Seman
tic Search with Information Integration

Author: Yikun Xian & Liu Zhang
Date: 2011-08-18
Subject: Computer Science
Level: Bachelor
Course code: 2DV00E
Abstract

Since the search engine was first released in 1993, the development has never been slow down and various search engines emerged to vied for popularity. However, current traditional search engines like Google and Yahoo! are based on key words which lead to results impreciseness and information redundancy. A new search engine with semantic analysis can be the alternate solution in the future. It is more intelligent and informative, and provides better interaction with users.

This thesis discusses the detail on semantic search, explains advantages of semantic search over other key-word-based search and introduces how to integrate semantic analysis with common search engines. At the end of this thesis, there is an example of implementation of a simple semantic search engine.

Keywords: Semantic Analysis, Search Engine, NLP, J2EE
Acknowledgments

Thanks to…
… Supervisor Tobias Andersson Gidlund for his help on thesis,
… Professor Mathias Hedenborg for his lectures, where we learn how to write thesis,
… Our classmates who attend the same course.
Växjö, May 2011
Table of content

1 Introduction..................................................................................................................1
  1.1 Problem Description ............................................................................................1
  1.2 Motivation..............................................................................................................1
  1.3 Goals and Criteria ...............................................................................................1
  1.4 Outline..................................................................................................................2

2 Background ..............................................................................................................3
  2.1 J2EE and MVC .......................................................................................................3
  2.2 Spring + Struts2 framework ..................................................................................4
  2.3 Nutch ....................................................................................................................4
  2.4 NLTK ....................................................................................................................5
  2.5 Similar Semantic Search Engine ...........................................................................6

3 Natural Language Processing .................................................................................8
  3.1 Introduction to NLP ..............................................................................................8
    3.1.1 What is NLP? ..................................................................................................8
    3.1.2 Comparison with other data processing .........................................................8
    3.1.3 Core nature of NLP .......................................................................................8
    3.1.4 Progress of NLP development ........................................................................9
  3.2 Natural Language Analysis Mechanism ..............................................................9
    3.2.1 Sentence segmentation and word tokenization .............................................9
    3.2.2 Stemmer .......................................................................................................10
    3.2.3 Part-of-speech tagging ..................................................................................10
    3.2.4 Text Classification .......................................................................................12
    3.2.5 Chunking and name entity recognition .......................................................14
    3.2.6 Context-Free Grammars .............................................................................16
    3.2.7 Feature-based Grammar .............................................................................17
  3.3 NLTK – an Open Source Package .......................................................................19
    3.3.1 A set of tools ...............................................................................................19
    3.3.2 Various methods to implement semantic search .........................................20

4 Search Engine .........................................................................................................23
  4.1 Nutch Architecture ..............................................................................................23
  4.2 Improvements and optimization ..........................................................................24

5 Integration and Implementation .............................................................................26
  5.1 Technological Integration Overview .......................................................................26
  5.2 J2EE and Nutch......................................................................................................27
  5.3 J2EE and NLTK ......................................................................................................28
  5.4 Nutch and NLTK ...................................................................................................28
    5.4.1 Integrate fuzzy search ...................................................................................28
    5.4.2 Integrate text classification ..........................................................................28
1 Introduction

This chapter gives a description of this thesis, including the problem to be solved and the goals and criteria that are needed to be met. Moreover, it describes our motivation to do it and the outline of it.

1.1 Problem Description

As the fast development of our society, Internet has become a necessity in the daily life. Abundant information is hidden in billions of web pages on the Internet, but luckily, search engines like Google and Yahoo!, which are mainly based on key words, make it easier to find what we want.

Key-word-based search engine, which simply matches those key words with HTMLs on different web pages and list the web pages containing these words, however may cause a series of problem.

- Users have to be familiar with the rule of constructing key words.
- Low match rate leads to low precision.
- Search results only contain segments of web pages and links.
- Too much irrelevant information on web pages is presented.
- Monotonous hints do not appeal to various users.

Luckily, semantic network associated with Web 2.0 appears to solve these problems. Relatively speaking, semantic search improves result accuracy through understanding user intent and contextual meaning of the natural language. This thesis including our project is mainly focused on how to solve these problems through this new technology.

1.2 Motivation

Every time, when we want to look for the cheapest plane tickets in the Internet, only cheap ticket-booking websites will be listed. It costs a lot of time to open links, compare prices and choose the most suitable one. For those who seldom use computer, it is difficult for them to search something with specific key words. Inconvenience leads to complaint on such search engines. This, together with our interest in search engine, drives us to figure out a feasible way to relieve the problem.

Artificial Intelligence is becoming more and more popular these days, so we attempt to develop an intelligent search engine. When we read the papers related to semantic analysis, we think it is a good idea to combine the semantic analysis with search engine so that the search engine will become more intelligent. For example, if the user inputs “What is the cheapest plane ticket from Sweden to China.” the result will be a ticket list from the cheapest one to the dearest one and all these tickets are available. Instead of listing all website links directly, the search engine does semantic analyzes first. We believe this kind of search engine will make people do Internet search easier and more convenient.

1.3 Goals and Criteria

In this part, we will talk about the goals and the criteria which this thesis will reach.
• The first goal is to understand semantic analysis mechanism thoroughly. We will refer to relative materials carefully and study the method of semantic analysis. We will learn and improve the semantic analysis algorithm, then implement it in a certain programming language.

• The second goal is to find an open source search engine and skillfully using it. There are so many open source software nowadays. As the search engine has already been developed for a long time, there must be an open source search engine so that we can modify it as we want. Using its API will make it easier for our project.

• The third goal is to add the semantic analysis to the search engine. When we finished the above goals, the next difficulty is to combine them together. Only if these two parts work together, will the intelligent search engine be constructed.

• The last goal is to use a novel way to show the searching result. After a lot of information is found, it is needed to be filtered. The program can identify the most closely related information and order them from the most important part to the least important part, then, the result will be shown to the user.

1.4 Outline
This thesis’s structure is as following:

• Chapter 2 presents the background knowledge of this thesis and the necessary technology of the project to be introduced like J2EE, MVC architecture, Spring and Struts2 framework, an open source search engine Nutch and the nature language processing engine NLTK. This chapter can be skipped if all these knowledge has been known.

• Chapter 3 describes semantic analysis mechanism and related theories in detail.

• Chapter 4 talks about the detailed information about the open source search engine Nutch.

• Chapter 5 discusses how to combine the semantic analysis part and search engine part and how this project is implemented.

• Chapter 6 illustrates the test result and compares the key-word-based search with semantic search.

• Chapter 7 reaches a conclusion on this thesis and the work which can be continually done in the future.
2 Background

This chapter talks about the technologies that are used to solve the problem and realized the project. It gives a brief description about J2EE, MVC, Spring plus Struts2, Nutch and NLTK.

2.1 J2EE and MVC

J2EE is short for Java 2 Platform, Enterprise Edition. It defines the standard for developing multitier enterprise applications. The J2EE platform simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

The J2EE platform takes advantage of many features of the Java 2 Platform, Standard Edition (J2SE), such as "Write Once, Run Anywhere" portability, JDBC API for database access, CORBA technology for interaction with existing enterprise resources, and a security model that protects data even in internet applications. Building on this base, the Java 2 Platform, Enterprise Edition adds full support for Enterprise JavaBeans components, Java Servlets API, Java Server Pages and XML technology. The J2EE standard includes complete specifications and compliance tests to ensure portability of applications across the wide range of existing enterprise systems capable of supporting the J2EE platform. In addition, the J2EE specification now ensures Web services interoperability through support for the WS-I Basic Profile.

All of the advantages for J2EE technology are suitable for our project, so we choose it as the basic technology.

MVC is short for Model View Controller. It is a kind of software architecture, currently considered an architectural pattern used in software engineering. The pattern isolates “domain logic” which is the application logic for the user from the user interface, permitting independent development, testing and maintenance of each.

Following is the detailed architecture for MVC.

![Figure 2.1: Basic MVC architecture](image)

In figure 2.1, the solid line represents a direct association; the dashed line shows an indirect association such as via an observer.

MVC architecture makes it possible to tease these components apart and makes the code easier to be re-used and independently tested. It is a very good architecture
for developing our project. As a result, we choose MVC as our project’s architecture.

2.2 Spring + Struts2 framework

The Spring Framework is an open-source application framework for the Java platform. Using it will likely make an application easier to configure and customize. The Spring container provides a consistent mechanism to configure applications and integrates with almost all Java environments, from small-scale applications to large enterprise applications.

Struts2 is an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt a model-view-controller (MVC) architecture. It provides the controller (a servlet known as ActionServlet) and facilitates the writing of templates for the view or presentation layer (typically in JSP, but XML/XSLT and Velocity are also supported). The web application programmer is responsible for writing the model code, and for creating a central configuration file struts-config.xml that binds together model, view and controller. Struts2 supports internationalization by web forms, and includes a template mechanism called "Tiles" that (for instance) allows the presentation layer to be composed from independent header, footer, and content components.

As our project is a web application program, we choose Spring plus Struts2 framework. It not only inherits MVC architecture perfectly but also make writing web application program easier.

2.3 Nutch

Nutch is an effort to build an open source search engine based on Lucene Java for the search and index component. Nutch is coded entirely in the Java programming language, but data is written in language-independent formats. It has a highly modular architecture, allowing developers to create plug-ins for media-type parsing, data retrieval, querying and clustering. The fetcher ("robot" or "web crawler") [1] has been written from scratch specifically for this project.

Nutch is suitable for what we looked for. We use JAVA to do this project. Nutch is coded in JAVA as well. So it is easy to integrate it into our project. Nutch has a scalable and relatively web crawler so that it can get abundant and suitable data from other websites. It obeys robots.txt rules. It also can be run on a cluster of 100 machines. Search is a slowly work, so this feature makes searching become faster. The most important thing for us to choose this search engine is its data quality. We can bias the crawling to fetch the important web page first. That offers a good base for intelligent searching.
Figure 2.2 shows the official website of Nutch where Nutch binary zipped file and source file can be freely downloaded.

2.4 NLTK
NLTK is short for Natural Language Toolkit. It is a suite of libraries and programs for symbolic and statistical natural language processing for the Python programming language. NLTK includes graphical demonstrations and sample data. It is accompanied by extensive documentation, including a book that explains the underlying concepts behind the language processing tasks supported by the toolkit.

As semantic analysis is based on natural language processing, this toolkit gives us a good chance to study semantic analysis well. It has implemented a lot of tools which can be used during different steps of natural language processing, and provides various APIs that can be extended by programmers. Moreover, due to its open source, it makes us easier to learn its code and integrate the engine into our project. It is really a wonderful tool.
Figure 2.3 shows the official website of NLTK where NLTK binary zipped file and source file are easily accessible.

2.5 Similar Semantic Search Engine

Although the semantic search engine on the market is not that famous as the Google or Yahoo! search engine, there exist some similar products that are quite well implemented. A quite accurate semantic search engine named Wolfram Alpha is used in our case as the reference.

Wolfram Alpha is mainly based on answering question and developed by a research group. The online service answers factual queries directly by computing the answer from structured data, rather than providing a list of web pages. Since the search engine is quite mutual, which can process the easy question the same as what we will do in our project, and the results are quite accurate.

So, in this case, we are not willing to simulate Wolfram Alpha functions, but based on the theory we studied, create a new semantic search engine with the theories we think of. Finally, we may use Wolfram Alpha to prove our theories are correct.
Figure 2.4 shows the index page of Wolfram Alpha. It is quite intelligent because most easy questions can be recognized and answered in a correct way, and also is useful during our test.
3 Natural Language Processing

The foundation of the semantic search is to understand the natural language. In this chapter, we will discuss the concept of NLP, basic theory of NLP and introduce an open source NLP engine.

3.1 Introduction to NLP

Natural Language Processing is a promising field in computer science, which aims to make machines try to understand the language human use in daily life. This section will give a brief introduction on the concept, the difference, and the history of NLP.

3.1.1 What is NLP?

The term NLP, known as Natural Language Processing, often describes the function of computers, which can perform intelligent tasks like human-machine communication and web-based question answering. Because of the difference between the physical structure of computers and the nervous system of humans, a medium ‘translation’ process is required to make computers understand the language that humans use like English or Swedish, and, this process is implemented through NLP system. Thus, strictly speaking, NLP is associated with the goal of enabling a computer system to genuinely comprehend natural language as a human being does.

NLP is a branch of artificial intelligence, which is playing an increasingly important role in computer science. More productions will be created embedded with NLP system to perform better interactions with humans. Our search engine is such an application that can execute semantic analysis by an NLP engine.

3.1.2 Comparison with other data processing

What distinguishes language processing applications from other data processing systems is their use of knowledge of language. Considering the class StringTokenizer in Java, it simply tokenizes a string of characters into several words according to the separator. On the opposite, the language processing applications can analyze the meaning of the string and the relation to others besides tokenization. However, this indicates the natural language processing needs extra pre-defined data like dictionaries and grammars to perform more intelligently, or it will not work well.

3.1.3 Core nature of NLP

In order to realize the goal of natural language processing, components of natural language including vocabulary, grammar and real meaning are required. With these tools and data, the computer may achieve the same ability as humans. As long as the computer can recognize each word’s part-of-speech, figure out the structure of the sentence in the light of an inner-built grammar regulation library, and understand the relation among all the phrases, the analysis result will be similar to the one understood by humans. This is because the procedure of analyzing strings that the computer does is exactly what human does while communication.

Unfortunately, the steps of processing language are not that easy for computers.
One big challenge during tagging the part-of-speech of each word, for example, is the word sense disambiguation. Because of the flexible polysemy of the word like ‘fish’, it involves context comprehension besides looking up in the dictionary to decide whether the word ‘fish’ is a noun or a verb.

3.1.4 Progress of NLP development
The earliest two foundational paradigms appearing in the 1950s were the automation model and the probabilistic algorithm, which led to the first machine speech recognizers in the early 1950s. In the early 1960s, speech and language processing had split clearly into two paradigms: the symbolic paradigm and the stochastic paradigm. During the period from 1970 to 1983, the stochastic paradigm became extremely important in the development of speech recognition algorithms. Then, the logic-based paradigm and the natural language understanding paradigm were created and unified on systems that used predicate logic as a semantic representation while the discourse modeling paradigm focused on four key areas in discourse. In the 1990s and early 2000s, some probabilistic methods and other such data-driven approaches spread from speech into part-of-speech tagging, parsing and attachment ambiguities and semantics. In the 2010s, the acceleration of research and development began driven by three trends. Large amounts of spoken and written material become widely available. A more serious interplay with the statistical machine was focused on, and techniques like the vector machines, multinomial logistic regression and graphical Bayesian model were supported. [2]

3.2 Natural Language Analysis Mechanism
To translate a raw document into a final machine-understand language is a long and complicated process. Basically, there should be some tokenizers and parsers to recognize the single words and their part-of-speeches or features. In a higher level, grammar analyzers are used to analyze the structure of sentences before the relations or meanings can be inferred based on the previous work. This section will mainly discuss the theory of key parts during natural language processing.

In order to make it easier but still typical, we specify the natural language we mainly discuss is English and ignore the character encoding mechanism in different machines here.

3.2.1 Sentence segmentation and word tokenization
While reading any passages, humans regard the word as the basic element and sentences are just the result of millions of permutations. Only when each word has been detected, further work like part-of-speech tagging can proceed. Before tokenizing the words, sentences are kinds of media that consists of different words and composes a whole passage. Thus sentence segmentation and word tokenization is simply a preliminary job before any further analysis.

In most common languages, there is always an obvious delimiter between every two sentences. The delimiter in this case is known as punctuation. Although punctuations in various languages are quite different, it is still easy to detect as long as
there is a library including all delimiters of a certain language. In English, such
punctuations as period, question mark and exclamatory mark can be regarded as
delimiters and a regular expression can be easily created to separate sentences. One
problem is that the word like ‘Mr.’ or other abbreviations followed by a full stop may
be regarded as the last word of the sentence due to the misunderstood delimiter full
stop here. We can predefine some special case in the regular expression to avoid such
mistakes.

The process of word tokenization is similar to that of sentence segmentation
except the delimiter changes to the space. However, it will not work if tokenizing the
Chinese because there is no space between every two Chinese words. The common
algorithm to tokenize such words is known as Viterbi Algorithm [3] based on Hidden
Markov Model [4], but we will not discuss it here.

3.2.2 Stemmer
In linguistic parlance, the stemmer is really a morphological analyzer associating
words of the same term with a root form. Because most of words stored in the
dictionary are in the root form, it is necessary to check every word from the source
text to confirm it can be recognized by the analyzer later.

The stemmer or morphological analyzer contains two types including Inflectional
Morphology and Derivational Morphology [5]. The first one expresses the syntactic
relations between words of the same part-of-speech (e.g. ‘analyze’ & ‘analyzes’) while the second one is on the opposite (e.g. ‘analyze’ & ‘analysis’).

The stemmer needs a wide range of linguistic rules and lexicons, but a better
algorithm can make it more effective. One of the algorithms named Porter Stemmer
Algorithm performs quite well and is introduced into NLTK, an open source NLP
engine that is used in our project.

3.2.3 Part-of-speech tagging
The part-of-speech tagger can mark each word in the text with a tag like noun, verb,
adjective, etc. only after the previous work has done. It is so important that it can the
foundation of the grammar analysis or relation detection.

There are various ways to realize the tagging, but basically, two kinds of taggers
are often used: the rule-based tagger and the stochastic tagger. Literally, the first one
is based on some predefined rules. For example, all the words ending with ‘ing’ can
be tagged as the ‘VPC’ and ending with ‘ed’ can be tagged as the ‘VSP’. Here, the
easiest way to implement the rule-based tagger is surely to depend on the regular
expression mapping to each tag, as the following:

\[
\begin{align*}
\text{r'\.*ing}$ & \rightarrow \text{‘VPC’} \quad \text{// present continuous tense} \\
\text{r'\.*ed}$ & \rightarrow \text{‘VSP’} \quad \text{// simple past tense}
\end{align*}
\]

However, it is a tough job to create a set of perfect regular expressions, which can
recognize all the words in English because there are always some exceptions like ‘bed’
is purely a noun. A better solution will be the stochastic tagger, shown in the
following section.

3.2.3.1 N-gram Model

N-gram model is a kind of probabilistic model which predicts the next word in a sentence from the previous N-1 words. So if N=1, it is called Unigram and called Bigram and Trigram if N=2 and 3. For example, given the sentence starts with ‘I want to eat Swedish …’, most of ordinary people will quickly associate the next word after ‘Swedish’ with ‘meal’, ‘food’ and so on, but definitely not the words like ‘book’, ‘car’ or even verbs. Because when people see the verb ‘eat’ and adjective ‘Swedish’, they will naturally link it to the food related words, instead of others. This is a typical Trigram model, for (3-1) words are referenced.

Normally, in the N-gram language model, each word probabilistically depends on the N-1 preceding words. On the other hand, if a sentence T consists of a sequence of words, \( W_1, W_2, W_3 \ldots W_{N-1}, W_N \), then the probability of T existence is:

\[
P(T) = P(W_1W_2W_3\ldots W_{N-1}W_N) = P(W_1)P(W_2|W_1)P(W_3|W_1W_2) \ldots P(W_N|W_1W_2W_3\ldots W_{N-1})
\]

\[
= \prod_{i=1}^{N} P(W_i|W_1W_2W_3\ldots W_{i-1}) \quad [6]
\]

But, in order to calculate \( P(W_N|W_1W_2W_3\ldots W_{N-1}) \), the Maximum Likelihood Estimate or MLE should be applied:

\[
P(W_N|W_1W_2W_3\ldots W_{N-1}) = \frac{C(W_1W_2W_3\ldots W_N)}{C(W_1W_2W_3\ldots W_{N-1})}
\]

The rest of the work becomes simple as we just count these words in the corpus and compute the probability of the sentence.

Also, according to Markov Assumption that the probability of a word depends only on the previous word, the probability of a sentence T in Bigram Model will be:

\[
P(T) \approx P(W_1)P(W_2|W_1)P(W_3|W_2) \ldots P(W_N|W_{N-1})
\]

\[
P(W_N|W_{N-1}) = \frac{C(W_{N-1}W_N)}{W_{N-1}}
\]

In most cases, Bigram and Trigram are used widely and the corpus to be trained in this model will be too huge if N larger than 3.

Referring to the above sentence ‘I want to eat Swedish …’, if there is corpus that only contains these words and the data has been trained, the frequency of each word is shown as the following:

\[
C(I) = 3225; C(want) = 1243; C(to) = 3045; C(eat) = 922;
C(Swedish)=211; C(food)=1512; C(lunch) = 522;
\]
And the frequency of every possible combination of two words is also computed:

\[ C(\text{want}|I) = 1054; C(\text{to}|\text{want}) = 845; C(\text{eat}|\text{to}) = 884; \]
\[ C(\text{Swedish}|\text{eat}) = 23; C(\text{food}|\text{Swedish}) = 110; C(\text{dinner}|\text{Swedish}) = 2; \]

Thus, we can compute the probability of the sentence \( T=\text{I want to eat Swedish food…} \) like this:

\[
\begin{align*}
P(\text{I want to eat Swedish food}) &= P(I) \times P(\text{want}|I) \times P(\text{to}|\text{want}) \times P(\text{eat}|\text{to}) \times P(\text{Swedish}|\text{eat}) \times P(\text{food}|\text{Swedish}) \\
&= \left(\frac{3225}{10680}\right) \times \left(\frac{1054}{1243}\right) \times \left(\frac{845}{3045}\right) \times \left(\frac{23}{922}\right) \times \left(\frac{110}{211}\right) \\
&= 0.00009241
\end{align*}
\]

Obviously, the probability in this combination is the largest, so the word ‘food’ is more likely to follow the word ‘Swedish’.

### 3.2.3.2 Training data and test data
Before computing the frequency of each word and its related phrases, some existing data will be used as reference during the future word prediction. As shown above, the probability can determine the prediction accuracy while the amount of training data in the corpus can determine the probability. The larger is the training data, the more accurate will the possible word be.

### 3.2.3.3 N-Gram part-of-speech tagging
Although the N-gram model has marvelous function to predict the missing word, a fully informative corpus is required and trained into a training set. If it is applied to the part-of-speech tagging, the word is substituted with the tag and the analysis mechanism is quite the same. So a tagger can be created based on the N-gram model.

To train the corpus, a unigram tagger may work well which involves inspecting the tag of each word and storing the most likely tag for any words in a dictionary that is already stored inside the tagger. Similarly, the N-gram tagger is a generalization of a unigram tagger whose context is the current word together with the part-of-speech tags of the N-1 preceding tokens.

However, the problem could appear if N becomes larger that the specificity of the context increases, along with the increasing chance that the test data we predict to tag contains contexts that were not present in the training data. As a result, there must be a balance between the accuracy and the coverage of the results.

### 3.2.4 Text Classification
Similar to the previous section, to classify a phrase as a specified category can be implemented by a set of rules. According to the stochastic method, we can create a training data to decide what features they have if relevant. Another test data can be processed to classify into different groups. Before discussing about the detail how to realize such method of text classification, we have to refer to some basic theory.
3.2.4.1 Bayes’ Rule
In order to identify each term in a document, we have to build a statistical model of the categories which will be assigned first. Within these so-called training data, probability of a term occurrence given a category can be estimated directly from the data. But what we need to do is to compute and transform the probability of a term occurrence given a category into the probability of a category given a term occurrence.

For example, if there is a term \( t \) in a document specified by a category \( C_i \), the probability of the \( t \) occurrence given \( C_i \) is:

\[
P(t| C_i)
\]

But we are only interested in the probability of the \( C_i \) given term \( t \):

\[
P(C_i|t) \text{ or } P(C_i|T_D) \text{ where } T_D \text{ is the set of terms in document } D.
\]

By using Bayes’ Rule, we can get:

\[
P(C_i|T_D) = \frac{P(D|C_i)P(C_i)}{P(D)} \quad [7]
\]

3.2.4.2 Naïve Bayes classifiers
The term ‘Naïve Bayes’ refers to a statistical approach to language modeling that uses Bayes’ Rule above but simply assumes conditional independence between features. In most common form of Naïve Bayes, the probability that a document belongs to a given category is a function of the observed frequency with which terms occurring in that document also occur in other documents known to be members of that class.

Ignoring conditional dependencies between terms, by using the multiplication rule, we can combine such probabilities, given a document \( D \), which is a vector with \( n \) terms:

\[
D = (t_1, t_2 \ldots t_n),
\]

\[
P(D|C_i) = \prod_{j=1}^{n} P(t_j|C_i)
\]

\[
P(C_i|D) = P(C_i) \prod_{j=1}^{n} P(t_j|C_i) \quad [8]
\]

Suppose we have no information regarding in the terms in a document, and have to make a blind guess as to where it should be classified. The most common way to estimate the prior for a given category is simply to count the number of training data which occurs in that category. As a result, Naïve Bayes can be modeled in several different ways including normal, lognormal, gamma and Poisson density functions.
For a Naïve Bayes classifier, every feature will determine which category should be assigned to a given input value. In order to select a category for the input value, the Naïve Bayes classifier begins by calculating the prior probability of each label, which is determined by checking the frequency of each category in the training data.

On the other hand, Naïve Bayes classifier chooses the most likely category for an input, under the assumption that every input value is generated by first choosing a category for that input value, and then generating each feature, entirely independent of every other feature. This simplifying assumption is known as the Naïve Bayes Assumption and makes it easier to combine the contributions of the different features.

### 3.2.5 Chunking and name entity recognition

To reach a higher level of NLP, we need to extract some meanings of the document and refine some characteristic key words from the meaning. The key words may compose the entity in the context and the process of figuring out the entity, known as entity recognition, is what we need to discuss in this section.

#### 3.2.5.1 NP chunking

The task of noun phrase chunking or NP chunking aim at searching for entity or, in this case, called chunk which corresponds to individual noun phrases. One easy way to recognize each noun phrase is to build a huge database that contains all the combination of nouns. This is too big a project that is either hard to realize or will not bring a high efficiency of looking up words. One feasible method is to recognize the noun phrases through part-of-speech tagging.

In order to create an NP chunker, we have to first define a chunk grammar, consisting of rules that indicate how sentences should be chunked. For example, regular expression is used here to parse the sentence ‘Mary cut her long golden hair’:

```plaintext
// chunk determiner/possessive, adjectives and nouns
Grammar = r'NP:{<DT|PP|$>?<JJ>*<NN>}
// chunk sequence of proper nouns
{<NNP>+}'
```

Then, two phrases can be identified as noun phrases: ‘Mary’ and ‘her long golden hair’. We can also show this sentence analysis process in the tree form:
However, using regular expression to create a chunker cannot always be correct, because there may be two sentences with exactly same part-of-speech tags, but with different combination of noun phrases. In this case, we can create a classifier-based chunker. As mentioned in the previous section, since the classifier needs a training data, the classifier-based chunker can only work within a prepared training set. Considering these two sentences:

Joey/NN sold/VBD the/DT farmer/NN rice/NN ./.
Nick/NN broke/VBD my/DT computer/NN monitor/NN ./.

Obviously, they have the same part-of-speech tags, but in the first one, ‘farmer’ and ‘rice’ are separate chunks while in the second one, ‘computer monitor’ is a single chunk. A training data should be created which contains these words and their probabilities in different combination. After computing all the probabilities of the word after the first noun, the word with largest probability can then be recognized whether it is a single chunk or not.

3.2.5.2 Name entity recognition and relation extraction
The NER or name entity recognition is such a process that the expressions representing people, places, companies, etc. in a document can be recognized. Thus the process can not only require the computer to identify the boundaries of a phrase, but also classify the phrase. For example, the word ‘Washington’ alone can both be regarded as a person and a city, but it must be different in a document related to the context.

One big problem is that many name entities are proper nouns, so a database including all the proper nouns may be required so as to easily recognize the name entities or even their relation. And there do exist many organizations focusing on the research of NER, like TAC/KBP.

An alternate approach to NER is to write a program which can automatically learn the name entities. A powerful technology called \textit{HMM} or Hidden Markov
Models is used to extract proper names from the document. Since we do not use this model in our implementation, it will not be discussed in detail in this section. Another similar way to do the name entity recognition will be mentioned in the later chapter.

As far as the relation extraction, it can be performed using either rule-based systems, which typically look for specific patterns in the text that connect entities and the intervening words, or using machine-learning systems, which typically attempt to learn such patterns automatically from a training corpus.

### 3.2.6 Context-Free Grammars

In English, a sentence consists of several words and these words can also be used to create another sentence. All these combination rules are based on the grammar and only with a unified grammar can communication happen. Thus, an integral NLP can do nothing without an English grammar. The most widely used mathematical system for modeling constituent structure in English or other natural languages is the CFG or Context-Free Grammar.

A context-free grammar $G$ is defined by a 4-tuple with 4 parameters $(N, \Sigma, R, S)$, where $N$ is a set of non-terminal symbols, $\Sigma$ is a set of terminals, $R$ is a set of rules each of the form $A \to \beta$, in which $A$ is a nonterminal, and $S$ is a start symbol. [9] Whatever the natural language it is, it must be defined via the concept of derivation. So the formal definition of the language $L_G$ generated by a grammar $G$ as the set of strings composed of terminal symbols which can be derived from the designated start symbol $S$.

A simple CFG is like the following

- $S \to NP \ VP$ // a sentence consists of a noun phrase and a verb phrase
- $NP \to DET \ N$
- $VP \to V \ NP$
- $DET \to \text{“the”} \mid \text{“a”}$
- $N \to \text{“dog”} \mid \text{“man”}$
- $V \to saw$

If there is a sentence ‘The dog saw a man’ is parsed by using this simple grammar, it can be recognized easily, especially in the tree format.
Because the CFG involves derivation, a special is needed to process the input sentences according to the rules of a grammar. There are two parsing algorithms. The top-down method is called recursive descent parsing while the bottom-up method is called shift-reduce parsing. The first one uses a grammar to predict what the input will be before inspecting the input. If the input is available to the parser all along, the second one will be more sensible to consider the input sentence from the very beginning. Since this has involved the knowledge of compilation, we will not discuss any further.

### 3.2.7 Feature-based Grammar

The grammar is generated based on context-free grammar, with the additional feature structure. The simple context-free grammar can be written as:

\[
S \rightarrow NP \ VP \\
NP \rightarrow \text{DET} \ N \\
VP \rightarrow \text{V} \\
\text{DET} \rightarrow \text{‘this’} \\
N \rightarrow \text{‘dog’} \\
V \rightarrow \text{‘runs’}
\]

This grammar can correctly match the sentence ‘this dog runs’, however, if the dog is in plural form and the sentence now to be ‘these dogs run’, it does not work well. The solution is to add a property to the tags, like this:

![Grammar tree of sentence ‘the dog saw a man’](image)

*Figure 3.2: Grammar tree of sentence ‘the dog saw a man’*
Now it can both match the single form of dog and plural form of dogs, the grammar tree is as following:

```
S → NP[NUM=?n] VP[NUM=?n]
VP[NUM=?n] → V[NUM=?n]
DET[NUM=sg] → ‘this’
DET[NUM=pl] → ‘these’
N[NUM=sg] → ‘dog’
N[NUM=pl] → ‘dogs’
V[NUM=sg] → ‘runs’
V[NUM=pl] → ‘run’
```

The grammar tree is as following:

```
S
 /   \
|     |
NP[NUM=sg] VP[NUM=sg]
     /
   /
DET[NUM=sg] N[NUM=sg] V[NUM=sg]
     |     |     |
   this dog runs
```

*Figure 3.3: Grammar tree of sentence ‘this dog runs’*

```
S
 /   \
|     |
NP[NUM=pl] VP[NUM=pl]
     /
   /
DET[NUM=pl] N[NUM=pl] V[NUM=pl]
     |     |     |
   these dogs run
```

*Figure 3.4: Grammar tree of sentence ‘these dogs run’*

The grammar with this form is the feature-based grammar, and more flexible grammar can be created to match more natural language format. And this grammar can be well utilized in some field which will be exemplified later.
3.3 NLTK – an Open Source Package

It is very hard to create a NLP engine by ourselves, because it requires a wide command of knowledge like probability and compilation. Thus, the best solution will be looking for an open source engine and here it is, NLTK or Natural Language Toolkit. The engine is written in Python, and has four prominent features, simplicity, consistency, extensibility and modularity. It not only provides a set of rich APIs, but also integrates large test data including many corpora. Thus, it is enough for us to create semantic search engine together with a search engine (discussed in next chapter) based on the above theory.

3.3.1 A set of tools

The process of translation from natural language to what a computer can understand needs a set of powerful tools to execute the steps like segmentation, tagging, classification, parsing and extraction. And there are always many serious theory supports behind each step, so not only one tool does the NLTK contain in every step, but several feasible solutions. Since not all the tools are suitable for our project, we simply make use of some as long as coming into effect.

In the segmentation and tokenization step, NLTK provides several word tokenizers including a default one named word_tokenize and advanced ones like TreebankWordTokenizer. Whichever the tokenizer is, the implementations are all based on regular expressions with an additional data. Furthermore, the data that supports the exception of word tokenization is a semantically oriented dictionary of English called WordNet. It is quite similar to a traditional thesaurus except its richer structure. What we can leverage the embedded WordNet tool, is to look for the synonyms, hyponyms or meronyms and holonyms to expand the key words in searching.

In the tagging step, there is a well-realized tagger, namely, the N-gram tagger, in the NLTK. Although this is not the only tagger that can be used in the package, we just concentrate on this tagger because it does perform well during the test. To create the training set, the related Unigram tagger can be used. Besides, Bigram tagger and Trigram tagger are also provided by the package. There is also an evaluation system, which is for the matching accuracy of the test set in the light of the training set.

In the classification step, the famous Naïve Bayes classifier is implemented by the NLTK. The system will calculate label likelihoods with Naïve Bayes, which begins by calculating the prior probability of each label, based on how frequently each label occurs in the training data. Every feature then contributes to the likelihood estimate for each label, by multiplying it by the probability that input values with that label will have that feature. The resulting likelihood score can be thought of as an estimate of the probability that a randomly selected value from the training set would have both the given label and the set of features, assuming that the feature probabilities are all independent.

In the information extraction step, despite the difficulty of name entity recognition, NLTK provides a sample gazetteer or geographical dictionary, such as the Alexandria Gazetteer and Getty Gazetteer, used to recognize the proper nouns of
places. It also provides a classifier that has already been trained to recognize the name entities named ne_chunk.

However, when it comes to a higher level of natural language analysis, tools are not that easy to realize. Luckily, NLTK still provides several context-free grammar parsers like recursive descent parser and shift reduce parser. Although the grammar itself is not complete, we can create the test context-free grammar by ourselves by referencing the sample ones.

There are still some other useful tools made by NLTK, but all the tools above-mentioned are enough for our semantic analysis.

3.3.2 Various methods to implement semantic search
The implementation of semantic analysis is mainly based on the API provided by NLTK package. And since the analysis engine serves the search engine, so some methods will cover the basic knowledge of the search engine.

3.3.2.1 Fuzzy Search
The basic principle behind normal search engine is to match the key words that user inputs with the text stored in the database. If there are some words in the webpage equal to the input string, this page might be the search result. However this cannot always be accurate, for there are too many synonyms in English along with different tense, plurals, etc. The fuzzy search here is simply a way to expand the range of the key words.

In natural languages, the hypernym refers to a word or a phrase with general meaning and able to summarize some other words. On the opposite, the hyponym is word or phrase whose semantic field is included within that of another word. For example, if the original key word is ‘tree’, and the searched document only contains the words like trunk, leaves, stem and so on. We, human beings, can easily infer from the common sense that this document largely is a passage about trees in spite of lack of the word ‘tree’.

We can use the WordNet as mentioned above to look up the hypernyms, hyponyms and synonyms of a certain word extracted from the user-input string. In most cases, if the input string is not that long and basically with two or three words, these words might be nouns or verb phrases which also contain nouns. Under this situation, we can add hyponyms and synonyms to the key words to expand the range of the search result.

3.3.2.2 Classification Search
The only sources of text that need to be analyzed are input string and html files or webpages. To be frankly, a relation should be found in the texts from two different sources, which can be used by the search engine to easily find the related results. However, it is very hard to automatically detect if there is a relation between two sentences or passages. What we can utilize from the tools of NLTK is the classifier. As long as the analyzer can understand the meaning of input string, it can classify the string into a certain label or category and this turn out to be the same work as text
classification.

Obviously, a training set must be prepared preliminarily, so that a Naïve Bayes classifier can figure out the possible category the input string belongs to. Considering the source data of the training set, all the web pages stored in the database should be very suitable.

The ideal result of this analysis is that a quite correct category can be confirmed and the search results in top ranks are all about the category and relative topics. Thus, whether this method can lead to accuracy largely depends on the scale and dimension of training set. More words of the input string can be matched in the training set; it is more likely that the probability of relevance approaches an ideal value.

### 3.3.2.3 Information Extraction

According to the research by iPropect [10] on the habit of humans when using the search engine, over 80% of users using search engines will not look the third page of search results and the study confirms the importance of the first three pages of search results. This indicates that users definitely want the most similar results compared to others, regardless of the amount the results, even if there are millions of matched hits.

The solution to this situation is that we can present the useful information alone instead of the regular search results. This results in another big problem that how the search knows which part of the documents is most closely to the input string. On the other hand, this equals to how to extract the sentences conforming to the meaning of the input string.

The easiest way is to recognize the name entities if the string to search is about a noun phrase. We can alternately search a set of features of a specified name entity, instead of the key word itself. Moreover, since the name entities have been found, it will be quite easier to look for the relations among these name entities. A triple form \((X, \alpha, Y)\), where \(X\) and \(Y\) are name entities and \(\alpha\) is a string of words connected to \(X\) and \(Y\), can be applied to mark the relation. [11] With the help of part-of-speech tags and regular expression, the particular form of the combination of \(X\) and \(Y\) can be found and sentences or paragraphs that contain such relations might be the useful information of the input string.

### 3.3.2.4 Simple question and answer

Dialogs in human daily life are quite common, which must contain questions and answering. When human want to answering a question, he should retrieve the information he has stored and form into a structured data and present it to others. This process can also be simulated by computers as long as there is a database full of data. The easiest way to get the answer is to query the database with SQL.

Applying with this thought, we can translate the questions into SQL by feature based grammar. The grammar let us to assemble a meaning representation for a sentence in tandem with parsing the question. Each phrase structure rule is supplemented with a recipe for constructing a value for the feature. After generating the SQL string, we can use to find the answer in the databases. However, this method
has a lot of defect. For example, the construction of database is very hard and the
table name in the SQL string may not equal to the real table name in the database
(both of them are synonyms).

Our solution of question and answer is simply a basic method, but for further
studying, many complex theories will be required which are beyond our knowledge.
4 Search Engine

In order to integrate the NLTK into the search engine, we choose an open source search engine package named Nutch instead of Google APIs. This chapter mainly focuses on the architecture and components of the Nutch and disadvantages of key-word based search.

4.1 Nutch Architecture

Nutch is a complete open source search engine implemented in Java, together with three components, the crawler, the indexed database managed by Lucene and the searcher. The reason why we run our own search engine is due to the Nutch characteristics like transparency, which we can integrate code into it, especially the semantic analysis engine.

![Diagram of Nutch Architecture](image)

*Figure 4.1: Complete workflow of Nutch from crawling to indexing and searching*

The crawler system is driven by the Nutch crawl tool, which can automatically grab the information from the Internet. Such five steps are needed during the process of crawling as inject URLs, generate segments, fetch content, parse content and update URLs. The recursive circulation of these steps makes the data grow bigger and bigger in exponential increase.

The index system embedded with Lucene changes the unstructured data like html or pdf from the Internet into the texts through parsing and analyzing and stores them
into the inverted index database. The Lucene inverted index mechanism improves the efficiency of searching key words by matching the attributes of each record instead of matching the record to get attributes. On the other hand, the inverted index equals to answering the question like ‘Which documents contain the key words?’ rather than ‘What words are there in this document?’.

![Figure 4.2: File structure of Nutch index database](image)

The searcher system in Nutch is easily implemented and the process only includes matching the input key words with the indexed database. Although it will do some simple analysis before matching like removing the irrelevant words ‘the’ or ‘a’, lower all the words, removing punctuation, etc., the analysis process is still not enough to be regarded as a semantic analysis. Thus, this is the key part that we need to create and integrate the NLTK into it.

### 4.2 Improvements and optimization

As there are three main components in the Nutch, we can do nothing with the crawler and indexer except the searcher.

The searcher will not only get the input string, which is one source of text, and match with the index database, which is the other source of text. What we can do with the input string is to leverage the fuzzy search or text classification, which is discussed in the previous chapter. These two methods mainly focus on expanding the meaning of key words in order to get more hits during the matching step. To optimize the input string, the NLTK module will be called to process the raw text (input string) first. The analysis result will then be returned to the searcher and notify the searcher to proceed.
Due to the fact that the text analysis requires much extra time, an optimization solution can be built to relieve the delay. However, this involves the mechanism of calling different modules and has nothing to do with the improvement of searching result, so we ignore this problem just here. On the other hand, considering the taggers and classifiers in the NLTK, a training data is required to predict the test data, so it is a good habit to train the data in the index database in advance. This can bring the efficiency and accuracy in the process of analysis.

The input string must always be shorter than the database documents, so the natural language processing will not reduce the efficiency too much in this case, but the situation is totally different if the semantic analysis occurs operated on the database during the runtime. Millions of data will be re-parsed if a grammar parser is used. So it is better to maintain the index matching work done by Lucene and the parser can then analyze the results or hits, which are much smaller compared to the complete data.

To improve the accuracy of results, many other ways can also be utilized, but we simply concentrate on the semantic analysis method.
5 Integration and Implementation

Both the NLTK and Nutch provide tremendous functions, and each of them has some independent tools based on command line or GUI. Furthermore, two packages are written in different languages. In this chapter, we will show the implementation how to integrate NLTK, the natural language processing engine, and Nutch, the search engine, into the J2EE architecture and frameworks.

5.1 Technological Integration Overview

The whole architecture of the project is based on the J2EE standard.

All the MVC architecture is implemented in Java except for the semantic analyzer in Python. Because the Python in NLTK is written in C, it is impossible to run directly on the JVM. The alternative solution is to call execute file generated from Python in Java, despite of its extra delay. On the server side, Struts2 is used to framework the control level and Spring is to manage all the beans by its container.

The crawler and searcher are created in model level from Nutch APIs. During the process of user input string, execute file from Python is called directly in Java program in a sub process and get the return results until the sub process is over.

Two kinds of databases are accessed through different functions. The index database is built on the server file system by Lucene, which is for searching web pages, while the other one is a real MySQL database, which is used to do queries if
the input string is simple questions.

Figure 5.2 shows two projects, which exist in the same workspace in the IDE MyEclipse. Due to different programming language used by two projects, different perspectives can be altered during the development according to the required language.

### 5.2 J2EE and Nutch

The standard MVC framework in J2EE are often divided into five levels, known as presentation, controlling, business, data persistence and database. But in our case, the database is an exception that Lucene manages its inverted index database on the local file system. This simplifies the implementation to fetch data or web pages. The special file structure of inverted index provides a high efficiency to search the key words, for its aim is to answer the question like ‘What documents contain the key words’ and it is suitable for searching. And this is the reason why we need not build another database which stores the path of the web pages. So our architecture mainly focus on the MVC without data persistence level.

The framework Struts2 and Spring are used to make the architecture better and improve the run-time efficiency. We rewrite the crawler and the searcher. The crawler function is now embedded into the webpages, which makes it easier to update the URLs and index database. The searcher function is realized through the APIs provided by Nutch, but we still add some semantic functions with it and we will discuss it later.
5.3 J2EE and NLTK
The text analyzer NLTK is added to model level together with the searcher and crawler in Nutch. And there is a module particularly used to generate and call python executable modules. There is nothing important in this integration except the delay for executable programs. To lower the time expense, we decided to make the executable program running all the time at first, but it does not work when passing the parameters to the sub process and the sub process of the executable program will close at once as long as it has finished.

5.4 Nutch and NLTK
This is the key part of integration. We will discuss about four above-mentioned methods to implement semantic analysis in detail.

5.4.1 Integrate fuzzy search
The easiest one is the fuzzy search, which equals to expanding the key words range. Before doing the search, the searcher will call the python module to create a sub process. The sub process actually runs the executable program generated from python, and look up the key words, mostly nouns, in the embedded dictionary WordNet in the NLTK. The results will be returned in the array form, and recreated into a new expression like regular expression.

5.4.2 Integrate text classification
The second method is the classification search and the difficulty is about the training data. The source of training data can be the web pages and must be trained first and create a Naïve Bayes classifier in advance. The accuracy of the matching depends on the scale of the training data, so large amount of data need to be trained. However, this method also depends on the length of the input text. It is better if there are more features that can be easily recognized as a category, or ambiguity is the big problem. If the category that the input information belongs to is quite close, we can either search the specified web pages category or add the category name to the original sentence. Because we do not rewrite the index database APIs encapsulated by Nutch based on Lucene, it is hard to search in a specified category containing related web pages. Instead, we symbolically add a key word into the search text. This can narrow the search range, improve the hit percentage and lessen the search time.

5.4.3 Integrate text extraction
The third method is the most useful one in presenting the results, but it is hard to realize and not that efficient in our case. An alternate way to implement used by both Nutch searcher and NLTK name entity recognition and relation extraction. We can recognize the name entity in input string first. Since it costs too much work to list all the proper nouns or relative noun phrases, we simply use the classifier provided by NLTK that has been trained to recognize named entities assessed by function ne_chunk. Although it is not accurate as expected, we can just use it as test and verify the method is feasible. After the named entity has been detected, next step is to find
the relation between named entities. However, this is even harder, because we need to find all the named entities in the web pages in the same way. If the accuracy is low, the relations among the named entities will be mistaken by the program. The simplest one is the ‘is-a’ relation. Once the input string is about a proper noun or a noun phrase, the most probable answer is hidden in the paragraph which contains the ‘is-a’ relation.

The whole process of this method is quite low-efficient in our case. First, we can find the named entity in the input string through python module in NLTK and return it as the new key words. Next, the searcher in Nutch will search the key word in index database and return a group of possible URLs. Then, we will call the python module again to find the relation between named entities in the web pages linking to the given URLs. After that, the related paragraph will be fetched and written into the text files. Finally, the searcher will collect these text files containing the possible answers and transfer them to the view level.

5.4.4 Integrate question answering
Machine learning is promising field of AI, but it is too complex to realize perfectly. Our question answering is simply based on the SQL as already introduced, so it can only cover a single area, which also contains a lot of things. The grammar we use to translate natural language into SQL is the feature-based grammar. The searcher will call the python module to analyze the input question and get a returned value of SQL, and then, query the database with the generated SQL to get the easy answer. Besides, the answers will be regarded as key words and search the index database again to get the web pages links. And the result page will both show the easy answer and the web page links.
6 Test Results

This chapter will show some results of our search engine. Some comparisons will be done about the effect before and after using the semantic analysis engine.

6.1 Search result

Figure 6.1 shows the result of normal search, it only matches the key word ‘news’ with the text in the html files and present all the possible results. Although the result is not that accurate, it only costs 0.525 seconds.

![Figure 6.1: Results of normal searching](image)

Figure 6.2 show the search result together with a dictionary explanation. Although it costs 8.555 seconds, the result is more accurate than that in Figure 6.1. Meanwhile, more news websites are shown at the top of the page. This is the effect of expanding the range of key words or fuzzy search.
Figure 6.3 shows the result of information extraction on web pages. Only useful information is presented in the results list rather than titles, links or a small segment of content.
Figure 6.4 shows the simple question and answer. Because of lack of data, not many results can be found, but the system can actually understand the real meaning of the question. And the answers are just single words which can be directly linked to a new search.
6.2 Comparison with normal search engine

Compared to the normal search shown in Figure 6.1, the semantic search results are richer and more accurate. The simple example in Figure 6.2 only show the results after using synonyms and hyponyms to re-construct key words during the search, but the improvement of results is very obvious.

Also, the other example in Figure 6.3 shows an easy question and answer program. We just want to get universities in the state of Arizona in US. The answer in our project is very easy, but it does actually work. We try the same question on the Wolfram Alpha, where the results are quite complete, shown in Figure 6.4. We can find that our result is a subset of that in Wolfram Alpha, which means the difference between our project and the professional semantic search engine is the data in database. As long as we can collect the same data as that in the professional one, more accurate information will be obtained. But the complete data is the most disgusting part of the search engine or semantic analysis engine during development. Thus, we can only try a little data for test.
When it comes to other semantic analysis methods, we cannot show results by using them because they are only implemented in command line.

6.3 Existing problem
Because different programming languages are used in the project, there should be a suitable way to make them compatible. The method here is to let the Java call the executable program generated from Python. This will cost extra time to finish the procedure. Thus, a better way can be found to make up with delay while calling modules written in different languages.

The second problem is the time cost during the semantic analysis and search. Because the prototype of Nutch is operated on the distributive system, but only for test, we run it on a single computer, which may cost more time for the computer to calculate. The improvement can be using the distributive system engine like Hadoop to improve the efficiency of the process.

The third problem is to collect enough data or to fulfill a complete database. Countless information is needed to be obtained during the process of construction. It costs too much time and is not worthwhile in our project. This may lead to incompleteness of searching, but this deficit does not influence the incorrectness of the solution we figure out.
7 Conclusion and Future Work

According to the searching result of last chapter, the method to realize the semantic search is implemented and the problem we mentioned at first is well relieved. In the last chapter, we will reach a conclusion based on our research and development and propose some plans that can be done in the future.

7.1 Conclusion

The problem addressed by this thesis is how to optimize normal search engine through semantic analysis, which aims at adjusting the inflexibility of key words, understanding the meaning of natural language, and improving the accuracy of results or hit percentage. We have discussed the basic procedures and introduce some methodology of natural language processing. NLTK, the open-source natural language processing engine, Nutch, the open-source search engine, are both introduced and used to integrate into a whole J2EE project.

For NLTK engine, it provides most useful APIs to deal with natural language analysis. In low level during the process, functions like sentence segmentation, word tokenization and part-of-speech tagging work very well, because not much pre-defined data is required and they can be implemented easily based on some sophisticated algorithms. However, in high level, functions like text classification, chunking and grammar analysis are not always accurate as we want. It is because a large amount of training data should be prepared that will be used during the analysis. Even if the data is as complete as we can imagine, the accuracy or hit percentage of results can still not reach 100%, for mistakes must always exist.

For Nutch engine, it is quite complete search engine with a crawler, an index database and a searcher. The accuracy of search results is largely based on how much web pages we have crawled and stored in the database. Since it costs several weeks to crawl the whole Internet, we choose to get a small part of web sites only for test.

To solve the problem we mentioned in Chapter 1, we figure out four methods. Fuzzy search is the first and the easiest way, which expands the range of key words, re-construct them and obtain as much search results as possible. It does work well and result is shown in Figure 6.2. Second method is called text classification. However, it is too hard to perform satisfactorily because a set of well-organized training data is required, classifiers should be re-written in accord with different tags, and database must be classified into several parts in the light of classification tags as well. Although the search result of using this method in our project is quite the same as that of normal search, we still believe it can work well under certain circumstance. The third method is to extract useful information alone. It does not always work well because training data is also required. Luckily, if the content of web pages, where information will be analyzed and extracted, is similar to the training data, it performs as we imagined and Figure 6.3 shows one of the successful search results. Same as the fourth method, namely, simple question and answer, it can only work well if the database has store the information of the answer.

All in all, four methods can optimize the search results in some extent. Fuzzy
search and text classification can improve match rate, while information extraction can filter irrelevant information on web pages, and simple question and answer makes users interact with machines more naturally.

7.2 Future work

The future work mainly focuses on the optimization of current projects.

About the runtime efficiency, a distributive system can be used to improve the parallel computing during semantic analysis and key words search. Besides, the communication among different modules can be processed more loosely coupled.

About the matching accuracy, some more advanced algorithms can be applied into steps of natural language processing like text classification. This will make the machine have a better understanding of natural language.

A big problem is that many accurate analysis methods require a complete linguistic grammar list and vocabulary library, which is the same as what human beings need. This is really difficult and almost regarded as an independent branch of natural language processing.

If all the mentioned defects can be solved, we believe the semantic search engine will work better.
Bibliography


8. **Introduction to Natural Language Processing**, Mary Dee Harris.


10. **Tutorial on hmm and applications**. Lawrence R.Rabiner. [Document].


Appendix A

1. Reference

[5] – *Natural Language Processing for Online Application; Text Retrieval, Extraction and Categorization* by Peter Jackson and Isabelle Moulinier Chapter 1.
[8] – *Natural Language Processing for Online Application; Text Retrieval, Extraction and Categorization* by Peter Jackson and Isabelle Moulinier Chapter 4.

2. Index of Proper Noun

<table>
<thead>
<tr>
<th>Proper Noun</th>
<th>Explanation</th>
<th>Page for first appearing</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language is the predominant markup language for web pages. It is the basic building-blocks of webpages.</td>
<td>1</td>
</tr>
<tr>
<td>Web 2.0</td>
<td>It is associated with web applications that facilitate participatory information sharing, interoperability, user-centered design, and collaboration on the World Wide Web.</td>
<td>1</td>
</tr>
<tr>
<td>Artificial Intelligence</td>
<td>AI is the intelligence of machines and the branch of computer science that aims to create it.</td>
<td>1</td>
</tr>
<tr>
<td>API</td>
<td>An application programming interface is a particular set of rules and specifications that software programs can follow to communicate with each other.</td>
<td>2</td>
</tr>
<tr>
<td>J2EE</td>
<td>It is a widely used platform for server programming in the Java programming language.</td>
<td>2</td>
</tr>
<tr>
<td>MVC</td>
<td>It is a software architecture, currently</td>
<td>2</td>
</tr>
<tr>
<td><strong>Spring</strong></td>
<td>It is an open source application framework for the Java platform.</td>
<td>2</td>
</tr>
<tr>
<td><strong>Struts2</strong></td>
<td>It is an open source application framework for the Java platform.</td>
<td>2</td>
</tr>
<tr>
<td><strong>Nutch</strong></td>
<td>It is an effort to build an open source search engine based on Lucene Java for the search and index component.</td>
<td>2</td>
</tr>
<tr>
<td><strong>NLTK</strong></td>
<td>It is a suite of libraries and programs for symbolic and statistical natural language processing.</td>
<td>2</td>
</tr>
<tr>
<td><strong>JDBC</strong></td>
<td>Java DataBase Connectivity refers to an API for the Java programming language that defines how a client may access a database.</td>
<td>3</td>
</tr>
<tr>
<td><strong>CORBA</strong></td>
<td>Common Object Request Broker Architecture is a standard defined by the Object Management Group that enables software components written in multiple computer languages and running on multiple computers to work together.</td>
<td>3</td>
</tr>
<tr>
<td><strong>XML/XSLT</strong></td>
<td>Extensible Markup Language is a set of rules for encoding documents in machine-readable form. Extensible Style sheet Language Transformations is a declarative, XML-based language used for the transformation for XML documents.</td>
<td>4</td>
</tr>
<tr>
<td><strong>Lucene</strong></td>
<td>Lucene is a free and open source information retrieval software library, originally created in Java by Doug Cutting. It is supported by the Apache Software Foundation and is released under the Apache Software License.</td>
<td>4</td>
</tr>
<tr>
<td><strong>Python</strong></td>
<td>Python is an interpreted, general-purpose high-level programming language whose design philosophy emphasizes code readability. Python aims to combine &quot;remarkable power with very clear syntax&quot;, and its standard library is large and comprehensive. Its use of indentation for block delimiters is unique among popular programming languages.</td>
<td>5</td>
</tr>
<tr>
<td><strong>NLP</strong></td>
<td>Natural Language Processing is a field of computer science and linguistics concerned with the interactions between computers and</td>
<td>8</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>human.</td>
<td>The automation model provides an alternative to VSPackages for extending Visual Studio. Known in earlier versions of Visual Studio as the extensibility model, the automation model is a programming interface that gives you access to the underlying routines that drive the integrated development environment (IDE), and lets you customize, adjust, and automate it.</td>
<td></td>
</tr>
<tr>
<td>automation model</td>
<td>It uses result of a random process.</td>
<td></td>
</tr>
<tr>
<td>probabilistic algorithm</td>
<td>It was the work of Chomsky and others on formal language theory and generative syntax throughout the late 1950s and early to mid-1960s.</td>
<td></td>
</tr>
<tr>
<td>symbolic paradigm</td>
<td>It took hold mainly in departments of statistics and of electrical engineering.</td>
<td></td>
</tr>
<tr>
<td>stochastic paradigm</td>
<td>It was begun by the work of Colmerauer and his colleagues on Q-systems and metamorphosis grammars.</td>
<td></td>
</tr>
<tr>
<td>logic-based paradigm</td>
<td>It took off during the period, beginning with Terry Winograd’s SHRDLU system, which simulated a robot embedded in a world of toy blocks.</td>
<td></td>
</tr>
<tr>
<td>natural language understanding paradigm</td>
<td>The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states – called the Viterbi path – that results in a sequence of observed events, especially in the context of Markov information sources, and more generally, hidden Markov models. The forward algorithm is a closely related algorithm for computing the probability of a sequence of observed events. These algorithms belong to the realm of information theory.</td>
<td></td>
</tr>
<tr>
<td>Viterbi Algorithm</td>
<td>A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. An HMM can be considered as the simplest dynamic Bayesian network.</td>
<td></td>
</tr>
<tr>
<td>Hidden Markov Model</td>
<td>Inflectional morphology manifests primarily in the form of a prefix, suffix, or vowel</td>
<td></td>
</tr>
<tr>
<td><strong>Derivational Morphology</strong></td>
<td>It changes the meaning of words by applying derivations.</td>
<td>10</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------------------</td>
<td>----</td>
</tr>
<tr>
<td><strong>Porter Stemmer Algorithm</strong></td>
<td>It is a process for removing the commoner morphological and inflexional endings from words in English.</td>
<td>10</td>
</tr>
<tr>
<td><strong>MLE</strong></td>
<td>Maximum-likelihood estimation is a method of estimating the parameters of a statistical model.</td>
<td>11</td>
</tr>
<tr>
<td><strong>Markov Assumption</strong></td>
<td>Markov property refers to the memoryless property of a stochastic process. Its namesake is the Russian mathematician Andrey Markov.</td>
<td>11</td>
</tr>
<tr>
<td><strong>Bayes' Rule</strong></td>
<td>Bayes' theorem expresses the conditional probability, or &quot;posterior probability&quot;, of a hypothesis H, in terms of the &quot;prior probability&quot; of H, the prior probability of E, and the conditional probability of E given H.</td>
<td>13</td>
</tr>
<tr>
<td><strong>prior probability</strong></td>
<td>Prior probability distribution, often called simply the prior, of an uncertain quantity p (for example, suppose p is the proportion of voters who will vote for the politician named Smith in a future election) is the probability distribution that would express one's uncertainty about p before the &quot;data&quot; (for example, an opinion poll) is taken into account. It is meant to attribute uncertainty rather than randomness to the uncertain quantity. The unknown quantity may be a parameter or latent variable.</td>
<td>13</td>
</tr>
<tr>
<td><strong>Naïve Bayes Assumption</strong></td>
<td>It assumes that the effect of a variable value on a given class is independent of the values of other variable.</td>
<td>14</td>
</tr>
<tr>
<td><strong>regular expression</strong></td>
<td>A regular expression, also referred to as regex or regexp, provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor, a program that either serves as a parser generator or examines text and identifies parts that match the provided specification.</td>
<td>14</td>
</tr>
<tr>
<td><strong>TAC/KBP</strong></td>
<td>It is a project focusing on the extraction of</td>
<td>15</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>top-down</td>
<td>A top-down approach is essentially the breaking down of a system to gain insight into its compositional sub-systems.</td>
<td>16</td>
</tr>
<tr>
<td>bottom-up</td>
<td>A bottom-up approach is the piecing together of systems to give rise to grander systems, thus making the original systems sub-systems of the emergent system.</td>
<td>16</td>
</tr>
<tr>
<td>WordNet</td>
<td>WordNet is a lexical database for the English language. It groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets. The purpose is twofold: to produce a combination of dictionary and thesaurus that is more intuitively usable, and to support automatic text analysis and artificial intelligence applications.</td>
<td>18</td>
</tr>
<tr>
<td>GUI</td>
<td>A type of user interface that allows users to interact with electronic devices with images rather than text commands.</td>
<td>25</td>
</tr>
<tr>
<td>Hadoop</td>
<td>Apache Hadoop is a software framework that supports data-intensive distributed applications under a free license.</td>
<td>33</td>
</tr>
</tbody>
</table>