lists, we use the system developed by Pantel and Lin [55] as gold standard on a much smaller data set. We use the same 6GB corpus that was used for training by Pantel and Lin [55] so that the results are comparable. We randomly choose 100 nouns and calculate the top N elements closest to each noun in the similarity lists using the randomized algorithm described in Section

Table 4.5: Final Quality of Similarity Lists

	Top 1	Top 5	Top 10	Top 25	Top 50	Top 100
Accuracy	70.7%	71.9%	72.2%	71.7%	71.2%	71.1%

Table 4.6: Sample Top 10 Similarity Lists

JUST DO IT	computer science	TSUNAMI	Louis Vuitton	PILATES
HAVE A NICE DAY	mechanical engineering	tidal wave	PRADA	Tai Chi
FAIR AND BALANCED	electrical engineering	LANDSLIDE	Fendi	Cardio
POWER TO THE PEOPLE	chemical engineering	EARTHQUAKE	Kate Spade	SHIATSU
NEVER AGAIN	Civil Engineering	volcanic eruption	VUITTON	Calisthenics
NO BLOOD FOR OIL	ECONOMICS	HAILSTORM	BURBERRY	Ayurveda
KINGDOM OF HEAVEN	ENGINEERING	Typhoon	GUCCI	Acupressure
If Texas Wasn't	Biology	Mudslide	Chanel	Qigong
BODY OF CHRIST	environmental science	windstorm	Dior	FELDENKRAIS
WE CAN	PHYSICS	HURRICANE	Ferragamo	THERAPEUTIC TOUCH
Weld with your mouse	information science	DISASTER	Ralph Lauren	Reflexology

4.3. We then compare this output to the one provided by the system of Pantel and Lin [55]. For every noun in the top N list generated by our system we calculate the percentage overlap with the gold standard list. Results are reported in Table 4.5. The results shows that we are able to retrieve roughly 70% of the gold standard similarity list. In Table 4.6, we list the top 10 most similar words for some nouns, as examples, from the web corpus.

4.6 Conclusion

The technique described above reduces the amount of time required to calculate similarity matrix among vectors. One can then proceed to apply any clustering algorithm to achieve the desired results. For example, one can apply this technique to k-means

algorithm [52]. K-means has a complexity of $O(n)$. However, application of randomized algorithm would lead to further reduction in complexity of K-means. Thus, in this chapter we presented a technique to speed up the calculation of similarity matrix and should not be confused with the actual clustering algorithm. The similarity matrix is generally used as an input to a clustering algorithm. In the next chapter, we make use of a clustering Algorithm called CBC (Clustering by Committee) [55] to generate huge *is-a* lists.

NLP researchers have just begun leveraging the vast amount of knowledge available on the web. However, most language analysis tools are too infeasible to run on the scale of the web. A case in point is generating noun similarity lists using co-occurrence statistics, which has quadratic running time on the input size. In this chapter, we solve this problem by presenting a randomized algorithm that linearizes this task and limits memory requirements. Experiments show that our method generates cosine similarities between pairs of nouns within a score of 0.03.

In many applications, researchers have shown that more data equals better performance [4, 5, 22]. Moreover, at the web-scale, we are no longer limited to a snapshot in time, which allows broader knowledge to be learned and processed. Randomized algorithms provide the necessary speed and memory requirements to tap into terascale text sources. We hope that randomized algorithms will make other NLP tools feasible

at the terascale and we believe that many algorithms will benefit from the vast coverage of our newly created noun similarity list.

Chapter 5

Knowledge Harvesting

5.1 Introduction

In this chapter, we apply the pattern learning and fast clustering algorithm described in the previous chapters to harvest knowledge from text.

5.2 Data Source

To perform our experiments, we need a large terabyte sized corpus. Getting a Terabyte of data is not a trivial task, since it is not readily available. We had to manually download data from the Internet. The following describes the process briefly:

1. We set up a spider to download roughly 70 million web pages from the Internet to fill up 1 Terabyte of disk. Initially, we used the links from Open Directory project[1] as seed links for our spider.

2. Each webpage is then stripped of HTML tags, tokenized, and sentence segmented.

3. Each document is language identified by the software TextCat[2], which implements the paper by Cavnar and Trenkle [16]. This paper uses well known n-gram algorithm for language identification. We retain only English documents.

4. The web contains a lot of duplicate or near-duplicate documents. Eliminating them is critical for obtaining better statistics from our collection. One simple example of near duplicate webpage is the presence of different UNIX man pages on the web maintained by different universities. Most of the pages are almost identical copies of each other, except for the presence of email (or name) of the local administrator. The problem of identifying near duplicate documents in linear time is not trivial. We eliminate duplicate and near duplicate documents by using the algorithm described by Kolcz et al. [44]. This process of duplicate elimination is carried out in linear time and involves the creation of signatures for each document. Signatures are designed so that duplicate and near duplicate

[1] http://www.dmoz.org/
[2] http://odur.let.rug.nl/~vannoord/TextCat/

documents have the same signature. This algorithm is remarkably fast and has high accuracy. This entire process of removing non English documents and duplicate (and near-duplicate) documents reduces our document set from 70 million web pages to roughly 31 million web pages. This represents roughly 116GB of uncompressed text.

Hence, from a Terabyte of downloaded, we were able to use only 116GB of corpus ($\approx$ 12%). In the next section, we study the learning curve for extracting *is-a* relation from this corpus. In the next section we describe the implementation of algorithms to harvest knowledge from this corpus.

5.3 Pattern-based system: Instance-concept extraction

5.3.1 Multi-level Pattern Algorithm Implementation

The multi-level pattern-learning algorithm introduced in the previous chapter is very general and can be applied to a variety of information extraction/text-mining tasks. We perform a case study of the pattern-learning algorithm by selecting one particular domain, viz. learning concept-instance relations. This domain is roughly characterized by the questions of type *What is X?* or *Who is X?*. *What/Who is* questions have proved to be some of the most difficult ones in recent QA research. The implementation of the algorithm involves the following steps:

Table 5.1: Some seeds used for instance-concept pairing.

X (Instance)	Y (Concept)
platinum	metal
caldera	lava lake
leukemia	cancer
molybdenum	metal
meerkat	mongoose

1. As seeds, we use examples of questions *What is X?* and their corresponding answers Y . E.g., *What is platinum? metal*.

2. For every such seed-answer pair (X,Y), we extract at most 100 sentences from a search engine that contains both these terms.

3. We apply a Part of Speech tagger [11] to each such sentence.

4. We pass every pair of sentences through the multi-level pattern learning algorithm and extract the optimal pattern given by the algorithm. We do some book-keeping to learn different POS variations of the anchors (X,Y).

5. We retain the top K patterns based on their frequencies.

For one experiment, we used 119 question and answer pairs as seeds. These question-answer pairs were selected from the TREC9 [69] and TREC10 [70] question sets. Some of the seeds used in the procedure are shown in Table 5.1.

A 6GB newspaper corpus was used to perform our experiments (TREC9 3.5GB, TREC2002 3GB). The search engine used was INQUERY [14]. The application of the

multi-level pattern learning algorithm on the above seeds led to the generation of more than 3500 patterns. Some of the multi-level patterns were:

1. X_JJ#NN|JJ#NN#NN|NN _CC Y_JJ#JJ#NN|JJ|NNS|NN|JJ#NNS
   |NN#NN|JJ#NN|JJ#NN#NN
   ...*Caldera* or *lava lake*...
2. X_NNP#NNP|NNP#NNP#NNP#NNP#NNP#CC
   #NNP|NNP|VBN|NN#NN|VBG#NN|NN ,_, _DT
   Y_NN#IN#NN|JJ#JJ#NN|JJ|NN|NN#IN
   #NNP|NN P#NNP|NN#NN|JJ#NN|JJ#NN#NN
   ...*leukemia*, the *cancer* ...
3. X_NNP#NNP|NN#NN|NN ,_, a_DT Y_NN|NN#NN
   |JJ#NN|JJ#NN#NN that_(WDT|IN)
   ...*tuberculosis*, a *disease* that...)
4. X_VBN|NN and_CC a_DT host_NN of_IN other_JJ Y_NNS|JJ#NNS
   (*platinum* and a host of other *metals*)
5. Y_JJ#NN#NN|NN#NN ,_, _RB called_VBN X_JJ#NN#NN|NN ._
   ...*potassium nitrate*, also called *salt peter*..

Note that we store different POS variations of the anchors X and Y. The reason is quite straightforward: we need to determine the boundary of the anchors X and Y and a reasonable way to delimit them is to use POS information.

5.3.2 Complexity

The algorithm treats each sentence independent of other sentences. Hence the algorithm is linear *O(n)*, where *n* is the number of sentences in the corpus.

5.3.3 Pattern Filtering

All the patterns produced by the multi-level pattern learning algorithm were generated from positive examples. From these patterns, we need to find the most important ones. This is a very critical step because we generally find that frequently occurring patterns have low precision (e.g., the pattern X Y[3]), while rarely occurring patterns have high precision. From the Information Extraction point of view neither of these patterns are very useful. We need to find patterns with relatively *high occurrence* and *high precision*. Fortunately, we can employ the principle of cross-entropy from Information theory, which precisely gives the required measure. We define the cross-entropy of each pattern as follows:

$$H(x|y) = P(x,y)logP(y|x) \tag{5.1}$$

where,

x =event that a Pattern P fired

y =event that required relation is correct.

Intuitively, $P(x,y)$ is a measure of frequency while $P(y|x)$ is the accuracy of each pattern. We calculate the cross-entropy of each pattern obtained from the pattern learning algorithm and select the top K patterns having highest measure of cross-entropy. To

[3]X followed by Y as in *Ford car*

implement this measure, we randomly sampled 10 examples for each of the top 600 patterns obtained from the multi-level pattern learning algorithm and tagged them as correct or incorrect. We select the top 15 patterns that have the highest cross-entropy. These patterns are:

1. X , or Y
2. X , (a|an) Y
3. X , Y
4. Y , or X
5. X , _DT Y _(WDT|IN)
6. X is a Y
7. X , _RB known as Y
8. X (Y)
9. Y such as X
10. X, _RB called Y
11. Y like X and
12. _NN , X and other Y
13. Y , including X ,
14. Y , such as X
15. Y , especially X

Some of these patterns are similar to the ones discovered by Hearst [34] (e.g., Pattern 9.) while others, such as the appositive ones (e.g., Pattern 5.), are similar to the ones chosen manually by Fleischman et al. [28].

5.3.4 Instance-Concept Extraction

Having acquired the extraction of patterns, we could simply use them to filter candidate answers in a QA system and pinpoint the exact answers. However, we wish to construct a large permanent repository of information for use in QA, MT, summarization

Table 5.2: Some random entries in the instance-concept table.

Frequency	Instance	Concept
21	Kadin	industry
8	Shrimp	eyes
8	Glasssteagall Act	depression-era law
6	month	Military spokesman
5	salsa	sauce
4	radiation therapy	cancer treatment

and other tasks. We therefore employ the pattern collection to perform large-scale text harvesting. We first POS tag a 15 GB newspaper using Brill's tagger. POS tagging is a relatively cheap process as compared to syntactic parsing and can easily scale to a large corpus size. Using the 15 lexical-POS patterns patterns described above, we extract instance-concept pairs corresponding to the variables X and Y along with their frequencies. These patterns are stored in a table as shown in Table 5.2. The extracted relations contain information about both proper nouns and common nouns. In addition to providing the instance-concept relation, (useful for answering questions such as What is X? and Who is X?), the table also contains Hyponym relations and Abbreviation expansion information. The following table gives examples of instances extracted by certain concepts:

colors:

6	pinks	colors
6	asphyxia	colors
5	hot pink	colors
5	cappuccino	colors
4	salmon	colors

4	neon	colors
3	plum	colors

desserts:

7	tiramisu	desserts
6	zucotto	desserts
6	carrot cake	desserts
3	bread pudding	desserts
2	custards	desserts
2	cheesecake	desserts
1	warm gingerbread	desserts

abbreviation expansions:

3177	GDP	gross domestic product
148	IP	intellectual property
38	IP	internet protocol
48	GPS	global positioning system
10	GPS	global positioning satellite
33	XML	extensible markup language
5	SMT	surface mount technology

airline companies:

22	American	airlines
12	United	airlines
11	Continental	airlines
10	UAL corp.	airlines
10	Air France	airlines
9	Delta	airlines
7	AMR corp.	airlines
6	United Airlines	airlines

actresses:

16	Hunt	actress
14	Hanover	actress
10	Antoinette Perry	actress
9	Winona Ryder	actress
9	Michelle Williams	actress
8	Susan Sarandon	actress
8	mother	actress
8	Gila Almagor	actress

Table 5.3: Sample Evaluation Tags.

Instance	Concept	Tag
ALD	atomic layer deposition	C
Vancomycin	other medicines	P
forfeiture	extreme remedy	P
tea	cultural events	I
camacho	former WBC	I

8 Dorothy Arnold actress

African countries:

26 South Africa African countries
17 Ethiopia African countries
10 Uganda African countries
8 Burkina Faso African countries
7 Tunisia African countries
7 Rwanda African countries
7 Angola African countries
6 Zimbabwe African countries
6 Nigeria African countries
6 Mozambique African countries
6 Mali African countries
5 Morocco African countries
4 Zaire African countries

5.3.5 Precision Evaluation

To measure precision of the extracted knowledge, we randomly sample 100 examples from instance-concept pairs and assigned them to one of the three categories: Correct (C), Partially correct (P), and Incorrect (I). Examples of some graded instance-concept pair are shown in Table 5.3.

Table 5.4: Precision Tagging.

	Correct	Partial
Precision	48%	12%

Based on the above criteria the precision results are shown in Table 5.4.

The errors arising from flase-positives could be classified into 3 categories:

1. **Syntactic errors**: Here the instance or the concept is incomplete. The error is due to the fact that the POS pattern was not good enough to exactly delimit the anchors, while a syntactic parser could have recovered the complete answer. E.g., we get, the concept *former WBC* instead of *former WBC secretary* for the instance *Camacho*.

2. **Semantic errors**: These errors arise because the instance-concept relation is too specific or too general with regard to the context of world knowledge. E.g., *tea* has a concept of *cultural events* (*tea* is too specific) while *Wash. Chevron chemical* has a concept called *unit* (too general). This is a subjective decision made by humans.

3. **Complete errors**: Most of the errors belong to this category. Here the answers are completely wrong because the instance and concept are totally unrelated.

5.4 Improving Precision of Pattern-based system: Using Machine Learning Filter

A system with only about 48% precision as described above is not very useful. In this section, we describe work on improving the precision of the instance-concept pair obtained from the *is-a* relation.

We first begin by using a Noun Phrase Chunker to improve syntactic errors described in the previous section. For each sentence from which the *is-a* relation is extracted we perform Noun Phrase Chunking using CASS [8]. Using simple heuristics, we only extract those instance-concept pairs such that they form part of a Noun Phrase.

In the next step, each extracted noun phrase is passed through a Machine Learned filter which is a model to predict the correctness of the given *is-a* relation. In the following section subsections we describe the model in detail.

5.4.1 Model

We model a Maximum entropy model to predict the correctness of a given *is-a* relation using the following equation:

$$p(c|a,b) = \frac{exp(\sum_{m=1}^{M} \lambda_{mc} f_m(a,b),c)}{\sum_{c'} exp(\sum_{m=1}^{M} \lambda_{mc'} f_m(a,b),c')} \tag{5.2}$$

where,

a is the concept part of *is-a* relation.

b is the instance part of *is-a* relation.

$f_m,\ m = \{1, 2..., M\}$ are M feature functions.

$\lambda_m,\ m = \{1, 2..., M\}$ are the corresponding M model parameters.

$c', c\ \epsilon\{false, true\}$ the decisions to be made for every instance-concept pair.

The features used to model Equation 5.2 can be classified into the following four main categories:

1. **Capitalization features**: These features determine if certain nouns of the instance-concept begin with a capitalized letter or not. Some features are used to check if the entire instance is capitalized.

2. **Pattern-based feature**: These features dtermine what kind of pattern triggered this particular instance-concept pair.

3. **Lexicalized features**: These types of features check to see if the head noun of the concept contains suffixes such as *er, or, ing, ist, man, ary,* and *ant*. Honorific mentions such *Mr., Dr.,* and *Ms.* are also checked.

4. **Co-occurrence based features**: In this category we calculate how many times the instance-concept pair was independently observed in the corpus.

Table 5.5: Precision and Recall Results on True relations using Machine Learning Filter.

Sample Size	Precision	Recall
500	78%	84%

5.4.2 Training

We randomly sample 1000 examples from the extracted list of *is-a* relations, and tell a human to tag them as *correct* or *incorrect*. We use 500 examples from the above set for training and 500 examples for testing and development.

We use Gradient Iterative Scaling algorithm (GIS) [23] to train our Maximum Entropy model implemented by YASMET [4].

5.4.3 Results

The Results of the output of the Machine Learned filter are shown in Table 5.5. We caution the reader that these are only the precision and recall results for the output of the Machine Learning filter. They do not measure the actual precision wherein we fuse duplicate instance-concept pairs into one output. Similarly they do not measure the actual recall of the system: "How many pairs of correct instance-concept pairs does the system contain from the entire instance-concept pair set in English language?". In the

[4]YASMET – Yet Another Small Maximum Entropy Toolkit – `http://www.isi.edu/ och/YASMET/`

next section, we apply the algorithm thus developed (from newspaper text) to a noisy corpus constructed from the Web in the next section.

5.5 Clustering-based system: Instance-concept extraction

5.5.1 Introduction

The clustering technique gives a list of nouns that are semantically equivalent. However, these classes (clusters) have no labels assigned to them. Table 5.6 lists the top 15 elements of 3 different clusters. To construct *is-a* hierarchies, we need a label assigned to each cluster. For example, in Table 5.6, the cluster containing the elements {*Poland, Hungary, Romania,...*} needs the label *country*. Thus from this label we can infer the relation: *Poland* is a *country*, *Hungary* is a *country*, *Romania* is a *country* etc. By using this approach, we can make use of the rich co-occurrence statistics to assign *is-a* labels. In the next section, we proceed to describe an algorithm to extract *is-a* relations using the clustering technique.

5.5.2 Algorithm

We use a modified technique described in [56] to automatically label semantic classes.

1. We start out with a huge natural language corpus. We identify all nouns by using a noun phrase identifier. For each noun phrase, we identify the context words

Table 5.6: Top 15 Similarity Lists

Cluster 1	Cluster 2	Cluster 3
Poland	tomato	Integrity
Hungary	garlic	Honesty
Romania	pepper	Professionalism
Bulgaria	onion	Fairness
Macedonia	red pepper	Dignity
Albania	parsley	Objectivity
Slovakia	carrot	Truthfulness
Lithuania	mushroom	Competence
Estonia	potato	Trustworthiness
Latvia	spinach	Impartiality
Slovenia	celery	Purity
Czechoslovakia	basil	commitment
Serbia	cilantro	consistency
Austria	bell pepper	teamwork
Rumania	green onion	accountability

surrounding it. Our context window length is restricted to two words to the left and right of each noun. We use the context words as features of the noun.

2. Having collected all nouns and their features, we now proceed to construct feature vectors for nouns. We first construct a frequency count vector $C(e) = (c_{e1}, c_{e2}, ..., c_{ek})$, where k is the total number of features and c_{ef} is the number of times feature f occurring in word e. Here, c_{ef} is the number of times word e occurred in context f. We then construct a mutual information vector $MI(e) = (mi_{e1}, mi_{e2}, ..., mi_{ek})$ for each word e, where mi_{ef} is the point-wise mutual information between word e and feature f, which is defined as:

$$mi_{ef} = log \frac{\frac{c_{ef}}{N}}{\sum_{i=1}^{n} \frac{c_{if}}{N} \times \sum_{j=1}^{k} \frac{c_{ej}}{N}} \tag{5.3}$$

where n is the number of words, and $N = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}$ is the total frequency of all features of all words. Mutual information is commonly used to measure the association strength between two words [20]. A well-known problem is that mutual information is biased toward infrequent elements/features. We therefore multiply mi_{ef} with the following discounting factor:

$$mi_{ef} = \frac{c_{ef}}{c_{ef} + 1} \times \frac{min(\sum_{i=1}^{n} c_{ei}, \sum_{j=1}^{k} c_{jf})}{min(\sum_{i=1}^{n} c_{ei}, \sum_{j=1}^{k} c_{jf}) + 1} \tag{5.4}$$

3. We then construct a similarity matrix between every pair of vector e_i and e_j. Calculating the full similarity matrix would take time complexity $O(n^2k)$, where n is the total number of vectors with a maximum of k features. With large amounts of data, say n in the order of millions, using an $O(n^2k)$ algorithm would be infeasible. However, we use the randomized algorithm described in Chapter 4 to linearize this step and reduce the complexity to $O(nk)$.

4. We then perform clustering over this data set using CBC (Clustering By Committee Algorithm) [55]. We initially construct a set of committees with each committee representing a semantic class. A committee is a set of representative

elements that unambiguously describes the members of the respective semantic class. Once a set of committees is determined, all the vectors in the collection are assigned to one or more semantic classes. This is a soft clustering technique, which means that a vector could be assigned to more than one class. This algorithm is to the best of our knowledge the state of the art in word clustering.

5. In the next step, we proceed to assign labels to each semantic class. For each semantic class we choose all elements that are above a particular threshold γ. We then apply the pattern technique described in Section 5.3 to assign labels to each element individually. Some of the elements in a semantic class may not have any label assigned to them, since the pattern-based system does not guarantee a label for every element. We then choose the top N labels that are shared by most of the elements and output their names as the labels of the semantic class. The value of γ was chosen experimentally to be 0.15. Using this technique, the top three labels assigned to the cluster containing the elements {*Poland, Hungary, Romania,...*} are *country, nation, state*. More examples are given in Table 5.7.

6. We then construct an *is-a* list by saving the top 3 labels from each cluster. Thus we can rewrite entries in Table 5.7 as *Poland* is a *country*, *Poland is a nation*, *Poland* is a *state*, *tomato* is a *food*, *tomato* is a *product*, *tomato* is a *spice*, *Hungary* is a *country* etc.

Table 5.7: Top 15 Similarity Lists with Assigned Labels

country, nation, state	*food, product, spice*	*issue, value, concept*
Poland	tomato	Integrity
Hungary	garlic	Honesty
Romania	pepper	Professionalism
Bulgaria	onion	Fairness
Macedonia	red pepper	Dignity
Albania	parsley	Objectivity
Slovakia	carrot	Truthfulness
Lithuania	mushroom	Competence
Estonia	potato	Trustworthiness
Latvia	spinach	Impartiality
Slovenia	celery	Purity
Czechoslovakia	basil	commitment
Serbia	cilantro	consistency
Austria	bell pepper	teamwork
Rumania	green onion	accountability

5.6 Hybrid system: Instance-concept Extraction

5.6.1 Introduction

In the previous two sections we demonstrated two ways of getting *is-a* lists:

1. Pattern-based system

2. Clustering-based system.

The pattern-based system mostly made use of rich local features to assign *is-a* relations. These patterns were learned automatically by using seed examples. The clustering system made a rich use of co-occurrence (global) features to assign *is-a* labels. To scale

up the process for the clustering system we made use of randomized algorithms. The Pattern-based system has a precision of 66% (116GB corpus, strict criteria) while the Clustering-based system has a precision of 62% (116GB corpus, strict criteria)[5]. To improve precision, we can combine the two lists to form a hybrid system. This hybrid system can make use of rich local features and rich co-occurrence features simultaneously in assigning *is-a* labels. algorithm. The next subsection describes this algorithm.

5.6.2 Algorithm

We start out with two lists obtained by using the Hybrid-based system and the Pattern-based system. Our goal is to assign the top three labels (concept) for each instance. For lack of sufficient training data, we use a simple Clustering-based system to combine the two lists. After many experimental evaluations, we settled on the following rule based algorithm:

1. Output all elements that are common to both lists.

2. Output elements from the Pattern-based list if frequency of occurrence is greater than β.

3. Output elements from the Clustering-based list if the similarity score of the element (instance) with the cluster centroid is greater than or equal to γ.

[5]Detailed evaluation results are available in the next Chapter

We experimentally choose $\beta = 2$ and $\gamma = 0.10$. The reader may find the above rule based algorithm very puzzling. The following observations were made during the experimentation process that led to the above algorithmic design:

1. Frequency of occurrence was a much better indicator of correctness than the confidence score given out by the ML system. Web data is very noisy, hence relations that were observed less frequently were more likely to be incorrect than more frequently occurring ones.

2. Relations that are shared by both the Pattern-based system and the rule based system are likely to be more accurate.

3. For the clustering system, a similarity score less than 0.10 with the cluster centroid had a higher error rate in assigning *is-a* labels.

4. While combining the lists, we also had to make sure that we did not lose recall at the cost of precision.

5. Clustering system is more resilient to noise than the Pattern-based system. This is not surprising since the Pattern-based system mostly relies on local features, while the clustering system relies on co-occurrence features.

5.7 Conclusion

In this chapter, we proposed two basic algorithms (Clustering-based and Pattern-based) for high speed extraction of knowledge from text. We also proposed a third system: Hybrid, that combines the Clustering-based and the Pattern-based system. In the next system, we evaluate our *is-a* tables.

Chapter 6

Evaluations

6.1 Introduction

In this chapter, we evaluate the the huge *is-a* lists that we extract from text described in the previous chapter. Evaluations in knowledge harvesting are generally performed under two dimensions:

1. How accurate is the extracted knowledge (Precision)?

2. What is the coverage of the extracted knowledge (Recall)?

Both recall and precision evaluations are discussed in the next few sections.

6.2 Precision

All the three systems: The Precision-based system (with the Machine Learning Filter), Clustering-based system and the Hybrid-based are each individually evaluated for precision in the next few subsections.

6.2.1 Pattern-based System

We perform our experiments on a 70 million webpage corpus (116 GB) downloaded from the Internet. To understand the learning curve, we run the Pattern-based algorithm with filter to different subsets corpora: 1.5 GB, 17GB, 57 GB and 116 GB. We extract *is-a* relations from each of these corpora and evaluate their precisions. For each of the above system (constructed from corpora of different sizes) we extract 50 random instances (from the *is-a* list). For each relation we extract the at the top 3 frequently occurring concept. We present these examples (50 randomly generated instances along with their system generated from 4 different system) to a human judge. Sample outputs generated by the system are displayed in Table 6.1. For each relationship we asked the judge to assign a score of *correct*, *partially correct*, or *incorrect*. We evaluate on two different criteria: strict (only correct answers are considered), and lenient (both correct and partially correct answers are considered). We compute the average precision and MRR score (Mean Rank reciprocal) of all the systems. For each word, a system receives

Table 6.1: Sample labels assigned by the Pattern-based system

Random instances	Concepts
Evan Treborn	young man/college student
Heartgard	medicine/pet medicine/preventive
Goddess Saraswati	Hindu deity

Table 6.2: Pattern-based system: Precison Evaluation

	Strict			Lenient		
Corpus Size	Top 1	Top 3	MRR	Top 1	Top 3	MRR
1.5 GB	64%	68%	66%	72%	74%	73%
17 GB	62%	70%	65.3%	68%	76%	71.3%
57 GB	50%	54%	51.7%	62%	66%	63.7%
116 GB	66%	70%	67.67%	74%	78%	75.7%

an MRR score of $1/M$, where M is the rank of the first name judged correct. Results are displayed in Table 6.2.

Results show that the Pattern-based system has an accuracy in the range of 50-66% (strict) and 62-74% (lenient).These results are significantly lower than then the 78% accuracy reported for newspaper text. The precision of each system also improves with more amount of data.

6.2.2 Clustering-based System

As in Subsection 6.2.1, we perform our precision experiments on a 70 million webpage corpus (116GB) downloaded from the Internet for the clustering-based system. We evaluate them by strict and lenient criteria. Results are displayed in Table 6.4. Results

Table 6.3: Sample Labels assigned by the Pattern System

Random instances	Concepts
Six Feet Under	show/series/TV show
OH CANADA	song/favorite/favourite
Gallitzin	town/place/small town

Table 6.4: Clustering-based system: Precison Evaluation

	Strict			Lenient		
Corpus Size	Top 1	Top 3	MRR	Top 1	Top 3	MRR
1.5 GB	60%	72%	65.3%	62%	72%	66.3%
17 GB	60%	76%	67.7%	64%	78%	70.6%
57 GB	67.4%	78.3%	71.7%	69.6%	80.4%	73.9%
116 GB	62%	78%	69.7%	64%	78%	70.7%

show that the Pattern-based system has an accuracy in the range of 60-67% (strict) and 62-72% (lenient). These results are better than those of the Pattern-based system.

6.2.3 Hybrid System: Precision Evaluation

We evaluate our Hybrid system similar to evaluations in Subsections 6.2.1 and 6.2.2, Results are displayed in Table 6.6. Results show that the Hybrid system has an accuracy in the range of 62-72% (strict) and 68-84% (lenient). This is an improvement over the precision of both the Pattern-based and Clustering-based system. There is also a big increase in the Top 3 precision accuracy (86% strict, 90% lenient)

Table 6.5: Sample labels assigned by the Hybrid system

Random instances	Concepts
Dogpile	engine/site/meta
Endocarditis	infection/disease/complication
Heartbreakers	comedy/movie

Table 6.6: Hybrid system: Precison Evaluation

	Strict			Lenient		
Corpus Size	Top 1	Top 3	MRR	Top 1	Top 3	MRR
1.5 GB	62%	66%	63.7%	68%	72%	69.6%
17 GB	72%	74%	73%	76%	80%	77.6%
57 GB	68%	84%	75.3%	72%	84%	77.6%
116 GB	70%	86%	77.3%	84%	90%	87%

6.2.4 Large Scale Precision Evaluation

All the above evaluations were performed by only one human and each system was evaluated against 50 randomly selected outputs. To have stronger bounds, one may need 3 humans evaluating against 1000 randomly selected outputs. Hence our precision evaluation could be considered as weak. Our experience, in evaluating this system indicates that more than 50% of the system outputs to be evaluated are rarely heard by an average human. For example, one of the system output to be evaluated for system precision was:

Instance: *Evan Treborn* Concept: *college student*

For this to be judged appropriately, the human evaluator had to look up a search engine and investigate a few pages of output before coming to the correct conclusion. In some

cases, the result proved to be inconclusive to the human assessor in which case we had to drop the example altogether from our evaluation.

We now try to come up with an economic cost to evaluate the full system. Assume we have 3 humans, with 1000 random samples for each system. Let us assume it takes 1 minute on average for a human judge to make a decision. We have 3 systems (Pattern-based, Clustering-based and Hybrid) each evaluated as a function of four different corpora sizes (1.5 GB, 17GB, 57 GB and 116 GB). Hence, the total number of man.minutes (or man.hours) taken for this evaluation would be:

$(1000\ samples) * (3\ humans) * (3\ systems) * (4\ corpora\ sizes) * (1\ minute)$

$= 36,000\ man.minutes$

$= 600\ man.hours$

This is not an easy task (based on our experience) and hence for this evaluation to be done efficiently, we estimate that the human judges should be paid $20/hour. Hence the amount of money required to perform a sound precision evaluation would come down to $600 * 20 = \$12,000$. This is an astonishingly huge cost. Also, if want our results to be consistent we must choose the same 3 humans to perform the task. Thus each human would take an estimated 200 hours, or 5 weeks[1] to complete the task. The other big problem with this kind of evaluation is that it is not readily reusable. In case, we decide to tune some parameters to change the system output, we may again need to

[1]This is by assuming a 40 hour work week.

Table 6.7: Approximate corpus size and instance-concept pairs extracted

			Pattern		Clustering		Hybrid	
Corpus	# words	# sent	# rels	Pre.	# rels	Pre.	# rels	Pre.
1.5 GB	309M	12.5M	13,754	64%	14,434	60%	9,916	62%
17 GB	3.4B	141.66M	88,207	62%	158,556	60%	128,474	72%
57 GB	11.4B	475M	230,007	50%	470,003	67%	413,437	68%
116 GB	26B	1B	396,790	66%	1,047,942	62%	1,034,932	70%

perform $12,000 worth of human evaluations. Trying to develop a standard reusable test framework for evaluating precisions of such system is a big research challenge.

6.3 Learning curve

For the learning curve, we study the Knowledge harvesting task as a function of corpus size. For this purpose the data set is divided into different sets: 1.5 GB, 17GB, 57 GB and 116 GB. Table 6.7 gives interesting facts about the various corpora and the associated extracted relations and their precision for the three different systems: Pattern based, Clustering-based and the Hybrid system.

The graph in Figure 6.1 shows the relation between the number of extracted instance-concept pair. The number of relations extracted for all three systems looks linear in the size of the corpus. Thus, scaling to even more data will likely produce more instance-concept pairs. The Pattern-based system produces significantly fewer relations than the other two systems.

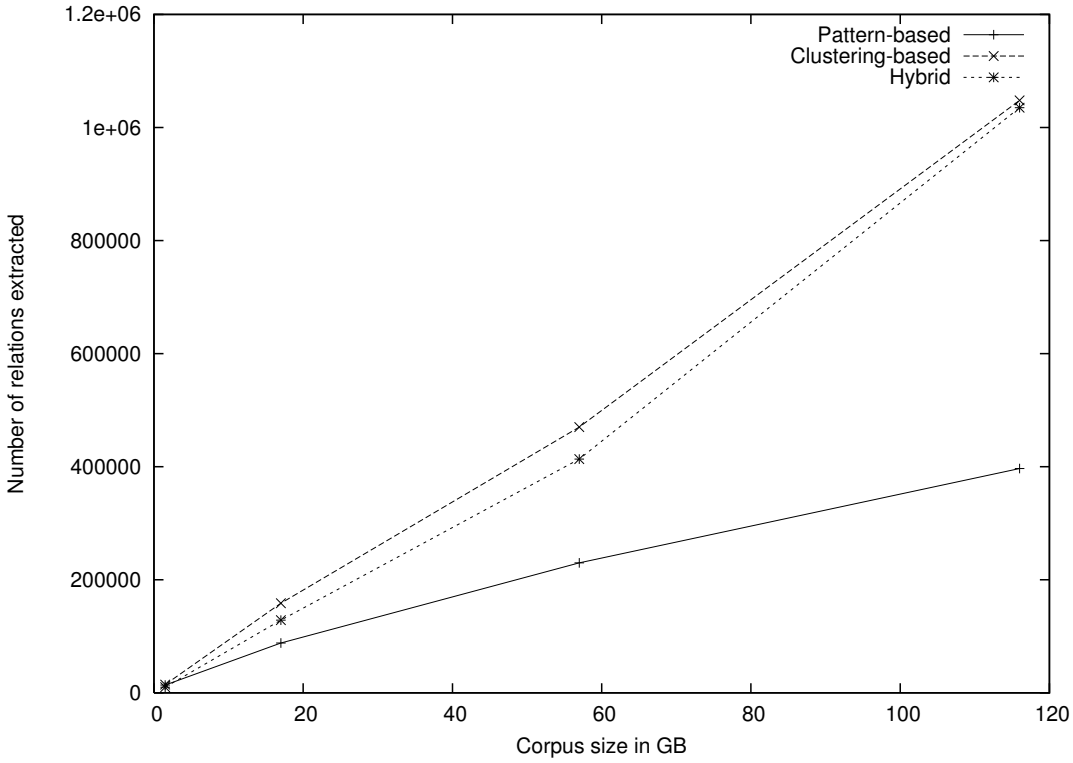


Figure 6.1: Graph showing the number of unique instance-concept pair extracted as a function of corpus Size.

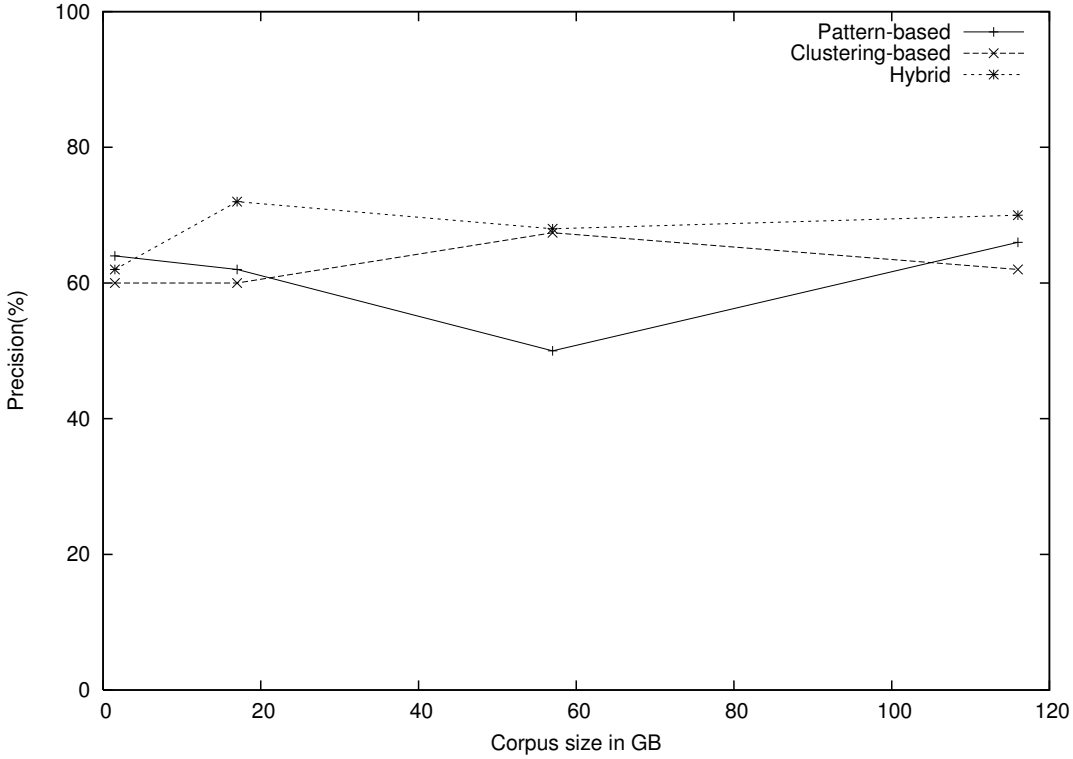


Figure 6.2: Graph showing precision of extracted relations as a function of a the corpus Size.

Figure 6.2 shows the relationship between precision of the extracted relations and the corpus size. We suspect the precision curve to be a straight line with no slope. Thus precision more or less remains constant for each system independently of the amount of data.

Thus, scaling the entire process to a more data has a lot of promise. We expect to see more relations because proper nouns are potentially an open set and we do learn a lot of proper nouns. Similarly, each instance can have more than one concept. E.g., the instance *South Africa* has the concepts *country* and *African country* which we count separately.

6.4 Recall

Evaluating recall (coverage) for such knowledge harvesting system is generally considered a very difficult task. The difficulty stems from the basic question: How does one define the breadth (covereage) of knowledge? Answering this question is possibly a never ending debate. One could argue that encylopedia knowledge is the holy grail that computers should aim to aquire. Others would argue we should measure against avearge human knowledge. In order to steer away from such arguments, we propose application based evaluations. These evaluations have a definite evaluation criteria. Also these applications provide validation to prove that the work performed in this thesis has some useful basis. We use four application based evaluation:

1. Semantic Retrieval for Question Answering (QA) Systems,

2. Categorizing items in *In the News* section of *Google News*

3. Answering Ask Jeeves Sample Logs

4. Answering TREC QA Definitional questions.

These evaluations are discussed in the next subsections.

6.4.1 Semantic Retrieval for Question Answering

A typical QA system consists of at least Information Retrieval/Passage Retrieval and Answer Selection modules. The reason for this typical pipeline architecture is that the Answer Selection module is much slower and hence should ideally be given only promising passages; viz., those that have a high likelihood of the correct answer being present. To improve Passage Retrieval, a knowledge base such as the concept-instance(*is-a*) relations could be exploited effectively by performing so called semantic indexing. The idea of semantic indexing is to index the IR collection with the expected semantic answer-type for a given question. For example, given the question *What color is the cranberry juice?* , the likelihood of the correct answer being present in the passage is greatly increased if we know the set of all possible *colors* and pre-index them in the document collection. Having done this, we need to search for only *colors* in the text.

Our Information Retrieval stage for Question Answering contains two parts: a Document Retrieval component and a Passage Retrieval component. The goal of the Information Retrieval component is the following: Given a question Q and a large set of documents D_1, D_2..., D_n (in the order of millions), the system has to retrieve a set of N_1 documents such that the possibility of finding the correct answer to the given question among them is maximized.

The goal of the Passage Retrieval component is the following: Given a question Q, a set of N_1 documents (obtained from Document Retrieval), the system has to retrieve a set of N_2 passages such that the possibility of finding the correct answer to the given question in the passage is maximized.

The definition of what constitutes a passage varies from system to system. Some systems may employ Text Tiling [35] to segment a document into several passages, with each passage having a common topic, and segmentation occurring at that point of text where relatively major changes in topic are found. Other systems may choose to treat each paragraph as a separate passage or treat each set of k adjacent sentences as a passage. For our experimental setup, we choose to perform Passage Retrieval based on sentence segmentation, where each passage consists of exactly 1 sentence. The sentence segmentation is performed by MXTERMINATOR [62]. The Information Retrieval engine used for the above purpose is a general purpose one called INQUERY [14].

Table 6.8: Examples of Questions used and the corresponding semantic-answer type

Question	Semantic answer type
What city is Disneyland in?	city
What is the play "West Side Story" based on?	play
What color is the top stripe on the U.S. flag?	color
What document did Button Gwinnett sign on the upper left hand side?	document

We use the ACQUAINT 2002 collection as the source of ¡documents. ACQUAINT 2002 is a 3 GB newspaper corpus. To perform this evaluation we extract 179 questions from the TREC2003 collection. These questions are of the form *What X is Y?* or *Which X is Y?* (e.g., *What city is Disneyland in?* or *What country did Catherine the Great rule?*), where the required Semantic Answer type is *X* (e.g., *city* or *country* respectively). For each Semantic Answer type corresponding to a given question we index the entire IR collection (TREC 2002 corpus) based on the information from our instance-concept knowledge base. Thus if the required semantic answer type is a country we would index the entire IR collection with all countries {*United States , Britain , Russia , etc.*} from the list of instance-concept relations. Some examples of the extracted semantic type are given in Table 6.8.

We perform passage retrieval with and without the use of the semantic answer type. Experiments are performed using the three different systems: Pattern-based, Clustering-based and Hybrid, with each system constructed from corpora of different sizes (1.5 GB, 17GB, 57GB and 116GB). We evaluate the performance with respect

Table 6.9: Semantic Retrieval For QA (Total sentences containing the correct answer). Baseline: 1310.

Corpus Size	Clustering	Pattern	Hybrid
1.5 GB	1515	1634	1575
17 GB	1555	1705	1674
57 GB	1576	1716	1727
116 GB	1527	1709	1746

to two measures. The first measure is to find the total number of occurrences of all answers in the entire collection. The second measure is to determine if the top ranking passage (before answer selection) has the required correct answer. In our experiments we restrict the size of each passage to 1 sentence.

The results of the passage retrieval task are displayed in Table 6.9 and 6.10. In Table 6.9, the baseline system without semantic retrieval has 1310 answers in the total collection extracted from all 179 questions. With the use of semantic indexing we see big improvements in system outputs. The Hybrid system outperforms the Clustering-based and Pattern-based system in this case. Table 6.10 displays the answers obtained for the first sentence. The baseline for this case is 36/179. We see 13-30% improvements for the Pattern-based system and 3-36% improvement for the hybrid system. However, the improvement for the Clustering-based system is not that big.

Table 6.10: Semantic Retrieval For QA Top1: Baseline 36/179

Corpus Size	Clustering	Pattern	Hybrid
1.5 GB	44/179	41/179	39/179
17 GB	39/179	46/179	43/179
57 GB	36/179	46/179	45/179
116 GB	39/179	47/179	48/179

6.4.2 Google News

Google maintains a news website[2] where articles from various news sources are presented. The main feature of Google News is the automatic categorization of various news articles into different clusters. Each cluster represents one news topic. There is also another less known feature of Google News: Automatic identification of important entities that appear in the news. These are generally found in the *In The News* section of the web page. A Google News corpus is constructed by downloading its main web page everyday[3] at a particular time for a period of a year. We extract all entities from *In the News Section*. From this list, we randomly selected 50 entities (instances). For each such entity we look up their corresponding top three concepts from our instance-concept table for the three systems (Clustering-based, Pattern-based and Hybrid) built as a function of corpus size (1.5GB, 17GB, 57GB and 116GB). Some examples are shown in Table 6.11. We also compare our results to that of WordNet. These instance-concept pairs are then manually evaluated by a human. Results are judged according to

[2] `http://news.google.com`
[3] Corpus collected by Hal Daumé III

Table 6.11: Sample Labels assigned from Google News

Google News Entity	Tag assigned
Advance Placement	exam
Anwar Ibrahim	politician
Bernard Ebbers	executive

Table 6.12: Google News Evaluation: Top1 Precision

	Strict			Lenient		
WordNet	32%			32%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	8%	12%	4%	8%	12%	4%
17 GB	42%	40%	38%	46%	46%	38%
57 GB	62%	50%	64%	74%	56%	66%
116 GB	64%	54%	66%	66%	66%	72%

three criteria: Top 1 Precision, Top 3 Precision and MRR. These Results are shown in Table 6.12, 6.13 and 6.14. Systems are evaluated by 2 criteria: Strict and Lenient. In the strict criteria, credit is given only for fully correct answers. In the lenient criteria, credit is given even for partially correct answers. The Hybrid system achieves superior performance as compared to the clustering and the Pattern-based system. All three systems for the 116GB corpus perform significantly better than the WordNet system. Performance improves with more data.

Table 6.13: Google News Evaluation: Top3 Precision

	Strict			Lenient		
WordNet	32%			32%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	8%	12%	4%	8%	12%	4%
17 GB	52%	54%	44%	52%	60%	44%
57 GB	82%	68%	78%	84%	74%	82%
116 GB	80%	72%	86%	82%	82%	86%

Table 6.14: Google News Evaluation: MRR

	Strict			Lenient		
WordNet	32%			32%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	8%	12%	4%	8%	12%	4%
17 GB	46.32%	23.16	41%	48.32%	48.32%	41%
57 GB	72%	28.33	70.66%	74%	63.66%	73.66%
116 GB	71.33%	62.32%	75%	73.66%	73.66%	78.66%

6.4.3 Ask Jeeves

Ask Jeeves is a search engine that specializes in answering queries formulated using Natural Language questions. The technology used by Ask Jeeves to perform its Information Retrieval is not known. However, for the purpose of this evaluation we only examine Ask Jeeves query logs. From these queries, we extract questions of the type: *What is X?* and *Who is X?*. Fifty random questions are selected from each of these two types. For each question, we extract its respective instance, look up their corresponding concepts from our instance-concept table, and present the top 3 corresponding concept as the answer. We use all the three systems: Clustering-based, Pattern-based and Hybrid, each built as a function of corpus size (1.5GB, 17GB, 57GB, and 116GB). Sample outputs are given in Tables 6.15 & 6.19. These instance-concept pairs are then manually evaluated by a human. We also compare the results on WordNet[53]. Results are displayed in Tables 6.16 & 6.20, Tables 6.17 & 6.21, and Tables 6.18 & 6.22 for Top 1 Precision, Top3 precision and MRR respectively for the 2 different types of questions (*Who,What*). As before, results are evaluated by 2 criteria: Strict and Lenient. Looking at the results, we observe that WordNet performs better for *What is X?* questions as compared to *Who is X?* questions. This is because the sample containing *What is X?* questions has more common nouns which are in WordNet. The three systems also seem to have performance in the 40-45% (Top 1 precision, strict criteria, 116GB); (see Table 6.20) for *Who is X?* questions, which is lower than for Google News. One possible

Table 6.15: Sample answers assigned from Ask Jeeves: *What is X?*

What is X?	Answer
Coral reef	habitat
neutrino	subatomic particle
tangelo	citrus fruit

Table 6.16: Ask Jeeves *What is X?* Evaluation: Top1 Precision

	Strict			Lenient		
WordNet	54.16%			54.16%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	20.83%	25%	18.75%	20.83%	12%	18.75%
17 GB	45.83%	52.08%	45.83%	45.83%	25%	50%
57 GB	56.25%	54.16%	45.83%	56.25%	56.25%	52.08%
116 GB	58.33%	56.25%	58.33%	58.33%	58.33%	66.67%

explanation for this decrease in performance, is that our web corpus was constructed in June-September 2004, while the Ask Jeeves log was collected in 2002. However, the Google News corpus was collected only in 2004.

Table 6.17: Ask Jeeves *What is X?* Evaluation: Top3 Precision

	Strict			Lenient		
WordNet	54.16%			54.16%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	25%	39.58%	18.75%	25%	39.58%	18.75%
17 GB	52.08%	54.16%	52.08%	52.08%	54.16%	52.08%
57 GB	70.83%	66.67%	60.41%	72.91%	68.75%	62.5%
116 GB	64.58%	77.08%	70.83%	64.58%	77.08%	72.91%

Table 6.18: Ask Jeeves *What is X?* Evaluation: MRR

	Strict			Lenient		
WordNet	54.16%			54.16%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	22.91%	32.29%	18.75%	22.91%	32.29%	18.75%
17 GB	48.95%	53.12%	48.26%	48.95%	53.12%	50.69%
57 GB	62.83%	60.41%	52.23%	63.88%	62.5%	56.94%
116 GB	61.45%	65.96%	64.23%	61.45%	67%	69.44%

Table 6.19: Sample answers assigned from Ask Jeeves: *Who is X?*

Who is X?	Answer
CRF	Corticotrophin releasing factor
Pete Sampras	tennis player
Katie Couric	anchor

Table 6.20: Ask Jeeves *Who is X?* Evaluation: Top1 Precision

	Strict			Lenient		
WordNet	4.25%			4.25%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	2.12%	6.38%	4.25%	2.12%	8.51%	4.25%
17 GB	14.89%	23.40%	10.63%	17.02%	23.40%	10.63%
57 GB	36.17%	31.91%	27.65%	40.42%	34.04%	29.78%
116 GB	44.68%	40.42%	44.68%	44.68%	42.55%	46.80%

Table 6.21: Ask Jeeves *Who is X?* Evaluation: Top3 Precision

	Strict			Lenient		
WordNet	4.25%			4.25%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	2.12%	6.38%	4.25%	2.12%	8.51%	4.25%
17 GB	19.14%	29.78%	17.02%	21.27%	31.91%	17.02%
57 GB	44.68%	42.55%	36.17%	48.93%	46.80%	36.17%
116 GB	61.70%	51.06%	57.44%	63.82%	53.19%	57.44%

Table 6.22: Ask Jeeves *Who is X?* Evaluation: MRR

	Strict			Lenient		
WordNet	4.25%			4.25%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	2.12%	6.38%	4.25%	2.12%	8.51%	4.25%
17 GB	17.02%	26.59%	13.47%	19.14%	27.65%	13.47%
57 GB	40.42%	36.51%	31.56%	44.68%	39.36%	32.62%
116 GB	51.40%	45.74%	50.71%	52.46%	47.87%	51.77%

6.4.4 QA Simple Definition Questions

For this evaluation, we select the 50 questions from the TREC2003 [71] and 62 questions from the TREC2004 [72] question set. These questions are of the form *Who is X?* and *What is X?*. For each question (e.g., *Who is Niels Bohr?*, *What is feng shui?*) we extract its respective instance (e.g., *Neils Bohr* and *feng shui*), look up their corresponding concepts from our instance-concept table, and present the corresponding concept as the answer. We extract at most the top 3 answers provided by each system. We manually evaluate the three systems (Clustering-based, Pattern-based and Hybrid) built as a function of corpus size (1.5GB, 17GB, 57GB and 116GB) by strict and lenient criteria. Sample outputs from the system are shown in Table 6.23 and 6.27. We also compare our results to those obtained using WordNet [53].

This evaluation is actually different from the ones performed by the TREC organizers for definition questions. However, we believe that by remaining consistent across

Table 6.23: Sample answers assigned from TREC 2003 Definitional questions: *Who/What is X?*

Who is X?	Answer
Aaron Copland	composer
Aga Khan	prince
vagus nerve	tissue

Table 6.24: TREC 2003 Definitional Question Evaluation: Top1 Precision

	Strict			Lenient		
WordNet	36.17%			36.17%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	2.04%	16.3%	4.25%	2.04%	20.04%	4.25%
17 GB	32.65%	38.77%	36.17%	34.69%	38.77%	40.42%
57 GB	51.02%	44.89%	53.19%	53.06%	44.89%	59.57%
116 GB	57.14%	55.10 %	63.82%	59.18%	59.18%	72.32%

all systems during the process, the above evaluations give an indication of the coverage of the knowledge base. We measure the performance on the top 1 and the top 3 answers returned by each system as shown in Tables 6.24,6.28, and 6.25,6.29 respectively. MRR scores are reported in Tables 6.26 and 6.30. The Hybrid system mostly outperform the Clustering-based and Pattern-based system. All these three systems significantly outperforms the WordNet system with the 116GB corpus.

Table 6.25: TREC 2003 Definitional Question Evaluation: Top3 Precision

	Strict			Lenient		
WordNet	36.17%			36.17%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	2.04%	16.32%	4.25%	2.04%	20.40%	4.25%
17 GB	40.81%	53.06%	36.17%	40.81%	55.10%	40.42%
57 GB	57.14%	65.30%	61.70%	59.18%	65.30%	61.70%
116 GB	65.30%	75.51%	70.21%	67.73%	75.51%	74.46%

Table 6.26: TREC2003 Definitional Question Evaluation: MRR

	Strict			Lenient		
WordNet	36.17%			36.17%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	2.04%	16.32%	4.25%	2.04%	20.40%	4.25%
17 GB	36.05%	45.23%	36.17%	37.41%	46.25%	40.42%
57 GB	54.08%	54.42%	57.44%	56.12%	54.76%	60.63%
116 GB	60.54%	64.28%	66.67%	62.58%	66.66%	73.40%

Table 6.27: Sample answers assigned from TREC 2004 Definitional questions: *Who/What is X?*

Who/What is X?	Answer
boll weevil	pest
Johnny Appleseed	film
Wicca	religion

Table 6.28: TREC 2004 Definitional Question Evaluation: Top1 Precision

	Strict			Lenient		
WordNet	28.07%			28.07%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	5.26%	14.03%	3.5%	5.26%	14.03%	3.50%
17 GB	26.31%	35.08%	26.31%	28.07%	36.84%	28.07%
57 GB	45.61%	47.36%	42.10%	49.12%	47.36%	43.85%
116 GB	50.87%	49.12 %	54.38%	52.63%	52.63%	57.89%

Table 6.29: TREC 2004 Definitional Question Evaluation: Top3 Precision

	Strict			Lenient		
WordNet	28.07%			28.07%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	5.26%	14.03%	3.58%	5.26%	14.03%	3.50%
17 GB	31.57%	49.12%	29.82%	36.84%	49.12%	31.57%
57 GB	54.38%	59.64%	52.63%	57.89%	59.64%	52.63%
116 GB	61.40%	63.15 %	64.91%	61.40%	64.91%	64.91%

Table 6.30: TREC 2004 Definitional Question Evaluation: MRR

	Strict			Lenient		
WordNet	28.07%			28.07%		
Corpus Size	Clustering	Pattern	Hybrid	Clustering	Pattern	Hybrid
1.5 GB	5.26%	14.03%	3.5%	5.26%	14.03%	3.50%
17 GB	28.65%	40.64%	27.77%	31.87%	41.52%	29.53%
57 GB	49.41%	52.92%	46.78%	52.92%	52.92%	47.66%
116 GB	55.55%	55.55%	58.77%	56.72%	58.18%	60.52%

6.5 Conclusion

In this chapter we presented four different ways of evaluating our *is-a* relations. Generally, the Hybrid system achieves better performance than Pattern-based and Clustering-based system. All three systems outperforms WorNet in most of the cases. In the next chapter, we conclude our work and enlist areas for future research.

Chapter 7

Conclusion and Future Work

Motivated by the need to construct semantic repositories, we proposed a hybrid approach to build *is-a* relations. We now review the contributions and possible areas of future work in the next few sections.

7.1 Randomized Algorithm

The work in this thesis is to the best of our knowledge the first attempt in using randomized algorithms for unsupervised learning in Natural Language Processing. It is very refreshing to hear that a well studied problem, such as finding semantically similar nouns, could be recast as finding nearest neighbors in hamming space. This conversion helps in reducing the complexity from an infeasible $O(n^2k)$ to $O(nk)$, where n is the number of nouns and k is the feature set size. This suggests that NLP research can benefit from work in Theoretical Computer Science and Math community.

7.2 Beyond Noun Clustering: Noun Pair Clustering

In this thesis, we described a fast technique for clustering of nouns. We can extend similar techniques to perform other tasks. One such specific task is noun-pair clustering. In this section, we sketch out some ideas on how this can be achieved. This is very intial work but promises to be an interesting direction for future research.

In the problem of noun clustering, we try to group of nouns that are semantically similar. For the problem of noun pair clustering, we want to group pairs of nouns that share similar relationships. This problem is similar to finding SAT/GRE type analogies.

For example, consider two sentences:

1. Pour *coffee* in a *mug*.

2. Pour *juice* in a *glass*.

We can say the pair *Coffee:mug* relation is analogical to the pair *Juice:glass*. Another example of semantically similar noun pairs are: *Brazil:Portuguese* AND *USA:English*. Such analogies may help us automatically uncover relationships between pair of words. The challenge is to determine whether such lists be constructed automatically? The answer is yes, since we have all the tools required for such effort:

1. A large corpus.

2. High Speed Clustering technique.

We modify the Clustering based algorithm described in Chapter 5, Subsection 5.5.2, to construct semantically related noun pairs:

1. We start out with a huge natural language corpus. We identify all the nouns in the corpus by using a noun phrase identifier. We select all noun pairs that occur within a window of 5 words (occurring at least 100 times).

2. For each noun pair, we identify the context words surrounding it. Our context window length is restricted to one word to the left of the first noun, one word to the right right of second noun and all the words between the noun pairs. We use the context words as features of the noun pair vector.

3. Having collected all nouns pairs and their features, we now proceed to construct feature vectors (and values) for nouns pairs. Randomized algorithms are used to construct similarity matrix. The noun-pair vectors are then clustered using CBC. This is exactly similar to Steps 2, 3, 4 and 5 of the Clustering-based algorithm in Subsection 5.5.2.

Sample output from the algorithmis are given in Table 7.1. The results are not as good as those of single noun clustering but there is a lot of potential for improvement. Some of them are enlisted below:

1. Propogating features from single noun-clustering to noun-pair clustering.

Table 7.1: Sample Noun Pair Similarity Lists

Coffee:Mug	wheel:car	Los Angeles:California	USC:UCLA
Olive oil:saucepan	leg:ground	Chicago:Illinois	Bush:Gore
Water:saucepan	foot:floor	Atlanta:Georgia	Dean:Clark
Oil:saucepan	heel:ground	Denver:Colorado	Kerry:Edwards
Olive Oil:skillet	toe:ground	Orlando:Florida	Kerry:Dean
Butter:saucepan	knee:bed	Las Vegas:Nevada	Clinton:Dole
Milk:bowl	shoulder:wall	San Francisco:California	Dean:Kerry
Olive oil:bowl	wheel:ground	Salt Lake City:Utah	Bush:Nader

2. Making modifications to the choice of centroids for CBC by careful experimentation.

3. Restricting our set to only those noun-pairs that are connected by a verb.

We also would want to find a very good way of evaluating such results.

7.3 Pattern Learning

Similarly, the pattern learning described in Chapter 3 could be used for learning other relations. This algorithm is elegant and simple enough that it could easily be duplicated. All the algorithm only needs, is a list of seed examples. Possible areas of future research would be extraction of other relations such biographical extraction, product categorization etc. This technique along with the one discussed for noun pair clustering could be used for learning more complex relations between nouns.

7.4 Terabyte Challenge

This thesis is to the best of our knowledge the first one to systematically attack the problem of working with terabytes of data in NLP by proposing both an engineering and an algorithmic solution to the problem. We verified the famous conjecture: More data helps![4, 5, 22]. The learning curve in Figure 6.1 suggests that even after downloading 1 Terabyte of data the amount of *is-a* relations that we learn is still linear. NLP researchers have just begun leveraging the vast amount of knowledge available on the web. By searching IR engines for simple surface patterns, many applications ranging from word sense disambiguation, question answering, and mining semantic resources have already benefited. However, most language analysis tools are too infeasible to run on the scale of the web. We need to find faster solutions to run our algorithms on in the Web.

7.5 Tabulanator Idea

Traditional NLP generally involves systems that make use of notational transformation to convert data from one representation to another. E.g., for an MT system, we want to convert a foreign language input to an English language output. For a Question Answering system, we want to transform a question input to an answer output. While building such systems one needs to mostly concentrate on understanding and building these transformation rules (either by using hand-built rules or empirical evidence).

However, this thesis presents a rather different approach in building an NLP system. It is similar to the one proposed by Fleischman et al. [28]. Instead of concentrating on transformation rules, we attempt to build huge tables by computing the results offline. Hence, transformation rules are replaced by fast look-up tables. Techniques like these would become extremely popular, if we are able to design algorithms that could cover a good amount of this possible input space. Koehn [40] reports that there is a 85% chance of seeing a 4-gram and a 65% chance of seeing a 5-gram on the Internet. The corresponding numbers reported on a much smaller newspaper corpus are 42% (4-gram) and 25% (5-gram) respectively. This means that with more growth of data on the Internet these numbers will likely get higher and higher which may lead to a rapid increase in the use of such table-like techniques.

Reference List

[1] Adibi, Jafar and Wei-Men Shen Data Mining Techniques and Applications in Medicine *Tutorial*, 1999.

[2] Agichtein, Eugene and Luis Gravano Snowball: Extracting Relations from Large Plain-Text Collections In the proceedings of the *5th ACM International Conference on Digital Libraries (ACM DL)*, 2000.

[3] Agichtein, Eugene and Luis Gravano Querying Text Databases for Efficient Information Extraction In the proceedings of the *19th IEEE International Conference on Data Engineering (ICDE)*. 2003.

[4] Banko,M., E. Brill. Mitigating the Paucity of Data Problem. In Proceedings of the *Conference on Human Language Technology*, San Diego, CA. 2001.

[5] Banko, Michele, Eric Brill. Scaling to a Very Very Large Corpora for Natural Language Disambiguation. Proceeding of the *Association for Computational Linguistics*, Toulouse, France. 2001.

[6] Barzilay, Regina Information Fusion for Mutlidocument Summarization: Paraphrasing and Generation *PhD Thesis*, Department of Computer Science, Columbia University, 2003.

[7] Berland, M, and E. Charniak. Finding Parts in Very Large Corpora. *Proceedings of the 37 th Annual Meeting of the Association for Computational Linguistics* College Park, MD. 1999.

[8] Berwick, Robert, Steven Abney, and Carol Tenny, editors. Principle-Based Parsing. *Kluwer Academic Publishers.* 1991.

[9] Brin. S. Extracting patterns and relations from the World Wide Web. In Proceedings of the 1998 International Workshop on the Web and Databases (WebDB 98), 1998.

[10] Box, G. E. P. and M. E. Muller *Ann. Math. Stat.* 29, 610–611. 1958.

[11] Brill, E., 1995. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*. 1995

[12] Brill, E., J. Lin, M. Banko, S. Dumais, and A. Ng. 2001. Data-Intensive Question Answering. Proceedings of the *TREC-10 Conference*. NIST, Gaithersburg, MD, 183–189. 2001.

[13] Broder, Andrei On the Resemblance and Containment of Documents. Proceedings of the *Compression and Complexity of Sequences*. 1997.

[14] Callan, J.P., W.B. Croft, and J. Broglio. TREC and Tipster Experiments with Inquery. *Information Processing and Management* 31(3), 327-343. 1995.

[15] Caraballo, Sharon A. Automatic construction of a hypernym-labeled noun hierarchy from text. In Proceedings of *ACL*, pp 20–26. College Park, MD. 1999.

[16] Cavnar, W. B. and J. M. Trenkle N-Gram-Based Text Categorization. In Proceedings of Third Annual Symposium on *Document Analysis and Information Retrieval*, Las Vegas, NV, UNLV Publications/Reprographics, 161–175. 1994.

[17] Cederberg, S. and Widdows, D. Using LSA and Noun Coordination Information to Improve the Precision and Recall of Automatic Hyponymy Extraction. Proceedings of *CoNLL*, pp. 118. 2003.

[18] Charikar, Moses Similarity Estimation Techniques from Rounding Algorithms In Proceedings of the *34th Annual ACM Symposium on Theory of Computing*. 2002.

[19] Charniak, E. A Maximum-Entropy-Inspired Parser. Proceedings of *NAACL*. 2000

[20] Church, K. and Hanks, P. 1989. Word association norms, mutual information, and lexicography. In Proceedings of *ACL-89*. pp. 76–83. Vancouver, Canada.

[21] Ciaramita, M. and Johnson, M. Supersense Tagging of Unknown Nouns in WordNet. Proceedings of *EMNLP*. 2003.

[22] Curran, J. and Moens, M. Scaling context space. In Proceedings of *ACL-02* pp 231–238, Philadelphia, PA. 2002.

[23] Darroch, J. N., and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43:1470–1480. 1972

[24] Etzioni, Oren, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S.Weld, and Alexander Yates. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. To Appear in *Artificial Intelligence*. 2005.

[25] Etzioni, Oren, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S.Weld, and Alexander Yates Methods for Domain-Indepedent Information Extraction from the Web: An Experimental Comparison. In Proceddings of *AAAI*. 2004.

[26] Etzioni, Oren, Michael Cafarella, Doug Downey, Stanley Kok Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld and Alexander Yates Web-scale Information Extraction in KnowItAll. In Proceedings of *WWW*. 2004.

[27] Fayyad, Usama M, Gregory Piatetsky-Shapiro, Padhraic Smyth and Ramasamy Uthurusamy. Advances in Knowledge Discovery and Data Mining. *M.I.T. Press*, 1996.

[28] Fleischman, M., E. Hovy and A. Echihabi. Offline Strategies for Online Question Answering: Answering Questions Before They Are Asked. Proceedings of the 41 st Meeting of the *Association for Computational Linguistics (ACL)*. Sapporo, Japan. 2003

[29] Girju, Roxana, Adriana Badulescu and Dan Moldovan. Learning Semantic Constraints for the Automatic Discovery of Part-Whole Relations. *Proceedings of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. Edmonton, Canada. 2003.

[30] Goemans, M. X. and D. P. Williamson Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *JACM* 42(6): 1115–1145. 1995.

[31] Gray, Jim. A Conversation with Jim Gray *ACM Queue* vol. 1, no. 4 - June 2003.

[32] Grishman, Ralph Information Extraction: Techniques and Challenges. *Information Extraction (International Summer School SCIE-97)*, ed. Maria Teresa Pazienza, Springer-Verlag. 1997

[33] Gusfield, Dan. Algorithms on Stings, Trees and Sequences *Cambridge University Press*, New York. 1997

[34] Hearst, Marti Automatic Acquisition of Hyponyms from Large Text Corpora. *Proceedings of the Four-teenth International Conference on Computational Linguistics*, Nantes, France. 1992.

[35] Hearst, Marti TextTiling: Segmenting Text into Multi-Paragraph Subtopic Passages. *Computational Linguistics*, 23 (1), pp. 33–64. 1997.

[36] Hearst, Marti What is Text Mining? *http://www.sims.berkeley.edu/ hearst/text-mining.html*. 2003.

[37] Hobbs, Jerry R., Douglas Appelt, John Bear, David Isreal, Megumi Kameyama, Mark Stickel, and Mabry Tyson. FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. In E.Roche and Y. Schabes, editors, *Finite State Devices for Natural Language Processing*. MIT Press, Cambridge, MA. 1996.

[38] Hovy, E.H., U. Hermjakob, and C.-Y. Lin. The Use of External Knowledge in Factoid QA. Proceedings of the *TREC-10 Conference*. NIST, Gaithersburg, MD, 166–174. 2001.

[39] Hovy, E.H., U. Hermjakob, D. Ravichandran. 2002. A Question/Answer Typology with Surface Text Patterns. Proceedings of the *DARPA Human Language Technology Conference*, San Diego, CA, 247–250. 2002.

[40] Koehn, Philipp. Noun Phrase Translation. PhD thesis, *University of Southern California*. 2003.

[41] Hindle, D. Noun classification from predicate-argument structures. In Proceedings of ACL-90. pp. 268–275. Pittsburgh, PA. 1990.

[42] Indyk, P., Motwani, R. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality Proceedings of *30th STOC*, 604–613.

[43] KnowledgeMaster. *http://www.greatauk.com. Academic Hallmarks*. 1999.

[44] Kolcz, A., A. Chowdhury, J. Alspector 2004. Improved robustness of signature-based near-replica detection via lexicon randomization. Proceedings of *ACM-SIGKDD*. 2004.

[45] Lehnert, Wendy, Claire Cardie, David Fisher, Ellen Riloff, and Robert William. Description of the CIRCUS System as Used for MUC-3 *Proceedings*, Third Message Understanding Conference (MUC-3), San Diego, California, pp. 223–233. 1991.

[46] Lenat, D. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38. Research Service. 1995.

[47] Lin, Dekang. Principar - an Efficient, Broad-Coverage, Principle-Based Parser. In Proceedings of *COLING-94*. pp. 42-48. Kyoto, Japan. 1994.

[48] Lin, D. Automatic retrieval and clustering of similar words. In Proceedings of *COLING/ACL-98*. pp. 768–774. Montreal, Canada. 1998.

[49] Lin, Chin-Yew. The Effectiveness of Dictionary and Web-Based Answer Reranking. In Proceedings of the *19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan. 2002.

[50] Magnini, B, M. Negri, R. Prevete, and H. Tanev. Is it the Right Answer? Exploiting Web Redundancy for Answer Validation. Proceedings of the 40th Meeting of the *Association of Computational Linguistics*, Philadelphia, PA, 425–432. 2003.

[51] Mann, Gideon S. Fine-Grained Proper Noun Ontologies for Question Answering Proceedings of *SemaNet: Building and Using Semantic Networks*, Taipei, Taiwan. 2002.

[52] McQueen, J. Some methods for classification and analysis of multivariate observations. In Proceedings of 5th Berkeley Symposium on Mathematics, Statistics and Probability, 1:281–298. 1967.

[53] Miller, G. A., R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: An on-line lexical database. *Journal of Lexicography*, 3(4):235–244. 1990.

[54] Moldovan Dan and Roxanna Girju. Knowledge Discovery from Text. *Tutorial ACL*, Sapporo, Japan. 2003.

[55] Pantel, Pantel and Dekang Lin. Discovering Word Senses from Text. In Proceedings of *SIGKDD-02*, pp. 613–619. Edmonton, Canada. 2002.

[56] Pantel, Patrick and Deepak Ravichandran Automatically Labelling Semantic Classes. In Proceedings of *NAACL/HLT*, Boston, MA. 2004.

[57] Paşca, Marius Acquisition of categorized named entities for web search. In: Proceedings of the *13th ACM Conference on Information and Knowledge Management (CIKM-04)*, Washington, D.C. 2004.

[58] Paşca, Marius Finding Instance Names and Alternate Glosses on the Web: WordNet Reloaded. In Proceedings of *CICLing*, pp. 280–292. Mexico City, Mexico. 2005.

[59] Rabin, M. O. Fingerprinting by random polynomials. Center for research in Computing technology , Harvard University, *Report* TR-15-81. 1981.

[60] Ravichandran, Deepak, and Eduard Hovy. Learning Surface Text Patterns for a Question Answering System. Proceedings of the 40th Meeting of the *Association of Computational Linguistics* (ACL). Philadelphia, PA. 41–47. 2002.

[61] Ravichandran, Deepak, Eduard Hovy and Franz Josef Och. Statistical QA - Classifier vs Re-ranker: What's the difference? In Proceedings of the *ACL Workshop on Multilingual Summarization and Question Answering–Machine Learning and Beyond*, Sapporo, Japan. 2003.

[62] Reynar, Jeffrey C. and Adwait Ratnaparkhi. A Maximum Entropy Approach to Identifying Sentence Boundaries. In Proceedings of the Fifth Conference on *Applied Natural Language Processing*. Washington, D.C. 1997.

[63] Riloff, Ellen Automatically Generating Extraction Patterns from Untagged Text. *Proceedings of the Thirteenth National Conference on Artifical Intelligence*, pp. 1044–1049. 1996.

[64] Salton, G. and McGill, M. J. *Introduction to Modern Information Retrieval*. McGraw Hill. 1983.

[65] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng, Learning syntactic patterns for automatic hypernym discovery *Proceedings of Neural Information Processing*. Vancouver, Canada. 2004.

[66] Sundheim, Beth, ed., Third Message Understanding Conference (MUC-3). *Proceedings*, San Mateo, CA. 1991.

[67] Sundheim, Beth, ed., Fourth Message Understanding Conference (MUC-4). *Proceedings*, San Mateo, CA. 1992.

[68] Sundheim, Beth, ed., Fifth Message Understanding Conference (MUC-5). *Proceedings*, San Mateo, CA. 1993.

[69] Voorhees, E. Overview of the Question Answering Track. *Proceedings of the TREC-9 Conference*. NIST, Gaithersburg, MD, 71–80. 2000.

[70] Voorhees, E. Overview of the Question Answering Track. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 157–165. 2001.

[71] Voorhees, E. Overview of the Question Answering Track. *Proceedings of the TREC-12 Conference*. NIST, Gaithersburg, MD. 2003.

[72] Voorhees, E. Overview of the Question Answering Track. *Proceedings of the TREC-12 Conference*. NIST, Gaithersburg, MD. 2004.

[73] Wagner, R. A. and Fisher, M. J. The string to string correction problem. *Journal of the ACM*, 21(1), 168–173, 1974.

[74] Weiner, P. Linear Pattern Matching Algorithm. *Proceedings of 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11. 1973.

Appendix A

Random Permutation Function

We define $[n] = \{0, 1, 2, ..., n-1\}$.

$[n]$ can thus be considered as a set of integers from 0 to $n-1$.

Let $\pi : [n] \rightarrow [n]$ be a permutation function chosen at random from the set of all such permutation functions.

Consider $\pi : [4] \rightarrow [4]$.

A permutation function π is a one to one mapping from the set of $[4]$ to the set of $[4]$.

Thus, one possible mapping is:

$\pi : \{0, 1, 2, 3\} \rightarrow \{3, 2, 1, 0\}$

Here it means: $\pi(0) = 3$, $\pi(1) = 2$, $\pi(2) = 1$, $\pi(3) = 0$

Another possible mapping would be:

$\pi : \{0, 1, 2, 3\} \rightarrow \{3, 0, 1, 2\}$

Here it means: $\pi(0) = 3$, $\pi(1) = 0$, $\pi(2) = 1$, $\pi(3) = 2$

Thus for the set $[4]$ there would be $4! = 4 * 3 * 2 = 24$ possibilities. In general, for a set $[n]$ there would be $n!$ unique permutation functions. Choosing a random permutation function amounts to choosing one of $n!$ such functions at random.

Appendix B

Sample System Output

We enlist the outputs of different systems for five sample concepts: *color*, *car*, *pasta*, *scientist* and *TV show*. Only the top 25 instances for each concept are chosen. Results are displayed for all the three systems: Pattern-based, Clustering-based and Hybrid. The output of each system is compared against different corpora sizes: 1.5GB, 17GB, 57GB and 116GB. One can notice that the output of the Hybrid system is much cleaner than the other two systems.

Table B.1: List of colors – Pattern-based System

	1.5GB	17GB	57GB	116GB
1	lip	lip	red	red
2	red	red	black	blue
3	pink	black	blue	black
4	black	blue	lip	white
5	Religion	white	white	green
6	texture	green	yellow	yellow
7	green	yellow	green	style
8	blue	pink	style	size
9	Lynx	Purple	size	lip
10	yellow	orange	brown	pink
11	Calico	color	orange	orange
12	Blacky	shape	color	Purple
13	purple	gray	pink	brown
14	shade	burgundy	Purple	gray
15	Grey	silver	texture	color
16	words	Mango	silver	texture
17	size	texture	religion	religion
18	white	RED , BLUE & PURPLE	black and white	silver
19	sepia	brown	gray	black and white
20	mask	race	shape	shape
21	lime	National Origin	background	fabric
22	buckskin	Aqua	pastel	pastel
23	parti	style	shade	cream
24	racism	frequency	frequency	shade
25	side	black and white	cream	pattern

Table B.2: List of colors - Clustering-based System

	1.5GB	17GB	57GB	116GB
1	RED	Turquoise	Forest Green	navy blue
2	White	RED	RED	Ochre
3	Yellow	BLACK	Light Blue	VIOLET
4	BLUE	WHITE	BLACK	MAROON
5	PINK	AMETHYST	Hunter Green	Palomino
6	BLACK	PERIDOT	WHITE	BLACK
7	PURPLE	lapis	Dark Green	yellow green
8	GREEN	CARNELIAN	Sky Blue	Light Grey
9	Brown	Malachite	YELLOW	WHITE
10	Grey	YELLOW	bottle green	MAUVE
11	Gray	onyx	KELLY GREEN	METALLIC SILVER
12	violet	Aquamarine	PINK	Tobiano
13	TAN	Garnet	MAUVE	blueish
14	Orange	CITRINE	PURPLE	overo
15	GOLD	Tourmaline	GREEN	OFF WHITE
16	SILVER	Topaz	blue	greenish yellow
17	Lavender	rose quartz	navy blue	DARK GREY
18	pale	Emerald	khaki	orange red
19	DARK	Moonstone	sage green	Pearl White
20	lilac	SAPPHIRE	Burgandy	Sabino
21	dark blue	PINK	GREY	Buckskin
22	Fawn	opal	HOT PINK	royal blue
23	Tabby	BLUE	Lime Green	APPALOOSA
24	buff	GREEN	Fuschia	GRULLA
25	Crimson	lapis lazuli	BROWN	orange yellow

Table B.3: List of colors – Hybrid System

	1.5GB	17GB	57GB	116GB
1	red	red	red	red
2	pink	black	black	blue
3	black	blue	blue	black
4	green	white	white	white
5	blue	green	yellow	green
6	yellow	yellow	green	yellow
7	purple	pink	brown	pink
8	Grey	Purple	color	orange
9	white	orange	pink	Purple
10	lip	color	Purple	brown
11	Religion	gray	texture	gray
12	texture	burgundy	silver	color
13	Lynx	silver	black and white	black and white
14	brown	brown	gray	cream
15	gray	Mango	shade	shade
16	violet	Aqua	cream	taupe
17	tan	black and white	burgundy	grey
18	orange	grey	Appaloosa	beige
19	lavender	shade	amber	maroon
20	pale	taupe	Aqua	Navy
21	dark	violet	taupe	Burgundy
22	lilac	dark	beige	navy blue
23	dark blue	amber	additive	violet
24	fawn	crimson	dark red	dark blue
25	tabby	magenta	navy blue	Amber

Table B.4: List of cars – Pattern-based System

	1.5GB	17GB	57GB	116GB
1	Gun	Gun	credit card	credit card
2	Hemi	chassis	Gun	Carsbymail.co.uk
3	Traxxas Rustler	SGV	job	Buenos Aires
4	CHP CAD	custom	Buenos Aires	Gun
5	Ford Falcon	Lotus	Mercedes	job
6	Suddenly	VW	Carsbymail.co.uk	Mercedes
7	Wheel	Buenos Aires	BMW	BMW
8	Smart Car Leasing LeaseGuide.com	classic	custom	Ford
9	indica	Mercedes	driver	bike
10	VW	audi	chassis	Ferrari
11	TCRA	Ferrari	Carl	VWs
12	Mile	Rolls Royce	Ferrari	jaguar
13	List Limousines	Car Rallye	Ford	Cadillacs
14	clutch	Notable TV Guest Appearances Gun	Classic	audi
15	Toyota Supra TT	Ferraris	Ferraris	Carl
16	Henry Fonda	Southeast Missourian	Jaguar	chassis
17	Albert Renick	indica	Cadillac	Porsche
18	Mercedes	mile	SGV	Honda
19	audi	Corrado	Cadillacs	mustang
20	Japanese HW	PP	Aston Martin	Aston Martin
21	BF Injection	jaguar	Honda	Ride
22	TIME.COM Win Money	Memphis	audi	corvette
23	GTO Cobra	Rachele	rambler	Ferraris
24	Lofoten	Hatchback	SUVs	Cadillac
25		Kung Foundation	mile	Toyota

Table B.5: List of cars - Clustering-based System

	1.5GB	17GB	57GB	116GB
1		BMW	Cadillacs	TOYOTAS
2		Mercedes	BMW	TOYOTA
3		PORSCHE	BUICKS	CHEVROLET
4		Audi	CHEVROLETS	Lincoln Continental
5		VOLVO	MAZDA	CHEVY
6		SAAB	PONTIACS	Volkswagen Passat
7		Volkswagen	Audi	Buick LeSabre
8		Mercedes Benz	VolksWagen	NISSAN
9		HONDA	Oldsmobiles	BUICK REGAL
10		BENZ	Renault	Subarus
11		VW	PORSCHE	Toyota Avalon
12		LEXUS	VOLVO	Ford Falcon
13		Ferrari	Peugeot	VW Bug
14		Peugeot	VW	Subaru Outback
15		MAZDA	MERCEDES	PONTIAC
16		NISSAN	NISSAN	Chevy Impala
17		Renault	TOYOTA	ACURAS
18		JAGUAR	HONDA	Dodge Dart
19		Maserati	SAAB	GMC
20		Acura	SUBARU	VOLVO
21		Coupe	LEXUS	BUICK
22		SUBARU	HYUNDAI	Nissans
23		SEDAN	CHRYSLERS	Mazdas
24		Lamborghini	Isuzu	HONDAS
25		CONVERTIBLE	CHRYSLER	LEXUS

Table B.6: List of cars – Hybrid System

	1.5GB	17GB	57GB	116GB
1	Gun	Ferrari	Mercedes	Mercedes
2		VW	BMW	BMW
3		Rolls Royce	Ferrari	Ford
4		Mercedes	Ford	Ferrari
5		audi	Ferraris	jaguar
6		jaguar	Jaguar	Cadillacs
7		Hatchback	Cadillac	audi
8		Coupe	Cadillacs	Porsche
9		Gun	audi	corvette
10		chassis	Aston Martin	mustang
11		SGV	Honda	Aston Martin
12		Lotus	SUVs	Cadillac
13		classic	Toyota	VW
14		Buenos Aires	Bentley	Porsches
15		Car Rallye	VW	SUVs
16		Notable TV Guest Appearances Gun	corvette	BMWs
17		Corrado	roadster	Roadster
18		Southeast Missourian	mustang	Miata
19		indica	Coupe	Lexus
20		mile	Porsche	Hondas
21		PP	Lexus	Rolls Royce
22		Ferraris	Rolls Royce	Toyota Prius
23		saab	Porsches	sedan
24		mercedes benz	Buick	Nissan
25		benz	Nissan	Coupe

Table B.7: List of pasta – Pattern-based System

	1.5GB	17GB	57GB	116GB
1	ditali	lasagne	spaghetti	spaghetti
2		spaghetti	penne	penne
3		ravioli	ravioli	ravioli
4		fettuccine	fettuccine	fettuccine
5		Caesar salad	fusilli	linguine
6		penne	lasagne	agnolotti
7		risotto	linguine	fusilli
8		Peperonata Pasta	Cous Cous	lasagne
9		rigatoni	macaroni	Cous Cous
10		ditali	capellini	macaroni
11			tagliatelle	gemelli
12			lasagna	tortellini
13			gemelli	bowties
14			bowties	capellini
15			ditali	ditali
16			Couscous	tagliatelle
17			agnolotti	farfalle
18			Peperonata Pasta	Couscous
19			tortellini	Little Italy
20			spaghettini	Peperonata Pasta
21			farfalle	ditalini
22			Black Fettuccine	spaghettini
23			Angel Hair	Spaghetti Carbonara
24			Antonio Orlando	Fettuccini
25			Little Italy	Paglia & Fieno

Table B.8: List of pasta – Clustering-based System

	1.5GB	17GB	57GB	116GB
1		PENNE	PASTA	risotto
2		linguine	SPAGHETTI	Ravioli
3		Tortellini	SALAD	PENNE
4		ORZO	Noodle	Linguine
5		Rigatoni	SOUP	fettuccine
6		Ziti	PESTO	TORTELLINI
7		SPAGHETTI	SAUSAGE	Linguini
8		Fettuccine	SHRIMP	PASTA
9		PASTA	PENNE	Lasagne
10		Vermicelli	Lasagna	Gnocchi
11		Lasagna	Ravioli	Lasagna
12		linguini	MaCaRoNi	SPAGHETTI
13		Macaroni	SEAFOOD	Fettuccini
14		fettuccini	PIZZA	marinara
15		Ravioli	fettuccine	PESTO
16		Pesto	Linguine	cream sauce
17		lasagne	marinara	Rigatoni
18		risotto	APPETIZER	Polenta
19		Noodle	VEGGIE	ZITI
20		egg noodle	SANDWICH	Fettucine
21		Gnocchi	risotto	Dumpling
22		wild rice	Linguini	MEATBALL
23		MANICOTTI	Meatball	orzo
24		Polenta	Tortellini	Fettucini
25		meatball	GRAVY	manicotti

Table B.9: List of pasta – Hybrid System

	1.5GB	17GB	57GB	116GB
1		lasagne	spaghetti	spaghetti
2		ravioli	penne	penne
3		fettuccine	ravioli	ravioli
4		spaghetti	fettuccine	fettuccine
5		Caesar salad	fusilli	linguine
6		penne	lasagne	fusilli
7		risotto	linguine	lasagne
8		rigatoni	macaroni	macaroni
9		linguine	capellini	agnolotti
10		tortellini	tagliatelle	tortellini
11		orzo	tortellini	tagliatelle
12		ziti	farfalle	capellini
13		vermicelli	Rigatoni	farfalle
14		linguini	Fettuccini	spaghettini
15		fettuccini	Cous Cous	Fettuccini
16		egg noodle	gemelli	Gnocchi
17		gnocchi	bowties	ziti
18		manicotti	ditali	fettucine
19		polenta	Couscous	Angel Hair
20		marinara	agnolotti	Rigatoni
21		pollo	Peperonata Pasta	manicotti
22		macaroni and cheese	spaghettini	Cous Cous
23		cream sauce	meatball	gemelli
24		bow tie	gnocchi	bowties
25		antipasto	main course	ditali

Table B.10: List of scientists – Pattern-based System

	1.5GB	17GB	57GB	116GB
1	James Bell	Newton	Einstein	Einstein
2	physicist	chemist	Galileo	Galileo
3	technician	engineer	physicist	Albert Einstein
4	J.A Monro	artist	scientist	physicist
5	John Macadam	Dad	engineer	Kary Mullis
6	Riggi	Einstein	artist	engineer
7	Traveler	Scientist Horst Frank	Kary Mullis	scientist
8	Stephen Morse	Albert Einstein	Albert Einstein	biologist
9	chemist	physicist	Newton	Newton
10	VHS Boris Karloff	Nobel	Pierre Perrin	Carl Sagan
11	Franklin	Professor	chemist	Darwin
12	Szent	Edward Miles	Darwin	chemist
13	Cynthia Rosenzweig	Benes	Evolution Articles	Gordon Freeman
14	protagonist	inventor	Frankenstein	FRANKENSTEIN
15	Merlin	Thomas Rathier	archaeologist	Pierre Perrin
16	Mathematician	Keniclius	Carl Sagan	Some
17	Dad	Craig	Fenimore Mulder	Guinness
18	Consequently	Isaac Newton	Gordon Freeman	Nobel prize
19	Drexler	Galileo	Rachel Carson	biochemist
20	Carroll	Kary Mullis	Oppenheimer	Boris Karloff
21	telescope	Lambert Dolphin	geologist	Prof
22	Thomas Henry Huxley	Peyton Rous	experiment	anthropologist
23	Peter Coyote	Experiment	Isaac Newton	Chris McKay
24	Albert Einstein	Robinson	Nobel	geologist
25	Isaac Newton	sociologist	Guinness	George Washington Carver

Table B.11: List of scientists – Clustering-based System

	1.5GB	17GB	57GB	116GB
1		archaeologist	Sir Isaac Newton	Biologist
2		Isaac Newton	Biologist	JAMES CLERK MAXWELL
3		Sir Isaac Newton	Fred Hoyle	FRED HOYLE
4		Archeologist	David Hilbert	NIELS BOHR
5		Robert Boyle	George Boole	Humphry DAVY
6		Sociologist	John Napier	ALIBEK
7		Michael Faraday	Leonhard Euler	Shealy
8		anthropologist	Geologist	STEVEN WEINBERG
9		James Clerk Maxwell	John Wallis	Thomas Huxley
10		Paleontologist	Georg Cantor	Richard Crenna
11		Geologist	Daniel Bernoulli	Richard Leakey
12		Max Planck	WILLIAM CROOKES	Garrett Hardin
13		biologist	Edwin Arnold	Clifford Geertz
14		physicist	John Herschel	WERNER HEISENBERG
15		Astronomer	ecologist	Vernon Smith
16		BOTANIST	BOTANIST	EDMUND HALLEY
17		linguist	Edmund Halley	David Brewster
18		social scientist	Pierre de Fermat	Geologist
19		Albert Einstein	William Herschel	Desmond Morris
20		philologist	John Egan	WILY
21		ethnologist	EDMOND HALLEY	Ronald Colman
22		Geographer	Zoologist	Mary Douglas
23		art historian	Paleontologist	MAX PLANCK
24		zoologist	anthropologist	Leo McKern
25		Ethnographer	SCIENTIST	Norman Myers

Table B.12: List of scientists – Hybrid System

	1.5GB	17GB	57GB	116GB
1		chemist	physicist	Albert Einstein
2		Albert Einstein	scientist	physicist
3		physicist	engineer	Kary Mullis
4		sociologist	Albert Einstein	engineer
5		Isaac Newton	chemist	scientist
6		geologist	Frankenstein	biologist
7		ethnologist	archaeologist	Carl Sagan
8		ornithologist	Carl Sagan	chemist
9		astrophysicist	geologist	Gordon Freeman
10		Brian Cox	Isaac Newton	Boris Karloff
11		Vincent Price	Paul Davies	geologist
12		Peter Coyote	George Washington Carver	biochemist
13		Gregory Peck	anthropologist	George Washington Carver
14		botanist	Marie Curie	Sir Isaac Newton
15		archaeologist	astronomer	entomologist
16		geophysicist	Sir Isaac Newton	Isaac Newton
17		Alan Bates	Robert Young	archaeologist
18		linguist	John Stuart Mill	Paul Davies
19		Max Planck	biochemist	Stephen Hawking
20		astronomer	Kepler	Dawkins
21		entomologist	Stephen Hawking	mathematician
22		George Washington Carver	Nikola Tesla	Michael Hansen
23		social scientist	paleontologist	immunologist
24		Sir Isaac Newton	entomologist	Abdul Qadeer Khan
25		Newton	ornithologist	Rachel Carson

Table B.13: List of TV Shows – Pattern-based System

	1.5GB	17GB	57GB	116GB
1	Lassie	SEG	SEG	Survivor
2	Grapevine	Dawson	Dawson	Star Trek
3	Moonlighting	Survivor	Seinfeld	SEG
4	"The"	Seinfeld	ER	Seinfeld
5	Xena	Praise Be	Survivor	Dawson
6	Tragedy	extra	Oprah	Sesame Street
7		Warner Bros	Sesame Street	Buffy
8		Moonlighting	Australian Idol	Cop
9		Good News Week	Xena	Oprah
10		SOUTH PARK	Izumi	Australian Idol
11		Fantasy Island	SKY KING	American Idol
12		CBC Radio	Praise Be	Baywatch
13		"The"	diff	Jeopardy
14		Will & Grace	American Bandstand	diff
15		red dwarf	Buffy	Montel
16		Baywatch	Montel	David Letterman
17		Kikaida	David Letterman	Xena
18		Lassie	red dwarf	Alfred Hitchcock Presents
19		Watteru Baai Desu Yo	West Wing	Letterman
20		Entertainment Tonight	Jeopardy	Popstars
21		Dynasty	Happy Days	Politically Incorrect
22		Tony Todd	Popstars	SKY KING
23		West Wing	Baywatch	Pop Idol
24		Jeopardy	Lassie	Izumi
25		diff	Pop Idol	LASSIE

Table B.14: List of TV Shows – Clustering-based System

	1.5GB	17GB	57GB	116GB
1		General Electric Theater	Tales of the Unexpected	STUDIO ONE
2		Miami Vice	Highway To Heaven	TOUCHED BY AN ANGEL
3		Studio One	GENERAL ELECTRIC THEATER	GENERAL ELECTRIC THEATER
4		THIRD WATCH	STUDIO ONE	Highway To Heaven
5		Highway To Heaven	Letter to Loretta	TALES FROM THE CRYPT
6		Alfred Hitchcock Presents	Simon & Simon	MIAMI VICE
7		L.A Law	Ford Television Theatre	Letter to Loretta
8		Letter to Loretta	THIRD WATCH	MARRIED WITH CHILDREN
9		Tales Of The Unexpected	Miami Vice	THIRD WATCH
10		Simon & Simon	Death Valley Days	THIS IS YOUR LIFE
11		Diagnosis Murder	Four Star Playhouse	Simon & Simon
12		Touched By An Angel	Civil Wars	Eight Is Enough
13		Tales From the Crypt	DuPont Show	Four Star Playhouse
14		Empty Nest	Cavalcade of America	Tales Of The Unexpected
15		Fantasy Island	Tales from the Darkside	Kraft Television Theatre
16		This Is Your Life	Any Day Now	Alcoa Presents
17		Married With Children	Lux Video Theatre	Any Day Now
18		Midsomer Murders	Eight Is Enough	Cavalcade of America
19		Any Day Now	This Is Your Life	Ford Television Theatre
20		Hill Street Blues	It Takes a Thief	Alfred Hitchcock Presents
21		Mad About You	Tales From The Crypt	DuPont Show
22		Alcoa Presents	Kraft Television Theatre	Civil Wars
23		Silk Stalkings	Matt Houston	New York Undercover
24		Party Of Five	New York Undercover	It Takes A Thief
25		Ford Television Theatre	Alfred Hitchcock Hour	Doctor WHO

Table B.15: List of TV Shows – Hybrid System

	1.5GB	17GB	57GB	116GB
1		Dawson	Dawson	Star Trek
2		Seinfeld	Seinfeld	Dawson
3		Moonlighting	Xena	Baywatch
4		Will & Grace	SKY KING	Politically Incorrect
5		diff	diff	SKY KING
6		Dynasty	red dwarf	American Bandstand
7		Lassie	Baywatch	Alfred Hitchcock Presents
8		Fantasy Island	Lassie	Temptation Island
9		West Wing	West Wing	Who Wants
10		Full House	Forever Knight	Queer Eye
11		Maverick	Happy Days	Felicity
12		Ellery Queen	Moonlighting	Real TV
13		Forever Knight	Taggart	Gunsmoke
14		Baywatch	Relic Hunter	Forever Knight
15		Entertainment Tonight	Suddenly Susan	Happy Days
16		Trading Spaces	Hawaiian Eye	Entertainment Tonight
17		Flying Nun	Full House	Moonlighting
18		SEG	Ellery Queen	Trading Spaces
19		Survivor	White Shadow	Different Strokes
20		Praise Be	Police Woman	Taggart
21		Warner Bros	Nash Bridges	Bay Watch
22		Good News Week	Lone Gunmen	Relic Hunter
23		general electric theater	Will & Grace	LA Law
24		miami vice	Lancer	Sabrina The Teenage Witch
25		studio one	Fantasy Island	Suddenly Susan