Classification of HTML Documents

by

Wei Xie

A dissertation
presented to the University of Ballarat
in fulfilment of the
thesis requirement for the degree of
Master of Computing

School of Information Technology and Mathematical Sciences
University of Ballarat
Ballarat, VIC 3350, Australia

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ballarat to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Ballarat to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Dedication

This thesis is dedicated to my beloved father Guofang Xie, mother Huoan Wu and wife Hua Tian.

Acknowledgements

I am grateful to my principal supervisor, Dr. Musa Mammadov, for his excellent guidance, support, encouragement and tolerance, and his generosity in sharing his time, knowledge and expertise during the two years of my research. I would also like to thank my associate supervisors Associate Professor John Yearwood and Dr. Zari Dzalilov for their excellent guidance.

Thanks to Professor Alex Rubinov for giving me the opportunity of doing research with all these ideal academics in the University of Ballarat.

My regards go to Professor Sidney Morris, Head of school of Information Technology and Mathematical Sciences, Prefessor Mirka Miller, Research Degrees Co-ordinator of the school, Ms. Elizabeth Matuschka and Ms. Gaylene Gravestocks Research Administrative Officers; and all the staff in the University's Research Office.

I would also like to thank all my friends, especially Mr. Arunava Banerjee, and colleagues at the Global Innovation Center, thanks to all of you who have constantly encouraged me during this period of my research.

I appreciate the University of Ballarat and the School of Information Technology and Mathematical Sciences for the financial assistance given to me during the period of my research.

My heart-felt regards and thanks go to my parents and wife for their support, encouragement and guidance during the whole period of my study.

Abstract

Text Classification is the task of mapping a document into one or more classes based on the presence or absence of words (or features) in the document. It is intensively being studied and different classification techniques and algorithms have been developed. This thesis focuses on classification of online documents that has become more critical with the development of World Wide Web. The WWW vastly increases the availability of on-line documents in digital format and has highlighted the need to classify them. From this background, we have noted the emergence of "automatic Web Classification". These mainly concentrate on classifying HTML-like documents into classes or categories by not only using the methods that are inherited from the traditional Text Classification process, but also utilizing the extra information provided only by Web pages. Our work is based on the fact that, Web documents, contain not only ordinary features (words) but also extra information, such as meta-data and hyperlinks that can be used to advantage the classification process.

The aim of this research is to study various ways of using the extra information, in particularly, hyperlink information provided by HTML-documents (Web pages). The merit of the approach, developed in this thesis, is its simplicity, compared with existing approaches. We present different approaches of using hyperlink information to improve the effectiveness of web classification. Unlike other work in this area, we will only use the mappings between linked documents and their own class or classes. In this case, we only need to add a few features called linked-class features into the datasets, and then apply classifiers on them for classification. In the numerical experiments we adopted two well-known Text Classification algorithms, Support Vector Machines and BoosTexter. The results obtained show that classification accuracy can be improved by using mixtures of ordinary and linked-class features. Moreover, out-links usually work better than in-links in classification. We also analyse and discuss the reasons behind this improvement.

Contents

List of Tables

List of Figures

Preface

Classification of objects based on their inherent properties is a common problem encountered in different fields of knowledge. One of these fields is Information Science. In this area, one classifies items based on their content. A great example of this is that librarians categorize books into classes or topics according to their content. This has been and is, done manually when the number of items or books is small. Classification based on content, however, becomes very difficult when the number of items grows. Especially as the World Wide Web (WWW) has been developing, the problem mentioned above becomes more and more apparent. *Yahoo!*'s directory service is always the best example of how classifying huge numbers of documents into proper categories as fast as possible would help people in accessing them, or looking for a particular resource. At *Yahoo!*, the job was mainly done manually by experienced editors in the past. When the number of incoming documents reached millions, the problem went beyond human endeavor. Manual classification is no longer suitable for the requirements. For example, there are tens of hundreds of articles written and published everyday, so editors at *Yahoo!* absolutely have not enough time to analyze, or even just vaguely read through all of these articles. Looking for a specified article from them, therefore, becomes an impossible mission. This highlights the need for the development of automated Text Classification. Simply speaking, the task of automated Text Classification is to automatically classify all items based on their content into pre-defined classes. Moreover, Information Retrieval, which selects a subset of documents relevant to a user query from a large document collection. These two areas are currently very active research areas with variours applications, such as Question Answering and Decision Support Systems.

The main goal of Text Classification is to assign classes or categories to documents based on their content. From a classification point of view, a document is formed by a set of words and is usually represented by the "bag-of-words" method. Furthermore, words are thought of as features because all words characterize the document in which they exist. Classification algorithms, in general, utilize those "informative features" and can be degraded by "noisy features". These "informative features" will be able to help classification algorithms to correctly categorize documents into one or more relevant classes. In the same parlance, a feature set would mean a set of features selected for proper categorization of documents.

In this work, we will concentrate on *multi-class multi-label* classification only. Here, multi-class means that a set of pre-defined categories will be advised by experts; multi-label means that each document in the corpus may belong to one or more class. This configuration is very close to the real world situation that one item belongs to more than one category simultaneously. For example, a document could talk about finance and politics at the same time.

The problem of interest presented in this thesis is conceived as an investigation into classification problems with Web pages as the items being classified. Classification problems of this nature are termed *Web Classification*. Web classification used to be a branch of Text Classification, but recently it became a more independent research problem. Web pages have very interesting properties that traditional documents do not have, this includes meta-data and hyperlinks. Hyperlink is the connection between two Web pages. In other words, it describes a potential relationship between the two connected documents. Meta-data, in general, can be used to describe a Web page itself. Sometimes, combining hyperlinks and meta-data can also tell the relationship among pages in a possibly more accurate way. Due to the diversity of meta-data (there are many different types of meta-data), people have been trying intensively to find some better use for it. Researchers also consider the possible use of hyperlinks, although they would like to use the combination of meta-data and hyperlinks. Some interesting results and improvement on performance of classification process have been observed, However the classification procedure became much more complicated in either the phase of pre-processing and/or learning, and more computational power may be needed. In all these cases, people often ignore the rich information provided by hyperlink itself. Hyperlinks can construct a whole picture of the relationship among all documents in the corpus. This information is already sophisticated enough to improve the performance of classification. Hence, a better way of using hyperlink information is desired. The merit of the approach which developed in this thesis, is its simplicity compared with other existing approaches.

This section introduces the classification problem investigated in this work and presents the research objective of this work.

Research Objective

From the earlier discussion, it is evident that our research on Web classification can focus on looking for a better way of using this extra information provided by Web pages, especially hyperlinks, based on existing classification algorithms. Such an approach would take into account the influential power of relationship between two on-line documents. Hence, the main emphasis of this work is:

- To construct a database which contains the information that satisfies the requirements of our project. We will generate a new data collection based on the internal

WWW network of the University of Ballarat, because currently there is no such Web collection which satisfies the criteria of containing both link information and multi-class multi-label. The existing Web collections are usually multi-class single-label. Other generic data collections contain only ordinary features (words) without link information, such as WebKB and Reuters. The new database should contain not only words in all documents, but also the connections among the documents. From the database, we will generate datasets which contain the proper representation of hyperlink information, some documents will be selected to form new datasets. All selected documents in the corpus then will be manually classified into one or many classes based on their content. All link information, namely linked class features, are represented numerically. The numbers should not be too dominant compared with other features because we would like to see how ordinary features cooperate with the link information. Hence, linked class features will be normalized based on the entire corpus.

- To study how the variety of link information can be used to improve the accuracy of Text Classificaiton processes with different classification algorithms.

Outline of the Research Methodology

From the above discussion, the methodological approach for such a study would encompass the following steps (further elaboration is provided in chapter 4):

1. *New database formation* - Web pages will be retrieved from the intranet of the University of Ballarat. All words and special properties of Web pages will be kept in the database for possible future use.

2. *Data modeling and dataset generation* - Hyperlink information will be modeled in various ways. Based on the models, datasets with or without hyperlink information will be generated accordingly.

3. *Selection of well-known classification algorithms* - Well-known classification algorithms will be tested on our new multi-label datasets without using the hyperlink information. This process will provide us a baseline performance of these classification algorithms on traditional Text Classification datasets. This baseline will be used for comparison purposes later, as a test-benchmark, to see the effectiveness of both hyperlink information and classification algorithms.

4. *Using the same algorithms on the problem under investigation* - The algorithms selected in the previous step are to be used on the datasets that include hyperlink information. The results will be compared against the test-benchmark we obtained from the previous step.

Summary of chapters in the thesis

Having introduced the research problem, objectives and methodology, this section aims at providing a brief overview of the content of this thesis. This exposition is split into chapters based on the materials discussed. The chapters with their brief summary are list below:

- *Chapter 1: Introduction to Data Mining, Multi-label Classification and Web Classification* - An introductory chapter which lay the foundation structures for further reading. From "Introduction to Data Mining", " Development of Machine Learning" to "Multi-label Text Classification", it provides readers with a broad picture about the areas that have been traversed. Based on these concepts, more light is thrown on the domain under investigation.

- *Chapter 2: Literature Review* - The aim of this chapter is to provide readers with a view of Web Classification and its basis, the Text Classification literature. The chapter begins with a brief discuss on modern Text Classification procedures, then more details on each step of this procedure are also presented. It is further followed by a description on recent development of Web Classification techniques. The chapter ends with an account of the evaluation measures currently being widely used.

- *Chapter 3: Informativeness of Features* - This chapter aims at providing a view of techniques of modeling Text Classification data. Mainly, we will concentrate on introducing the widely used technique Information Gain (IG). We also present different interpretations of the IG formula; and how different interpretations would affect the measurement of informativeness of features and the consequence on the classification performance.

- *Chapter 4: Preparation for Text Classification Processes* - We present the methodological steps of our research in this chapter.

- *Chapter 5: Classification Algorithms* - This chapter provides the reader with an overview on the classification algorithms we used in this work. An analysis of the goodness and weakness of each of these algorithms will be given as well.

- *Chapter 6: Experiments and Results Analysis* - After introducing the algorithms, this chapter presents the results obtained from experiments using the above-mentioned algorithms on our new formed Text Classification datasets. An analysis on why we get such results will be given at the end of this chapter.

- *Chapter 7: Conclusion and future work* - This chapter aims at providing the conclusions that can be drawn from the work. Furthermore, it also outlines the possible future research directions.

Chapter 1

Introduction to Data Mining, Multi-label Classification and Web Classification

In the modern era of the Information Age, *Knowledge Discovery* from information has become more and more attractive in so many different areas, such as science, engineering, medication and business. Nowadays, people are getting more abundant information from pre-processed data than ever before. There are so many databases which now have huge volumes of data stored in them that have been generated for different areas. A few examples are, the use of bar-codes for tracking commercial products being bought or sold, computerized transaction methods for different government and non-government institutions and also the increasing capacity especially in the field of scientific research to collect complex images like those of weather and geo-science patterns by geo-synchronous satellites.

Furthermore, because of the advent of World Wide Web (a.k.a WWW), we have seen information systems flooded with large volumes of data from email, news-groups, research materials, and other communicating and sharing information. Thus, data related to customer behavior, market trends and even data from their own operational statistics is getting harder and harder to be understood by companies or organizations. Moreover, scientists and engineers have found themselves looking at volumes of complex datasets like weather-pattern change or effects of global warming generated from images taken by satellites. Before it can be used, this data has to be converted to 'information' (which is "processed data"); and then further refined to be 'knowledge" which people could. The generation of knowledge from databases is the main focus of *Data Mining* which has come to exist as an intersection of different areas of knowledge especially *Machine Learning*, *Artificial Intelligence*, *Database Technology* and others. Machine Learning can further be sub-divided into *Supervised Learning* and *Unsupervised Learning*.

The aim of this chapter is to give a brief introduction to Classification as a branch of Supervised Learning. Further, multi-label classiciation is introduced. Utilizing these concepts further insight into the investigation problem is provided. In [45], the fascinating question of how machines might learn from examples is raised from the Data Mining paradigm during the investigation of automated learning. The main area presented in this work is in this area of data mining, in particular supervised learning. In what follows, a dataset or database would typically consist of "examples" in general cases and in particular "documents" in Text Classification datasets.

Cognitive Intelligence, the way of the human brain performs operations, such as perceptual recognition, generalization, recall and thinking [56], have always fascinated researchers. In this type of problem area, scientists were concerned with the development of a machine that mimiced brain functions. Amongst the models proposed the perceptron model by Rosenblat in 1958 [56] started the era of "*Intelligent Systems*". Scientists looked for ways to make machines learn new knowledge from external examples and the method of Empirical Learning or Inductive Learning was born. Empirical Learning was typically accomplished by supplying external examples to the learning procedure and constructing rules during the process of learning. Empirical Learning can be further sub-divided into Supervised Learning and Unsupervised Learning.

In supervised learning, we usually provide examples of the form (x_i, y_i) to the learning program and ask it to learn a function f such that $f(x_i) = y_i$ for all i. The function f is also expected to capture general patterns during the training phase, so that it can later be applied to recognize new examples. In general, x_i can be a complex representation of an object, event or pattern; y_i can be considered as a categorical description for x_i. This process is called Supervised Learning because we supply y_i to the learning procedure. In contrast, Unsupervised Learning is only given the examples x_i and asked to find some regular patterns, so that the program can generate values for y_i. Most unsupervised learning algorithms search for 'regularities' like clusters of x_i. In this work, we will only concentrate on those methods which belong to Supervised Learning. *Classification* is a part of Supervised Learning that utilizes the knowledge generated (usually from a training phase) to classify an object which is under investigation. It can also be thought of as Knowledge Discovery which ultimately attempts to classify an object as belonging to a particular category or class. Since the main goal of this work is based on Classification, an introductory section to it is included next in the treatise. Furthermore, a follow-through into Multi-label Classification is included as it is one of the main criteria for the problem under investigation.

Throughout the thesis, every example will contain two types of information, "features" and "classes". For example, in the case of "documents", "features" are words in documents, and "classes" are categories to which the documents belong. Another example, in the case of Web pages, features include not only words in the Web pages, but also other properties of Web pages, such as *hyper-links* and *meta-data*. In this work, a document or Web page is represented by a vector of its features.

1.1 Multi-label Classification

Classification has previously been introduced as a part of *Supervised Learning*. In particular, Classification is to classify objects into their related categories. The categories are topics that are chosen by experts based on the content of the objects. As an example, in a library, librarians choose a relevant topic or a class to which a book belongs based on the content of the book, so that it can be coded and properly placed in the collection. Manual classification has been introduced and has been used for long time. It is, however, no longer preferred because of the growing number of items. In such scenarios, automated classification is being used and further investigated. Automated classification generally uses a mathematical function for mapping documents to a discrete set of classes. Hence, from a mathematical perspective, a classifier is a mapping from feature space ($\mathcal{X}$) to a discrete set of classes ($\mathcal{Y}$). Formally a classifier can be stated as follows:

Given a set of training data $(x_1, y), \ldots, (x_i, y)$, a classifier is a function $\mathcal{H} : \mathcal{X} \rightarrow \mathcal{Y}$ that essentially maps an object $x_i \in \mathcal{X}$ to its category or class $y \in \mathcal{Y}$; where x_i is a vector representation of the object and y is the class-label which might be '1' or '0' for binary classification, or have certain values for multi-class and multi-label settings. To explain with an example, consider the problem of email classification of 'spam' and 'non-spam' emails which is essentially a binary classification problem. In this case, x_i is some representation of the email message and y is the class ('spam' or 'non-spam') [62].

Another application of classification is Web databases. In these databases, every document is collected randomly from the Internet and usually represented by its content. We then select related topics based on the content. All documents in the database must be pre-classified into those cateogies properly according to their content. For example, the directory service of Yahoo! contains links to pre-classified documents segregated by each document's content. Further, news-groups would want to classify news items based on the interest of their customers. Email filters would also want to classify incoming or outgoing emails as 'spam' or 'non-spam' emails. In the next section, we wil introduce the different configurations of classification found in literature.

1.1.1 Classification Configurations

We found three main classification configurations in the literature:

1. *Binary configuration*: In this configuration, ($\mathcal{Y}$) is characterized by two broad classes, such as 'spam' or 'non-spam' as we mentioned at the end of previous section. All examples in this setting will be assigned a positive or negative integer based on whether or not they belong to the corresponding class. That is, if an email is spam, then '1' is assigned; otherwise '-1' is assigned. Because classification is in the computer science area, the class assignment may use '1' and '0' instead. Although

this configuration has been considered as the simplest method in the literature, it is still worthwhile studying it seriously because of the fact that other complicated approaches use it as the basis to break complexity down into binary classes.

2. *Multi-class configuration*: In the real world, examples generally have different topics or belong to different classes based on their content. Multi-class configuration, therefore, has to be introduced into this area. As multi-class setting attempts to classify examples based on their main topic, so that examples can belong to one and only one class. In both binary and multi-class configurations, the goal of learning is to find a classifier to minimize the probability that $y \neq \mathcal{H}(x)$ for a new example, where y is the class and x is a vector representation of the object and $\mathcal{H}$ is the required classifier [62].

3. *Multi-label configuration*: This configuration is much closer to real life. One object can belong to one or many topics. Binary and multi-class settings will only find the most suitable topic from many topics based on the different criteria of each configuration and the content of the target which is being classified. In this case, the two previously mentioned configurations would not be enough to describe the real relationship between an object and its categories since it is not proper to model or classify the object according to only one single class or label. One may therefore introduce *Multi-label* configuration to resolve this problem. For example, a document may talk about meteorology; simultaneously it may also talk about how computerized processing could involve meteorology or weather prediction. Hence, the document would belong to the two classes, 'meteorology' and 'computer-aid-processing'. Thus it is necessary to allow an example to belong to more than one class. For multi-label configuration, the goal is to find a classifier $\mathcal{H} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$ giving a real number. The interpretation for the classifier is that for a new example (x, Y) where $Y \subset \mathcal{Y}$ (since an example may have many classes), ordering of the classes (or categories) in $\mathcal{Y}$ are performed for each example with the possibility that appropriate classes will appear at the top of the ranking. Thus, if $\mathcal{H}(x, c_1)$ has a higher value than $\mathcal{H}(x, c_2)$, then it can be said that class c_1 is to be ranked higher than c_2 where $c_1, c_2 \in Y$ [62].

Above, we have introduced three different classification configurations from the simplest, binary, to the more complex multi-label configuration. In this work, we will focus on *Multi-label Classification*, so we will delve a bit deeper into multi-label settings in the next section.

1.1.2 Multi-label classification

We have introduced Multi-label configuration in the previous section. In this section, we will further discuss this concept. As can be found, even in this document, different

sections talk about different topics to illustrate a common goal. Hence looking at it from Text Classification point of view, this document can be classified as being related to "Data Mining", "Text Classification" and "Web Classification" (assuming that these are the choices for the classes). Thus, it is evident that most documents have multi-label characteristics.

An illustration of "bag-of-words" representation of documents, generated from the University of Ballarat Web database which is a multi-label Text Classification and Web classification database that we collected and generated based on the University of Ballarat's WWW intranet, is shown in Table 1.1. As we can see from the table, each document is represented as a "bag-of-words", where the last three columns on the right denote the classes of the document. Each row in the table corresponds to a document. Because we use multi-label settings here, more than one class can be seen. The middle column in the table shows the feature vector or word vector, in other words, the words that are used in each document. Here, the representations of words are their weight which is calculated using *TFIDF* (we will explain it later in chapter 2). Weight is a numerical entity that is understood by the algorithms. Be aware of that, some features have value '0.00000' as their weight. This means that the feature is not present in the corresponding document.

Doc#	Words are used in documents	**Classes**
292	0.000000 0.000000 0.000000 0.000000 0.000000 0.491215	6
6577	0.014042 0.000000 0.011892 0.000000 0.000000 0.000000	6 0 8
24132	0.092618 0.000000 0.000000 0.000000 0.052626 0.000000	5 6 10
21655	0.004591 0.042391 0.000000 0.000000 0.000000 0.000000	2 0 4
7048	0.019067 0.000000 0.016147 0.000000 0.000000 0.000000	5
5498	0.380056 0.000000 0.000000 0.000000 0.198658 0.198658	4 6 8
16998	0.000000 0.000000 0.029523 0.000000 0.150264 0.000000	3 9

Table 1.1: Sample from UB Web database.

As you can see from the illustration, multi-label classification deals with examples that are assigned to multiple classes (topics). Thus, in such cases, a ranking of the classes is performed for each example, so that the relevant classes appear at the top. Because classification becomes much more difficult in this configuration, attempts to tackle this problem using a binary classification scheme can be found in literature. Algorithms try to break this complexity into numerous classes and then classify using one class verses the rest approach.

In this section, we will explain and illustrate classification configurations and the multi-label classification criteria we used in this work. Keeping these concepts in mind, we will investigate the research problem of this work more fully in the next section.

1.2 Web Classification

The advent of the World Wide Web (WWW) has rejuvenated interest in Text Classification problems. Currently, the Web contains more than a billion pages that are inter-connected with each other via hyperlinks. This makes the task of locating specific information on the Web increasingly difficult. A study which was conducted by Chen and Dumais in 2000 [11] showed that people often prefer navigating through directories of pre-classified content, and that providing a categorical view of retrieved documents enables them to find more relevant information in a shorter time. The common and wide use of category hierarchies for navigation that are supported in Yahoo! and other major Web portals has also shown the practical needs for hypertext or Web classification. Thus, categorizing these on-line documents into meaningful semantic categories is a rewarding and challenging research problem.

1.2.1 Research Problems Raised

As we discussed in the previous section, multi-label classification is a more complete solution to Text Classification problems although it may need more computational power than other configurations. It seems that, Web categorization can fit perfectly into multi-label configuration. Thus, we could model the Web pages into multi-label classification configuration, and then apply a classification algorithm on the generated datasets. Web pages, however, are different from normal documents. We should make use of the advantages of the differences that distinguish Web pages from normal documents, to improve the performance of the classification process in terms of accuracy, efficiency and others.

Hyperlinks, contents of linked documents, and meta-data (such as relational attributes) about related Web sites are the differences between a Web page and a normal document. All of these provide rich sources of information for hyperText Classification. That is not a major concern in traditional Text Classification. Thus, Web classification poses new research challenges because of the rich representation of a document and the connectivity between documents, or the combination of both of them. The question of how to effectively use such information is raised and becomes important. More specifically, how useful or reliable are hyperlinks when they are used in predicting topic labels of Web pages or Web sites? What kind of meta-data about Web sites could be exploited and how useful are they, if available? What learning algorithms are more powerful for discriminating informative links from noisy links with respect to classification? What algorithms are more robust in terms of dealing with a noisy vocabulary of hundreds of thousands of words?

1.2.2 Research Status Overview

These open problems have just begun to be addressed in current Web classification research. Chakrabarti et al. in 1998 [9] reported on their experiments on a patents database and a subset of Yahoo!. In their work, they treated the text from neighboring documents (linked to/from the target document) as local text of the target document. However, comparing the results of both using and ignoring the links, they observed worse performance when links were userd in the process. They also developed an iterative algorithm which labeled test documents using the labels of surrounding documents. The labels are either given to algorithm by an expert (human), or induced from the previous iteration of the process. This iterative classification process showed improved accuracy. It is, however, not clear how well their ideas can be generalized to Web classification tasks because all their hypotheses were tested on a Patents database which might have different properties than a Web corpus.

Oh et al. in 2000 [48] also developed an algorithm for exploiting hyperlink information based on the work of [9]. The algorithm is a single pass algorithm which labels the test (target) documents by using only the neighboring documents that are similar to the target document. Both the induced class, and the text of the similar neighboring documents were used for classification. Improved performance of their classifier on an encyclopedia corpus (with hyperlinks) was observed. Again, whether or not it will work well on a Web collection is still unknown.

Slattery and Mitchell in 2000 [67] studied the use of hyperlinks from a different angle. They used FOIL (First Order Inductive Learner) [53], a relational learning algorithm to exploit the relational structure of the Web, and a Hubs & Authorities style algorithm [34] to exploit the hyperlink topology. By combining these two algorithms they had improved classification accuracy over FOIL, on a corpus of university Web pages in three classes.

Yang et al. in 2001 [26] summarized six hypertext regularities for their future research in the hyperText Classification area. They employed three well-known classification algorithms, Naive Bayes, Nearest Neighbor and FOIL; and applied these algorithms to three different Web corpus, two consist of company Web sites and one consists of university Web pages. Their study showed that naively using hyperlink information can hurt classification performance, and that carefully using the information from the linked neighborhood could be extremely helpful. They also found that the meta-data is a useful source of information, and combining the meta-data and the text of a Web page could result in better performance, as they observed.

While the work listed above suggests interesting ideas with some observations, hypertext categorization remains an under-explored area. More exploration of alternative approaches and different applications are needed before we could make our conclusion.

In this chapter, we have given an overview to both Text Classification and Web

Classification. In the next chapter, we will journey into these two areas to explore current research in these areas.

Chapter 2

Literature Review

In chapter 1, we discussed the general concepts and the importance of text classification for our information age. We also showed that the need for Web Classification is increasing, especially over the past a few years. On-line documents (mainly Web pages) provide alternative ways or extra information to traditional text classifiers. Web classification is a part of text classification, it is attracting more and more attention from the public. This chapter reviews the literature on both text classification and Web classification. For comparison of classification results, evaluation measures are used. They will be addressed in this chapter as well.

2.1 Text Classification

Librarians invented the listing of items (such as books) based on the subjects for proper classification and inclusion into the collection. The items would be analyzed and an appropriate description of these items would be given. The information is to represent the content of the items. Librarians based on their own knowledge, therefore, assign a reasonable class or category to the item. Firstly, people used the author name and title in the list. Searching through the catalog to find books was indeed an improvement. The problem here is that users must be able to remember the author's name and the title of the book for which they are looking. Secondly, topic based list was invented. Items were indexed based on their topics. This time, people needed only to provide the keywords for the topic in which they are interested to display the topic-relevant books. Choosing proper or correct keywords, however, becomes a difficulty. In 1950, with the advent of computerized information searching, indexes were explored further with the strategy of using the terms in the document collection to create an index. The problem was that so many irrelevant terms were retrieved while attempts to retrieve relevant terms [60]. This also made the collection of terms huge in size.

In the information age, however, these methods can no longer meet the requirements. This is due to the fact that many more books and other kinds of publications have been or are being published day by day. Especially when the Internet came, a huge amount of information has been put on-line. For example, in July 2000 the WWW network contained approximately 2.1 billion Web pages and 4 billion Web pages by early 2001. Thus manual classification is not applicable at all. Hence, researchers in *Artificial Intelligence* (AI) and *Machine Learning* in particular, started investigating various ways of classifying documents to its class(es). They also defined the term Text Classification as "it aims to automatically classify text documents into pre-defined classes or types based on their contents". So many techniques have been applied for text classification purposes. We list only some of them because describing all of them is voluminous.

- Regression Models (Yang and Chute 1992)
- Relevance Feedback
  - TFIDF (Rocchio 1971, Salton 1991)
  - Probabilistic TFIDF (Joachims, 1997)
- Decision Tree (Quinlan 1986, Koller and Sahami 1997)
- Bayesian Probabilistic approaches
  - Naive Bayes (Lewis and Ringuette 1994, McCallum et. al 1998)
- Induction Rule Learning (Apt, Damerau and Wesis 1994, Cohen and singer 1996)
- Nearest Neighbour (Mitchell 1997, Yang and Pederson 1997, Yang 1999)
- Neural Networks (Weiner, Pederson and Weigend, 1995)
  - Self-Organising maps (Kohonen 1989, Lin et. al 1991, Kaski et. al 1996)
- Support Vector Machines (Joachime 1998, Dumais et. al 1998)

Above are some well-known algorithms that have been developed for text classification purposes. In the next section, we will survey the literature to delve into the current development in the text classification area. The literature review will show related work in text classification area. It will cover some of the processes of classification, such as data modeling, pre-processing, document indexing and dimensionality reduction. Classification algorithms, however, will not be covered in this section. We will discuss this issue in chapter 5.

2.1.1 Different Models in Text Classification

In the previous section, we discussed the reason why the modern classification process is usually done by using computers, and it is called automated classification. Due to computerized algorithms not taking any kind of human readable documents as input directly, such as .txt, .pdf, .html and postscript, we must translate data into machine readable format before we enter the text classification procedure. There are many different feature representation models that are being used in text classification.

- *Bag-Of-Words:* The most commonly used representation model for text classification. In this model, words are extracted from documents to form a vector or a collection of terms representing the document. In this form, it is generally known as a "feature (or document)" vector. Until now, the format of data is still in "text". Following the rules of Vector Space Model (VSM) [60], all features in the vector should be represented by their weights corresponding to the documents. Weights here are usually represented by some methods, such as term frequency and inverse-document frequency. These methods will be further discussed in the later sections.

- *Bag-Of-Concepts:* In the traditional bag-of-words model, the content of a document is modeled as the set of words comprising the document. In contrast, the bag-of-concepts model proposed by Sahlgren and Coster in [57] relies on the concept that semantics or contents of a document may be viewed as the union of the meaning of words it contains. Similar to Feature Selection, the authors propose Random Indexing which is a vector space methodology for producing concept vectors. In such cases the word-matrix of bag-of-words representation is transformed to a co-occurrence matrix containing weights for each word used for each concept. Hence each word here would be represented as a bag-of-concepts. This representation was tested on Reuters-21578 text-collection with the Support Vector Machines classifier. The results reported show that the bag-of-words concepts outperforms bag-of-concepts approach if one considered all classes of the dataset, however, the bag-of-concepts approach outperforms the bag-of-words approach when only the top ten classes are considered. Also a combination of both was tested with the same classifier which showed an improvement in classification accuracy.

- *Word-Cluster:* The word-cluster representation was proposed by Baker et. al [2] based on the distributional clustering approach of Pereira et. al [50]. In this representation, words are viewed as distributions over document classes. Distributional clustering developed in [2] had previously shown good results in Language Modeling. The main concept highest on the fact that if two different words have a similarity vote for a particular classes or category then joining those words to form one unit would achieve 'dimensional reduction' required as well as capturing the necessary semantic structure for classification. Also it is shown that it does not hurt the performance but improves it.

Distributional clustering tries to measure the similarity based on the classes. An example of the clusters formed from 20 NewsGroups dataset has been shown in [2]. It is argued that if a particular word is depicted according to the cluster rather than the actual classes, then the performance of the algorithm improves significantly. This approach has been tested in [2] with a Naive Bayes algorithm, which gave better accuracy. Further more, in [4], the word-cluster representation calculated by using Information Bottleneck [70] was applied in association with the Support Vector Machines classifier. The comparison was carried out on 3 datasets for text classification. In these experiments, the word-cluster model outperformed bag-of-words in 20 NewsGroups dataset only. In the other two datasets, Reuters-21578 and WebKB, bag-of-words outperformed word-cluster. This was mainly attributed to the structure of the other two datasets. Thus it can be said that word-cluster is affected by the structure of the dataset.

- *Hierarchical Approaches:* It is possible to apply standard classification techniques to hierarchical classification by constructing a structure that takes one class for every leaf in the hierarchy. Koller and Sahami [35] constructed a hierarchical set of classifiers by training separate classifiers at each internal node of the tree (classification hierarchy). A document is then classified in a top-down fashion, starting at the root category and then selecting the best sub-categories until the bottom level is reached. The divide and classify approach of the algorithm uses a hierarchical structure to break up a large problem space into smaller sub-spaces by dividing the classification task into a set of smaller classification problems corresponding to splits in the hierarchy. As an example of this approach, the task of classifying documents into categories of animal husbandry, crop farming, computer software and computer hardware become much more simpler if it is determined whether a document belongs to computer and then into the next hardware or software. Applied on Reuters-22173 dataset, the results showed that in some cases along the nodes of the tree, hierarchical approaches outperformed the vector representation of features. Furthermore, the error reduction was greater than in the vectorized representation as reported.

You might already be aware that data modeling is a very important process and preparation for computerized data processing. It converts raw data into some kind of computer-usable information which allows computerized algorithms to take and understand the input. Moreover, processing results can vary based on the different data modeling techniques. Choosing a suitable data modeling technique, however, can be critical to the rest of classification procedure. In the next section, we will introduce the common procedures in text classification and show how they use the modeled data.

2.1.2 Procedures in Text Classification

After we translate data from a human readable format to machine readable information, we now are ready to enter the text classification process and discuss each of these important steps.

The Text Classification process typically consists of pre-processing, document indexing, dimensionality reduction and the classification process. The procedure starts with pre-processing the modeled data which often involves the stop-words removal and stemming (turning words back to their rhetorical root). Next, the documents will be indexed by vectors. A vector contains each word in the corresponding document. The words in the vectors are represented by their weights. Weights vary from document to document and are also based on different weight calculating methods. After that, because a document may contain a huge amount of unique words, dimensionality reduction must be introduced to shrink the size of datasets and reduce the computational difficulty. Hence, the efficiency of text classification is closely related to the dimensionality reduction process. Dimensionality reduction also affects the performance of the classification process too. Words can characterize categories, so that the words in a document can bind the document to a particular class. In this sense, when dimensionality reduction gets rid of some words from the entire collection, it may eliminate those discriminative words as well. The last step is to classify those documents into their corresponding class(es) by considering the inter-relationship between words in the documents and classes. Now, we will introduce those processing steps in more detail.

Pre-processing of Data

At this stage, terms that do not provide any information about class selection are to be removed. Two concepts should be introduced here:

1. *Stop-word Removal:* [23, 76] A document can contain a lot of words. Some of those words are not meaningful to text classification. This kind of words are called stop-words. Stop-words carry no information and are very frequently used in documents, such as articles, pronouns, prepositions and conjunctions. For example, if we have a sentence "hello, the world", "hello, the" are stop-words; and only "world" is carrying some information and will be useful to the text classifiers.

2. *Word Stemming:* [52] In addition to getting rid of all the stop-words, we must take care of all the words that have the same conceptual meaning. This means that we have to return all this kind of words back to their own stems. A stem is the portion of a word which is left after the removal of its affixes, such as prefixes and suffixes. This process is called word-stemming. Word-stemming will group words that share the same morphological root. For example, "profess" is the stem for many words:

"professed", "profession", "professional", "professionally" and "professionalism". This process will reduce the number of words and improve the significance of those words that share the same conceptual meaning in a document.

As for stemming, the first stemmer was published in Lovins paper [39]. Sometime later, Porter [52] wrote his own stemmer which became the defacto standard stemmer and now is widely used for text classification purposes. In our work, Porter's stemmer was used for stemming words to its morphological root.

These two types of words are meaningless for most cases and appear very frequently, this increases the dimensionality of the dataset unnecessarily. The increased number of variables will cause large computational times as well as computational complexity, hence they need to be removed to make the dataset clear of unnecessary words.

Indexing of Data

The dataset will now have terms that characterize a document and can be used to generate class information. One way of classification modelling is to model the associations between documents and classes. "Document" can be represented by the terms within it. "Class" here means the topics that humans assign to the document. Both of them can be represented by the well-known and commonly used VSM model already mentioned in section 2.1.1. If the two representations of a document are merged, we can model the association between features and classes . To allow mathematical programming to carry out the text classification process, we must generate an indexing scheme for the terms in the documents. Eventually, these indexes (numbers) will be put into classifiers to find out the association between terms (documents) and classes.

In VSM, a document d_j is usually represented by a vector of feature weights

$$d_j = \left[(w_{1,j}, w_{2,j}, \ldots, w_{|\mathcal{F}|,j}\right]$$

,

where $|\mathcal{F}|$ is the total number of features that appear in the entire document collection, and $0 \leq w_{i,j} \leq 1$. $w_{i,j}$ is the weight of word i in document j. Hence, if $w_{i,j} > 0$, it means that word i exists in document j. For a set of features $\mathcal{F}$, there must be at least one feature i which has weight $w_{i,j} > 0$ in document j. Combining all documents' feature vector produces a documents' matrix $\mathcal{W}$. The matrix is usually very sparse and the dimension of $\mathcal{W}$ could be very large. This, of course, introduces a difficulty of text classification, that is high dimensionality of the feature space.

There are four different ways to calculate weights for features, namely Binary Weights, Term Frequency (TF), Inverse Document frequency (IDF) and TFIDF.

- *Binary Weight*: If word i presents in document j, the weight $w_{i,j} = 1$; $w_{i,j} = 0$ otherwise. This type of weight cannot tell how frequently a word is being used in a document. This means the weight cannot show the importance of a word.

- *Term Frequency (TF)*: If word i appears in document j, the weight $w_{i,j}$ is equal to the number of times that the word i is used in document j. TF is affected significantly by the length of documents. Longer documents will usually get a higher value for a particular word's weight than those in shorter documents. TF also does not consider how frequently a word is used in the entire document collection. The fact is that, if a word is used more frequently within all documents, the word is usually less meaningful for documents.

- *Inverse Document Frequency (IDF)*: For feature t_i in document d_j,

$$idf(t_i, d_j) = \log \frac{|\mathcal{D}|}{|\mathcal{D}(t_i)|}, \tag{2.1}$$

  where t_i is the term; $|\mathcal{D}|$ is the total number of documents in a collection and $|\mathcal{D}(t_i)|$ is the number of documents that contain term t_i within $\mathcal{D}$. This method does not measure how important a word is in a particular document.

- *TFIDF*: Combining TF and IDF shows us how discriminative a word is both in a particular document and in the whole document collection. This process is based on two empirical observations:

  – The more frequently used words in a document more likely represent the category which the document belongs to;
  – The more frequently used words in an entire document collection more likely tell nothing about the category of a particular document.

  For feature t_i in document d_j, TFIDF is defined as

$$tfidf(t_i, d_j) = tf(t_i, d_j) \times idf(t_i, d_j), \tag{2.2}$$

  where $tf(t_i, d_j)$ denotes the frequency of a word t_i in document d_j.

  TFIDF values are usually normalized by the cosine normalization. This normalization will limit the values of the weights between 0 and 1.

$$w_{i,j} = \frac{tfidf(t_i, d_j)}{\sqrt{\sum_{k=1}^{|\mathcal{F}|}(tfidf(t_k, d_j))^2}}. \tag{2.3}$$

- *TFICF*: In [30], another indexing technique was proposed. It is based on the TFIDF approach. This algorithm was named Term Frequency - Inverse Category Frequency (TFICF). It takes into account terms for each particular category.

Dimensionality Reduction

The basic question in classification is to know how to determine the relevance between a document and a topic (class). After initial pre-processing, the given dataset will contain term-weights. The next requirement is to reduce the dimensions of the dataset as much as possible. There are two main approaches to dimensionality reduction in text classification *Feature Selection* and *Re-parametrization*. From the name we know that feature selection is to select the best features to form a subset of the original dataset; and re-parameterisation will generate a new dataset based on the original dataset by using transformation or combination techniques. For examples, *Document Frequency*, *Information Gain*, χ^2 *statistic*, *Mutual Information* and *Term Strength* are the most commonly and widely used feature selection techniques. The Latent Semantic Indexing (LSI) [20] is based on the re-parametrization approach.

1. **Feature Selection:**

   Feature Selection is applied when applications that require to account for hundreds of variables in classification are encountered. The main objectives of features selection can be summarized as [28]:

   - Reducing training and utilization times
   - Reducing dimensionality problems to improve prediction performance, efficiency and effectiveness.

   In [28], it is further stated that "In the text classification problem, the documents are represented by a 'bag-of-words', vocabularies of hundreds of thousands of words are common, but an initial pruning of the most and least frequent words may reduce the effective number of words to 15,000". Moreover, in [28], "constructing and selecting subsets of features" are also considered as a dimensional reduction since they reduce the feature space by combining features. Five well-known dimensionality reduction techniques found in the literature are listed below:

   - *Document Frequency (DF):* The removal of infrequent terms from documents is the main aim in this method. Based on the assumption that rare terms in documents are not useful for classification, these terms are removed. A frequency threshold is set by the user and any term that might fall below that threshold is removed. In [32], it showed that although the most redundant terms are claimed to have been removed, they still contain some information content that might be useful for text classification purposes. The advantage of this method is its simplicity and the ease with which it scales up to very big datasets.

- *Information Gain (IG):* [80] Information gain measures the term-goodness criteria for category prediction. It measures the information gained by knowing the presence or absence of a term in a document using an entropy-based formula. Intuitively, IG measures the average entropy reduction caused by occurrence or non-occurrence of a term t with each category C_i. After calculating the IG values for all terms, those terms that have IG values below a predefined threshold will be removed. The dimensionality of datasets is then reduced. Because we are using this measure in the entire work, we will discuss this issue more in details in chapter 3. The IG formula is given below:

$$\begin{aligned} IG(t) &= -\sum_{i=1}^{|C|} P(C_i)\log P(C_i) \\ &\quad +P(t)\sum_{i=1}^{|C|} P(C_i|t)\log P(C_i|t) \\ &\quad +P(\bar{t})\sum_{i=1}^{|C|} P(C_i|\bar{t})\log P(C_i|\bar{t}). \end{aligned} \tag{2.4}$$

  where $P(t)$ is the probability of term t appearing in the entire corpus and $P(\bar{t})$ is the reverse case; $P(C_i)$ is the probability that class C_i was hit by documents within the whole corpus; $P(C_i|t)$ and $P(C_i|\bar{t})$ are the probability that class C_i was hit by documents when term t is present and absent respectively.

- χ^2 *statistic:* Calculating the lack of independence between a term and a class is the main focus of this measure. A contingency table for each term t and each class c is defined here with the following meaning for the cells:

A	B	A+B
C	D	C+D
A+C	B+D	N

Table 2.1: two-way contingency table for the Chi-square Statistic

  - A is the number of times the term t and class c co-occur.
  - B is the number of times the term t occurs without c.
  - C is the number of times c occurs without term t.
  - D is the number of times neither occurs.
  - N is the total number of documents.

  Thus the term-goodness criterion in χ^2 is defined as:

$$\chi^2(w,c) = \frac{N(AD - CB)}{(A+C)(B+D)(A+B)(C+D)} \tag{2.5}$$

In [80], while generating feature selection, average of the χ^2 value for each term and each category and also maximum χ^2-value of each term for the full category set was considered.

- *Mutual Information (MI):* Mutual information uses the same two-way contingency table used by χ^2 (Table 1) with the same correspondence for the each cell. Considering the contingency table MI is defined as:

$$\begin{aligned} MI &= \log \frac{P(w \wedge c)}{P(w)P(c)} \\ &\approx \log \frac{A \times N}{(A+C)(A+B)} \end{aligned} \tag{2.6}$$

  where A, B, C and N have their usual significance as above. In [80], a comparison to the χ^2 statistic and MI found that the normalization obtained in χ^2 helped in the comparison of terms across categories, however, with low frequency terms this comparison breaks down the normalization obtained when any cell of the contingency table is lightly populated cannot be correctly compared to χ^2 distribution with one-degree of freedom. In the case of MI, the score for each term is influenced by rare terms which have high scores than common terms.

- *Term-Strength (TS):* This measure is radically different from those described above. This method estimates term importance based on how commonly a term is likely to appear in "closely-related" documents. From a training set of documents, it derives document pairs based on the rule that the cosine value of the two document vectors is above a predefined threshold. This threshold is usually defined experimentally. A parameter which is called AREL, the average number of related documents per document is also used in threshold tuning. That is, similarity scores will be calculated for every document in a training set, and then we try all different thresholds on the similarity values of document pairs. Finally we choose the threshold which produces a reasonable value for AREL. Hence, AREL is also decided experimentally. In [80], Yang and Pedersen reported that, AREL value between 10 to 20 would give a satisfactory performance.

  TS then is calculated based on the estimated conditional probability that a term occurs in the second half of a pair of related documents given that it occurs in the first half. Let d_1 and d_2 be an arbitrary pair of documents that are distinct but related, and t be a term, then TS should be defined as follows:

$$s(t) = P(t \in d_2 | t \in d_1). \tag{2.7}$$

TS assumes that documents share words and are related to each other, and that terms in the heavily overlapping area of related documents are relatively informative. Because TS does not use any information related to term-class, it is not a task specific method. In this sense, it is similar to the DF criterion, but far different from IG, MI and χ^2 statistic.

Yang and Pedersen in [80] compared all of the feature selection methods above based on the average precisions of text classification results by applying two different well-known classification algorithms onto two commonly used datasets. They found that IG and MI were the most effective in their experiments. DF produced similar results. In this case, Yang and Pedersen suggested that one could use DF instead of IG and χ^2 because calculating DF uses less computational power of computers than other methods. DF therefore can be useful for dimensionality reduction of very large datasets. TS is proven to work for up to 50% vocabulary reduction but failed when reduction became more aggressive. In contrast, MI performed poorer than others because of its bias towards favoring rare terms, and its sensitivity to probability estimation errors.

In fact, feature selection can be separated further into two different approaches: *wrapper* and *filter*. The wrapper approach attempts to identify the best feature subset for a given classification algorithm using an iterative error-correction process that consists of a feature subset update and classification performance measure. The second approach in the feature selection process is the filter approach. Filter approach attempts to use the merits of the feature set from the data alone. We will list some of them below:

- *Mathematical Programming:* It was tested on a feature selection task in [8]. The problem of discrimination of two given sets of points in n-dimensional feature spaces using as fewer features as possible was expressed as a mathematical programming problem. By generating a separating plane in a feature space as small dimension as possible, the program attempts to select the most relevant features. Also the minimization of the average distances of misclassified points is done simultaneously. Results showed in [8] were positive. This method successfully selected fewer features.
- *Information Bottleneck (IB):* In 1999, Tishby et. al. proposed the information bottleneck algorithm [70]. In 2001, Bekkerman and his colleagues (including Tishby) tested this algorithm on the feature selection task in [3]. The paper showed that the algorithm generated a more efficient word-cluster representation of documents. They compared the classification accuracy using the

Support Vector Machines classifier with two different datasets: *Reuters-21578* and *20 NewsGroups*. They applied both the Information Bottleneck based and the bag-of-words based feature selection method on these two datasets to reduce the dimensionality of the original datasets. They found that, on reuters-21578, bag-of-words outperformed IB based approach. On the other hand, IB performed better on 20 NewsGroups datasets. This was attributed to the way texts are labeled in the two datasets.

- *Feature Selection Framework:* In [82], a framework which unified several standard feature selection methods and also facilitated the proposal of a new method that optimally combined 'positive and negative sets' was presented. The positive and negative sets correspond to positive and negative values developed by the classifiers while selecting features. The value is positive if a term is a member of a particular category, negative otherwise. Some feature selection methods, such as the χ^2 statistic, take into account both the positive and negative values of a term as evidence of being in a particular class or category. The rest, such as *Correlation Coefficient* [64], may only consider the positive values since they think negative values are not very useful. In this framework, they combined the feature selection methods to produce an optimal mixture. The paper showed that considering the negative values as the indicator that a term being not in a particular class is reasonable and feasible. Setting an optimal size for the feature set improves performance as well.
- *Categorical Descriptor Term (CDT):* In 2004, How and Narayanan proposed a feature selection method which was called CTD [30]. Very similar to the TFIDF scheme, this method is designed to choose feature sets for each category explicitly based on the inverse category frequency (ICF). ICF is the relative frequency of a term in a category or class. Rather than considering the case that a term appears in a larger number of documents or less number of documents in a class, TFICF weights terms equally throughout all documents in a specific class. Then the terms are selected based on the criteria that the terms have more discriminative power for a class if they appear in lesser number of documents. This method was tested on three datasets, 20 NewsGroups, Reuters-21578 and SITE. The Naive Bayes algorithm was applied in testing. It is reported that only when a number of features is greater than 1000, CTD can outperform the other known feature selection methods.

2. **Re-parametrization:**

   Re-parameterisation is another way of doing dimensionality reduction. The most popular re-parametrization-based method which has been widely studied and used is the so-call Latent Semantic Indexing (LSI) which reduces the term-document matrix into a set of k (k is usually smaller than the number of terms or words) factors [20]. We will discuss this method a bit further in this section as an example.

- *Latent Semantic Indexing (LSI):* LSI is used to resolve two classical questions to do with information retrieval tasks. One of the two questions is *synonymy*, that is when querying on "cars", losing the documents referencing to "automobiles"; the other is *polysemy*, that is when we querying "surfing", documents related to the Internet will come out. To deal with these questions, we represent a document by using the latent (underlying or hidden) concepts referred to by the terms, rather than by the terms directly. As you can imagine, because terms and concepts do not have a many-to-many relationship with each other, this hidden structure varies from corpus to corpus (document collections).

  LSI attempts to capture this hidden structure by using linear algebra. Generally, LSI projects the vectors which contain terms into a new, low-dimensional space obtained by Singular Value Decomposition (SVD) of the term-document matrix A. This low-dimensional space is spanned by the eigenvectors of A^TA that correspond to the few largest eigenvalues, and thus, presumably, to the few most striking correlations between terms.

  Indeed, it has been repeatedly reported that LSI outperforms, with regard to precision and recall in standard collections and query workloads, the more conventional vector-based methods, and that it does address the problems of polysemy and synonymy [22, 21, 5]. In [49], Christos and his colleagues the first time mathematically proved why LSI can outperform the conventional vector space methods.

Labeled Documents

After completing the above three steps, the documents will contain very few redundant terms and all terms are uniquely indexed and weighted accordingly. Every document can be represented as a vector of term weights with its relevant class(es) assigned. We also need to be aware that there must be at least one non-zero weight existing in every document. In general, an empty document, should otherwise be removed immediately. Untill now the labeled documents are ready to go into the text classification process.

Classification Methods

Different classification algorithms can be used to identify the inter-relationship between features in the documents and their related classes. It is also proven that different algorithms yield different results for different datasets [81]. No classification method that suits all cases of input data. In our work, we consider two classification algorithms. They are listed below and we will discuss them further in chapter 5.

- *Support Vector Machines (SVM):* This algorithm is based on Risk Minimization Principle in Statistical Learning Theory. Main working principle of SVM is to find

an optimal hyperplane based on the given examples that separates a class from the rest. The main propositions of SVM can be found in [15, 32, 47, 71].

- *BoosTexter:* The algorithm (AdaBoost) is based on the concept of Boosting in Machine Learning. BoosTexter attempts to generate accurate classification rules from the training examples provided, which are then utilized in the testing phase with new examples. BoosTexter was proposed in [62].

Training and Test Phases

The classification process consists of two steps, namely *training* and *testing*.

- *Training phase:* At this stage, a classifier or learner will try to understand the association between terms in documents and classes. The learning results are usually represented by a mapping function matrix which maps from documents to classes. This matrix is often formed as VSM as well. It can be used later to determine the inter-relationship between classes and new documents (test documents).
- *Test phase:* Different from the documents in training datasets, documents in the test datasets have no classes assigned, although they go through pre-processing, feature selection and dimensionality reduction procedures. They are represented by VSM. After the training phase, the test datasets (new documents) will be classified into the set of classes by using the learnt knowledge (i.e. mapping function) from the previous step.

Evaluation Process

Documents ranking can be done based on the highest ranked class for a single-label text classification. In the case of multi-label criteria, the documents can be ranked according to the classes predicted. In order to check the correctness of a classification process, the ranking procedure will often be applied to evaluate the prediction. We will discuss this issue further in section 2.3.

In this section, we showed an overall picture of the text classification process and related work in the current literature. In the next section, we will have a close look at a particular area of text classification – Web Classification.

2.2 Web Classification

In the earlier section, an overview and literature review of text classification was given. In this section, we will concentrate on a particular area of text classification, namely *Web*

Classification. We wish to discover what is attracting people in this area, how it can be isolated from text classification and what has been done by people in the past decade.

2.2.1 Overview

As the size of World Wide Web (WWW) rapidly expands, the need for good automated hypertext classification techniques becomes more and more apparent. The WWW network contains more than one billion Web pages and documents that are inter-connected by hyperlinks. This makes the task of locating specific information on the Web increasingly difficult. In 2000, a study conducted by Chen and Dumais showed that users often prefer navigating through directories of pre-classified content, and that providing a categorical view of retrieved documents enables them to find more relevant information in a shorter time [11]. The common use of categorical hierarchies for navigation support in Yahoo! and other major Web portals has also demonstrated the practical needs for hypertext categorization.

Hypertext classification poses new challenges due to the extra information of a hypertext document and the inter-connections among these documents. Current Web classification techniques use a variety of information to classify a target Web page, such as the content of linked documents, meta-data of a page, meta-data from related Web pages and the hyperlinks among all on-line documents. All of these provide much richer information to the classifiers of the Web page classification tasks. In contrast, traditional text classification is not concerned about any of these extra information, except the context of target document itself.

2.2.2 Current Research Efforts

Iterative Algorithms

In [9], Chakrabarti et al., they reported the results of their experiments on a patents database and a subset of *Yahoo!*. One of their approaches uses the text components of neighboring documents as the local text of the target Web pages. neighboring documents here means the Web pages that are connected from/to the target Web page through hyperlinks. The researchers combined both text content of a target Web page and the text content from all of its neighboring documents to form a new feature set for the target Web page. The classification results based on this approach were worse than classifying the same documents collection with none of the neighboring documents' content involved. The researchers from [9] also proposed another approach to improve Web classification. They developed an iterative algorithm which labeled test documents using the labels of the neighbor documents. The labels assigned to the classifier were predicted by a previous iteration or guessed from the text content. Using the iterative labeling process, the researchers observed an improvement on the classification accuracy. In [9], they used

a Patents database as the data resource, because this kind of databases usually have their own properties that are different from a Web page collection, it is still not very clear if this approach can be generalized to a real WWW network.

In 2000, Oh et al. from Korea proposed another algorithm in [48] for exploiting hyperlink information among Web pages. Their work was developed based on the algorithm proposed in [9]. In this improved algorithm, an adjacency matrix was formed to construct the connections among all Web pages in the corpus. The connection here does not only mean the semantic or referential link between a pair of Web pages, but also the two Web pages must be homogeneous. As described in the paper, the homogeneity was calculated using the similarity between the two Web pages. This process reduces the likelihood of getting more noise from neighbor documents. Then, based on the term frequencies of the corresponding terms in the neighboring documents, they re-calculate the weight of every term in a target Web page. This ensures the influence of the neighboring documents on the target documents. They also assigned a probable class to all neighbor documents that had not been classified by guessing, a Bayesian classifier or pre-defined classes. A confidence factor will be assigned to every neighbor document. The confidence factor is to show how reliable a class assignment is. For those documents that have the class which is assigned by a classifier or pre-defined classes will be assigned with the maximum confidence factor. A partial confidence factor will be assigned otherwise. After all the above processes, the data is ready to be classified. In contrast to the work in [9], they observed a slight improvement on both the classification results and efficiency of the classifier. These two algorithms, however, are not perfectly comparable to each other, because they were using different datasets.

Hybrid Algorithms

The results shown above indicate that simply assuming that linked documents are on the same topic, and then incorporating the features of linked documents, is not generally effective.

In 1998, Slattery and Craven proposed a hybrid classification algorithm FOIL-PILFS for hypertext classification [17, 66]. FOIL-PILFS here stands for FOIL with Predicate Invention for Large Feature Spaces. This method combines two different types of classification algorithms, where include a statistical text-learning algorithm and a relational rule learner that forms a hybrid-algorithm. The hybrid-algorithm can use both the traditional features (the same as standard text classification) and the extra features that only hypertext documents would provide. The extra features include anchor text, text from neighboring documents, capitalized words and alphanumeric words. FOIL-PILFS is well suited for the requirement of hypertext classification because it takes into account the relationship between a target document and all its neighboring documents. This strategy can be a good complement to the standard text classification methods on hypertext

classification tasks. As an example, in [66] they mixed Naive Bayes (a well known statistical learning method) [38, 42] and FOIL (a commonly used relational learning method) [54, 55] together. In this case, FOIL will be able to consider the weights of the words or features, whereas it traditionally only consider binary cases (presence or absence) of specific keywords. Furthermore, because FOIL-PILFS can learn the statistical properties of a given set of pages and hyperlinks, it is capable of selecting features based on the corresponding statistical properties of the classification task at hand. As the authors reported, this relational/statistical text learning approach successfully improved the classification accuracy, in contrast to the baseline produced by using only FOIL. They also believe that this approach can be generalized to be used not only in hypertext classification tasks, but also in other domains that include both relational structure and potentially large feature space.

Popescul et al. in [51] proposed a similar hybrid algorithm. This algorithm integrates the structure navigation from IPL (Inductive Logic Programming) with regression modeling. They also used first-order rules at each step of the IPL's relational structure search to generate features for the potential inclusion in the regression model. As an example, in [51], they used FOIL as its relational component and used LR (Logistic Regression) as its statistical component. They also set up twelve datasets which they obtained from ResearchIndex (CiteSeer) [6, 36] to exploit the citation structure among these scientific documents. In this work, the researchers did not only consider the case of the so-called "flat" features (the immediately incoming and outgoing citations), but also the relational representation which describes the relationship between the target document and another document through the immediately linked documents. They also took into account the Bibliometric Interpretation [25, 68]. Especially, two concepts from it are concerned carefully. (1) *bibliographic coupling* which is the degree of similarity between two documents based on documents cited in common. (2) *co-citation* stands if two documents are cited together by a third document. In this case, we say that the two documents are co-cited with each other. The researchers found that combining both word-based and citation-based features together would improve the classification accuracy, but it only happens in some regimes. They also found that using propositionalized citation-based relational structure features determined by FOIL along with word counts in logistic regression often significantly improve the classification precision (but recall may suffer). This algorithm differs from the algorithm proposed in [17, 66] in the "direction" the combination takes place. In [17, 66], the algorithm uses a word-based statistical method to generate new predicates for each loop of a relational structure search which is the core modeling component. In [51], however, the hybrid algorithm has a statistic method as a modeling component for which the features are supplied by an IPL-based relational structure search.

Hypertext Regularities

Because of the uncertainty of usefulness of these extra hyper-data that exists in all the HTML-like documents, another task in hypertext classification is needed to identify some hypertext hypotheses and to see how they would affect the accuracy of the hypertext classification. For this reason, Yang et al. in [81, 26] proposed five regularities for the general hypertext classification procedure. They also gave an in-depth investigation of the validity of these regularities across several datasets by using a range of classifiers. The five hypertext regularities are listed below.

- **No Regularity:** If documents are linked randomly or the content from linked documents is independent of the target document's class, then they do not expect any possible effect from these hyperlinks or their related content. In this case, the hypertext classification task becomes the standard text classification. Then a standard text classifier can be applied without any modification. They believe that the inter-connection between Web pages is rarely arbitrary. This regularity can then be used to produce a baseline classifier which compares classifiers that are designed for using other hypertext regularities. Hopefully, any classifier which takes hypertext regularities into account would improve the performance of a standard text classifier.

- **"Encyclopedia" Regularity:** If the target document shares the same class as the majority of its linked documents, they expect that the linked documents would provide some useful information to the classifiers. As shown above, the ETRI-Kyemong encyclopedia corpus which was used in [48] exhibits this property well. This is because most of the linked documents share topically similar content with the target document. With this kind of regularity, augmenting the target document's representation with the content from the linked documents would improve the performance of the text classifiers, since more topic related words are added to the target document. This is the reason why the experiments in [48] produced a better classification. In [9], the classification performance on a patents database suffered a lot from applying this approach because this structure does not exist for patents.

- **"Co-Referencing" Regularity:** This is similar to the "encyclopedia" regularity shown above. In this case, however, the target document is not linking to the documents that are in the same class, instead they are topically related to each other. For example, a news article about a particular event might link to some other articles that report the background of this event. This approach may have some predictive power too, but they must be treated separately. In [9], they also implemented this approach. They treated the words from the linked documents as they were from a separated vocabulary by adding tags or prefix to the words. On the patents database, however, they found that the performance of the classification

suffered again. This again is indicating that the patents database does not have this structure.

A variant of this regularity which relaxes the requirement that all of the neighboring documents have to share the same topic. In this sense, only some of the linked documents may have the same topic as the target document. If this is the case for some datasets, it needs to identify those linked documents which share the same or similar topics with each other. This involves the process of clustering or a text-based similarity calculation which is applied among all the documents that are linked to the target document. In contrast to the previous regularity which uses various "bag-of-words" representations from all neighboring documents and standard text classification algorithms, this approach uses only the "bag-of-words" from the documents that are similar to the target document and the classifier used should be modified accordingly. In 1998, Craven et al. [18] applied FOIL to the WebKB University corpus and found that it did improve classification performance. This indicates that the WebKB corpus supports this regularity. Moreover, in 2000, Oh et al. in [48] used this regularity in their experiments and indeed improved the classification results by applying the cosine-similarity based filter to select the linked documents.

- **Preclassified Regularity:** Rather than considering the topic of neighboring documents in encyclopedia regularity and co-referencing regularity, the hyperlink structure itself could also provide useful information to the classification task. In some cases, it may be more efficient while it is keeping the classification accuracy as good as the other regularities. In this approach, a page or a small set of pages may have already been well classified. There could be at least one page which contains a list of hyperlinks that connect to other pages that are in the same class. A very good example of this regularity is the *Yahoo!* topic hierarchy. If we could find these "hub" pages in the datasets, then classification performance could be improved. Moreover, if this regularity stands, we have no need to look at the content of any document. Thus, we only need to locate those "hub" pages and classify those documents that link to the "hub" pages accordingly. In [33], the successful use of a SVM kernel function for hyperlinks by Joachims et al. Illustrated one way of exploiting this regularity.

- **Meta-Data Regularity:** Because we are dealing with hypertext, there generally are meta-data available from external sources. This kind of data can be exploited in the form of additional features. To obtain these features, one can use information extraction techniques. Furthermore, cooperating with the other regularities discussed above, these extracted features can then be used in a similar fashion by using the identity of the related documents and the text of related documents in various ways. The availability and quality of this kind of information depends on the datasets in the classification tasks because of the uncertainty of such information.

In 2000, Cohen [12] described some experiments where he automatically located and extracted such features for several (non-hypertext) classification tasks.

The internal "meta-data" of a Web page can also be considered. This type of "meta-data" exists within Web pages, such as ALT, META and TITLE tags. Although this type of information contained by every HTML document is technically not meta-data at all, because it is internal rather than external to the page. The information existing within these tags could aid in the classification procedure by treating them differently from other parts of the Web pages. A typical example is that, in [26, 69, 17], this approach was used and the classification performance was improved.

Three well-known supervised classification algorithms were used in [81], namely Naive Bayes [38, 42], K-nearest neighbor (KNN) [19, 78, 79, 77] and FOIL [53]. Based on the five explicitly defined regularities, eight alternative representations and three multiple established hypertext datasets from different domains (Hoover-28, Hoover-255 and WebKB Univ-6), they applied the above three classifiers and found that the usefulness of the regularities varied, depending on both the datasets and the classifiers being used.

Probabilistic Model

In 2001, Cohn and Hofmann [14] proposed their probabilistic model based on their earlier work. This algorithm takes into account both the content and links of linked documents. Linked documents here mean both hypertext documents that are connected by hyperlinks and scientific papers that are referentially linked by the citations existing in the papers. Essentially, this algorithm is based on the work they did on Probabilistic Latent Semantic Analysis (PLSA) [20, 29] and Probabilistic Hypertext-Induced Topic Selection (PHITS) [13, 34].

- PLSA [29] is a statistical variant of Latent Semantic Analysis (LSA) [20] which builds a factored multinomial model based on the assumption of an underlying document generation process. The starting point of (P)LSA is the term-document matrix N of word counts, where N_{ij} denotes how often a term (single word or phrase) t_i occurs in document d_j. The difference between PLSA and LSA is that, in LSA, N is decomposed by a SVD and the factors are identified with the left or right principal eigenvectors. In contrast, PLSA performs a probabilistic decomposition which is closely related to the non-negative matrix decomposition presented in [37]. Each factor is identified with a state z_k of a latent variable with associated relative frequency estimates $P(t_i|z_k)$ for each term in the corpus. A document d_j is represented as a convex combination of factors mixing with weights $P(z_k|d_j)$, i.e., the predictive probabilisties for terms in a particular document are constrained to

be of the functional form $P(t_i|d_j) = \sum_k P(t_i|z_k)P(z_k|d_j)$, with non-negative probabilisties and two sets of normalization constraints $\sum_i P(t_i|z_k) = 1$ for all k and $\sum_k P(z_k|d_j) = 1$ for all j.

- PHITS [13] performs a probabilistic factoring of document citations used for bibliometric analysis. Bibliometric analysis attempts to find topics in a document collection, as well as influential authors and papers on those topics based on citation frequency patterns. This algorithm has been proved to be useful not only for the references of traditionally printed documents, but also in analyzing the hyperlink structure of the WWW [34]. Traditionally, bibliometric analysis starts with a document-citation matrix A. $A_{ij} = 1$ denotes document d_i is cited by document d_j, or equivalently, d_j has a hyperlink to document d_i. A "community" of a roughly similar citation pattern is generated by calculating the Principal eigenvectors of AA'. The coefficient of a document in one of these eigenvectors represents how likely the document would be cited within this community. Reversely, by calculating the document's coefficient in the Principal eigenvectors of $A'A$, we could know how many authoritative documents are cited by this document. In contrast, PHITS which is a probabilistic model replaces the eigenvector analysis, yielding a model which has clear statistical interpretations. PHITS is mathematically identical to PLSA but with only one distinction: instead of modeling the terms in a document, PHITS models "in-links", which are the citations to a document. It substitutes a citation-source probability estimate $P(c_l|z_k)$ for the PLSA's term probability estimate; and the principal factors are interpreted by indicating the principal citation communities. For a given factor/topic z_k, the probability a document d_j is cited $P(d_j|z_k)$ is interpreted as the document's authority with respect to the topic.

The mixing these two algorithms we get a simultaneous decomposition of the contingency tables associated with word occurrences and citation/links into "topic" factors. Both PLSA and PHITS are based on a similar decomposition, here they define the following joint model for predicting citations or links and terms in documents:

$$P(t_i|d_j) = \sum_k P(t_i|z_k)P(z_k|d_j), \quad P(c_l|d_j) = \sum_k P(c_l|z_k)P(z_k|d_j).$$

The decompositions shown above share the same document-specific mixing proportions $P(z_k|d_j)$. This combines the conditional probabilities for terms and citations: each "topic" has some probablity $P(c_l|z_k)$ of linking to document d_l as well as some probability $P(t_i|z_k)$ of having term t_i. The advantage of this joint modeling approach is that it integrates content- and link information in a principled manner.

Cohn and Hofmann in the paper [14] used WebKB and Cora [43] as their datasets. They set up three experiments for possible different applications of a joint model, namely Classification and Intelligent Web Crawling with Reference Flow. All the experiments showed that the joint modeling worked much better than using the two parts individually.

Modeling hyperlink distributions

In 2003, Qing and Lise [41, 40] proposed a new statistical framework for modeling link distributions. They make use of link information between training and test sets by combining both ordinary features (words) and link distributions together. Link distributions take care of the categories of linked documents rather than the content of the linked documents. It can be modeled in three different ways, namely count-link model, binary-link model and mode-link model. The count-link model assigns different values to every link distribution based on the frequencies of different categories of the neighboring documents. The binary-link model assigns 1 to the link distributions as soon as the corresponding category appears in the neighboring documents, 0 otherwise. The mode-link only counts the category which has the highest frequency among the neighboring documents. For each of these three different models, they take into account in-link, out-link and co-link to obtain link distribution values. Co-link here means the co-citation link, which a Web page X and Y both connect to Web page Z, so that X co-cited with Y. After modeling the data, they proposed an iterative classification algorithm (ICA) based on the previous work done by Chakrabarti et al. [9] and Neville and Jensen [9]. This general approach has been studied in numerous fields, including relaxation-labeling in computer vision [31], inference in Markov random fields [10] and loopy belief propagation in Bayesian networks [46]. The challenging aspect of the Qing and Lise's approach is that the attempt to make use of the non-regular structure of the inference. Their ICA has two stages: bootstrap and iteration. At the beginning, all documents are unlabeled, thus the link statistics are unknown. At the bootstrap stage, only those ordinary features, such as words, will be taken into account to classify and assign an initial category to each document. During the iteration stage, the neighboring documents' category information will be considered, used and updated based on the previous round. This procedure can be done in three steps: 1. compute the link statistics based on the current assignments to linked documents; 2. compute the posterior probability for the category of this document; 3. the category with the largest posterior probability is chosen as a new category for the current document. The algorithm terminates when there are no longer any updates to the categories or a maximum number of steps has been reached. The experiments were done on two standard datasets Cora [43] and WebKB [16] and a dataset that they constructed from CiteSeer entries [27]. The results reported in the papers show that the proposed algorithm worked and slightly increased the classification results.

We should be aware that all experiments that we have introduced above were done on single labeled datasets. In the current literature, there is no such algorithm which make use of the categories of linked documents and that can be applied to multi-label datasets. In the next section, we will introduce some well-known and widely-used evaluation measures.

2.3 Evaluation Measures

In text classification, after classifiers are trained on a set of training examples, we will apply the trained classifiers on a distinct set of test documents. After the classification predictions come out, a series of performance measures can be employed. Namely, recall, precision, fallout, accuracy and error.

For each category or class, these measures can be defined as follows:

	Expert Yes	Expert No
Prediction Yes	a (Assigned correctly)	b (Assigned wrongly)
Prediction No	c (Rejected wrongly)	d (Rejected correctly)

Table 2.2: Contingency table for evaluation measure.

$$recall = \frac{a}{a+c} \tag{2.8}$$

$$precision = \frac{a}{a+b} \tag{2.9}$$

$$fallout = \frac{b}{b+d} \tag{2.10}$$

$$accuracy = \frac{a+d}{a+b+c+d} \tag{2.11}$$

$$error = \frac{b+c}{a+b+c+d} \tag{2.12}$$

Here the letter 'a', 'b', 'c' and 'd' are from table 2.3. a is the number of documents correctly assigned to the category; b is the number of documents incorrectly assigned to the category; c is the number of documents incorrectly rejected from category; and d is the number of documents correctly rejected from category. Moreover, $a+c$ represents the total number of documents that belong to the category; $a+b$ indicates the total number of documents assigned to the category; $b+d$ represents the number of documents that should not be in the category; and $a+b+c+d$ is the total number of documents evaluated for the category.

For evaluating the performance averaged across categories or classes, there are two main measures: (1) *macro-averaging* method which computes the global performance scores by averaging per class scores and (2) *micro-averaging* method that computes the global performance scores directly.

In applications, however, these measures produce some difficulties. This is related to the fact that recall and precision behave differently. As recall increases, precision decreases. In such cases, therefore, the intersection point is considered in the literature

as a reliable measure. This difficulty is avoided by using a different set of measures that were introduced in [62].

There are three well-known measuring methods we can find from the literature. They are *One-Error*, *Coverage* and *Average Precision*. The first one, one-error, is a simple generalization of classification error for multi-label problems. The one-error measure is also directly related to the training error [61]. The other two evaluation measures are based on measures that are used in information retrieval and for evaluating the performance of the various classification algorithms in terms of their label rankings.

As noted earlier, we assume that a multi-label system induces an ordering of the possible labels for a given instance. That is, the output of the learning system is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ which ranks labels according to $f(x, \dot{})$ so that label l_1 is considered to be ranked higher than l_2 if $f(x, l_1) > f(x, l_2)$.

We will find it convenient to refer to the rank of a given label l, for instance x under f which we denote by $rank_f(x, l)$. That is, formally, $rank_f(x, \dot{})$ is a one-to-one mapping onto 1,. . . ,k such that if $f(x, l_1) > f(x, l_2)$ then $rank_f(x, l_1) < rank_f(x, l_2)$.

1. **One-Error:** This measure evaluates how many times the top-ranked label was not in the set of possible labels. Thus, if the goal of a multi-class system is to assign a single label to a document, the one-error measures how many times the predicted label does not match the expert opinion. We call this measure one-error hypothesis H since it measures the probability of not even getting one of the labels correct. We denote the one-error of a hypothesis f by $one - err(f)$. We can define a classifier $H : \mathcal{X} \rightarrow \mathcal{Y}$ that assigns a single label for a document x by setting $H(x) = arg \max_{l \in \mathcal{Y}} f(x, y)$. Then, for a set of labeled documents $S = \langle (x_1, Y_1), \ldots, (x_m, Y_m) \rangle$, the one-error is

$$one - err_S(H) = \frac{1}{m} \sum_{i=1}^{m} [H(x_I) \notin Y_i] \tag{2.13}$$

   Note that, if we use single labeled datasets, one-error is equivalent to the ordinary error measure.

2. **Coverage:** While the one-error evaluates the performance of a system for the top-ranked label, the goal of the coverage measure is to assess the performance of a system for all the possible labels of the documents. That is, coverage measures how far we need, on average, to go down the list of labels in order to cover all the possible labels assigned to a document [62]. Coverage is loosely related to precision at the level of perfect recall. Formally, we define the coverage of f with respect to $S = \langle (x_1, Y_1), \ldots, (x_m, Y_m) \rangle$ to be

$$coverage_S(H) = \frac{1}{m}\sum_{i=1}^{m}\max_{l\in Y_i} rank_f(x_i, l) - 1 \tag{2.14}$$

For single-label classification problems, coverage is the average rank of the correct label, otherwise it is zero if the system does not produce a classification error.

3. **Average Precision:** The above measures are not complete for multi-label classification problems: We can achieve good (low) coverage but suffer high one-error rates, and vice versa. In order to assess the label ranking of a multiclass system as a whole, we use the non-interpolated average precision, which is a performance measure frequently used for evaluation of information retrieval (IR) systems [59]. Note, however, that non-interpolated average precision is typically used in IR systems to evaluate the document ranking performance for query retrieval. In our work, we use average precision for evaluating the effectiveness of the label rankings. Formally, we define average-precision for a ranking H with respect to a training set S, denoted $avgprec()$ for short, to be

$$avgprec_S(H) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{|Y_i|}\sum_{l\in Y_i}\frac{|\{l' \in Y_i | rank_f(x_i, l') \leq rank_f(x_i, l)\}|}{rank_f(x, l)} \tag{2.15}$$

In words, this measure evaluates the average fraction of labels ranked above a particular label $l \in Y_i$ which actually are in Y_i. Note that $avgprec_S(f) = 1$ for a system f which ranks the labels perfectly for all documents so that there is no document x_i for which a label not in Y_i is ranked higher than a label in Y_i.

As we mentioned, because one-error and coverage are not complete for multi-label cases, in this work we will use only Average Precision (AP) as our measuring method.

In this chapter, we presented the whole picture for both Text Classification and Web Classification domains and their development. In the next chapter, we will talk about informativeness of features, and how we can alter the formula to affect the final classification results.

Chapter 3

Dimensionality Reduction

3.1 Dimensionality Reduction

In text classification, the high dimensionality of the feature space (i.e. the large number of terms) is usually a problem. Therefore, Dimensionality Reduction is usually applied to datasets before classification. This is, sometimes, very critical to the final classification results in terms of both accuracy and computational efficiency for some classifiers.

The aim of dimensionality reduction is to reduce the size of the term vector space from T to T', where $|T| \gg |T'|$, and the set T' is called the reduced feature or term set.

Recently, many machine learning and statistical methods are being or have been developed to solve text classification problems. The major problem for most of those methods is the high-dimensionality of conventional document feature space. Typically, words and phrases are being used as features in text classification. Consider documents in general, even a medium-size document will produce a huge feature space with hundreds of thousands of terms in it. Most of the recently developed text classification algorithms are not designed for dealing with such a large feature space. No doubt, this will exceed the existing computational capability of modern computers. Moreover, if we generate the feature space naively by putting all words in, the words that appear most frequently (usually the general words) could mislead the classifier. It is therefore highly desirable to reduce the size of the document feature space in some way.

Here, a simple feature exclusion technique, Document Frequency Thresholding, could be used. Basically, it gets rid of those words that are within a pre-defined range of frequency. What this technique does is based on a simple assumption that neither rare words nor common words are meaningful to the classification discriminators. Using this technique will certainly reduce the size of the document feature space. It could, however, disregard some important words as well.

From the literature, we find there are many dimensionality reduction methods that are being used for different purposes and datasets, such as *Correlation Coefficient*, *Information Gain* and *Mutual Information*. Among them, Information Gain (IG) is one of the most popular features selection method that has been applied to text classification.

3.2 Information Gain

IG takes into account the term goodness criterion, which is equivalent to the number of bits of information gained for a category prediction by knowing the presence or absence of a given feature in a document. In other words, it measures the rank and selects the most category-predictive or most informative words.

IG is based on the probability of document containing a given term and belonging to a given category. Let $\{c_i\}_{i=1}^{|C|}$ denote the set of categories in the target space. The information gain of feature t is defined as (see also [80]):

$$\begin{aligned} IG(t) &= -\sum_{i=1}^{|C|} P(C_i)\log P(C_i) \\ &\quad +P(t)\sum_{i=1}^{|C|} P(C_i|t)\log P(C_i|t) \\ &\quad +P(\bar{t})\sum_{i=1}^{|C|} P(C_i|\bar{t})\log P(C_i|\bar{t}) \end{aligned} \tag{3.1}$$

where,

$$P(t) = \frac{\text{number of documents contain } t}{\text{total number of documents}}; \tag{3.2}$$

$$P(\bar{t}) = 1 - P(t); \tag{3.3}$$

$$P(C_i) = \frac{\text{number of documents in } C_i}{\text{total number of documents}}; \tag{3.4}$$

$$P(C_i|t) = \frac{\text{number of documents contain } t \text{ and in } C_i}{\text{number of documents contain } t}; \tag{3.5}$$

$$P(C_i|\bar{t}) = \frac{\text{number of documents NOT contain } t \text{ and in } C_i}{\text{number of document NOT contain } t}. \tag{3.6}$$

The formula appeared in Hayri's paper [65] is also listed below:

$$IG(t) = \sum_{i=1}^{|C|} P(t, C_i) \log \frac{P(t, C_i)}{P(t)P(Ci)} + \sum_{i=1}^{|C|} P(\bar{t}, C_i) \log \frac{P(\bar{t}, C_i)}{P(\bar{t})P(Ci)}. \tag{3.7}$$

Where, $P(t)$, $P(\bar{t})$ and $P(C_i)$ are defined the same as the formula 3.2, 3.3, 3.4 above; and

$$P(t, C_i) = P(t|C_i)P(C_i) = P(C_i|t)P(t) \tag{3.8}$$
$$P(\bar{t}, C_i) = P(\bar{t}|C_i)P(C_i) = P(C_i|\bar{t})P(\bar{t}) \tag{3.9}$$

There is also another interpretation of for $P(t, C_i)$ is as:

$$P(t, C_i) = P(t)P(C_i). \tag{3.10}$$

The above formula is for the case that t and C_i are completely independent. However, this is contradictory to the concept of text classification which assumes that the event t and C_i must be related. This interpretation is therefore not considered.

3.3 Equivalence of the two IG Formulae

Although formulae (3.1) and (3.7) look very different, they describe the same object information gain. In this section, therefore, we show that they are equivalent. We then show (see formula 3.7):

$$X = \sum_{i=1}^{|C|} P(t, C_i) \log \frac{P(t, C_i)}{P(t)P(Ci)}; \tag{3.11}$$
$$Y = \sum_{i=1}^{|C|} P(\bar{t}, C_i) \log \frac{P(\bar{t}, C_i)}{P(\bar{t})P(Ci)}. \tag{3.12}$$

We divide formula (3.1) into three parts as shown below.

$$U = -\sum_{i=1}^{|C|} P(C_i) \log P(C_i) \tag{3.13}$$

$$V = P(t) \sum_{i=1}^{|C|} P(C_i|t) \log P(C_i|t) \tag{3.14}$$

$$W = P(\bar{t}) \sum_{i=1}^{|C|} P(C_i|\bar{t}) \log P(C_i|\bar{t}). \tag{3.15}$$

First we simplify and transform the equation X as follows.

$$X = \sum_{i=1}^{|C|} P(t)P(C_i|t) \log \frac{P(t)P(C_i|t)}{P(t)P(Ci)} \tag{3.16}$$

$$= \sum_{i=1}^{|C|} P(t)P(C_i|t) \log \frac{P(C_i|t)}{P(Ci)} \tag{3.17}$$

$$= \sum_{i=1}^{|C|} P(t)P(C_i|t) \left[\log P(C_i|t) - \log P(Ci)\right] \tag{3.18}$$

$$= \sum_{i=1}^{|C|} P(t)P(C_i|t) \log P(C_i|t) - \sum_{i=1}^{|C|} P(t)P(C_i|t) \log P(Ci). \tag{3.19}$$

By going through very similar procedures, we obtain

$$Y = \sum_{i=1}^{|C|} P(\bar{t})P(C_i|\bar{t}) \log P(C_i|\bar{t}) - \sum_{i=1}^{|C|} P(\bar{t})P(C_i|\bar{t}) \log P(Ci). \tag{3.20}$$

We then add the two parts together and get:

$$
\begin{aligned}
X + Y &= \sum_{i=1}^{|C|} P(t)P(C_i|t)\log P(C_i|t) && (3.21)\\
&\quad - \sum_{i=1}^{|C|} P(t)P(C_i|t)\log P(Ci) && (3.22)\\
&\quad + \sum_{i=1}^{|C|} P(\bar{t})P(C_i|\bar{t})\log P(C_i|\bar{t}) && (3.23)\\
&\quad - \sum_{i=1}^{|C|} P(\bar{t})P(C_i|\bar{t})\log P(Ci). && (3.24)
\end{aligned}
$$

It is clear that (3.21) and (3.23) are equal to V and W respectively. For the rest, we only need to prove that the sum of (3.22) and (3.24) are equal to U. We have

$$
\begin{aligned}
(3.22) + (3.24) &= -\sum_{i=1}^{|C|} P(t)P(C_i|t)\log P(Ci) - \sum_{i=1}^{|C|} P(\bar{t})P(C_i|\bar{t})\log P(Ci) && (3.25)\\
&= -\sum_{i=1}^{|C|} \log P(C_i)\left[P(t)P(C_i|t) + P(\bar{t})P(C_i|\bar{t})\right] && (3.26)\\
&= -\sum_{i=1}^{|C|} P(C_i)\log P(C_i). && (3.27)
\end{aligned}
$$

This is the same as U in (3.1). Now, the parts in the two formulae are matched to each other. Eventually, we proved that these two IG formulae are the same.

3.4 Modification of IG formulae

In the previous section (Section 3.2), we introduced the formal interpretations of IG formulae, which needs to count the number of documents for calculating different probabilities. We call this interpretation "document based" and denote it with IG_{doc} later in the thesis. "Document based" means that we consider if a document belongs to a class or classes, rather than if a feature appears in a class or classes. Next, we are going to introduce our interpretations of the IG formulae, which are term based. We explain the term based interpretations.

In this project, we further divide the term based concept into two parts. In the first part, we use the frequency as the value of a feature in a document.

$$\begin{aligned} IG_{tf}(t) &= -\sum_{i=1}^{|C|} P(C_i)\log P(C_i) \\ &\quad +P(t)\sum_{i=1}^{|C|} P(C_i|t)\log P(C_i|t) \\ &\quad +P(\bar{t})\sum_{i=1}^{|C|} P(C_i|\bar{t})\log P(C_i|\bar{t}) \end{aligned} \tag{3.28}$$

where,

$$P(t) = \frac{\text{sum of frequencies of term } t}{\text{sum of frequencies of all terms}} \tag{3.29}$$

$$P(\bar{t}) = 1 - P(t) \tag{3.30}$$

$$P(C_i) = \frac{\text{sum of frequencies of terms in } C_i}{\text{sum of frequencies of all terms}} \tag{3.31}$$

$$P(C_i|t) = \frac{\text{sum of frequencies of term } t \text{ in } C_i}{\text{sum of frequencies of all term } t} \tag{3.32}$$

$$P(C_i|\bar{t}) = \frac{\text{sum of frequencies of term NOT } t \text{ in } C_i}{\text{sum of frequencies of all terms } \bar{t}} \tag{3.33}$$

In the second part, we use TFIDF values as the value of a feature in a document, so that we obtain:

$$\begin{aligned} IG_{tfidf}(t) &= -\sum_{i=1}^{|C|} P(C_i)\log P(C_i) \\ &\quad +P(t)\sum_{i=1}^{|C|} P(C_i|t)\log P(C_i|t) \\ &\quad +P(\bar{t})\sum_{i=1}^{|C|} P(C_i|\bar{t})\log P(C_i|\bar{t}) \end{aligned} \tag{3.34}$$

where,

$$P(t) = \frac{\text{sum of tfidf of term } t}{\text{sum of tfidf of all terms}} \tag{3.35}$$

$$P(\bar{t}) = 1 - P(t) \tag{3.36}$$

$$P(C_i) = \frac{\text{sum of tfidf of terms in } C_i}{\text{sum of tfidf of all terms}} \tag{3.37}$$

$$P(C_i|t) = \frac{\text{sum of tfidf of term t in } C_i}{\text{sum of tfidf of all term } t} \tag{3.38}$$

$$P(C_i|\bar{t}) = \frac{\text{sum of tfidf of term NOT } t \text{ in } C_i}{\text{sum of tfidf of all terms } \bar{t}} \tag{3.39}$$

The reason we propose these interpretations is: first, different IG values will give us different rankings for the same features, so that we could then generate more datasets for experiments; second, from the statistic point of view, the terms are event as well, so that they are eligible to be taken into account independently, rather than based on documents; we are calculating the IGs of terms, so taking terms into account directly may produce better results. We will present comparisons of text classification results of the datasets whose features are ranked based on the three different IG interpretations in section 6.1.

In this section, we explained the concept and main aim of IG and provided two different interpretations. In chapter 6, we will presente how the different interpretations of IG formula could affect the final classification results. We will also show that the term frequencies based IG calculation would give us the best fit ranks of features for the SVM algorithm.

Chapter 4

Preparation for Text Classification Processes

4.1 Data Collection

The commonly used data sets available cannot satisfy the requirements of our project. They are not both multi-class multi-labeled and having link information included. For example, WebKB, the well-known Web collection which contains Web pages collected from four universities is single labeled database. For this reason, we need to generate our own data collection from scratch. Therefore, we developed, a crawler, which is able to retrieve all kinds of common on-line documents in WWW, such as HTML, PDF (Portable Document Format) and Microsoft word documents. The crawler was run for two weeks. The retrieval range was limited to within Ballarat University's intranet for efficiency of both retrieval speed and storage usage. We will be trying to use the link information that exists in HTML-like documents. Other types of documents that do not have such rich link information will not be retrieved. Thus, only HTML documents will be retrieved and stored.

All the retrieved data was stored in a database system, which contains 4 related tables, namely "LINK", "URL", "URLWORDS" and "WORD".

WORD table contains all the unique words in the documents retrieved. Because the words in this table are not repeated, a unique integer number is assigned to each word. All words, therefore, have their own unique identifier which will distinguish them from each other. If we obtain a sentence "Hello, hello, the world" and store it in table WORD. We will then have the table contents listed in Table 4.1. From the table, we notice that: 1) the process is not case sensitive, and all capital letters have been converted to their lower case; 2) repeated words are only stored once, for example, the second "hello" in the sentence has been ignored automatically.

Identifier	Word
1	hello
2	the
3	world

Table 4.1: An example of table WORD.

URL stands for "Uniform Resource Locator" (previously "Universal Resource Locator"). It represents a particular resource on a network, uniquely. In this project, the resources we are discussing refers mainly to those on-line documents in different formats. The URL table, therefore, contains the information about all the documents we retrieved. The information includes the URL itself, the title, meta-title, keywords and meta-contents of a corresponding on-line document. We are also able to assign a unique integer number to the URL so the program will not retrieve the documents that have already been retrieved. This means that there are no duplicated URL in the table. There is an example shown in Table 4.2. We should be aware that the meta-title and meta-contents may be blank. This is because some web designers do not use these kind of meta-data in their Web pages; or the document is not a HTML-like document, for example, a PDF document which does not have this type of information embedded.

Id	URL	Title	Meta-Title	Kword	Meta-Cont
1	www.ballarat.edu.au	Ballarat University	N/A	Ballarat, University	N/A
2	www...au/internal	internal website	N/A	internal website, Ballarat, University	N/A

Table 4.2: An example of table URL.

Table URLWORDS contains URLs and the words contained in the corresponding documents. All the URLs and words in this table are replaced by their own identifiers. Table 4.3 shows that, document 1 contains words 1 and 2; and document 3 has word 10. The URLs may be repeated because one document will usually contain more than one word. One row shows a relationship between the document and one of its words, therefore, we need many rows to represent all the relationships for each of the unique words in the document. The frequency attribute shows how many times a word appears in a particular document. For example, word number 10 appears in document three 37 times.

LINK table shows what URLs are retrieved from HTML-like documents. It contains documents' URLs and those links that exist in the documents. This table builds the relationship between a HTML-like document and other on-line documents. All URLs are represented by their own identifiers. Table 4.4 shows that, document 1 connects to document 2; and document 2 links to document 5 and 25. For example, the first row

URL	Word	Frequency
1	1	20
1	2	5
3	10	37

Table 4.3: An example of table URLWORDS.

tells us that the homepage of Ballarat university's (www.ballarat.edu.au) links to the university's internal page (www.ballarat.edu.au/internal).

URL	Link
1	2
2	5
2	25

Table 4.4: An example of table LINK.

4.2 Pre-processing

After we retrieved the data and stored it in the database system which we described in the previous section, we need to pre-process the data to allow the application of some classifiers to it.

4.2.1 Documents Selection and Manual classification

Because of time limitation, although we obtained more than five hundred thousand Web pages from Ballarat University's intranet, we only chose around 2000 documents to form our dataset. To facilitate this selection, a program was written to randomly choose 2000 documents. Each one of these documents was required to contain at least 50 words.

Based on the contents of these selected documents, we created a set of classes or categories. There are 11 of them in total, namely "Services", "Resources", "Help", "Policies", "Plans", "Announcements", "Research", "Teaching & Learning", "Personnel", "Finance" and "Advertisements". They are also numbered from 1 to 11 with unique identifiers as shown in Table 4.5.

After that, we read through each of the 2000 selected documents to manually assign them into one or more categories within the 11 classes based on our knowledge. This is called manual classification. It assigns a so-called Expert Opinion to the documents. In this procedure, we set 3 as the maximal number of categories that a document would belong to. This is for simplicity. As a result, a document would belong to one or at most three classes in our final datasets.

Identifier	Abbreviation	Category
1	S	Services
2	RS	Resources
3	H	Help
4	PO	Policies
5	PL	Plans
6	A	Announcement
7	RH	Research
8	TL	Teaching & Learning
9	PE	Personnel
10	F	Finance
11	AD	Advertisements

Table 4.5: Table of all categories.

4.2.2 Stop-words and Word-stemming

As we mentioned in "Pre-processing of Data" section in chapter 2, we need to apply stop-words removal and stemming processes on to our new Web collection to refine the data we retrieved. A program was written for the purpose of removing all the stop-words and stemming the rest of them. We used this program on the 2000-document collection. After the above two procedures, there were 4720 words left.

4.2.3 Document Indexing

As we described in the "Indexing of Data" section in chapter 2, TFIDF takes into account the popularity of a word both in individual documents and across the entire corpus. In general, the TFIDF measure is considered relatively more fair than TF and IDF measures. Different from the Term-Strength measure, however, we have to admit that TFIDF's performance may vary from task to task. That is, in some tasks, the TFIDF measure may adversely impact on the final classification results. From the literature, TFIDF measure has been reported as working well with the two classification algorithms we introduced in chapter 2 (see also chapter 5 for more details about the algorithms). In this work, therefore, we choose TFIDF as the weighting system in this project. By normalization, all TFIDF values of features are limited within 0 and 1.

4.2.4 Representing Link Information

This project is different from ordinary text classification because we try to make use of the link information that exists in HTML-like documents. In this case, therefore, we have

had to find a way to represent the link information in a document to prepare it for use by the text classifiers.

Both in-links and out-links connect to other documents. As we mentioned earlier in this chapter, every document is assigned to at least one classes. Those connected documents belong to some classes. These classes are called linked classes. We count the number of documents in a linked class. The number can be considered as TF when we put the linked classes information as extra features into the datasets.

- **In-linked classes**: Documents from outside point to the target document. We count the number of times a linked class is hit by in-linked documents.
- **Out-linked classes**: Documents from outside are pointed by the target document. We count the number of out-linked documents that belong to each of these linked classes.
- **Combined-linked classes**: Add in-linked classes to the corresponding out-linked classes. This results in 11 linked class features in the datasets, but the values of each linked-class features are the sum of both in- and out-linked classes.

After that, we apply TFIDF weighting processes onto the datasets to calculate weights for all linked-class features based on the entire corpus (taking into account both ordinary features and linked-class features across all documents).

4.3 Feature Selection Based on IG

As previously mentioned, documents may contain a large number of words. This makes the dimension of the VSM very large. Moreover, more features in a datasets does not mean that it will always give better results. In fact, in many cases, if we have many less meaningful features involved in text classification processes, the results can be worse. Feature selection process, therefore, is very important and critical to text classification accuracy.

In this project, the feature selection is done based on the IG values of every feature in different datasets. In other words, we calculate IG values for every feature in different datasets; and then based on the IG values, we select the most informative features for the corresponding datasets. (see chapter 2 and 3 for more details about IG)

After feature selection based on IG values, we generate datasets that contain from 100 to 1000 features respectively. This allows us to have more datasets for experiments. Meanwhile, the number of linked-class features involved may vary from dataset to dataset. This gives us the chance to see how those linked class features can affect the classification results. This will be shown in chapter 6.

4.3.1 IG Calculation for Both Ordinary Features and Linked Class Features

We calculated Information Gain values for all ordinary features and linked-class features. The IG values for every linked-class feature in different types of datasets are shown in Table 4.6. This does not clearly show if the linked-class features are informative.

All the features in three different datasets are sorted separately based on their IG values. The three datasets are Data-IN, Data-OUT and Data-COMBINE. This will produce a set of ranks for all the features in datasets. Table 4.7 shows the ranks for each of the linked-class features in different dataset.

After we applied IG_{doc} interpretation on the datasets, because there are three different interpretations of IG formulae, we also need to calculate Term-tf-Based and Term-tfidf-Based IG values for every feature. Feature IG_{tf} values and ranks of every linked-class feature are shown in Table 4.8 and 4.9. Feature IG_{tfidf} values and ranks of linked-class features are shown in Table 4.10 and 4.11.

4.3.2 Features Selection

According to the ranks, we select the features. Because the three different IG interpretations of the feature selection processes are the same, we use IG_{doc} as an example to show how it works.

Considering the 100-feature set, according to the ranks, Data-IN, Data-OUT and Data-COMBINE will contain 93 ordinary features and 7 linked-class features. This is shown in Table 4.12:

As the size of datasets increases, more linked-class features are selected into the datasets. The progressive steps are shown in Table 4.13, 4.14, 4.15 and 4.16. The additional linked-class features are shown in boldface.

From the above mentioned tables, we find that the linked-class features are very informative compare with the total number of features in the datasets (i.e. 4720 ordinary features plus 11 linked-class features). Seven out of eleven of them are in the top 100. Even the lowest ranked linked-class feature can be selected into a 500 feature set. Based on this fact, when we try to use the link information to improve the accuracy of Text Classification, we should not simply use all of them all the time. Accordingly, in different datasets, we will choose some of the linked-class features according to their ranks in the entire dataset. They may, otherwise, produce side-effects on the accuracy of TC.

	Data-IN	**Data-OUT**	**Data-COMBINE**
Linked-Class 1	0.31202022	0.37906395	0.32616667
Linked-Class 2	0.19367386	0.19288451	0.19499561
Linked-Class 3	0.17747708	0.15420193	0.16758418
Linked-Class 4	0.42695795	0.39993115	0.35211090
Linked-Class 5	0.22716998	0.24375235	0.22822936
Linked-Class 6	0.08342022	0.12232282	0.12012413
Linked-Class 7	0.04739141	0.03093586	0.04376227
Linked-Class 8	0.03576569	0.11570696	0.11477989
Linked-Class 9	0.66178589	0.65863231	0.63846821
Linked-Class 10	0.14161506	0.15136418	0.17940109
Linked-Class 11	0.04658101	0.08358460	0.07963537

Table 4.6: IG_{doc} values for each of the linked class features in different datasets.

	Data-IN	**Data-OUT**	**Data-COMBINE**
Linked-Class 1	23	13	19
Linked-Class 2	62	62	62
Linked-Class 3	67	73	69
Linked-Class 4	12	12	16
Linked-Class 5	53	50	53
Linked-Class 6	168	101	103
Linked-Class 7	259	430	296
Linked-Class 8	368	111	111
Linked-Class 9	9	9	9
Linked-Class 10	80	75	67
Linked-Class 11	264	170	175

Table 4.7: IG_{doc} rank for each of the linked-class features in different datasets.

	Data-IN	**Data-OUT**	**Data-COMBINE**
Linked-Class 1	0.33845463	0.37051562	0.54613478
Linked-Class 2	0.13766450	0.12675080	0.22898600
Linked-Class 3	0.23605631	0.10113099	0.27361446
Linked-Class 4	0.45373358	0.30487337	0.59248733
Linked-Class 5	0.15978502	0.18809483	0.31017613
Linked-Class 6	0.03617843	0.03958722	0.06685970
Linked-Class 7	0.11617969	0.04886759	0.13238405
Linked-Class 8	0.06433266	0.05819289	0.10502029
Linked-Class 9	0.56492027	0.62824703	1.00000000
Linked-Class 10	0.03907442	0.03549618	0.04790957
Linked-Class 11	0.01741421	0.01503613	0.02370090

Table 4.8: IG_{tf} values for each of the linked class features in different datasets.

	Data-IN	**Data-OUT**	**Data-COMBINE**
Linked-Class 1	7	6	5
Linked-Class 2	29	33	17
Linked-Class 3	16	38	12
Linked-Class 4	5	10	4
Linked-Class 5	23	17	8
Linked-Class 6	136	120	58
Linked-Class 7	38	100	28
Linked-Class 8	70	84	39
Linked-Class 9	3	3	1
Linked-Class 10	122	139	99
Linked-Class 11	323	392	202

Table 4.9: IG_{tf} rank for each of the linked-class features in different datasets.

	Data-IN	**Data-OUT**	**Data-COMBINE**
Linked-Class 1	0.17281269	0.14514073	0.20356686
Linked-Class 2	0.08364676	0.06536322	0.11881272
Linked-Class 3	0.08382564	0.03820868	0.07863477
Linked-Class 4	0.24323978	0.14691091	0.26138003
Linked-Class 5	0.11574679	0.11707617	0.19353046
Linked-Class 6	0.03090490	0.04243020	0.06313626
Linked-Class 7	0.02432499	0.00919524	0.01493978
Linked-Class 8	0.02912022	0.05959266	0.07182961
Linked-Class 9	0.39099791	0.38302429	0.62909171
Linked-Class 10	0.01845825	0.03424789	0.03035541
Linked-Class 11	0.01353047	0.02617892	0.02967360

Table 4.10: IG_{tfidf} values for each of the linked class features in different datasets.

	Data-IN	**Data-OUT**	**Data-COMBINE**
Linked-Class 1	18	23	17
Linked-Class 2	61	77	43
Linked-Class 3	60	161	65
Linked-Class 4	14	21	13
Linked-Class 5	41	41	18
Linked-Class 6	193	134	83
Linked-Class 7	243	736	455
Linked-Class 8	202	88	72
Linked-Class 9	4	4	2
Linked-Class 10	343	177	200
Linked-Class 11	491	228	209

Table 4.11: IG_{tfidf} rank for each of the linked-class features in different datasets.

	Linked-class Features
Data-IN	1, 2, 3, 4, 5, 9, 10
Data-OUT	1, 2, 3, 4, 5, 9, 10
Data-COMBINE	1, 2, 3, 4, 5, 9, 10

Table 4.12: Linked classes features were selected into 100-feature set (by IG_{doc}).

	Linked-class Features
Data-IN	1, 2, 3, 4, 5, 9, 10, **6**
Data-OUT	1, 2, 3, 4, 5, 9, 10, **6**, **8**, **11**
Data-COMBINE	1, 2, 3, 4, 5, 9, 10, **6**, **8**, **11**

Table 4.13: Linked classes features were selected into 200-feature set (by IG_{doc}).

	Linked-class Features
Data-IN	1, 2, 3, 4, 5, 6, 9, 10, **7 11**
Data-OUT	1, 2, 3, 4, 5, 6, 8, 9, 10, 11
Data-COMBINE	1, 2, 3, 4, 5, 6, 8, 9, 10, 11, **7**

Table 4.14: Linked classes features were selected into 300-feature set (by IG_{doc}).

	Linked-class Features
Data-IN	1, 2, 3, 4, 5, 6, 7, 9, 10, 11, **8**
Data-OUT	1, 2, 3, 4, 5, 6, 8, 9, 10, 11
Data-COMBINE	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Table 4.15: Linked classes features were selected into 400-feature set (by IG_{doc}).

	Linked-class Features
Data-IN	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Data-OUT	1, 2, 3, 4, 5, 6, 8, 9, 10, 11, **7**
Data-COMBINE	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Table 4.16: Linked classes features were selected into 500-feature set (by IG_{doc}).

4.4 Generation of Datasets with Link Information

In this project, we are concerned with different ways regarding the use of link information. They are in-linked classes, out-linked classes and combined-linked classes. Based on the existing datasets that only contain the ordinary features, we put 11 extra features in to form new datasets. These extra 11 features represent the linked-class information, and the weights of each of them in different documents. We then name the newly generated datasets Data-IN, Data-OUT and Data-COMBINE respectively.

For a given document d, we denote its classes information by

$$c(d) = \left[(c_1(d), c_2(d), \ldots, c_{|C|}(d)\right]$$

where $c_k(d) = 1$ if document d belongs to class k; $c_k(d) = 0$, otherwise.

The first case, consider a document d and assume that, $\{\widetilde{d}_j\}_{j=1}^{|N_{in}|}$ denotes the set of in-linked documents of d. The in-linked classes information of document d_j is

$$c(d_j) = \left[c_1(\widetilde{d}_j), c_2(\widetilde{d}_j), \ldots, c_{|C|}(\widetilde{d}_j)\right],$$

where $c_k(\widetilde{d}_j) = 1$ if document d_j belongs to class k, otherwise $c_k(\widetilde{d}_j) = 0$.

For these in-linked classes information, we consider a vector

$$in(d) = \left[in_1(d), in_2(d), \ldots, in_{|C|}(d)\right]$$

where

$$in_k(d) = \sum_{j=1}^{|N_{in}|} c_k(\widetilde{d}_j), \quad k = 1, \ldots, |C|.$$

The value of $in_k(d)$ is the frequency of in-linked-class feature k in document d. In other words, this value can be considered the same as the TF value of in-linked-class feature k. As we mentioned, we use TFIDF to weight each of the words in a document. After we calculate the TF values of in-linked-class features, IDF, another basic element of TFIDF is obtained by using the following methods.

For IDF values of the in-linked-class features in a document d, we consider a vector

$$in(idf) = \left[in(idf)_1, in(idf)_2, \ldots, in(idf)_{|C|}\right]$$

where

$$in(idf)_k = \sum_{j=1}^{|N|} a_k(d_j), \quad k = 1, \ldots, |C|,$$

and

$$a_k(d_j) = \begin{cases} 1 & \text{if } in_k(d_j) > 0; \\ \\ 0 & \text{otherwise.} \end{cases}$$

After we calculated both TF and IDF values, we obtain TFIDF value of every feature in different documents by using

$$ilc_k(d_j) = \frac{in_k(d_j)}{in(idf)_k},$$

and normalize them by using the equation 2.3.

Each document d, therefore, will be represented by a vector of features

$$d = \left[w_1(d), \ldots, w_{|\mathcal{F}|}(d), ilc_1(d), \ldots, ilc_{|C|}(d)\right],$$

where w_i represents the weight (TFIDF value) of ordinary word i in document d.

By putting all the documents together, we obtain a dataset Data-IN which contains both ordinary features and in-linked-class features for every document.

For the second case, we consider the use of out-linked classes information. We have a set of out-linked documents $\{\widetilde{d}_j\}_{j=1}^{|N_{out}|}$ of document d. The out-linked classes information of document d can be presented as a vector

$$out(d) = \left[out_1(d), out_2(d), \ldots, out_{|C|}(d)\right]$$

where

$$out_k(d) = \sum_{j=1}^{|N_{out}|} c_k(\widetilde{d}_j), \quad k = 1, \ldots, |C|.$$

The value of $out_k(d)$ is the frequency of out-linked-class feature k in document d. In other words, this value can be considered the same as the TF value of out-linked-class feature k. IDF of out-linked-class features k is obtained by using the following methods.

For IDF values of the linked-class features in a document d, we consider a vector

$$out(idf) = \left[out(idf)_1, out(idf)_2, \ldots, out(idf)_{|C|}\right]$$

where

$$out(idf)_k = \sum_{j=1}^{|N|} a_k(d_j), \quad k = 1, \ldots, |C|,$$

and

$$a_k(d_j) = \begin{cases} 1 & \text{if } out_k(d_j) > 0; \\ \\ 0 & \text{otherwise.} \end{cases}$$

After we calculated both TF and IDF values, we obtain TFIDF value of every feature in different documents by using

$$olc_k(d_j) = \frac{out_k(d_j)}{out(idf)_k}, \tag{4.1}$$

and normalize them by using the equation 2.3.

We then obtain a vector

$$d = \left[w_1(d), \ldots, w_{|\mathcal{F}|}(d), olc_1(d), \ldots, olc_{|C|}(d)\right].$$

for each document and get a new dataset which is called Data-OUT.

After considering the previous two ways of using in-linked and out-linked class information separately, we could combine the two different information group to form the third way of using linkage information of an on-line document. The linked documents can be called combined-linked documents. In this case, we will have a set of neighbor documents which contains both in-linked and out-linked documents. This can be represented as $\{d_j\}_{j=1}^{|N_{io}|}$, where $|N_{io}| = |N_{in}| + |N_{out}|$. There could be some neighboring documents that appear in both the in-linked and out-linked document sets. If so, it is regarded as two documents in the combined-linked document set. The combined-linked classes information can be re represented by a vector

$$io(d) = \left[io_1(d), io_2(d), \ldots, io_{|C|}(d)\right]$$

where

$$io_k(d) = \sum_{j=1}^{|N_{io}|} c_k(d_j) = \sum_{j=1}^{|N_{in}|} c_k(\widetilde{d_j}) + \sum_{j=1}^{|N_{out}|} c_k(\widetilde{\widetilde{d_j}}), \quad k = 1, \ldots, |C|.$$

The value of $io_k(d)$ is the TF of combined-linked-class feature k in document d. IDF of combined-linked-class features k is obtained by using the following methods.

For IDF values of the linked-class features in a document d, we consider a vector

$$io(idf) = \left[io(idf)_1, io(idf)_2, \ldots, io(idf)_{|C|}\right]$$

where

$$io(idf)_k = \sum_{j=1}^{|N|} a_k(d_j), \quad k = 1, \ldots, |C|,$$

and

$$a_k(d_j) = \begin{cases} 1 & \text{if } io_k(d_j) > 0; \\ \\ 0 & \text{otherwise.} \end{cases}$$

After we calculated both TF and IDF values, we obtain TFIDF value of every feature in different documents by using

$$iolc_k(d_j) = \frac{io_k(d_j)}{io(idf)_k}, \tag{4.2}$$

and normalize them by using the equation 2.3.

Finally, we obtain a vector

$$d = \left[w_1(d), \ldots, w_{|\mathcal{F}|}(d), iolc_1(d), \ldots, iolc_{|C|}(d)\right].$$

to represent every document d in the dataset Data-COMBINE.

4.5 An Example of Counting Linked-Class Frequencies

For example, we assume that document d has d_1, d_2, d_5, d_{111} and d_{712} as its in-linked documents

$$d \Rightarrow \{d_1,\ d_2,\ d_5,\ d_{111},\ d_{712}\}.$$

We then find all the classes to which these in-linked documents belong.

$$d_1 \Rightarrow \{c_1,\ c_2,\ c_{11}\};$$
$$d_2 \Rightarrow \{c_2,\ c_4,\ c_5\};$$
$$d_5 \Rightarrow \{c_1,\ c_2\};$$
$$d_{111} \Rightarrow \{c_1\};$$
$$d_{712} \Rightarrow \{c_2,\ c_4\}.$$

After this, we count the in-linked classes occurrences in_k for each of the classes within the in-linked documents.

		in_1	in_2	in_3	in_4	in_5	in_6	in_7	in_8	in_9	in_{10}	in_{11}
	d_1	1	1	0	0	0	0	0	0	0	0	1
	d_2	0	1	0	1	1	0	0	0	0	0	0
	d_5	1	1	0	0	0	0	0	0	0	0	0
	d_{111}	1	0	0	0	0	0	0	0	0	0	0
+	d_{712}	0	1	0	1	0	0	0	0	0	0	1
	$in(d)$	3	4	0	2	1	0	0	0	0	0	1

In-linked classes occurrences.

The same document d most likely will also have some out-linked documents. The in-linked documents and out-linked documents may be different. If you find a same neighbor document appears in both in-linked and out-linked document sets, it means that, there is a *loop* between the original document and the neighbor document. Here, a loop means that the original document and a linked document are pointing to each other. The number of out-linked documents may also not be the same as the in-linked documents. We assume that document d also has out-linked documents

$$d \Rightarrow \{d_1,\ d_2,\ d_{10},\ d_{71}\}.$$

These out-linked documents belong to some classes too.

$$d_1 \Rightarrow \{c_1,\ c_2,\ c_{11}\};$$
$$d_2 \Rightarrow \{c_2,\ c_4,\ c_5\};$$
$$d_{10} \Rightarrow \{c_5\};$$
$$d_{71} \Rightarrow \{c_1,\ c_5\}.$$

Now, we count the out-linked classes occurrences out_k for document d.

		out_1	out_2	out_3	out_4	out_5	out_6	out_7	out_8	out_9	out_{10}	out_{11}
	d_1	1	1	0	0	0	0	0	0	0	0	1
	d_2	0	1	0	1	1	0	0	0	0	0	0
	d_{10}	0	0	0	0	1	0	0	0	0	0	0
+	d_{71}	1	0	0	0	1	0	0	0	0	0	0
	$out(d)$	2	2	0	1	3	0	0	0	0	0	1

Out-linked classes occurrences.

After we finished all above processes, now, we can obtain the combined-linked classes occurrences io_k by adding the in-linked and out-linked classes occurrences together.

		io_1	io_2	io_3	io_4	io_5	io_6	io_7	io_8	io_9	io_{10}	io_{11}
	$in(d)$	3	4	0	2	1	0	0	0	0	0	1
+	$out(d)$	2	2	0	1	3	0	0	0	0	0	1
	$io(d)$	5	6	0	3	4	0	0	0	0	0	2

Combined-linked classes occurrences.

4.6 Evaluation Process

4.6.1 Four-fold Cross Validation

In our project, we use 4-fold cross-validation. Because documents are multi-labeled, we arrange these folds as follows.

The first step, we must consider all the combinations of multi-labeled classes and partition them based on the classes they belong to. The second, we fold each of the partitions, rather than the entire dataset, so that we could always keep the pattern for a particular class's combination from the testing set in the training set. For this purpose, we will have to consider the number of documents that exist in a class's combination partition.

The procedure of document selection for both the training set and the test set in different fold are described in Table 4.17.

N.docs	Fold1	Fold2	Fold3	Fold4
1	test			
2		test		
3			test	
4				test
5	test			
6		test		
7			test	
⋮	⋮	⋮	⋮	⋮

Table 4.17: Documents selection for number of documents greater than 2.

In above mentioned table, if any cell is marked with "test", it means the corresponding document(s) should be selected into the test set; and a blank cell means the corresponding document(s) will be put into the training set. If the number of documents is 1, there will be no document selected into the test sets in all 4 folds. If the number is 2, one document will be put into the training set and another will be in test set; and in different folds, we swap them around.

As we have already discussed, the datasets are generated fold. Because of this, there are two facts that we must consider. The first is that, the documents in the training set and the test set should be different in different folds. The other is that, the properties of documents in the test sets are unknown except their ordinary features (words). This indicates that the link information cannot be revealed when a document is in test sets and has to be fold-based, changes along the fold changes. Therefore, we should not use and filter out any link information of a document from the test sets when we select and generate linked-class features for every fold.

4.6.2 Evaluation Measures

As mentioned in the previous section, all testing will be done based on four-fold corss-validation. Every test on a dataset will actually be performed on four different folds that

have the same feature set, but different documents in them. To evaluate the results for one dataset, we must evaluate the results for all four of its folds. After that, the average precision will be calculated based on the formula listed below:

$$AP = \frac{\sum_{i=1}^{4} d_i \times ap_1}{\sum_{i=1}^{4} d_i} \tag{4.3}$$

where AP is the average precision across four folds' average precision, d_i is the number of documents in ith fold and ap_i is the average precision of the test results from ith fold.

We believe that only this setting can achieve impartiality of evaluation.

4.6.3 Summary

In this section, we have summarized the methodological steps we took to test our approach. A figure 4.1 illustrates the process we discussed in this chapter.

In our project, we have had to generate our own datasets, rather than using any of existing datasets. The main reason is the lack of support of link information and multi-label data. Thus, a crawler was made to retrieve HTML-like documents from the internal WWW networks of the University of Ballarat. All documents retrieved are decomposed into basic elements, such as words and links and stored accordingly in a database. After this, because the number of documents we retrieved was huge, for feasibility reasons, we selected only 2000 of them. This ensured that the size of future datasets is reasonable in term of the number of documents. Based on the content of these 2000 documents, we compiled a set of classes that covered all topics contained in the documents; and then, manually classified the documents into their class or classes.

After the above processes, the raw data contained 9972 words. We then remove all stop-words and returned all words to their stems. When the two processes were completed, the size of raw data was reduced and only meaningful words were left. The final number of features was 4720. This is about 52.67% reduction on the dimensionality of the feature space. After we had all the meaningful words, we put the linked-class features into datasets. The linked-class features show the status of linked-classes of a target document. Next, we indexed both the ordinary and linked-class features by applying the TFIDF weighting system.

Because the dimensionality of the feature space is still very large, we further reduced it by applying a feature selection procedure. In this project, we use the IG values of all features as the criterion of feature selection. Based on features' IG values, different sized datasets were formed. The datasets' sizes may vary from 100 to 1000 features. This procedure sped up the entire classification process.

In the experiments, we applied two different types of classification algorithms, SVMs and BoosTexter. SVMs and BoosTexter both are very well-studied and commonly used

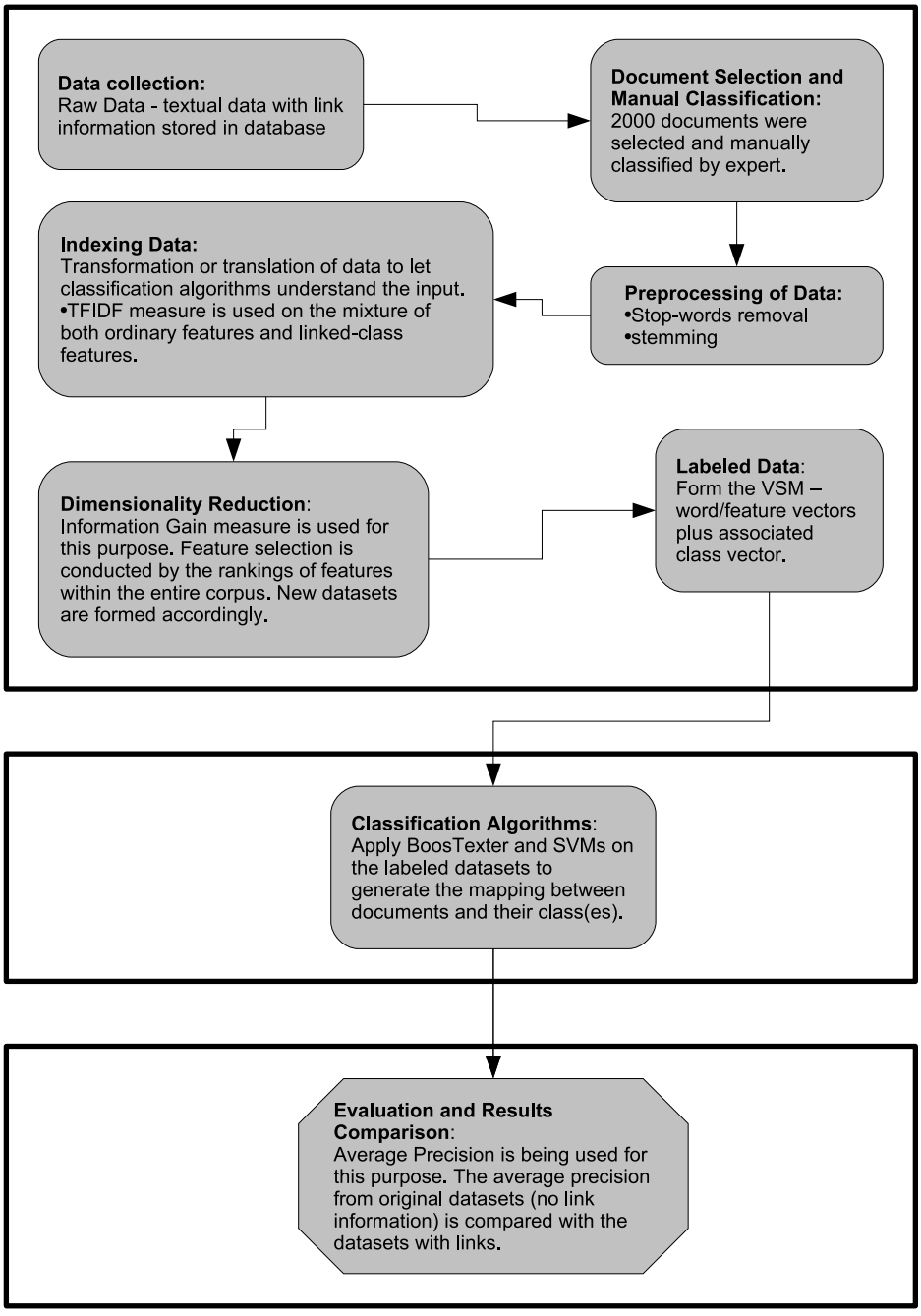

Figure 4.1: Illustration of processing steps in the project.

classification methods. We chose these two algorithms to test our datasets. We then applied them to the datasets that we had generated. After that, we used average precision to measure the classification results. First, we evaluated the results from original datasets which do not contain any link information. This set up the benchmark for future comparison. We then evaluate all other results. From observing the difference, we knew if our approach works.

In the next chapter, we will introduce the selected text classification algorithms that we used in this work. At the same time, we will try to analyze the strength and weakness of these algorithms for traditional Text Classification problems and Web Classification.

Chapter 5

Classification Algorithms

There has been extensive work done on text categorization, including techniques based on Decision Trees, Neural Networks, Nearest Neighbor methods, Rocchio's method, Support Vector Machines, Linear Least Squares, Naive Bayes, rule-based methods and more. It would be impossible to implement all of these algorithms and test them on our datasets, we select some of the best performed on average. They are Support Vector Machines and BoosTexter. One reason why we choose these two algorithms is that SVMs are optimization based classification algorithm, which tries to find an optimized result from the information provided by all features in a dataset. BoosTexter, on the other hand, is a rule-based classification algorithm, which tries to find a proper rule for each feature across every document in the dataset. By using these two radically different classification algorithms, we will see clearly the strengths and weakness of the way which we implement our approach. In this section, we will discuss these two algorithms a bit more in details.

5.1 Support Vector Machines

5.1.1 An Overview

Support Vector Machines (SVMs) are a set of related supervised learning methods used for classification and regression. They belong to a family of generalized linear classifiers. They also can be considered as a special case of Tikhonov regularization. A special property of SVMs is that they simultaneously minimize the empirical classification error and maximize the geometric margin. Hence, it is also known as the maximum margin classifier. This section aims to describe the central ideas of SV learning and SVMs.

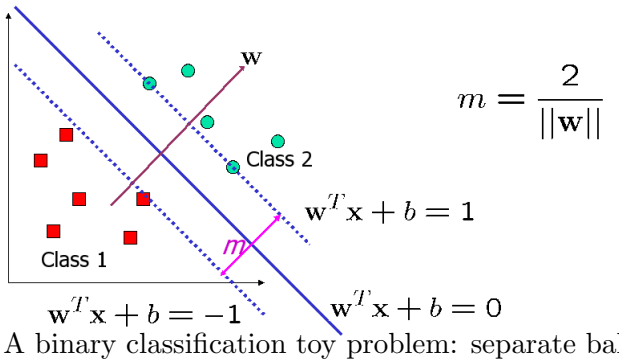

Figure 5.1: A binary classification toy problem: separate balls from diamonds.

5.1.2 Hyperplane Classifiers

Designing learning algorithms needs to come up with a class of functions whose capacity can be computed. The authors of [75, 73] considered the class of hyperplanes

$$(w \cdot x) + b = 0 \quad w \in \mathbb{R}^N, b \in R, \tag{5.1}$$

corresponding to decision functions

$$f(x) = sign((w \cdot x) + b), \tag{5.2}$$

and proposed a learning algorithm for separable problems, termed the Generalized Portrait, for constructing f from empirical data. This is based on two facts. First, among all hyperplanes separating the data, there exists a unique one yielding the maximum margin of separation between the classes,

$$\max_{w,b} \min\{||x - x_i|| : x \in \mathbb{R}^N, (w \cdot x) + b = 0, i = 1, \ldots, l\}. \tag{5.3}$$

Second, the capacity decreases as the margin increases

The optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it half-way between the two classes. Because the problem is separable, there exists a weight vector w and a threshold b such that $y_i \cdot ((w \cdot x_i) + b) > 0 (i = 1, \ldots, l)$. Rescaling w and b such that the point(s) closest to the hyperplane satisfy $|(w \cdot x_i) + b| = 1$, we obtain a canonical form (w, b) of the hyperplane, satisfying $y_i \cdot ((w \cdot x_i) + b) \geq 1$. Note that, in this case, the margin, measured perpendicularly to the hyperplane, equals $2/||w||$. This can be seen by considering two points x_1, x_2 on opposite sides of the margin, i.e. $(w \cdot x_1) + b = 1$, $(w \cdot x_2) + b = -1$, and projecting them onto the hyperplane's normal vector $w/||w||$.

To construct this Optimal Hyperplane, one solves the following optimization problem:

$$\text{minimize} \qquad \tau(w) = \frac{1}{2}||w||^2 \tag{5.4}$$

$$\text{subject to} \qquad y_i \cdot ((w \cdot x_i) + b) \geq 1, \quad i = 1, \ldots, l. \tag{5.5}$$

This constrained optimization problem is dealt with by introducing Lagrange multipliers $\alpha_i \geq 0$ and a Lagrangian

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{l} \alpha_i (y_i \cdot ((x_i \cdot w) + b) - 1). \tag{5.6}$$

Lagrangian L must be minimized with respect to the primal variables w and b and maximized with respect to the dual variables α_i which is a saddle point that has to be found. Intuitively, if a constraint 5.5 is violated, then $y_i \cdot ((w \cdot x_i) + b - 1 < 0$, in which case L can be increased by increasing the corresponding α_i. Meanwhile, w and b will have to change such that L decreases. To prevent $-\alpha_i(y_i \cdot ((w \cdot x) + b - 1)$ from becoming arbitrarily large, the change in w and b will ensure that, provided the problem is separable, the constraint will eventually be satisfied. Similarly, one can understand that for all constraints which are not precisely met as equalities, i.e. for which $y_i \cdot ((w \cdot x) + b) - 1 > 0$, the corresponding α_i must be 0: this is the value of α_1 that maximizes L. The latter is the statement of the Karush-Kuhn-Tucker complementarity conditions of optimization theory.

The condition that at the saddle point, the derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b} L(w, b, \alpha) = 0, \quad \frac{\partial}{\partial w} L(w, b, \alpha) = 0, \tag{5.7}$$

leads to

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{5.8}$$

and

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i. \tag{5.9}$$

The corresponding solution vector thus has an expansion in terms of a subset of the training patterns, namely those patterns whose α_i is non-zero, called Support Vectors. By the Karush-Kuhn-Tucker complementarity conditions

$$\alpha_i \cdot [y_i((w_i \cdot w) + b - 1] = 0, \quad i = 1, ..., l, \tag{5.10}$$

the Support Vectors as on the margin. All remaining examples of the training set are irrelevant: their constraint 5.5 does not play a role in the optimization problem, and they do not appear in the expansion of 5.9. This nicely displays the intuition of the problem: as the hyperplane in figure 5.1.2 is completely determined by the patterns closest to it, the solution should not depend on the other examples.

By substituting 5.8 and 5.9 into L, one eliminates the primal variables and arrives at the Wolfe dual of the optimization problem: find multipliers α_i which

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{5.11}$$

$$\text{subject to} \qquad \alpha_i \geq 0, \quad i = 1, \ldots, l, \text{ and } \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{5.12}$$

The hyperplane decision function can thus be written as

$$f(x) = \text{sign} \left(\sum_{i=1}^{l} y_i \alpha_i \cdot (x \cdot x_i) + b \right) \tag{5.13}$$

where b is calculated by using 5.10.

There is often only a subset of the constraints that become active. For example, if we put a ball in a box, most likely the ball will roll into one of the corners. Then, the constraints related to the walls are not necessary at all. Thus they are able to be removed without affecting anything else.

5.1.3 Feature Space and Kernels

To construct SV machines, the optimal hyperplane algorithm has to be augmented by a method for computing dot products in feature spaces that are non-linearly related to input space. The basic idea is to map the data into some other dot product space F, which is called the feature space, via a non-linear map

$$\Phi : \mathbb{R}^N \rightarrow F \tag{5.14}$$

and then, perform the above linear algorithm in F.

For example, suppose we have a set of documents $x \in \mathbb{R}^N$ where the most information is stored in the d-th order products (monomials) of entries x_j of x, that is $x_{j1}, \ldots, x_{jd}$, where $j_1, \ldots, j_d \in \{1, \ldots, N\}$. If this is the case, we may want to work with all those monomial features first in the feature space F of all products of d entries. This approach, however, fails for realistically sized problems: for N-dimensional input data, there exists $(N + d - 1)!/(d!(N - 1)!)$ different monomials.

This problem can be overcome by noting that both the construction of the optimal hyperplane in F and the evaluation of the corresponding decision function 5.13 only require the evaluation of dot products $(\Phi(x) \cdot \Phi(y))$, and never the mapped patterns $\Phi(x)$ in explicit form. This is crucial, because the dot products in some cases can be evaluated by a simple kernel

$$k(x, y) = (\Phi(x) \cdot \Phi(y)). \tag{5.15}$$

For example, the polynomial kernel

$$k(x, y) = (x \cdot y)^d \tag{5.16}$$

can be shown to correspond to a map Φ into the space spanned by all products of exactly d dimensions of $\mathbb{R}^N$. For instance, set $d = 2$ and $x, y \in \mathbb{R}^2$, we have [72]

$$(x \cdot y)^2 = (x_1^2, x_2^2, \sqrt{2}x_1x_2)(y_1^2, y_2^2, \sqrt{2}y_1y_2)^\top = (\Phi(x) \cdot \Phi(y)), \tag{5.17}$$

defining $\Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$.

By using $k(x, y) = (x \cdot y + c)^d$ with $c > 0$, we can take into account all products of order up to d.

More generally, the following theorem of functional analysis shows that the kernels k of the positive integral operators give rise to maps Φ such that 5.15 holds [44, 1, 7].

Theorem 5.1.1 *If k is a continuous symmetric kernel of a positive integral operator T, i.e.*

$$(Tf)(y) = \int_{\mathcal{C}} k(x, y) f(x) dx \tag{5.18}$$

with

$$\int_{\mathcal{C}\times\mathcal{C}} k(x, y) f(x) f(y) dx dy \geq 0 \tag{5.19}$$

for all $f \in L_2(\mathcal{C})$ ($\mathcal{C}$ being a compact subset of $\mathbb{R}^N$), it can be expanded in a uniformly convergent series (on $\mathcal{C} \times \mathcal{C}$) in terms of T's eigenfunctions ψ_j and positive eigenvalues λ_j,

$$k(x, y) = \sum_{j=1}^{N_F} \lambda_j \psi_j(x) \psi_j(y), \tag{5.20}$$

where $N_F \leq \infty$ is the number of positive eigenvalues.

Note that, originally proven for the case where $\mathcal{C} = [a, b]$ ($a < b \in \mathbb{R}$), this theorem also holds true for general compact spaces.

A similar way to characterize Mercer kernels is that they give rise to positive matrices $K_{ij} := k(x_i, x_j)$ for all $\{x_1, \ldots, x_l\}$ [58]. One of the implications that needs to be proved to show this equivalence follows from the fact that K_{ij} is a Gram matrix: for $\alpha \in \mathbb{R}^l$, we have $(\alpha \cdot K\alpha) = || \sum_{i=1}^{l} \alpha_i \Phi(x_i)||^2 \geq 0$.

From 5.20, it is straightforward to construct a map Φ into a potentially infinite-dimensional l_2 space which satisfies 5.15. For example, we may use

$$\Phi(x) = (\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), \ldots). \tag{5.21}$$

We would alternatively represent the feature space as the Hilbert space $\mathcal{H}_k$ which contains all linear combinations of the functions $f(\cdot) = k(x_i, \cdot)(x_i \in \mathcal{C})$, rather than an l_2 space. To ensure that the map $\Phi : \mathcal{C} \to \mathcal{H}_k$, which is in this case defined as

$$\Phi(x) = k(x, \cdot), \tag{5.22}$$

satisfies 5.15, we need to provide $(H)_k$ with a suitable dot product $\langle \cdot, \cdot \rangle$. In view of the definition of Φ, this dot product needs to satisfy

$$\langle k(x, \cdot), k(y, \cdot) \rangle = k(x, y), \tag{5.23}$$

which amounts to saying that k is a reproducing kernel for $\mathcal{H}_k$. For a Mercer kernel 5.20, such a dot product does exist. Because k is symmetric, ψ ($i = 1, \ldots, N_F$) can be chosen to be orthogonal with respect to the dot product in $L_2(C)$, i.e. $(\psi_j, \psi_n)_{L_2(C)} = \delta_{jn}$, using the Kronecker δ_{jn}. From this, we can construct $\langle \cdot, \cdot \rangle$ such that

$$\langle \sqrt{\lambda_j}\psi_j, \sqrt{\lambda_n}\psi_n \rangle = \delta_{jn}. \tag{5.24}$$

Substituting 5.20 into 5.23 proves the desired equality.

Apart from 5.16, sigmoid kernels and radial basis function kernels are also used. The sigmoid kernels are defined as follows

$$k(x, y) = \tanh\left(\kappa(x \cdot y) + \Theta\right) \tag{5.25}$$

for suitable values of gain κ and threshold Θ. As an example, in [1, 44, 63] they defined radial basis function kernels as below

$$k(x, y) = \exp\left(\frac{-||x - y||^2}{2\sigma^2}\right), \tag{5.26}$$

with $\sigma > 0$. Note that, when using Gaussian kernels, the feature space $\mathcal{H}_k$ thus contains all superpositions of Gaussians on $\mathcal{C}$ (plus limit points), whereas by definition of Φ 5.22, only single bumps $k(x, \cdot)$ have pre-images under Φ.

5.1.4 Support Vector Machines

SV machines should be able to compute an optimal hyperplane in a given feature space. To achieve this, we substitute $\Phi(x_i)$ for each training example x_i. The weight vector 5.9 then becomes an expansion in feature space, and will thus typically no longer correspond to the image of a single vector from the input space. Because all patterns only occur in dot products, by substituting Mercer kernels k for the dot product, based on 5.13, we can produce decision functions in the more general form

$$\begin{aligned} f(x) &= \operatorname{sign}\left(\sum_{i=1}^{l} y_i \alpha_i \cdot (\Phi(x) \cdot \Phi(x_i)) + b\right) \\ &= \operatorname{sign}\left(\sum_{i=1}^{l} y_i \alpha_i \cdot k(x, x_i) + b\right) \end{aligned} \tag{5.27}$$

and the following quadratic program:

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{5.28}$$

$$\text{subject to} \qquad \alpha_i \geq 0,\ i = 1, \ldots, l,\ \text{and} \sum_{i=1}^{l} \alpha_i y_i = 0. \tag{5.29}$$

Practically, a separating hyperplane may not exist. For example, if the noise level in a data set is very high, a large overlap of classes could occur. To allow for the possibility of examples violating, one introduces slack variables [15, 72]

$$\xi_i \geq 0, \qquad i = 1, \ldots, l, \tag{5.30}$$

along with relaxed constraints

$$y_i \cdot ((w \cdot x_i) + b) \geq 1 - \xi_i, \qquad i = 1, \ldots, l. \tag{5.31}$$

A classifier which generalizes well is then found by controlling both the classifier capacity (via $||w||$) and the number of training errors, minimizing the objective function

$$\tau(w, \xi) = \frac{1}{2}||w||^2 + C \sum_{i=1}^{l} \xi_i \tag{5.32}$$

subject to the constraints 5.30 and 5.31, for some value of the constant $C > 0$ determining the trade-off. Here, we use boldface Greek letters as a shorthand for the corresponding vectors $\xi = (\xi_1, \ldots, \xi_l)$. Incorporating kernels, and rewriting it in terms of Lagrange multipliers, this again leads to the problem of maximizing 5.28, subject to the constraints

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l, \text{ and } \sum_{i=1}^{l} \alpha_i y_i = 0. \tag{5.33}$$

For both separable and non-separable cases, the only difference is the upper bound C on the Lagrange multiplier α_i. From this, the influence of an individual pattern gets limited. As above, the solution takes the form of 5.27. Then, the threshold b can be calculated by exploiting the fact that for all SVs x_i with $\alpha_i < C$, the slack variable ξ_i is zero (This again follows from the KKT complementarity conditions), and thus

$$\sum_{i=1}^{l} y_j \alpha_j \cdot k(x_i, x_j) + b = y_i. \tag{5.34}$$

If one uses an optimizer that works with the double dual, one can also recover the value of the primal variable b directly from the corresponding double dual variable.

5.2 BoosTexter

5.2.1 An Overview

BoosTexter is an algorithm for text classification purposes. It is based on boosting, a well-known machine-learning technique. The main idea of boosting is to combine many simple and moderately inaccurate categorization rules into a single highly accurate categorization rule. These simple rules are trained sequentially. Conceptually, each rule is trained on the examples that were the most difficult to classify based on the preceding rules [62]. The boosting algorithm proposed in [62] was based on a new and improved family of boosting algorithms that are described and analyzed in [61]. This new family extends and generalizes Freund and Schapire's AdaBoost algorithm [24], which has been extensively studied and shown to perform well not only on text classification tasks, but also on other standard machine learning tasks.

As mentioned in earlier chapters, modern text categorization problems are usually multi-class and multi-labeled. This means that: first, there are more than two possible categories; second, examples (documents) can be relevant to more than one class. While most machine learning systems are designed for dealing with multi-class data, much less common are systems that can handle multi-label data. While numerous categorization algorithms, such as K - Nearest Neighbors, can be adapted to multi-label categorization problems, when machines learning and other approaches are applied to text categorization problems, a common method is to decompose a multi-label problem into many independent binary classification problems.

Alternatively, BoosTexter is a system which embodies four versions of boosting. These four versions are based on two extensions of AdaBoost that were specifically intended for multi-class, multi-label cases. In the first extension, the learning algorithm is to predict all of the correct labels. In the second extension, the goal is to design a classifier that ranks the labels so that the correct labels will receive the highest ranks.

5.2.2 Classification Configuration

In this section, we show the settings that we use to study multi-label text categorization.

Let $\mathcal{X}$ denote the domain of possible text documents and let $\mathcal{Y}$ be a finite set of labels or classes. We denote the size of $\mathcal{Y}$ by $k = |\mathcal{Y}|$.

In a single-label text classification task, each document $x \in \mathcal{X}$ is assigned to a single class $y \in \mathcal{Y}$. Thus, the goal of a classification task, in general, is to find a classifier $\mathcal{H} : \mathcal{X} \to \mathcal{Y}$ that minimizes the probability that $y \neq \mathcal{H}(x)$ on a newly observed example (x, y). In the multi-label case, each document $x \in \mathcal{X}$ may be assigned with multiple labels in $\mathcal{Y}$. A labeled example is, therefore, a pair (x, Y) where $Y \in \mathcal{Y}$ is the set of labels

assigned to x. The single-label case actually is a special case in which $|Y| = 1$ for all examples (documents). For $Y \subseteq \mathcal{Y}$, let us define $Y[l]$ for $l \in \mathcal{Y}$ to be

$$Y[l] = \begin{cases} +1 & \text{if} \quad l \in Y \\ -1 & \text{if} \quad l \notin Y. \end{cases}$$

BoosTexter is primarily interested in classifiers which produce a ranking of the possible labels for a given document with the hope that the appropriate labels will appear at the top of the ranking. Formally, the goal of learning is to produce a function of the form $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ with the interpretation that, for a given instance x, the labels in $\mathcal{Y}$ should be ordered according to $f(x, \cdot)$. That is, a label l_1 is considered to be ranked higher than l_2 if $f(x, l_1) > f(x, l_2)$. If Y is the associated label set for x, then a successful learning algorithm will tend to rank labels in Y higher than those not in Y.

5.2.3 Boosting Algorithms for multi-label multi-class problems

The purpose of boosting is to find a highly accurate classification rule by combining many weak or base hypotheses, each of which may be only moderately accurate. The boosting algorithm finds a set of weak hypotheses by calling the weak learner repeatedly in a series of rounds. These weak hypotheses are then combined into a single rule called the final or combined hypothesis.

In the simplest version of AdaBoost for single-label classification, the boosting algorithm maintains a set of importance weights over training examples. These weights are used by the weak learning algorithm whose goal is to find a weak hypothesis with moderately low error with respect to these weights. The boosting algorithm, therefore, can use these weights to force the weak learner to concentrate on the examples which are hardest to classify.

In multi-label cases, it is appropriate to maintain not only a set of weights over training examples, but also labels. As boosting progresses, those training examples and their corresponding labels that are difficult to be classified will be getting incrementally higher weights, on the other hand, those who are easier to be predicted correctly will be getting lower weights. For example, in a news classification problem, it may be easier to determine whether an article belongs to News, but harder to decide if its topic is particularly Sport. In this case, as boosting progresses, the weight of the label News for that article decreases while the weight of Sport increases. The aim of this process is to force the weak learning algorithm to concentrate on examples and labels that will be most beneficial to the overall goal of finding a highly accurate classification rule [62].

Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$ where $x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y}$
Initialize $D_1(i, l) = 1/(mk)$
For $t = 1, \ldots, T$:

- Pass distribution D_t to weak learner.
- Get Weak hypothesis $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:
$$D_{t+1}(i, l) = \frac{D_t(i, l) exp(-\alpha_t Y_i[l] h_t(x_i, l))}{Z_t}$$
where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$f(x, l) = \sum_{t=1}^{T} \alpha_t h_t(x, l).$$

Figure 5.2: The algorithm AdaBoost.MH.

AdaBoost.MH

The first boosting algorithm in BoosTexter is called AdaBoost.MH. This is specifically designed for multi-label classification problems. Let S be a set of training documents $\langle (x_1, Y_1), \ldots, (x_m, Y_m) \rangle$ where each instance $x_i \in \mathcal{X}$ and each label $Y_i \subseteq \mathcal{Y}$. As we mentioned, AdaBoost.MH maintains a set of weights as a distribution D_t over examples and labels. Initially, this distribution is uniform. At each round t, the distribution D_t is passed to the weak learner with the training data S. The weak learner computes a weak hypothesis h_t. The output of the weak learner is a hypothesis $h : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Then, $h(x, l)$ can be interpreted as a prediction as to whether or not label l is assigned to document x. In other words, it predicts the value of $Y[l]$. The magnitude of the prediction $|h(x, l)|$ is interpreted as a measure of "confidence" in the prediction. Next, we will discuss the weak learner a bit more.

A parameter α_t is then chosen and the distribution D_t is updated. In general, α_t is positive, the distribution D_t is updated in a manner that the weight of example-label pairs will be increased if they are misclassified by h_t. The final hypothesis ranks documents using a weighted vote of the weak hypotheses.

This algorithm is derived using a natural reduction of the multi-class, multi-label data

to binary data. Under this reduction, each example (x, Y) is mapped to k binary-labeled examples of the form $((x, l), Y[l])$ for all $l \in \mathcal{Y}$; that is, the instance or document part of each derived example is formally a pair (x, l), and the binary label associated with this instance is $Y[l]$. In other words, we can think of each observed label set Y as specifying k binary labels (depending on whether a label l is or is not included in Y), and we can then apply binary AdaBoost to the derived binary data. The algorithm that results from such a reduction is equivalent to AdaBoost.MH.

In [61], Schapire and Singer also proved a bound on the empirical Hamming loss of this algorithm. The upper-bound of Hamming loss of this algorithm is $\prod_{t=1}^{T} = Z_t$, where Z_t is the normalization factor computed on round t. This upper-bound can be used in guiding both the choice of α_t and the design of the weak learning algorithm. Together, theses choices should be geared on each round t toward the minimization of

$$Z_t = \sum_{i=1}^{m} \sum_{l \in \mathcal{Y}} D_t(i, l) exp(-\alpha_t Y_i[l] h_t(x_i, l)). \tag{5.35}$$

Note, the space and time-per-round requirements of AdaBoost.MH are $O(mk)$, do not include the call to the weak learner.

AdaBoost.MR

We next describe the second boosting algorithm used in BoosTexter. It is called AdaBoost.MR. Whereas AdaBoost.MH is designed to minimize Hamming loss, AdaBoost.MR is designed specifically to find a hypothesis which ranks the labels in a manner that hopefully places the correct labels at the top of the ranking.

With respect to a labeled observation (x, Y), we focus now only on the relative ordering of the crucial pairs l_0, l_1 for which $l_0 \notin Y$ and $l_1 \in Y$. A classification rule f misorders a crucial pair l_0, l_1 if $f(x, l_1) \leq f(x, l_0)$ so that f fails to rank l_1 above l_0. The goal now is to find a function f with a small number of misorderings so that the labels in Y are ranked above the labels not in Y. In other words, the goal is to minimize the average fraction of crucial pairs which are misordered, a quantity that we call the empirical ranking loss:

$$\frac{1}{m} \sum_{i=1}^{m} \frac{1}{|Y_i||\mathcal{Y} - Y_i|} |(l_0, l_1) \in (\mathcal{Y} - Y_i) \times Y_i : f(x, l_1) \leq f(x, l_0)|.$$

We assume that Y_i is never empty nor equal to all of $\mathcal{Y}$ for any instance. If there are such instances in the training set we can simply discard them since there is no ranking problem to be solved in this case and they do not carry any information.

Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$ where $x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y}$

Initialize $D_1(i, l_0, l_1) = \begin{cases} 1/(m \cdot |Y_i| \cdot |\mathcal{Y} - Y_i|) & if\ l_0 \notin Y_i\ and\ l_1 \in Y_i \\ 0 & else. \end{cases}$

For $t = 1, \ldots, T$:

- Train Weak learner using distribution D_t
- Get Weak hypothesis $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i, l_0, i_1) = \frac{D_t(i, l_0, l_1) exp(-\frac{1}{2}\alpha_t(h_t(x_i, l_0) - h_t(x_i, l_1)))}{Z_t}$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$f(x, l) = \sum_{t=1}^{T} \alpha_t h_t(x, l).$$

Figure 5.3: The algorithm AdaBoost.MR.

AdaBoost.MR maintains a distribution D_t over $1, \ldots, m \times \mathcal{Y} \times \mathcal{Y}$ and denote the weight for instance x_i and the pair l_0, l_1 by $D_t(i, l_0, l_1)$. This distribution is 0, however, except on the relevant triples (i, l_0, l_1) for which l_0, l_1 is a crucial pair relative to (x_i, Y_i).

The weak hypotheses have the same form as in the previous algorithm $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Here we consider the weak hypotheses as providing a ranking of labels as described earlier. The update rule is a bit different. Let l_0, l_1 be a crucial pair relative to (x_i, Y_i) (D_t is 0 in all other cases). Assuming momentarily that $\alpha_t > 0$, this rule decreases the weight $D_t(i, l_0, l_1)$ if h_t gives a correct ranking ($h_t(x_i, l_1) > h_t(x_i, l_0)$, and increases this weight otherwise.

For the Hamming loss, the empirical ranking loss of this algorithm is at most $\prod_{t=1}^{T} Z_t$. Thus, as before, our goal in choosing α_t and h_t should be minimization of

$$Z_t = \sum_{i, l_0, l_1} D_t(i, l_0, l_1) exp(\frac{1}{2}\alpha_t(h_t(x_i, l_0) - h_t(x_i, l_1))) \tag{5.36}$$

This algorithm is somewhat inefficient when there are many labels since, ideally, we

Given: $(x_1, Y_1), \ldots, (x_m, Y_m)$ where $x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y}$
Initialize $v_i(i,l) = (m|Y_i||\mathcal{Y} - Y_i|)^{-\frac{1}{2}}$
For $t = 1, \ldots, T$:

- Train Weak learner using distribution D_t (as defined by equation 5.37
- Get Weak hypothesis $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$v_{t+1}(i,l) = \frac{v_t(i,l)exp(-\frac{1}{2}\alpha_t Y_i[l]h_t(x_i,l))}{\sqrt{Z_t}}$$

  where

$$Z_t = \sum_i \left[\left(\sum_{l \notin Y_i} v_t(i,l)exp(\frac{1}{2}\alpha_t h_t(x_i,l)) \right) \left(\sum_{l \in Y_i} v_t(i,l)exp(\frac{1}{2}\alpha_t h_t(x_i,l)) \right) \right]$$

Output the final hypothesis:

$$f(x,l) = \sum_{t=1}^{T} \alpha_t h_t(x,l).$$

Figure 5.4: A more efficient version of AdaBoost.MR: on each round of boosting and for each example, the running time is linear in the number of labels ($O(mk)$)

need to maintain $|Y_i||\mathcal{Y} - Y_i|$ weights for each example in training data, and each weight must be updated on every round. The space complexity and time-per-round complexity, therefore, can reach $\theta(mk^2)$. Which is very bad in terms of efficiency. In fact, the same algorithm can be implemented in a very different way which only uses $O(mk)$ space and time-per-round. Based on the nature of the updates, Schapire and Singer [61] showed that it is reasonable to only maintain weights v_t over $1, \ldots, m \times \mathcal{Y}$. To do so, the algorithm needs to maintain the condition that when l_0, l_1 is a crucial pair relative to (x_i, Y_i), then

$$D_t(i, l_0, l_1) = v_t(i, l_0)\dot{v}_t(i, l_1) \tag{5.37}$$

at all times. The pseudo-code for this implementation is shown in Figure 5.2.3. Note, all space requirements and all per-round computations are $O(mk)$, with the possible exception of the call to the weak learner which is discussed in the following section.

5.2.4 Weak Hypotheses

In this section we will introduce the implementation of the weak learner and the choice of the parameter α_t. For BoosTexter, there are three implementations of weak learners for AdaBoost.MH and one for AdaBoost.MR. BoosTexter, therefore, can work with any of these four implementations.

Boosting is intended to be a general purpose method that can be combined with any classifier. It has been used in practice with decision trees and neural networks. BoosTexter focuses on the use of boosting with very simple classifiers. Particularly, for all the methods above, the hypotheses have the same basic form as a single-level decision tree. The test at root level of this tree is a simple check for the presence or absence of a term in the given document. All words and pairs of adjacent words are potential terms. Based only on the outcome of this test, the weak hypothesis outputs predictions and confidences that each label is associated with the document. For instance, a news categorization problem, a possible term can be John Howard, and the corresponding predictor is: "If the term *John Howard* appears in the document then predict that the document belongs to News with high confidence, To Finance with low confidence, and that it does not belong to Sports with high confidence. If, on the other hand, the term does not appear in the document, then predict that it does not belong to any of the classes with low confidence."

We formally denote a possible term by w, and let us define $w \in x$ to mean that w exists in document x. Based on the term, we will be interested in weak hypotheses h which make predictions of the form:

$$h(x,l) = \begin{cases} c_{0l} & \text{if} \quad w \notin x \\ c_{1l} & \text{if} \quad w \in x \end{cases}$$

where c_{jl} is a real number. The three weak learners for AdaBoost.MH differ only with respect to possible restrictions which were place on the values of these numbers.

The weak learners explore all possible terms. For each term, values c_{jl} are chosen and a score is determined for the resulting weak hypothesis. As soon as a weak learner finishes searching all terms, the weak hypothesis with the lowest score is selected and returned by the weak learner. For AdaBoost.MH, this score will always be an exact calculation of Z_t as we defined in equation 5.35 since, as we noted, minimization of Z_t is a reasonable guiding principle in the design of the weak learning algorithm. For AdaBoost.MR, on the other hand, we cannot find a analytical solution for the problem of minimizing Z_t. Instead, an approximation of Z_t is used. We are going to discuss how to choose values c_{jl} and the use of approximation of Z_t in the coming section.

AdaBoost.MH with Real-Valued Predictions

The first weak learner is called *real* AdaBoost.MH. It permits unrestricted real-valued predictions c_{jl}.

Considering minimization of Z_t, the values c_{jl} should be computed as follows for a given term w: Let $X_0 = \{x : w \notin x\}$ and $X_1 = \{x : w \in x\}$. Given the current distribution D_t, we calculate the following for each possible label l, for $j \in \{0, 1\}$, and for $b \in \{-1, +1\}$:

$$W_b^{jl} = \sum_{i=1}^{m} D_t(i, l) \left[x_i \in X_j \wedge Y_i[l] = b\right]. \tag{5.38}$$

For readability of notation, we denote W_{+1}^{jl} and W_{-1}^{jl} as W_+^{jl} and W_-^{jl} respectively. Here, W_+^{jl} (W_-^{jl}) is the weight (with respect to the distribution D_t) of the documents in partition X_j which are (are not) labeled by l.

It can be shown that Z_t is minimized for a particular term by choosing

$$c_{jl} = \frac{1}{2} \ln \left(\frac{W_+^{jl}}{W_-^{jl}} \right), \tag{5.39}$$

and by setting $\alpha_t = 1$. These settings imply that

$$Z_t = 2 \sum_{j \in \{0,1\}} \sum_{l \in \mathcal{Y}} \sqrt{W_+^{jl} W_-^{jl}}. \tag{5.40}$$

Thus, we choose the term w for which this value of Z_t is the smallest.

Actually, W_+^{jl} and W_-^{jl} may be very small or even zero, in which case $c_j l$ as defined in equation 5.39 will be very large or infinite in magnitude. In practice, such large predictions may cause many problems, and there may be theoretical reasons to suspect that large, overly confident predictions will increase the tendency to overfit. To limit the magnitudes of the predictions, in BoosTexter's implementation, the values c_{jl} are calculated as below:

$$c_{jl} = \frac{1}{2} \ln \left(\frac{W_+^{jl} + \varepsilon}{W_-^{jl} + \varepsilon} \right). \tag{5.41}$$

In BoosTexter, ε is set to $1/mk$. This has the effect of bounding $|c_{jl}|$ by roughly $\frac{1}{2} \ln (1/\varepsilon)$ because both W_+^{jl} and W_-^{jl} are limited between 0 and 1.

AdaBoost.MH with Real-Valued Predictions and Abstaining

In the previous section we discussed real AdaBoost.MH which assigns confidence values both when terms are present and absent in a document. This implicitly indicates that the absence of a term carries information about the possible classes a document may belong to. From our intuitive knowledge, above tacit assumption may not stand for all cases. We, therefore, may ask the weak learner to abstain whenever the given term does not appear in a document. This can be accomplished by forcing each weak hypothesis to output a confidence value of zero for documents which do not contain the given term. Hence, this version of AdaBoost.MH algorithm is call *real abstaining* AdaBoost.MH.

For a given term w, this weak learner chooses predictionsc_{1l} for documents which contain w exactly as before. For the rest of documents, the prediction values c_{0l} are all set to zero. Hence, the term w has no influence on the classification if it does not appear in the document. As in previous section, α_t is set to 1.

Let $W_0 = \sum_{i:xi \in X_0} D_t(i,l))$ be the weight of all documents that do not contain w. It can be shown that

$$Z_t = W_0 + 2 \sum_{l \in \mathcal{Y}} \sqrt{W_+^{jl} W_-^{jl}}, \tag{5.42}$$

and, as before, on each round we choose a term w for which the value Z_t is the smallest.

One advantage of this algorithm over the previous one is an improvement in the running time because we need only consider the documents that include a given term w when computig Z_t. Since, typically the number of documents that include a non-trivial term is only a small fraction of the training data, this version is in practice fifteen percent faster than the previous one.

AdaBoost.MH with Discrete Predictions

The next weak learner forces the predictions c_{jl} of the weak hypotheses to be either $+1$ or -1. This is the more standard setting in which predictions do not carry confidences. This version is called *discrete* AdaBoost.MH.

With this restriction on the range of weak hypotheses, we can still minimize Z_t for a given term w using the following method. With the same notation defined in section 5.2.4, we set

$$c_{jl} = \text{sign}\left(W_+^{jl} - W_-^{jl}\right)$$

which can be viewed as a (weighted) majority vote over examples in block X_j for each label l. Let

$$r_t = \sum_{j \in \{0,1\}} \sum_{l \in \mathcal{Y}} \left| W_+^{jl} - W_+^{jl} \right|. \tag{5.43}$$

Then it can be shown that, for the purposes for minimization Z_t, we should choose

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 + r_t}{1 - r_t} \right)$$

giving

$$Z_t = \sqrt{1 - r_t^2}.$$

AdaBoost.MR with Discrete Predictions

In this section, we will describe a weak learner for AdaBoost.MR. Here, we will minimize Z_t as defined in equation 5.36. Unfortunately, the exact minimization of this quantity is not as straightforward as it was for AdaBoost.MH. Therefore, only discrete predictions in $-1, +1$, are considered, also an approximation for Z_t is used as a score, rather than an exact computation. This version of the weak learner is called the *discrete* AdaBoost.MR.

For a given hypothesis h_t, let

$$r_t = \frac{1}{2} \sum_{i, l_0, l_1} D_t(i, l_0, l_1)(h(x_i, l_1) - h(x_i, l_0)).$$

Then, similar to the analysis for the discrete AdaBoost.MH, it can be shown that $Z_t \leq \sqrt{1 - r_t^2}$ if we choose

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 + r_t}{1 - r_t} \right). \tag{5.44}$$

Because there is no way to efficiently minimize Z_t exactly, a weak learner should find a way to minimizes the upper bound $\sqrt{1 - r_t^2}$. Then, the upper bound is used as the score in choosing the best weak hypothesis.

For efficiency, it is important to note that the quantity r_t can be computed efficiently in terms of the weights v_t (defined in question 5.37). Let

$$d_t(i, l) = \frac{1}{2} v_t(i, l) \sum_{l' : Y_i[l'] \neq Y_i[l]} v_t(i, l').$$

Then it can be shown [61] that

$$r_t = \sum_{i,l} d_t(i,l) Y_i[l] h(x_i, l).$$

Hence, for a particular term w, we should choose

$$c_{jl} = \text{sign}\left(\sum_{i:x_i \in X_j} d_t(i,l) Y_i[l] \right)$$

which gives

$$r_t = \sum_{j \in \{0,1\}} \sum_{l \in \mathcal{Y}} \left| \sum_{i:x_i \in X_j} d_t(i,l) Y_i[l] \right|. \tag{5.45}$$

Thus, the term w will be chosen to maximize this quantity, and predictions will be assigned correspondingly. The parameter α_t is set as in equation 5.44.

Searching for a good weak hypothesis can be very time consuming when the training set is large. An inverted list is used. This list stores each term in the list of documents in which it appears. On each round, when searching for a good weak hypothesis, the program scans the list and for each term it evaluates term's prediction confidences c_{jl} according to the version of AdaBoost algorithms that is running. A straightforward implementation would require scanning the entire collection for each term. Precomputing certain values, however, can save a significant amount of time. For example, in AdaBoost.MH, the program first computes on each round once for all j the following values

$$W^{jl} = \sum_{i:x_i \in X_j} D_t(i,l).$$

Next, the program sums over the documents in which each term appears to find the values W_+^{jl} for each term by using the inverted list. Then, we calculate $W_-^{jl} = W^{jl} - W_+^{jl}$, and find c_{jl} and the corresponding values for Z_t. Hence, the amount of time spent on each round searching for a weak hypothesis is proportional to the total number of occurrences of all the terms in the training collection. After a weak hypothesis is found, it takes $O(mk)$ time to update the distribution $D_t(i,l)$.

The multi-label text categorization system introduced above is call BoosTexter. It can be used with any of the four implementations of weak learners described above. A brief summary of the different implementations is give in table 5.2.4.

Version	Loss	Prediction	α_t
Real MH	Hamming	$c_{jl} = \frac{1}{2} \ln \left(\frac{W_+^{jl}}{W_-^{jl}} \right)$ $(j \in \{0,1\})$	1
Real & abstaining MH	Hamming	$c_{ol} = 0$ $c_{1t} = \frac{1}{2} \ln \left(\frac{W_+^{jl}}{W_-^{jl}} \right)$	1
Discrete MH	Hamming	$c_{jl} = \text{sign} \left(W_+^{jl} - W_-^{jl} \right)$	$\frac{1}{2} \ln \frac{1+r_t}{1-r_t}$ [r_t defined in eq 5.43]
Discrete MR	Ranking	$c_{jl} = \text{sign} \left(\sum_{i:x_i \in X_j} d_t(i,l) Y_i[l] \right)$	$\frac{1}{2} \ln \frac{1+r_t}{1-r_t}$ [r_t defined in eq 5.43]

Table 5.1: Summary of the properties of the four weak learners for multi-label text categorization.

In this chapter, we have discussed the two algorithms which we used in this project, in next chapter, we will show the classification results that they produced, and analyze why out-links worked in most of the cases.

Chapter 6

Experiments and Results Analysis

In the previous chapter, we had a look at the theory behind the two classification algorithms that we used. Now in this chapter, we will present the classification results that were produced by the algorithms. We will also discuss the possible impact on average precision from different interpretations of Information Gain (IG) formula (see also chapter 3). Finally, we will give a brief analysis on why we obtained such results.

6.1 The study of different Interpretations of IG formula In Text Classification

According to the discussion about IG in chapter 3, we implemented both the original and our new interpretations of IG formulae in the C programming language. We used the programs to generate IG values for the features in all datasets. Then, based on the rankings of all the features in a particular dataset, we generated new sub-datasets which the number of features in each sub-dataset ranged from 100 to 1000 respectively. We then applied the SVM algorithm to these datasets to obtain the classification results. We will show the reason why we chose only SVM in this case in section 6.2.3. Roughly speaking, SVMs are more stable than BoosTexter in dealing with our datasets. Comparing these results will allow us to see the differences and possible impact on the final classification results between the original and our new interpretations of IG formulae.

In the following tables, "Doc-Based" means the IG values of the features are calculated based on original interpretation of IG formula, counting the number of documents for calculating IG; "Term-tf-Based" and "Term-tfidf-Based" indicate that the IG values are computed based on term frequencies and term TFIDF values respectively. Moreover, "Original" means the datasets doesn't contain any linkage information; Data-IN, Data-OUT and Data-COMBINE indicate the datasets that contain in-link-class, out-linked-class and combined-linked-class features respectively.

	Doc-Based	Term-tf-Based	Term-tfidf-Based
Features 100	82.24	82.37	81.30
Features 200	82.84	86.49	85.50
Features 300	87.73	88.10	87.69
Features 400	88.77	89.13	88.76
Features 500	90.01	89.63	90.08
Features 600	90.95	90.46	90.73
Features 700	91.58	90.95	91.43
Features 800	92.12	91.76	91.98
Features 900	92.14	92.24	92.38
Features 1000	92.17	92.48	92.86

Table 6.1: IG comparison - average precision of training phase for original datasets (SVM).

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Features 100	76.53	76.23	75.50
Features 200	77.74	78.74	77.70
Features 300	79.10	79.24	78.97
Features 400	79.55	79.20	79.19
Features 500	79.67	79.26	79.77
Features 600	79.41	79.89	79.98
Features 700	79.69	80.29	80.49
Features 800	80.23	80.50	80.56
Features 900	80.15	80.59	80.68
Features 1000	80.42	80.65	80.82

Table 6.2: IG comparison - average precision of testing phase for original datasets (SVM).

Next, we present the comparison of SVM classification results which were obtained from datasets generated based on different IG interpretations. See Table 6.1 to 6.8 for details.

Recall that the aim of IG computation is to show how informative a particular feature is within a corpus in which it exists. Having this in mind, after ranking all features within a corpus, we selection features (*a.k.a* feature exclusion) into datasets based on the rank of each feature. It will allow us to have datasets that are smaller in size but keep or even improve the average precision. Here, we must clarify that, the term "feature selection" we used in this thesis does not mean the particular research area of Feature Selection, but a set of processes which select a specific number of features based on their IG values to form new datasets.

Now look at the tables from table 6.1 to 6.8. We can see the differences between the original interpretation and our new term-based interpretations. The Doc-Based columns

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Features 100	82.23	82.96	81.95
Features 200	86.06	86.64	85.54
Features 300	87.54	88.07	87.68
Features 400	88.76	89.22	88.87
Features 500	89.88	89.80	89.62
Features 600	90.77	90.53	90.67
Features 700	91.39	91.04	91.43
Features 800	91.93	91.72	91.94
Features 900	92.10	92.27	92.30
Features 1000	92.20	92.52	92.69

Table 6.3: IG comparison - average precision of training phase for Data-IN (SVM).

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Features 100	76.58	76.06	75.75
Features 200	77.84	78.02	77.84
Features 300	78.16	78.45	78.85
Features 400	78.87	78.93	79.06
Features 500	78.83	79.23	78.99
Features 600	78.77	79.35	79.51
Features 700	79.18	79.69	79.93
Features 800	79.58	80.16	80.19
Features 900	79.75	80.25	80.21
Features 1000	79.70	80.21	80.13

Table 6.4: IG comparison - average precision of testing phase for Data-IN (SVM).

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Features 100	82.69	83.26	82.30
Features 200	86.49	87.24	85.84
Features 300	87.85	88.36	88.01
Features 400	89.20	89.60	89.19
Features 500	90.31	90.16	90.18
Features 600	91.17	90.88	90.99
Features 700	91.85	91.29	91.73
Features 800	92.22	92.10	92.23
Features 900	92.48	92.49	92.59
Features 1000	92.50	92.80	93.18

Table 6.5: IG comparison - average precision of training phase for Data-OUT (SVM).

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Features 100	76.62	76.64	75.82
Features 200	78.35	78.66	78.05
Features 300	79.21	79.12	79.51
Features 400	79.91	79.59	79.73
Features 500	79.88	79.56	80.27
Features 600	79.89	80.09	80.66
Features 700	79.88	80.43	80.92
Features 800	80.25	80.70	80.89
Features 900	80.61	80.88	80.82
Features 1000	80.76	80.87	80.94

Table 6.6: IG comparison - average precision of testing phase for Data-OUT (SVM).

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Features 100	82.47	83.31	82.53
Features 200	86.10	86.79	85.76
Features 300	87.58	88.17	87.82
Features 400	88.81	89.06	87.78
Features 500	89.78	89.97	89.86
Features 600	90.69	90.58	90.72
Features 700	91.37	91.28	91.47
Features 800	91.91	91.97	92.02
Features 900	92.12	92.29	92.36
Features 1000	92.21	92.56	92.84

Table 6.7: IG comparison - average precision of training phase for Data-COMBINE (SVM).

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Features 100	76.55	76.49	76.09
Features 200	77.94	78.19	77.67
Features 300	78.35	78.60	79.08
Features 400	78.88	79.18	79.53
Features 500	79.15	79.12	79.25
Features 600	79.10	79.75	79.81
Features 700	79.21	79.85	79.94
Features 800	79.65	80.25	80.02
Features 900	80.10	80.30	80.12
Features 1000	81.80	80.41	80.09

Table 6.8: IG comparison - average precision of testing phase for Data-COMBINE (SVM).

show the classification results on the datasets that are generated based on the original interpretation of IG formula. The two Term-Based columns show the results on the datasets that are generated based on our term frequencies and term TFIDF values interpretations of IG formula respectively.

By observing the results, we find that the term-based interpretations gives better results than doc-based one. Moreover, term frequencies based interpretation gave significatly better results than term TFIDF values based one. Especially, for those smaller datasets, term frequencies based feature selections gave much better results than other two methods. This satisfies the purpose of calculating IG.

We now also present another two tables 6.9 and 6.10, which will show overall average precision among all datasets, including Original, Data-IN, Data-OUT and Data-COMBINE.

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Original	89.06	***89.36**	**89.27**
Data-IN	89.29	***89.48**	89.27
Data-OUT	89.68	***89.82**	89.62
Data-COMBINE	89.30	***89.60**	**89.32**

Table 6.9: IG comparison - overall average precision of training phase among all datasets (SVM).

	Doc-Based	Term-freq-Based	Term-tfidf-Based
Original	79.25	***79.46**	**79.37**
Data-IN	78.72	**79.04**	***79.05**
Data-OUT	79.54	**79.65**	***79.76**
Data-COMBINE	79.07	***79.22**	**79.16**

Table 6.10: IG comparison - overall average precision of testing phase among all datasets (SVM).

Bold numbers in the term-based columns means better results over doc-based ones. All the numbers labeled with a asterisk are best results for a particular data type among all three different interpretations.

From the results shown in Table 6.10 and 6.9, the term based interpretations in general work better than document based one with the SVM algorithm. The term frequency based interpretation is the best one. It does not provide the best results in all cases, but it does provide better results over the original interpretation every time.

6.2 The Results from Text Classification

More than 500 experiments were done on the datasets we generated earlier in this project to see how the linked-class features would affect the final classification results. The experiments use SVM and BoosTexter respectively. As we mentioned, we have four types of data, Original, Data-IN, Data-OUT and Data-COMBINE. Recall that, Original datasets only contain ordinary features (words from documents); Data-IN and Data-OUT contains either in-linked-class features or out-linked-class features respectively; Data-COMBINE includes both in- and out-linked class features with their weights summed. For each of the data types, we generated a three different datasets based on the three different interpretations of IG formula, say IG_{doc}, IG_{tf} and IG_{tfidf}. In addition, we also divided each of the datasets into two parts, say training sets and testing sets, and four folds. All this settings here will allow us to see the changes of average accuracies of every test clearer.

6.2.1 Classification Results from SVM

The text classification results from SVM for all datasets in our project are listed in tables from 6.11 to 6.16.

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	82.24	82.23	**82.69**	82.47
Feature 200	82.84	86.06	**86.49**	86.10
Feature 300	87.73	87.54	**87.85**	87.58
Feature 400	88.77	88.76	**89.20**	88.81
Feature 500	90.01	89.88	**90.31**	89.78
Feature 600	90.95	90.77	**91.17**	90.69
Feature 700	91.58	91.39	**91.85**	91.37
Feature 800	92.12	91.93	**92.22**	91.91
Feature 900	92.14	92.10	**92.48**	92.12
Feature 1000	92.17	92.20	**92.50**	92.21

Table 6.11: SVM - training on datasets which are generated based on IG_{doc}

In these tables, numbers where an average precision from a link information involved dataset is higher than the original dataset are displayed in bold type. Therefore, by observing the tables, we found that, for both training and testing phases, Data-OUT provides the best results every time, Data-COMBINE is the second best and Data-IN is the third. To make it clearer, we also generated the overall average accuracy for each of the data types. This is shown in Tables 6.17 to 6.18.

From a summary table of tables 6.17 and 6.18, it is much more obvious that the Data-OUT always returns the best results from SVM in both training and testing phases

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	76.53	76.58	**76.62**	76.55
Feature 200	77.74	77.84	**78.35**	77.94
Feature 300	79.10	78.16	**79.21**	78.35
Feature 400	79.55	78.87	**79.91**	78.88
Feature 500	79.67	78.83	**79.88**	79.15
Feature 600	79.41	78.77	**79.89**	79.10
Feature 700	79.69	79.18	**79.88**	79.21
Feature 800	80.23	79.58	**80.25**	79.65
Feature 900	80.15	79.75	**80.61**	80.10
Feature 1000	80.42	79.70	**80.76**	81.80

Table 6.12: SVM - testing on datasets which are generated based on IG_{doc}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	82.37	82.96	**83.26**	83.31
Feature 200	86.49	86.64	**87.24**	86.79
Feature 300	88.10	88.07	**88.36**	88.17
Feature 400	89.13	89.22	**89.60**	89.06
Feature 500	89.63	89.80	**90.16**	89.97
Feature 600	90.46	90.53	**90.88**	90.58
Feature 700	90.95	91.04	**91.29**	91.28
Feature 800	91.76	91.72	**92.10**	91.97
Feature 900	92.24	92.27	**92.49**	92.29
Feature 1000	92.48	92.52	**92.80**	92.56

Table 6.13: SVM - training on datasets which are generated based on IG_{tf}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	76.23	76.06	**76.64**	76.49
Feature 200	78.74	78.02	**78.66**	78.19
Feature 300	79.24	78.45	**79.12**	78.60
Feature 400	79.20	78.93	**79.59**	79.18
Feature 500	79.26	79.23	**79.56**	79.12
Feature 600	79.89	79.35	**80.09**	79.75
Feature 700	80.29	79.69	**80.43**	79.85
Feature 800	80.50	80.16	**80.70**	80.25
Feature 900	80.59	80.25	**80.88**	80.30
Feature 1000	80.65	80.21	**80.87**	80.41

Table 6.14: SVM - testing on datasets which are generated based on IG_{tf}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	81.30	81.95	**82.30**	82.53
Feature 200	85.50	85.54	**85.84**	85.76
Feature 300	87.69	87.68	**88.01**	87.82
Feature 400	88.76	88.87	**89.19**	87.78
Feature 500	90.08	89.62	**90.18**	89.86
Feature 600	90.73	90.67	**90.99**	90.72
Feature 700	91.43	91.43	**91.73**	91.47
Feature 800	91.98	91.94	**92.23**	92.02
Feature 900	92.38	92.30	**92.59**	92.36
Feature 1000	92.86	92.69	**93.18**	92.84

Table 6.15: SVM - training on datasets which are generated based on IG_{tfidf}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	75.50	75.75	**75.82**	76.09
Feature 200	77.70	77.84	**78.05**	77.67
Feature 300	78.97	78.85	**79.51**	79.08
Feature 400	79.19	79.06	**79.73**	79.53
Feature 500	79.77	78.99	**80.27**	79.25
Feature 600	79.98	79.51	**80.66**	79.81
Feature 700	80.49	79.93	**80.92**	79.94
Feature 800	80.56	80.19	**80.89**	80.02
Feature 900	80.68	80.21	**80.82**	80.12
Feature 1000	80.82	80.13	**80.94**	80.09

Table 6.16: SVM - testing on datasets which are generated based on IG_{tfidf}

	Original	Data-IN	Data-OUT	Data-COMBINE
IG_{doc}	89.06	89.29	**89.68**	89.30
IG_{tf}	89.36	89.48	**89.82**	89.60
IG_{tfidf}	89.27	89.27	**89.62**	89.32

Table 6.17: SVM - overall average precision for different datasets in training phase

	Original	Data-IN	Data-OUT	Data-COMBINE
IG_{doc}	79.25	78.72	**79.54**	79.07
IG_{tf}	79.46	79.04	**79.65**	79.22
IG_{tfidf}	79.37	79.05	**79.76**	79.16

Table 6.18: SVM - overall average accuracy for different datasets in testing phase

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	88.08	87.97	**88.59**	88.04
Feature 200	93.87	94.06	**94.15**	94.26
Feature 300	96.61	96.99	**97.20**	97.04
Feature 400	97.25	96.46	**97.60**	97.60
Feature 500	97.50	97.83	**97.94**	97.83
Feature 600	97.73	97.97	**98.02**	97.98
Feature 700	97.83	98.04	**98.21**	98.05
Feature 800	97.90	98.40	**98.28**	98.21
Feature 900	97.94	98.26	**98.33**	98.20
Feature 1000	98.22	98.23	**98.30**	98.33

Table 6.19: BoosTexter($round = 300$) - training on datasets which are generated based on IG_{doc}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	78.78	79.22	**79.64**	78.75
Feature 200	80.19	79.71	**80.24**	79.69
Feature 300	81.17	79.40	80.62	79.92
Feature 400	81.86	80.45	81.66	79.75
Feature 500	81.23	80.31	81.11	79.30
Feature 600	80.84	80.00	**81.46**	80.17
Feature 700	81.33	80.70	80.60	80.79
Feature 800	81.09	80.28	**81.22**	79.58
Feature 900	81.83	80.21	81.13	79.90
Feature 1000	81.10	80.03	80.64	80.77

Table 6.20: BoosTexter($round = 300$) - testing on datasets which are generated based on IG_{doc}

among all other data types. Data-IN and Data-COMBINE also obtained better results in some case, mainly at the training phase.

6.2.2 Classification Results from BoosTexter

As introduced in chapter 5, BoosTexter [62] is a rule-based text multi-class multi-label text classification algorithm. It has been well studied and widely used in various tasks. In our project, BoosTexter is also used. The classification results are shown in Tables 6.19 to 6.26.

From the BoosTexter results, we should be able to observe that, out-linked-class features worked very well in the training phase for all experiments. In the test phase,

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	88.17	88.64	**88.61**	88.62
Feature 200	93.82	94.07	**94.48**	94.33
Feature 300	96.72	96.81	**97.19**	96.91
Feature 400	97.28	97.36	**97.49**	97.54
Feature 500	97.51	97.66	**97.79**	97.67
Feature 600	97.71	97.73	**98.04**	97.82
Feature 700	97.76	97.95	**98.12**	97.98
Feature 800	97.78	97.97	**98.19**	98.25
Feature 900	98.06	98.32	**98.34**	98.29
Feature 1000	98.20	98.34	**98.48**	98.25

Table 6.21: BoosTexter($round = 300$) - training on datasets which are generated based on IG_{tf}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	79.16	78.72	**79.29**	76.57
Feature 200	80.14	79.21	**80.99**	78.21
Feature 300	80.99	80.21	80.09	79.22
Feature 400	81.36	80.44	80.26	79.81
Feature 500	81.48	79.06	81.06	80.10
Feature 600	81.58	80.53	**81.79**	80.20
Feature 700	80.87	80.64	**81.67**	80.62
Feature 800	81.21	80.40	81.16	80.75
Feature 900	81.91	81.13	81.45	80.18
Feature 1000	81.91	81.85	81.26	80.24

Table 6.22: BoosTexter($round = 300$) - testing on datasets which are generated based on IG_{tf}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	87.94	88.16	**88.54**	88.04
Feature 200	93.80	93.65	93.76	93.62
Feature 300	96.62	96.77	**96.93**	96.95
Feature 400	96.96	97.38	**97.54**	97.47
Feature 500	97.66	97.51	**97.83**	97.76
Feature 600	97.90	97.91	**98.13**	97.96
Feature 700	97.86	98.07	**98.20**	98.11
Feature 800	98.09	98.13	**98.38**	98.16
Feature 900	98.18	98.18	**98.36**	98.34
Feature 1000	98.15	98.29	**98.42**	98.24

Table 6.23: BoosTexter($round = 300$) - training on datasets which are generated based on IG_{tfidf}

	Original	Data-IN	Data-OUT	Data-COMBINE
Feature 100	78.99	76.98	78.41	76.84
Feature 200	80.58	80.02	79.89	79.79
Feature 300	80.61	80.03	**81.44**	79.92
Feature 400	81.25	80.08	80.86	80.21
Feature 500	81.65	79.57	**82.23**	80.64
Feature 600	81.45	80.55	**81.45**	80.69
Feature 700	81.66	80.44	81.64	80.46
Feature 800	82.14	81.38	81.94	80.14
Feature 900	82.62	80.72	81.48	80.47
Feature 1000	81.32	81.85	81.10	80.50

Table 6.24: BoosTexter($round = 300$) - testing on datasets which are generated based on IG_{tfidf}

	Original	Data-IN	Data-OUT	Data-COMBINE
IG_{doc}	96.29	96.42	**96.66**	96.55
IG_{tf}	96.30	96.48	**96.67**	96.57
IG_{tfidf}	96.32	96.40	**96.61**	96.47

Table 6.25: BoosTexter ($round = 300$) - overall average precision for different datasets in training phase

	Original	Data-IN	Data-OUT	Data-COMBINE
IG_{doc}	80.94	80.03	80.83	79.86
IG_{tf}	81.06	80.22	80.90	79.59
IG_{tfidf}	81.23	80.16	81.04	79.97

Table 6.26: BoosTexter ($round = 300$) - overall average accuracy for different datasets in testing phase

when the size of the feature set is equal to 100, 200 and some other numbers, the out-linked-class features produced better results. This simply means that out-linked-class features are good and can help to improve the performance of BoosTexter. In Table 6.27, we present the best average precisions we obtained from 100, 200 and 300 feature sets.

# Feature	The Best Precision	Data Type	Feature Selection
100	79.64	Data-OUT	IG_{doc}
200	80.99	Data-OUT	IG_{tf}
300	81.44	Data-OUT	IG_{tfidf}

Table 6.27: BoosTexter - best precision achieved in test phase.

We observe that, the best average precisions for datasets with 100, 200 and 300 features out of all cases (Data-IN, Data-OUT, Data-COMBINE, IG_{doc}, IG_{tf} and IG_{tfidf}) is achieved when BoosTexter worked with Data-OUT. We note that, in the 100-feature datasets, there were 7 out-linked-class features; in 200-feature and 300-feature datasets, there were 10 out of 11 out-linked-class features. This means that the improvement on average precision was caused by the use of out-linked-class features.

Looking at Table 6.20, for example, the IG_{doc} based Data-OUT that contains 200 features obtained better accuracy than the Original datasets. This means that out-linked-class features are informative and improved the BoosTexter's performance. According to Table 4.7, in 200-feature sets, 10 out of 11 out-linked-class features have been selected into the datasets, these out-linked-class features remain unchanged until 500-feature sets. In 300- and 400-feature sets' cases, BoosTexter did not work as well as it did with 100- or 200-feature sets. This however would not be expected to happen, unless some of the newly added-in features are NOT imformative. These features are ***ordinary features***, rather than out-linked-class features. This analysis can be extended on to IG_{tf} and IG_{tfidf} based datasets as well. Furthermore, we also noted that, for IG_{tfidf} based datasets, 100- and 200-feature sets did not provide better results, but 300-feature sets provided much better results. This means that it is very likely that in the previous two cases (100- and 200 features sets), based to IG_{tfidf} rankings, we excluded some very informative features from the datasets and left some less imformative ones behind. This is very similar to the cases in IG_{doc} and IG_{tf}.

With the above analysis in mind, we decided to implement another set of experiments

to test our analysis. First, we noted that the IG_{doc} based Data-OUT provided better results when the numbers of features were 100 and 200. For example, there are ten linked-class features are in the 200-feature sets and better results were presented. This means that these ten linked-class features are more informative than the ten ordinary features which were replaced by the ten linked-class features in the 200-feature sets. Second, we also noted that in the 300-feature sets, datasets still contained the same ten linked-class features, however, the classification results were not as good as the 100- or 200-feature sets. Because no linked-class features were changed, this means that the newly added ordinary features had some possible side effect on the results. In this case, we believe that the ten excluded (by the 200-feature sets) ordinary features, namely sub-set A, may be less informative than the ten ordinary features, namely sub-set B, which were excluded by the 300-feature sets. We therefore replaced sub-set A with sub-set B. In other words, we replaced less informative features with potentially more informative features. We applied BoosTexter on the new datasets (300 features). We present the results in Table 6.28.

	Original	Data-OUT (A-in)	Data-OUT (B-in)
Training	96.61	97.20	97.12
Test	81.17	80.62	81.69

Table 6.28: Results from BoosTexter for IG_{doc} based 300-feature Data-OUT with sub-set A in it and with sub-set A replaced by B.

As you can easily discover, when BoosTexter was used with Data-OUT which contains sub-set B instead of sub-set A, we obtained the best average precision from all datasets. Our approach in the new implementation worked again.

With regard to this issue, we note that the IG measure is not a perfect method for choosing the "most" informative features. In other words, the rankings shown in Table 4.7, 4.9 and 4.11 do not necessarily mean that, higher ranked features are more imformative than lower ranked ones.

6.2.3 Analysis and Discussion

From observing the results that we obtained by using the SVM and BoosTexter algorithms, now we see that Data-OUT provides the best average precision among datasets of these four different types. This raised another question: Why does Data-OUT get the best results? To discover the reasons behind this improvement, we must analyze the purposes of creation and use of out-links.

As we explained, in-links are the links that point into a Web page from outside; reversely, out-links are the links that point out to other Web pages from the Web page in networks. Therefore, the out-links are more controllable from the point of view of Web designers, but the in-links are not. This means the purpose of an out-link is more

specific than an in-link. In other words, when a Web designer points their own Web page wp_{base} to some other pages in the network, this potentially indicates that the topic or content of wp_{base} is very related and close to the out-linked document. Therefore, the possibility that they belong to the same classes is higher. Moreover, wp_{base} may be linked to other less related documents, but the amount of these kind of linked Web pages in general would be much less than the opposite case. However, in-links are not the same as out-links. In-links could come from all kinds of sources, it is unpredictable. Although the in-linked documents may share some common topics as the base document, how they are similar to each other is unsure.

One may think that an in-link must correspond to an out-link, so the in-links and out-links should give the same effects. However, this not true. For example, if we look at the internal home page of the University of Ballarat, it out-linked to so many other HTML documents in a large range of various topics. It seems that the out-links should produce a worse result. However, this is only for one document only; and mis-classifying one document would not affect on the final classification results too much. Now, we think about this problem from those out-linked documents of the internal hope page views. We will see the internal page adds one misleading feature to every page to which it out-links. It is obvious that the bad effect from in-links on each of the documents is certainly greater than that of the out-links. On the other hand, out-links from those documents very likely may not point back to the internal home page at all. Therefore, out-linked classes information should be more accurate. This is especially true when the number of in-links or out-links is less. If number of in-links grows, there may be a chance that an overwhelming number of related in-linked documents that will compensate for those noisy in-links.

Thus, when we added those out-linked-class features into the datasets, we added more informative features than most of the ordinary features that existed in the datasets; when we added combined-linked and in-linked features, the situation is uncertain. According to the above analysis, we know that Data-IN and Data-COMBINE will sometimes get better results, but Data-OUT should always get better results in all cases than the datasets with no link information involved.

In this chapter, we presented some of the experiments results which are directly related to our work.

Conclusion:

- Out-linked-class features work very well with both SVM and Boostexter algorithms. It improves the classification results without doing much extra work on modifying classification algorithms or extracting and merging all neighboring documents' content to the target document;

- Information Gain is not perfect for measuring whether or not a feature is meaningful to a particular classifier. For every single classifciation algorithm, a corresponding feature selection criteria should be chosen.

In the next chapter, we will conclude this project and show the possible research directions for future work.

Chapter 7

Conclusion and Future Work

In this chapter, we will conclude the dissertation by summarizing our contributions, discussing some limitations to our solution, and proposing several directions for future work.

7.1 Conclusion

This section summarizes the primary contributions of this thesis to research into text classification on HTML documents.

In this work, we only concentrated on text classification based on Supervised Learning methods. A dataset here, therefore, usually consists of documents represented by a set of features and each document is assigned to one or more classes. The main goal of text classification is to map a document into one or more classes based on the presence or absence of words (or features) in the document.

To classify HTML documents, we should take advantages of some of the special features HTML documents have. One of them considered in this work is link information. This idea came from the basic fact that people mostly like to link their web page to some others that share similar or even the same topics and contents. If this is the case, the linkage information or those linked documents should be able to aid the processes of text classification in some ways.

There has been considerable research done on this topic. Most researchers are trying to use the linked documents' contents to adjust given classifiers (or learners) with extra information about a document. In this case, they combine all the linked documents with the original document, i.e. summarising the words' frequencies. The sum of each term's frequencies will be processed with TFIDF; and then applied to the classifiers. This sounds correct. However, because of the amount of extra information added and the uncertainty

of the way which words are being used in a particular document, the reported results on these methods are not satisfactory at all as we discussed in chapter 2.

The goal of our research is to use the link information within a HTML document to improve the accuracy of text classification methods while we keep the originality of the datasets. Considering the shortcomings of others' approaches, we think minimizing the impacts from linked documents is very important. In our project, therefore, the key idea behind text classification processes on HTML documents is that, instead of using a large amount of the linked document's contents, we only consider the mapping between linked documents and their corresponding classes. Linked documents here mean that, the documents that are connected to the source documents directly. In other words, they are next door neighbors. We call them "one step neighbors". In this way, the accuracy of text classification could be higher; and classifying a large amount linked documents will be computationally more efficient than those other approaches because fewer extra features are involved.

All link information, in this work, is represented by linked-class features. They are used to augment the word term based vector representation of a document. The new representation, therefore, leads to a vector compound of word term weights plus linked-classes weights. After this, we process each of the linked documents of the original document to find what class or classes the linked document belongs to and add 1 to the corresponding linked class features. Now, we have frequencies of every linked class feature. We mix it with the ordinary dataset and apply them to TFIDF processes. After all the above processes, linked class features can be treated as a normal features in a document.

Experimental results have shown that the linked-class features may positively impact the final text classification results. As we showed in chapter 6, out-linked-class features worked perfectly with SVMs. We also found that in-linked-class features, in some cases, improved classification results. When we applied BoosTexter, however, no linked-class features worked consistently at first. As we proved, this was caused by the incorrectly selected ordinary features which are not informative for Boostexter. Eventually, when we replaced those less informative ordinary features with others, out-linked-class features also aided BoosTexter to obtain better results.

7.2 Future Work

In this thesis, we have shown that by using the linked-class features, especially the out-linked class features, we could improve the text classification accuracy. In this section, we aim to answer the question: What are the directions for future work?

Limitations of the Data Collection

In this project, all the data were collected from our university's intranet. Therefore, the whole network's topologies are well-designed and well-implemented both physically and conceptually. Physically well-designed means wiring and how people use hardware to implement a network. Conceptually well-designed indicates that the web pages are well-designed and links in the pages are related to some other pages that have similar or the same topics. These two facts are more uncertain in a bigger network, such as the Internet. Therefore, in the future work, we will try to collect some data from the Internet or any bigger networks and test it to see if our approach still works.

Multi-step Neighbors

As mentioned previously, we only used "one-step neighbor" documents to aid in the text classification process. It is also feasible that we could use "multi-step neighbor" documents. This means that, the neighbor documents will not only include the neighbor documents that are connected with itself directly, but also those documents that are linked with it through other documents. The depth (number of step) can be indicated. The issue hidden here is how we could give the features from one- or multi-step documents proper weights.

Smaller Datasets with Enough Accuracy

As we mentioned, because of the large amount of on-line documents, the efficiency of text classification is more critical for processing on-line documents. For example, we could think of a search engine which has over a billion documents retrieved from the Internet. If we were to classify them based on a matrix which has over one billion rows (documents) and a few thousands columns (features), it would be an extremely time consuming job. In general, smaller datasets will be processed faster. In other words, shrinking the size of datasets is one way to improve the efficiency. If this is the case, mixing some of the ordinary features (words) and some of the linked class features may produce accuracy as good as if we use a full set of ordinary features, or even better. This is, of course, uncertain at this point. We would do more experiments on this topic to see if our assumptions are correct.

Feature Reduction

Using phrases instead of words or LSI for linked class features, we could do the same thing by combining the one or more ordinary feature with linked class features based on some rules to form a new feature. This procedure would reduce the number of features

in the final dataset while keeping the properties of the original features. If this can be achieved, we can increase the efficiency of any text classification methods and also keep the accuracy. Therefore, finding proper rules to combine ordinary features and linked-class features would be very important to our future work.

Final Remark

In this dissertation, we have presented a novel approach of using linkage information to improve text classification accuracy on HTML documents. Our approach does not require much more effort than that of solely using the original text classification method; and needs much less effort than some other methods that use linked documents. However, experimental evaluation reveals that, our solution produces better accuracy than others while being very efficient. This provides us with a new way of using linkage information built-in into HTML-like documents. There are also many possible ways to achieve better results based on this concept. We are willing to work towards achieving more and better results from this work in the future.

Bibliography

[1] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] L. K. Baker and A. K. McCallum. Distributional clustering of words for text classification. In *Proceedings of SIGIR'98, 21st ACM International Comference on Research and Development In Information Retrieval*, 1998.

[3] R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter. On feature distributional clustering for text categorization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in informaiton retrieval*, 2001.

[4] R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter. Distributional word clusters vs. words for text categorization. *Journal Of Machine Learning Research*, 2002.

[5] M. W. Berry, S. T. Dumais, and G. W. OBrien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, 1995.

[6] K. D. Bollacker, S. Lawrence, and C. L. Giles. Discovering relevant scientific literature on the web. *IEEE Intelligent Systems*, 15(2):42–47, 2000.

[7] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM Press.

[8] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Feature selection via mathematical programming. *INFORMS Journal on Computing*, 1998.

[9] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. *SIGMOD Rec.*, 27(2):307–318, 1998.

[10] R. Chellappa and A. K. Jain. *Markov Random Fields: Theory and Applications*. Academic Press, 1993.

[11] H. Chen and S. Dumais. Bringing order to the web: automatically categorizing search results. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–152, New York, NY, USA, 2000. ACM Press.

[12] W. W. Cohen. Automatically extracting features for concept learning from the web. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 159–166, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[13] D. Cohn and H. Chang. Learning to probabilistically identify authoritative documents. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 167–174, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[14] D. Cohn and T. Hofmann. The missing link - a probabilistic model of document content and hypertext connectivity. In *Neural Information Processing Systems 13*, 2001.

[15] C. Cortes and V. N. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995.

[16] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide web. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 509–516, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[17] M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Mach. Learn.*, 43(1-2):97–119, 2001.

[18] M. Craven, S. Slattery, and K. Nigam. First-order learning for web mining. In *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, pages 250–255, London, UK, 1998. Springer-Verlag.

[19] B. V. Dasarathy. Nearest neighbor (nn) norms: Nn pattern classification techniques. *McGraw-Hill Computer Science Series*, 1991.

[20] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[21] S. T. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 23(2):229–236, 1991.

[22] S. T. Dumais, G. W. Furnas, T. K. Landauer, and S. Deerwester. Using latent semantic analysis to improve information retrieval. In *Proceedings of CHI88: Conference on Human Factors in Computing*, pages 281–285, New York, 1988. ACM.

[23] C. Fox. A stop list for general text. *SIGIR Forum*, 24(1-2):19–21, r 90.

[24] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[25] E. Garfield. *Citation indexing - its theory and application in Science, Technology and Humanities*. John Wiley and Sons, New York, 1979.

[26] R. Ghani, S. Slattery, and Y. Yang. Hypertext categorization using hyperlink patterns and meta data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 178–185, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[27] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: an automatic citation indexing system. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, pages 89–98, New York, NY, USA, 1998. ACM Press.

[28] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003.

[29] T. Hofmann. Probabilistic latent semantic analysis. In *Proc. of Uncertainty in Artificial Intelligence, UAI'99*, Stockholm, 1999.

[30] B. C. How and K. Narayanan. An empirical study of feature selection for text categorization based on term weightage. In *WI '04: Proceedings of the Web Intelligence, IEEE/WIC/ACM International Conference on (WI'04)*, pages 599–602, Washington, DC, USA, 2004. IEEE Computer Society.

[31] R. A. Hummel and S. W. Zucker. On the foundations of relaxation labeling process. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-5:267–287, May 1983.

[32] Thorsten J. Learning to classify text using support vector machines: methods, theory and algorithms. *Klwer Academic Publishers*, 2002.

[33] T. Joachims, N. Cristianini, and J. Shawe-Taylor. Composite kernels for hypertext categorisation. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 250–257, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

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

[35] D. Koller and M. Sahami. Hierarchically classifying documents using a very few words. In Fisher D. H., editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, 1997.

[36] S. Lawrence, C. L. Giles, and K. Bollacker. Digital libraries and autonomous citation indexing. *Computer*, 32(6):67–71, 1999.

[37] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, October 1999.

[38] D. D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In C.Nedellec and C. Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning, Chemnitz*, number 1398 in Lecture Notes in Computer Science, pages 4–15. Heidelberg: Springer Verlag, 1998.

[39] J. B. Lovins. Development of stemming algorithm. *Mechanical Translation and Computational Linguistics*, 1968.

[40] Q. Lu and L. Getoor. Link-based classification using labeled and unlabeled data. In *ICML Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, Washington, DC, August 2003.

[41] Q. Lu and L. Getoor. Link-based text classification. In *Workshop "Text-Mining & Link-Analysis" at Eighteenth International Joint Conference on Artificial Intelligence*, August 2003.

[42] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *In AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[43] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Inf. Retr.*, 3(2):127–163, 2000.

[44] J. Mercer. Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations. *Royal Society of London Philosophical Transactions Series A*, 209:415–446, 1909.

[45] T. Mitchell. Machine learning and data mining. *Communications Of ACM*, 1999.

[46] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In Kathryn B. Laskey and Henri Prade, editors, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 467–475, San Francisco, California, July 1999. Morgan Kaufmann Publishers.

[47] Vapnik V. N. *Estimation of dependencies based on empirical data*. Springer, 1982.

[48] H. J. Oh, S. H. Myaeng, and M. H. Lee. A practical hypertext catergorization method using links and incrementally available class information. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 264–271, New York, NY, USA, 2000. ACM Press.

[49] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: a probabilistic analysis. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168, New York, NY, USA, 1998. ACM Press.

[50] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. In *Proceeding of 30th Annual Meeting of the Association for Computational Lingiustics*, 1993.

[51] A. Popescul, L. H. Ungar, S. Lawrence, and D. M. Pennock. Towards structural logistic regression: Combining relational and statistical learning. In *Proceedings of the Workshop on Multi-Relational Data Mining (MRDM-2002) at KDD-2002*, pages 130–141, Edmonton, Canada, 2002.

[52] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[53] J. R. Quinlan. Learning logical definitions from relations. *Mach. Learn.*, 5(3):239–266, 1990.

[54] J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Machine Learning: ECML-93, European Conference on Machine Learning, Proceedings*, volume 667, pages 3–20. Springer-Verlag, 1993.

[55] J. R. Quinlan and R. M. Cameron-Jones. Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13(3&4):287–312, 1995.

[56] F. Rosenblatt. The perceptron: A probalistic model for information storage and organization in the brain. *Psychological Review*, 1958.

[57] M. Sahlgren and R. Coster. Using a bag-of-concepts to improve the performance of support vector machines in text categorization. In *Proceedings of the 20th International Conference on Computational Lingiustics*, 2004.

[58] S. Saitoh. *Theory of Reproducing Kernels and its Applications*. Longman Scientific & Technical, Harlow, England, 1988.

[59] G Salton. Developments in automatic text retrieval. *Science*, 1991.

[60] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[61] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998.

[62] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[63] B. Scholkopf, Kah-Kay Sung, C.J.C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45(11):2758–65, Nov. 1997.

[64] F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[65] H. Sever, A. Gorur, and M. R. Tolun. Text categorization with ILA. In Adnan Yazici and Cevat Sener, editors, *Computer and Information Sciences - ISCIS 2003, 18th International Symposium, Antalya, Turkey, November 3-5, 2003, Proceedings*, volume 2869 of *Lecture Notes in Computer Science*, pages 300–307. Springer, 2003.

[66] S Slattery and M. Craven. Combining statistical and relational methods for learning in hypertext domains. In *ILP '98: Proceedings of the 8th International Workshop on Inductive Logic Programming*, pages 38–52, London, UK, 1998. Springer-Verlag.

[67] S. Slattery and T. M. Mitchell. Discovering test set regularities in relational domains. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 895–902, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[68] H. Small and B. C. Griffith. The structure of scientific literatures i: Identifying and graphing specialties. *Science Studies*, 4(1):17–40, Jan. 1974.

[69] Aixin Sun, Ee-Peng Lim, and Wee-Keong Ng. Web classification using support vector machine. In *WIDM '02: Proceedings of the 4th international workshop on Web information and data management*, pages 96–99, New York, NY, USA, 2002. ACM Press.

[70] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. In *Invited Paper to The 37th annual Allerton Conference on Communication, Control and Computing*, 1999.

[71] V. N. Vapnik. *Nature of Statistical Learning Theory.* Springer, 1995.

[72] V. N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[73] V. N. Vapnik and A. Chervonenkis. A note on one class of perceptrons. *Automation and Remote Control*, 25, 1964.

[74] V. N. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian].* Nauka, Mosco, 1974. German translation: W. Wapnik and A. Tscherwonenkis, Theorie der Zeichenerkennung, Akademie-Verlag, Berlin, 1979.

[75] V. N. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

[76] W. J. Wilbur and K. Sirotkin. The automatic identification of stop words. *J. Inf. Sci.*, 18(1):45–55, 1992.

[77] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorisation and retrieval. In W.B. Croft and C.J. van Rijsbergen, editors, *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval, Dublin*, pages 13–22. Heidelberg: Springer Verlag, 1994.

[78] Y. Yang. An evaluation of statistical approaches to text categorization. *Inf. Retr.*, 1(1-2):69–90, 1999.

[79] Y. Yang, T. Ault, and T. Pierce. Combining multiple learning strategies for effective cross-validation. In P. Langley, editor, *Proceedings of ICML-00, 17th International Conference on Machine Learning, Stanford*, pages 1167–1182. San Francisco: Morgan Kaufmann Publishers, 2000.

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

[81] Y. Yang, S. Slattery, and R. Ghani. A study of approaches to hypertext categorization. *J. Intell. Inf. Syst.*, 18(2-3):219–241, 2002.

[82] Z Zheng, R Srihari, and S Srihari. A feature selection framework for text filtering. In *Proceedings of 3rd International Conference on Data Mining (ICDM'03)*, 2003.