

A COMPUTATIONAL ANALYSIS OF A
LANGUAGE STRUCTURE IN NATURAL LANGUAGE
TEXT PROCESSING


A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY

BY
SİNAN EŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER, 2005

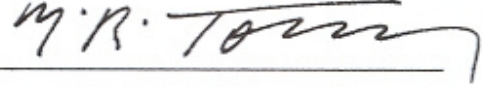
Approval of the Graduate School of Natural and Applied Sciences.



Prof. Dr. Yurdahan Güler

Director

I certify that this thesis satisfies all requirements as a thesis for the degree of Master of Science in Computer Engineering.



Prof. Dr. M. Reşit Tolun

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science in Computer Engineering.

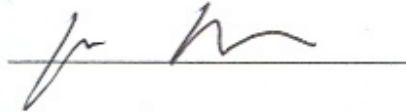


Asst. Prof. Dr. Ali Rıza Aşkun

Supervisor

Examining Committee Members:

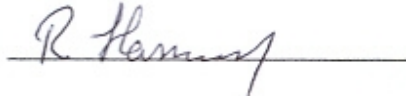
Prof. Dr. Hayri Sever



Asst. Prof. Dr. Ali Rıza Aşkun



Asst. Prof. Dr. Reza Z. Hassanpour



ABSTRACT

A COMPUTATIONAL ANALYSIS OF A LANGUAGE STRUCTURE IN NATURAL LANGUAGE TEXT PROCESSING

EŞ, SİNAN

M.Sc., Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Ali Rıza Aşkun

September 2005, 61 pages

Text categorization or classification is a general task of classifying un-organized natural language texts according to specific subject matter or category. Electronic mail (e-mail) filtering is a binary text classification problem which the user emails can be classified as legitimate (non-spam) or un-wanted mail (spam).

In this study, we tried to find a filtering solution that is able to automatically classify emails into spam and legitimate categories. In order to automatically and efficiently classify emails as spam or legitimate we took advantage of some Machine Learning methods and some novel ideas from Information Retrieval.

Keywords: Text Categorization, Email Filtering, Machine Learning.

ÖZ

DOĞAL DİL METİN İŞLEMEDE DİL YAPISININ SAYISAL ANALİZİ

EŞ, SİNAN

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Ali Rıza Aşkun

Eylül 2005, 61 sayfa

Metin sınıflandırma, belli konu başlığına veya kategorilerine göre düzenlenmiş doğal dil metinlerini sınıflandırmanın genel adıdır. Elektronik posta filtreleme, kullanıcı elektronik postalarının yasal veya istenmeyen olarak sınıflandırılabilirdiği ikili bir metin sınıflandırma problemidir.

Bu çalışmada, elektronik postaları otomatik olarak yasal veya istenmeyen kategorilerine ayırabilen bir filtreleme çözümü bulmaya çalışılmıştır. Elektronik postaları otomatik olarak yasal veya istenmeyen şeklinde sınıflandırmak için bazı Makine Öğrenim metotları ve Bilgi Elde Etme'nin bazı fikirlerinden faydalanarak, elektronik posta filtreleme işinde etkin sonuçlar elde edilmeye çalışılmıştır.

Anahtar Kelimeler: Metin Sınıflandırma, E-Posta Filtreleme, Makine Öğrenimi.

ACKNOWLEDGEMENTS

I would like to thank the following people for their time, effort and patience, enabling me to bring this thesis to its eventual completion.

Asst. Prof. Dr. Ali Rıza Aşkun, my supervisor, for his advice, helps and guidance during this thesis.

Asst. Prof. Dr. Reza Hassanpour, for his suggestions, helps and accepting to read and review this thesis.

Asst. Prof. Dr. Dilara Demirbulak, for showing interest to the subject matter and accepting to read and review this thesis.

Many thanks to my Family, for their patience and moral support during my years at the University.

I would also thank to my colleagues Saadettin Bolat, Hüseyin Şahin Akbal, Celal Küçükoguz and Ahmet Kabarcık for their valuable friendship.

TABLE OF CONTENTS

| | |
|--|------|
| ABSTRACT | iii |
| ÖZ..... | iv |
| TABLE OF CONTENTS | vi |
| LIST OF TABLES | viii |
| LIST OF FIGURES | ix |
| CHAPTER 1 | 1 |
| INTRODUCTION | 1 |
| 1.1 Background..... | 1 |
| 1.2 Thesis Objective | 2 |
| 1.3 Outline of the Thesis | 2 |
| CHAPTER 2 | 4 |
| LITERATURE SURVEY | 4 |
| 2.1 Definition of Text Categorization | 4 |
| 2.2 Text Categorization Tasks | 4 |
| 2.3 Applications of Text Categorization | 6 |
| 2.4 Document Representations with Bag of Words Approach..... | 7 |
| 2.4.1 Words Represented as Terms | 8 |
| 2.4.2 Common Methods for Weighting of Words..... | 11 |
| 2.5 Text Categorization Process | 13 |
| 2.6 Approaches for Text Categorization | 14 |
| 2.6.1 Knowledge Engineering Approach | 14 |
| 2.6.2 Machine Learning Approach..... | 16 |

| | |
|---|----|
| CHAPTER 3 | 26 |
| A BINARY TEXT CATEGORIZATION PROBLEM:..... | 26 |
| EMAIL SPAM FILTERING..... | 26 |
| 3.1. Introduction | 26 |
| 3.2. Background of the Study | 27 |
| 3.3. Machine Learning Applied to Spam Filtering | 29 |
| CHAPTER 4..... | 31 |
| SYSTEM DESIGN AND IMPLEMENTATION | 31 |
| 4.1. Description of the Problem Statement..... | 31 |
| 4.2. E-mail Spam Filter Tool Design..... | 32 |
| 4.3. Methodology..... | 33 |
| 4.4. E-mail Spam Filter Tool Implementation..... | 37 |
| 4.5. Training and Test Sets | 39 |
| CHAPTER 5 | 40 |
| RESULTS AND CONCLUSIONS | 40 |
| 5.1. Overview of Evaluating Text Classifiers | 40 |
| 5.2. Email Filtering System Evaluation..... | 41 |
| 5.3. Experimental Results..... | 42 |
| 5.4. Conclusions | 47 |
| 5.5. Future Work..... | 48 |
| REFERENCES | 49 |

LIST OF TABLES

| | |
|---|----|
| Table 5.1. Contingency Table for evaluating a binary classifier | 40 |
| Table 5.2. Experimental results obtained using Spam Assassin corpus | 43 |
| Table 5.3. System effectiveness measured on Spam Assassin corpus..... | 44 |
| Table 5.4. Experimental results obtained using Ling-Spam corpus..... | 45 |
| Table 5.5. System effectiveness measured on Ling-Spam corpus..... | 46 |

LIST OF FIGURES

| | |
|---|----|
| Figure 2.1. Action of a categorizer on a set of documents | 13 |
| Figure 2.2. Sample hand-crafted categorization rule | 15 |
| Figure 2.3. Naïve Bayes algorithms for learning and classifying text..... | 21 |
| Figure 2.4. A training algorithm for constructing decision trees | 23 |
| Figure 4.1. Architecture of the Email spam filter..... | 32 |
| Figure 5.1. System accuracy versus training data using Spam Assassin corpus.... | 43 |
| Figure 5.2. System accuracy versus training data using Ling-Spam corpus..... | 45 |

CHAPTER 1

INTRODUCTION

1.1 Background

Language is one of the fundamental aspects of human behavior and it is very important component of our lives. In written form it serves as long-term record of knowledge from one generation to the next. The written forms of natural language are called “texts”.

The goal of Natural Language Processing (NLP) is to create computational models of language in detail that we could write computer programs for some tasks involving natural language.

In this study we analyzed one of the text-based Natural Language research problems by means of Text Categorization (TC) or Text Classification. The goal in text categorization is to classify the topic or theme of a document.

As the growth of internet e-mail become a communication medium that enables fast, inexpensive and easy access communication among people all over the world. Thus, there has been an industry that uses e-mail messages as advertisement tool. Exactly most people don't want to receive these types of e-mails which are called unsolicited (bulk) mail or spam mail. So, it is necessary that the un-wanted mails should be filtered out by using some text-based methods.

Email spam filtering is a text categorization problem that classifies the user mails into binary categories as spam or non-spam.

1.2 Thesis Objective

In this thesis we seek to filter out spam messages from user's email by implementing an automated email spam filter that classifies mails as spam or non-spam (ham).

Our primary goal of this research was to survey existing methods and determine a method or a combination of methods that would work well for our Email spam filter. The secondary goal of this research was to create a tool to accurately and quickly categorize email messages as spam or non-spam at the local user level.

1.3 Outline of the Thesis

Chapter 2 of this thesis describes the Natural Language Processing problem, Text Categorization, overview some of TC applications and tasks, text categorization process and two main types of approaches (KE and ML) to TC problem.

Chapter 3 gives a brief description of the Email spam filtering as a binary text categorization problem; some background related to email filtering is also given. In addition to these, some methods from Machine Learning applied to Email spam filtering is described and given here.

Chapter 4 provides the description of the structure and components of the program. Architecture of our Email spam filter is also given in this part. Methodology is given here. The code structure of Email spam filter program is also given. A detailed description of the data (training and test) and corpus is also given.

Chapter 5 discusses the Email Filter System performance and the accuracy of its results. In this chapter the designed Email spam filter program's achievements and future work is reviewed.

CHAPTER 2

LITERATURE SURVEY

This chapter presents a survey of text categorization, including descriptions of some techniques etc.

2.1 Definition of Text Categorization

Text Categorization (TC) refers to the automatic labeling of documents, based on natural language text contained in or associated with each document, into one or more pre-defined categories.

Formal definition of text categorization can be given as follows:

Let $D = \{d_1, d_2, \dots, d_n\}$ a set of documents and $C = \{c_1, c_2, \dots, c_m\}$ be a set of categories. The task of the text categorization consists in assigning to each pair (d_j, c_i) of $D \times C$ (with $1 \leq i \leq m$ and $1 \leq j \leq n$) a Boolean value of 0 or 1, i.e. the value 0, if the document d_j doesn't belong to c_i .

2.2 Text Categorization Tasks

Some text categorization tasks assume the categories are independent, in which case each document can be assigned to no categories, one category, or multiple categories. In such cases, the categories are said to be binary categories, and a separate YES/NO decision is required for each category/document pair.

More formally, let D be the set of all documents and C be the set of all binary categories; a text categorization system then has to assign a Boolean value to each (d, c) pair, where $d \in D$ and $c \in C$.

Other text categorization tasks assume that the categories are mutually exclusive and exhaustive, in which case each document is assigned to exactly one category. Then the inclusion of a document in one category excludes the inclusion of the document in all other categories. In this case, the classifier must map each document $d \in D$ to the category $c \in C$ that is the best fit.

Certain tasks can be viewed as fitting into either of the two paradigms described in the previous two paragraphs. Text categorization tasks with exactly two mutually exclusive categories can be viewed as a binary categorization task involving either one of the categories, such that any document that doesn't belong to this category is automatically placed in the other category.

On the other hand, if the task is to categorize web sites as pornography or not pornography, the not pornography category is vague, consisting of all other types of websites, and it seems more natural to view this as a binary decision as to whether or not the website is pornographic.

Then there are cases that fall somewhere in between; for example, categorizing movie reviews as positive or negative can easily be thought of fitting into either paradigm. In any case, either type of text categorization system (i.e. those designed to work with mutually exclusive categories or those designed to work with binary categories) can be applied to these tasks.

Another special case of text categorization tasks occurs when mutually exclusive categories form a hierarchy, also known as taxonomy. One commonly

known taxonomy is the Yahoo directory [1] of web pages set up for easy browsing.

Some systems take such hierarchical structures into account by using a ladder approach. Instead of considering the bottom nodes of the hierarchical tree directly, a decision is first made at the top node of the tree as to which of its immediate children is appropriate, then at the second level, etc., until a terminal node is reached. The idea behind this is that each decision being made is simpler than an immediate decision involving all possible categories, especially when the taxonomy is large. Some researchers have found noteworthy improvement using this approach [2], [3], [4].

2.3 Applications of Text Categorization

Some of the current applications of text categorization include: the classification of e-mail as spam or not spam for filtering purposes [5], [6]; the classification of news articles into topical sections (e.g. Politics, Sports, Entertainment, etc.) for browsing [7], [8], [9]; classification of websites as pornography or not pornography for filtering purposes [10]; hierarchical classification of websites into a large variety of topics for browsing [11], [12]; extraction of certain types of metadata from text documents or websites to improve search capabilities [12]; the classification of reviews (e.g. reviews of movies or travel destinations) as positive or negative to summarize statistics [13], [14].

Many other common classification tasks could potentially be viewed as text categorization tasks. For example, word sense disambiguation refers to the automatic classification of words in a document with their correct, current sense. This can be viewed as a text categorization task where the categories would be the possible senses, which would be different for each possible word, and the text used

for the categorization would be the word's context (i.e. the text surrounding the word) [15]. Information retrieval [16] refers to the automatic discovery of documents in a corpus that match a user's query. This can be thought of as a text categorization task where the categories are relevant and not relevant (but the meanings of these categories would change with each new query). Topic detection and tracking (TDT) [17] refers to the detection of new stories that are similar to a set of example stories. This can be viewed as a text categorization task where the categories are similar and not similar (in this case, the meanings of these categories change for each set of stories).

2.4 Document Representations with Bag of Words Approach

Text categorization refers to the automatic labeling of documents based on text. In order to label documents, systems must first be given access to each document, and the document must be represented by the system in some way. Almost all modern text categorization systems represent documents using what is known as “bag of words” approaches. This means that each document is represented as a vector of weighted words, although exactly what constitutes a word can vary to some degree. Weights are generally computed by combining statistical features of words in some manner.

Let d be a document that needs representation, and T be the set of terms used by the representation. If a bag of words approach is being used, each term is a single word, and T is the set of all possible words. Let λ_t be the weight of a single term $t \in T$. Then d can be represented as $d = [\lambda_{t_1}, \lambda_{t_2}, \dots, \lambda_{t_{|T|}}]$. Bag of words approaches do not rely on syntax or semantics; in other words, the ordering of terms in a document does not matter, and the relationship between any two distinct terms is not considered. There are many possible weighting schemes, some of

which are discussed in Section 2.4.2, one thing that most have in common is that words that do not appear in a document are assigned a weight of zero. This turns out to be helpful, since most documents likely do not contain most of the possible words, and it is then only necessary to keep track of weights for words that do occur in each document.

2.4.1 Words Represented as Terms

The most of text categorization systems that is well known use single words as terms when representing documents. It is generally accepted in the information retrieval and text categorization literature that more complex representations do not lead to improved performance and are often less efficient [18], [19]. It might seem that using phrases such as bigrams (instances of two consecutive words) or trigrams (instances of three consecutive words) instead of, or in addition to, single words might be useful, since some context would be accounted for; however, for most bigrams and trigrams, there is scarce evidence, since the number of existing instances of bigrams and trigrams in a corpus is miniscule compared to the possible number of bigrams and trigrams in a language (the square and cube of the number of words in the language, respectively).

Although many approaches described here so far, use single words as terms, there are still several decisions to be made as to what exactly constitutes a word, and how to distinguish words automatically. Most systems do not have a complete dictionary of allowable words, and so they must use general rules to distinguish words from non-words.

Even after the general rule for determining word boundaries is decided, there are still many options that systems have in determining which, if any,

transformations must apply to each word. For example, case sensitivity determines whether or not two words that differ only in terms of capitalization should be treated as equal. In other words, if two words are the same except that one has one or more letters capitalized whereas the other has the same letters in lower case, should they be considered two instances of the same word or two separate words? At times, capitalization can mean the difference between a common word and a proper noun, but at other times, it can be the result of placement at the beginning of a sentence.

Another issue is stemming, a simple rule-based technique that converts different forms of a word to the same root token, or stem. For example, different tenses of a verb such as “help”, “helps”, “helped”, or “helping” are all converted to the same stem “help”, whereas different forms of a noun such as “house” or “houses” are all converted to the same stem “house”. However, common stemming algorithms (e.g. Porter stemming (Porter, 1980)) do not handle irregular verbs or unusual nouns correctly. At times, different forms of the same root fail to be converted to the same token (e.g. “child” stays as “child” and “children” stays as “children”), while, at times, forms of different words do get converted to the same token (e.g. “tire” and “tired” both get converted to “tire”). More accurate than stemming is to use a lexical database such as WordNet [20] to convert every word to its morphological base-word. However, this is also much more time consuming than stemming, and it is still not guaranteed to improve performance, as for certain text categorization tasks, the specific tense or form of a verb or noun may give a clue as to the appropriate category (for example explored in [21]), in which distinctions between singular and plural nouns or passive versus active tenses for verbs made a significant difference for her categorization task).

Another issue is whether to use all words for text categorization or to filter some out. Many current systems use a stop-word list, a hard coded list of common

words such as articles and prepositions that are ignored when they occur. Although these words would be given very low weights by machine learning methods (some are described later in this chapter), there tend to be many of them in a document, and so the noise introduced by these words can add up and make results less accurate. Another possibility is to automatically determine which words to filter out based on weights; for example, the next subsection of this chapter describes the inverse document frequency (*IDF*) weighting scheme, and this can be used to exclude all words with *IDF* values under some specific constant (this is explored in [22]). However, one must be careful when deciding whether or not to filter out such words, as they can be helpful for certain text categorization tasks. An excellent example is the breakthrough work of Mosteller and Wallace [23], the application, in which the authors determine that filler words such as “an”, “of”, and “upon” are very important for an authorship attribution task, whereas more meaningful content words are not.

Even when two instances of two words are spelled the same way with the same case sensitivity, they may not really represent the same word with the same meaning. For example, “can” is sometimes a noun (as in “I ate a can of beans.”) and sometimes a modal verb (as in “You can do it!”); “wind” is sometimes a noun (as in “There's a strong wind out today.”) and sometimes a verb (as in “I need to wind my watch.”). Such pairs of distinct words that are spelled the same are known as homographs, and ideally, a system should probably distinguish one sense from another, although few text categorization systems even try. One way that systems could differentiate homographs of each other when the two words represent different parts is to use a part-of-speech tagger (POS); the part-of-speech tag assigned to each word by the tagger can be included as part of the token (this is explored in [22]).

2.4.2 Common Methods for Weighting of Words

Using the techniques described in the previous sub-section a system can process a text document and determine which words are present. At that point, there are many possible weighting schemes that can be used to set up the vector representing the document. The simplest possible measure is a binary value for each word; 1 if word is present and 0 otherwise.

More typically, systems use more complex weights. One approach that is still somewhat simple is to weight each word in a document with the word's term frequency (*TF*) [24], [25]; which is the number of times that the word appears in the document. All other things being equal, it is expected that words that appear more often in a document are more important as to the overall meaning of the document. Consider, for example, a document that mentions Turkey once, compared to a document of equal length that mentions Turkey ten times; you would probably expect the second document to have more of a focus on that topic.

Of course, words such as “the” will likely appear many times in a document without being considered important for most text categorization tasks. Usually, a system combines term frequency with some other measure to take this into account. The most common measure that is often combined with term frequency is inverse document frequency (*IDF*) [24], [25]; which can be thought of as a measure of a word's rarity in a corpus (or a language, but this needs to be estimated based on a corpus). It is believed that words that are rare tend to be more specific, and may therefore contribute more to content. Let $DF(w)$ be a word's document frequency (*DF*), which is the number, documents out of a set of N total documents that contain one or more instances of the word. Then the *IDF* of the word can be computed as follows (Formula 2.1):

$$IDF(w) = -\log_2 \frac{DF(w)}{N} = \log_2 \frac{N}{DF(w)} \quad (2.1)$$

Therefore, very common words that appear in almost all documents have an *IDF* close to zero, whereas rare words have larger *IDFs*, with a maximum possible value of $\log_2 N$ for a word that only occurs in one document. (Words that are never seen in the N documents have an undefined *IDF* and, thus, are generally ignored; this makes sense if the N documents are those of the training set, since there is then no evidence indicating one category versus another for these words.) The specific logarithmic base being used is actually not important so long as it is consistent.

The most common method of combining a word's term frequency with its inverse document frequency is to simply multiply the two weights together:

$$TF * IDF(w) = TF(w, d) \times IDF(w) \quad (2.2)$$

This *TF*IDF* representation effectively combines a word's importance to a document with its specificity over a corpus. At times, you might see more complex combinations of the same measures or similar measures, but ever since the very influential work of Salton and Buckley in the late 1980's [24], [25]; *TF*IDF* has dominated weighting schemes in the text categorization and information retrieval literature.

2.5 Text Categorization Process

Definition of the “text categorizer” can be given as follows:

Given a set of categories $C = \{c_1, \dots, c_{|C|}\}$ and a set of previously unseen documents $D = \{d_1, d_2, \dots\}$, a categorizer is a function K that maps from D to the set of all subsets of C .

The standard modern approach to creating new categorizer functions is to build them using Machine Learning techniques (it is reviewed in the next section) from a set of training documents. This is a set of user-provided, pre-labeled documents that follows a category distribution similar to the distribution of D , and whose contents provide information about what sorts of documents should be mapped to what sorts of categories. Algorithms can then be developed that make generalizations about the relationship between document content and document category, encoding these generalizations in the learned function K . (see Figure 1.)

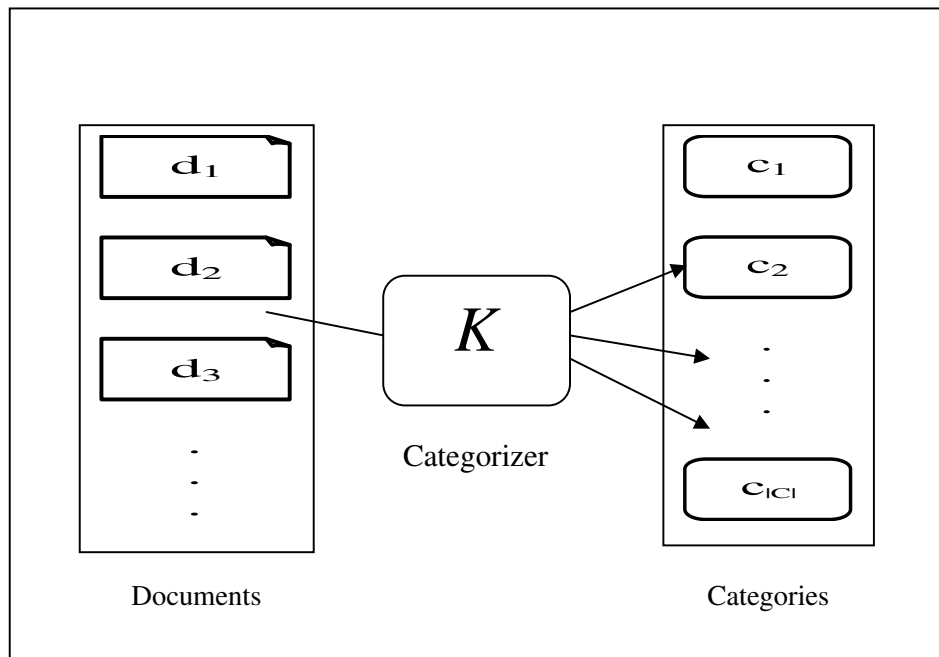


Figure 2.1. Action of a categorizer on a set of documents

In order to train a categorizer in the above manner, the user must begin with a “training corpus” or “training set”. This is a set of documents which are pre-labeled with categories that are considered to be fully correct - typically, they have been manually assigned by a domain expert, i.e. a person who is familiar with the type of material contained in the documents and who knows how to assign a set of categories to each document based on the documents’ content.

The basic outline for creating Text Categorization (TC) applications are relatively simple: the documents in “training set” are presented to the TC system, the system processes the documents’ content, and a specific categorization function K is produced that may be used to categorize future documents from the set D . In an application, however, many details of this process need to be managed in specific and often varying ways.

In addition to the above process, the designer of a TC system may wish to evaluate the trained system on a set of data previously unseen by the system, for which the desired results of categorization are known. Such a data set is commonly called a “test set”.

2.6 Approaches for Text Categorization

2.6.1 Knowledge Engineering Approach

In the 1980s, the most popular approach for the creation of automatic document classifiers consisted in manually building, by means of Knowledge Engineering (KE) techniques, an expert system capable of taking Text Categorization decisions. Such an expert system typically consists of a set of manually defined logical rules, one per category.

The most famous example of such a rule based approach is the Construe System [26]; assigns zero or more categories to stories for a Reuters news database. It was developed by the Carnegie Group and went into production in 1989, applying 674 distinct categories to a newswire feed, as well as recognizing over 17000 company names. A sample rule for the category AUSTRALIAN DOLLAR that is used in Construe project is shown in Figure 2.2.

```
(if
test:
  (or [australian-dollar-concept]
and [dollar-concept]
[australia-concept]
(not [us-dollar-concept])
(not [singapore-dollar-concept])))
action: (assign australian-dollar-category))
```

Figure 2.2. Sample hand-crafted categorization rule

The drawback of this approach is the knowledge acquisition bottleneck well-known from the expert systems literature. That is, the rules must be manually defined by a knowledge engineer with the aid of a domain expert (in this case, an expert in the membership of documents in the chosen set of categories): if the set of categories is updated, then these two professionals must intervene again, and if the classifier is ported to a completely different domain (i.e. set of categories) a different domain expert needs to intervene and the work has to be repeated from scratch.

On the other hand, it was originally suggested that this approach can give very good effectiveness results: Hayes et al. [26], reported a .90 “breakeven” result on a subset of the Reuters test collection, a figure that outperforms even the best classifiers built in the late 1990s by state-of-the-art ML techniques. However, no other classifier has been tested on the same dataset as Construe, and it is not clear whether this was a randomly chosen or a favorable subset of the entire Reuters

collection. As argued in Yang (1999), [27]; the results above do not allow us to state that these effectiveness results may be obtained in general.

It can readily be appreciated that the handcrafting of such rule sets is a non-trivial undertaking for any significant number of categories. The Construe project ran for about 2 years, with 2.5 person-years going into rule development for the 674 categories. The total effort on the project prior to delivery to Reuters was about 6.5 person-years.

Thus there is a powerful incentive to investigate automatic methods for text categorization. These are arising from fully automatic statistical methods that function as “black boxes” and require no human intervention, to programs that generate legible rules automatically, for subsequent editorial review. The next subsection provides an overview of these methods.

2.6.2 Machine Learning Approach

In the early 1990s, the Machine Learning (ML) approach to Text Categorization has gained popularity and has eventually become the dominant one, at least in the research community [28]. In this approach, a general inductive process (also called the learner) automatically builds a classifier for a category c_i by observing the characteristics of a set of documents manually classified under c_i or \bar{c}_i by a domain expert; from these characteristics, the inductive process gleans the characteristics that a new unseen document should have in order to be classified under c_i . In ML terminology, the classification problem is an activity of supervised learning, since the learning process is “supervised” by the knowledge of the categories and of the training instances that belong to them [19].

The advantages of the ML approach over the KE approach are evident. The engineering effort goes toward the construction not of a classifier, but of an automatic builder of classifiers (the learner). This means that if a learner is (as it often is) available off-the-shelf, all that is needed is the inductive, automatic construction of a classifier from a set of manually classified documents. The same happens if a classifier already exists and the original set of categories is updated, or if the classifier is ported to a completely different domain. In the ML approach, the pre-classified documents are then the key resource.

In the most favorable case, they are already available; this typically happens for organizations which have previously carried out the same categorization activity manually and decide to automate the process. The less favorable case is when no manually classified documents are available; this typically happens for organizations that start a categorization activity and opt for an automated modality straightaway. The ML approach is more convenient than the KE approach also in this latter case. In fact, it is easier to manually classify a set of documents than to build and tune a set of rules, since it is easier to characterize a concept extensionally (i.e., to select instances of it) than intensionally (i.e., to describe the concept in words, or to describe a procedure for recognizing its instances). Classifiers built by means of ML techniques nowadays achieve impressive levels of effectiveness, making automatic classification a qualitatively (and not only economically) viable alternative to manual classification.

Over the years, many machine learning approaches have been developed for automatic text categorization. In this section we will describe and focus on some of more common machine learning methods which have been used for building text classifiers:

2.6.2.1 Naïve Bayes Classifiers

Suppose that you have a feed of incoming documents. You have been manually assigning each such document to a single category for some time. Thus, for each category, you have a reasonable number of past documents already assigned.

Bayes' Rule

One approach to automating (or semi-automating) this process is to build statistical models of the categories you are assigning to, leveraging the assignments that you have already made. This approach assumes that you can compute, or estimate, the distribution of terms (words, bigrams, phrases, etc.) within the documents assigned to these categories. The idea is to use this term distribution to predict the class of unseen documents, but this only works under certain conditions, which we shall present, in a somewhat simplified form. Firstly, you need to be able to transform the probability of a term occurrence given a category (which you can estimate directly from your data) into the probability of a category given a term occurrence. Secondly, you need a method to combine the evidence derived from each of the terms associated with a document or category.

In other words, you know $P(t | C_i)$, for each term t and category C_i , but you are really interested in $P(C_i | t)$ or better yet $P(C_i | T_D)$, where T_D is the set of terms occurring in document D . In the following, we make no more distinction between document D and its representation as a set of terms, T_D .

The term ‘Naïve Bayes’ refers to a statistical approach to language modeling that uses Bayes’ Rule but assumes conditional independence between features (term occurrences). We thus compute the probability that document D belongs to a given class C_i by:

$$P(C_i | D) = \frac{P(D | C_i)P(C_i)}{P(D)} \quad (\text{Bayes' Theorem}) \quad (2.3)$$

In the most common form of Naïve Bayes, we assume that the probability that a document belongs to a given class is a function of the observed frequency with which terms occurring in that document also occur in other documents known to be members of that class.

In other words, ‘old’ documents known to be in the class suggest both:

1. Terms to look for, and
2. The term frequencies one would expect to see in ‘new’ documents.

The ‘old’ documents function as training or conditioning data, providing probability estimates upon which a statistical argument for classification of unseen data can be built.

Ignoring conditional dependencies between terms, we can use the multiplication rule to combine such probabilities. More formally, given a document, D , represented by a term vector consisting of n components or terms, $D = (t_1, \dots, t_n)$, and a class, C_i , from the range of target classes, the formula

$$P(D | C_i) = \prod_{j=1}^{j=n} P(t_j | C_i) \quad (\text{Naïve Bayes Assumption}) \quad (2.4)$$

captures the assumption that the probability of a term vector being generated by a document of a given class can be decomposed into a simple combination of the distribution of the terms within that class.

Before we can apply Bayes' Rule, we also need to estimate the prior probability of a particular class being any document's destination. Suppose we had no information regarding the terms in a document, and had to make a blind guess as to where it should be classified. Clearly, we would maximize our chances of success if we assigned it to the most popular class, according to our training data. The most direct way to estimate the prior for a given category is simply to count the number of training documents occurring in that category and divide by the total number of categories.

Given a value for $P(C_i | D)$, how do we decide whether the document belongs in the class or not? Given M classes, one approach is to compute $P(C_i | D)$ for all i such that $1 \leq i \leq M$, and then assign the document to the class that scores best. We can express this tersely by the formula below

$$C^* = \arg \max_{C_i} [P(C_i | D)] \quad (2.5)$$

where C^* is the favored class, and $\arg \max_y [f(y)]$ selects the value of subscript argument, y , that maximizes the function of y that follows in brackets. Thus we look for a category, C_i , that maximizes the value of $P(C_i | D)$. By Bayes' Rule,

$$\arg \max_{C_i} [P(C_i | D)] = \arg \max_{C_i} [P(D | C_i) \cdot P(C_i)], \quad (2.6)$$

enabling us to plug in the probability estimates discussed above. We can omit $P(D)$ from the right hand side of this equation, since it is an invariant across classes, and will therefore have no effect upon which category is selected.

Naïve Bayes algorithms for learning and classifying text [28], are shown in Figure 2.3.

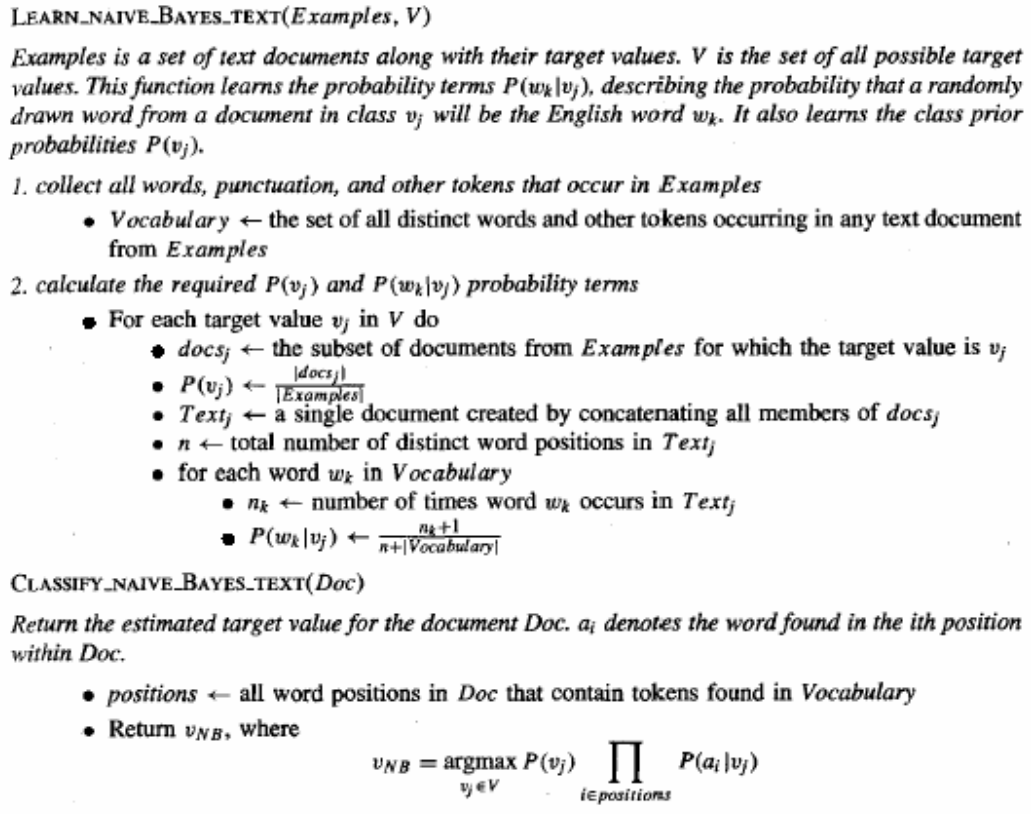


Figure 2.3. Naïve Bayes algorithms for learning and classifying text

2.6.2.2 Decision Trees

Naïve Bayes approach, model documents using a relatively large, fixed set of features, typically represented as vectors. Naïve Bayes looks at the distribution of terms, either with respect to their frequency or with respect to their presence or absence.

A quite different approach is to construct a tree that incorporates just those feature tests needed to discriminate between objects of different classes. The unique root can be thought of as representing the universe of all objects to be categorized. A non-terminal node of the tree is a decision point that tests a feature and chooses a branch that corresponds to the value of the result. A classification decision is then a sequence of such tests terminating in the assignment of a category corresponding to a leaf node of the tree. Leaf nodes represent the categories non-uniquely, i.e., there may be more than one leaf node with the same category label, with the path from the root to that leaf representing a distinct sequence of tests. It turns out that such trees can be formed by an inductive learning technique, based on a training set of pre classified documents and their features.

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

The decision tree method characterizes a data object, such as a document, in terms of a logical combination of features, which is simply a statement about that object's attributes, and does not involve any numeric computation. In text categorization applications, these features are most likely to be stemmed words. This is quite different from representing a document as a vector of weighted features, and then performing a numeric computation to see if some combination of feature weights meets a threshold. Consequently, decision tree classifiers do not

have to learn such thresholds, or other parameter values. What they learn is essentially a set of rules defined over a space of keywords.

A typical training algorithm for constructing decision trees (CDT) can be sketched as the following recursive function at Figure 2.4.

```
CDT(Node, Cases)
if Node contains no Cases, then halt,
else if the Cases at Node are all of the same class, then the decision tree for
Node is a leaf identifying that class,
    else if Node contains Cases belonging to a mixture of classes,
    then choose a test and partition Cases into subsets based on the outcome,
        creating as many Subnodes below Node as there are subsets,
        and call CDT on each Subnode and its subset of Cases,
    else halt.
```

Figure 2.4. A training algorithm for constructing decision trees

The main issue in the implementation of such an algorithm is how the program chooses the feature test that partitions the cases. Different systems have used different criteria, e.g., the ID3 decision tree program uses a measure of information gain, selecting the most ‘informative’ test [29]. The test that gains the most information is simply the test that most reduces the classification uncertainty associated with the current set of cases. Uncertainty is maximal when classes are evenly represented across the current set of cases and minimal when the cases are all of the same class.

There are a number of standard packages for Decision Tree (DT) learning, and most DT approaches to TC have made use of one such package. Among the most popular ones are ID3 used by [30]; C4.5 used by [31], [32] and [33].

One basic algorithm, ID3 learns decision trees by constructing them top down, beginning with the question "which attribute should be tested at the root of the tree?" To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree.

A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute). The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.

2.6.2.3 Nearest Neighbor Algorithms

Naïve Bayes classifiers learn through induction: they build an explicit model of the class by examining training data. The same can be said of decision tree methods such as C4.5. However, there is another kind of classifier that does not learn in this way.

'Nearest Neighbor' classifiers rely on rote learning. At training time, a Nearest Neighbor classifier 'memorizes' all the documents in the training set and their associated features. Later, when classifying a new document, D , the classifier first selects the k documents in the training set that are closest to D , then picks one or more categories to assign to D , based on the categories assigned to the selected k documents.

The K-Nearest Neighbor (kNN) approach towards text categorization is an example of an instance-based learning method (also known as an example-based method, a memory-based method, and even a lazy learning method) [19], [28].

The training for such methods basically just consists of recording which training examples fall in each possible category. Of course, reading in the training documents also requires a system to convert them to the appropriate bag of words representation, assuming the system does not require this representation as input in the first place. When future documents arrive (or a test set is evaluated), the new documents are compared to those in the training set.

CHAPTER 3

A BINARY TEXT CATEGORIZATION PROBLEM: EMAIL SPAM FILTERING

3.1. Introduction

Today, email is one of the most popular communication mediums because of its low cost and speed of transmission. People use email daily for personal needs and for business. These factors give an opportunity for marketers to advertise their wares to people via email what is referred as 'spam' or 'junk-mail' like as 'win a holiday trip', 'you won prize', or even 'adult' content.

Why Spam or unsolicited mail is undesirable?

- Such emails annoy web users and Internet Service Providers since it requires a lot of time to deal with such types of emails.
- It takes a lot of space in the recipient's inbox.
- Spam mails lead to the wastage of Internet bandwidth.

So there is a need for effective tools to aid users to identify spam messages. These are known as Anti - Spam Filters, whose functionality can be in the simplest form as detecting spams based on email message headers, and more complex techniques such as content-based filters which use keywords.

Text filtering is a specific case of binary text categorization in which the categories are relevant versus not relevant (or useful versus not useful). Unlike in information retrieval the meanings of these categories do not change based on

queries or user profiles, so text filtering, the way which we talk here is clearly a subset of binary text categorization.

The task of spam (or junk mail) filtering is to rule out unsolicited junk e-mail (spam) automatically from a user's mail stream.

3.2. Background of the Study

Today most Content-Based filters search for some keyword patterns within the message. These patterns, however, need to be built and tuned to achieve better results, require skills that many users may not have. Yan [34]; proposed a simple content-based filtering system that used an Internet browser to filter News articles. However, the user queries had to be built manually and updated by the user. Keyword-based rules, very similar to the former, are used in current systems such as ProcMail and are interested with the body of the email. The principle of this technique is that given a rule set, if all the keywords in a rule are found, the conclusion is drawn. However, this technique requires experts which can allow for easy mistakes to be made leading to awful consequences. Still, a user may change their criterion over time, so there is a need for more rewriting. Therefore, automation is required.

Automated and semi-automated filtering solutions are widely used by individuals, email administrators and Internet Service Providers. Much research has been conducted in this area with extensive use of techniques from the Machine Learning paradigm. Cranor and LaMacchia [35]; define automated filters as tools that find and delete all suspected spam, while semi-automated versions simply relocate spam for a human to examine at a later date. Microsoft Outlook enables filtering on the message headers in addition to providing the use of stoplists and the recognition of commonly used phrases.

Clearswift MAIL sweeper analyses incoming and outgoing email at the Internet Gateway stage, allowing its users to block inappropriate emails, complies with legal requirements, and increase user productivity. Its pattern matcher uses a characteristic data sequence to determine the type of information contained.

It is believed that learning – based classification approaches enable filters to be more automated, effective and user-friendly. For this reason, most research into the area has concentrated on this method of classification. In fact, some Text Categorization approaches have shown their prospects in the field of email filtering. ‘Automatic Text Categorization is a supervised learning task, defined as assigning category labels (pre-defined) to new documents based on the likelihood suggested by a training set of labeled documents’ [36]. Automatic Text categorization is now being applied in many contexts including document indexing, document filtering, personalized information delivery, word sense disambiguation, web page categorization, genre detection, or even authorship detection. ‘Until the late 1980s, the dominant approach involved Knowledge Engineering automatic categorizers - manually defining a set of rules encoding expert knowledge on how to classify documents under the given categories’ [19]. In the 1990s, Machine Learning replaced the previous approach and the success of these techniques in Text Categorization has recently led to alternative solutions. This is due to the fact that email filtering can be viewed as an instance of the Text Categorization problem, with only two classes existing, spam and legitimate.

Current research in the field of email filtering has concentrated on more complicated approaches from the Text Categorization and Machine Learning paradigms. Because of the complexity of these approaches, a different path to follow is that of more novel solutions to the problem. These include researching more simplistic techniques like the keyword-based method already discussed, but

the main effort is to define new ways of executing the same task, attempting to find a better or more effective solution. The novel solution proposed is to use the methodology of Information Retrieval systems presently used by Internet Search Engines. This technique is widely used by the general public, gives reasonably reliable results and is applicable to the same data, textual information, as emails.

3.3. Machine Learning Applied to Spam Filtering

When Machine Learning techniques applied to E-mail Spam Filtering is concerned, the following methods are generally used:

1. **Bayesian Learning:** Bayesian Spam Classifiers compute the probability of an email being spam or not by building a database of tokens (words or content of mails in the learning set of both spam and non-spam mails) and their respective probabilities of having been seen in spam and non-spam mails. When testing on a specific mail, we calculate the probability of that mail being spam or not given the known probabilities of the tokens found in the mail for spam and non-spam cases.
2. **Neural Networks:** Spam Filters utilizing artificial neural networks may classify incoming mails as spam or non-spam using pre-defined attributes, such as frequency of specific words like "money", number of capital letters in a row, presence of advertising tags in the email subject and others. The input size would thus be one greater than the number of pre-defined attributes (counting bias), the number of hidden layer neurons can be fixed using trial-and-error and a single output neuron (that outputs true to indicate spam and false to indicate non-spam mails).

3. **NLP – based Filters:** Such filters use Natural Language Processing to understand the meaning and semantics (relationships among words) of the natural language used in the mails. This would lead to more informed classification decisions because we work at a higher level of abstraction than the Bayesian case where we just count the frequency of tokens (words) occurring in the mails, independent of their context. However, this technique is difficult to implement practically.

4. **Hybrid Approach:** A hybrid approach to spam filtering employs more than one technique. We could have a layer at which we use white lists and blacklists to immediately classify incoming mail as one sort or the other. The mail itself can be parsed separately for the headers and the body and different tests applied in each case. There should also be scope for the end-user to resend specific mails to the classifier which was erroneously classified (especially in the case of False Positives) so as to retrain the database.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

4.1. Description of the Problem Statement

The aim of this thesis is to design and implement an E-mail Spam Filtering Tool, which takes a set of e-mails as input and categorizes each one as spam or non-spam mail. At the same time, it is important that filter should not erroneously classify valid mail as spam because this would cause much more serious problems for end-user than having to handle some spam mail by himself.

4.2. E-mail Spam Filter Tool Design

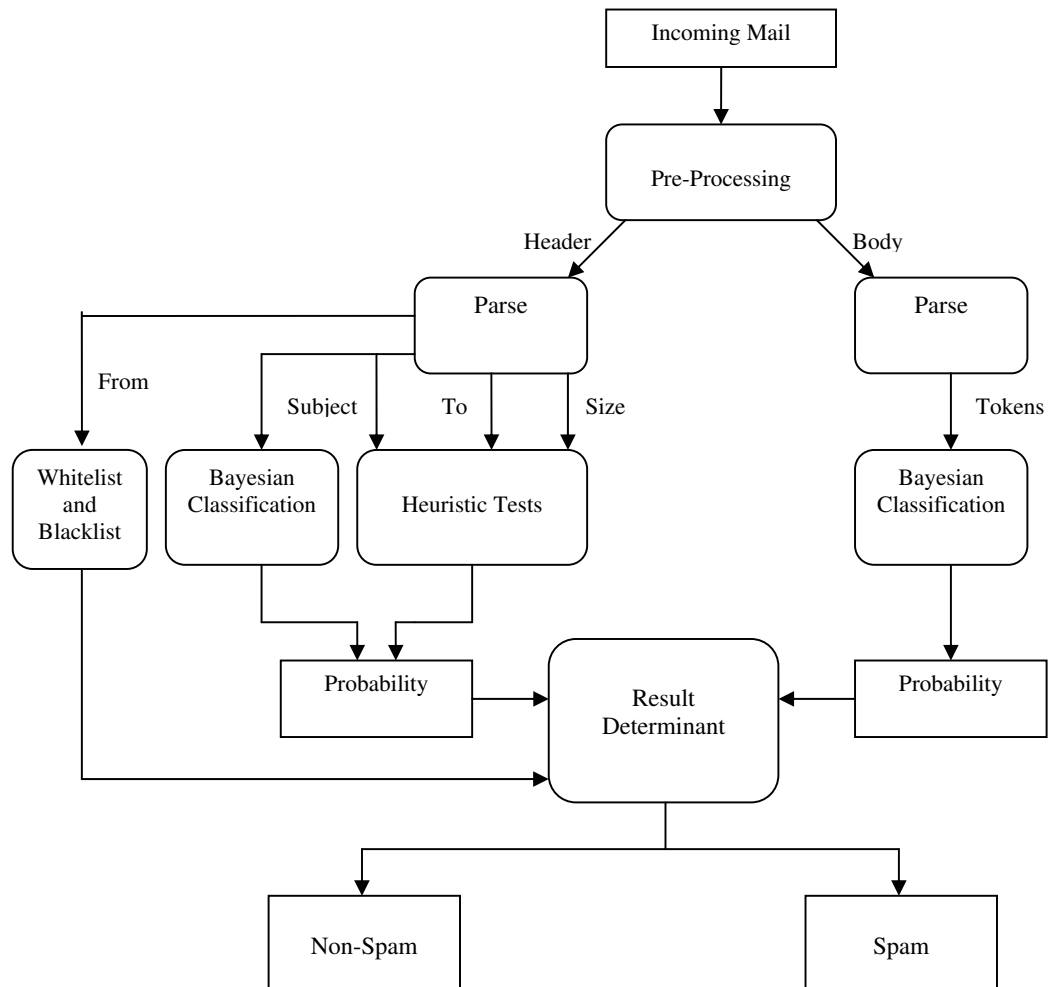


Figure 4.1. Architecture of the Email spam filter

The diagram above (Figure 4.1) shows the architecture of our Email Spam Filter. As we explained below we used a hybrid approach of pure Bayesian classification and we also use many heuristic tests and features such as whitelist and blacklist to classify emails.

The classification of emails is done as follows:

- Incoming mails are first pre-processed to separate the header and the body part of the email.
- The header is separately parsed to extract fields such as “From”, “To”, “Subject” and calculated mail size.
- The sender that is known "From" field is checked against the whitelist and blacklist configured by the user. This would directly classify the mail as spam or non-spam.
- Heuristic tests (detailed in Chapter 4.3) are applied on the "To", "Subject" and “mail size” attributes that assign certain probabilities to the mail of it being a spam instance.
- A Bayesian classification is separately run on the "Subject" field to investigate if it contains any words that are common to spam mails. We finally end up with a probability of the mail being spam based on only header analysis.
- Words or tokens are extracted from the body and used to compute the probability of the mail being a spam using the database previously learnt during training phase using the Bayesian technique.
- The two probabilities are combined and used to finally classify the mail is either spam or non-spam.

4.3. Methodology

There are various ways which spam filter may be applied to an e-mail network. One type is Server Side Filtering in which the spam filter may be integrated with SMTP servers which would eliminate spam on the Internet itself and save the wastage of bandwidth. Meanwhile, the problem with this approach is

that it would require the classifier to be general enough to process the mails of all users. The other one that we will use in our project is User Side Filtering Model, which would be placed between POP server and user's inbox. This may seem to cause wastage of Internet bandwidth, because spam is filtered out only at its destination. However, User Side model involves more Machine Learning techniques and would give a better accuracy. Thus our spam filter architecture assumes that filter is placed at as in the User Side Filtering.

In this project, we will use the Hybrid Approach (described in Chapter 3.3) for filtering spam mails. The designed filter implements a Bayesian Classifier to analyze the body of the mails and also parses the headers separately and applies various heuristic tests on the subject. Heuristic tests can be such as checking if it is a reply, checking if the subject is all capitals or contains certain words that would indicate that the mail is spam, by applying a Bayesian classifier on the subject.

The filter we designed uses the Bayesian approach to classify spam and non-spam mails. We use also some novel ideas as follows:

1. On the "header" part of emails heuristic tests are used that assign a separate spam probability to the mail.

The heuristic tests we have applied on the header are as follows

- Mail size: A mail of size greater than 10 KB is likely to be spam.
- "Subject" field: If found to be empty, the mail is most probably legitimate (non-spam).
- "Subject" field: If starts with "Re:" the mail is likely to be legitimate.
- "Subject" field: If found to be all upper case, the mail is likely to be spam.

- "Subject" field: A Bayesian classification is run on the subject's words to find out if it contains words frequently occurring in spam mail subjects (such as advertising tags).
- "From" field: The sender of the email is matched against the configured whitelist and blacklist.

It must be noted that many more such tests can easily be added to our implementation, we have kept only those that were found to be particularly useful for our test corpus

2. On the “subject” field of emails Bayesian classifier is applied separately.

We used a corpus consisting of about thousands of legitimate and spam mails each to train the Bayesian classifier and create the “database of tokens” along with the number of times they occur in legitimate and spam mails. During testing, the probabilities for the most frequently occurring words in the given mail of occurring in legitimate and spam mails are computed and used to calculate the overall probability of the mail being spam or not. Specifically, we are getting some ideas from the approach described by Paul Graham [37].

The probability of occurrence of a word given that the mail is spam is calculated using following formula (4.1):

$$\text{Required Probability} = ((a/\text{totalWordsInSpam}) / ((a/\text{totalWordsInSpam}) + (b/\text{totalWordsInLegitimate}))) \quad (4.1)$$

where,

a = count of occurrence of the given word in spam mails used to learn the database

and

b = count of occurrence of the given word in legitimate mails used to learn the database

For example, if the word “mortgage” occurs in 400 of 3000 spam mails and in 5 out of 300 legitimate emails, then its spam probability would be equal to:
 $((400/3000) / ((5/300)+(400/3000))) = 0.8889$

The probability that a given mail is spam from its 'n' most frequent tokens is calculated using following formula (4.2):

$$\text{Overall Probability} = ((t_1 * t_2 * \dots * t_n) / ((t_1 * t_2 * \dots * t_n) + ((1 - t_1) * (1 - t_2) * \dots * (1 - t_n)))) \quad (4.2)$$

where,

't1', 't2' ... 'tn' are probabilities of the 'n' most frequent tokens in the mail occurring in spam calculated from the previous formula (4.1).

How the actual spam filtering is done?

Once the spam and ham (legitimate) databases have been created, the word probabilities can be calculated (by using Formula 4.1) and the filter is ready to use.

When a new mail arrives, it is broken down into words and the most frequent words are singled out. From these words, the Bayesian filter calculates the probability of the new message being spam or not (by using Formula 4.2). If the *overall probability* is greater than a threshold, say 0.9, then the message is classified as spam.

4.4. E-mail Spam Filter Tool Implementation

The Email spam filtering tool codes are written in Java using JDK 1.5.0 release.

We can define the code structure of our Email Spam Filtering program with Classes and their Methods as following:

```
Class SpamFilter
{
    Classifier classifier;

    public static void main()
    {
        // Main function
    }
}

Class Classifier
{
    MailReader train; // Object to perform training
    MailReader test; // Object to perform testing

    public void Create()
    {
        // Creates training database and loads in train(MailReader)
    }

    public void Save()
    {
        // Saves training database in disk file
    }

    public void Load()
    {
        // Loads training database from disk file
    }

    public void Test(string filePath)
    {
        // Tests corpus of test mails against the training the training database (It
        uses function spam)
    }
}
```

```

    public boolean Spam()
    {
        // Checks whether the given mail is spam or non-spam(ham) (It uses
function Parseheader and Parsebody
    }

    public float Parseheader()
    {
        // Parses the header
    }

    public float Parsebody()
    {
        // Parses the body
    }

    public float subjectAnalysis(String subject)
    {
        // (Used by Parseheader) It analyses mail subject and returns a
probability according to various checks

    }

    public float spamCharacteristics()
    {
        // (Used by Parseheader) Other heuristics are used here
    }
}

```

Class MailReader

```

{
    MailHeader mailHeader;
    Hashtable probTable;

    Public void Hash(String filePath1, String filePath2)
    {
        // Creates DataBase of a corpus of emails and saves in a hashtable
    }
    private String LexAnalyzer(String msg)
    {
        // This function helps in parsing of mails and defines the rules
according to which the tokens are extracted from mails.
    }

    private boolean isSeperator(char tempChar)
    {
        // defines all characters to be used as token seperator
// eg. ':', ';', ',', ':', '\', '\'', '/', '\\', '&', '#', '@', '<', '>', '%', '=', '-', '_', '$',
    }
}

```

```

private boolean isSpecial(char tempChar)
{
    // defines all character to be neglected
    // eg. '(', ')', '?', '!', '*', ']', '['
}

public void CalcProb()
{
    //Calculates the probability for each token in the database table
}

public void LoadList (String whiteListf, String blackListf)
{
    // Loads whiteList and blackList
}
}

Class MailHeader
{
    // Various header items
}

```

4.5. Training and Test Sets

After designing and implementing our system, it is important that we must use some data sets to train and test our system. We used several corpuses of legitimate and spam mails obtained from a variety of sources - the Internet, personal emails and public websites that have many spam mail databases such as the Spam Assassin corpus [38] and most commonly Ling-Spam corpus [39].

The training sets are taken from these corpuses which include both ham and spam mails to train the classifier.

For the user to gain confidence the system must work and hence testing is also a necessity. Without these constraints, no thorough evaluation would be possible. The results of the system with respect to given test data, are discussed on the Chapter 5 of this thesis.

CHAPTER 5

RESULTS AND CONCLUSIONS

5.1. Overview of Evaluating Text Classifiers

In order to evaluate the performance of our system with some test data, we used some metrics applied to the evaluation of text classifiers.

The performance of classification systems is frequently evaluated in terms of effectiveness. Effectiveness metrics for a binary classifier rely on a 2x2 contingency table shown on Table1. TP_i denotes ‘true positives’, FP_i denotes ‘false positives’, FN_i denotes ‘false negatives’, and TN_i denotes ‘true negatives’ [40].

Table 5.1 Contingency Table for evaluating a binary classifier

| Category c_i | Expert assigns YES | Expert assigns NO | Total |
|------------------------|--------------------|-------------------|-----------|
| Classifier assigns YES | TP_i | FP_i | m_i |
| Classifier assigns NO | FN_i | TN_i | $N - m_i$ |
| Total | n_i | $N - n_i$ | N |

Recall and precision have been adapted to text classification. Precision is the proportion of documents for which the classifier correctly assigned category c_i and is given by:

$$P_i = \frac{TP_i}{m_i} \quad (5.1)$$

Recall is the proportion of target document correctly classified and it is given by:

$$R_i = \frac{TP_i}{n_i} \quad (5.2)$$

Since text classifiers can be constructed using machine learning techniques, machine learning criteria such as the accuracy of the classifier, have also been used to measure effectiveness. Accuracy is the proportion of correctly classified objects, which is given by:

$$Acc_i = \frac{TP_i + TN_i}{N} \quad (5.3)$$

where N is the total and equal to $TP_i + TN_i + FP_i + FN_i$.

5.2. Email Filtering System Evaluation

For a through evaluation of Email filtering system, a Machine Learning style evaluation were used which is described in Chapter 5.1.

An ideal email filter would offer higher *Accuracy* and *Spam Precision*. Spam Precision is of greatest concern to mail users as they would not want their legitimate mail discarded as junk (Sahami et al) [5]. It is also believed that Legitimate Recall is so important that the higher the value, the less legitimate messages are misclassified; this is what users require.

Let's define the metrics we will use for evaluating our System:
According to contingency table (Table 5.1) we can define following equations of the effectiveness measures:

$$Spam Precision_i = \frac{TP_i}{TP_i + FP_i} \quad (5.4), \quad Spam Recall_i = \frac{TP_i}{TP_i + FN_i} \quad (5.5)$$

$$Legitimate Precision_i = \frac{TN_i}{TN_i + FN_i} \quad (5.6), \quad Legitimate Recall_i = \frac{TN_i}{TN_i + FP_i} \quad (5.7)$$

$$Overall Accuracy_i = \frac{TP_i + TN_i}{TP_i + FP_i + TN_i + FN_i} \quad (5.8)$$

5.3. Experimental Results

The purpose is to present and analyze the experimental results produced by our Email filter in an attempt to evaluate the system performance when the system is trained and tested with two commonly used corpuses known as Spam Assassin [38], and Ling-Spam [39] corpus.

While evaluating the text classification systems it is important that system should be trained and tested with different data in order to check system's validity.

Firstly, the system is trained and tested by using Spam Assassin public corpus that includes total amount of about 9300 emails, where each email is categorized as ham and spam:

We tabulated the results obtained by varying the training ham and spam set size from 50 to 2000 and testing against the corpus of 2500 ham and 2400 spam mails. The results are shown in Table 5.2.

Table 5.2. Experimental results obtained using Spam Assassin corpus

| Training Set Size (Ham and Spam) | False Positives (among 2500 ham test email) | False Negatives (among 2400 spam test email) | Overall Accuracy (%) |
|----------------------------------|---|--|----------------------|
| 50 | 167 (%6.67) | 910 (%38.1) | 77.6 |
| 100 | 142 (%5.68) | 822 (%34.29) | 80.0 |
| 200 | 185 (%7.39) | 645 (%26.90) | 82.9 |
| 300 | 287 (%11.47) | 548 (%22.86) | 82.8 |
| 400 | 214 (%8.56) | 554 (%23.11) | 84.2 |
| 500 | 287 (%11.47) | 467 (%19.48) | 84.5 |
| 1000 | 273 (%10.91) | 489 (%20.40) | 84.3 |
| 1500 | 285 (%11.39) | 398 (%16.58) | 86.0 |
| 2000 | 288 (%11.51) | 343 (%14.29) | 87.1 |

A plot of the above results is as shown below:

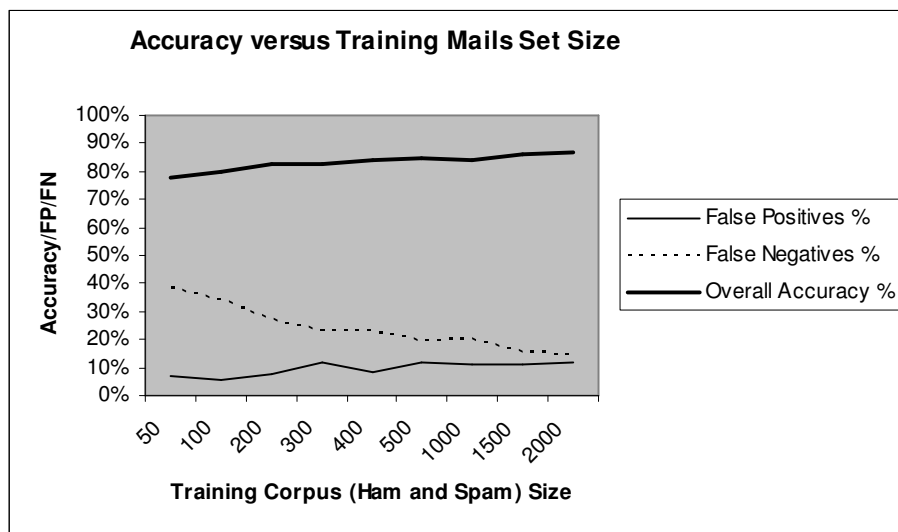


Figure 5.1. System accuracy versus training data using Spam Assassin corpus

We can see from the Figure 5.1 that as the False Negatives decrease, False Positives increase and it can be also seen that as the training data increases the system gives more accurate results.

An expanded table below (Table 5.3) shows the system performance on Spam Assassin corpus in terms of effectiveness measures previously defined:

Table 5.3. System effectiveness measured on Spam Assassin corpus

| Training Set Size (Ham and Spam) | True Positives % | False Positives % | True Negatives % | False Negatives % | Spam Precision % | Spam Recall % | Ham Precision % | Ham Recall % | Overall Accuracy % |
|----------------------------------|------------------|-------------------|------------------|-------------------|------------------|---------------|-----------------|--------------|--------------------|
| 50 | 93,33 | 6,67 | 61,90 | 38,10 | 93,33 | 71,01 | 61,90 | 90,27 | 77,6 |
| 100 | 94,32 | 5,68 | 65,71 | 34,29 | 94,32 | 73,34 | 65,71 | 92,04 | 80,0 |
| 200 | 92,61 | 7,39 | 73,10 | 26,90 | 92,61 | 77,49 | 73,10 | 90,82 | 82,9 |
| 300 | 88,53 | 11,47 | 77,14 | 22,86 | 88,53 | 79,48 | 77,14 | 87,06 | 82,8 |
| 400 | 91,44 | 8,56 | 76,89 | 23,11 | 91,44 | 79,83 | 76,89 | 89,98 | 84,2 |
| 500 | 88,53 | 11,47 | 80,52 | 19,48 | 88,53 | 81,96 | 80,52 | 87,53 | 84,5 |
| 1000 | 89,09 | 10,91 | 79,60 | 20,40 | 89,09 | 81,37 | 79,60 | 87,95 | 84,3 |
| 1500 | 88,61 | 11,39 | 83,42 | 16,58 | 88,61 | 84,24 | 83,42 | 87,99 | 86,0 |
| 2000 | 88,49 | 11,51 | 85,71 | 14,29 | 88,49 | 86,10 | 85,71 | 88,16 | 87,1 |

We know that an email filter would offer higher Accuracy. The experiment results we got as in Table 5.3 shows that our filter gives final Overall Accuracy of %87.1 with enough training and test data using Spam Assassin public corpus.

In order to make comparison and test whether the system is viable to different data sets, we decided to use another corpus known as Ling-Spam corpus. Ling-Spam corpus provides total amount of about 3000 emails, where each email is categorized as ham and spam already for evaluating email filtering systems:

The email filtering system is tested with 230 ham and 230 spam emails, different from the trained data both taken from the Ling-Spam corpus. The experimental results are shown below in Table 5.4.

Table 5.4. Experimental results obtained using Ling-Spam corpus

| Training Set Size (Ham - Spam) | Training Corpus Size | False Positives | False Negatives | Overall Accuracy % |
|--------------------------------|----------------------|-----------------|-----------------|--------------------|
| 50 - 50 | 100 | 111 (%48.26) | 50 (%21.73) | 65 |
| 100 - 100 | 200 | 97 (%42.24) | 45 (%19.56) | 69.1 |
| 150 - 150 | 300 | 78 (%33.91) | 43 (%18.69) | 73.7 |
| 200 - 200 | 400 | 64 (%27.82) | 39 (%16.95) | 77.6 |
| 250 - 250 | 500 | 43 (%18.69) | 39 (%16.95) | 82.2 |
| 300 - 250 | 550 | 35 (%15.21) | 39 (%16.95) | 83.9 |
| 400 - 250 | 650 | 20 (%8.69) | 40 (%17.39) | 87.0 |
| 500 - 250 | 750 | 12 (%5.21) | 42 (%18.26) | 88.3 |

A plot of the above results is as shown below:

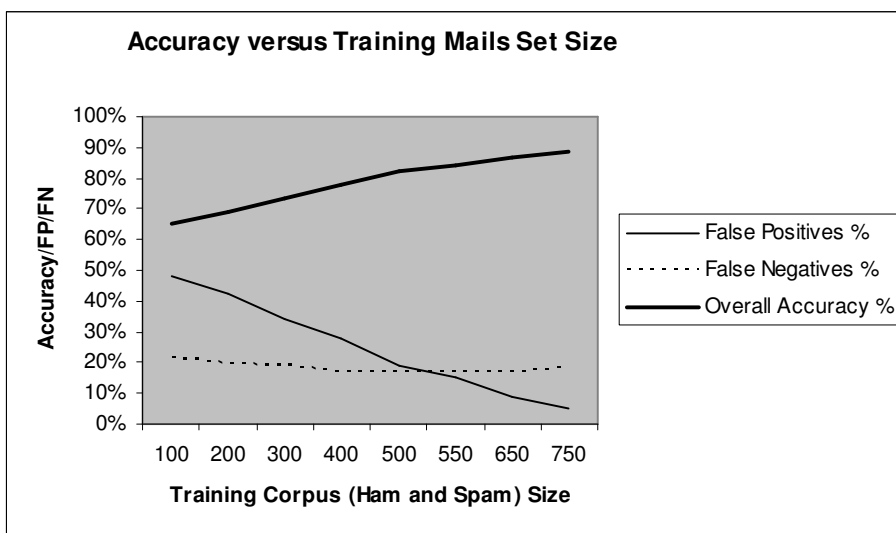


Figure 5.2. System accuracy versus training data using Ling-Spam corpus

It can be seen from the Figure 5.2 that as the training set increase the system decreases the error rate by means of decreasing False Positives and False Negatives, so Overall Accuracy increases. Finally, we got overall accuracy of %88.3 using Ling-Spam corpus.

An expanded table below (Table 5.5) shows the system performance on Ling-Spam corpus in terms of effectiveness measures previously defined:

Table 5.5. System effectiveness measured on Ling-Spam corpus

| Training Sets Size (Ham - Spam) | True Positives % | False Positives % | True Negatives % | False Negatives % | Spam Precision % | Spam Recall % | Ham Precision % | Ham Recall % | Overall Accuracy % |
|---------------------------------|------------------|-------------------|------------------|-------------------|------------------|---------------|-----------------|--------------|--------------------|
| 50 - 50 | 51,74 | 48,26 | 78,27 | 21,73 | 51,74 | 70,42 | 78,27 | 61,86 | 65,0 |
| 100 - 100 | 57,76 | 42,24 | 80,44 | 19,56 | 57,76 | 74,70 | 80,44 | 65,57 | 69,1 |
| 150 - 150 | 66,09 | 33,91 | 81,31 | 18,69 | 66,09 | 77,95 | 81,31 | 70,57 | 73,7 |
| 200 - 200 | 72,18 | 27,82 | 83,05 | 16,95 | 72,18 | 80,98 | 83,05 | 74,91 | 77,6 |
| 250 - 250 | 81,31 | 18,69 | 83,05 | 16,95 | 81,31 | 82,75 | 83,05 | 81,63 | 82,2 |
| 300 - 250 | 84,79 | 15,21 | 83,05 | 16,95 | 84,79 | 83,34 | 83,05 | 84,52 | 83,9 |
| 400 - 250 | 91,31 | 8,69 | 82,61 | 17,39 | 91,31 | 84,00 | 82,61 | 90,48 | 87,0 |
| 500 - 250 | 94,79 | 5,21 | 81,74 | 18,26 | 94,79 | 83,85 | 81,74 | 94,01 | 88,3 |

5.4. Conclusions

The transmission of spam through Internet is increasingly growing day by day. Many times, even the email filter is used; spam mails are still able to reach the user's inbox. Thus, an email filtering is researched as a binary text classification problem, and email filtering system is designed and implemented for classifying incoming mails as spam or ham.

It is known that the offered system should give high overall accuracy. At the same time the system should give higher spam precision values, so that false positives should be less, since user doesn't want their legitimate (ham) mails classified as spam.

We have seen from the experiments that the corpus used for training and testing plays a big role in system success. Our system had overall accuracy %87.1, spam precision %88.5, and spam recall %86.1 when the Spam Assassin corpus is involved (see Table 5.4). However, the system had overall accuracy %88.3, spam precision value %94.8, and spam recall of %83.8 when the Ling-Spam corpus is used (see Table 5.5). As we know that higher the Spam Precision is, the less ham (valid) emails would be misclassified as spam. Our system provides enough spam precision values and we can say that using Ling-Spam corpus got more efficient results even though the corpus size is smaller than the Spam Assassin corpus.

Ling-Spam corpus was chosen so that we should be able to make some comparison with other systems. In the previous work done by Androutsopoulos (et al) [41], Outlook was involved with the Ling-Spam corpus and it is seen that spam precision value %87.9, spam recall value %53.0. Naïve Bayes classifier resulted with spam precision of 96.8, and spam recall %81.1.

It can be concluded that when compared with other classifiers using effectiveness measures, our email filter system shows better performance than Outlook, and gives promising results when compared with its machine learning familiar known as Naïve Bayes classifier.

5.5. Future Work

In addition to filtering emails as spam or ham, an (improved) email classifier can be implemented so that the system should be able to categorize the incoming emails as Corporate, Money, Holiday, Academic, Personal etc...

Another point can be worked that Natural Language Processing techniques on email filtering such as the semantics of the emails may be used for resolving ambiguity on spam and ham mails for filtering purposes.

REFERENCES

- [1] <http://www.yahoo.com>
- [2] D. Koller and M. Sahami, “Hierarchically Classifying Documents Using Very Few Words”, Proceedings of ICML-97, 14th International Conference on Machine Learning, 1997.
- [3] S. T. Dumais, H. Chen, “Hierarchical classification of Web content”, Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval, 2000.
- [4] Andreas S. Weigend, Erik D. Wiener, Jan O. Pedersen. “Exploiting Hierarchy on Text Categorization”, Information Retrieval, 1999.
- [5] M. Sahami, S. Dumais, D. Heckerman, E. Horvitz. “A Bayesian approach to filtering junk email”, AAAI Workshop on Learning for Text Categorization, 1998.
- [6] Drucker, H., Wu, D., and Vapnik, V. “Support vector machines for spam categorization”, IEEE Transactions on Neural Networks, Vol. 10, No. 5, 1999.
- [7] Li, Y. and Jain, A. “Classification of text documents”, The Computer Journal, 1988.
- [8] Sable, C. and Church, K. “Using Bins to Empirically Estimate Term Weights for Text Categorization”, Conference on Empirical Methods in Natural Language Processing, 2001.
- [9] McKeown, K. R., R. Barzilay, D. Evans, V. Hatzivassiloglou, J. L. Klavans, A. Nenkova, C. Sable, B. Schiffman and S. Sigelman. “Tracking and Summarizing News on a Daily Basis with Columbia's Newsblaster”, Proceedings of the Human Language Technology Conference, 2002.
- [10] Lewis, D. and Schutze H. “Text Categorization and its Application to Filtering of Sensitive Content”, 2002.

- [11] Andrew McCallum, Ronald Rosenfeld, Tom Mitchell and Andrew Ng. “Improving Text Classification by Shrinkage in a Hierarchy of Classes.”, Proceedings of ICML-98, 15th International Conference on Machine Learning, 1998.
- [12] Soumen Chakrabarti, Byron E. Dom, Piotr Indyk. “Enhanced Hypertext Categorization using Hyperlinks”, SIGMOD Conference, 1998.
- [13] Bo Pang, Lillian Lee, Shivakumar Vaithyanathan. “Thumbs up? Sentiment Classification using Machine Learning Techniques”, Proceedings of EMNLP, 2002.
- [14] Turney, P.D. “Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews”, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02), 2002.
- [15] W. Gale, K. Church, and D. Yarowsky. “A method for disambiguating word senses in a large corpus”, Computers and Humanities, 26:415--439, 1993.
- [16] C. J. van Rijsbergen, *Information Retrieval*, Butterworths, 1979.
- [17] Charles Wayne. “Multilingual topic detection and tracking”, LREC2000, 2000.
- [18] David D. Lewis. “An evaluation of phrasal and clustered representations on a text categorization task”, In Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1992.
- [19] Sebastiani F. “Machine learning in automated text categorization”, ACM, 2002.
- [20] Fellbaum C., *WordNet, An Electronic Lexical Database*, MIT Press, 1998.
- [21] E. Riloff, “Little words can make a big difference for text classification”, In Proceedings of the 18th ACM SIGIR Conference, 1995.
- [22] C. L. Sable, V. Hatzivassiloglou, “Text-based approaches for non-topical image categorization”, International Journal of Digital Libraries, 3(3):261-275, 2000.

- [23] Mosteller, F., DL Wallace. “Inference and Disputed Authorship”, The Federalist. New York: Springer-Verlag, 1964.
- [24] Salton G., Buckley C., “Term-weighting approaches in automatic text retrieval”, Information Processing and Management, 1988.
- [25] Salton, Gerard, *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Addison Wesley, 1989.
- [26] Hayes, P. J., & Weinstein, S. P., “CONSTRUE/TIS: A system for content-based indexing of a database of news stories”, In Second Annual Conference on Innovative Applications of Artificial Intelligence, 1990.
- [27] Yang, Liu “A Re-Examination of Text Categorization Methods”, 22nd Annual International SIGIR, 1999.
- [28] Tom Mitchell, *Machine Learning*, McGraw Hill, 1997.
- [29] Quinlan, J.R. “Induction of decision trees”, Machine Learning, vol. 1, pp 81-106, 1986.
- [30] Fuhr, Buckley, “A Probabilistic Learning Approach for Document Indexing”, Information Systems, 1991.
- [31] William W. Cohen, Yoram Singer, “Simple, Fast, and Effective Rule Learner”, Proceedings of the sixteenth national conference on Artificial intelligence, 1999.
- [32] T. Joachims. “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”, Proceedings of ECML-98, 10th European Conference on Machine Learning, 1998.
- [33] David Lewis and Jason Catlett. “Heterogeneous uncertainty sampling for supervised learning”, Proceedings of ICML-94, 11th International Conference on Machine Learning, 1994.
- [34] Yan, T., W. & Molina, G. (1995). “SIFT – A tool for wide-area information dissemination”, Proc. 1995 USENIX Technical Conference, 1995.
- [35] L. Cranor and B. A. LaManchia., “Spam!”, Communications of the ACM , 1998.

- [36] Yang, Y. and Liu, X. “A Re-examination of Text Categorization Methods”, In Proceedings of SIGIR-99, 1999.
- [37] Paul Graham, “A plan for Spam”, 2002.
- [38] <http://www.spamassassin.org>
- [39] http://it.demokritos.gr/skel/i-config/downloads/lingspam_public.tar.gz
- [40] Manning D., Schütze H., *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.
- [41] Androutsopoulos et al, “Learning to Filter Spam E-Mail: A comparison of Naïve Bayesian and a Memory-Based Approach”, Proceedings of the workshop Machine Learning and Textual Information Access, 2000.