8-22-2012 12:00 AM

A New Web Search Engine with Learning Hierarchy

Da Kuang
The University of Western Ontario

Supervisor
Charles X. Ling
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

A NEW WEB SEARCH ENGINE WITH LEARNING HIERARCHY
(Thesis format: Monograph)

by

Da Kuang

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
School of Graduate and Postdoctoral Studies

CERTIFICATE OF EXAMINATION

Supervisor:

.....

Dr. Charles X. Ling

Examiners:

.....

Dr. Jamie Andrews

.....

Dr. Kamran Sedig

.....

Dr. Luiz Capretz

.....

Dr. Yang Xiang

The thesis by

Da Kuang

entitled:

A NEW WEB SEARCH ENGINE WITH LEARNING HIERARCHY

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

.....

Date

.....

Chair of the Thesis Examination Board

Abstract

This thesis proposes our first attempts to build a novel web search engine, named *SEE* (Search Engine with hiErarchy), with webpages classified into categories of a topic hierarchy. We also discuss how to improve SEE with minimal human supervision.

Most of the existing web search engines (such as Google and Bing) are in the form of keyword-based search. Typically, after the user issues a query with the keywords, the search engine will return a flat list of results. When the query issued by the user is related to a topic, only the keyword matching may not accurately retrieve the whole set of webpages in that topic. On the other hand, there exists another type of search system, particularly in e-Commerce websites, where the user can search in the categories of different faceted hierarchies (e.g., product types and price ranges). Since the total amount of data is relatively small, the categorization is often conducted by human efforts or based on the pre-defined labels. Is it possible to integrate the two types of search systems and build a web search engine with a topic hierarchy? Yahoo! in fact attempted to organize the Internet webpages into a hierarchical structure by labeling them using human effort. However, it failed eventually as the number of webpages increased dramatically on the Internet. Thus, to build such integrated web search engine, the main difficulty is how to classify the vast number of webpages on the Internet into the topic hierarchy. In this thesis, we will leverage machine learning techniques to automatically classify webpages into the categories in our hierarchy, and then utilize the classification results to build the new search engine SEE.

Firstly, we extract a reasonable hierarchy from the Open Directory Project (*ODP*), and use the webpages in ODP as our base training data to build the classifier. Then we utilize the top-down hierarchical classification approach to classify new Internet webpages into our hierarchy. By exploring different feature filtering methods, classifier parameters and calibration algorithms, we optimize the performance of our hierarchical predictive model. According to the evaluation results, the classification performance is satisfactory.

How to leverage the classification results to build the search engine SEE is the next step. Here, we face two major challenges: how to deal with the false positive classification errors and rank the results within each category, and how to handle the false negative errors introduced in the classification phase. Accordingly, we design a novel ranking function and a smart way to let user explore more results. We conduct a comprehensive evaluation by using a well-known data collection for information retrieval. The results demonstrate that the hierarchical version of SEE can achieve better search results than the flat version in most of the queries, particularly when the query is related to a topic.

In order for SEE to achieve better search performance, it is critical to improve the classification performance, which often requires human efforts to label more webpages as the training

data. Thus, it is worthwhile to study the problem on how to maximize the classification performance in the hierarchical setting with minimal human supervision. We propose a novel multi-oracle setting and a new active learning framework for the hierarchical classification. According to the experiment results, our methods can largely reduce the human labeling cost, and thus can be used to further improve the performance of SEE.

In summary, we implement a prototype of a novel search engine named SEE, where the user can do both keyword search and hierarchical browsing. Several challenges have been discussed and several novel solutions have also been proposed accordingly. According to the evaluation, SEE obtains promising results. More importantly, since the classifiers are built offline, this technology is scalable to be applied to any large-scale search engine.

The thesis is joint work with Xiao Li. We both contributed in designing the architecture of the entire system for SEE as well as the algorithms on how to improve SEE. I contributed more on Chapter 3 and Chapter 4, while Xiao contributed more on Chapter 5.

Keywords: search engine, hierarchy, faceted hierarchies, e-Commerce websites, classification, ranking, information retrieval, oracle, active learning

Acknowledgements

First, I would like to thank my supervisor, Dr. Charles Ling, who had the greatest role in my thesis research in the past four years. I will always be grateful for all his guidance and support for my research in the interesting topics of this thesis. Without his instruction, it's hard to imagine that I can finish this thesis.

Thanks are also given to all the members of our Data Mining Lab at Western, specially, my partner, Xiao Li. We cooperated on the search engine SEE and jointly published two papers. He made great effort on our research. Besides, I would like to thank to Eileen Ni, who carefully read my thesis and gave a lot of suggestions. Also, thanks my former colleague, Jun Du, who kindly helped me solve lots of research problems during my 4-year PhD study.

Finally, I would like to give my deepest thanks to my wife Sizhe. She gave me unconditional encouragement and support in helping me finish my thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of the Web (World Wide Web [4]) and Internet technologies (e.g., HTTP [21], HTML [4], servers and browsers) makes users' access to information much easier than before. Rather than going to the library to find information by reading books, Internet users only need to enter the URL of a webpage in the browser and the desired information will be displayed. Consequently, the Web became an ideal platform for the Internet users to supply and retrieve information on every aspect in their life.

As the amount of scientific information and the number of electronic journals on the Internet continue to increase [35], how to discover the desired information became a challenge for the Internet users. Therefore, a centralized organization is needed to manage and organize the web content on the Internet and provide convenience for the users to find useful information. It directly leads to the advent of the web search engines [10] such as Google and Bing. Typically, most of the modern web search engines are keyword-based. Specifically, they crawl the webpages on the Internet and index the full text of each webpage. When the user issues a query containing several keywords, they compare the keywords with the indexed documents by using some similarity metrics, and return a flat list of the closest ones. Figure 1.1 demonstrates an example for the keyword-based web search engine[1].

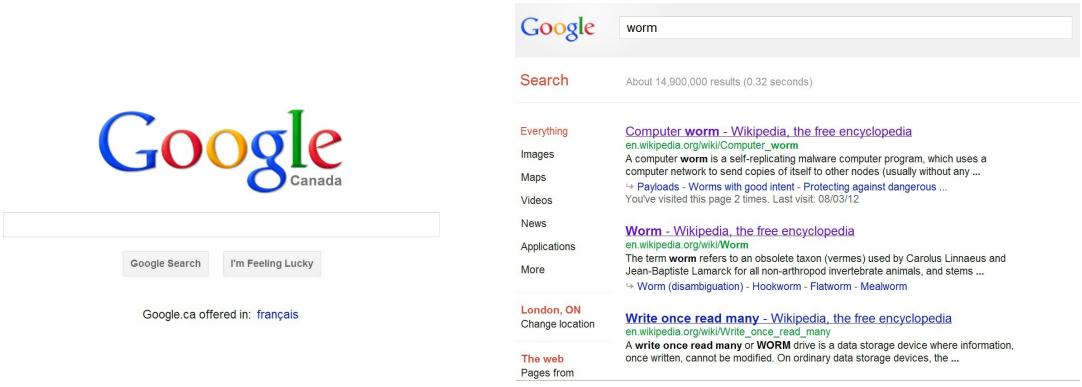

Figure 1.1: Keyword-based web search engine.

[1]Google recently introduced a few categories (e.g., images, maps, videos) as shown in the left subgraph of Figure 1.1

However, this keyword-based search engine may not work well on the *ambiguous queries* and *topic-related queries*, which are common when people search information. For example, to search worm as the animal in the nature, if we search "worm" as the keyword of the query, it is an ambiguous query, since worm can also mean the computer virus. If we restrict the search in the area of animal, we can search "animal worm". This is a topic-related query. However, the keyword "animal" may not match well the webpages that really talk about the animal worm but do not contain the keyword "animal". Suppose we have a topic category named "animal" with all animal-related webpages classified into it; we can simply search the keyword "worm" in the topic category and the results will be much better.

In fact, the concept of integration of keyword search and categorization has been proposed and applied into some other search systems, particularly for the product (or item) search in the e-Commerce websites. Those search systems are often called *Faceted Search System* [3]. For example, in Amazon and Kijiji as shown in Figure 1.2, on the left side, we have some category facets (hierarchical or flat) and we can search the keywords in the categories of the facets to narrow down the search domain. Since most of the products are pre-labeled by the users (when the products are uploaded), it is not difficult to integrate the categories into the search. However, it is difficult to implement this functionality in the general web search engine, since there exist a huge amount of webpages on the Internet and there is no predefined category label for each of the webpage. The task to manually categorize all the Internet webpages is impossible. Is it still possible to build a general web search engine integrating keyword search with categorization in topics?

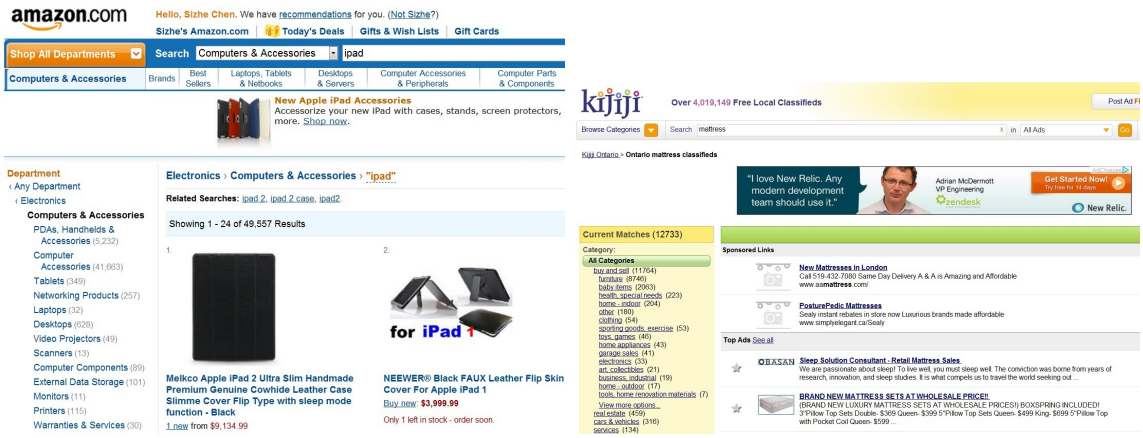

Figure 1.2: Faceted search system.

In this thesis, we utilize machine learning techniques to implement a novel web search engine, named *SEE* (Search Engine with hiErarchy), with all documents classified into the categories of a topic hierarchy. We leverage the *Open Directory Project*(*ODP*) to generate our hierarchical topics and train a hierarchical predictive model to categorize the Internet webpages into our hierarchy. Based on the results of the classification, we construct the actual search engine by indexing the prediction scores. This feature makes our search engine SEE scalable to billions of webpages. We design a new ranking function for SEE and a smart way to let the user explore additional results. According to the evaluation, SEE achieves more promising results compared to the search engine without hierarchy. We also study how to improve the

performance of SEE by using less human effort. We propose a novel active learning framework for hierarchical classification. The experimental results show that the human labeling cost can be greatly reduced by using our approach.

The rest of this chapter is organized as follows. We first review the two search systems (web search engine and hierarchical faceted search system) (Section 1.1). We then present the motivation for this new search engine (Section 1.2.1). We briefly discuss the difficulties in building SEE (Section 1.2.2). We also demonstrate the overall architecture and the user interface of SEE (Section 1.2.3). Finally, we list our major contributions in this thesis (Section 1.3).

1.1 Two Types of Search Systems

In this section, we will provide some preliminaries on the keyword-based web search engine as well as the faceted search system, so that people can have an intuition and basic idea on how roughly the keyword-based web search engines (such as Google and Bing) and the hierarchical faceted search engine work.

1.1.1 Keyword-based Web Search Engine

On the user interface of most keyword-based web search engines, the key component is a text box where the user can enter any combinations of keywords as the query. As the "search" button is clicked, a HTTP request containing the query is sent to the server. Then, the server processes the query, searches the indexed documents and returns a list of most relevant results back to the user. The most challenging problem for the search engine is how to retrieve the relevant ones from billions of indexed documents in a millisecond timescale based on the keywords. Webpages are usually semi-structured (e.g., most of HTML pages have title and body) or even unstructured data, which makes the searching more difficult, compared to searching the structured data such as the relational database. We will briefly introduce two typical models to retrieve documents based on keywords. The first model is called *boolean retrieval model* [34, 68]. In this model, keywords in the query are in combinations with logic operators (and, or, not) and each document is treated as a set of words. In order to quickly locate the documents matching the query, an *inverted index list* [30] is built for each of the words appearing in all the indexed documents. It is actually a mapping from a word to a list of documents containing the word. The inverted index lists are constructed and sorted during the indexing phase after the search engine crawls the webpages from the Internet, and more importantly they can be incrementally built up rather than rebuilt as new webpages arrive. Since the inverted index has been constructed beforehand, when the user issues a query of several keywords as well as the operators, it is very efficient to map the keywords to the containing documents. By simply conducting intersection or union on the sorted lists, even the billion-scale search engine can return the results in hundreds of milliseconds. A simple illustration for the boolean retrieval model is presented in Figure 1.3.

In normal Internet search, the operator "and" is the most commonly used. When a user enters the keywords separated by space, the operator "and" is implicitly applied. There are several problems with the boolean retrieval model. First of all, it is hard to express the complex

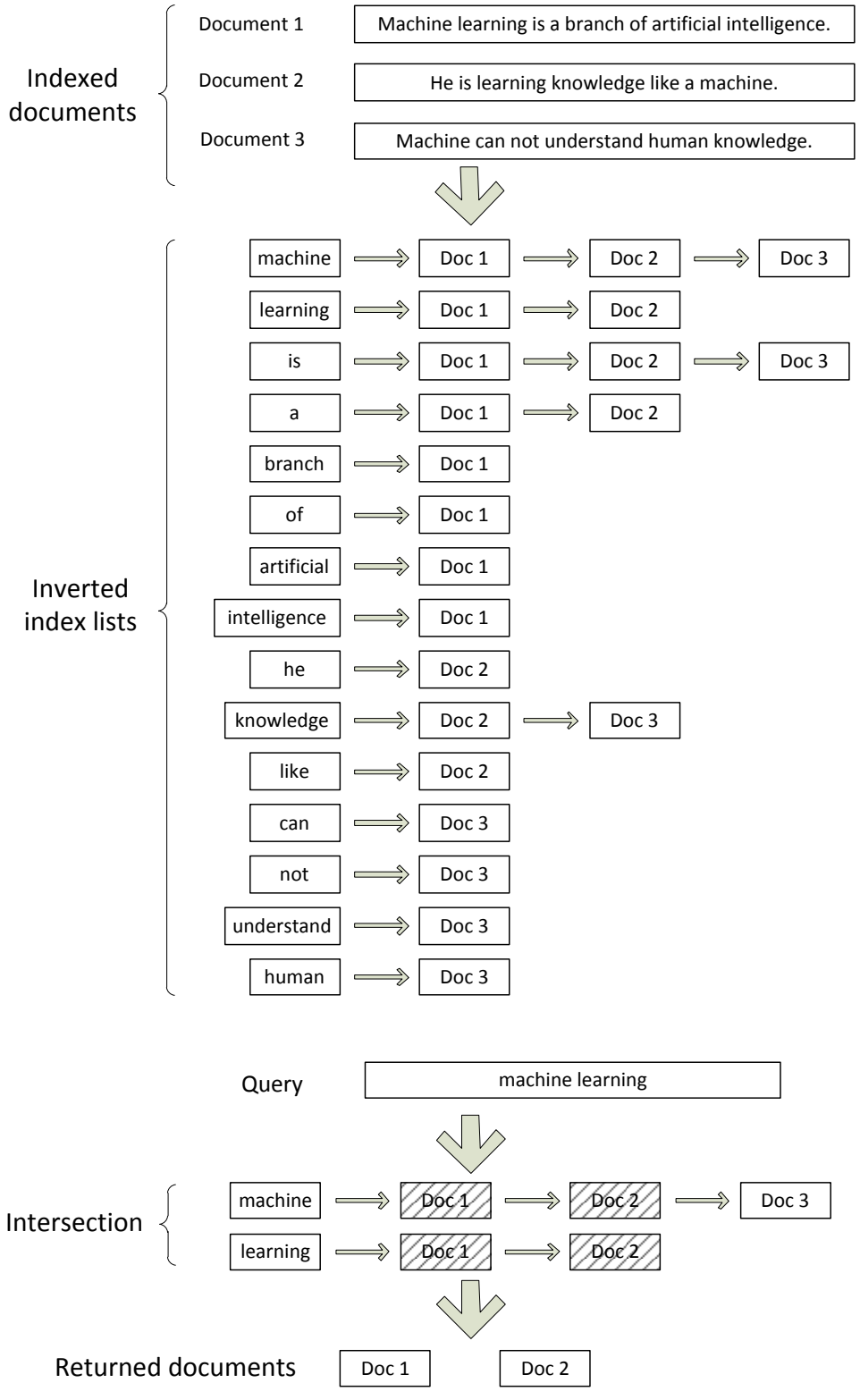

Figure 1.3: Simple illustration for boolean retrieval model.

	query vector	document vector
data	1	2
mining	1	1
is	0	1
to	0	1
discover	0	1
the	0	1
rules	0	1
from	0	1
$\vec{V}$	$\vec{V}(q) = \{1, 1, 0, 0, 0, 0, 0, 0\}$	$\vec{V}(d) = \{2, 1, 1, 1, 1, 1, 1, 1\}$
$\lvert\vec{V}\rvert$	1.41	3.32

Table 1.1: Vector conversion.

information need (e.g., the operator "or" and "not"). Secondly, the number of retrieved results is difficult to control, since all the documents matching the query will be returned. Finally, it is difficult to rank the results (e.g., the documents containing all the keywords of the query cannot be distinguished).

The second model is called *vector space model* [54] , which overcomes the problems of the boolean retrieval model. This model has been widely used in the modern web search engines as the basic function for information retrieval. The basic idea of vector space model is that both the query and the document are converted to word vectors, and similarity metrics are then applied on the two vectors to calculate a score on how similar the document is to the query. The score can be then used to truncate and rank the documents.

Cosine similarity [62] is a typical metric to calculate the score of two vectors. Given a query q and a document d, the score can be computed as

$$score(d, q) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|},$$

where $\vec{V}(q)$ and $\vec{V}(d)$ represent the word vectors for the query and the document respectively. The larger the value of cosine similarity, the more similar the document to the query. From the formula above, we can see, the numerator represents the dot product of the two vectors, and the denominator is the product of their Euclidean lengths for the normalization purpose. Suppose we have a query "data mining" and a short document "Data mining is to discover the rules from data." The conversion to the word vector can be conducted by using *bag of words* model, where each document is represented as an unordered collection of words, regardless of the grammar and order of the words. Thus, the query and document can be converted into vectors as shown in Table 1.1.

According to the cosine similarity, the score of the query and the document can be calculated as $\frac{1\times2+1\times1}{1.41\times3.32} = 0.64$.

To return the relevant results to the user, the vector space model does not need to calculate the score for all the indexed documents. Instead, the inverted index list can still be used in vector space model and only the documents that contain the keywords in the query will be considered. Thus, the web search engine can still instantly locate the results. Based on the scores computed, the search engine can rank the documents and return the top K to the user.

Another important advantage of vector space model is that the score generated can be combined with other factor scores (e.g., importance score, time series score) to produce more sophisticated ranking score.

1.1.2 Faceted Search System

Figure 1.4: Typical facets in Amazon.com.

Faceted search systems, such as in Amazon.com and Kijiji.ca, allow users to explore a collection of information by applying multiple filters in the facets. The search system can usually have more than one facet, and each facet can be either a flat list or a structured hierarchy of various categories. For example, in Amazon.com, the top facet is the department facet, where all the products are categorized into a hierarchy. If we search "iPad" in the category "tablets", besides the department facet, additional facets (e.g., display size, hard drive size, brand, price, etc.) will be displayed to the user (shown in Figure 1.4).

The key part of faceted search system is the classification of various products or items into the multiple facets. As many modern e-Commerce websites adopt the customer-to-customer business model, the label of the categories in each facet can be easily obtained from the customers who are selling products. For example, a customer wants to sell a new product, say a tablet, he/she needs to provide a detailed list of the specifications, such as the category it belongs to, the display size, the brand name and the desired price, etc. Those information will be used to classify the product into various facets to assist the searching.

Usually users search in two ways in the faceted search system. The first way is to directly search the keywords in the search box, and then the user can not only obtain a flat list of results matching the keywords but also a group of facets with the results classified into their categories (sometimes displaying the number of the results). Afterwards, the user can iteratively refine the search results by clicking the categories of those facets. Suppose a customer wants to buy an iPad on a e-Commerce website, the customer can search the keyword "iPad" as the query (see the left subgraph of Figure 1.2). The returned results may be grouped into multiple categories (e.g., "tablets" and "computer accessories", etc.), since the iPad cases will be classified into the category "computer accessories" rather than "tablets". By clicking into "tablets", the irrelevant

results in "computer accessories" can be filtered out. For the second way, the user can first click into a desired category and all the products in the category will be shown to the user. Then by issuing a query of keywords, the user can restrict the results within the category. For the customer who is planning to buy a tablet but not sure which type to buy, he/she can click into the category "tablets", have an overview and a comparison of different types, and then further search the keywords of the desired one.

The advantages of faceted search are twofold. First of all, users are able to view a summary of the search results and know how the returned results are organized by different criteria. Secondly, users can browse each of the categories in some facets and have an overview of what already exist. It is particularly useful when the user is not sure which keywords to search (or what products to buy).

1.2 Integrating Topic Facet into Web Search Engine

As the faceted search has tremendous benefits, why not introduce it to the keyword-based web search engine? In fact, Google has already embedded the faceted search into its original system. From Figure 1.1, besides the result list, on the left side there is a vertical bar listing some facets (e.g., result type and location). However, those facets are still limited compared to the diverse needs in search. The most intuitive way for users is to organize the documents into common sense topics, such as Arts, Science, Sports, which can further break down into sub-topics. In this case, diverse needs from different users can be covered better. Why does Google not include such a hierarchy into web search? It is probably due to the difficulty to classify the vast amount of webpages on the Internet into the hierarchical topics, since this task cannot be accomplished by human effort. In this thesis, we will report our first attempts to implement a new web search engine which integrates a hierarchical topic facet, by leveraging machine learning techniques to automatically classify webpages into our hierarchy. In the following parts, we will first explain the motivation of building this search engine in detail. Then we will briefly discuss the difficulties to construct it. Finally, we will present the user interface and the architecture of the new search engine SEE.

1.2.1 Why Do We Need such a Search Engine?

We already know some of the benefits that the facets will bring to the users in the e-Commerce websites as we mentioned in Section 1.1.2. Why would we need the hierarchical topic facet in the web search engine? Why not just use the keyword-based search? In what situations does this hierarchy benefit us most in search?

Simple keyword search may not work well when the user intends to search target information within a specific topic. In this scenario, usually the user will issue two types of queries, *ambiguous query* and *topic-related query*. For example, if the user intends to find worm only in the animal concept, he/she may search directly "worm" which is an ambiguous query, or "animal worm" which is a topic-related query. Here, the word "worm" itself is ambiguous, which can mean a kind of animal or a type of computer virus. In the query "animal worm", the keyword "animal" is intended to restrict the search within the topic "animal". As another example, if the user would like to find computer books, he/she may issue a query as "books"

which is an ambiguous query, or "computer books" which is a topic-related query. In this case, although the word "books" itself is not ambiguous, the query "books" is still ambiguous with respect to the user intention, since it can mean any kinds of books rather than the computer books. In the query "computer books", the keyword "computer" is intended to constrain the search domain. However, those two types of queries both have their limitations.

Ambiguous query may introduce many irrelevant results and leads to a low precision. In information retrieval, *precision* [1] is defined as the ratio of the number of relevant results retrieved over the total number of the retrieved results. Mathematically, it can be defined as

$$Precision = \frac{N_r}{N_r + N_{ir}},$$

where N_r is the number of relevant results retrieved and N_{ir} is the number of irrelevant results retrieved. Since the ambiguous query can match information in different domains, the number of relevant results N_r is supposed to be much smaller compared to the number of irrelevant results N_{ir}, which causes the low precision.

Topic-related query may truncate a lot of relevant results and lead to a low recall [1]. In information retrieval, *recall* is defined as

$$Recall = \frac{N_r}{N_{tr}},$$

where N_r is the number of relevant results retrieved and N_{tr} is the total number of the relevant results in all the indexed documents. Since the keyword(s) used to represent the topic may not match the documents in that topic that do not contain the used keyword(s), a certain number of relevant results may be filtered out (in boolean retrieval model) or low-ranked (in space vector model). It is reasonable that the recall will be low.

Why is it that the documents in a topic may not contain the keyword(s) used to represent the topic? It is because a topic is generally an abstract concept covering a wide range of sub-topics, and the name for the topic is used to describe the entire topic, thus may not appear in some specific documents. For example, in the topic "animal", a large part of the documents may talk about dog, bird, tiger or worm, without mentioning "animal". Also, a lot of documents in the topic of "computer" may be specifically on programming, Internet or CPU, and thus they do not necessarily contain the keyword "computer". Even they are talking about computer itself, the synonyms, such as PC, machine, can be used instead.

In order to find the desired documents in a topic, the user often needs to try a large number of queries with different keywords combinations. Sometimes, the user needs to guess what are the keywords that are likely to appear in the desired documents. For example, in order to find the animal worm, after the failure of the query "animal worm", the user may try to issue another query, say "soil worm", since worms used to live in the soil and the documents talking about worm are likely to contain the keyword "soil" too. If this query still fails to achieve the purpose, the use may consider to use "crawl worm". It is because the way that worms move is to crawl. This tedious searching process can easily annoy the user and the user will end up with no findings.

However, if a topic hierarchy is introduced into the web search engine and all the related documents are classified into each topic category, the user needs not try multiple keywords that may be contained in a topic, and the only query the user needs to input is the keywords for the

target information within the topic category. For the example to search computer book, if there exists a category "computer" containing all computer-related documents (e.g., programming, Internet or CPU, etc.), by using the keyword "books" as the query, all the relevant documents containing the keywords "programming books", "Internet books" or "CPU books" will be returned to the user. The hierarchy can help the users save a lot of time in searching the desired results in the web search engine.

1.2.2 Difficulties to Build the Search Engine

Although it is obvious that the topic hierarchy is beneficial, it is non-trivial to introduce such functionality into the web search engine. There are several difficulties in building the web search engine with a hierarchical topic facet.

First of all, in contrast to the e-Commerce website, the Internet does not have a policy to force the web creators to tag the webpages with the topic labels when they are published. The free Internet environment makes the major part of the webpages on the Internet unlabeled. Without the topic labels, how can we classify them into the categories in the topic hierarchy? The simplest but the most infeasible approach is to ask human to label the Internet webpages. Let's assume there are one billion webpages on the Internet. We can employ 1000 workers, and each of them labels 1000 webpages per day, then we need almost three years to finish the labeling for all the pages. To ensure the quality of the labels, one webpage may need to be labeled by multiple workers, which can cause even higher cost. In our work, we will leverage the power of machine learning techniques to automatically classify them into the categories of our topic hierarchy.

The second challenge is how to appropriately select topics for the hierarchy and how to accurately classify the webpages into the multi-level hierarchy. The common machine learning algorithms are designed for binary or multi-class classification problems. In our case, a webpage may simultaneously belong to multiple categories which are in a tree structure. For example, a news webpage talking about a concert held for a soccer game should be categorized into "arts" → "music" as well as "sports" → "soccer". In this thesis, we will utilize the data collection in Open directory Project as the training data and the hierarchical classification approaches to categorize the Internet webpages into the topic hierarchy.

Since the classification done by machine learning algorithms will not be 100% accurate, how would we build the web search engine SEE based on the imperfect classification? For each category, there are two types of errors (false positive errors and false negative errors). *False positive error* [44] for a category is that a document has been classified into the category but the document actually does not belong to the category. This type of errors may make the results noisy by introducing the documents from unrelated topics. *False negative error* [44] for a category is that a document has not been classified into the category but the document actually belongs to the category. This type of error may truncate some relevant results in a category. To handle the false positive errors, we propose a new ranking function. To deal with the false negative errors, we introduce a button on the user interface for users to explore additional results.

How to evaluate SEE is another challenge. The typical approach in information retrieval is to post some queries to the search engine and evaluate the relevance of the returned results to the query. However, very few of the previous works talk about how to conduct evaluation on the

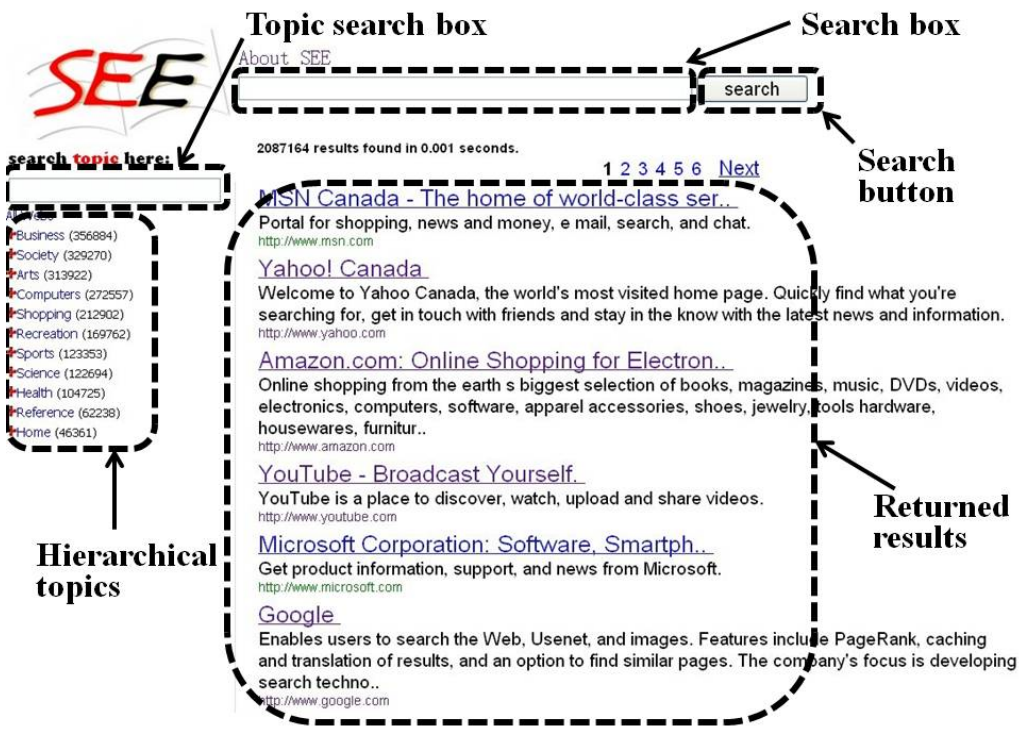

Figure 1.5: User Interface for SEE.

search engine with a hierarchy. Besides, in order to evaluate SEE, we need to know the relevant documents for each query. How can we get the query relevance is another problem. In our thesis, we will base our evaluation on *TREC* collection[2], and design an evaluation framework for SEE.

We also face the problem on how to further improve the performance of SEE. Introducing more training data will definitely improve the classification accuracy, but sometimes a huge number of training data may only slightly boost the accuracy, especially in the hierarchical setting. Besides, labeling a large number of documents requires a great amount of human efforts. How can we improve the classification accuracy while minimizing the human supervision? In our thesis, a new framework of active learning [55] will be proposed for hierarchical classification to reduce the human labeling cost.

1.2.3 System Overview

In order to provide an overview of the whole system, we will demonstrate the user interface of SEE including some typical search operations on the UI, as well as the architecture chart, in the following part.

Figure 1.5 illustrates the main user interface of our search engine. On the left side of the interface, we have a hierarchical tree for all the topics (Initially, only top level topics are shown). Furthermore, we can expand the tree, click into any category and search with any query

[2] http://trec.nist.gov/

within that category. When users are not familiar with the categories (topics) in the hierarchy, they can search the desired topic in the topic search box. It should be noted that when we do not search anything (no keywords in the search box) without choosing any category, instead of the blank page as in Google, the most popular websites in the world (such as Yahoo!, Google, etc.) are returned.

If we want to find specific information, say worm, a type of computer virus, how can we search in SEE? We can simply search "worms" in the category, named "computer" → "security" → "hacking". In fact, there are two possible sequences of steps to complete the search. The first sequence (demonstrated in Figure 1.6) is to first locate the category "hacking" and then search with the keyword "worms" within "hacking". Figure 1.7 demonstrates the second way, where we can first search with the keyword "worms" without clicking into any category, and then gradually narrow down the search domain to the category 'hacking". Both of the two step sequences can arrive at the same search results as we can see in the last subgraph of Figure 1.6 and Figure 1.7. However, there are only 27 results in the category. If we are unsatisfied with the 27 results, we can have the opportunity to explore more results by clicking a button called "Additional Results", when coming to the last page (3rd page in this case) of the returned results. By doing so, the user may have more chance to find the desired results. Figure 1.9 illustrates this operation to search more results about "worms". In this case, when we click the "Additional Results" button, one more result is found.

Another important feature of SEE is that people are able to browse any category without any keyword. In this case, the most popular webpages within the browsed category will be returned to the user. It offers a good way to help people know other unfamiliar areas (or domains). Figure 1.8 shows the results when the category "sports" is browsed. We can see that many popular sports websites are displayed, such as `www.nba.com`, `www.nfl.com`, etc.

Figure 1.10 presents the overall architecture of the whole system. The entire system can be divided into three sub systems, classification system, crawling system and web search engine. The three systems are inter-connected. In the classification system, we train a classifier based on existing labeled data. Then after we start running the crawling system on the Internet, the web crawler [31] will crawl new webpages and index them into the web search engine. Since the category labels of those webpages are known, we will use the constructed classifier to categorize them into our hierarchy. Afterwards, those classification information will also be indexed into the web search engine. When the user issues a query within a topic on the user interface, the request will be sent to the web server. Then the web server will search the indexed documents as well as the indexed classification information, and return the final list of returned results back to the user.

The most time-consuming part in our system is to train the classifier. This step is actually conducted offline. Once the classifier is ready, the classification for a new crawled webpage can be done instantly. Thus, as long as the crawler is running, we can easily build up a web search engine with billion-scale of webpages. Besides, since the topic classification information is also indexed, the response time for the query with topic selection can be very quick. Thus, we can see, our search engine SEE has a promosing scalability.

Figure 1.6: Steps to find information about “worm” in “computer” → “security” → “Hacking”: first locate category, then search keywords.

Figure 1.7: Alternative steps to find information about "worm" in "computer" → "security" → "Hacking": first search keywords, and locate category.

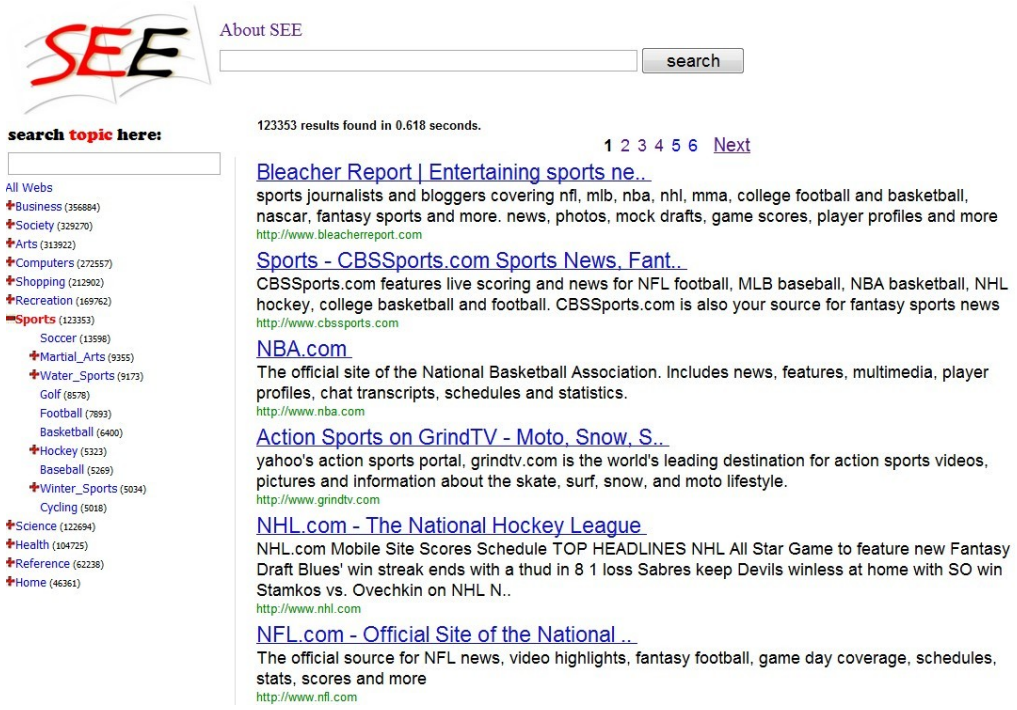

Figure 1.8: Browsing "sports" category without any keyword.

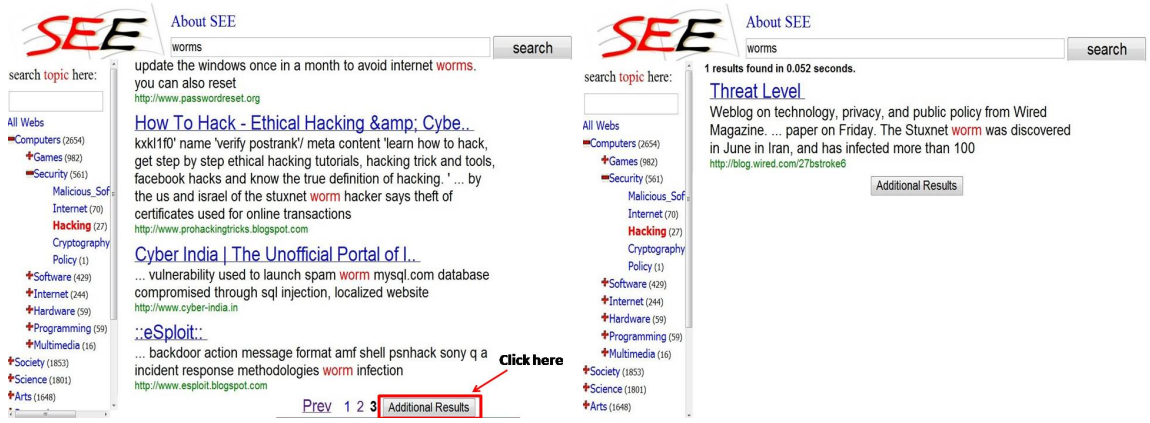

Figure 1.9: Explore more results about worm in "computer" → "security" → "Hacking": click "Additional Results" button. Right subgraph shows the results after the button is clicked. In this search case, only one more result is found.

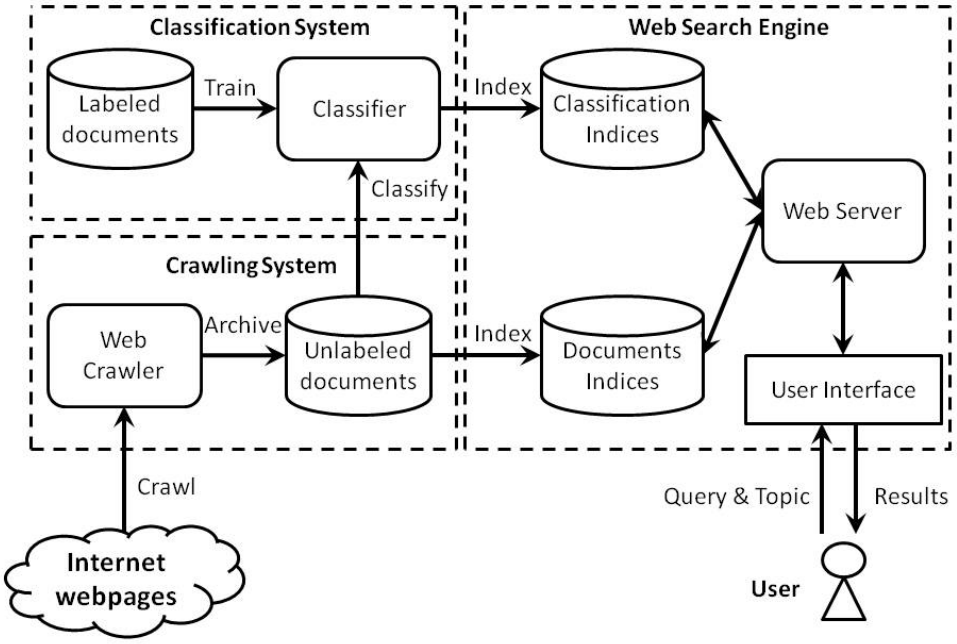

Figure 1.10: System architecture for SEE.

1.3 Contributions of the Thesis

Web search engine has been around for almost two decades, but keyword-based model has been always in the leading role. As we discussed earlier in this chapter a hierarchical topic structure will definitely benefit the search experience, why have those major web search engines still not implemented it? It is mainly because manually classifying vast number of Internet webpages is not realistic. In this thesis, we will leverage machine learning techniques to automatically complete this challenging task and build a web search engine with a topic hierarchy embedded.

In Chapter 3, we extract a reasonable hierarchy from Open Directory Project (*ODP*) and use the webpages in ODP as our base training data. Then we utilize the top-down hierarchical classification approach to classify new Internet webpages into our hierarchy. By exploring different feature selection methods, classifier parameters and calibration algorithms, we optimize the performance of our hierarchical predictive model. According to the evaluation results, the classification performance is satisfactory.

Chapter 4 discusses how to leverage the classification results to build the search engine prototype SEE. Here, we face two major challenges: how to deal with the false positive classification errors and rank the results within each category, and how to handle the false negative errors introduced in the classification phase. Accordingly, we design a novel ranking function and a new interface to let the users explore more results. We conduct a comprehensive evaluation by using a well-known data collection for information retrieval. The results demonstrate that the hierarchical version of SEE can achieve better search results than the flat version in most of the queries, particularly when the query is intended to search within a topic. The work

in Chapter 3 and 4 was published in the *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)* [32].

In order for SEE to achieve better search performance, it is critical to improve the classification performance, which often requires human efforts to label more webpages as the training data. Thus, it is worthwhile to study the problem on how to maximize the classification performance in the hierarchical setting with minimal human supervision. We propose a novel multi-oracle setting and a new active learning framework for the hierarchical classification. According to the experimental results, our methods can largely reduce the human labeling costs. The work was published in the *The 16th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2012)* [39].

We list our major contributions in the entire thesis as follows.

1. We propose the new idea of integrating a hierarchy into general web search engine and implement a prototype of this new search engine SEE.

2. We take advantage of machine learning and data mining techniques (e.g., SVM, hierarchical classification and data preprocessing) to categorize Internet webpages into our hierarchy. We design a parallel framework to improve the efficiency for the training and classification processes. Besides, we propose an algorithm to find the best parameter combination that maximizes the classification performance. The evaluation results demonstrate that our classification performance is good.

3. To build the new search engine SEE, we propose a novel ranking function and design a new interface to explore additional results when returned results are limited. We also conduct a comprehensive evaluation on SEE, and the results show that the hierarchical version of SEE can achieve better search results than the flat version without hierarchy in most of the queries, particularly when the query is suitable to search within a topic.

4. To further improve SEE with minimal human effort, we propose a novel multi-oracle setting and a new active learning framework for the hierarchical classification. According to the experimental results, our methods can greatly reduce the human labeling effort.

Chapter 2

Review of Previous Work

Since we are building a new type of search engine with a topic hierarchy embedded, we would like to know what similar search engines are. We will review some existing web search engines and discuss the similarities and differences with our search engine SEE. Furthermore, because we will use machine learning techniques to classify the webpages, we will also review some popular classification learning algorithms, and discuss their characteristics. However, most of reviewed classification algorithms are designed to deal with binary problems, and can only be used to train the base classifier. In our case, webpages need to be classified into multiple categories in the hierarchy. We will further leverage a technique called *hierarchical classification* to handle this task. In this chapter, we will also review three typical approaches for hierarchical classification.

2.1 Other Advanced Search Systems

Besides the keyword-based web search engines, such as Google and Bing, there exist several other forms of web search systems. They offer alternative ways to assist the users in searching on the Internet.

2.1.1 Web Clustering Engine

Web clustering engine [76, 7] is the most similar one to the search engine we are attempting to build. By grouping the results returned by a search engine into a hierarchy of labeled clusters, it provides a complementary way to explore the results in the flat list returned by the keyword-based web search engines. Figure 2.1 shows the user interface of a web clustering engine named Yippy. There are a number of web clustering engines in both research and commercial domains. The typical ones are AISearch [77], SnakeT [20], Carrot2 [60], CREDO [6], KartOO[1] and Yippy[2], etc.

Web clustering engines are helpful in the following aspects as mentioned in [7].

1. Fast subtopic retrieval. If the documents that pertain to the same subtopic have been

[1] www.Kartoo.com
[2] www.yippy.com

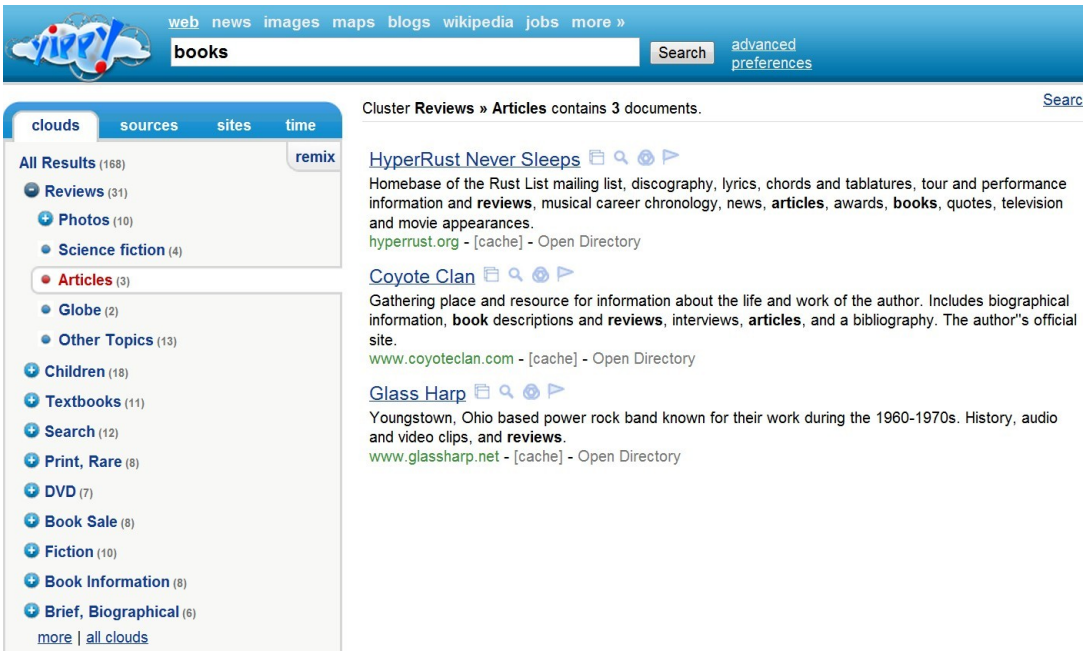

Figure 2.1: User Interface for a web clustering engine named Yippy.

correctly placed within the same cluster and the user is able to choose the right path from the cluster label, such documents can be accessed in logarithmic rather than linear time.

2. Topic exploration. A cluster hierarchy provides a high-level view of the whole query topic including terms for query reformulation, which is particularly useful for informational searches in unknown or dynamic domains.

3. Alleviating information overlook. Web searchers typically view only the first result page, thus overlooking most information. As a clustering engine summarizes the content of many search results in one single view on the first result page, the user may review hundreds of potentially relevant results without the need to download and scroll to subsequent pages.

It should be mentioned that most existing clustering engines do not construct the index for the documents, instead they are *meta search engines* [42, 13]. They forward the user queries to some other public search engines (e.g., Google, Bing, ODP, etc.), and utilize clustering algorithms to group those returned results and visualize them in various forms. Thus, the clustering is actually a post-processing step after the retrieval of the query-relevant results from the web search engine. There are generally four key steps to implement such web clustering engine: raw results retrieval, feature generation, cluster construction, and cluster visualization.

The main task of *raw results retrieval* is to repost the user query to one or multiple data sources, such as reputed web search engines, web directory and Wikipedia, and obtain about 100 to 1000 results[3] relevant to the query. The content of those results normally contains the title and URL of a webpage, snippets generated from the source system (e.g., search engine). Then all those information will be fed to the next step as the raw results.

[3]The number depends on the query.

The next step is the *feature generation*, which is a preprocessing step for the actual clustering operation. The major task is to transform the raw results (title, URL, snippets) into text features accepted by the common clustering algorithms. Several techniques in natural language process can be utilized here, such as tokenization, stemming, n-grams, and feature extraction and selection.

In *cluster construction*, three types of clustering algorithms can be used to cluster the documents based on the features produced in the last step.

1. **Data-centric algorithms**. This type of algorithm clusters the documents by finding the data center of each cluster and emphasizes the quality on the closeness of the documents in the cluster. Those algorithms are usually modified versions of traditional clustering methods (e.g., hierarchical, optimization, spectral, etc.) with some additional textual description for each cluster.

2. **Description-aware algorithms**. A serious problem of data-centric algorithms is that the generated textual description for each cluster may not be accurate and friendly for the users to search desired results. This problem is what description-aware algorithms attempts to solve. This algorithm aims to improve the quality of the textual labels for the generated clusters and make them interpretable to human, while still pursuing the quality of the clustering.

3. **Description-Centric Algorithms**. This type of algorithms is designed to focus on guaranteeing the quality of descriptions (labels) for each cluster, so that users can better understand the meaning of the cluster and improve their search experience. If it is difficult to describe a cluster, then it is not useful to present the cluster to the users.

The last step is *cluster visualization* [36]. This step is to display the clusters produced in the client-side browser. The clusters can be demonstrated in various chart formats. The most common approach is the hierarchical tree structure where each cluster is represented by a folder icon, as shown previously in Figure 2.1.

Intuitively, web clustering engines are very similar to SEE, particularly in the following aspects.

1. **Similar Goal**. Both web clustering engines and SEE aim to categorize the search engine results and narrow down the search domain and improve the search performance.

2. **Similar User Interface**. As we can see in Figure 2.1, on the left hand side, there is also a tree structure similar to the hierarchical tree in SEE (Figure 1.5). Users can drill down along the path and search the results in an appropriate category.

However, they are dramatically different. We will list some key characteristics of the web clustering engines and do comparisons with SEE.

1. **Limited Clustered Results**. As discussed before, clustering engines simply post-process the returned results (ranges from 100 to 1000) from web search engines and group them into clusters. The limited results as shown in Figure 2.1 may not meet the user's need. In SEE, the classification is conducted on the entire indexed documents and all the documents matching the user query can be returned.

2. **No Index of Documents**. The lack of indexed documents makes the clustering engines heavily dependent on other data sources. Furthermore, without index of documents, other than the returned results, users are not able to explore other possible results in each cluster. In SEE, we build our own index for all the documents as well as the classification information, so that the user can explore each topic category thoroughly. Without issuing any query, SEE becomes a web directory, users can browse the most popular webpages within any topic, by integrating the factor of page importance.

3. **Unpredictable Clusters**. An important feature of clustering is to explore unknown groups. However, this feature will inevitably lead to many unpredictable and unmeaningful clusters. For example, in Figure 2.1, by searching "books", the user may expect to see books in different fields, such as in computer, arts, business. However, the resulting clusters somehow mix everything together, which may confuse the users. In contrast, SEE uses predefined common-sense categories (see Section 3.1) and the results will always be classified into the existing categories without creating new categories.

4. **Insufficient Data Input**. Most clustering engines simply rely on the snippets to conduct the clustering task. Since the snippets are relatively short, the accuracy and reliability of the constructed clusters may not be satisfactory. In contrast, SEE utilizes the full text of documents to build the classifier, which is more likely to produce better classification results.

5. **Computational Inefficiency** The clustering process is often conducted on-the-fly after user posts the query. The high computational complexity of clustering (as high as $O(n^3)$) will significantly affect the response time. It may work well when the number of results returned is small. However, if we would like to return more results to the users, the clustering engines may fail to deliver the results in a short time. Since all the classification information is indexed, SEE can response the users instantly.

2.1.2 Web Directory

Another type of similar system to SEE is the web directory. Rather than a keyword-based search engine, a web directory does not return a list of ranked search results as users post queries. Different from a web clustering engine, a web directory does not construct a hierarchical tree of categories on-the-fly. A web directory maintains a good number of websites by human effort, and those websites are organized in a hierarchical structure. The users can browse the websites in each category as well as search the keywords within each category. The typical web directories include the *ODP* (*Open Directory Project*)[4], *Yahoo Directory*[5] and *Google Directory*[6]. In the following, we will give a brief introduction on the ODP[7], since later we will utilize it to build our search engine SEE.

The Open Directory Project, also known as *DMOZ*, is the largest and most comprehensive human-edited directory of the Web. There are a vast number of volunteer editors (usually

[4] `www.dmoz.org`

[5] `dir.yahoo.com`

[6] It has been shut down.

[7] The other two web directories are similar to ODP.

domain experts), who participate in the construction and maintenance for the ODP. The ODP was originally named Newhoo which was founded in June of 1998. Then in the same year Netscape purchased the website and still continued to run the site in a free mode. Up until now, the ODP includes almost five million websites and becomes the largest web directory in size, and over 50, 000 volunteer editors over the world have been involved in contributing to its development.

Like other web directories, the ODP categorizes its websites into categories of various topics. There are over one million categories and the deepest one goes up to 14-level depth. How does the ODP collect those websites? In fact, the ODP allows Internet users or enterprisers to submit their websites without any charge. When submitting a website, the user needs to specify the category that the website is most likely to belong to. After the submission, the website does not directly go to the chosen category; instead, the human editors will double check if the category is a good choice or not. There might be a good chance that the category needs to be changed or new categories need to be added. Once confirmed, the submitted website will be included in the corresponding categories.

The significant effect to submit the websites to the ODP is that it will increase the ranking of the website when searching in Google. When accepted by ODP, two significant links will link to the submitted website, one from ODP and the other from Google Directory. Both of them already have high *PageRank* [47]. The consequence will be that the PageRank of the submitted website will be significantly boosted. Thus, it is very beneficial to list the websites on ODP.

Besides, all the data in the ODP are publicly accessible and can be downloaded free, which is a reliable labeled data source for research purposes, particularly useful for machine learning and data mining research.

Figure 2.2 demonstrates the main interface for ODP. In the figure, the top level categories with some of their subcategories in ODP are displayed. By clicking into each of the categories, the corresponding subcategories will be expanded. The users can also input a query in the text box of any category, which searches the websites matching the keywords only within that category.

Intuitively, web directories are very similar to our search engine SEE. In fact, they are significantly distinguished from each other. We first present their similarities listed in following.

1. **Predefined hierarchical categories**. Both web directories and SEE utilize the predefined hierarchical categories to classify their documents (websites and webpages).

2. **Supporting hierarchical browsing and keywords search**. In both web directories and SEE, by clicking into each category, the user is able to browse the content within that category without any query. Furthermore, the user is able to input the keywords in any category and the matched results within that category will be listed to the user.

We will also list some features of web directories and discuss the differences with SEE.

1. **Not a search engine**. Although queries can be handled in web directories, the purpose is to list and categorize web sites. It is not used for the purpose of general search (e.g., question and answer search, article search), since it only returns websites rather than webpages. In another word, a web directory is a data provider rather than a search

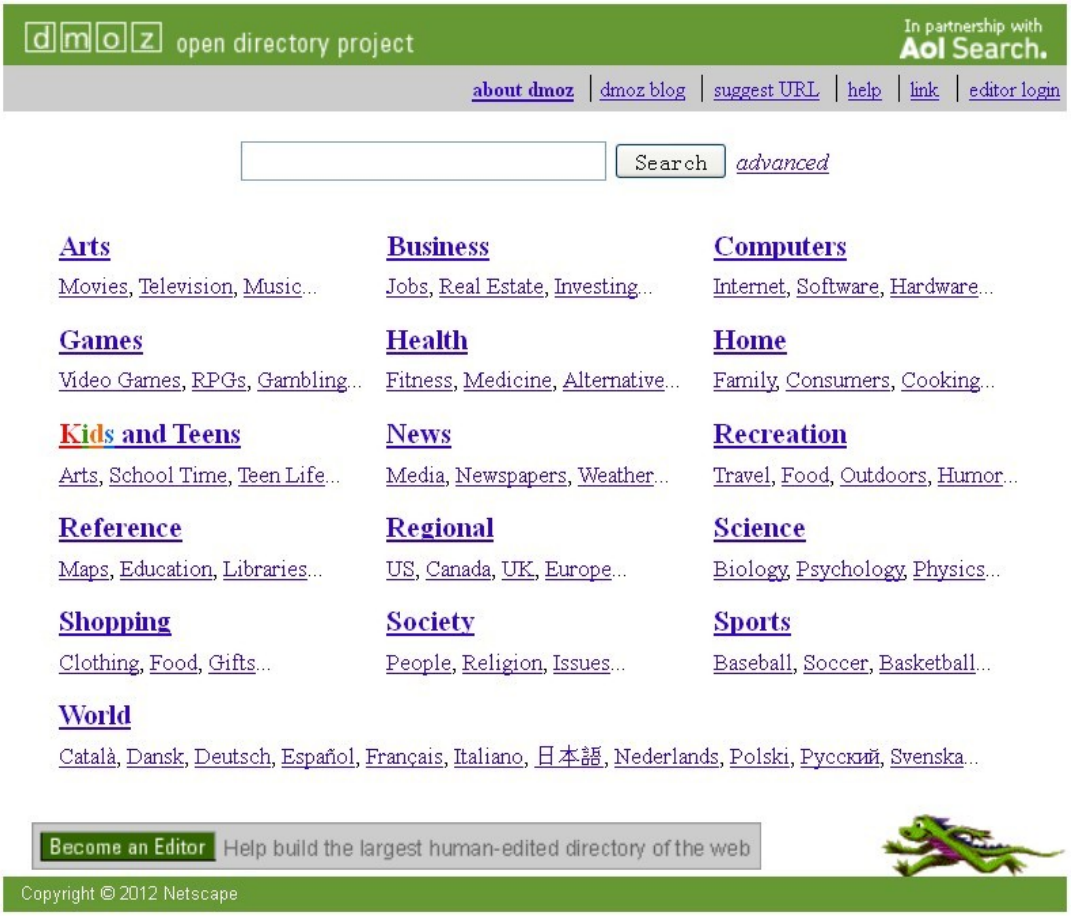

Figure 2.2: Interface of Open Directory Project.

engine. Moreover, they do not rank results. In contrast, SEE is a fully functional search engine.

2. **Limited scalability**. An important reason that web directories cannot be developed into a real search engine is that the categorization of websites is managed by human. Originally, Yahoo! was attempting to build a search engine in a form of web directory, but it failed to work as the number of published webpages increased dramatically on the Internet and manually categorization became infeasible. However, SEE leverages machine learning techniques to automatically categorize the webpages, thus it can be scaled easily as long as the classifier has been built.

2.2 Popular Classification Algorithms

Since SEE will utilize machine learning to classify the webpages, we will introduce several concepts related to machine learning and review some typical classification algorithms in the literature.

Machine learning can be defined as follows. A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [43].

Simply speaking, as a subdomain of artificial intelligence, machine learning empowers computers with the capability to learn from empirical data, without being programmed explicitly. In general, machine learning can be divided into two major areas, unsupervised learning and supervised learning. *Unsupervised learning* solves problems where we need to discover the hidden structures in unlabeled data. For example, the clustering technique used in web clustering engines falls into the field of unsupervised learning, since it just tries to find the similar groups of webpages without knowing the category labels. *Supervised learning* attempts to infer a function based on a set of training examples with labels provided. The function, also called *model*, can be used to predict the label values of new examples. If the label is numeric, the task is a *regression* task. If the label is discrete, the task becomes a *classification* problem, and the constructed model can be called *classifier*. In the thesis, classifying webpages into hierarchical topics is actually a classification problem. There are several types of classification problems, such as *binary classification*, *multiclass classification* [25], *multilabel classification* [64]. The task of *binary classification* is to classify examples into one of two target classes (*positive class* and *negative class*[8]). Rather than only two classes, *multiclass classification* can classify new examples into one of three or more classes. In *multilabel classification*, new examples can be classified into more than one classes.

In the following, we will introduce some classic learning algorithms for classification tasks and discuss their advantages and weaknesses, as well as which one is more suitable for our problem.

2.2.1 Decision Tree

Decision tree [49] is one of the most commonly used classification algorithms. Based on the training data, the decision tree algorithm builds a top-down tree structure to fit the data. Basically, the tree construction follows recursive steps starting from the root. Each time, the data is split by the current best attribute, until the distribution of examples on each node of the tree becomes pure (belonging to the same class) or satisfies a certain condition. When a new example is coming, its attributes will be tested from the root until the leaf node is reached, where the class of the example can be determined.

For example, suppose we would like to predict whether or not we will enjoy playing tennis today, according to some input attributes such as the outlook, windy or not, humidity, etc. Based on a training set consisting of previous playing records as shown in Table 2.1, we can build a decision tree as shown in Figure 2.3. We can see, each internal (including root) node of the tree is associated with an attribute, while each leaf node is connected to a class. For example, on the root node, according to the value of the attribute *outlook*, the whole training data can be divided into three parts (outlook==rainy, outlook==overcast and outlook==sunny). Two of those three parts are further divided, while the other one (outlook==overcast) directly

[8]In the rest of the thesis, *positive documents (or examples)* mean the documents (or examples) belonging to positive class, and *negative documents (or examples)* mean the documents (or examples) belonging to negative class

outlook	temperature	humidity	windy	play or not
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

Table 2.1: Weather dataset.

leads to a leaf node (the class"YES"). It means, when the outlook is overcast, we always enjoyed playing tennis according to the previous data. After the decision tree is constructed, if today is sunny but too humid, the classifier will output "NO", meaning that we will probably not enjoy the tennis playing today.

The key step in building a decision tree is to find the best attribute to split the data on each internal node. There are many proposed metrics [46, 41] to accomplish it. The most typical and commonly used metric is *entropy*, which is used to calculate the uncertainty associated with a random variable. In this case, the class attribute of each example can be the random variable. If the class attribute has n different values, then the entropy of an example set X can be defined as

$$Entropy(X) = \sum_{i=1}^{n} -p_i \times log_2 p_i,$$

where p_i is the proportion of X belonging to class i.

Generally, the purer the class on a node, the less uncertain the class is, and the lower the value of entropy. The best attribute is the attribute that produces the greatest *information gain*, which means the reduction in the entropy. Mathematically, the information gain of an attribute

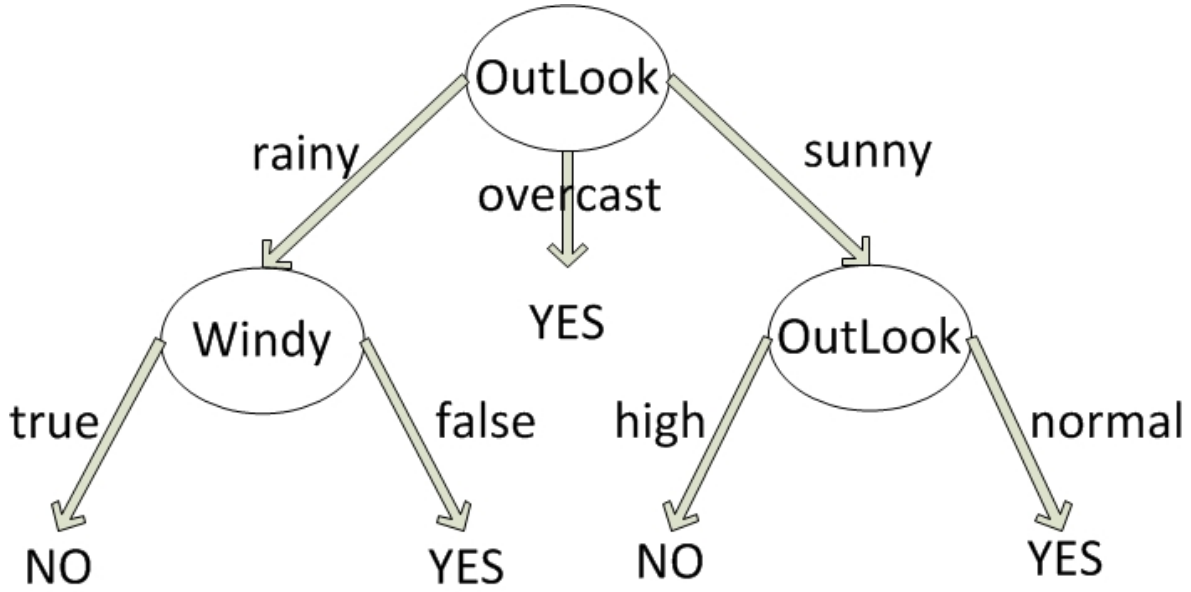

Figure 2.3: Decision tree structure.

A on a set of examples X can be defined as

$$Gain(X, A) = Entropy(X) - \sum_{v=v(A)} \frac{|X_v|}{|X|} Entropy(X_v),$$

where $v(A)$ is the set of all possible values for the attribute A and X_v is the subset of X such that $A = v$. For all the attributes, we simply choose the one with maximum $Gain(X, A)$ for each internal node to split the current data on the node.

Another issue with decision tree is how to avoid overfitting in decision tree? Decision tree can always perfectly classify the training examples by building a large (deep) decision tree. However, this approach sometimes overfits the training examples when we have noise in the training data, or when the size of training data is too small. To tackle this problem, usually a pruning step is adopted. Pruning typically removes the small branches in the tree, which are likely to be unreliable. It can make the decision tree more robust to noise or more accurate for unseen examples. More specifically, it uses a post-pruning rule as follows:

1. Build a decision tree following the above algorithm described.
2. Convert the tree into an equivalent set of rules.
3. Prune each rule by removing any preconditions that result in improving its estimated accuracy.
4. Sort the pruned rules according to estimated accuracies.

The advantages of decision tree can be listed as follows.

1. **White box model**. The model can be easily interpreted and gives good intuitions to the decision maker.

2. **Rules generation**. The decision tree can be converted into simple and useful rules, where experts have difficulties to formalize their knowledge.

There are also some disadvantages of decision tree.

1. **Hard to model complex concept**. Normally, the rule-based decision tree has difficulties to model the complex target concepts with many relevant attributes. It is due to the fact that the reliability of the rules generated by decision tree strongly correlates with the number of observed data. On the deep level of the decision tree, the number of training data significantly decreases, thus the rules become unreliable.

2. **High time complexity for large feature set**. The decision tree can be very efficient when the attribute set is small. However, when there are millions of attributes, in order to find the best attribute on each node, the linear comparison for all the attributes is very costly. If f is the number of the features and p is the number of the nodes on the tree, the time complexity can reach $O(f \times p)$.

To build our search engine SEE, we need to classify the text contents in the webpages, which usually have millions of attributes (words). Thus, as we mentioned above, decision tree is not a good choice.

2.2.2 Naive Bayes

Naive Bayes [50] is another typical classification algorithm which is based on Bayes rules. Given an example x, if we want to classify it into one class in a class set C, we simply calculate the *posterior probability* of x belonging to each class, and the one with the largest probability will be the predicted class. Mathematically, give an example x, the posterior probability of a class c can be calculated as,

$$Pr(c|x) = \frac{Pr(c)Pr(x|c)}{Pr(x)},$$

where $Pr(c)$ is the prior probability of the class c. For each of other classes, we calculate the posterior probability probability the same way. We will find $Pr(x)$ is the same for all the classes, thus we only need to compare $Pr(c)Pr(x|c)$ for different classes. $Pr(c)$ can be computed by calculating the percentage of the examples belonging to c in the whole training data. However, it is more difficult to calculate $Pr(x|c)$, since we do not know the true probability of x in the underlying distribution. By assuming that the attributes in x are independent, Naive Bayes uses an estimation to approximate $Pr(x|c)$ as follows:

$$Pr(x|c) \approx \prod_{i=1}^{n} Pr(x(A_i)|c),$$

where $x(A_i)$ is the value of x on the attribute i. By decomposing the example into attributes, we can easily compute the $Pr(x(A_i)|c)$ as the percentage of examples with attribute i equaling $x(A_i)$ in the examples belonging to class c in the training set. For example, by applying Naive Bayes to the same dataset as presented in Table 2.1, given the attributes of *today* as

$$outlook = rainy, temparature = hot, humidity = high, windy = FALSE,$$

we can calculate the posterior probability probability $Pr(yes|today)$ and $Pr(no|today)$ as follows.

$$\begin{aligned} Pr(yes|today) =& Pr(yes)Pr(outlook = rainy|yes)Pr(temparature = hot|yes) \\ & Pr(humidity = high|yes)Pr(windy = FALSE|yes) \\ =& \frac{9}{14} \times \frac{2}{9} \times \frac{1}{9} \times \frac{3}{9} \times \frac{6}{9} = 0.0035 \end{aligned}$$

$$\begin{aligned} Pr(no|today) =& Pr(no)Pr(outlook = rainy|no)Pr(temparature = hot|no) \\ & Pr(humidity = high|no)Pr(windy = FALSE|no) \\ =& \frac{5}{14} \times \frac{2}{5} \times \frac{2}{5} \times \frac{4}{5} \times \frac{2}{5} = 0.0183. \end{aligned}$$

Thus, according to the prediction of Naive Bayes ($Pr(no|today) > Pr(yes|today)$), today is not a good day to play tennis. Furthermore, by normalizing the two probabilities above, we can obtain a probabilistic likelihood for each class ($\frac{0.0035}{0.0035+0.0183} = 0.161$ and $\frac{0.0183}{0.0035+0.0183} = 0.839$). Those normalized probability can be useful in many practical problems (e.g., expected cost estimation).

There is a special case we should pay attention to. For example, if there is no examples with $temparature = hot$ in the class yes, then $Pr(temparature = hot|yes) = 0$, which will cause the $Pr(yes|today) = 0$. Apparently, in this case, the calculated posterior probability will be incorrect. In order to deal with this issue, a method called *laplace correction* is introduced to Naive Bayes. Basically, laplace correction increments one to both denominator and numerator of the probability calculation for each attribute. For example, if originally $Pr(temparature = hot|yes) = \frac{0}{9}$, after laplace correction, $Pr(temparature = hot|yes) = \frac{1}{10}$.

Naive Bayes has some advantages for classification problems, which are listed as follows.

1. **Efficient model construction**. By simply scanning all the examples in the training set once, the probability for each attribute value as well as the prior probability of each class can be calculated.

2. **Capability for large attribute set**. No matter how large is the attribute set, given a new example, the product of them times the prior of the class produces the posterior probability of that class.

There are also some disadvantages with Naive Bayes.

1. **Unrealistic independence assumption**. Naive Bayes makes an unrealistic assumption on the attribute independence. It assumes that given the class of an example, the attributes of the example are independent. In fact, most of the real cases, the attributes are correlated.

2. **Strong prior impact**. Given an example, to calculate the posterior probability for a class, we always need to multiply by the prior probability of the class. If the class is extremely imbalanced, such as 1% for a class and 99% for the other, the posterior probability will strongly bias the majority class.

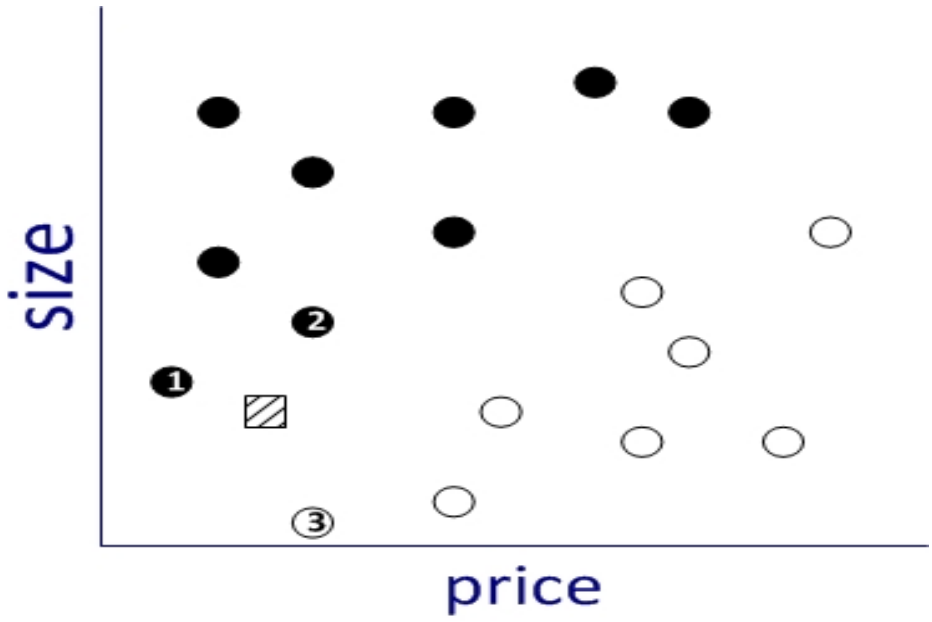

Figure 2.4: Demonstration of K Nearest Neighbor.

Since Naive Bayes can handle large attribute set, it has been already applied in text classification (e.g., spam filtering). However, according to some previous works [73, 27], in terms of text classification, the performance of Naive Bayes is not as good as some other learning methods such as K Nearest Neighbor and Support Vector Machine.

2.2.3 K Nearest Neighbor

K Nearest Neighbor (*KNN*) [15] is an instance-based classification algorithm. It belongs to *lazy learning methods* as opposed to *eager learning methods* such as Decision tree and Naive Bayes. Lazy learning method does not build the model in advance, and until an unseen example is coming, it begins to explore the training data and attempts to classify it.

The general idea of KNN is to decide the class of a new example by considering the class of the similar examples in the training set. Specifically, KNN classifies the new example based on the majority vote of the k training examples closest to this example, called *k nearest neighbors*. For example, given a house dataset with two attributes *price* and *size*, we need to predict if a new house is worth buying. Figure 2.4 demonstrates how KNN basically works. The solid circles represent the positive examples (houses purchased), while the hollow circles represent the negative examples (houses not purchased). Suppose $k = 3$, when a new example (dashed rectangle) is coming, it will be classified as positive examples, since 2 of the 3 closest examples (marked as 1,2 and 3) are positive.

An improved version of KNN, called *distanced-weighted KNN* [15], further takes the distance of the neighbors into consideration. The weight can be calculated by using the inverse square of the distance from each neighbor to the new example. The short the distance, the higher weight it puts on the neighbor.

KNN has some desirable features for classification as listed below.

1. **Effective local classification**. By weighting the distance, KNN can be a highly effective classification method for many practical problems. Since only k neighbors are considered, KNN is robust to noisy training data, and can be very effective given a large set of training data.

2. **No training cost**. Since KNN is a lazy learning method, no model is constructed for training and we directly classify new examples without paying any training cost.

Several weaknesses can be also found for KNN.

1. **High classification (prediction) cost**. Although we do not need to build the model for KNN, for each new example, KNN needs to go through all the training examples to find the k nearest neighbors of the new example. Suppose n is the number of training examples, the time complexity to classify one new example will be $O(n)$, compared to $O(k)$ for decision tree and Naive Bayes, where k is a constant. If we have a large number of examples to be classified, the time complexity will be very high.

2. **Choice of K**. The performance of KNN strongly relies on the choice of the parameter k. However, it is usually difficult to determine the optimal value for k.

To build the search engine SEE, we need to classify a large number of new webpages into the hierarchical categories. Thus, KNN is not a good choice for our task due to its high classification cost.

2.2.4 Support Vector Machine

The basic idea of *Support Vector Machine* [11], also called *SVM*, is to find the hyperplane that can well separate positive and negative examples in the training data with the largest margin. For example, we still use the same housing dataset in Section 2.2.3. By using SVM, we can find many different hyperplanes that can perfectly separate the solid circles (positive examples) and hollow circles (negative examples) as the lines shown in left subgraph of Figure 2.5. Among those lines, we can find a line that maximizes the margin between the positive and negative examples as the solid line shown in the right subgraph of Figure 2.5. We call the examples lying on the margin *support vectors*.

Mathematically, any hyperplane can be represented as a set of points X, which satisfies

$$W \cdot X - b = 0,$$

where $\cdot$ denotes the dot product and W is the normal vector to the hyperplane. Suppose the two classes of the training set are linearly separable, we can always find two parallel hyperplanes (dashed lines as shown in the right subgraph of Figure 2.5) such that no training examples stay between them and the distance between them is maximal. Mathematically, we can define them as

$$W \cdot X - b = 1, W \cdot X - b = -1.$$

The distance between the two hyperplane can be calculated as $\frac{2}{\|W\|}$. To maximize the margin, we need to minimize $\| W \|$. Thus, the model construction for SVM is to find the best W and

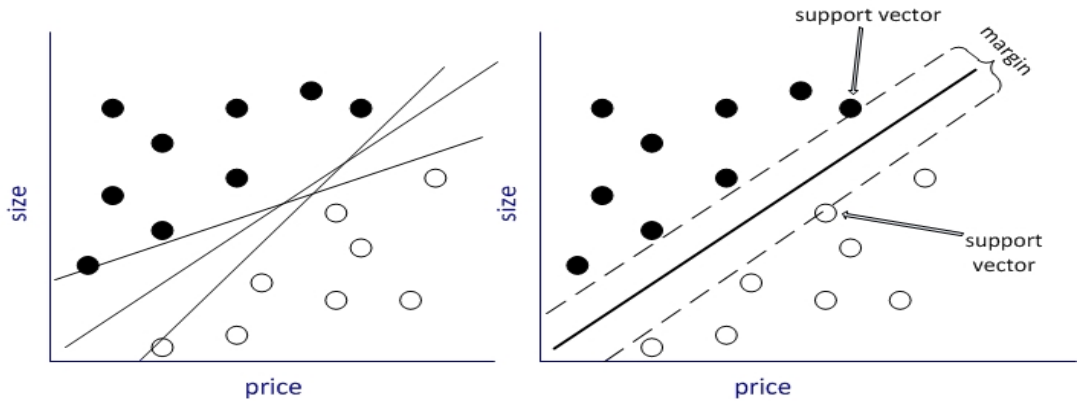

Figure 2.5: Demonstration of support vector machine.

b that minimizes $\| W \|$ and correctly classifies all the examples in the training set as well. To make the hyperplane classify the positive and negative examples perfectly, we should ensure, for each positive examples x_i in the training set, $W \cdot x_i - b \geq 1$, and for each negative examples x_i, $W \cdot x_i - b \leq -1$. The two inequations can be rewritten into one inequation as $y_i(W \cdot x_i - b) \geq 1$, where y_i is the class label ($y_i \in \{-1, 1\}$) for example i in the training set.

Thus, SVM can be transformed into an optimization problem as

$$\min_{w,b} \| W \|$$

$$subject\ to,\ y_i(w \cdot x_i - b) \geq 1,\ \forall i\ (1 \leq i \leq n),$$

where n is the number of training examples.

Since the problem includes a square root $\| W \|$, it is difficult to solve. However, we can substitute $\| W \|$ with $\| W \|^2$, without changing the original problem. Thus, it becomes a quadratic programming optimization problem.

$$\min_{W,b} \frac{1}{2} \| W \|^2$$

$$subject\ to,\ y_i(W \cdot x_i - b) \geq 1,\ \forall i\ (1 \leq i \leq n),$$

where n is the number of training examples.

We can further transform the problem to the following form by introducing Lagrange multipliers α.

$$\min_{W,b} \max_{\alpha \geq 0} \{\frac{1}{2} \| W \|^2 - \sum_{i=1}^{n} \alpha[y_i(W \cdot x_i - b) - 1]\}$$

By using the standard quadratic programming, now we can solve the problem. According to the "stationary" Karush–Kuhn–Tucker condition, the solution can be represented as a linear combination of vectors of the training examples

$$W = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

In fact, only the support vectors (satisfying $y_i(W \cdot x_i - b) = 1$) will have non-zero α. Therefore, we can calculate b by averaging over all the support vectors as

$$b = \frac{1}{N_{sv}} \sum_{i=1}^{N_{sv}} (W \cdot x_i - y_i).$$

The optimization problem introduced above is the *Primal form* for SVM. Since we know $\| W \|^2 = W \cdot W$, if we substitute $W = \sum_{i=1}^{n} \alpha_i y_i x_i$, then we can convert it into its *dual form* as

$$max\{\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i.j} \alpha_i \alpha_j y_i y_j x_i^T x_j\}$$

$$subject\ to\ \alpha_i \geq 0,\ \sum_{i=1}^{n} \alpha_i y_i = 0.$$

By calculating α, we can further compute w by $W = \sum_{i=1}^{n} \alpha_i y_i x_i$.

However, in certain circumstances, the positive and negative examples in the training set are not linear separable. How can we deal with this case?

One solution is that we can use the *soft margin* instead of the *hard margin*. By using the soft margin, we allow the imperfect separation by the hyperplane on the training set. We can reformulate the optimization problem as

$$\min_{W,b,\xi}\{\frac{1}{2} \| W \| + C \sum_{i=1}^{n} \xi_i\},\ subject\ to,\ y_i(W \cdot x_i - b) \geq 1 - \xi_i,$$

where the parameter C is the penalty for any of the training errors and ξ_i measures the degree of misclassification of an example i. Thus, the optimization problem becomes a problem to maximize the margin while minimizing the training errors.

The SVM we discussed above is called linear SVM, since the hyperplane is in a linear formula. To handle the non-linear separable data, another more sophisticated method is to use kernel. The basic idea of kernel to transfer the original attribute space of the data into a higher dimensional feature space so that the training data will be linear separable. Mathematically, if we rewrite the dual form and replace the term $x_i^T x_j$ with $k(x_i, x_j)$, we can obtain the following form.

$$max\{\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i.j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)\}$$

$$subject\ to\ \alpha_i \geq 0,\ \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Here, $k(x_i, x_j)$ is the kernel function. Different kernel functions can be used to solve different types of real problems. We list two common kernels as follows.

1. **Polynomial kernel**:$k(x_i, x_j) = (x_i \cdot x_j)^d$

2. **Gaussian kernel**: $k(x_i, x_j) = exp(-\gamma \parallel x_i - x_j \parallel)^2$

In this thesis, since it is not necessary to do the space transformation for textual documents, we will not introduce the kernel function in detail.

The advantages of using SVM as the classification algorithm are listed below.

1. **Effective and robust model**. By maximizing the margin, SVM has good generalization capability. With the soft margin introduced, SVM can be robust against the noise in the training set. By choosing the correct kernel and parameters, SVM can handle various types of data and large-scaled data sets.

2. **Efficient model construction**. By using dynamic programming, the optimization of model construction for SVM can be accomplished within a few iterations. There are several open libraries that implement SVM with slightly different approaches. The typical and most commonly used ones are *LibSVM* and *SVMLight*.

3. **Text classification**. Usually, the attribute (word) set for text (webpages and documents) is very large. The high dimensionality often leads to the training data being linear separable. Thus, linear SVM can be a good choice for text classification.

Several disadvantages can also be observed for SVM.

1. **Black box model**. SVM can do perfect job for classification, but it is difficult to interpret the model constructed by SVM, particularly for high dimensional data set.

2. **Choice of parameters**. There are several parameters in SVM that we need to specify in advance, such as C in the soft margin, and other parameters for different kernels. Inappropriate choice of the parameters may lead to poor classification performance.

Due to the advantages mentioned above, SVM is a good choice for our task. Furthermore, many previous works [27, 28, 5, 73] empirically verify that SVM is superior to other classification algorithms (Naive Bayes, KNN and Decision Tree) in terms of text classification. Thus, we will choose SVM, specifically linear SVM, as the base classification algorithm to construct the search engine with classified webpages. More specifically, we will use an open library of linear SVM called *LibLinear* (see Section 3.2.3.2 for detail) to implement our classification system.

2.3 Hierarchical Classification Approaches

To build SEE, we need to classify each new webpage into the categories in the hierarchy. One webpage may be classified into multiple categories which form a tree structure, since a webpage may belong to several topics in the same time. For example, a website talking about sales on crafts can be classified as "Shopping" and "Arts". It is different from the traditional binary or (multiclass) classification problem, where each new example is classified as one of the multiple classes. In our cases, the output of the model for an example is a hierarchical structure of multiple classes. How can we deal with this classification task? We will utilize a technique called *hierarchical classification* [16, 57, 61] to accomplish this task. In the following subsections, we will briefly introduce three classical approaches for hierarchical classification, as well as their weaknesses and strengths.

2.3.1 Flat Classification Approach

As the simplest strategy to handle hierarchical classification tasks, the flat classification approach [2, 24] completely ignores the hierarchical relations between categories (e.g., parent and child, siblings), and only classifies new examples into the leaf categories. A typical and commonly used method for this approach is called *one-vs-all*. Basically, it decomposes a complex classification problem with multiple classes into multiple binary classification problems on the leaf categories. The positive examples for the binary problem on a leaf category are examples on the category, while the negative examples are the examples belonging to all other leaf categories other than the current leaf category. Figure 2.6 illustrates the approach, from which we can see the positive examples on the leaf category "Music" are in the solid rectangle and the negative examples are in the dashed rectangle.

The training set can be used to build a binary classification model on each of the leaf categories. When a new example is coming, the model of each leaf category will generate a classification output (positive or negative), then the example will be classified into all the leaf categories with positive classifications. Since the categories of parent and children are of "IS-A" relation, once an examples is classified into a leaf category, it is implicitly classified into all its ancestor categories.

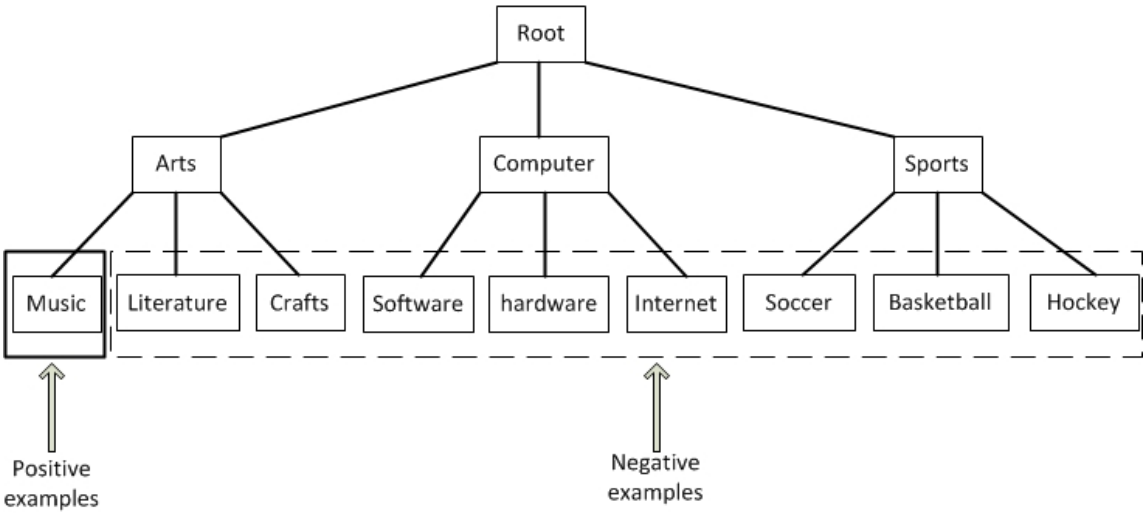

Figure 2.6: Illustration of flat classification approach.

The advantage of the flat classification approach is:

1. **Simple and intuitive**. By a simple transformation, the original complicated problem becomes a group of simplified binary tasks, which can be solved by traditional classification algorithms.

The disadvantages of the flat classification approach are primarily twofold.

1. **Imbalanced training sets**. The decomposed binary training set on each category will be very imbalanced, since the number of negative examples will be much larger than the number of positive examples. For example, in the hierarchy of SEE, we have 575 leaf

categories. For each leaf category, the positive examples come from one class (itself), while the negative examples consist of the other 574 classes. In this case, it is very difficult for most of the classification algorithms to perform well on the minority class, since simply predicting every example as negative can still lead to a high accuracy.

2. **High training cost**. Since the training set on every leaf category consists of all the examples on the leaf categories, it is very costly to train the models. The time complexity is $O(l \times n)$, where l is the number of leaf categories and n is the total number of training examples. For example, as shown in Table 3.2, we have 1,242,785 training examples in total, which equals the number of training examples on all the leaf categories. If we use the flat classification approach, we will need to train 575 models where each is built on more than one million examples. The massive size of data would cost a huge amount of memory, or even cannot be fit into the memory. Even if it can be loaded into the memory, it will still take a long time to execute the algorithm to construct the model.

2.3.2 Local Classifier Approach

To overcome the two disadvantages of the flat classification approach, the local classifier approach [65, 8, 71, 69, 17, 40] has been proposed. Instead of *globally* using the entire training examples to train the model on each leaf category, the local classifier approach *locally* uses the examples belonging to the parent category to train each model (All examples belong to the root). Figure 2.7 illustrates the idea of local classifier approach. For the category "Music", the positive examples on "Music" are still from "Music" itself, however, the negative examples only consist of the examples from the other subcategories under "Arts". Thus, the categories under "Computer" and "Sports" are excluded. Thus, we can see the whole training examples for each category are actually the positive examples from the training set of its parent category.

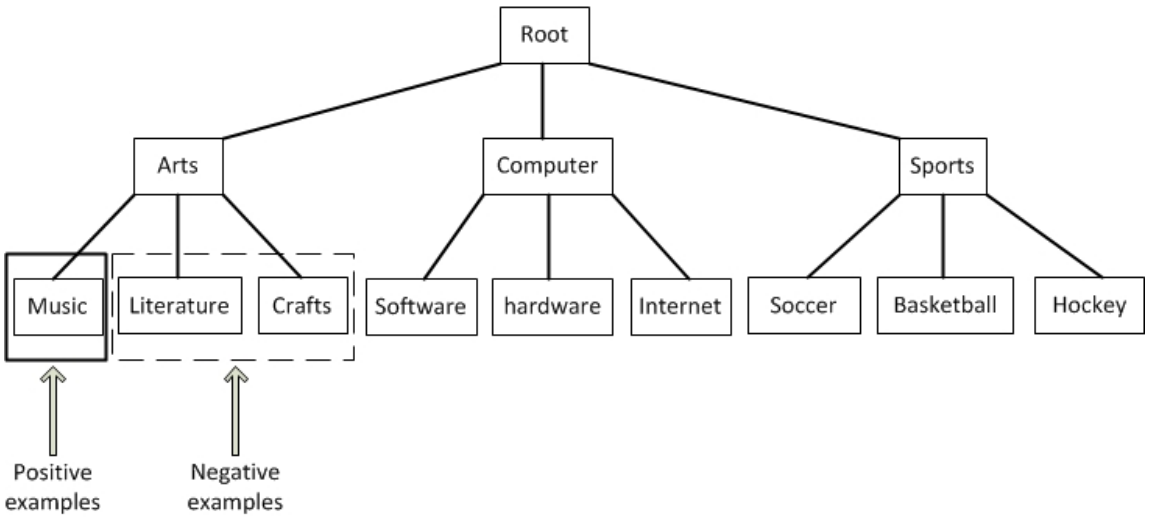

Figure 2.7: Illustration of local classifier approach.

By recursively localizing the training examples, we can generate the training set for each category of the hierarchy, except the root. Generally, the deeper the category, the smaller the

training set. Then the local classifier approach trains a binary classifier on each of the leaf and internal categories.

When a new example is coming, it adopts a top-down method to classify it into the categories of the hierarchy. The basic idea is that the descendant categories will have chances to classify an example only if all the ancestor categories classify it as positive. Specifically, first of all, all the classifiers on the top level categories[9] choose to predict the new example, and each one will output a classification (positive or negative). Then the example will be pushed down further to the subcategories of those top categories with positive classifications. This process iterates until it reaches the leaf category. Suppose we have a hierarchy as shown in Figure 2.7. Given a new example, if the classifier on "Arts" and "Sports" classifies it as negative, and the classifier on "Computer" classifies it as positive, then the example will stop being pushed down on "Arts" and "Sports" and be pushed down to all the subcategories of "Computer" for further classification. We will discuss how to classify new examples in detail in Section 3.2.4.

The advantages of the local classifier approach are:

1. **More balanced training sets**. Compared to the flat classification approach, the localized training set generated by local classifier approach is expected to have much less negative examples, and thus more balanced. It is definitely beneficial to the performance of the classification algorithms.

2. **Low training cost**. Since the training set on each category in the hierarchy is localized, the size of the training set decreases as the category goes deeper. Only the training sets on the top level categories use the entire training examples. It greatly reduces the training cost to construct the models, and speeds up the training process.

3. **Convenient model modification**. The localization of the training examples makes it convenient to add and delete new categories to the hierarchy. For examples, in Figure 2.7, if we would like to add a new category "Baseball" under "Sports", we only need to update the training sets on "Soccer", "Basketball", "Hockey" as well as "Sports", "Computer" and "Arts", but the subcategories under "Arts" and "Computer" can stay unchanged. When the examples of "Baseball" are included for training, the negative training examples of "Soccer", "Basketball", "Hockey", "Computer" and "Arts" as well as the positive training examples of "Sports" will need to be changed. However, it does not affect the training examples for the subcategories under "Arts" and "Computer", since they come from the positive examples of the parent categories.

The disadvantages of the local classifier approach are:

1. **Limited examples on deep categories**. The reduction of the training cost can also mean the reduction in training size, particularly on the leaf categories. Based on limited training examples, the classifier may not perform well on new examples.

2. **Error Propagation**. In the local classifier approach, a new example needs to be classified by the ancestor categories before reaching the deep category. It is possible that an ancestor classifier make mistakes on the example. The error will be propagated to all

[9]Top level categories are the children categories of the root

its subcategories. There are two types of errors that will be propagated, *false positive* and *false negative*. *False positive* is the misclassification of the actual negative example to positive. *False negative* is to misclassification of the actual positive example to negative. For example, in Figure 2.7, given a new example belonging to "Computer" → "Software", if the classifier on "Arts" makes a false positive error (misclassifies it as positive), then the example will be passed to all the subcategories of "Arts". However, since the classifiers on those subcategories are trained based on the examples belonging to "Arts", it is very likely that further errors will occur on this out-of-scope example. Besides, for the same example, the classifier on "Computer" may make a false negative error, meaning that it misclassifies it as negative on "Computer". In this case, the subcategory "Software" under "Computer" will not have the chance to classify it, thus further decreasing the overall classification performance.

2.3.3 Global Classifier Approach

In contrast to the flat and local approach, global classifier approach only builds one single but usually complex model for all categories, instead of multiple decomposed binary models. To classify new examples, the model can either directly output an entire hierarchy of categories, or still follow the top-down method used by the local classifier approach. There are various approaches [51, 56, 66] that can belong to the category of global classifier approach. Although they seem be share no core characteristic, in general two related characteristics are associated with them [57].

1. They consider the entire class hierarchy at once.

2. They lack the kind of modularity for local training of the classifier.

The key difference between global and local or flat approach is in the training phase (single model versus multiple models).

The advantage of the global classifier approach is:

1. **Time efficient**. By building only a single model rather than each model for all the categories in the hierarchy, the global classifier approach can significantly reduce the training time consumption.

The disadvantages of the global classifier approach are:

1. **Poor performance for complex hierarchy**. The performance of the global classifier approach may degrade dramatically as the number of categories in the hierarchy increases, since it is difficult for a single classifier to model the complicated relations among a large number of categories.

2. **Black box model**. Usually the model built by the global classifier approach is very complicated and hard to understand, which makes it difficult to analyze the model and thus further improve the model.

By considering the advantages and disadvantages of the three approaches introduced above, we decide to choose the local classifier approach, which is the most appropriate for our purpose. Because our total training size is over 1,000,000, we cannot afford the high training cost of the flat classification approach. Furthermore, with 662 categories our hierarchy is not that simple. Thus, the global classifier approach is less likely to perform well when modeling the complicated hierarchical relations. Overall, the local classifier approach is the best choice for our classification task, in terms of the training efficiency and effectiveness.

Chapter 3

Classification System

To implement the search engine SEE, we will need to create a hierarchy of topic categories and build classification models to categorize new webpages into our hierarchy. In this chapter, we will first introduce how we generate the hierarchy for SEE, and then discuss in detail how we preprocess the training data, design the parallel framework and maximize the performance of our classifiers. The evaluation results show that the constructed classifiers have good classification performance. This part is joint work with Xiao Li. We both contributed in designing the overall classification process. I contributed more in the hierarchy generation, performance maximization and performance evaluation, while Xiao contributed more in the data preprocessing, parallel framework design and most of the coding work in building the classification system.

3.1 Hierarchy and Training Data Generation

A reasonable and user-friendly hierarchy of topics is one of the most important elements in our search engine. It can greatly benefit the search experience of the users. There are several criteria that we think a good hierarchy should have for SEE.

1. **Common sense**. According to the statistics from Internet World Stats, by Dec 31, 2011, the total number of Internet users around the world has reached 360,985,492. Different users may have different education backgrounds, different expertise domains, different understanding even towards the same object. It is desirable to present a common sense hierarchy to various users so that majority of them will feel familiarized and convenient to find the desired topics. To satisfy various users, the hierarchy should also have a broad coverage of common topics.

2. **Tree structure with moderate width and depth**. Usually, tree-structured hierarchy is an acceptable form for users, since the tree structure has been around in many other domains (e.g., hierarchical menu, Windows directory). Besides, for the users' convenience, the hierarchy should not be too deep and the number of branches under each topic should not be too large. A complicated hierarchical structure (e.g., the long path from the root and the long list of subtopics) can easily confuse the users.

To generate our hierarchy, we can actually take advantage of some existing well-established taxonomies. There exist several taxonomies which have been already used by a large population of Internet users, such as the category structure in *Wikipedia* and the topic hierarchy in Open Directory Project (ODP) as we mentioned in Section 2.1.2.

However, the taxonomy in Wikipedia is a directed acyclic graph (*DAG*) rather than a tree-structured hierarchy. In fact, tree is a special structure of DAG which only allows one parent node for one node, but in DAG one node can have multiple parent node. For example, "Nature" is a category on the top level of the hierarchy (a subcategory of the root) as well as a subcategory under another top level category "Life". This type of topic structure will be confusing to the users, who get used to tree-based browsing. Another important reason that Wikipedia categories cannot be used in SEE is that the category relationship is not a strict "IS-A" relationship. For example, there is a connection path in Wikipedia: "Sports"→"Sport and politics"→"Olympics"→"Summer Olympic Games"→"Host cities of the Summer Olympic Games"→"Los Angeles". We can see, the city "Los Angeles" definitely does not belong to "Sports". The weak category relationship can not only mislead the users, but also increase the difficulty for learning algorithm to do accurate classifications.

The topic hierarchy of ODP is a tree structure and the relationship between a parent and a child basically follows the "IS-A" constraint. Thus, it can be a good base taxonomy for SEE. However, as we mentioned in Section 2.1.2, the entire hierarchy has over one million categories with a maximal depth of 14, which does not meet our criteria mentioned before. We need to extract some categories and customize them to be a reasonable hierarchy for SEE. The extraction follows the following rules.

1. Prefer popular and meaningful categories.

2. Prefer to choose the categories with more than 100 websites maintained under it.

3. Try to restrict the number of subcategories to be no more than 10 (moderate width).

4. Keep the depth of the hierarchy to be no more than 4 (moderate depth).

Based on the rules above, we extract 662 categories in total for our hierarchy. The top level has 11 topic categories, including *Arts*, *Business*, *Computer*, *Health*, *Home*, *Recreation*, *Reference*, *Science*, *Shopping*, *Society*, *Sports*. These 11 topics basically cover different domains in our regular life. We skipped the other five top categories in the original ODP, which has 16 top categories. We list the reasons why we did not select them in Table 3.1. The hierarchy has several leaf categories with a depth of four, however, most of the leaf categories are in the depth of three.

After we determine the hierarchy for SEE, we will need the training data to build the model for the classification of new webpages. Since a number of websites are maintained under each category in ODP, we can utilize the homepages of those websites as the training examples. Specifically, for a particular category in our hierarchy, we use the homepages of the websites belonging to all its descendants in ODP plus its own homepages as its training examples. For examples, for the top category *Health*, its training examples include the examples purely belonging to *Health*, such as the general health website *WebMD*, as well as the training examples in all the subcategories of *Health*, *Medicine*, *Conditions and Diseases*, *Addictions*, etc. Thus,

Category name	Reason to skip
Games	There is a similar subcategory under *Computer*.
Kids and Teens	It is just a kids version of the whole ODP structure.
News	It has too few websites (8000) to train the classifier.
Regional	The categorization is based on locations, thus infeasible to apply classifiers.
World	The webpages are in other languages, rather than English.

Table 3.1: The reason to skip five top categories.

Level	Number of Categories	Number of Example
1	11	1242785
2	99	31989
3	499	3941
4	53	2650

Table 3.2: The statistic information of the topic hierarchy at each level.

although we only extract four levels of categories in ODP, we basically use all the documents in ODP as our training examples except the documents in the five skipped top categories. We tabulate some simple statistics of our hierarchy on each level in Table 3.2.

3.2 Classification Methodology

After the hierarchy and training data are determined, we can build classifiers based on those training data by using the local classifier approach (introduced in Section 2.3.2). Then, those constructed classifiers can be utilized to classify new webpages into the hierarchy. In this section, we will discuss the detail on how we process the raw data, construct the classification model and categorize new webpages.

3.2.1 Data Preprocessing

As we mentioned in Section 3.1, we will utilize the webpages in ODP to build the classification model. However, since feature vectors are required for classifier training, those raw webpages (55.7 GB) cannot be used directly to train the classifier. We will apply a sequence of data preprocessing techniques (shown in Figure 3.1) to the raw webpages and output a qualified training set for the model construction.

Remove HTML tags. As we all know, webpages on the Internet consist of many HTML tags, such as <title>, <body>, <table> and so on. Since they have no meaning relevant to the

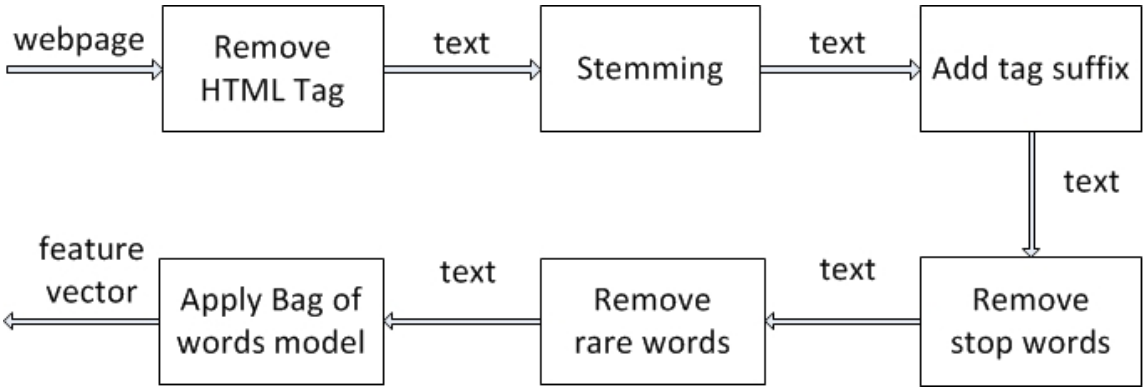

Figure 3.1: The sequence of data preprocessing.

topic of the webpage, introducing them as extra features (attributes) into the training set has little help to boost the performance of the classifier, instead the classifier may be misled by those irrelevant features. Thus, we remove all the HTML tags and scripts embedded in all the webpages, and only retain the text content. Therefore, the original webpages are converted into pure text documents.

Stemming. For the text documents, we choose to use a technique, called *stemming*. Stemming is a process to reduce the inflected words to their stem (or root) form. The intuition behind this is that words sharing the same morphological invariant (root) can be related to the same topic [58]. For example, "programming" and "programmer" will be stemmed as "program" and treated as the same feature. Stemming has been widely used in text classification, and demonstrated to improve the classification performance. Specifically, to stem a text document, for each word, we reduce it to its stem form (e.g., "programming" → "program").

After the HTML tag removal and stemming, we still need to handle a huge document collection with over four million unique words and 5.1 GB disk space. They cannot even be loaded into the memory of a normal PC for the further classifier training. Thus, we employ two techniques to further reduce the total number of words.

Remove rare words. We first remove the rare words over the entire collection. Specifically, we delete all the words that occur no more than three times (common choice in previous works [22]) in all the documents. Those rare words are of little value for the classification performance, since the classification patterns based on the rare words are unreliable.

Remove stop words. Furthermore, we delete all the *stop words* in the document collection. Stop words, such as "the", "is" ,"at" and "which", usually take a large space of a document, but do not carry any meaning. By removing the stop words, we can significantly decrease the storage space and memory usage, when training the classifier.

Add tag suffix. Although HTML tags are not semantic, different tags indicate different importance of the text content inside them, which can be utilized to benefit the topic classification. For example, a webpage with a word "software" in the tag <title> is more likely to belong to "computer" than a webpage with the word "software" simply appearing once in the tag <body>. There are also other helpful tags, such as description of <meta> and keywords of <meta>. However, as those information are missing in most of the webpages on the Internet, we treat the text in them the same as the text in the tag <body>. Based on this intuition, we

treat the same words in the tag <title> and in other tags of the webpage differently. Specifically, we append an suffix "_t" to each of the words in the <title> and "_b" to each of the words in other tags.

Apply *Bag of words* model. The next step is to transform each text document into a feature vector, which can be used as the input for the classification algorithm. We will utilize a typical feature representation model called *bag of words*, which is commonly used for text representation in document classification, natural language processing and information retrieval. The basic idea of this model is that we treat each text document as a collection of words but ignore the order of the words. In terms of how to represent the word collection in the bag of word model, there are different ways, including *binary format*, *occurrence format*, *TF format* and *TF-IDF format*.

For *binary format*, if a word appears in a document, the value of this feature (word) is 1; otherwise, it is 0. However, this format does not consider the weight of each word. For example, a document containing 100 times of "software" is much more likely to belong to "computer" than a document only mentioning "software" once.

To overcome this problem in binary format, *occurrence format* counts the number of occurrence of the words appearing in the document and use it as the value of the feature. The weakness of this format is that it does not take the length of the document into consideration. For example, the word "software" appears twice in a document with 5 words as well as in a document with 10000 words , but the shorter one should be more likely to belong to "computer".

By normalizing the document by its length, we can solve the problem in the occurrence format. This is the main idea of *TF format*, where TF represents term (word) frequency. It not only counts the number of occurrence of each word but also normalizes the occurrence by the length of the total words in the document. Mathematically, the TF of a word w in a document d can be defined as

$$tf(w,d) = \frac{|t \in d : t = w|}{|d|},$$

where t is any word in d. There is also an issue with the TF format. By using TF format, some common but non-discriminating words will have high weight. For example, the stop words such as "the", "and" are likely to appear frequently in every document but they are not able to distinguish different topics. In most documents under the category "computer", the word "computer" is likely to occur frequently, but it is also a non-discriminating word when we need to distinguish the topics under "computer" (e.g., "Software" vs. "Hardware"). Those highly weighted non-discriminating words will severely affect the classification performance.

Therefore, a more sophisticated format called *TF-IDF format* is proposed. In addition to TF, for each word, it also calculates the inverse document frequency (IDF). The inverse document frequency is to measure how common a word is across all documents in a collection. The more common a word, the lower the IDF. For example, the stop words (e.g., "the" and "and") will have very high value for IDF. To compute the IDF of a word w in a collection D, we can divide the number of documents by the number of documents containing this word, and then take the logarithm of the quotient.

$$idf(w,D) = log\frac{|D|}{|d \in D : w \in d|}$$

Document 1	Computer software is a subfield of computer.
Document 2	Computer hardware is different from computer software.
Document 3	Computer includes computer software and computer hardware.

	computer	software	is	a	subfield	of	hardware	different	from	includes	and
Document 1	$\frac{2}{7} \times log\frac{3}{3}$	$\frac{1}{7} \times log\frac{3}{3}$	$\frac{1}{7} \times log\frac{3}{2}$	$\frac{1}{7} \times log\frac{3}{1}$	$\frac{1}{7} \times log\frac{3}{1}$	$\frac{1}{7} \times log\frac{3}{1}$	0	0	0	0	0
Document 2	$\frac{2}{7} \times log\frac{3}{3}$	$\frac{1}{7} \times log\frac{3}{3}$	$\frac{1}{7} \times log\frac{3}{2}$	0	0	0	$\frac{1}{7} \times log\frac{3}{2}$	$\frac{1}{7} \times log\frac{3}{1}$	$\frac{1}{7} \times log\frac{3}{1}$	0	0
Document 3	$\frac{3}{7} \times log\frac{3}{3}$	$\frac{1}{7} \times log\frac{3}{3}$	0	0	0	0	$\frac{1}{7} \times log\frac{3}{2}$	0	0	$\frac{1}{7} \times log\frac{3}{1}$	$\frac{1}{7} \times log\frac{3}{1}$

Table 3.3: Illustration of TF-IDF format. In each cell, the left is the TF value while the right is the IDF value.

Then the TF-IDF value of a word w in document d as regard to a collection D can be calculated as

$$tf - idf(w, d, D) = tf(w, d) \times idf(w, D)$$

From the formula above, we can see, $tf-idf(w, d, F)$ can suppress the weight of those common but non-discriminating words. Table 3.3 presents a simple example to apply TF-IDF format to a collection of three documents.

Due to the superiority of TF-IDF format, we choose it as the feature representation for the text documents and generate the training set for the model construction.

3.2.2 Parallel Classification Framework

After the data preprocessing, each feature vector will be a training example, associated with several categories in the hierarchy. Based on the local classifier approach mentioned in Section 2.3.2, we can construct one binary training set on each category and build a binary classifier, and use top-down approach to classify new examples into the categories in the hierarchy.

Although in the data preprocessing we have reduced the feature space by removing the rare words and stop words, we still have over 1,277,386 unique features and 1,054,652 examples in each of the 11 training sets on the top level categories of the hierarchy. In addition, we need to build 662 binary classifiers for the whole hierarchy. The huge feature space, large training size and considerable number of classifiers significantly increase the time complexity for the model construction and classification process.

Therefore, we design a novel parallel computing framework to improve the efficiency of the training and classification process. Specifically, we build a small cluster with five PCs. Three PCs have eight CPU cores and the other two have four CPU cores. Thus, ideally, we can run 32 parallel threads simultaneously, but in fact only a maximum of 27 threads can be used taking away the system processes. All those PCs are connected to a Gigabit switch, so that the data transfer speed between them can reach as high as 100Mb per second. Then we configured a MPI (Message Passing Interface) environment on those PCs so that they can communicate with each other. Under this environment, a parallel algorithm is implemented to train the classifiers and make classification on the new examples.

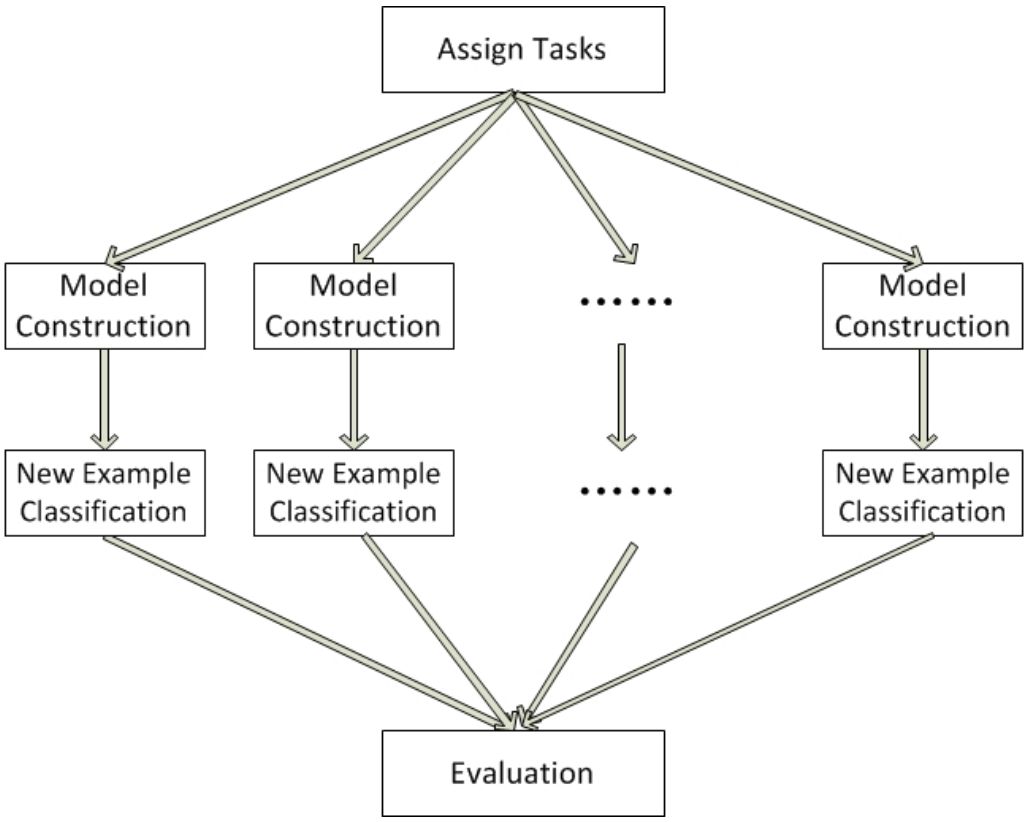

Figure 3.2: The process of the parallel framework.

Before the running of the program, we copy the preprocessed training set to each of the five PCs, so that the threads launched on each PC can locally load the large-scale data to the memory, rather than obtaining the data from a remote centralized data center. It can significantly boost the efficiency. Basically, we design a master and several slaves in the algorithm. Both master and slaves will be executed in a form of thread. The task of the master thread is to assign a number of model construction tasks (categories) to each of the slave threads, and gather the classification results from slaves and evaluate the classification performance. The task of each slave thread is to construct the binary models of the categories assigned by the master thread and classify the new examples into the hierarchy. All the slave threads can be executed simultaneously over the multiple CPU cores. To make sure the balance of the work load on each slave, the master adopts a level-wise strategy to randomly select the categories. Specifically, we maintain a list of categories for each of the four levels of the hierarchy. We repeatedly iterate each slave thread, and in each iteration the master pops one category from each list to assign to the slave if the list is not empty. The whole process terminates when no category is left in any of the four lists. Figure 3.2 demonstrates the framework of our parallel architecture.

3.2.3 Model Construction

For each slave thread, the first step is to build a binary model for each of the categories assigned by the master. There are three key factors in constructing each binary classifier (feature filtering, parameter tuning and probability calibration).

3.2.3.1 Feature Filtering

As we mentioned in the last section, the number of features in each binary training set is large, particularly for the top level categories. To further boost the efficiency of the model construction, we can utilize another typical technique in text classification, called *feature filtering*. The goal of feature filtering is to reduce the high dimension of input space without degrading the classification performance, so that the training phase becomes more efficient. In fact, well-selected features can even substantially improve classification performance.

For text classification, there are a number of feature filtering methods [74], such as Information Gain (IG), Chi Square (CHI), Document Frequency (DF), Bi-Normal Separation (BNS) [22] and so on. In our work, we only consider *DF* and *BNS*. The reason is that DF, IG and CHI are strongly correlated and usually obtain similar results [74], and DF is much simpler than the other two methods. Bi-Normal Separation has been recently proposed and shown to be very effective, particularly when data is class-imbalanced. The definition of the two methods (DF and BNS) are shown as follows.

1. **Document Frequency**. The DF of a word is simply measured by counting how many documents contain this word. Mathematically, given a word w, its DF value can be defined as
$$DF(w) = |d \in D : w \in d|,$$
where D is the collection of all the documents and d is any document in D.

2. **Bi-Normal Separation**. BNS uses the standard normal distribution's inverse cumulative probability function[1] F^{-1} to calculate the measure. Mathematically, it can be defined as
$$BNS(w) = |F^{-1}(\frac{tp(w)}{pos}) - F^{-1}(\frac{fp(w)}{neg})|,$$
where $tp(w)$ is the number of positive documents containing the w, $fp(w)$ is the number of negative documents containing w, pos is the total number of positive documents, and neg is the total number of negative documents.

By applying DF, we generally remove the rare words and retain the common words over all documents. The intuition behind the method is that choosing the common words can improve the chances that the features will be present in the new examples needed to be classified.

BNS takes the class labels (positive and negative) into consideration. It biases the discriminating words either for positive examples or negative examples. For example, if 80%

[1] Cumulative probability function calculates the probability that a random variable is less than or equal to a certain value under a probability distribution.

documents about "soccer" have the word "dribble", and none of the non-soccer documents contain "dribble", then the word "dribble" will have very high BNS value.

In our hierarchy, we have 662 binary training sets on different categories with various characteristics. For example, the top level training sets are generally large with vast number of features, while the leaf training sets are usually small with less features. Uniformly applying a single feature filtering method to all the categories in the hierarchy may not produce an optimal performance. Different feature filtering methods (DF and BNS) can fit different training sets. Thus, we will explore the best method for each training set and try to maximize the performance on each single category, thus optimizing the overall classification performance in the whole hierarchy.

3.2.3.2 Classifier Tuning

We will use linear SVM as our classification algorithm to classify the text documents. Specifically, an open library called *LibLinear* [19] will be utilized to train the classifiers. LibLinear is an optimized version of linear SVM [9] and very efficient for training large-scale problems. For example, it only takes a few seconds to train a binary classifier from the *RCV1* (Reuters Corpus Volume 1) dataset with more than 600,000 documents.

The optimization problem of LibLinear can be defined as

$$\min_{W,b,\xi}\{\frac{1}{2} \parallel W \parallel + C(w_+ \sum_{i=1}^{n_+} \xi_i + w_- \sum_{i=1}^{n_-} \xi_i)\},\ \textit{subject to},\ y_i(W \cdot x_i - b) \geq 1 - \xi_i,$$

where ξ_i measures the degree of misclassification of an example i and $w_+ + w_- = 1$. The formula above is actually transformed from the optimization formula defined in Section 2.2.4. As we discussed in Section 2.2.4, SVM with soft margin attempts to maximize the margin between the positive examples and the negative examples, while minimizing the errors on the training examples by multiplying each training error by a constant C. Generally, large C leads to fewer training errors and small margin, while small C results in more training errors and large margin. Since many real-world datasets are class-imbalanced (negative examples are much more than positive examples), LibLinear treats the training errors on the two classes differently by splitting C into two parts for positive errors and negative errors individually. w_+ is the weight for the positive errors and w_- is the weight for the negative errors. Usually, for class-imbalanced datasets, we need to put more weight on positive (minority class) errors to guarantee the classifier is consistent with the positive training examples. By doing so, the classifier is likely to make less errors to classify new positive examples.

Therefore, we need to specify two parameters before training a classifier, C and w_+ ($w_- = 1 - w_+$). Different combinations of the two parameters may lead to different classification performance. Thus, for each category in our hierarchy, we tune the two parameters of LibLinear, in order to obtain an optimal performance for the resulting classifier.

3.2.3.3 Probability Calibration

Only the hard categorization is insufficient to build our search engine SEE (we will explain it in the next chapter). The key output of the classifier for a new webpage is the probabilistic

likelihood (chance) of this webpage belonging to the categories in our hierarchy. For example, from the output of the classifiers, we know a new webpage has 90% chance belonging to "sports", 10% chance belonging to "arts", 30% chance belonging to "science", etc. Although LibLinear can produce a score for each example indicating how far it is from the classification boundary, the score is not in the range $[0, 1]$, and moreover not calibrated. Calibration is an important property for the score or probability produced by classifiers. Intuitively, if we say the probability outputted by a classifier is calibrated, then for all the examples to which a classifier assigns a probability 0.8 belonging to a category, then 80% of these examples should belong to that category.

There are two typical methods to transform the output (score) of classifier into calibrated probability, *Platt Scale* [48] and *Isotonic Regression* [75].

Platt Scale fits the training data to a sigmoid sharp as

$$P(y = 1|s) = \frac{1}{1 + e^{A \times s + B}},$$

where s is the original score produced by the classifier for a training example, and $P(y = 1|s)$ is the calibrated probability, which is 0 or 1 for the examples in the training set. Then gradient descent is used to find A and B. When a new example needs to be classified, we can calculate its calibrated probability by using the same formula.

Isotonic regression attempts to find a monotonically increasing function $\hat{M}$ that maps the original score s into a probability within $[0, 1]$. A typical algorithm for calculating the isotonic regression is pair-adjacent violators (PAV). PAV uses a stepwise-constant isotonic function to fit the training data according to a mean-squared error criterion.

For each classifier on a category, we apply both Platt Scale and isotonic regression, and find the best calibration method for each classifier that optimizes the classification performance.

3.2.3.4 Maximizing Performance

As we mentioned in Section 3.2.3, there are three major factors that significantly affect the performance of a classifier: feature filtering, classifier parameters and probability calibration. For the feature filtering methods, we consider DF and BNS. For classifier parameters, we set $C \in 1, 10, 100, 1000$ and $w_+ \in 0.5, 0.7, 0.95$. For probability calibration methods, we consider Platt scaling and isotonic regression. We design a novel algorithm to find the best combination of the three factors on each of the categories in the hierarchy. We attempt to find the optimal combination that maximizes classification performance (e.g., F1-score) on each of the categories in the hierarchy by using *3-fold cross validation*. The detailed algorithm is demonstrated in Algorithm 1.

N-fold cross validation is a standard approach to evaluate the performance of a learning algorithm. Basically, we split the training set into N parts. Usually, we stratify the training set so that each part has the same class distribution. Each time we choose $N - 1$ parts to train a classifier and the other part is used to evaluate the classifier since we know the class label, and we repeat the process N times. Finally, we combine the evaluation results over the N repeats.

By executing the algorithm above, we can obtain the optimal $\{f_{best}, C_{best}, w_{+_{best}}, cal_{best}\}$ for each category. Then we train each classifier with its best parameters, which can be used to classify new webpages.

Algorithm 1
Input: Training set T_i on each of the categories
Output: The best combination of $\{f_{best}, C_{best}, w_{+_{best}}, cal_{best}\}$ for each category, where f is the feature filtering method and *cal* is the calibration method.

```
for i ∈ [1, 662] do
    MaxP_i = 0
    for f ∈ {DF, BNS} do
        Apply f on train_i and get a reduced set reduced_i
        for C ∈ {1, 10, 100, 1000} do
            for w_+ ∈ {0.5, 0.7, 0.95} do
                for cal ∈ {Platts scaling, isotonic regression} do
                    use 3-fold cross validation on reduced_i to calculate the performance
                    measure P_i, where C and w_+ are used to train the classifier
                    if P_i > MaxP_i then
                        {f_best, C_best, w_+best, cal_best} = {f, C, w_+, cal}
                        MaxP_i = P_i
                    end
                end
            end
        end
    end
end
```

3.2.4 Document Categorization

After the model construction, we can use the "best" classifiers to categorize new webpages (examples) into the hierarchy in a top-down fashion.

Basically, we set a truncation threshold for each of the categories in the hierarchy, say 0.5. When a new example is coming, we first let each of the classifiers in the top level of the hierarchy output a probability for the example. If the probability is greater than the threshold on a category, then the example is classified as belonging to the category, and we further pass the example to the classifiers of all the subcategories of that category; otherwise, the example is classified as not belonging to the category as well as all the descendant categories of that category, and thus we stop passing down the example to the subcategories of that category. This process iterates until all categories are considered. After the categorization, for each new webpage, we will know which categories it is classified into. The detailed pseudo code is presented in Algorithm 2. From the algorithm, we can see the outputted set C contains the categories that a new page is classified into.

3.3 Classification Evaluation

In this section, we will present the evaluation of our classification performance. Firstly, we will introduce several measures utilized in our evaluation and then demonstrate the experimental results as well as the analysis.

Algorithm 2
Input: Given a new example e, a truncation threshold t and a hierarchy H; **Output:** The category set C in H that e is classified to;
$queue$={}
C={}
topCats = subcategories of the root of H
for $c \in topCats$ **do**
  $queue \leftarrow c$
end
while *queue is not empty* **do**
  $c = queue.pop()$
  let the classifier on c produce the likelihood probability $p_c(e)$ for e
  if $p_c(e) \geq t$ **then**
    $C \leftarrow c$
    for $child \in c.subcats$ **do**
      $queue \leftarrow child$
    end
  end
end

3.3.1 Evaluation Measure

Since most of the 662 binary datasets in our hierarchy are class-imbalanced, we can still utilize *recall* and *precision* (introduced in Section 1.2.1), which are particularly effective when the two classes are imbalanced. The traditional measures such as *accuracy* and *error rate* are not suitable for the class-imbalanced dataset. For example, to simply classify all examples as negative in a dataset consisting of 990 negative examples and 10 positive examples, the classifier can achieve an accuracy as high as 99%. However, this classifier is actually useless.

The definition for *recall* and *precision* for classification evaluation is slightly different from that in information retrieval. Here, we always assume positive class is the minority class.

Recall and Precision Given a set of positive examples P and a set of negative examples N, for all the examples in P and N, a classifier classifies a set of examples T as positive and a set of examples F as negative.

$$recall = \frac{|T \cap P|}{P}$$

$$precision = \frac{|T \cap P|}{T}$$

Generally, high recall and high precision cannot be achieved at the same time. To have a high recall, the classifier needs to predict more examples as positive, where the misclassified ones will decrease the precision. Thus, in order to evaluate the overall performance of the classifiers, we utilize another measure named *F1-score*, which is a combinatorial measure of recall and precision. It ranges from 0 to 1. Mathematically, it can be defined as

$$F1 = \frac{recall + presision}{2 \times recall \times precision}.$$

Level	Number of Categories	Average Positive Class Ratio	Average F1-Score
1	11	0.2156	0.792156
2	99	0.1137	0.827007
3	499	0.1053	0.820650
4	53	0.1030	0.841153

Table 3.4: The average F1-Score over the categories at each level.

As reported in many previous works [40, 16] for text classification, the F1-score cannot reach very high value (usually 0.8 is a pretty high value), particularly when the class is imbalanced.

3.3.2 Results and Analysis

In Section 3.2.3.4, we obtained the best performance (F1-score) for each category and we report some statistics of the performance. It should be noted that the predictive results presented are obtained by using 3-fold cross validation on the training data rather than on the new webpages from the Internet. Therefore, the results reported here may not accurately reflect the real situation. The reason we do not evaluation on new webpages is that new webpages on the Internet have no class labels and it is difficult to evaluate the performance on them. However, the diversity of the training set (webpages from various topics in ODP) can still make our training data approximate the distribution of the Internet webpages. Besides, the large number of training examples can also avoid overfitting of the classifiers. Thus, the results reported here can still be regarded as an important indication for the classification performance on new Internet webpages.

Table 3.4 presents the average F1-score over the categories at each level of the hierarchy. We can see the lowest average F1-score (the first level) of the four levels is close to 0.8. The classification performance is pretty impressive, since the class is imbalanced (with the average positive class ratio from 0.1 to 0.2).

In order to observe the performance of each individual classifier, we also show the distribution of the F1-score of different categories in each level in Figure 3.3. In the X-axis, we split $[0, 1]$ into 10 ranges and the Y-axis represents the percentage of categories in each level having the F1-score falling into those ranges. For example, for all the 499 categories in level 3, 33% have F1-score within $[0.9, 1]$, 33% within $[0.8, 0.9)$, and only about 10% within $[0.6, 0.7)$, etc. We can discover that for all the four levels, most of the categories have their F1-score higher than 0.7, and very few categories perform poorly (with F1-score less than 0.6).

Thus, by optimizing the classification performance, the classifiers are expected to achieve reasonable results to classify new webpages.

Furthermore, we find some other interesting results for the optimized parameters. Figure 3.4 plots the distribution of the feature filtering method in the optimized parameters. It can be observed that DF is selected as the best feature filtering method for most of the categories in the hierarchy. Furthermore, for the first two levels, none of the categories choose BNS. As

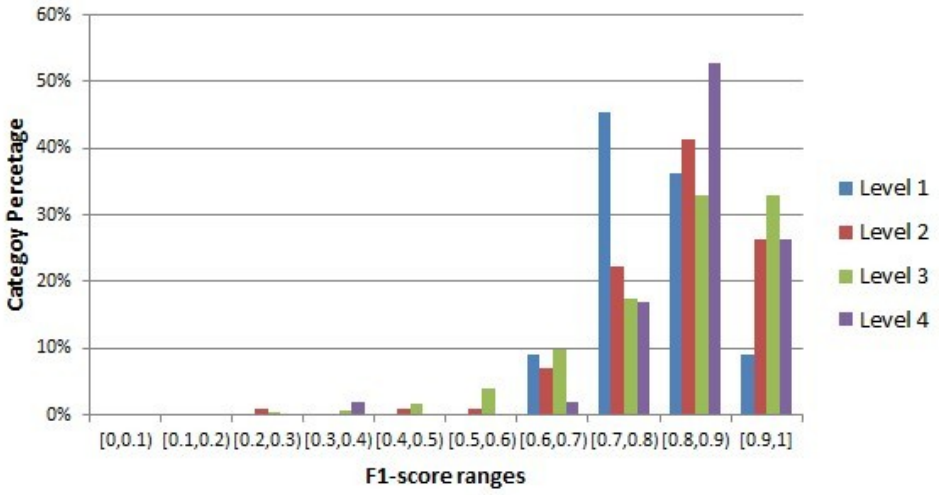

Figure 3.3: The distribution of F1-score over different categories.

discovered in our experiments, the tendency of BNS to select rare words can greatly reduce the size of the training data, which negatively affects the classification performance when the original data is large. For example, the training sets in the first level generally have a large number of examples with more than one million features. One original training set in the first level can usually take 300MB disk space. By applying BNS to reduce the features to 10%, we can decrease the needed disk space to an extremely small size (say 5MB). However, many of the reduced examples contain no features and become useless for training the classifier, which significantly degrades the classification performance.

Figure 3.5 demonstrates the distribution of the best calibration approach over different categories. It is clear that isotonic regression dominates Platt scaling in all the four levels. However, there is a trend that Platt scaling become more and more popular as the level goes deeper. In our experiment, we found that the step-wise isotonic regression is more likely to produce calibrated probability when the number of training examples is large, while Platt scaling is less sensitive to the data size since it fits the data to a sigmoid sharp. It is the reason that the deep categories (with less training examples) prefer Platt scaling as the calibration approach to isotonic regression.

Figure 3.6 and Figure 3.7 draw the distribution of the two parameters (C and w_+) for LibLinear. Generally, the higher the level, the smaller the C and the larger the w_+. C is used to control the fitting of the model with the training data. Usually, the larger the C, the better the model fits the training data. For the categories in the higher levels, the training sets are often large and large C can easily cause the model to overfit the training data, thus high-level categories favor the smaller C (e.g., $C = 1$). We can use w_+ to balance the fitting of positive and negative training examples. Higher w_+ will lead to high penalty when a positive example is inconsistent with the model. In the high level, because we allow the inconsistency between the model and the training data by choosing a smaller C, we hope positive examples to be more consistent with the model since the positive (rare) class is often more important in class-imbalanced problems. However, in the low-level categories, more consistency between the model and the (smaller) training data can usually result in high classification performance. In

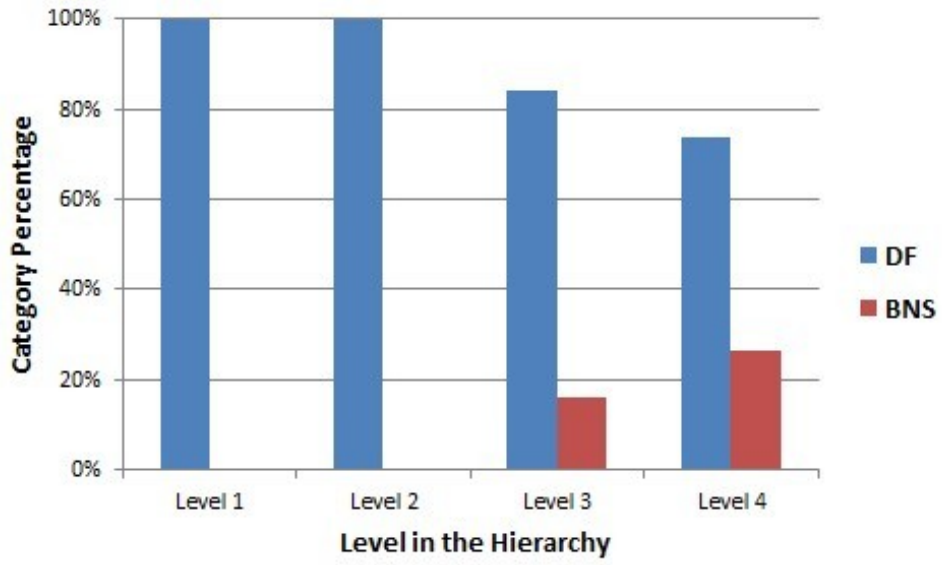

Figure 3.4: The distribution of best feature filter over different categories.

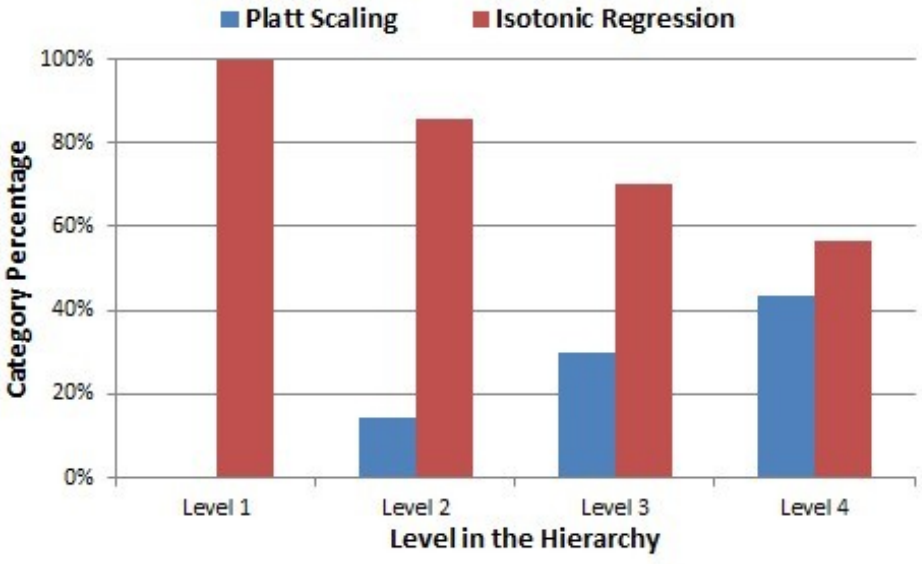

Figure 3.5: The distribution of best calibration method over different categories.

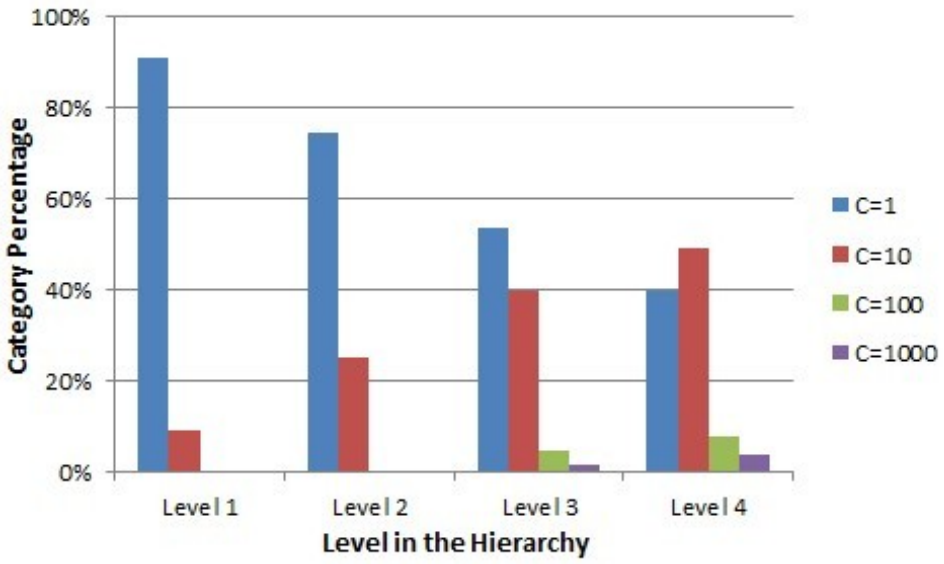

Figure 3.6: The distribution of best parameter C over different categories.

this case, w_+ actually takes less effect, since there are fewer training inconsistencies for both positive and negative class. It is the reason that the higher the level, the larger the w_+ is chosen.

3.4 Summary

In this chapter, we discussed how we choose the proper hierarchy, generate the training data, process the raw data, construct the optimized classifiers and categorize new documents. The evaluation results demonstrate that the performance produced by our classifiers is satisfactory.

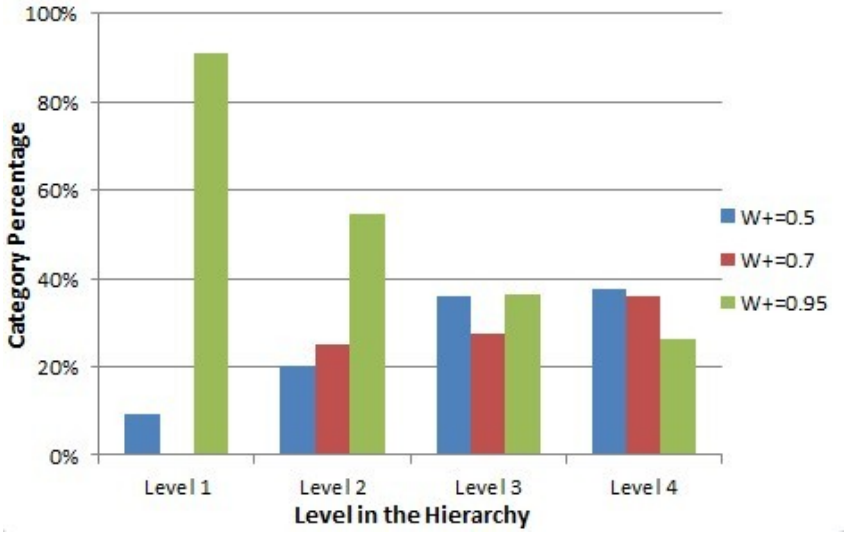

Figure 3.7: The distribution of best parameter w_+ over different categories.

Chapter 4

Search Engine Prototype

In this chapter, we will discuss how we build the novel search engine (SEE) by utilizing the classification results obtained in the last chapter. Firstly, we address two key issues in designing SEE and propose two novel solutions to solve them respectively. Then, we introduce how we develop the entire system from the implementation level. Finally, we conduct a comprehensive evaluation for SEE compared with the traditional flat search engine. This part is also joint work with Xiao Li. We both contributed in designing the overall system architecture. I contributed more in designing the ranking function and the new interface to explore additional results as well as the evaluation for SEE, while Xiao contributed more in designing the system to crawl the webpages from Internet and querying PageRank from Google API.

4.1 Ranking Function Design

An important aspect of modern web search engines is to rank the returned results when a query is issued by the user. The simplest ranking function is to sort the results by their relevance to the keywords in the query. The most common approach to calculate the *query relevance score* is to compute the similarity between the query and each indexed document by using vector space model, which was introduced in Section 1.1.1.

However, this simple ranking function does not consider the popularity (or importance) of the webpages, which plays a significant role in web searching. For example, it is common that students would like to visit an official website of a university, say Western university. When a query "Western University" is issued to a search engine, it is very likely that many documents match the query equally well. In this case, how can we guarantee the official website to appear in the top of the returned results? By including a popularity score into the ranking function, the official website that has a much higher weight would generally be top-ranked. There are several algorithms [29, 47, 23] proposed to calculate the *popularity score*. Among them, the most typical one is PageRank [47], which is a link-based algorithm. The PageRank score of a webpage is defined recursively as a function of the PageRank scores of all the webpages that link to it. A webpage that is linked to by many popular pages with high PageRank score can also receive a high score. For example, if a page has links from `www.yahoo.com`, `www.facebook.com` and `www.amazon.com`, it can be expected that this page would have a high score of PageRank. Most of the existing web search engines (e.g., Google and Bing) have

already included the popularity score into their ranking algorithms. An intuitive but effective way to integrate the popularity score is to multiply the query relevance score with the popularity score. In this thesis, we will use PageRank as the popularity score.

To rank the returned results in SEE, the two scores mentioned above are not enough. In SEE, when the user issues a query within a particular category (e.g., "sports"), we need to return all the relevant documents categorized into "sports" and also rank those results to display to the user. As the classifier is not 100% accurate, one type of misclassification, false positive, can occur. For a specific category, false positive means that the documents that actually do not belong to it are actually categorized into it. It would be annoying and confusing if those false positive documents are displayed and even top-ranked to the user. Thus, to rank the results in SEE, we should take this case into consideration. We take advantage of the classification probability of each document belonging to a particular category.

Usually, the false positive documents are the boundary cases, where the classifier is likely to make mistakes. Their probabilities belonging to the category may be just above the truncation threshold as mentioned in Section 3.2.4. If the threshold is 0.5, then a false positive document may probably have a probability such as 0.51. In contrast, the documents that actually belong to a category are usually assigned a high probability far from 0.5 such as 0.9. Based on this intuition, we can directly use the classification probability as an additional score to rank the returned results within a category. By doing so, we hope to lower down the rank of those false positive documents, so that they are less likely to appear in the top of the returned results.

Due to the error propagation of the local classifier approach we use for hierarchical classification, the false positive documents on a category can further be misclassified into its subcategories. Thus, the likelihood of a document belonging to a category relies on the joint probability of all the categories along the path from the root. Suppose we have two documents ($D1$ and $D2$) on "sports"→"soccer". For $D1$, $Pr(sports|D1) = 0.9$ and $Pr(soccer|sports, D1) = 0.51$, while for $D2$ $Pr(sports|D2) = 0.51$ and $Pr(soccer|sports, D2) = 0.51$. It is apparent that $D2$ is much less likely to belong to "soccer" than $D1$, since with the chance of 0.51, $D2$ is probably unrelated to "sports".

Therefore, we introduce an additional score for our ranking called *category score*.

Category score Given a webpage w and a category c, the category score of w associated to c is defined as,

$$score_C(c, w) = CPr(c, w) \times \prod_{c' \in \Uparrow c} CPr(c', w), \tag{4.1}$$

where $\Uparrow c$ is the set of the ancestor categories of c, and $CPr(c, w)$ is the conditional probability on how likely w belongs to c in the condition that w belongs to all the ancestor categories of c.

Simply speaking, the category score of a document on a category is the multiplication of the probabilities on all the categories along the path from the root to that category in the hierarchy.

Based on the category score, we further define our ranking function as a function of three scores (query relevance score, popularity score and category score).

Ranking score Given a query q, a webpage w and a category c, the ranking score of w related to q on category c is defined as,

$$score_R(q, w, c) = score_Q(q, w) \times score_P(w) \times score_C(c, w), \tag{4.2}$$

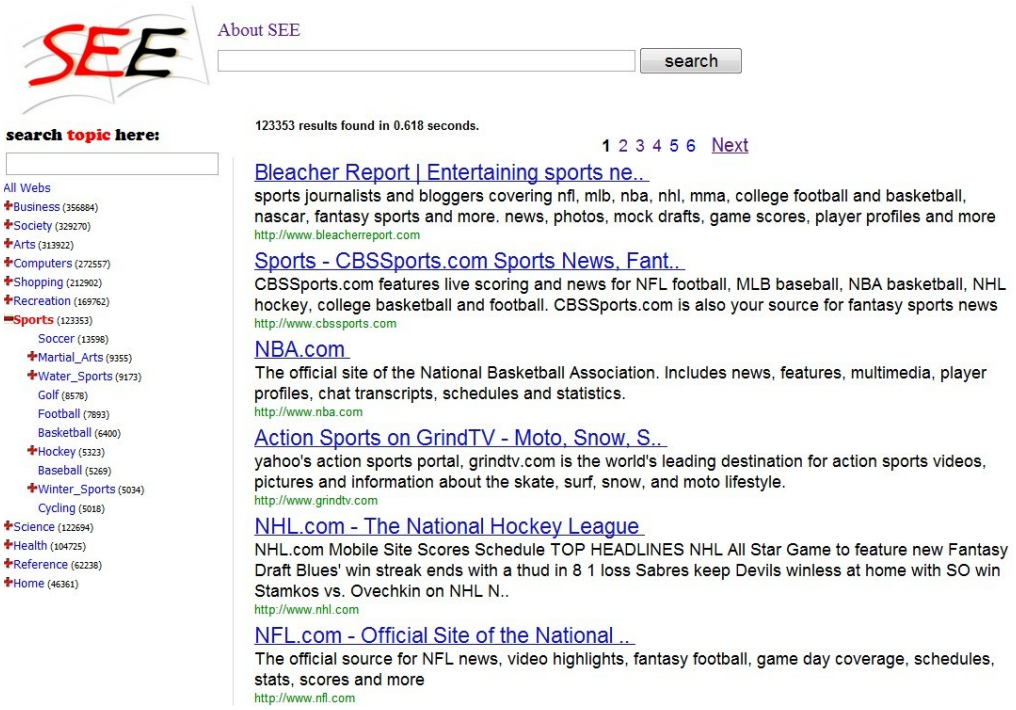

Figure 4.1: Browsing "sports" category without any keywords.

where $score_Q(q, w)$ is the query relevance score, $score_P(w)$ is the popularity score and $score_C(c, w)$ is the category score.

Sometimes, when the scales of the three scores greatly differ from each other, we may need to re-scale some of the scores.

By using the ranking function above, we can introduce an important and novel feature to our search engine SEE. If users simply browse a category without any keyword, $score_Q(q, w)$ is constant for all webpages. SEE will return all the indexed webpages classified into this category with the most popular and most category-relevant ones at the top. In this case, SEE simply becomes a web directory, and people can browse the most popular webpages in each topic category of the whole hierarchy. Figure 4.1 shows the first page of the returned results when the user simply browses the category "sports" in SEE. We can see that the most popular sports websites, such as `www.nba.com`, `www.nhl.com` and `www.nfl.com`, are displayed to the user.

The feature provides an alternative way for users to explore unfamiliar topics and areas. It is common that people with expertise in one area know nothing about another area. Suppose a musician suddenly becomes interested in programming, and would like to find some useful information on the Internet. However, the unfamiliarity to programming would prevent him coming up with useful keywords. Thus, the traditional keyword-based search engine may not work well in this case. If the musician switches to use SEE, he can first click into the category "computer"→"programming" without any keywords. The most popular websites displayed would provide an overview and guidelines to programming, and then he can further explore the area with some specific keywords.

4.2 Exploring Additional Results

The ranking function we proposed actually deals with one type of misclassification errors, false positive. How does the other type of misclassification errors, false negative, affect our search engine? How can we handle it effectively? In this section, we will address those issues, discuss three potential methods and propose a novel solution.

For a specific category, false negative means that the documents that actually belong to the category are not classified into it. Therefore, the users of the search engine cannot find those false negative documents in the category as well as in all its descendent categories, no matter what keywords they use. For instance, the classifier on the category "sports" produces a probability 0.49 on a document that really belongs to "sports". If we set the truncation threshold to be 0.5, then the document will not be classified into "sports". Furthermore, the document would not be classified into all the descendent categories of "sports". Even if the document also belongs to "soccer", the classifier on "soccer" will not have the chance to be able to classify this document, since it has already been truncated in "sports".

To deal with this problem, we will discuss three heuristic solutions: *no threshold*, *flexible threshold* and *probability intervals*.

No threshold is to remove the truncation on the probability by using threshold, so that for each category we will not have any false negative documents. Hence, in each category, the searchable documents are the entire indexed documents and the user will never miss any useful results when searching. If the user simply browses without any keyword, no matter what category is browsed, SEE will return the same amount of documents (all the indexed documents). However, the ranking of the returned results in different categories varies. For example, if we browse the category "sports" and "shopping", the number of returned results is the same. However, in "sports", `www.nba.com` may be at the top of the returned list, but in "shopping" `www.amazon.com` may be top-ranked. The problem with this solution is that it may introduce a lot of noisy documents when users search within a specific category. Since all indexed documents are available to match the query issued by the user in each category, it is likely that some category-unrelated documents have high query relevance scores and thus are top-ranked. These noisy documents will severely degrade the quality of the search results. Thus, this solution is not a reasonable choice.

Flexible threshold still adopts the threshold to truncate the documents within each category, but allows the user to lower down the threshold for more results. If the original threshold is t in category C, the documents with probability within $[t, 1]$ can be found by the users. If we lower down the threshold to t', now the users can find more documents with probability within $[t', 1]$. By decreasing the threshold, some original false negative examples will be classified as true positive. Suppose the classifier on "sports" outputs that `www.nba.com` has a probability 0.45 belonging to "sports". If the threshold is set to 0.5, `www.nba.com` will not be classified as "sports", and thus a false negative. If the threshold can be adjusted a bit lower, say 0.4, `www.nba.com` will be classified as positive in "sports", and the users are able to find `www.nba.com` in the category "sports". However, this approach has a serious problem. That is, although more documents are classified as positive, they may not easily be found by the users as they are likely to have low ranks. According to our ranking function, the order of the results is determined by query relevance score, popularity score and category score. Since the newly included documents (within $[t', t)$) have low category scores and are mixed with the original

documents (within $[t, 1]$), they are usually lowly ranked. Unless they are highly relevant to the query or extremely popular, it is very likely that no changes can be seen within the first few pages after the user lowers down the threshold. The problem will make our search engine unfriendly to the search experience of the users. Therefore, we will not adopt this solution.

Probability intervals offers three probability intervals $[0.5, 1]$, $[0.3, 0.5)$ and $[0.1, 0.3)$ for the users to explore. Initially, when a user logs onto SEE and searches in any specific category, only the documents with probability within $[0.5, 1]$ that match the query will be returned. When the user reaches the last page of the returned results, we design a new interface and display a button named "Additional Results" to the user. The button can be clicked twice, and accordingly the documents with the probability falling into $[0.3, 0.5)$ and $[0.1, 0.3)$ will be returned respectively. This solution is more intuitive and friendly to the users' search experience. Usually the reason that a user would like to explore additional results is that the total number of returned results is small and all the results cannot meet the need of the user. In this case, it is reasonable to let the user explore more results in the last page of the returned results. Besides, replacing the documents within one probability interval with another can solve the problem associated with the solution *flexible threshold*. The documents in the new interval (e.g., $[0.3, 0.5)$) will not mixed with the original interval (e.g., $[0.5, 1]$), and thus the new documents can have good ranking and easily be discovered by the user. One possible drawback of this solution is that completely replacing the documents within one probability interval with another can cause the loss of the original documents. Fortunately, since the user has already come to the last page, it is not that meaningful to revisit these already-checked documents.

Besides, for both solutions (flexible threshold and probability intervals), the decrease in threshold will result in extra false positives. For instance, when a user searches in the category "sports", a website `www.music.com` with 0.35 probability belonging to "sports" may be returned, if the threshold is set to 0.3 or the probability interval is changed to $[0.3, 0.5)$. It will introduce extra noise to the search results. However, in order to find the desired results, we think it is worthwhile to provide such functionality to the users. Based on the benefits of *probability intervals* over the other two methods, we will choose it for users to explore additional search results.

Figure 4.2 demonstrates how *probability intervals* works as people click the "Addition Results" button in a real case. Suppose a user would like to find job information about "software engineer" in Boston, he/she may issue a query like "software engineer boston" in the "business"→"employment"→"job search". However, only 11 results are returned and none of them are relevant as shown in the top subgraph of Figure 4.2. It means that only 11 indexed documents have their probabilities within $[0.5, 1]$ belonging to "business", "employment" and "job search", and also match the query "software engineer boston". At this time, he/she can click the "Addition Results" on the second (last) page (see the middle subgraph of Figure 4.2). Finally, although only one more result is found, it is highly relevant and useful to the user (see the bottom subgraph of Figure 4.2). It is clear that the probability of this result belonging to "job search" is within $[0.3, 0.5)$ and thus it is a false negative initially. When the "Addition Results" button is clicked, the interval $[0.3, 0.5)$ completely replaces $[0.5, 1]$, and thus this result is returned to the user. It should be noted that we just offer a way for users to find those misclassified results that match the query and cannot guarantee that users can always find relevant results. If in an interval there is no documents satisfying this condition, no documents will be returned to the user. Even if some results are returned, they may not be needed by the user.

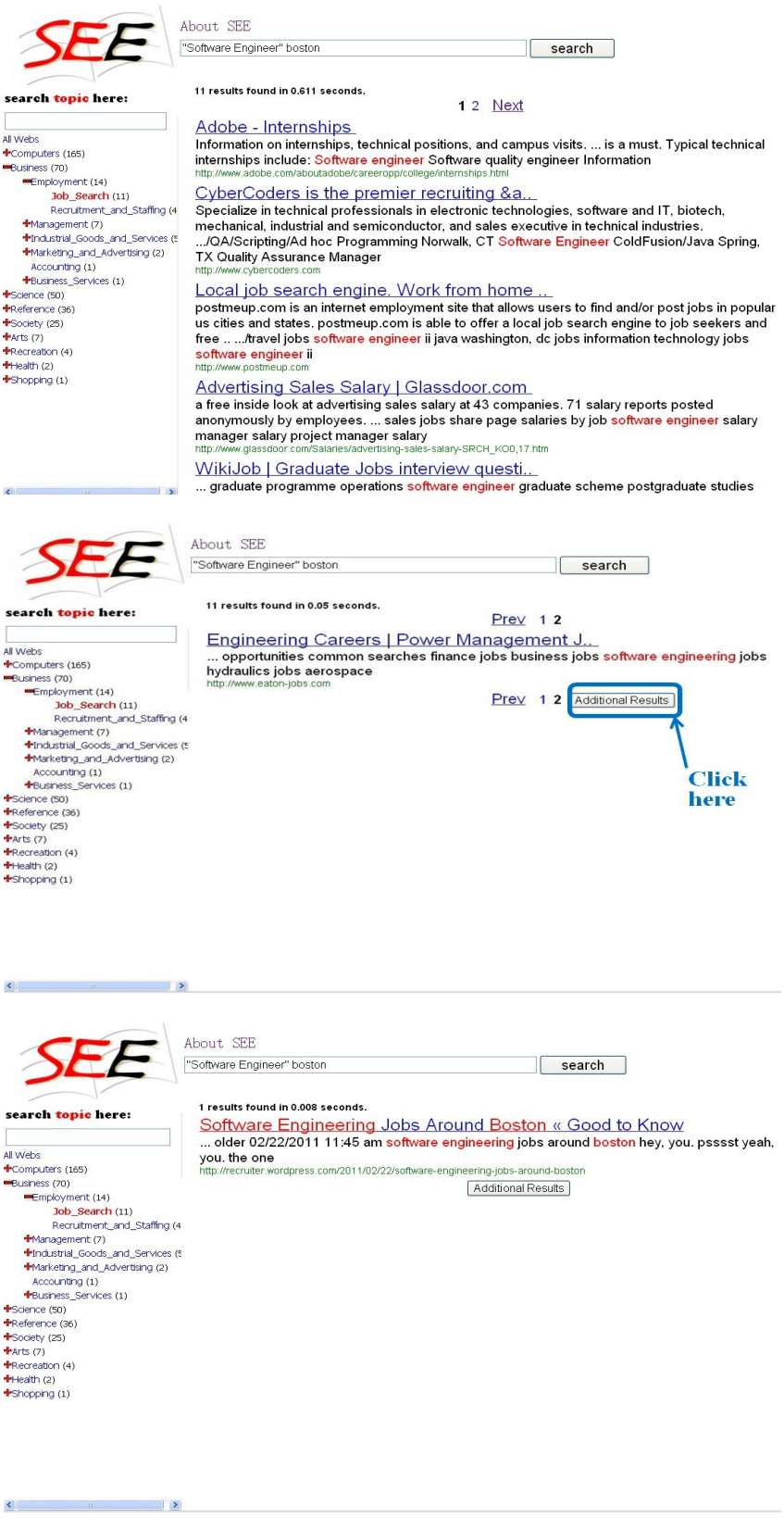

Figure 4.2: Illustration on how *probability intervals* works as people click the "Addition Results" button. In this case, a user would like to find job information about "software engineer" in Boston.

4.3 System Implementation

In this section, we will discuss the implementation detail of SEE, including the design and implementation of different components in SEE, data flow between different components and how we integrate the ranking function and additional results exploration into the system.

In the last chapter, we have already discussed how we construct the classification system for the whole system. Once the classification system is ready, we can start the crawling system to download the webpages on the Internet, and then use the classification system to automatically categorize them. After the categorization, for each webpage, a list of 662 probabilities will be generated, and each probability represents the conditional probability of this webpage belonging to a category in the hierarchy. Each webpage together with the 662 probabilities will be processed in the indexing system of SEE.

Since SEE is just a prototype of a new search engine rather than a commercial search engine, we did not conduct a large scale web crawling over the whole Internet. Most of the indexed webpages are crawled from some public data collections and their PageRank is obtained by querying Google APIs. In the following parts, we will skip the crawling system, but focus on the indexing and search server, user interface and web server.

4.3.1 Indexing and Search Server

We utilize an open source search server called *Apache Solr*[1] to index the webpages into SEE.

> "Solr is the popular, blazing fast open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest Internet sites. Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Solr's powerful external configuration allows it to be tailored to almost any type of application without Java coding, and it has an extensive plugin architecture when more advanced customization is required."[59]

For each crawled HTML webpage, we index the url and text in the tag <title>, <keywords>, <description> and <body>, all the 662 probabilities and the PageRank score into the Apache Solr indexing system. Apache Solr provides a set of HTTP APIs (JSON and XML) to process the indexed documents, such as search, delete and addition. The most commonly used API is the search API that is used to receive query request and response the search results. For example, if we would like to search "music" and get the results in JSON, the search API can be accessed by `http://kdd.csd.uwo.ca:88/see/result?q=music&wt=json`, where `http://kdd.csd.uwo.ca:88` is the host address, `/see` is the web application, `/result` is

[1] `lucene.apache.org/solr/`

Parameter name	Description
q	keywords of the query
start	the page index of the complete returned results
rows	the number of results in each result page
hl	whether to highlight the matched keywords
sort	the field used to sort the returned results
fq	query on a specific field
wt	format of returned results

Table 4.1: Common parameters for the search API in Apache Solr.

the search API, q is a parameter specifying the keywords of the query and *wt* is another parameter specifying the format of the returned results. Table 4.1 tabulates some common parameters for the search API.

Upon receiving a search request, Apache Solr searches q in all indexed text field (e.g., url, title, keywords, description, body) by setting different (integer) weights for different fields. By considering the importance of different fields in searching, we set $weight_{url} = 1$, $weight_{title} = 10$, $weight_{keywords} = 5$, $weight_{description} = 5$ and $weight_{body} = 2$. To calculate the query relevance score, Apache Solr uses the following formula.

$$score_Q(q,d) = coord(q,d) \times queryNorm(q) \times \sum_{t \in q}(tf(t \in d) \times idf(t)^2), \tag{4.3}$$

where $coord(q,d)$ is a score factor based on how many of the query terms are found in the specified document, $queryNorm(q)$ is a normalizing factor used to make scores between queries comparable, $tf(t \in d)$ is the term's frequency in document d, and $idf(t)$ stands for inverse document frequency. According to the ranking function proposed in Section 4.1, we need to combine popularity score and category score with the query relevance score. Since we already indexed the other two scores, we can easily use them to generate the final ranking score by leveraging the boosting mechanism and the function query in Apache Solr. The boosting mechanism can be realized by rewritting the parameter q in the search API. Suppose we would like to search "music" in the category "arts" and rank the results according to our ranking function. Instead of specifying only the keywords for q (e.g., $q = music$), we can place a boosting function for q such as $q = \{!boost\ b = product(page\_rank, arts\_prob)\ defType = dismax\}music$. By doing so, the query relevance score of each document to the query will be multiplied by the factor $product(page\_rank, arts\_prob)$, where $page\_rank$ is the indexed PageRank score, $arts\_prob$ is the indexed probability of each document belonging to "arts" and *product* is the function query provided by Apache Solr. Finally, the multiplication ($page\_rank$,$arts\_prob$ and query relevance to "music") will be used to rank the documents.

Besides, if the parameter fq is not empty in the request, Apache Solr will filter out the documents that do not satisfy the field query. If a field F specified by fq is numeric, we can use a numeric interval (e.g., [50, 100]) to filter out documents with $F > 100$ and $F < 50$.

After retrieving all the documents that satisfy the request, Apache Solr will rank the documents based on the ranking function and return them to the user.

4.3.2 User Interface and Web Server

To communicate with the search API of Apache Solr, we design an intuitive and effective user interface as well as a middle-level web server (developed in *Java*) to connect the user interface and Apache Solr. Among all the components of the user interface shown in Figure 1.5, the most important part is the hierarchical tree on the left side. We implement the tree by using *Javascript* and *AJAX*. When a user initially logs onto SEE, only the top-level categories of the hierarchy are displayed to the user in the tree. When the user clicks one category in the tree, an AJAX request will be sent to the web server to get its subcategories, and then the clicked category will be expanded on the client side by Javascript. It should be noted that each category on the tree is associated with a number in the brackets. The number indicates the number of results satisfying the current search condition, which includes the selected category, the truncation threshold and the search keywords. In the last subgraph of Figure 1.5 in Section 1.2.3, the number (27) beside "computer"→"security"→"hacking" means that there are 27 results having the probability greater than 0.5 for "hacking" and also matching the keyword "worms". The document number is also computed in the web server. Besides the subcategories, for each category, the web server will query the search server by using the search API, and extract the number of related documents in the meta information of the returned results (in JSON or XML format). The document numbers for all the subcategories will also be returned to the client side and displayed in the tree by Javascript.

Above the hierarchical tree, we introduce a topic (category) search box in the user interface. When users are not sure about the location of the category in the hierarchy, they can use the search box to find the category first. We use the partial keyword matching for the category search. After the user types in a few indicative characters of the category name, the top ten matched categories will be shown and the user can select the desired category from them. This functionality makes it more convenient for the new users of the search engine, since they may not know the structure of our category hierarchy. However, we believe that over time a well-thought-out category hierarchy can become familiar to users.

The primary responsibility of the web server is to construct the url, query the search API of the search server and return the results from the search server to the client side. There are several occasions (shown in the following) triggered by the user that the web server will reconstruct the query url and access the search server.

1. When the search button is clicked.
2. When the category is clicked.
3. When the button "Additional Results" is clicked.
4. When the page link is clicked.

Suppose a user searches "Michael Jackson" in "arts"→"music" without clicking the "Additional Results" button, the following is the url constructed by the web server to query the search API.

`http://localhost:88/see/result?hl=true&start=0&q={!boost~b=product(page_rank,arts_prob,music_prob)}Michael~Jackson&fq=arts_prob:[0.5~1]&fq=music_prob:[0.5~1]`

`http://localhost:88` is the host, `/see` is the web application of SEE on web server, `/result` is the servlet to construct the query. `hl=true` is to turn on the highlight of the keywords in the returned snippets. `start=0` is to get the first page of the returned results. In `product(page_rank,arts_prob,music_prob)`, we actually include the probabilities of all the ancestor categories of "music" into the calculation of the final ranking score. Here, since "music" only has one ancestor category, `arts_prob` is multiplied. `fq=arts_prob:[0.5~1]&fq=music_prob:[0.5~1]` is to apply the probability interval [0.5, 1], so that the documents with probability lower than 0.5 on "arts" and "music" will be truncated. If the user clicks the "Additional Results" button, this part will be replaced with `fq=arts_prob:[0.3~0.5)&fq=music_prob:[0.3~0.5)`, or even `fq=arts_prob:[0.1~0.3)&fq=music_prob:[0.1~0.3)`.

One important reason that we need another web server other than the search server is that we want to hide the implementation details from the users. Thus, on the user interface, we only show limited information to the user (e.g., category ID), and we complete the complex url construction on the server side.

4.4 Search Evaluation

In this section, we will conduct a comprehensive evaluation on the SEE compared with the keyword-based search engine. Since we are just building a prototype of the new search engine and do not have the resource to crawl a vast amount of webpages, we will not directly compare with the commercial search engines such as Google and Bing. We will utilize partial webpages from a well-known data collection for information retrieval called *ClubWeb09*[2], and categorize and index them into SEE. Then we compare two versions of SEE. They are SEE with the category hierarchy, and SEE without the category hierarchy, which is essentially the same as the traditional keyword-based search engine.

4.4.1 Data Collection

The data collection used for our evaluation is called *ClubWeb09*, which is from TREC (Text Retrieval Conference)[3]. The TREC consists of different workshops focusing on different information retrieval research areas (tracks). TREC provides the necessary infrastructure for large-scale evaluation of text retrieval methodologies to facilitate and encourage research in the information retrieval community. Each track holds a challenge (or contest) where a particular type of data sets and test problems are provided to the participates. Participants run their own retrieval systems on the data, and return with a list of the retrieved top-ranked documents. The organizer pools the individual results, judges the retrieved documents for correctness, and evaluates the results.

[2] `http://www.lemurproject.org/clueweb09.php`
[3] `http://trec.nist.gov/`

Judgement score	Description
4	This page represents a home page of an entity directly named by the query; the user may be searching for this specific page or site.
3	This page or site is dedicated to the topic; authoritative and comprehensive, it is worthy of being a top result in a web search engine.
2	The content of this page provides substantial information on the topic.
1	The content of this page provides some information on the topic, which may be minimal; the relevant information must be on that page, not just promising-looking anchor text pointing to a possibly useful page.
0	The content of this page does not provide useful information on the topic, but may provide useful information on other topics, including other interpretations of the same query.

Table 4.2: Five-point scale of relevance judgement in ClubWeb09.

ClubWeb09 is from the web track, which is to explore information seeking behaviors common in general web search. The data set was crawled from the Web during January and February 2009. The entire collection contains 1,040,809,705 web pages in 10 languages, and takes 25TB disk space.

Besides the text documents, ClubWeb09 also offers a list of 50 test tasks and the relevance judgement of the documents related to each task in the collection. Each task consists of a suggested query (keywords) and the description of the intended search topic. The relevance judgement of each document to each task is in a five-point scale as shown in Table 4.2. In a challenge of TREC, for a particular task, the participates need to issue the suggested query to their own retrieval system, and returned results are then evaluated based on the relevance judgement. However, in our evaluation, since we introduce a topic hierarchy into the search engine, the suggested queries, which are designed to test different keyword-based retrieval systems, are insufficient and sometimes inappropriate in our case. Therefore, for each task, we will design some new queries, but still based on the suggested query.

Since the total number of documents in ClubWeb09 is too large to be fully downloaded and indexed, we only extract a small portion from it to conduct the evaluation. The official website of ClubWeb09 actually provides a search engine called *Indri* for users to extract their desired documents from ClubWeb09. We take advantage of Indri to obtain the documents for our evaluation. In order to cover the 662 categories of our hierarchy as much as possible, for each category, we design some related keywords, query Indri and extract the top 1000 documents returned by Indri. Besides, in order to test how well SEE can retrieve the relevant documents (with judgement score greater than 0), we also extract the documents that are relevant to the 50 tasks from ClubWeb09. Finally, we extract 630,920 documents from ClubWeb09 in total. Then we categorize them by using the optimized classifiers obtained in Section 3.2.3.4 and index them into SEE. For each task, we issue queries to both SEE with hierarchy and without hierarchy, compare their returned documents with the actual relevant documents, and measure their performance.

i	rel_i	log_i	$\frac{rel_i}{log_2 i}$
1	4	0	N/A
2	2	1	2
3	0	1.59	0
4	3	2.0	1.5
5	1	2.32	0.431

Table 4.3: Element values to calculate DCG.

4.4.2 Evaluation Metric

To evaluate the performance of the search engines, we will use a typical measure for information retrieval named DCG (Discounted Cumulative Gain) [26]. DCG is particularly useful to evaluate the performance when the documents are judged by graded relevance (e.g., the five-point scale in ClubWeb09) rather than binary relevance. The intuition of DCG is that the usefulness (gain) of a document in a list is discounted by its position. In fact, DCG accumulates the discounted gain over all the documents up to a position. Mathematically, DCG at a position p can be defined as

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i},$$

where rel_i is the graded relevance of the result at position i. DCG is actually based on the following two assumptions.

1. Highly relevant documents are more useful when appearing earlier in a search engine result list (have higher ranks).

2. Highly relevant documents are more useful than marginally relevant documents, which are in turn more useful than irrelevant documents.

Here is an example on how to calculate the DCG measure given a list of documents with graded relevance. Suppose a search engine returns a list of documents for a query and we would like to calculate DCG for the first five documents. We also know the true relevance score for the first five documents are 4,2,0,3,1. Table 4.3 lists the values of different elements to calculate DCG. Then, DCG_5 can be computed as

$$DCG_5 = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i} = 4 + 2 + 0 + 1.5 + 0.431 = 7.931$$

4.4.3 Evaluation Results

We conduct the evaluation on SEE from two aspects. Firstly, to show the effectiveness of the hierarchical search engine over traditional keyword-based search engine, we will compare SEE

with hierarchy (called *SEE+H*) and SEE without hierarchy (called *SEE-H*, which is only based on keywords). Then, we will demonstrate the usefulness of the "Additional Results" button.

4.4.3.1 Comparison with keyword-based Search Engine

From all the 50 tasks in ClubWeb09, we remove 11 tasks that are not related to any topic category in our hierarchy. The purpose is to test the performance difference between SEE+H and SEE-H. For the removed tasks, SEE+H and SEE-H are actually of the same performance, since without specified category SEE+H has no difference with SEE-H. Then for each selected task, we find a corresponding category in our hierarchy that is likely to contain the information needed. Based on the suggested query, we design reasonable queries to search in SEE+H and SEE-H, respectively, and calculate their DCG at three different positions (top 5, top 10 and top 30). Table 4.4 lists the targets and suggested queries for all the 39 tasks.

Moreover, according to the property of each task, we further divide the 39 tasks into *category-intended tasks* (the first 29 tasks in Table 4.5) and *category-unintended tasks* (the last 10 tasks in Table 4.6). If a task has some indication to search information within a category (or topic), we call the task *category-intended task*. If the information searched in the task is specific and does not rely on any category, we call the task *category-unintended task*. More specifically, given a task, we attempt to design the most specific query to accurately express the information in the task. If one part of the keywords can be replaced with or included by one of the categories in our hierarchy, we call the task *category-intended task*; otherwise, we call it *category-unintended task*. For example, the first task is "Find information about horse hooves, their care, and diseases of hooves". The most specific query can be "horse hooves diseases", where the keyword "diseases" is in the domain of the category "Health". Therefore, it is a category-intended task. However, for the last task "Find information on plate tectonics and the major continental plates", the query "continental plates" is good enough to express the information in the task, and it cannot be further decomposed. Thus, it is a category-unintended task.

For each category-intended task, we design three queries to test SEE-H, since SEE-H is essentially the keyword-based search engine and people may try different queries when searching. The first query contains only the keywords for the needed information without any keywords for the category. In this case, since the returned documents usually contain a lot of irrelevant results from other categories, we call the query *ambiguous query*. In order to find more specific results, people often add some category-related keywords. Thus, following this intuition, we design another two queries by integrating two intuitive keywords respectively. We call those queries *topic-related queries*, since partial keywords are related to a topic. For example, one of the tasks is to find information about iron as an essential nutrient. The first query is simply "iron", and the other two queries are "iron nutrient" and "iron food", where "nutrient" and "food" are the possible keywords to constrain the search domain within a topic. To evaluate SEE+H, we simply search one query in the intended category. Table 4.5 presents the designed queries and the intended category for each category-intended task. For each task, query 1 is the ambiguous query, and query 2 and 3 are the topic-related queries.

For each category-unintended tasks, as the information is specific enough, we do not need to include additional keywords to refine the search. Hence, for both SEE+H and SEE-H, we only use one (the same) query. Besides, although the tasks are not category-intended, there

Task ID	target	suggested query
1	Find information about horse hooves, their care, and diseases of hooves.	horse hooves
2	Find events sponsored by the Association of Volleyball Professionals.	avp
3	Find locations and information of Discovery Channel stores and their products.	discovery channel store
4	Find information about iron as an essential nutrient.	iron
5	Find information about jobs in Connecticut.	ct jobs
6	Find information about penguins.	penguins
7	Find information about computer worms, viruses, and spyware.	worm
8	Find information about Flushing, a neighborhood in New York City.	flushing
9	Find information about PVC pipes and fittings.	pvc
10	Find beginners instructions to sewing, both by hand and by machine.	sewing instructions
11	Find information about the Sun, the star in our Solar System.	the sun
12	Find information on kiwi fruit.	kiwi
13	Find dealers that sell or rent Bobcat tractors and construction equipment.	bobcat
14	Find information about the NASA Voyager spacecraft and missions.	voyager
15	Find reviews of computer keyboards.	keyboard reviews
16	Find information about Afghanistan's history, government, religion, and culture.	afghanistan
17	Find information about joints in the human body.	joints
18	Find information about human memory.	memory
19	Where can I find information about forearm pain?	forearm pain
20	Find information on obsessive-compulsive disorder.	ocd
21	Find information on the MGB sports car.	mgb
22	Find information about the television show "ER".	er tv show
23	Find information about the Pink Floyd album, "The Wall"	the wall
24	Find the homepage of Raffles Hotel in Singapore.	raffles
25	Find information on the Nissan Titan truck.	titan
26	Find recipes for rice.	rice
27	Find information about the history, culture, and geography of South Africa.	south africa
28	Find information on taking the SAT college entrance exam.	sat
29	Find background information about man-made satellites.	satellite
30	Find information about the office of US President.	president of united states
31	Find information on the USS Yorktown, an aircraft carrier that is part of the museum exhibit at the Patriots Point museum in Charleston Harbor, SC.	uss yorktown charleston sc
32	How do I go about building a fence myself?	how to build a fence
33	How can I file my Federal income tax return online?	income tax return online
34	Find information about very-low-density lipoprotein, a type of cholesterol.	vldl levels
35	Find music, tour dates, and information about the musician Neil Young.	neil young
36	Find information about tornadoes, what causes them, and where they occur.	tornadoes
37	Find information on raised garden beds and boxes.	raised gardens
38	Find "reasonable" dieting advice, that is not fads or medications for weight loss.	dieting
39	Find information on plate tectonics and the major continental plates.	continental plates

Table 4.4: Targets and suggested queries for 39 tasks.

	SEE-H			SEE+H	
Task ID	query1	query2	query3	query	intended category
1	horse hooves	horse hooves care	horse hooves diseases	horse hooves	Health
2	avp	avp volleyball	avp sports	avp	Sports
3	discovery channel	discovery channel store	discovery channel products	discovery channel	Shopping
4	iron	iron nutrient	iron food	iron	Health
5	ct	ct jobs	Connecticut jobs	ct	Business→Employment
6	penguins	penguins animal	penguins south pole	penguins	Recreation→Pets
7	worm	worm computer	worm virus	virus	Computers→Security →Malicious Software
8	flushing	flushing new york	flushing review	flushing	Recreation→Travel
9	pvc channel	pvc pipe	pvc fitting	pvc	Business→Industrial Goods and Services
10	instructions	sewing instructions	needlework instructions	instructions	Arts→Crafts→ Needlework
11	sun	sun solar	sun astronomy	sun	Science→Astronomy→ Solar System
12	kiwi	kiwi fruit	kiwi food	kiwi	Home→Cooking→ Fruits and Vegetables
13	bobcat	bobcat dealer	bobcat tractor	bobcat	Business
14	voyager	voyager spacecraft	voyager NASA	voyager	Science→Astronomy
15	keyboard reviews	computer keyboard reviews	Peripherals keyboard reviews	keyboard reviews	Computers→Hardware →Peripherals
16	afghanistan	afghanistan history	afghanistan culture	afghanistan	Society→History
17	joints	joints human	joints health	joints	Health
18	memory	memory human	memory health	memory	Health
19	forearm	forearm pain	forearm ache	forearm	Health→Conditions and Diseases
20	ocd	ocd disorder	ocd health	ocd	Health→Conditions and Diseases
21	MGB	MGB car	MGB sports car	MGB	Recreation→Autos
22	ER	ER tv	ER tv show	ER	Arts→Television →Programs
23	wall	wall Pink Floyd	wall music	wall	Arts→Music
24	raffles	raffles hotel	raffles Singapore	raffles	Recreation→Travel
25	titan	titan truck	titan nissan	titan	Recreation→Autos
26	rice	rice recipe	rice cooking	rice	Home→Cooking
27	south africa	south africa history	south africa culture	south africa	Society→History
28	sat exam	sat exam college	sat entrance exam	sat exam	Reference→Education
29	satellite	man made satellite	satellite astronomy	satellite	Science→Astronomy

Table 4.5: Designed queries and the intended category for category-intended tasks.

	SEE-H	SEE+H	
Task ID	query	query	related category
30	president united states	president united states	Society→Politics
31	uss yorktown charleston sc	uss yorktown charleston sc	Reference→Museums
32	build fence	build fence	Home
33	income tax return online	income tax return online	Business→Accounting
34	vldl levels	vldl levels	Health
35	neil young	neil young	Arts→Music
36	tornadoes	tornadoes	Science→Earth Sciences→Atmospheric Sciences
37	raised garden	raised garden	Home→Gardening
38	dieting	dieting	Health
39	continental plates	continental plates	Science→Earth Sciences→Geology

Table 4.6: Designed queries and the related category for category-unintended tasks.

are still categories related to those tasks. To evaluate SEE+H, we just choose the most related category and search the query within that category. Table 4.6 tabulates the used queries and the related category for each category-unintended task.

Table 4.7 compares the performance of SEE-H and SEE+H, in terms of the DCG values at three positions for the 29 category-intended tasks. The Bold cell represents the winner in one comparison of DCG at the position of 5, 10 or 30. From the table, it is clear that SEE+H wins in most of the comparisons (19 wins for top 5 and top 10, and 18 wins for top 30). For those cases that SEE+H does not win, SEE+H can mostly keep itself in the second place in the comparisons (see the last row in Table 4.7). Furthermore, by looking to the DCGs obtained by SEE+H and SEE-H, we can observe that when SEE+H is the winner it can usually perform much better than any of the queries in SEE-H. For example, in the 5th task, in terms of DCG of top 30, SEE+H achieves 17.19, while SEE-H only obtains as high as 3.90. Therefore, when the tasks are category intended, SEE+H is more likely to achieve better results than SEE-H. In SEE+H, the user only needs to click into the intended category and search the keywords of the information needed. However, in SEE-H, even if the user changes the keywords three times, the results are not as good as in SEE+H.

For the 10 category-unintended tasks, the results of the comparison are presented in Table 4.8. It should be noted that the same query is issued for SEE-H and SEE+H for each task, but we do one more step in SEE+H (choose a related category). From the table, we can see, SEE+H performs even worse than SEE-H over the 10 category-unintended tasks. For the top 5 and top 10, it only wins in 3 out of 10 comparisons, and for the top 30, its performance improves a little (4 wins out of 10). Thus, SEE+H does not have advantage over SEE-H in category-unintended tasks.

Why SEE+H performs better for category-intended tasks than category-unintended tasks? In what situation does the category hierarchy help our search? In the following parts, we will discuss those issues.

For most of the web search tasks, they contain the primary target information such as the instruction in task 10 and the tornado in task 36. People attempt to use the most appropriate keywords to describe the target information (“instructions” in task 10 and “tornadoes” in

	Top 5				Top 10				Top 30			
	SEE-H			SEE+H	SEE-H			SEE+H	SEE-H			SEE+H
Task ID	query1	query2	query3	query	query1	query2	query3	query	query1	query2	query3	query
1	4.19	3.93	**4.99**	3.56	5.89	5.29	**8.36**	6.64	11.22	12.80	**17.29**	9.33
2	3.56	**14.25**	2.56	8.98	6.20	**19.63**	4.25	14.77	12.24	**30.08**	10.10	27.47
3	2.06	3.99	2.06	**4.50**	3.45	6.02	3.38	**6.53**	7.77	**12.44**	5.37	11.66
4	3.56	**6.86**	1.00	6.12	5.25	8.71	2.16	**9.81**	9.74	**15.71**	3.86	15.57
5	2.00	2.49	1.00	**4.75**	2.00	3.43	1.67	**8.05**	2.95	3.90	2.55	**17.19**
6	3.56	0.63	6.05	**6.76**	5.25	1.29	**8.10**	7.50	10.23	5.50	**17.25**	7.50
7	2.56	0.50	0.63	**6.12**	4.59	1.48	3.03	**9.15**	9.76	6.45	8.69	**16.42**
8	3.50	1.26	0.00	**4.50**	**5.09**	1.26	1.02	4.50	**11.50**	4.43	4.08	4.50
9	**3.56**	1.49	2.93	1.43	**5.25**	3.83	3.32	3.16	8.95	**9.18**	7.93	5.04
10	1.43	2.13	0.00	**7.69**	1.76	4.44	0.69	**9.39**	2.23	7.58	1.52	**13.52**
11	2.56	1.50	0.00	**7.68**	4.25	1.89	0.00	**12.02**	8.49	4.22	0.00	**13.48**
12	2.56	0.63	1.63	**8.12**	4.25	1.30	1.95	**10.24**	7.85	6.07	3.87	**10.24**
13	10.42	9.42	10.25	**11.18**	14.25	13.92	14.59	**15.46**	21.12	24.12	20.46	**26.39**
14	2.56	**10.05**	7.05	9.68	3.95	**13.90**	10.84	13.30	9.77	**21.46**	18.65	17.53
15	**1.00**	0.43	**1.00**	0.86	1.00	1.20	1.00	**3.01**	2.72	2.09	1.46	**3.01**
16	**4.56**	2.00	2.63	4.06	**5.90**	3.69	4.29	5.44	10.15	6.68	8.39	**10.37**
17	2.06	1.56	1.63	**3.56**	3.75	2.19	2.37	**6.57**	6.99	4.50	3.26	**13.90**
18	**2.93**	0.00	0.00	2.00	**3.93**	0.32	0.33	2.63	**7.19**	1.49	1.07	5.53
19	3.56	4.56	4.49	**5.19**	4.90	6.25	7.26	**8.23**	9.36	11.83	11.33	**14.81**
20	2.63	2.00	0.93	**5.69**	4.66	2.72	1.32	**8.07**	10.64	7.37	4.14	**14.67**
21	5.45	**7.06**	6.32	6.82	8.09	9.00	8.01	**9.66**	14.39	14.47	13.64	**15.31**
22	2.93	3.69	3.06	**5.00**	3.53	5.07	5.11	**6.57**	7.55	8.47	8.64	**17.31**
23	2.06	**6.12**	1.00	4.82	3.42	**8.66**	1.00	6.99	5.99	11.18	2.44	**11.54**
24	3.56	2.79	0.63	**4.26**	**5.89**	3.74	2.74	5.35	**10.17**	6.47	5.44	7.57
25	3.56	1.29	4.93	**9.75**	5.25	2.35	6.55	**11.49**	8.79	5.36	**12.53**	11.72
26	4.06	0.00	0.00	**5.63**	6.43	1.68	1.74	**9.02**	10.81	2.64	1.96	**16.10**
27	1.86	2.36	2.63	**4.79**	2.81	2.66	2.63	**7.97**	6.23	5.74	4.91	**8.70**
28	2.00	2.00	0.00	**5.13**	2.93	2.00	1.49	**8.79**	5.91	4.91	7.06	**15.16**
29	1.56	1.63	0.00	**3.56**	2.94	5.12	0.00	**6.65**	6.12	7.92	0.24	**11.20**
first place	4	5	2	19	5	3	2	19	4	4	3	18
second place	11	7	4	8	8	8	4	9	10	9	3	7

Table 4.7: The comparison of SEE with hierarchy and without hierarchy for category-intended tasks

	Top 5		Top 10		Top 30	
Task ID	SEE-H	SEE+H	SEE-H	SEE+H	SEE-H	SEE+H
30	**5.01**	2.90	**5.01**	2.90	8.03	**9.21**
31	**6.11**	4.43	**6.11**	4.43	**12.20**	4.43
32	**3.73**	0	**3.73**	0	**4.54**	2.06
33	**10.25**	7.25	**10.25**	7.25	**14.81**	10.14
34	**9.45**	8.91	**9.45**	8.91	**17.49**	17.77
35	**8.89**	8.69	**8.89**	8.69	**16.92**	15.23
36	4.77	**5.34**	4.77	**5.34**	11.30	**12.71**
37	2.13	**2.14**	2.13	**2.14**	2.90	**2.14**
38	**3.50**	1.96	**3.5**	1.96	**4.71**	3.83
39	9.19	**13.22**	9.19	**13.22**	17.07	**23.09**
first place	7	3	7	3	6	4

Table 4.8: The comparison of SEE with hierarchy and without hierarchy for category-unintended tasks

task 36). Those keywords can usually match most of the indexed documents in the search engine. In the traditional keyword-based search engine like SEE-H, for task 10 to find sewing instructions, only the keyword "instructions" is insufficient and ambiguous, since other irrelevant documents (e.g., computer instructions) may also be matched and those noisy results may confuse the user. This task is actually intended to search "instructions" within the category "sewing". To reduce the noisy results, intuitively, normal users would add an extra keyword to restrict the search domain in "sewing". However, choosing a proper keyword is actually tricky for some users. Since the documents belonging to the category "sewing" can contain various relevant keywords such as "sewing", "needlework", "cross stitch" and "embroidery", etc, the users, particularly those who are not familiar with the domain "sewing", will have difficulties in finding good keywords. If keywords are not chosen well, the results will be poor with very few relevant documents. Even if the user knows all the relevant keywords in "sewing", he/she may need to issue different keywords combinations multiple times (e.g., "sewing instructions", "needlework instructions" and "embroidery instructions", etc.), in order to retrieve all the relevant documents. In contrast, in SEE+H, we only need to search the keyword "instructions" within the category "needlework" and it will match all the sewing-related documents pre-categorized into the category. Therefore, the results of SEE+H are expected to be much better in category-intended tasks.

Figure 4.3 illustrates the general idea on why SEE+H can work better than SEE-H in the category-intended task (task 10). The solid frame represents all the documents containing the keyword "instructions", and the dashed frame represents the documents containing "instructions" and related to the category "sewing". In SEE-H, if the keyword is too general such as "instructions", the returned results are too noisy, including all the irrelevant documents in the solid frame. On the other hand, if the keywords are too specific such as "sewing instructions", the returned results would have too few relevant documents like the documents in the shaded rectangle. In SEE+H, we have a way to restrict the search domain in the category "sewing" (the dashed frame), so that the user only needs to search "instructions" and the returned doc-

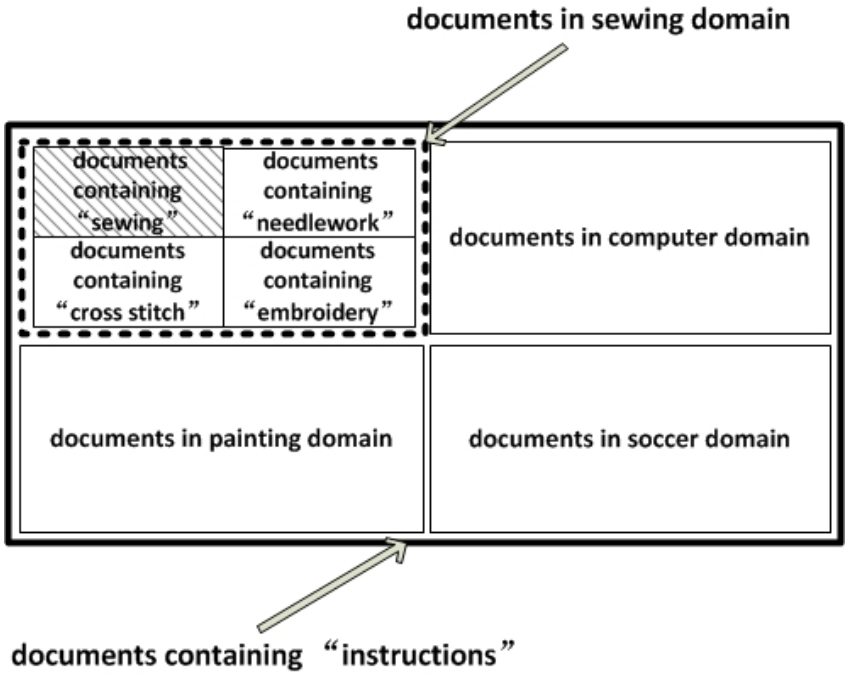

Figure 4.3: Illustration why SEE works better than category-intended tasks.

uments can include all the relevant documents about sewing instructions (even without the keyword "sewing"). We have known why SEE+H works well in category-intended tasks. But why SEE+H has a poor performance in category-unintended tasks such as task 36? In task 36, the target information is tornado, and the intention of the user is all information about tornadoes without any further domain restriction. Thus, to search in SEE-H, only the keyword "tornadoes" is enough to locate most of the relevant documents. On the other hand, in SEE+H, as we know tornado is related to the atmospheric sciences, we can search "tornadoes" within that category. However, it does not help filtering out the noisy documents, since tornadoes are not likely to be related to other domains. Conversely, it actually filters out some useful results (false negative documents), as the classifiers are not 100% accurate. Therefore, it is reasonable that SEE+H works poorly in category-unintended tasks.

From the discussion above, when the user is intended to search the target information within a category, the search engine with hierarchy would likely be helpful.

4.4.3.2 Search for Additional Results

In section 4.2, we introduced a button named "Additional Results" into the user interface. We will evaluate the usefulness of the button in search. The purpose of the button is to provide an opportunity for the user to find more desired results when few documents are returned. Thus, we only test the cases for SEE+H in the 29 category-intended tasks where the number of returned results are no more than 100. Table 4.9 shows the document ranking of top 10 of the additional results when the button is clicked. Take the first row as an example. When the user searches "horse hooves" in the category "Health", SEE+H only returns 44 results. When the user reads over the 44 results, comes to the last page and clicks the button to explore additional

Task ID	query	intended category	number of results before button clicked	ranking of additional results
51	horse hooves	Health	44	2,2,1,1,2,0,0,0
52	avp	Sports	52	2,1,2,3,4,2,2,1,4,1
63	flushing	Recreation→Travel	70	1,0,0,0,0
93	raffles	Recreation→Travel	63	1,2,0,0,0,0,0
94	titan	Recreation→Autos	76	0,0,0,0,0,0,0

Table 4.9: The document ranking of top 10 of the additional results when the button is clicked.

results, the top 10 of the additional results are ranked as $2, 2, 1, 1, 2, 0, 0, 0$[4]. From the table, we can observe that the additional results (at least top results) are pretty useful in the most of the five tasks (the first four tasks).

4.5 Summary

In this chapter, we discussed the core techniques of our search engine SEE. We propose a novel ranking score function and a new interface to explore additional results. We also present the detailed implementation of SEE. Then we leverage a well-known data collection named ClueWeb09 to conduct a comprehensive evaluation on SEE. The results demonstrate that when the search is intended to search with a category, SEE is likely to perform better than the traditional keyword-based search engine, and thus can improve the search experience of the users.

[4]Total number of additional results is 8 here. Rank 2 means the document is relevant to the query, rank 1 means slightly relevant and rank 0 means irrelevant.

Chapter 5

Improving SEE with Minimal Human Supervision

The last two chapters have shown the effectiveness and usefulness of SEE. Suppose SEE has been deployed and used by a number of users, how to continuously improve its performance? In this chapter, we will address this issue.

Since the performance of SEE largely relies on the performance of the classifiers, improving the classifiers is an intuitive and direct approach. However, in order to boost a classifier a bit, usually a large number of training documents are required. In reality, the *labeled* web documents are very limited compared to the total number of *unlabeled* webpages on the Internet. Thus, to obtain new training documents, human experts need to be employed to label the unlabeled webpages, which is very costly. Moreover, in our case, the experts have to provide 662 labels to each document, since each training document is associated with 662 categories in the hierarchy. How can we build reliable hierarchical classifiers from a relatively small number of labeled documents? Can we reduce the human labeling effort significantly? Those are the questions we are going to explore in this chapter.

To tackle the lack of the labeled examples, active learning can be a good choice [63, 52, 70]. The idea of active learning is that, instead of passively receiving the training examples, the learner actively selects the most "informative" examples for the current classifier and gets their labels from the oracle (i.e., human expert). Usually, those most informative examples can benefit the classification performance most, so the classifier can be boosted quickly with a few examples labeled. Several works have successfully applied active learning in text classification [63, 18, 72]. However, to our best knowledge, no previous works have been done in hierarchical text classification with active learning due to several technical challenges. For example, as a large taxonomy can contain thousands of categories, it is impossible to have one oracle to provide all labels. Thus, similar to DMOZ (mentioned in Section 2.1.2), multiple oracles are needed. What would be a realistic setting for multiple oracles for active learning in hierarchical text classification? How can we leverage the hierarchical relation to further improve active learning?

In this chapter, we study how active learning can be effectively applied to hierarchical text classification so that the number of labeled examples (or oracle queries) needed can be reduced significantly. We propose a novel setting of multiple oracles, which is currently in use in many real-world applications (e.g., ODP). Based on this setting, we propose an effective framework

for active learning in hierarchical text classification. Moreover, we explore how to utilize the hierarchical relation to further improve active learning. Accordingly, several novel strategies and heuristics are devised. According to our experiments, active learning under our framework significantly outperforms the baseline learner, and the additional strategies further enhance the performance of active learning for hierarchical text classification. Compared to the best performance of the baseline hierarchical learner, our best strategy can reduce the number of required labeled examples by 74% to 90%. Those novel active learning strategies can be employed to continuously improve the performance of SEE, with less human effort. This chapter is also joint work with Xiao Li. We both contributed in generating the idea and designing the active learning framework and strategies, while Xiao contributed more in coding the algorithms.

5.1 Multi-Oracle Setting

When active learning is applied to text classification, as far as we know, all previous works (e.g., [18, 72]) explicitly or implicitly assume that given a document that might be associated with multiple labels, there always exist oracles who can perfectly answer all labels. In hierarchical text classification, it is very common that the target hierarchy has a large number of categories (e.g., DMOZ has over one million categories) across various domains, and thus it is unrealistic for one oracle (expert) to be "omniscient" in everything. For example, an expert in "business" may have less confidence about "computer", and even less about "programming". If the expert in "business" has to label "programming", errors can occur. Such error introduces noise to the learner.

Therefore, it is more reasonable to assume that there are multiple oracles who are experts in different domains. Each oracle only gives the label(s) related to his or her own domains. Thus, the labels provided by multiple oracles will be more accurate and reliable than the labels given by only one oracle. Although previous works have studied active learning with multiple oracles [14, 45], as far as we know, their settings are quite different from ours as their oracles provide labels for all examples for only one category, while in our case, different oracles provide labels for examples in different categories in the hierarchy.

Our setting of multiple oracles is actually implemented in DMOZ (mentioned in Section 2.1.2). DMOZ holds a large number of categories, and each category is generally maintained by at least one human editor whose responsibility is to decide whether or not a submitted website belongs to that category.[1] We adopt the similar setting of DMOZ. In our setting, each category in the hierarchy has one oracle, who decides solely if the selected document belongs to the current category or not (by answering "Yes" or "No").

5.2 A Novel Active Learning Framework for Hierarchical Classification

Here, we mainly discuss pool-based active learning where a large pool of unlabeled examples is available for querying oracles. Figure 5.1 shows the basic idea of our hierarchical active

[1] See `http://www.dmoz.org/erz/` for DMOZ editing guidelines.

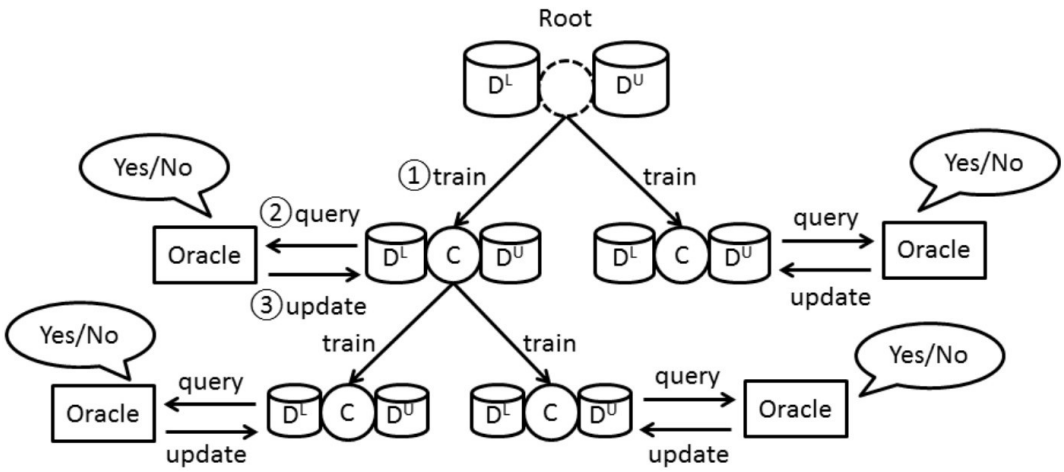

Figure 5.1: The hierarchical active learning framework. The typical active learning steps are numbered 1, 2, 3 in the figure.

learning framework. Simply speaking, at each iteration of active learning, classifiers on different categories *independently* and *simultaneously* select the most informative examples from the unlabeled pool for themselves, and ask the oracles on the corresponding categories for the labels. The major steps of our hierarchical active learning algorithm are as follows:

1. We first train a binary classifier (C) on each category to distinguish it from its sibling categories. The training set (D^L) is constructed by using the positive examples from the training set of the parent category [57].[2]

2. Then, we construct the local unlabeled pool (D^U) for each classifier (see Section 5.2.1), select the most informative examples from the local unlabeled pool for that classifier, and query the corresponding oracle for the labels.

3. For each query, the oracle returns "Yes" or "No" to indicate whether the queried example belongs to that category or not. Based on the answers, the classifier updates its classification model (see Section 5.2.2).

4. This process is executed simultaneously on all categories at each iteration and repeats until the terminal condition is satisfied.

There are two key steps (step two and three) in the algorithm. In step two, we introduce the local unlabeled pool to avoid selecting *out-of-scope* (we will define it later) examples. In step three, we tackle how to leverage the oracle answers in the hierarchy. We will discuss them in the following subsections.

[2]On the root of hierarchy tree, every example is positive.

5.2.1 Unlabeled Pool Building Policy

From step one of our algorithm, we know that the training examples for a deep category (say c) must belong to its ancestor categories. However, it is likely that many unlabeled examples do not belong to the ancestor categories of c. We define those examples as *out-of-scope* examples. If those out-of-scope examples are selected by c, we may waste a lot of queries. Thus, instead of using one shared unlabeled pool [18] for all categories, we construct a local unlabeled pool on each of the categories. To filter out these out-of-scope examples, we use the predictions of the ancestor classifiers to build the local unlabeled pool. Specifically, given an unlabeled example x and a category c, only if all the ancestor classifiers of c predict x as positive, then we will place x into the local unlabeled pool of c.

5.2.2 Leveraging Oracle Answers

For the two answers ("Yes" or "No") from oracles, there are several possible ways to handle them. We give a brief overview here and discuss the detailed strategies in Section 5.3.

If the answer is "Yes", we can simply update the training set by directly including the queried example as a positive example. To better leverage the hierarchical relation, we can even add the positive example to all the ancestor categories. Furthermore, since the positive example is possibly a negative example on some of the sibling categories, we may consider including it as a negative example to the sibling categories.

If the answer is "No", we can not simply add the example as a negative example, since we do not know whether the queried example actually belongs to the ancestor categories. Thus, we could simply discard the example. Alternatively, we can also query the oracle on the parent category to see if the example belongs to the parent category, but the extra query may be wasted if the answer is "No".

In the following parts, we will first present our experimental configuration, and then empirically explore whether our framework can be effectively applied to hierarchical classification and whether different strategies described above can indeed improve active learning.

5.3 Empirical Study

5.3.1 Datasets

We utilize four real-world hierarchical text datasets (*20 Newsgroups*, *OHSUMED*, *RCV1* and *DMOZ*) in our experiments. They are common benchmark datasets for evaluation of text classification methods. We give a brief introduction of the datasets. The statistic information of the four datasets is shown in Table 5.1.

The first dataset is *20 Newsgroups*[3]. It is a collection of newsgroup documents partitioned evenly across 20 different newsgroups. We group these categories based on subject matter into a three-level topic hierarchy which has 27 categories. The second dataset is *OHSUMED*[4]. It is a clinically-oriented MEDLINE dataset with a hierarchy of twelve levels. In our experiments,

[3] http://people.csail.mit.edu/jrennie/20Newsgroups/

[4] http://ir.ohsu.edu/ohsumed/

Table 5.1: The statistic information of the four datasets. Cardinality is the average number of categories per example (i.e., multi-label datasets).

Dataset	Features	Examples	Categories	Levels	Cardinality
20 Newsgroups	61,188	18,774	27	3	2.202
OHSUMED	12,427	16,074	86	4	1.916
RCV1	47,236	23,049	96	4	3.182
DMOZ	92,262	12,735	91	3	2.464

we only use the sub-hierarchy under subcategory "heart diseases" which is well-studied and usually taken as a benchmark dataset for text classification [33, 53]. The third dataset is *RCV1* [38]. It includes three classification tasks: topic, industrial and regional classification. In our experiments, we focus on the topic classification task.[5] The last dataset is *DMOZ*. We extract a partial hierarchy from ODP rooted at "Science" and it has three-level category hierarchy. The purpose is to test the effectiveness of the active learning method on SEE, since SEE uses partial hierarchy of ODP as the training data.

5.3.2 Performance Measure

To evaluate the performance in hierarchical classification, we adopt the *hierarchical F-measure*, which has been widely used in hierarchical classification for evaluation [67, 12, 57]. We first define two related measures *hierarchical precision* and *hierarchical recall*:

$$hP = \frac{\sum_i |\hat{P}_i \bigcap \hat{T}_i|}{\sum_i |\hat{P}_i|} \quad hR = \frac{\sum_i |\hat{P}_i \bigcap \hat{T}_i|}{\sum_i |\hat{T}_i|}, \tag{5.1}$$

where $\hat{P}_i$ is the set consisting of the most specific categories predicted for test example i and all its (their) ancestor categories and $\hat{T}_i$ is the set consisting of the true most specific categories of test example i and all its (their) ancestor categories. And the definition of the hierarchical F-measure is as follows,

$$hF = \frac{2 \times hP \times hR}{hP + hR}. \tag{5.2}$$

5.3.3 Active Learning Setup

In our experiment, linear SVM (LibLinear) is still used as the base classifier on each category in the hierarchy. We set $C = 1000$ and w_+ as the negative class proportion in the training set. For example, if the class ratio of positive and negative class in the training set is 1:9, then $w_+ = 0.9$. The purpose is to give more penalty to the error on the minority class.

[5] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

For active learning, due to the simplicity and effectiveness of *Uncertainty Sampling*[6] [37], we adopt uncertainty sampling as the strategy to select the informative examples from the unlabeled pool. It should be noted that our hierarchical active learning framework is independent of the specific active learning strategy. Other strategies, such as expected error reduction [52] and representative sampling [70] can also be used. We will study them in the future.

We split all the four datasets into labeled (1%), unlabeled (89%) and testing (10%) parts. As we already know the labels of unlabeled examples, we will use the simulating oracles instead of the real human oracles (experts). The training process is decomposed into a sequence of iterations. In each iteration, each category simultaneously selects a fixed number of examples[7] from its local unlabeled pool and queries the oracles (one query will be consumed when we ask one oracle for one label). After each category updates its training set, we recompute the parameter w_+ and update the classification model. The entire training process terminates when the number of queries consumed exceeds the predefined query limit. To reduce the randomness impact of the dataset split, we repeat this active learning process for 10 times. All the results (curves) in the following experiments are averaged over the 10 independent runs and accompanied by error bars indicating the 95% confidence interval. In this section, we will first experimentally study the standard version of our active learning framework for hierarchical text classification, then propose several improved versions and compare them with the previous version.

5.3.4 Standard Hierarchical Active Learner

In order to validate our active learning framework, we will first compare its standard version (we call it standard hierarchical active learner) with the baseline learner. The standard hierarchical active learner uses intuitive strategies to handle oracle answers (see Section 5.2.2) in deep categories. If the oracle answer is "Yes", the standard hierarchical active learner directly includes the example as a positive example; if "No", it simply discards the example. On the other hand, the baseline learner is actually the non-active version of the standard hierarchical active learner. Instead of selecting the most informative examples, it selects unlabeled examples randomly on each category.

We set the query limit as $50\times|C|$ where $|C|$ is the total number of categories in the hierarchy. Thus, in our experiments the query limits for the four datasets are 1,350, 4,300, 4,800 and 4,850 respectively. We denote the standard hierarchical active learner as *AC* and the baseline learner as *RD*. Figure 5.2 plots the average learning curves for *AC* and *RD* on the four datasets. As we can see, on all the datasets *AC* performs much better than *RD*. This result is reasonable since the unlabeled examples selected by *AC* are more informative than *RD* on all the categories in the hierarchy. From the curves, it is apparent that to achieve the best performance of *RD*, *AC* needs significantly fewer queries (approximately 43% to 82% queries can be saved)[8]. However, there is an exception. That is, at the later learning stage of DMOZ, the advantage of *AC* over *RD*

[6]Uncertain sampling in active learning selects the unlabeled example that is closest to the decision boundary of the classifier.

[7]We heuristically use logarithm of the unlabeled pool size to calculate the number of selected examples for each category.

[8]In *20 Newsgroups*, *RD* uses 1,350 queries to achieve 0.46 in terms of the hierarchical F-measure, while *AC* only uses 750 queries. Thus, $(1350 - 750)/1350 = 44.4\%$ of the total queries are saved. The savings for other datasets are 82.5%, 72.9% and 43.3%.

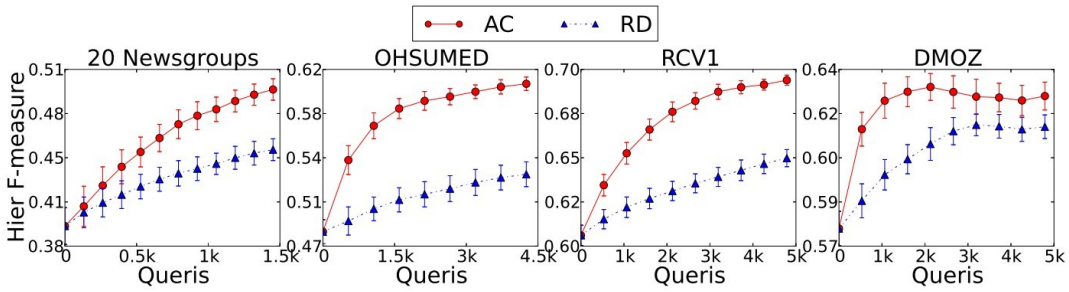

Figure 5.2: Comparison between *AC* and *RD* in terms of the hierarchical F-measure. X axis is the number of queries consumed and Y axis is the hierarchical F-measure.

becomes smaller. It is probably due to the imbalance of datasets in the hierarchy of DMOZ. With severe imbalance, it is more likely the oracle will say "No", since we have more negative examples. In this case, the selected example will be discarded and never learned by both *AC* and *RD*. This situation becomes more serious after most of the positive examples have already been learned. That is why the performance of *AC* and *RD* becomes similar in the later learning stage.

Although the standard hierarchical active learner (*AC*) significantly reduces the number of oracle queries compared to the baseline learner (*RD*), we should note that there is no interaction between categories in the hierarchy (e.g., each category independently selects examples and queries oracle). Our question is: can we further improve the performance of the standard active learner by taking into account the hierarchical relation of different categories? We will explore several leveraging strategies in the following subsections.

5.3.5 Leveraging Positive Examples in Hierarchy

As mentioned in Section 5.2.2, when the oracle on a category answers "Yes" for an example, we can directly include the example into the training set on that category as a positive example. Furthermore, according to the category relation in a hierarchy, if an example belongs to a category, it will definitely belong to all the ancestor categories. Thus, we can propagate the example (as a positive example) to all its ancestor categories. In such cases, the ancestor classifiers can obtain *free* positive examples for training without any query. It coincides with the goal of active learning: reducing the human labeling cost!

Based on the intuition, we propose a new strategy *Propagate* to propagate the examples to the ancestor classifiers when the answer from oracle is "Yes". The basic idea is as follows. In each iteration of the active learning process, after we query an oracle for each selected example, if the answer from the oracle is "Yes", we propagate this example to the training sets of all the ancestor categories as positive. At the end of the iteration, each category combines all the propagated positive examples and the examples selected by itself to update its classifier.

We integrate *Propagate* to the standard hierarchical active learner (we name the integrated version as *AC*+) and then compare it with the original *AC*. The first row of Figure 5.3 shows the learning curves of *AC*+ and *AC* on the four datasets in terms of the hierarchical F-measure. Overall, the performance of *AC*+ is slightly better than that of *AC*. By propagating positive examples, the top-level classifiers of *AC*+ can receive a large number of positive examples and

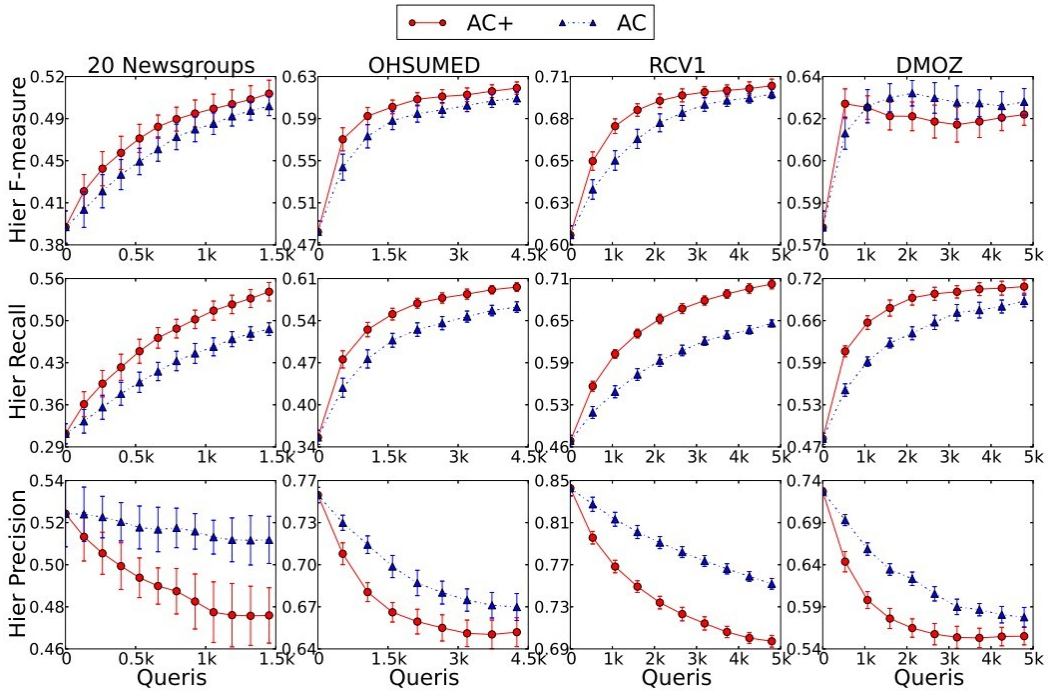

Figure 5.3: Comparison between *AC+* and *AC* in terms of the hierarchical F-measure (first row), recall (second row) and precision (third row).

thus the (hierarchical) recall of *AC+* increases faster than *AC* as shown in the second row. This is the reason why *AC+* can defeat *AC* on the first three datasets. However, from the third row, we can see the hierarchical precision of *AC+* actually degrades very sharply since the class distribution of the training set has been altered by the propagated positive examples. It thus weakens the boosting effect in the hierarchical recall and hinders the improvement of overall performance in the hierarchical F-measure.

Since positive examples can benefit the hierarchical recall, can we leverage negative examples to help maintain the hierarchical precision so as to further improve *AC+*? We will propose two possible solutions in the following.

5.3.6 Leveraging Negative Examples in Hierarchy

We introduce two strategies to leverage negative examples. One is to query parent oracles when the oracle answers "No"; the other is to predict the negative labels for sibling categories when the oracle answers "Yes".

For deep categories, when the oracle answers "No", we actually discard the selected example in *AC+* (as well as in *AC*, see Section 5.3.4). However, in this case, the training set may miss a negative example and also possibly an informative example. Furthermore, if we keep throwing away those examples whenever oracle says "No", the classifiers may not have chance to learn negative examples. On the other hand, if we include this example, we may introduce noise to the training set, since the example may not belong to the parent category, thus an out-of-scope example (see Section 5.2.1).

How can we deal with the two cases? We introduce a complementary strategy called *Query*. In fact, the parent oracle can help us decide between the two cases. We only need to issue another query to the parent oracle on whether this example belongs to it. If the answer from the parent oracle is "Yes", we can safely include this example as a negative example to the current category. If the answer is "No", we can directly discard it. Here, we do not need to further query all the ancestor oracles, since the example is already out of scope of the current category and thus cannot be included into its training set. There is a trade-off. As one more query is asked, we may obtain an informative negative example, but we may also waste a query. Therefore, it is non-trivial if this strategy works or not.

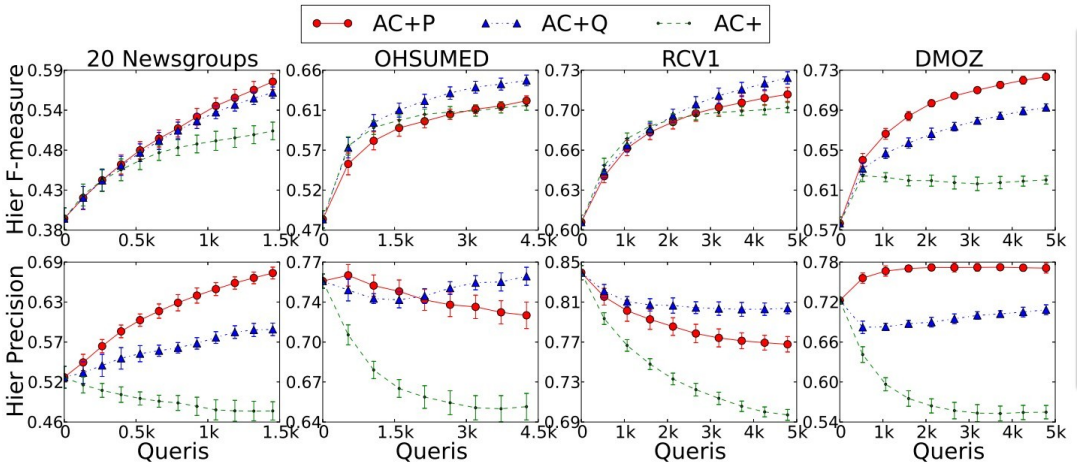

Figure 5.4: Comparison between *AC+P*, *AC+Q* and *AC+* in terms of the hierarchical F-measure (upper row) and precision (bottom row).

When the oracle on a category (say "Astronomy") answers "Yes" for an example, it is very likely that this example may not belong to its sibling categories such as "Chemistry" and "Social Science". In this case, can we add this example as a negative example to its sibling categories? In those datasets where each example only belongs to one single category path, we can safely do so. It is because for the categories under the same parent, the example can only belong to at most one category. However, in most of the hierarchical datasets, the example belongs to multiple paths. In this case, it may be positive on some sibling categories. If we include this example as negative to the sibling categories, we may introduce noise.

To decide which sibling categories an example can be included as negative, we adopt a conservative heuristic strategy called *Predict*. Basically, when a positive example is included into a category, we add this example as negative to those sibling categories that the example is least likely to belong to. Specifically, if we know a queried example x is positive on a category c, we choose m sibling categories with the minimum probabilities (estimated by Platts Calibration [48]). We set

$$m = n - \max_{x \in D_L} \Psi_{\uparrow c}(x), \tag{5.3}$$

where D_L is the labeled set, $\uparrow c$ is the parent category of c, n is the number of children categories of $\uparrow c$, and $\Psi_{\uparrow c}(x)$ is the number of categories under $\uparrow c$ that the example x belongs to.

We integrate the two strategies *Query* and *Predict* discussed above into *AC+* and then compare the two integrated versions (*AC+Q* and *AC+P*) with the original *AC+*. Since in *AC+* positive examples are propagated, we can use this feature to further boost *AC+Q* and *AC+P*. For *AC+Q*, when the parent oracle answers "Yes", besides obtaining a negative example, we can also propagate this example as a positive example to all the ancestor categories. For *AC+P*, as a positive example is propagated, we can actually apply *Predict* to all the ancestor categories.

We plot their learning curves for the hierarchical F-measure and the hierarchical precision on the four datasets in Figure 5.4. As we can see in the figure, both *AC+Q* and *AC+P* achieve better performance of the hierarchical F-measure than *AC+*. By introducing more negative examples, both methods maintain or even increase the hierarchical precision (see the bottom row of Figure 5.4). As we mentioned before, *AC+Q* may waste queries when the parent oracle answers "No". However, we discover that the average number of informative examples obtained per query for *AC+Q* is much larger than *AC+* (at least 0.2 higher per query). It means that it is actually worthwhile to issue another query in *AC+Q*. Another question is whether *AC+P* introduces noise to the training sets. According to our calculation, the noise rate is at most 5% on all the four datasets. Hence, it is reasonable that *AC+Q* and *AC+P* can further improve *AC+*.

However, between *AC+Q* and *AC+P*, there is no consistent winner on all the four datasets. On OHSUMED and RCV1, *AC+Q* achieves higher performance, while On 20 Newsgroup and DMOZ, *AC+P* is more promising. Thus, we can see that *AC+P* is a good choice to improve the performance of SEE. We also try to make a simple combination of *Query* and *Predict* with *AC+* (we call it *AC+QP*), but the performance is not significantly better than *AC+Q* and *AC+P*. We will explore a smarter way to combine them in our future work.

Finally, we compare the improved versions *AC+Q* and *AC+P* with the non-active version *RD*. We find that *AC+Q* and *AC+P* can save approximately 74% to 90% of the total queries. The savings for the four datasets are 74.1%, 88.4%, 83.3% and 90% respectively (these numbers are derived from Figures 5.2 and 5.4).

To summarize, we propose several improved versions (*AC+*, *AC+Q* and *AC+P*) in addition to the standard version (*AC*) of our hierarchical active learning framework. According to our empirical studies, we discover that in terms of the hierarchical F-measure, *AC+Q* and *AC+P* are significantly better than *AC+*, which in turn is slightly better than *AC*, which in turn outperforms *RD* significantly. In terms of query savings, our best versions *AC+Q* and *AC+P* need significantly fewer queries than the baseline learner *RD*.

5.4 Summary

In this chapter, we propose a novel multi-oracle setting for active learning in hierarchical classification as well as an effective active learning framework for this setting. We explore different solutions which attempt to utilize the hierarchical relation between categories to improve active learning. We also discover that propagating positive examples to the ancestor categories can improve the overall performance of hierarchical active learning. However, it also decreases the precision. To handle this problem, we propose two novel strategies to leverage negative examples in the hierarchy. Our empirical study shows both of them can further boost the performance. Our best strategy proposed can save a considerable number of oracle queries (74%

to 90%) compared to the baseline learner. It is clear that those novel active learning strategies, particularly the strategy named *AC+P*, can be employed to continuously improve the performance of SEE, with less human effort.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we reported our first attempt to build a new type of search engine called SEE. In addition to the keywords search, we introduced a topic hierarchy into the search engine, so that the users can browse the most popular webpages within each topic category and search target information in a desired category to avoid ambiguity. To implement a prototype of SEE, we firstly build a back-end classification system by leveraging an existing labeled data source called ODP. When new webpages are crawled, the classification system automatically classifies them and then indexes them into the search engine. As all the classification information are indexed, the response for the user queries is very fast. To implement SEE, we design a novel ranking algorithm and a novel user interface to let users explore additional results. According to the evaluation results, SEE has a good performance in terms of search quality, particularly when the user is intended to search in a category. One important and novel feature of SEE is that once the classification system is constructed, it can index as many documents as the server is capable of. Thus, SEE has a promising scalability. Besides, we also discussed how to improve SEE by proposing a new active learning framework for hierarchical text classification.

The thesis primarily addressed two issues: implementing SEE and improving SEE (Chapter 5). For the implementation, we discussed the classification system (Chapter 3) and the detailed design of the search engine (Chapter 4), separately.

In Chapter 3, we talked about the classification system for SEE in the following steps.

- We gave an introduction on how we extracted the topic hierarchy for SEE as well as how we generated the training data for the construction of the classifiers. We design an intuitive and reasonable criteria to select a topic hierarchy from Open Directory Project, and choose the webpages under the categories of this topic hierarchy as the raw training data.

- We discussed the detailed classification methodology, which includes how we preprocessed the raw webpages, how we dealt with the computational issue, how we optimized the classifiers and how we categorize new documents. We took sophisticated methods to preprocess the training data, and designed a parallel framework to improve the computational efficiency. More importantly, we designed a novel algorithm to find the parameters

that maximize the classification performance, and leveraged hierarchical classification to categorize new documents.

- We conducted an evaluation on the constructed classifiers and the results demonstrated that the classification performance is satisfactory.

In Chapter 4, we discussed the detailed design and implementation of the search engine SEE following the steps below.

- We proposed a novel ranking function for SEE, integrating the likelihood of each document belonging to a category. It can deal with the false positive errors introduced in the classification phase. With the new ranking function, users can browse the most popular webpages within each topic category.

- We introduced a button into the user interface to let users explore additional results when few results are returned. It can deal with the false negative errors introduced in the classification phase. When results are limited, this feature can help users find more results.

- We discussed the detailed implementation of the system from a development perspective. It includes the indexing system, user interface and the web server, as well as the data flow between them.

- We conducted a comprehensive evaluation on SEE to test its effectiveness. We used a well-know data collection ClueWeb09, and DCG as the metric. We compared SEE with the traditional keyword-based search engine and also test the usefulness of the button to explore additional results. The results demonstrated that SEE is particularly useful when the user is intended to search within a topic category.

In Chapter 5, we studied how to further improve the performance of SEE with minimal human effort, by proposing a novel active learning framework for hierarchical classification.

- We pointed out the weakness of the traditional single-oracle mode in hierarchical setting, and introduced a new multi-oracle setting where each expert is responsible to answer the query on a particular category. The labels provided in our setting are more accurate and reliable than the labels given by only one oracle.

- We proposed a novel active learning framework for hierarchical classification. Based on that, we further designed several novel strategies to reduce the human labeling cost. The results demonstrated that our active learning strategies can save 74%-90% oracle queries compared to the random strategy.

We list our major contributions over the entire thesis as follows.

1. We propose the new idea of integrating hierarchy into general web search engine and implement a prototype of this new search engine SEE.

2. We take advantage of machine learning and data mining techniques (e.g., SVM, hierarchical classification and data preprocessing) to categorize Internet webpages into our hierarchy. We design a parallel framework to improve the efficiency for the training and classification processes. Besides, we propose an algorithm to find the best parameter combination that maximizes the classification performance. The evaluation results demonstrate that our classification performance is good.

3. To build the new search engine SEE, we propose a novel ranking function and design a new interface to explore additional results when returned results are limited. We also conduct a comprehensive evaluation on SEE, and the results show that the hierarchical version of SEE can achieve better search results than the flat version without hierarchy in most of the queries, particularly when the query is intended to search within a topic.

4. To further improve SEE with minimal human effort, we propose a novel multi-oracle setting and a new active learning framework for the hierarchical classification. According to the experimental results, our methods can greatly reduce the human labeling cost.

6.2 Future Work

We are planning to conduct a comprehensive usability study for SEE. In order to diversify the participates, we consider conducting it on the crowd-sourcing platforms such as Amazon Mechanic Turk, where the users are from various areas of the society. Besides, we will design more questions for the users to answer after using SEE, such as how much time and how many keywords they use in their search.

Moreover, we are considering to build an image and video search engine, also with hierarchy. The difficulty will be how to classify the image and video into the categories in the hierarchy.

Another application of SEE is to apply it to a local domain (e.g., a company or a university) to classify all their documents into a predefined hierarchy, and construct a local search service for them.

Bibliography

[1] R.A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*, volume 82. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] J.G. Barbedo and A. Lopes. Automatic genre classification of musical signals. *EURASIP Journal on Advances in Signal Processing*, 2007(1):157–157, January 2007.

[3] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. In *Proceedings of the international conference on Web search and web data mining*, pages 33–44. ACM, 2008.

[4] T.J. Berners-Lee. The world-wide web. *Computer Networks and ISDN Systems*, 25(4-5):454–459, 1992.

[5] A. Cardoso-Cachopo and A. Oliveira. An empirical comparison of text categorization methods. In *String Processing and Information Retrieval*, pages 183–196. Springer, 2003.

[6] C. Carpineto, A. Della Pietra, S. Mizzaro, and G. Romano. Mobile clustering engine. *Advances in Information Retrieval*, pages 155–166, 2006.

[7] C. Carpineto, S. Osiński, G. Romano, and D. Weiss. A survey of web clustering engines. *ACM Computing Surveys (CSUR)*, 41(3):17, 2009.

[8] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Hierarchical classification: combining bayes with svm. In *Proceedings of the 23rd international conference on Machine learning*, pages 177–184. ACM, 2006.

[9] C.C. Chang and C.J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[10] H. Chu and M. Rosenthal. Search engines for the world wide web: A comparative study and evaluation methodology. In *Proceedings of the annal meeting-american society for information science*, volume 33, pages 127–135, 1996.

[11] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[12] N. Daraselia, A. Yuryev, S. Egorov, I. Mazo, and I Ispolatov. Automatic extraction of gene ontology annotation and its correlation with clusters in protein networks. *BMC Bioinformatics*, 8(1):243, July 2007.

[13] L. Ding, T. Finin, A. Joshi, R. Pan, R.S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

[14] P. Donmez and J. G. Carbonell. Proactive learning: cost-sensitive active learning with multiple imperfect oracles. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 619–628, New York, NY, USA, 2008. ACM.

[15] S.A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-6(4):325 –327, april 1976.

[16] S. Dumais and H. Chen. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263. ACM, 2000.

[17] A. Esuli, T. Fagni, and F. Sebastiani. Boosting multi-label hierarchical text categorization. *Information Retrieval*, 11(4):287–313, 2008.

[18] A. Esuli and F. Sebastiani. Active learning strategies for multi-label text classification. In *Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval*, ECIR '09, pages 102–113, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[20] P. Ferragina and A. Gulli. A personalized search engine based on web-snippet hierarchical clustering. *Software: Practice and Experience*, 38(2):189–225, 2008.

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol–http/1.1, 1999.

[22] G. Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of Machine Learning Research*, 3:1289–1305, 2003.

[23] T.H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, 2003.

[24] B. Hayete and J.R. Bienkowska. Gotrees: predicting go associations from protein domain composition using decision trees. In *Pacific Symposium on Biocomputing'05*, volume 10, pages 127–138, 2005.

[25] C.W. Hsu and C.J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[26] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[27] T. Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 137–142, London, UK, UK, 1998. Springer-Verlag.

[28] T. Joachims. Transductive inference for text classification using support vector machines. In *Machine learning-Intertional workshop then conference*, pages 200–209. MORGAN KAUFMANN PUBLISHERS, INC., 1999.

[29] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[30] D.E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[31] M. Kobayashi and K. Takeda. Information retrieval on the web. *ACM Computing Surveys (CSUR)*, 32(2):144–173, 2000.

[32] D. Kuang, X. Li, and C.X. Ling. A new search engine integrating hierarchical browsing and keyword search. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Three*, IJCAI'11, pages 2464–2469. AAAI Press, 2011.

[33] W. Lam and C.Y. Ho. Using a generalized instance set for automatic text categorization. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 81–89, New York, NY, USA, 1998. ACM.

[34] F.W. Lancaster and E.G. Fayen. *Information retrieval: on-line*. Information sciences series. Melville Pub. Co., 1973.

[35] S. Lawrence and C.L. Giles. Searching the world wide web. *Science*, 280(5360):98–100, 1998.

[36] A. Leuski and J. Allan. Strategy-based interactive cluster visualization for information retrieval. *International Journal on Digital Libraries*, 3(2):170–184, 2000.

[37] D.D. Lewis and W.A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[38] D.D. Lewis, Y. Yang, T.G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[39] X. Li, D. Kuang, and C.X. Ling. Active learning for hierarchical text classification. In *Proceedings of the 16th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining - Part I*, PAKDD'12, pages 14–25, Berlin, Heidelberg, 2012. Springer-Verlag.

[40] T.Y. Liu, Y. Yang, H. Wan, H.J. Zeng, Z. Chen, and W.Y. Ma. Support vector machines classification with a very large-scale taxonomy. *ACM SIGKDD Explorations Newsletter*, 7(1):36–43, 2005.

[41] R.L. Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine learning*, 6(1):81–92, 1991.

[42] W. Meng, C. Yu, and K.L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys (CSUR)*, 34(1):48–89, 2002.

[43] T.M. Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 1997.

[44] J. Neyman and E.S. Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference: Part i. *Biometrika*, 20(1/2):175–240, 1928.

[45] S. Nowak and S. Rüger. How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In *In Proceeding of the 11th ACM SIGMM International Conference on Multimedia Information Retrieval*, pages 557–566, 2010.

[46] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy sets and systems*, 138(2):221–254, 2003.

[47] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[48] J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[49] J.R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[50] I. Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, volume 3, pages 41–46, 2001.

[51] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Learning hierarchical multi-category text classification models. In *Proceedings of the 22nd international conference on Machine learning*, pages 744–751. ACM, 2005.

[52] N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *In Proceeding of 18th International Conference on Machine Learning*, pages 441–448, 2001.

[53] M.E. Ruiz and P. Srinivasan. Hierarchical neural networks for text categorization. In *In Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 281–282, 1999.

[54] G. Salton, A. Wong, and C.S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[55] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[56] C.N. Silla and A.A. Freitas. A global-model naive bayes approach to the hierarchical prediction of protein functions. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 992–997. IEEE, 2009.

[57] C.N. Silla and A.A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1):31–72, 2011.

[58] I. Smirnov. Overview of stemming algorithms. *Mechanical Translation*, 2008.

[59] Apache Solr. Apache lucene - apache solr, August 2012.

[60] J. Stefanowski and D. Weiss. Carrot 2 and language properties in web search results clustering. *Advances in Web Intelligence*, pages 955–955, 2003.

[61] A. Sun and E.P. Lim. Hierarchical text classification and evaluation. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 521–528. IEEE, 2001.

[62] P.N. Tan, M. Steinbach, V. Kumar, et al. *Introduction to data mining*. Pearson Addison Wesley Boston, 2006.

[63] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Maching Learning Research*, 2:45–66, 2002.

[64] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[65] G. Valentini. True path rule hierarchical ensembles. *Multiple Classifier Systems*, pages 232–241, 2009.

[66] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

[67] K. Verspoor, J. Cohn, S. Mniszewski, and C. Joslyn. Categorization approach to automated ontological function annotation. In *Protein Science*, pages 1544–1549, 2006.

[68] S. Wartik. Boolean operations. *Information Retrieval: Data Structures and Algorithms, William B. Frakes, Ricardo Baeza-Yates (eds), Prentice Hall PTR*, pages 264–292, 1992.

[69] F. Wu, J. Zhang, and V. Honavar. Learning classifiers using hierarchically structured class taxonomies. *Abstraction, Reformulation and Approximation*, pages 902–902, 2005.

[70] Z. Xu, K. Yu, V. Tresp, X. Xu, and J. Wang. Representative sampling for text classification using support vector machines. In *Proceedings of the 25th European conference on IR research*, ECIR'03, pages 393–407, Berlin, Heidelberg, 2003. Springer-Verlag.

[71] G.R. Xue, D. Xing, Q. Yang, and Y. Yu. Deep classification in large-scale text hierarchies. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 619–626. ACM, 2008.

[72] B. Yang, J. Sun, T. Wang, and Z. Chen. Effective multi-label active learning for text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 917–926, New York, NY, USA, 2009. ACM.

[73] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49. ACM, 1999.

[74] Y. Yang and J.O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[75] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699. ACM, 2002.

[76] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 46–54. ACM, 1998.

[77] S.M. zu Eißen and B. Stein. The aisearch meta search engine prototype. In *Proceedings of the 12th Workshop on Information Technology and Systems (WITS 02), Barcelona Spain. Technical University of Barcelona*, 2002.

Curriculum Vitae

Name: Da Kuang

Post-Secondary Education and Degrees:

Xiangtan University
Xiangtan, Hunan, China
1999 - 2003 B.Sc.

Xiangtan University
Xiangtan, Hunan, China
2003 - 2006 M.Sc.

University of Western Ontario
London, ON
2008 - 2012 Ph.D.

Related Work Experience:

Teaching Assistant
The University of Western Ontario
2008 - 2012

Research Assistant
The University of Western Ontario
2008 - 2012

Publications:

1 D. Kuang, X. Li, and C.X. Ling. A new search engine integrating hierarchical browsing and keyword search. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Three*, IJCAI11, pages 2464−2469. AAAI Press, 2011.

2 X. Li, D. Kuang, and C.X. Ling. Active learning for hierarchical text classification. In *Proceedings of the 16th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining - Part I*, PAKDD12, pages 14−25, Berlin, Heidelberg, 2012. Springer-Verlag.

3 D. Kuang, C.X. Ling, and J. Du. Foundation of mining class-imbalanced data. In *Proceedings of the 16th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining - Part I*, PAKDD12, pages 219−230, Berlin, Heidelberg, 2012. Springer-Verlag.

4 E. Ni, D. Kuang, C.X. Ling. Decisive Supervised Learning. Submitted to *2012 IEEE International Conference on Data Mining (ICDM)*.