K-Spectrum Support Vector Machine Classifier for Spam Filtering

By

Ming Yang

Bachelor of Computer Science, UNB, 2006

A REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s): Prabhat Mahanti, PhD, Computer Science, UNBSJ
Kim, Dongmin, PhD., Faculty of Business, UNBSJ

Examining Board: Janet Light, PhD, Computer Science, Chair, UNBSJ
Prabhat Mahanti, PhD, Computer Science, UNBSJ
Kim, Dongmin, PhD., Faculty of Business, UNBSJ
Christopher J.O. Baker, PhD, Computer Science, UNBSJ

THE UNIVERSITY OF NEW BRUNSWICK

May, 2013

Abstract

Traditionally machine learning approaches including Support Vector Machine (SVM) for spam filtering use the bag of words text representation technique to represent its features. However, this technique does not take the word order information into account and is not suitable for languages that do not use white spaces as word delimiters. Therefore, it is appealing to treat every email as a string of symbols by using a string-based approach.

In this report, we implement a contiguous string-based approach, which is called k-spectrum kernel, for use with SVM in a discriminative approach to the spam classification problem. When using the k-spectrum SVM spam classifier, email texts are implicitly mapped into a high-dimensional feature space. The classifier produces a decision boundary in this feature space, and emails are classified based on whether they map to the positive (spam) or negative side (non-spam) of the boundary. Our experimental results demonstrate that the k-spectrum SVM spam classifier could offer an effective and accurate alternative to other approaches of spam filtering, such as generally used approaches including Naive Baysian and SVM classifier that is based Bag-of-Words (BOW).

Dedication

To my husband, Xiaoyi Li, whose meta-support has sustained me through this effort!

Acknowledgements

I would like to take this opportunity to express my deep sense of gratitude and profound feeling of admiration to my supervisors – Dr. Prabhat Mahanti and Dr. Dongmin Kim. I will be eternally grateful for their support and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Electronic mail is one of the most widely used Internet technologies of today [38]. It allows people to communicate with each other in an efficient and inexpensive way. With the explosive growth of the Internet and the speed it is able to handle, communication has been revolutionized with electronic messaging systems. However, the technology has been abused by spammers who send out thousands of unsolicited bulk messages or spam emails to people against their wishes. Research shows that the volume of spam has become very high and today ranging from 88 to 92% all emails in the world [1]. Therefore, people are spending more and more time classifying and filtering e-mails to facilitate retrieval when necessary.

A number of solutions have been proposed and used to help prevent e-mail spam. Some techniques require actions by end users; for example, users can define a friend list or a black list to accept legitimate e-mail but reject e-mail from addresses that are on the black list. Also, people can manually define some rules by specifying lists of words or phrases disallowed in incoming messages. For example, messages containing the phrase “Low Price” would be rejected if the phrase is defined by a rejection rule. However, the addresses of spammers and the words or frequent terms in spam email change over time, so it is time consuming for users to constantly tune the rules. Therefore, it is highly

desirable to develop a filtering system that can automatically learn how to classify e-mails and filter spam.

Since machine learning techniques in text categorization have achieved great success [22], many spam filters use machine learning techniques to analyze the textual part of email messages and to improve the auto detection of spam messages. The e-mail classification can be considered a special case of text categorization with the categories being spam and non-spam. The most well-known approach is Naive Bayesian classification, which is widely used by many spam filters [2]. Another machine learning technique called Support Vector Machine (SVM) [4] also has achieved a high accuracy in filtering spam messages.

Traditionally, machine learning approaches for spam filtering use the bag-of-words text representation technique; that is, every message is treated as a set of words that appear a certain number of times [6, 7]. However, the bag-of-words text representation technique does not take the word order information into account and can lose inflection information due to removal of stop words [8]. Therefore, it is appealing to treat every message as a string of symbols [11]. Using a string-based approach can eliminate the need to define word boundaries, which is especially attractive for oriental languages such as Chinese and Japanese because they do not use white spaces as word delimiters [9, 10]. Also, such an approach takes non-alphabetical symbols into account and is robust to spelling errors [7], which is particularly useful for spam classification, because many spammers try to fool spam filters by misspelling typical spam words (e.g. ‘V1agra’ instead of ‘Viagra’).

1.2 Problem to Solve

In this report, we will implement a contiguous string-based approach, which is called k-spectrum kernel [23], for use with SVM in a discriminative approach to the spam classification problem.

There are two kinds of "string" when mapping a message to a high dimensional feature vector by representing messages as strings of symbols. One uses non-contiguous substrings as features, and the other uses contiguous substrings called character-level n-grams [12]. Both approaches represent each message by a substring (contiguous or non-contiguous) feature vector, where each entry of the vector represents the frequencies of a feature. This feature vector is then used with SVM to classify spam. Table 1.1 lists the advantages and disadvantages of both approaches.

Lodhi et al. [8] propose a non-contiguous string-based approach called String Subsequence Kernel Approach (SSK) to categorize text documents and compared its performance with the contiguous string-based (n-gram) approach. The n-gram contiguous kernel outperformed the SSK kernel in most cases when the testing datasets were larger [8]. Also, SSK has a high computational cost [8, 27]. In [27]; it took 16h50min for a SSK classifier to run on a small number of 3902 samples of recipient email addresses, which makes it not practical for spam filtering. Therefore, this report will focus on the implementation of contiguous string-based for spam classification.

String-based Approach	Advantage(s)	Disadvantage(s)
Non-contiguous string-based Approach	• Reserve all the word order information • Does not require linguistic knowledge • Suitable for oriental languages that do not use white spaces as word delimiters	• High computational complexity
Contiguous string-based (n-gram) Approach	• Does not require linguistic knowledge • Suitable for oriental languages that do not use white spaces as word delimiters • More efficient implementation • Lower computational complexity	• Does not reserve all the word order information

Table 1.1 Advantages and disadvantages of string-based approaches

Paper [23] proposes a computationally efficient kernel, the k-spectrum kernel, which is based on the contiguous string-based (n-gram) approach and integrated with a SVM

classifier to classify the biological data. While the k-spectrum SVM classifier was evaluated on the REUTER corpus in the context of general text classification tasks, to my best knowledge, there is no evaluation or description of its performance on spam corpuses. Therefore, it is worth investigating and testing whether the k-spectrum approach can also be applied to spam classification tasks. If the performance of the string-based classifiers is satisfactory, they may be used for spam filters in the future.

Since the naïve approach of the k-spectrum kernel can still have a high computation cost if the size of the text is large, this report will use suffix tree to reduce the cost of computing the kernel.

1.3 Literature Review

Some of the first published work on content-based spam filtering was carried out by Ben Medlock [17] using a SVM classifier. It used the bag-of-words approach to represent the email text as features and compared the testing results with other classification algorithms. The SVM classifier was reported to have acceptable test performance in terms of accuracy and speed. However, the training and test sets were not publicly available, thus rendering the experiments non-replicable.

Androutsopoulos et al. [28] present results for spam filtering on the LingSpam corpus. They investigate the effect of attribute-set size, training-corpus size, lemmatization, and stop-lists and confirm that the Naive Bayesian classifier has a high precision and recall rate on unseen messages. Tiago et al. [35] compared the Linear SVM classifier that uses

the bag-of-words approach to represent its features with seven different versions of Naive Bayes classifiers on the Enron Corpus; the results favored the SVM classifier.

Lodhi et al. [8] propose a novel string kernel that uses all the non-contiguous strings of texts as features. In [8], the performance of their proposed string kernel is compared to n-gram kernel which is simply a linear kernel that measures the similarity between texts that are transformed to features by using n-gram (contiguous substring). However, these kernels with SVM are only proposed and tested for general text classification.

1.4 Contributions

While Lodhi et al. [8] have shown that the contiguous string-based approaches outperforms the bag of words approach in the context of general text classification, and Leslie et al. [23] have reported promising results on biological data, few researchers have applied these findings to the context of Spam classification. In this regard, there is a need to study and test whether these approaches can also be applied to spam classification tasks. The main contributions of my report are summarized as follows:

- Implement the k-spectrum SVM classifier by applying a contiguous string-based approach in conjunction with SVM to spam classification, and aim to provide performance data
- Test the k-spectrum SVM spam classifiers on the LingSpam corpus and compare their performances with other content-based spam filtering techniques (e.g. Bayesian filtering) examined in [28] and [35]. It is worth evaluating whether the k-spectrum SVM classifiers perform better than other popular filtering techniques. If

the k-spectrum classifier is better or at least the same, it may be used for spam filtering in the future.

- Compare the performance of the SVM spam classifier based on traditional bag-of-words representation with the SVM spam classifier based on contiguous-based substring representations (k-spectrum classifier). Currently, most machine learning approaches for spam filtering, including SVM, use the bag of words text representation technique, but there are several disadvantages of bag of words, which were described earlier. The string-based approach can overcome these disadvantages, but the comparisons involving spam filtering have not been done by other researchers before.
- Evaluate whether the k-spectrum SVM spam classifier is effective in languages that do not use white spaces as delimiters by performing tests on TREC 2006 Chinese Spam corpus.

In the following two chapters, the report will provide a brief introduction to SVM and Suffix Trees. Chapter 4 will describe the detailed implementation of the k-spectrum kernel classifier for spam filtering, and Chapter 5 will explain the performance and testing results of the implemented spam classifier.

Chapter 2

Support Vector Machine and K-Spectrum Kernel

This chapter provides an introduction to Support Vector Machine (SVM) and string kernel functions. This chapter also reviews the main idea behind the k-spectrum kernel [23], which is a contiguous string-based kernel originally proposed for biological analysis. When using k-spectrum in conjunction with SVM as a spam classifier, email texts are implicitly mapped into a high-dimensional vector space. The features used by the spam classifier are a set of all possible contiguous substrings of fixed length k. The classifier produces a decision boundary, and emails are classified based on whether they map to the positive (spam) or negative side (non-spam) of the boundary.

2.1 Overview of Support Vector Machine

A support vector machine constructs a hyperplane that separates two classes of data in high dimensional space while maximizing the margin between them, which can be used for text classification, image recognition, and other tasks.

This section presents some basic knowledge of SVM, which draws from [2]. Suppose a data set contains n labeled example vectors $\{(x_1, y_1)\ldots(x_n, y_n)\}$, where x_i is a p-dimensional vector which contains features describing example i, and y_i is the class label for that example and indicates the class to which the point x_i belongs. For spam filtering, y_i is either 1 or -1 depending on the class of email. For example, the spam class is assigned the numerical class label 1, and the ham class is assigned label -1. Thus, the training data set can be described as follows:

$x = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$,

$x_i \in R, y_i \in \{1, -1\}$.

The learning task amounts to looking for the optimal separating hyperplane between the two classes such that data points on one side will be labeled $y_i = 1$ and the ones on the other side as $y_i = -1$. The hyperplane found by a Support Vector Machine should maximize the margin between the classes' closest points (Figure 2.1). Such closest points (e.g. x_1 and x_2) are called support vectors, which have the most influence on the position of the optimal separating hyperplane. The margin is the perpendicular distance between the separating hyperplane and the hyperplane through the support vectors.

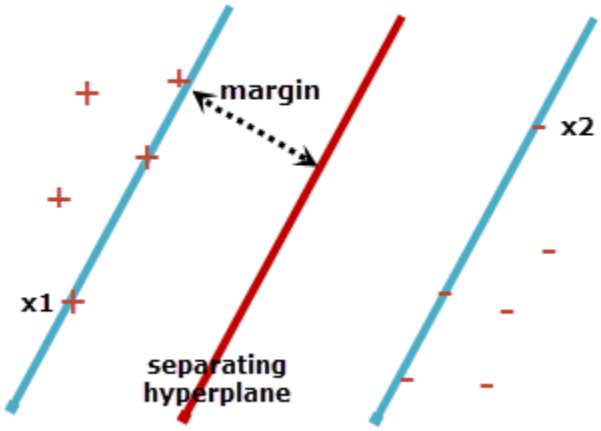


Figure 2.1 Support vectors and margin

The separating hyperplane can be written as

$\{x \in H \mid w \cdot x + b = 0\}, w \in H, b \in R$

, where $w \cdot x$ is the inner or dot product, b is the threshold, and the weight vector w is the normal vector to the separating hyperplane. Suppose all the training data satisfy the following constraints:

$x_i \cdot w + b \geq +1$ *for* $y_i = +1$

$x_i \cdot w + b \leq -1$ *for* $y_i = -1$

These equations can be converted to one set of inequalities:

$y_i (x_i \cdot w + b) - 1 \geq 0$ *for all* $i = 1, \ldots, n$.

Now consider the hyperplanes passing through the support vectors. The hyperplane for the closest points can be formulated as $w \cdot x + b = 1$ on one side and $w \cdot x + b = -1$ for the other side. Suppose x_1 and x_2 are two points on these hyperplanes (Figure 2.1). We can deduce that $w \cdot (x_1 - x_2) = 2$ because $w \cdot x_1 + b = 1$ and $w \cdot x_2 + b = -1$. The distance between these two hyperplanes is calculated by projecting $(x_1 - x_2)$ onto the separating hyperplane normal vector $w/||w||$, which gives $(x_1 - x_2) \cdot w/||w|| = 2/||w||$. As half the distance, the margin equals to $1/||w||$.

Since the support vector algorithm looks for the separating hyperplane with the largest margin, maximizing the margin is equivalent to minimizing $||w||$ subject to $y_i (x_i \cdot w + b) - 1 \geq 0$. However, this optimization problem is difficult to solve because $||w||$ involves a

square root. By substituting $||w||$ with $\frac{1}{2}||w||^2$, the problem can be converted to the following without changing the solution:

Minimizing $\frac{1}{2}||w||^2$ (2.1)

Subject to $y_i (x_i \cdot w + b) - 1 \geq 0$ for all $i = 1, \ldots, n$. (2.2)

(2.1) and (2.2) form a so-called constrained optimization problem, which can be solved by introducing non-negative Lagrange multipliers [2] α_i, $i = 1, \ldots, n$, one for each of the inequality constraints (1.2). The primal formulation [2] becomes:

$$L(w, b) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{n} \alpha_i \quad (2.3)$$

Then take the derivatives with respect to b and w and set them to zero:

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad (2.4)$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \quad (2.5)$$

Substituting $w = \sum_{i=1}^{n} \alpha_i y_i x_i$ (2.5) back into (2.3) and using the fact that $||w||^2 = w \cdot w$, one can get the dual formulation [3]:

Maximize $W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j$

Subject to $\alpha_i \geq 0$ for all $i = 1, 1, \ldots, n$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$ (2.6)

2.2 Kernel Function

The optimal hyperplane algorithm described in the previous section is a linear classifier, which only works for data that can be separated using a straight line. In 1992, Bernhard Boser, Isabelle Guyon and Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes [5]. The resulting algorithm generates an alternative representation of the data by mapping the data points into a space with a higher dimensional feature space through a replacement:

$$x_i \cdot x_j \rightarrow K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$$

, where $\Phi(\ \ )$ is the mapping function and $K(x_i, x_j)$ is called the kernel function. Data that cannot be separated in the input space can be separated in a space of high enough dimensionality. For example, consider the 2-dimensional dataset in Figure 2.2; the two classes cannot be separated by a straight line. However, with a third dimension such that the + symbols are moved forward and the - symbols are moved back, the two classes become separable by a hyperplane in the plane of the page.

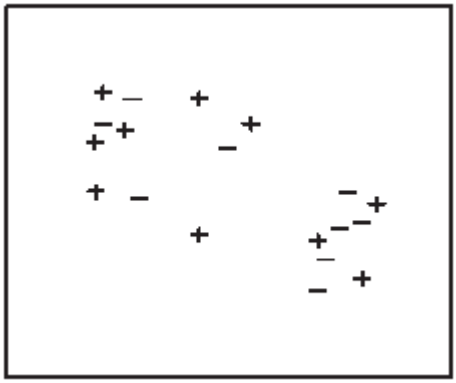


Figure 2.2 Two classes that cannot be separated by a straight line

For binary classification with a given choice of kernel, the learning task therefore becomes:

Maximize $W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$

Subject to $\alpha_i \geq 0$ for all $i = 1, 1, \ldots, n$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$ (2.7)

2.3 Why String Kernel?

The most popular approach used by most document classification methods is called the Bag-of-Words model, which maps texts to a multi-dimensional feature space, where each entry refers to the count of a feature. It usually removes stop words, ignores the word orders, and requires a language-dependent process to split text into words, which leads to extra work when dealing with languages other than English which do not use white spaces as word delimiters [9, 10]. Therefore, it is appealing to use a string-based text representation approach where texts are considered as symbol sequences. This approach

can eliminate the need to define word boundaries, which is attractive in the case of languages that do not use white spaces as word delimiters, such as Chinese and Japanese. Also, it is robust to spelling errors [7], which is especially useful for spam classification, because many spammers try to fool spam filters by misspelling typical spam words, such as using ‘V1agra’ instead of ‘Viagra’.

2.4 K-spectrum Kernel

Perhaps the most natural way to compare two texts (strings) is to count the number of contiguous substrings of length *k* they have in common. Christina Lesline [23] introduced a string kernel method called the spectrum kernel for use with SVM to the protein classification problem. The k-spectrum kernel is based on the n-gram model which represents texts as a high dimensional feature vector, and each feature corresponds to a contiguous substring of length *n*[8]. N-grams are n adjacent characters; for example, the text ‘hello world’ would be composed of the following:

2-gram: he el ll lo ()w wo or rl ld

3-gram: hel ell llo lo() o()w ()wo wor orl rld

4:gram: hell ello llo() lo()w o()wo ()wor worl orld

, where () represents the white space. Since the texts are decomposed into short substrings, errors such as misspelling tend to affect only a limited number of these features, leaving the whole text intact [12].

The spectrum kernel function operates on strings by counting the n-grams in common between two strings [23]. Therefore, the kernel can be defined as:

$$k_p(s,t) = \sum_{u \in \Sigma^p} \phi_u^p(s)\phi_u^p(t)$$
$$\phi_u^p(s) = |\{(v_1, v_2) : s = v_1 \cup v_2\}|$$
$$u \in \Sigma^p$$

For example, consider strings "hat", "act", "ace", and "cat". Their 2-grams are given in the following table:

φ	ha	at	te	ea
hat	1	1	0	0
ate	0	1	1	0
tea	0	0	1	1
eat	0	1	0	1

Table 2.1 2-grams for 'hat', 'act', 'ace', and 'cat'

Since all other dimensions indexed by other strings of length 2 are equal to 0, the kernel matrix for the above strings is:

K	hat	ate	tea	eat
hat	2	1	0	1
ate	1	2	1	1
tea	0	1	2	1
eat	1	1	1	2

Table 2.2 kernel matrix for 'hat', 'act', 'ace', and 'cat'

The number of non-zero features is bounded by *length(x) – k + 1*, which can be very large even for strings of fairly small size. Therefore, it is important to use an efficient approach to count the number of occurrences of each feature. In this report, we will use a

suffix tree to construct the feature space for the SVM classifier. The main ideas behind suffix trees will be reviewed in the next chapter.

Chapter 3

Linear Suffix Tree Construction

As described earlier, the feature space can be very large even for fairly small *k*. A very efficient method for constructing the feature space is to build suffix trees for email texts and count the occurrences of all contiguous substrings of length *k*. In our report, suffix tree generation was primarily implemented to facilitate counting of suffix tree sub branches. In this chapter, we provide a brief introduction to suffix trees and show how to use suffix trees to count the occurrences of substrings efficiently.

3.1 Introduction to Suffix Trees

A suffix tree is a tree in which every suffix of a string *s* is represented in a way that allows for fast implementation of many important string operations. Suppose the string s_i is a substring of *s* if there are strings *u* and *v* such that

$$s = us_iv$$

If *v* is an empty string, s_i is known as a suffix, while if *u* is an empty string, s_i is known as a prefix. For example, the string ‘banana’ has 7 suffixes: ‘banana$’, ‘anana$’, ‘nana$’, ‘ana$’, ‘na$’, ‘a$’, and $. The character ‘$’ is a termination character that is added to the end of the string s to ensure that no suffix of the resulting string can be a prefix of any other suffix [26]. Any string length *m* can be degenerated into *m* suffixes, and these

suffixes can be stored in a suffix tree. The definition of a suffix tree is given in section 5.2 of Gusfield [25]:

'A suffix tree *T* for an m-character string *s* is a rooted directed tree with exactly *m* leaves numbered 1 to *m*. Each internal node, other than the root, has at least two children and each edge is labeled with a non-empty substring of *s*. No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf *i*, the concatenation of the edge-labels on the path from the root to leaf *i* exactly spells out the suffix of S that starts at position *i*. That is, it spells out *S[i..m]* '

For example, the suffix tree for the string 'nana' is shown in the following diagram.

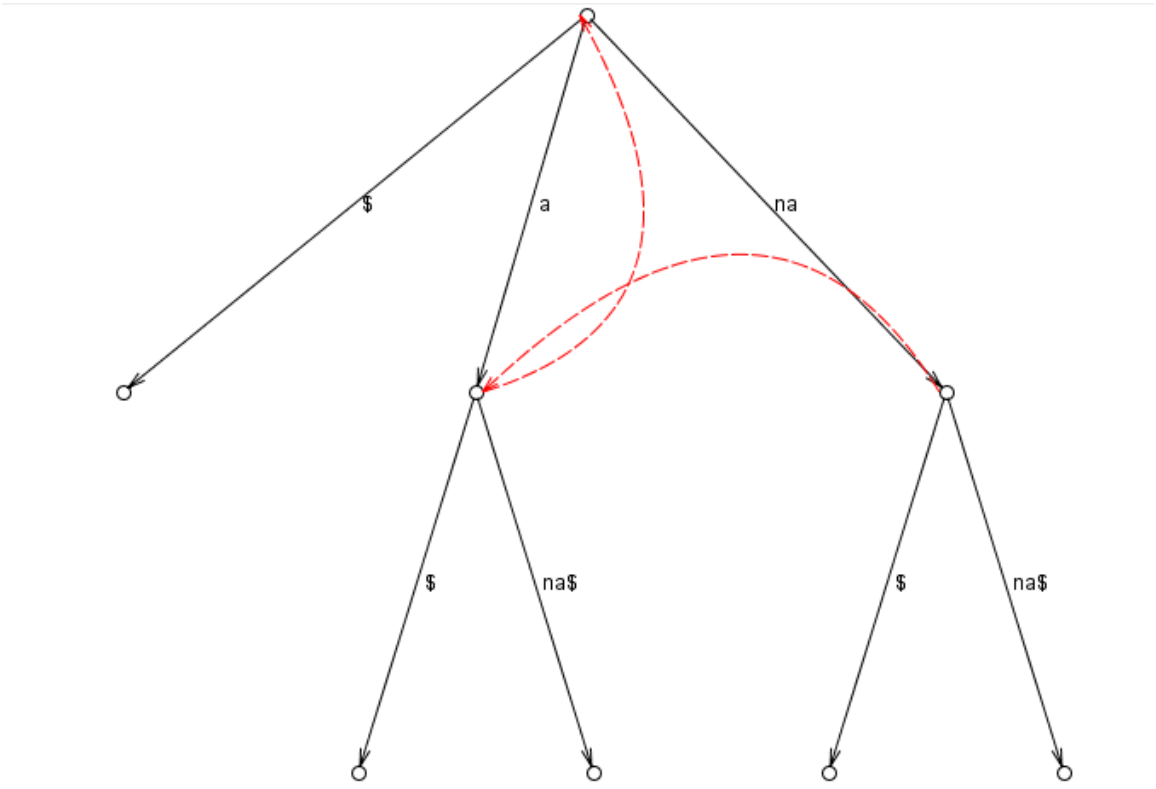


Figure 3.1 suffix tree for the string 'nana'

3.2 Naive Method to build a Suffix Tree

In this section, a straightforward method is presented to create a suffix tree for String *s*. The algorithm is as follows:

Create a single edge T_1 with edge labeled *S[1..m]$*

For $i = 2$ to $m+1$ do (where m is the length of s)

    Construct T_i by adding *s[i,m]$* into T_{i-1}

End

T_{m+1} is the suffix tree of string *s*

To add *s[i…m]$* into the tree (T_{i-1}) after T_1 has been entered, the longest path from the root of T_{i-1} whose label matches a prefix of *s[i,m]$* needs to be found. If the matching path is found in the middle of an edge, then the edge is broken into two edges by inserting a new node before the first mismatched character and right after the last character that is matched on the edge. The time complexity for this naïve method to build a suffix tree for String s is $O(m^2)$ [25].

3.3 Ukkonen's Linear-time Suffix Tree Algorithm

The naïve method takes $O(m^2)$ time to build a suffix string, which makes it impractical. Esko Ukkonen [26] introduced a linear-time algorithm for building a suffix tree. The algorithm can be implemented as a simple but inefficient method, followed by several 'common-sense' tricks that speed up the running time [25].

The construction process is started by creating the suffix tree for all prefixes of string *s*. It constructs an implicit suffix tree T_i for each prefix *s[1..i]* of string *s*, starting from T_i and increasing the length of the prefix by one character per *i* until T_m is built. The suffix tree would have *m* (the length of string *s*) leaves, and each leaf represents a suffix of *s*. At a higher level, the suffix tree construction algorithm is divided into *m* phases. Construction of the first phase (*i = 1*) is easy, which is just a single edge from the root labeled by the first character of the string and ends in leaf 1. Each other phase *i+1* is further divided into *i+1* extensions, one for each of the *i+1* suffixes of *s[1..i+1]*. In extension *j* of phase *i+1*, the end of the path labeled with substring *s[j..i]* is first found, and then the path is extended by adding character *s(i+1)* if *s(i+1)* cannot be found in the path[25]. In extension *i+1* of phase *i+1*, the empty suffix of *s[1..i]* is extended by putting the single character *s(i+1)* into the tree if it is not in the current tree. The algorithm is as follows:

```
1. Construct T_i;
2. /* Construct T_{i+1} from T_{i+1} */
  For i = 1 to m − 1 do /* Phase i + 1 */
        For j = 1 to i + 1 do /* Extension j */
```

Find the end of the path *p* from the root whose label is *s*[*j*, *i*] in current tree and extend *p* with *S[i+1]*

End For

End For

3. Convert T_m into a suffix tree *S*;

When extending the path *p* (a suffix of *s[j..i]*) by adding the new character in the *(i+1)th* stage, the following three rules should be followed:

1) If the path *p* ends at a leaf, the character *s(i+1)* is added to the end of the edge leading into the leaf node.

2) If the path *p* ends in the middle of an edge and the following character is not equal to *s(i+1)*, create a new edge starting from the end of *p* and add a new internal node at that point. The leaf at the end of the new edge is given the number *j*.

3) If the path *p* ends in the middle of an edge and the following character is equal to *s(i+1)*, this means the suffix is already present in the current tree, so do nothing.

For example, consider the left implicit suffix tree in the following image for string *axabx*: when a sixth character *c* is added to the string, the first four suffixes (*axabx*, *xabx*, *abx*, and *bx*) are extended by rule 1, and the 5th suffix (*x*) is extended by rule 2. The result is shown on the right side of figure 3.2.

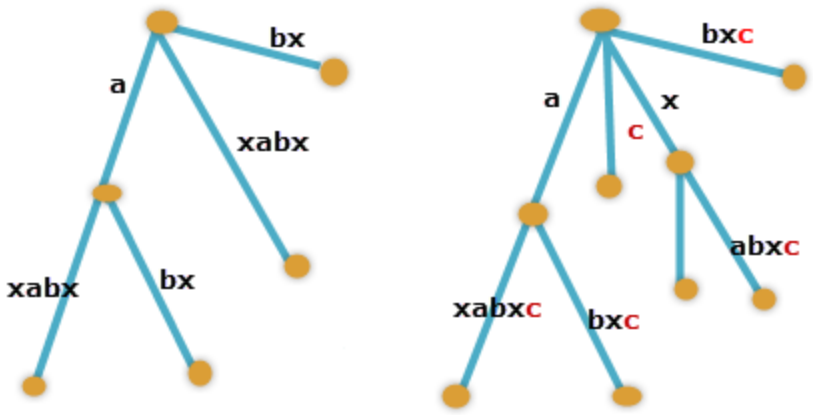


Figure 3.2 Suffix trees for ‘axabx’ and ‘axabxc’

3.4 Implementation Tricks

Using the construction algorithm described in the previous section, there would be *m* phases of constructing the suffix tree, and for each phase *i+1*, the entire length of the suffix down from the root needs to be traveled to locate the ends of all the *i+1* suffixes of *s[1..i]*, so the time complexity for extension *j* of phase *i+1* would be *O(i+1-j)*,and T_m would be created in $O(m^3)$ time. In this section, the implementation tricks are described to reduce the time complexity from $O(m^3)$ to *O(m)*.

3.4.1 Trick 1: Suffix Link

Suffix links are an important feature of suffix trees. If there is a node v in the tree with a label *cα*, where *c* is a character and *α* is a string, then the suffix link of *v* points to *s(v)* which is node labeled with *α*. If *α* is empty, then *s(v)* is the root. The following image shows a suffix tree for string *CACAO* with suffix links. The arrows represent the suffix

links, for example, the node labeled with *CA* has a suffix link that points to the node labeled with *A*.

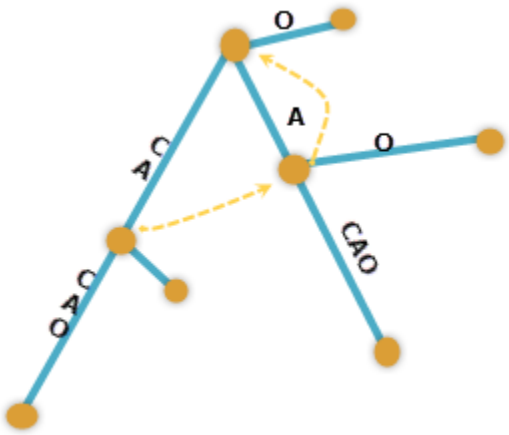


Figure 3.3 Suffix tree for string *CACAO* with suffix links

The suffix links keep track of what or how much of the string has been matched so far. When a mismatch is found, we can jump along the suffix link to see if there is any match later in the text and the string. Not only that, this jumping trick also helps us to save computation time while constructing suffix trees. Thus suffix links are very important constructs of suffix trees. Suffix links exist for every internal node of a suffix tree, which is proved in [25].

3.4.2 Trick 2: Skip/Count

The skip/count trick helps to locate the place for adding new character *s(i+1)* in phase *i+1* much faster. The use of this trick reduces the time on traveling down from *s(v)* to the leaf to be equal to the number of nodes in the path instead of the length of the path.

Assume the number of characters on each edge is known and the length of the path down from *s(v)* is *n*. Since there cannot be more than one edge out of *s(v)* that starts with the same character, the first character in the suffix will appear on only one edge out of *s(v)*. Suppose the number of characters on this edge is *p*; if $p < n$, then set *n* to be *n-p*, and the next position on the string to be matched is *q=p+1*. Then select the edge out of the current node which starts with character *s(q)*; if the label on this path is less than *n*, then again set n to *n-p* and *q=q+p*. Continued in this manner until the edge *p* is greater than *n*. Then the suffix ends exactly *n* characters down that edge. This saves work because now we can reach the end of each edge in constant time as we know the number of characters on each edge as per our implementation assumption. In the following figure, there is a copy of substring *cdefgh* out of node *s(v)*, which is found after 4 nodes down-skips.

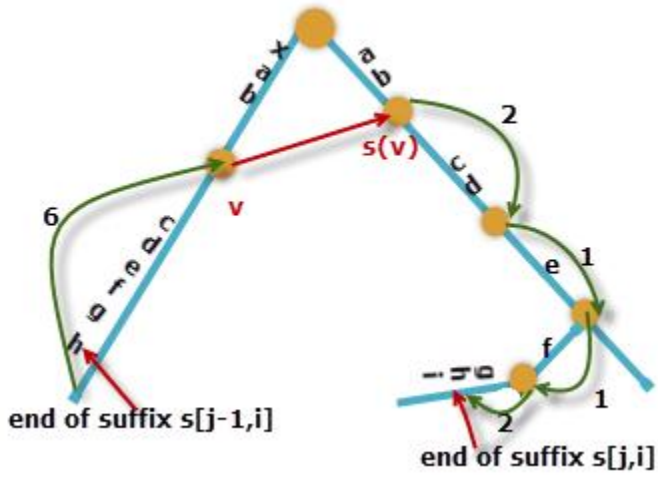


Figure 3.4 Skip/count trick

3.4.3 Rule 3 is a show stopper

If at any phase, rule 3 is applied in extension *j*, then rule 3 will be applied in all further extensions. The reason is that when rule 3 is applied, the path label *s[j..i]* must be followed by character *s(i+1)*, which means the paths labeled *s[j+1..i]*, *[j+2..i]*…are all followed by *s(i+1)*, so rule 3 can also be applied to extension *j+1*, *j+2*,…,*i+1*. Therefore, we can just stop the current phase of the algorithm at the point when rule 3 is applied.

3.4.4 Once a leaf, always a leaf

If a leaf is created and labeled *j* (for the suffix starting at position *j*) at some point during the construction process, this will remain a leaf with label *j* in all the successive tree creations throughout the whole process. Therefore, once a leaf is created, we can simply extend it according to rule 1 described previously, by appending the character *s(i+1)* to its end. Suppose any edge in an implicit suffix tree is represented by *s[p..q]* and index *q* is equal to *i*, which is the phase index. In phase *i+1*, *q* gets incremented to *i+1*, so the edge is labeled with substring *s[p..i+1]* and can be represented by *(p, e)* where *e* is a global index that is set *i+1* one in each phase. In phase *i+1*, rule 1 needs to be applied to extensions from 1 to the last extension in phase *i*. There is no need to do any explicit work to implement these extensions. The only thing required is constant work to increment *e* instead.

By putting all these tricks together, since the implicit extensions in every phase takes O(m) time, the computational cost for building the suffix tree is O(m). The detailed proof can be found in chapter 6 of [25].

3.5 Substring Count

This section demonstrates how to use suffix trees to find the number of occurrences of a string *s*. Assume we have built a suffix tree for text *T*; by performing a breadth-first search, we first find the path that matches the string *s*. If no path is found, the number of occurrences of *s* is 0. If a path is found, go to the node *n* at which the path ends and count the number of leaves beneath that node. Since every path from node *n* to its leaves spells out a suffix of T having s as a substring, the number of occurrences of string *s* is also the number of leaves beneath node *n*. Therefore, to find all matches of string *s*, we build a suffix tree for the text in O(m) and then use the tree to search for all occurrences of *s*. It takes $O(n + k)$ to find all occurrences of String *s* (of length *n)*, where and *k* is the number of occurrences of *s*. The total time needed in this approach is $O(n + m + k)$.

For example, consider text *T* = ‘abcabddaabc$’ and string *s* = ‘ab’. The suffix tree is shown in the following figure.

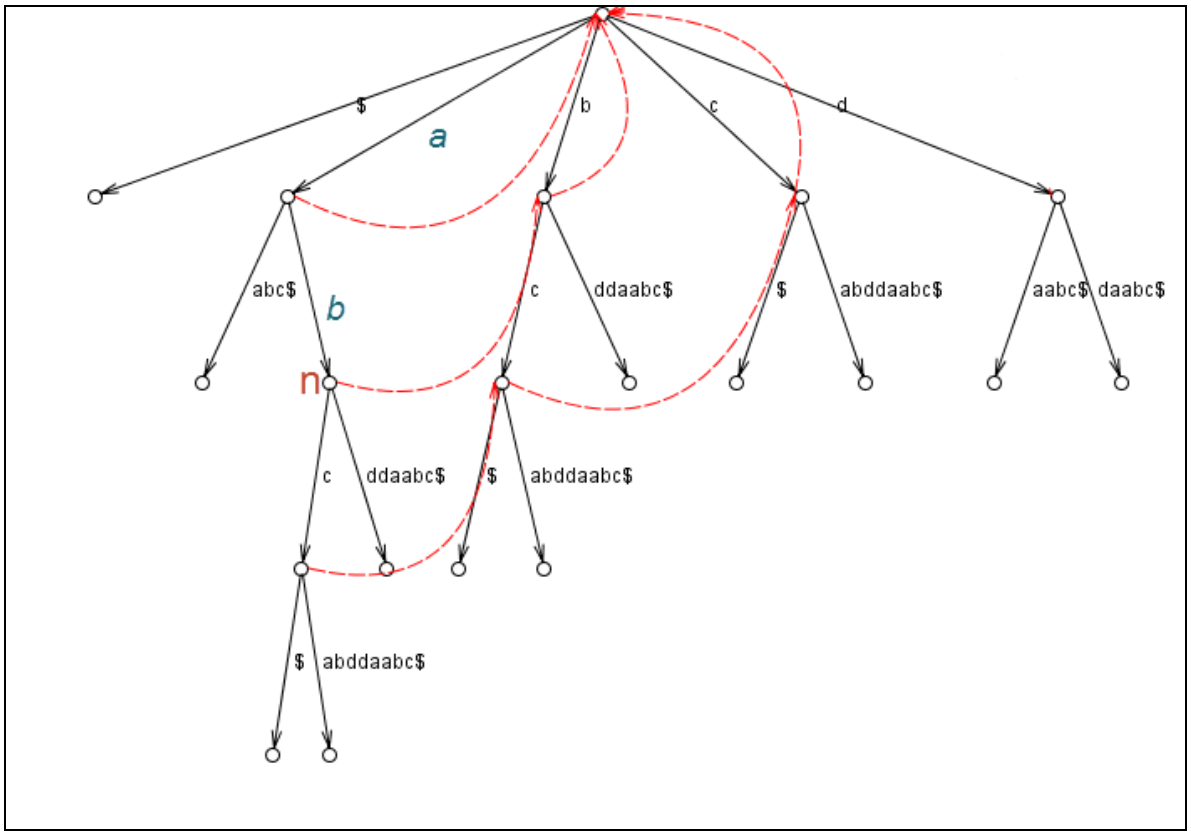


Figure 3.5 Count the number of occurrences of String *s* in suffix tree

The path that matches ‘ab’ is shown in bold, and it ends at node *n*. There are 3 leaves beneath node *n* and they correspond to the 3 occurrences of ‘ab’ in text T.

Chapter 4

Implementation of SVM Spam Filter

In this chapter, the SVM-based spam filter is implementation and testing is described. Three publicly available spam corpuses (Lingspam, Enorn and TREC 2006 Chinese) are used to train and test the performance of the k-spectrum SVM spam classifier. In Section 4.1, we will provide more details about these three spam corpuses. To classify e-mail as spam and non-spam, the SVM-based spam filter must first be trained. As in the use of Bayesian filters, this training involves a computational analysis of message content using a set of both spam and non-spam messages. Essentially the filter will learn to distinguish spam from non-spam. The details of the spam filtering process will be explained in Section 4.2.

4.1 Spam Corpus

We use three publicly available spam corpuses to train and test the k-spectrum SVM spam classifier:

- **Ling-spam Corpora**

  The Ling-spam corpus consists of 2893 email messages, of which 2412 are legitimate emails obtained by downloading digests from the Linguist List and 481 are spam messages received by an author of the corpora [28]. There are four data sets constructed from the original 2893 messages:

  1. Lemmatiser disabled and stop-list disabled
  2. Lemmatiser enabled and stop-list disabled

3. Lemmatiser disabled and stop-list enabled
4. Lemmatiser enabled and stop-list enabled

Lemmatiser enabled means that a lemmatizer is used to convert each word to its base form. Stop-list enabled means that the 100 most frequent words of the Britsh National Corpus [30] are removed from the original messages.

- **Enron-spam Corpora**

  The Enron-spam corpus contains six non-encoded dataset in both raw and pre-processed form [31]. We use only the pre-processed messages in our experiments. The HTML tags and attachments are removed from the messages. Duplicate messages and messages written in non-Latin character sets are also not included. The following table lists the characteristics of the six datasets.

Enron dataset	**Spam**	**Non-spam**	**Total**
Enron 1	1500	3672	5975
Enron 2	1496	4361	5857
Enron 3	1500	4012	5512
Enron 4	4500	1500	6000
Enron 5	3675	1500	5175
Enron 6	4500	1500	6000

- **TREC 2006 Chinese [29]**

The TREC 2006 Chinese Corpus is provided by the CERNET Computer Emergency Response Team (CCERT) at Tsinghua University. It consists of 64620 messages, of which 42854 are spam messages and 21766 are legitimate messages. As in the case of the Ling-spam and Enron-spam corpuses, the HTML tags and attachments are removed from the original messages.

4.2 Implementation Details of k-spectrum SVM Spam Filtering Process

The spam filtering process based on the string-based approach follows three basic steps:

Step 1: Preprocessing of email messages by constructing suffix trees

In this step, we build a suffix tree for each message (spam or non-spam) and store the suffix trees into a list. For each message x_i, we also need to collect the set of k-length distinct substrings into a list L_i. All substring lists are then combined into one global list L_g, which represents all k-length substrings contained by the input dataset. The suffix tree list S_g and global substring list L_g are used to build feature space for the next step. Here is the pseudo code for this step.

```
S_g = new ArrayList();
L_g = new ArrayList();

For each message x_i
    Build suffix tree t_i;
```

Add t_i to S_s;

Get all distinct k-length substrings and add them to list L_i;

Update list L_s by adding L_i to L_s;

End For

Suppose the input dataset contains two short messages:

x_1 = 'Hi Ming!'

x_2 = 'Hello!'

After step 1, S_s will contain two elements t_i and t_2, which are the suffix trees for x_1 and x_2. Suffix tree t_1 is shown in Figure 4.1 and suffix tree t_2 is shown in Figure 4.2. The trees shown in these two figures are called implicit suffix trees which may not have a leaf for each suffix but it encodes all the suffixes of S[1..i]. Every suffix of S[1..i] would be found on some path from the root of such an implicit suffix tree t_i. For example, suffix 'Hi ' can be found on the path 'Hi Ming!'.

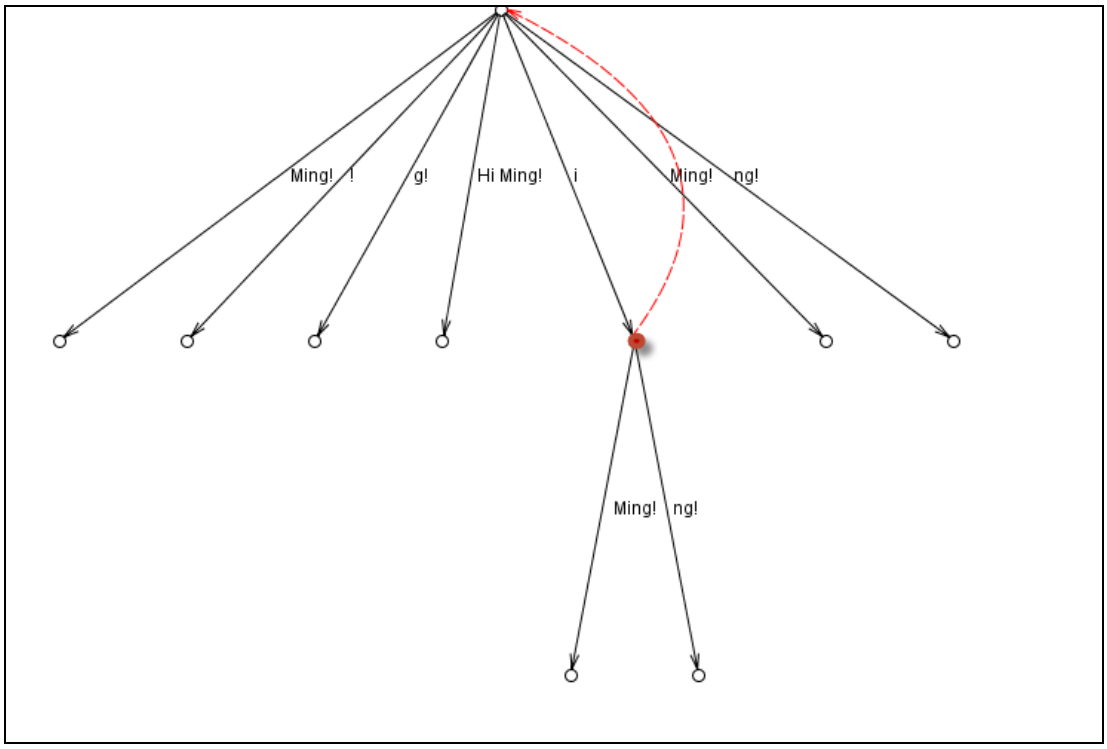


Figure 4.1 Suffix tree t_1 representing 'Hi Ming!' (x_1)

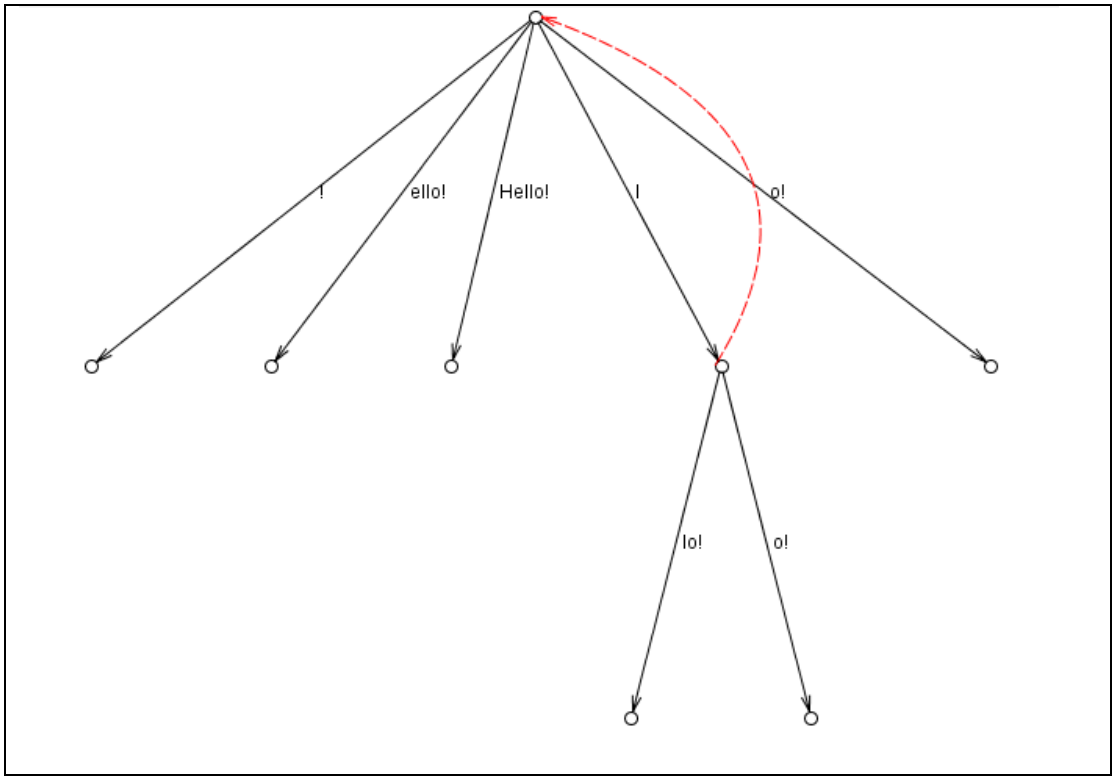


Figure 4.2 Suffix tree t_1 representing 'Hello!' (x_2)

If k is set to 3, L_s will contain 'Hi ', 'i M', ' Mi', 'Min', 'ing', 'ng!', 'Hel', 'ell', 'llo', and 'lo!'. If k is set to 4, L_s will contain 'Hi M', 'i Mi', ' Min', 'Ming', 'Hell', 'ello', and 'llo!'.

Step 2: Generate feature vectors

After this step, every e-mail message in the training set is transformed into a vector of numbers. Each substring s_j in L_s is considered as a feature. The number of times s_j appears in the message is the value of the feature, and there is one feature vector per message. For example, if k is set to 3, message x_1 'Hi Ming!' can be represented as a vector of (1,1,1,1,1,1,0,0,0,0,0), and message x_2 'Hello!' can be represented as a vector of (0,0,0,0,0,0,1,1,1,1) . The number in the vector corresponds to the frequency of each substring's appearance in message x_1 .

To get the frequency of substring s_i's appearance in message x_i, we will search the substring in the suffix tree t_i and count the number of leaves beneath the matching node. For example, we need to get the frequency of the appearance of substring 'i' in message x_1 'Hi Ming!'. First, we search 'i' in the suffix tree t_1, and we find a matching node (solid circle in Figure 1). The number of leaves beneath the matching node is 2, which means there are two occurrences of substring 'i' in message x_1. If no matching node is found, the number of frequencies will be set to zero. Here is the pseudo code for step 2:

```
For each suffix tree t_i in S_s
    Initialize feature vector v_i;
```

```
    For each substring string s_j in L_s
        Search substring s_j in t_i;
        If matching node is NOT found
            add 0 to v_i;
        Else{
            Count the number of leaves beneath the matching node;
            Add the number to v_i;
        }
    End For
End For
```

Step 3: E-mail classification.

In this step, a SVM classifier is implemented with kernels to map the feature vectors from step 2 onto a space and determine the optimal hyperplane between two classes (spam and non-spam). A linear kernel will be used to find the maximal margin in this report.

After the spam classifiers have been trained, new data (testing messages) are then mapped onto that same space and predicted to belong to a category based on which side of the hyperplane they fall on. The performances of the spam classifiers are then evaluated. There are five measurements used: spam precision, spam recall, false negative rate, false positive rate, and accuracy rate.

- Spam precision - Proportion of messages classified as spam that actually are spam.

- Spam Recall - Proportion of the number of correctly-classified spam to the number of spam that are (correctly and incorrectly) classified as spam.
- False negative rate - Proportion of spam messages that are misclassified as legitimate to the number of total spam messages.
- False positive rate - Proportion of legitimate messages that are misclassified as spam messages to the number of total legitimate messages.
- Accuracy rate – Proportion of all messages that are correctly classified.

More details of these five measurements will be illustrated in chapter 5.

4.3 LIBLINEAR

We use more than 2000 email messages as the training dataset and each message contains at least 200 characters, so the feature space is huge even for fairly small k. It is important to use a SVM classification tool that is suitable for solving large-scale classification problems. LIBLINEAR is an open source library that has been developed to deal with data having a large number of features [32]. It inherits many features of the SVM library [33] and supports L2-regularized logistic regression (LR), L2-loss and L1-loss SVM classifiers.

4.4 Java Implementation Generalized Suffix Tree

A Java implementation of Generalized Suffix tree is also used to build suffix trees at step 1. Its implementation is based on Ukkonen's paper [34], and the source code can be downloaded at [33]. It also implements the 'search' operation, which can be used to

search the substrings in a particular suffix tree. A few more methods are also added to the original source code. For example, the method ‘countNode()’ is added to count the number of leaves beneath the matching node.

Chapter 5

Experiment Results

In this chapter, we present experiments on three well-known public corpuses: the Ling-Spam corpus, the Enron corpus, and the TREC 2006 Chinese corpus. Five important measures are used to evaluate the performance of the k-spectrum SVM spam classifier: false positive rate, false negative rate, spam precision, spam recall, and accuracy rate. In section 5.1, we will present more details about these 5 important performance measures. In section 5.2, we will present the experiment results on the 6 Enron Corpora and compare them with the results from [34]. In section 5.3, we will test our spam classifier on the Chinese corpus to evaluate its performance on emails that contain characters other than English.

The details of these three corpora are shown in table 5.1.

Corpus	Sub-datasets	Spam	Non-spam	Total
Ling-Spam	Lemmatiser disabled and stop-list disabled	481	2412	2893
	Lemmatiser enabled and stop-list disabled	481	2412	2893
	Lemmatiser disabled and stop-list enabled	481	2412	2893
	Lemmatiser enabled and stop-list enabled	481	2412	2893
Enron Dataset	Enron 1	1500	3672	5975
	Enron 2	1496	4361	5857
	Enron 3	1500	4012	5512
	Enron 4	4500	1500	6000
	Enron 5	3675	1500	5175
	Enron 6	4500	1500	6000
TREC 2006 Chinese		42854	21766	

Table 5.1 Corpora sizes

5.1 Experiments with Ling-Spam

In this section, our experiments with the k-spectrum SVM spam classifier are evaluated on the Ling-Spam corpus, which consists of 2412 legitimate email messages and 481 spam messages. As mentioned in the previous chapter, two modules are applied to the Ling-spam: stop- list and Lemmatizer. The stop-list and Lemmatizer can either be enabled or disabled, which divides the Ling-Spam corpus into four datasets. In this experiment, all four datasets are trained and tested so that the testing results can be compared with the Na ïve Bayesian classifier performance from paper [28].

To evaluate the SVM classifier on training datasets, the accuracy measure is defined as follows:

$$\text{Accuracy rate} = \frac{\text{Correctly classfied Emails}}{\text{Total Emails}} = \frac{N_{ll} + N_{ss}}{N_s + N_l}$$

, where N_{ll} denotes the number of correctly classified legitimate emails, N_{ss} denotes the number of correctly classified spams, N_s denotes the number of total spams, and N_l denotes the number of total legitimate emails.

To further evaluate the results, we also use Precision, Recall, false positive rate, and false negative rate as the measuring metrics. The spam precision and recall are defined as:

$$\text{Spam Precision} = \frac{N_{ss}}{N_{ss} + N_{ls}}$$

$$\text{Spam Recall} = \frac{N_{ss}}{N_{ss} + N_{sl}}$$

, where N_{ls} denotes the number of legitimate emails that are classified as spam, and N_{sl} denotes the number of spams that are classified as legitimate emails.

The false positive rate and false negative rates are defined as:

False Positive Rate = $\frac{N_{ls}}{N_l}$

False Negative Rate = $\frac{N_{sl}}{N_s}$

To reduce random variation, we use 10-fold cross validation for our experiments. In 10-fold cross validation, the input dataset (emails) is portioned into 10 equal size subparts. Of the 10 subparts, 9 subparts are used as training data, and the remaining 1 part is used for testing. This process is then repeated 10 times, with each of the 10 subparts used exactly once as the testing data. The 10 results then can be averaged to get a single estimation.

In this experiment, the number k (length of substrings) ranges from 3 to 9 for all 4 combinations of enabled/disabled stop-list and lemmatizer datasets. Table 5.2 shows the experiment results of the k-spectrum SVM spam classifier:

Dataset	k	False Positive Rate	False Negative Rate	Spam Precision	Spam Recall	Accuracy
Lemmatizer Enabled Stop- list Enabled	3	0.0054	0.0416	0.9726	0.9584	0.9886
	4	0.0046	0.0291	**0.9770**	0.9709	0.9914
	5	0.0046	0.0312	0.9769	0.9688	0.9910
	6	0.0062	0.0353	0.9687	0.9647	0.9889
	7	0.0070	0.0520	0.9641	0.9480	0.9855
	8	0.0075	0.0644	0.9615	0.9356	0.9831
	9	0.0066	0.0665	0.9656	0.9335	0.9834
Lemmatizer Disabled Stop- list Enabled	3	0.0062	0.0353	0.9687	0.9647	0.9889
	4	0.0041	0.0249	0.9791	0.9751	0.9924
	5	0.0033	0.0291	**0.9832**	0.9709	0.9924
	6	0.0058	0.0312	0.9708	0.9688	0.9900
	7	0.0058	0.0541	0.9701	0.9459	0.9862
	8	0.0075	0.0769	0.9610	0.9231	0.9810
	9	0.0075	0.0728	0.9612	0.9272	0.9817
Lemmatizer Disabled Stop- list Disabled	3	0.0046	0.0270	0.9770	0.9730	0.9917
	4	0.0054	0.0208	0.9731	0.9792	0.9920
	5	0.0033	0.0229	**0.9833**	0.9771	0.9934
	6	0.0046	0.0270	0.9770	0.9730	0.9917
	7	0.0046	0.0312	0.9769	0.9688	0.9910
	8	0.0054	0.0499	0.9723	0.9501	0.9872
	9	0.0058	0.0541	0.9701	0.9459	0.9862
Lemmatizer Enabled Stop- list Disabled	3	0.0037	0.0249	**0.9812**	0.9751	0.9927
	4	0.0046	0.0208	0.9772	0.9792	0.9927
	5	0.0037	0.0249	**0.9812**	0.9751	0.9927
	6	0.0046	0.0333	0.9769	0.9667	0.9907
	7	0.0037	0.0333	0.9810	0.9667	0.9914
	8	0.0050	0.0478	0.9745	0.9522	0.9879
	9	0.0070	0.0561	0.9639	0.9439	0.9848

*Best spam precision in bold

Table 5.2 Results from the Ling-Spam Corpus

Figures 5.1 – 5.4 show the spam precision and recall rates for all 4 data sets. In this report, we focused on Spam Precision as the most important fact since blocking a legitimate

message or letting it go to the Junk folder is more severe than classifying a spam as a legitimate message. The highest precision rates are emphasized in table 5.1 and figures 5.1 – 5.4 for each data set. Except for data set 1 (Lemmatizer enabled stop-list enabled), all data sets achieve the highest precision rate at k = 5.

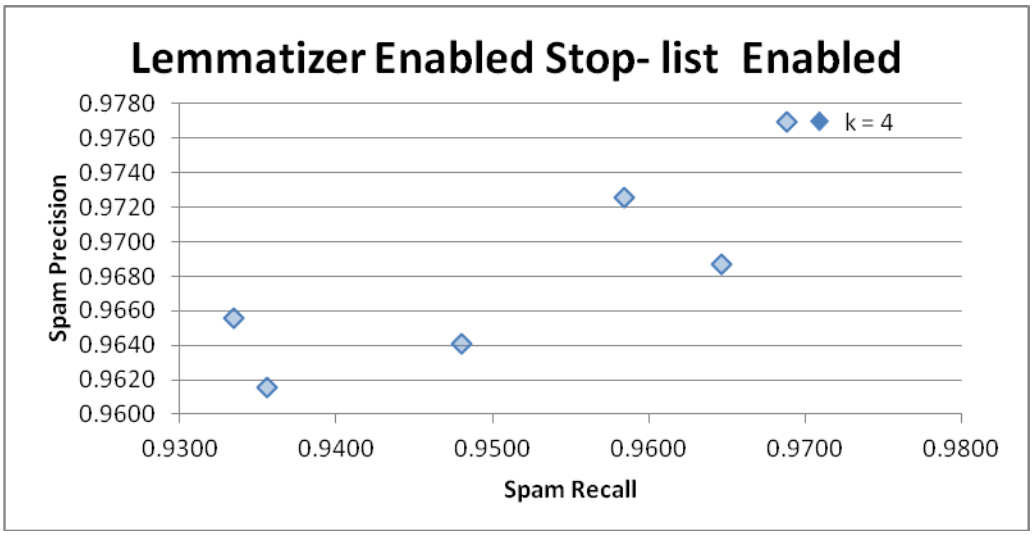
Using a suffix tree to construct the feature space, all occurrences of a certain substring of length n can be found in O(n + k) time, which is totally independent of the size of the text. Any substrings can be found in time proportional to its length alone and after only spend linear time preprocessing the text, which is the prime motivation for using suffix trees to count substring matches[25]. Table 5.3 shows performance metrics on the speed of our spam classifier. The time spent on spam classification and testing process (10 cross-fold validation) increases as k increases. When k is 4, the feature space construction process by using suffix tree is fastest for all four datasets.

Dataset	k	Time of Feature Space Construction	Time of Spam Classification and Test
Lemmatizer Enabled Stop- list Enabled	3	76	12
	4	78	23
	5	80	42
	6	84	96
	7	90	125
	8	90	156
	9	95	186
Lemmatizer Disabled Stop- list Enabled	3	89	11
	4	74	25
	5	81	40
	6	80	64
	7	90	106
	8	91	128
	9	95	165
Lemmatizer Disabled Stop- list Disabled	3	149	60
	4	136	65
	5	146	83
	6	154	100
	7	166	137
	8	184	181
	9	193	243
Lemmatizer Enabled Stop- list Disabled	3	122	20
	4	120	22
	5	135	38
	6	131	67
	7	137	101
	8	141	148
	9	151	195

Table 5.3 Performance metrics

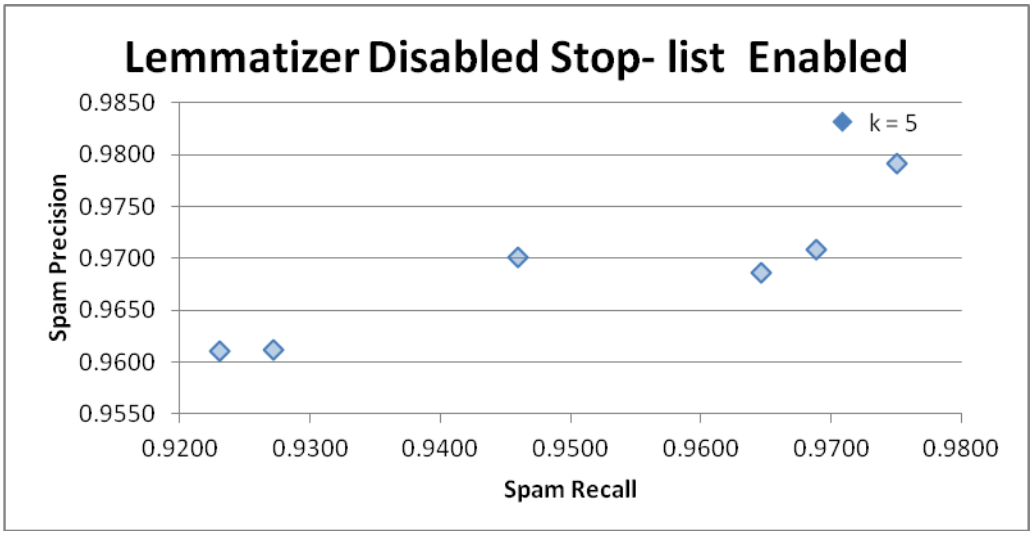
Our best configuration is achieved with k = 5 and both Lemmatizer and stop-list disabled (98.33% spam precision and 97.71% spam recall), so the effect of stop-list and lemmatizer are negligible if using our k-spectrum SVM classifier. The best estimation

result from paper [28] is 100% spam precision and 63% spam recall. Our spam precision is slightly lower than the one from paper [28], but our spam recall and accuracy are significantly higher.



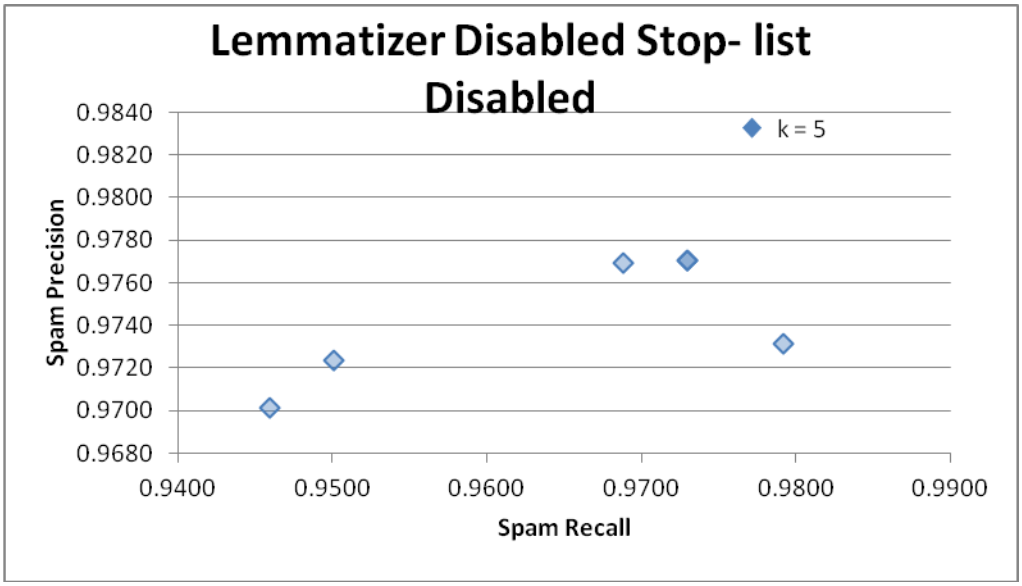
* The highest precision rate is at k = 4

Figure 5.1 Spam Recall vs. Spam Precision (Lemmatizer enabled stop-list enabled)



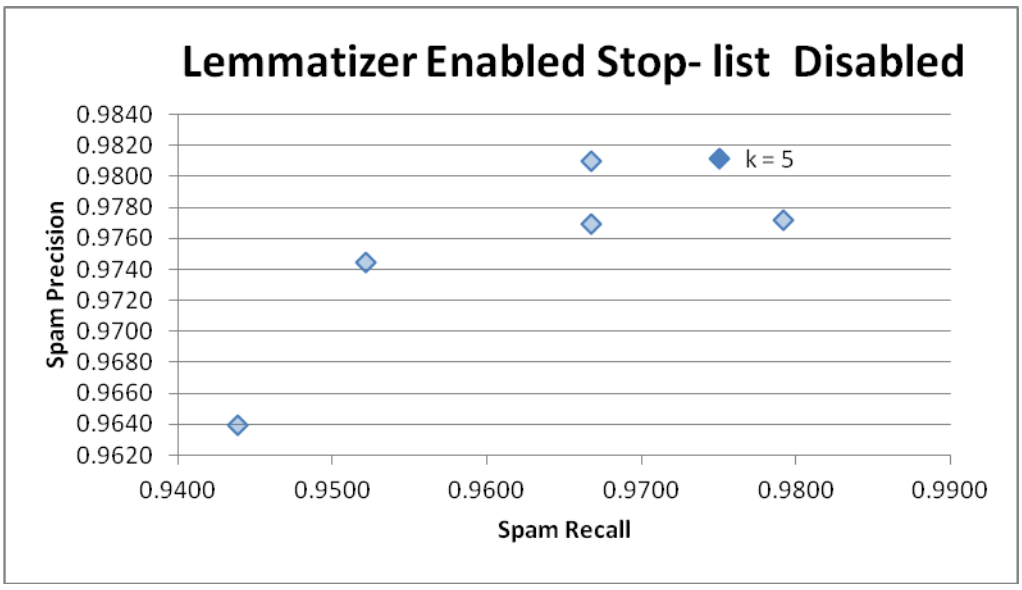
* The highest precision rate is at k = 5

Figure 5.2 Spam Recall vs. Spam Precision (Lemmatizer disabled stop-list enabled)



* The highest precision rate is at k = 5

Figure 5.3 Spam Recall vs. Spam Precision (Lemmatizer disabled stop-list disabled)



* The highest precision rate is at k = 5

Figure 5.4 Spam Recall vs. Spam Precision (Lemmatizer enabled stop-list disabled)

5.2 Experiments with the Enron Corpus

In this section, our testing is performed on the Enron corpus, which is composed of emails extracted from the mailboxes of 6 different users in the Enron Corporation. The details of the Enron Corpus have already been discussed in the previous chapter.

The following table presents the performance of the k-spectrum SVM classifier. We still use the same measures as last section: false positive rate, false negative rate, spam precision, spam recall, and accuracy. The table also includes the testing results of two other classifiers from [35]: a basic Naïve Bayesian classifier and an SVM classifier that uses bag of words (BOW) as its features.

Dataset	**k**	**False Positive rate**	**False Negative rate**	**Spam Precision**	**Spam Recall**	**Accuracy**
	2	0.02968	0.07333	0.92729	0.92667	0.95766
	4	0.02097	0.04533	**0.94897**	0.95467	0.97196
	6	0.02233	0.03000	0.94664	0.97000	**0.97545**
	8	0.02832	0.02467	0.93363	**0.97533**	0.97273
	Bayesian	/	/	0.8509	0.9133	0.9286
Enron1	SVM(BOW)	/	/	0.8741	0.8333	0.9170
	2	0.02637	0.07152	0.92354	0.92848	0.96210
	4	0.00825	0.02406	0.97594	**0.97594**	**0.98771**
Enron2	6	0.00619	0.03543	**0.98163**	0.96457	0.98634

	8	0.00665	0.05481	0.97990	0.94519	0.98105
	Bayesian	/	/	0.9757	0.8000	0.9438
	SVM(BOW)	/	/	0.9067	0.9067	0.9523
	2	0.02592	0.06667	0.93085	0.93333	0.96299
	4	0.00573	0.03733	0.98432	**0.96267**	**0.98567**
	6	0.00324	0.05133	0.99095	0.94867	0.98367
	8	0.00249	0.08467	0.99277	0.91533	0.97515
	Bayesian	/	/	**1**	0.5733	0.8841
Enron3	SVM(BOW)	/	/	0.9648	0.9133	0.9674
	2	0.04600	0.01422	0.98468	0.98578	0.97783
	4	0.04067	0.00644	0.98654	0.99356	0.98500
	6	0.05200	0.00467	0.98288	0.99533	0.98350
	8	0.07067	0.00422	0.97689	**0.99578**	0.97917
	Bayesian	/	/	**1**	0.9467	0.96000
Enron4	SVM(BOW)	/	/	**1**	0.9889	**0.9917**
	2	0.05000	0.02367	0.97952	0.97633	0.96870
	4	0.02533	0.00735	0.98969	0.99265	0.98744
	6	0.02400	0.00490	0.**99725**	0.99510	**0.98957**
	8	0.03333	0.004082	0.98652	**0.99592**	0.98744
	Bayesian	/	/	0.9880	0.8967	0.9189
Enron5	SVM(BOW)	/	/	0.9970	0.8940	0.9228
	2	0.08400	0.03156	0.97190	0.96844	0.95533
	4	0.03000	0.00844	**0.99002**	0.99156	**0.98617**
	6	0.05733	0.00489	0.98116	0.99511	0.98200
	8	0.06800	0.00467	0.97773	**0.99533**	0.97950
	Bayesian	/	/	0.9898	0.8600	0.8833
Enron6	SVM(BOW)	/	/	0.9528	0.8978	0.9005

*Best results in bold

Table 5.4 Results from Enron Corpus

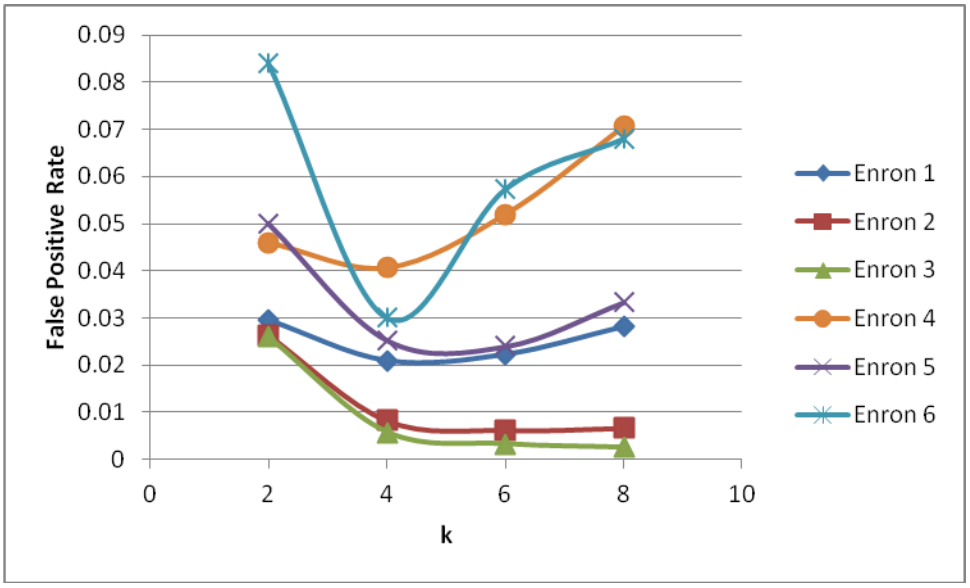


Figure 5.5 False positive rates (Enron Corpora)

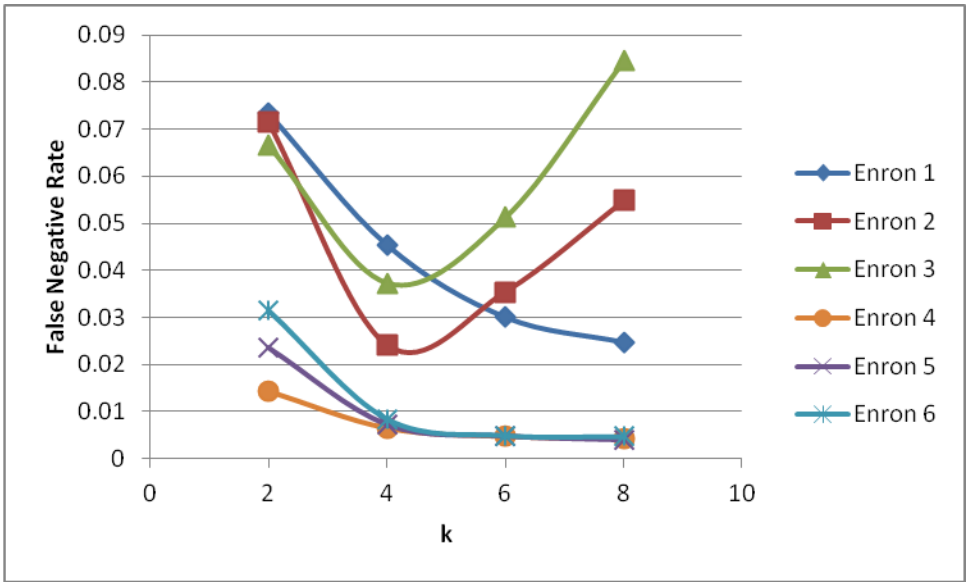


Figure 5.6 False negative rates (Enron Corpora)

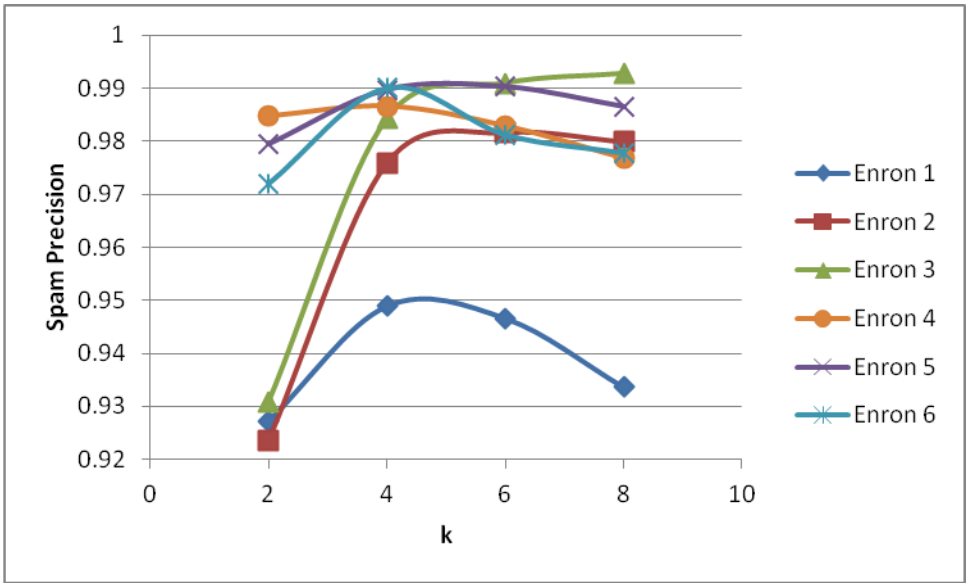


Figure 5.7 Spam Precision (Enron Corpora)

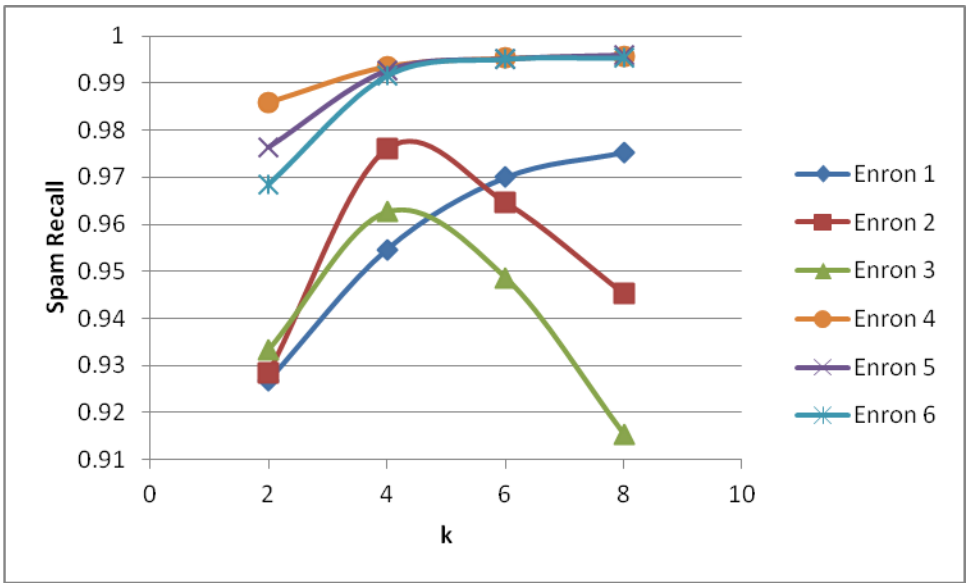


Figure 5.8 Spam Recall (Enron Corpora)

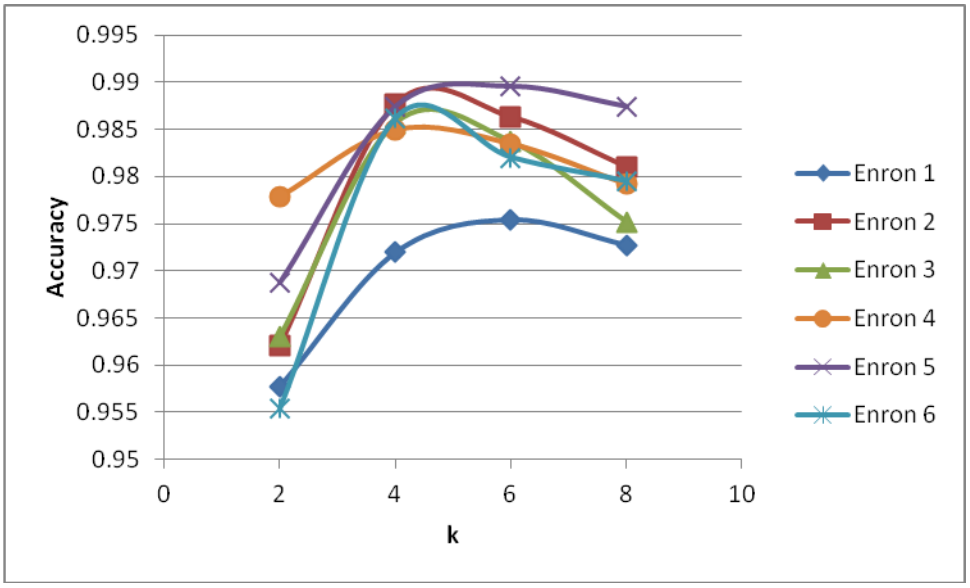


Figure 5.9 Accuracy rates (Enron Corpora)

All of the performance measures on Enron 1, Enron 2, Enron 5, and Enron 6 are higher than the Na ïve Bayesian classifier and BOW-based SVM classifier. For Enron 3, the Na ïve Bayesian classifier has the highest spam precision rate (100%), but our k-spectrum SVM d SVM kernel has a better performance in terms of spam recall and accuracy rates. For Enron 4, the Na ïve Bayesian classifier and BOW-based SVM classifier have higher spam recall and accuracy rates, but our k-spectrum SVM classifier has a better spam recall rate.

Overall, the experiments indicate that our k-spectrum SVM spam filter performs better than the Na ïve Bayesian classifier and BOW-based classifier, because it achieves an accuracy rate higher than 97% for the entire Enron corpus, which is obviously higher than the other two classifiers. Also, the k-spectrum SVM spam filter achieved better results in most datasets (Enron 1, 2, 5, and 6).

5.3 Experiments with TREC 2006 Chinese Corpus

Unlike English, many Asian languages do not use white spaces to mark word boundaries. For example, Chinese sentences are written as continuous characters, so linguistic knowledge is required to divide a Chinese email message into words. An alternative way of segmenting a Chinese email is using n-gram to divide the whole string into units of fixed length (n). Our String-based approach uses substrings of fixed length (n-grams) as features for the SVM classifier, so it can also be used to classify email messages that contain Chinese Characters.

In this section, the k-spectrum SVM spam classifier is evaluated on the TREC 2006 Chinese corpus, which consists of 42854 spam messages and 21766 legitimate emails. In this experiment, the 10-fold cross fold evaluation is also used, and the value of k (length of substrings) is tested from 2 to 7.

The following table and figure show the experimented results of the k-spectrum SVM spam classifier on the TREC 2006 Chinese corpus:

k	Nss	Nls	Nsl	false postive rate	false negative rate	spam precision	spam recall	Accuracy
2	42712	304	142	0.01397	0.00331	0.99293	0.99669	0.99310
3	**42720**	**152**	**134**	**0.00698**	**0.00313**	**0.99645**	**0.99687**	**0.99557**
4	42712	169	142	0.00776	0.00331	0.99606	0.99669	0.99519
5	42731	244	123	0.01121	0.00287	0.99432	0.99713	0.99432
6	42747	388	107	0.01783	0.00250	0.99100	0.99750	0.99234
7	42751	1023	103	0.04700	0.00240	0.97663	0.99760	0.98258

*Best result in bold

Table 5.5 Results from TREC 2006 Chinese corpus

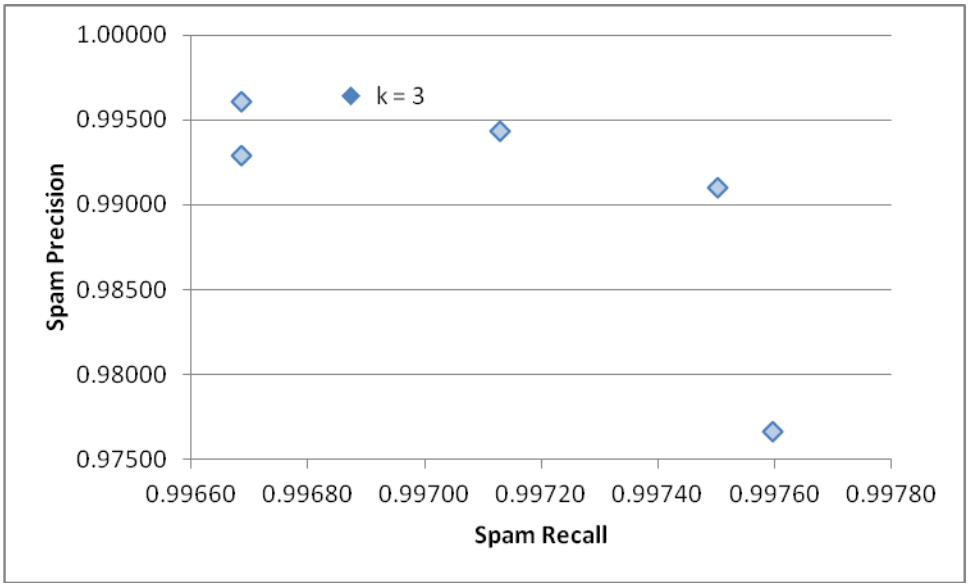


Figure 5.10 Spam Recall vs. Spam Precision (TREC 2006 Chinese corpus)

The testing results on Chinese corpus are also promising. All spam precision rates except the last one (k = 7) are above 0.99; the best spam precision (0.99645) and accuracy rates (0.99557) are achieved at k = 3. When k reaches 7, the spam precision rate drops significantly; the number of legitimate emails that are misclassified as spams is 1023, which leads to a higher false positive rate of 0.047. However, the highest spam rate is achieved at k = 7. There is usually a natural tension between spam and legitimate email misclassification rates. A spam filter usually improves one at the expense of the other.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this report, we have presented and implemented a contiguous string-based approach, which applies SVM and k-spectrum kernel [23] in combination to the spam classification problem. The k-spectrum kernel was originally proposed for biological data analysis in [23], but we believe that our experiments provide evidence that the k-spectrum kernel in conjunction with SVM could offer an effective and accurate alternative to spam filtering as well.

In order to apply the k-spectrum SVM spam filter on large datasets efficiently, we used suffix trees to construct the feature space for the SVM classifier. We conducted experiments on the Ling-Spam corpus, the Enron corpus, and the TREC 2006 Chinese corpus, and used five important measures to evaluate the performance of k-spectrum SVM spam classifier: false positive rate, false negative rate, spam precision, spam recall, and accuracy rate.

The following are the observations made from the research conducted along with the important contributions made by the report:

- Our experimental results demonstrate that the k-spectrum SVM spam filter classifier outperforms the SVM classifier that uses the bag-of-word technique to represent features and is competitive with the well-known Naive Bayesian approach.

- The k-spectrum SVM spam filter classifier has obtained a high accuracy rate of 0.99557 with an acceptably false positive rate of 0.00698 on the TREC 2006 Chinese corpus, which provides evidence that it is a very effective approach to classifying emails that are written in languages that do not use white space as delimiters.

- The k-spectrum SVM spam filter does not require any knowledge of the language being used and can still provide great performance.

- The k-spectrum SVM spam filter has achieved significantly higher spam recall and accuracy rates on almost all datasets we have tested.

6.2 Future Work

Some spam filters analyze the content of the message based on known spam emails and information provided by email users, and then assign a spam score based on the probability that the message is Spam. The higher the score the more likely it is to be spam. They may also allow users to set a threshold value which is treated as an adjustable parameter that governs the tradeoff between the spam detection and false positive rates. The threshold value specified by the user will then be compared to each message's spam score. Any email with a spam score greater than or equal to this value will be considered as a spam. For example, if you have the threshold set to 80, then any message with a spam score that is greater than or equal to 80 will be considered as a spam. Appropriate actions will then be taken, such as sending the message to the Inbox mail folder, rejecting it, or sending it to the Junk mail folder.

However, the k-spectrum SVM classifier does not provide a probability for the email messages; the email is classified as either spam or non-spam. In this case, users will not be able to set their own spam thresholds to determine how emails should enter their email system. Therefore, it would be useful to convert the outputs of the SVM classifier into probabilities. Joseph Drish [37] proposed a histogram technique known as binning to convert the classification outputs into probability values. John C. Platt [36] presented another approach to extract probabilities by first training the SVM classifier and then training the parameters of an additional sigmoid function to map the outputs into probabilities. Both methods should be studied and implemented to map our k-spectrum SVM classifier outputs into probabilities, which will definitely improve the flexibility of our spam filter.

Glossary of Terms

Lemmatizer

A lemmatizer is used to convert each word to its base form.

N-Gram

N-gram model which represents texts as a high dimensional feature vector, and each feature corresponds to a contiguous substring of length *n*.

SSK

String Subsequence Kernel (SSK) is based on searching of the same non-contiguous subsequences in input strings, and according to all occurrences it returns the similarity rate.

Non-contiguous String

A non-contiguous string is a string which is formed from the original string by deleting some or none of the characters without disturbing the relative positions of the remaining characters.

K-spectrum SVM Classifier

In our report, we implemented the k-spectrum SVM Classifier to classify spams. It is based on k-spectrum kernel and SVM approach.

Bibliography

[1] Messaging Anti-Abuse Working Group, Email Metrics Report Third and Fourth Quarters 2009, http://www.maawg.org/about/MAAWG20072Q_Metrics_Report.pdf

[2] J.M. Mart´ınez-Otzeta, B. Sierra, E. Lazkano, M. Ardaiz, and E. Jauregi, "Edited naive bayes", Ibero-American Journal of Artificial Intelligence, 10(31), pp 63-69 (2006).

[3] Jiansheng Wu , Tao Deng, Research in Anti-Spam Method Based on Bayesian Filtering, Proceedings of the 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, p.887-891, December 19-20, 2008

[4] Md. Rafiqul Islam , Morshed U. Chowdhury , Wanlei Zhou, An Innovative Spam Filtering Model Based on Support Vector Machine, Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06), p.348-353, November 28-30, 2005

[5] Chih-Chin Lai , Ming-Chi Tsai, An Empirical Performance Comparison of Machine Learning Methods for Spam E-Mail Categorization, Proceedings of the Fourth International Conference on Hybrid Intelligent Systems, p.44-48, December 05-08, 2004

[6]P. Jackson and I. Moulinier. Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization. John Benjamins Publishing Co, 2002

[7] Michael W. Berry and Malu Castellanos. Document Representation and Quality of Text: An Analysis, Springer-Verlag London Limited, 2008

[8] Huma Lodhi , Craig Saunders , John Shawe-Taylor , Nello Cristianini , Chris Watkins, Text classification using string kernels, The Journal of Machine Learning Research, 2, p.419-444, 3/1/2002

[9] Fuchun Peng , Dale Schuurmans , Shaojun Wang, Augmenting Naive Bayes Classifiers with Statistical Language Models, Information Retrieval, v.7 n.3-4, p.317-345, September-December 2004

[10] A. N. Aizawa. Linguistic techniques to improve the performance of automatic text categorization. In Proceedings of the 6th Natural Language Processing Pacific Rim Symposim (NLPRS), pages 307--314, Tokyo, Japan, 2001.

[11] Dell Zhang , Wee Sun Lee, Extracting key-substring-group features for text classification, Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, August 20-23, 2006, Philadelphia, PA, USA

[12] Cavnar B. W., Trenkle M. J. N-Gram-Based Text Categorization. Proceedings of SDAIR-94, 3rd Annual Symposium on Document and Information Retrieval, 1994

[13] PU1 Corpus, http://www.kdnet.org/kdnet/control/dataset_details?item_id=40

[14] TREC 2006 Corpus, http://plg.uwaterloo.ca/~gvcormac/treccorpus06/about.html

[15] Ion Androutsopoulos , John Koutsias , Konstantinos V. Chandrinos , Constantine D. Spyropoulos, An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages, Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, p.160-167, July 24-28, 2000, Athens, Greece

[16] Le Zhang , Jingbo Zhu , Tianshun Yao, An evaluation of statistical spam filtering techniques, ACM Transactions on Asian Language Information Processing (TALIP), v.3 n.4, p.243-269, December 2004

[17] Drucker H, Wu D, Vapnik V (1999) Support vector machines for spam categorization. IEEE Trans NeuralNetw 10(5): 1048–1054

[18] Simon Tong , Daphne Koller, Support vector machine active learning with applications to text classification, The Journal of Machine Learning Research, 2, p.45-66, 3/1/2002

[19] Medlock B, An adaptive approach to spam filtering on a new corpus. In: Proceedings of the third conference on email and anti-spam, CEAS'2006

[20] Kanaris, I., K. Kanaris, and E. Stamatatos, Spam Detection Using Character N-grams, In G. Antoniou, G. Potamias, C. Spyropoulos, D. Plexousakis (Eds): Advances in Artificial Intelligence: Proceedings of the 4th Hellenic Conference on AI (SETN 2006), Springer LNCS, 3955, pp. 95–104, 2006.

[21] B. Scholkopf and A. J. Smola. Learning with Kernels. MIT Press, Cambridge, MA, 2002.

[22] Fabrizio Sebastiani, Machine learning in automated text categorization, ACM Computing Surveys (CSUR), v.34 n.1, p.1-47, March 2002

[23] Leslie, C.; Eskin, E.; Noble, W.S. (2002), *The spectrum kernel: A string kernel for SVM protein classification*, **7**, pp. 566–575

[24] N. Cristianini and J. Shawe-Taylor *Kernel Methods for Pattern Analysis*, 2004 :Cambridge Univ. Press

[25] Gusfield, Dan (1999) [1997]. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. USA: Cambridge University Press. pp. 4. ISBN 0-521-58519-8.

[26] E. Ukkonen. On-line construction of suffix trees. Algorithmica, 14(3), 1995.

[27] Seewald A, Kleedorfer F. Technical Report. Osterreichisches Forschungsinstitut fur Artificial Intelligence; Wien, TR-2005–13: 2005. Lambda pruning: an approximation of the string subsequence kernel.

[28] Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., Paliouras, G., & Spyropoulos, C. D. (2000). An evaluation of naive bayesian anti-spam filtering. arXiv preprint cs/0006013

[29] Gordon Cormack, TREC 2006 Spam Track Overview, 2006

[30] BNC frequency lists, available from ftp://ftp.itri.bton.ac.uk/pub/bnc

[31]V. Metsis, I. Androutsopoulos and G. Paliouras, "Spam Filtering with Naive Bayes - Which Naive Bayes?". Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS 2006), Mountain View, CA, USA, 2006.

[32] Fan, Rong-En, et al. "LIBLINEAR: A library for large linear classification." The Journal of Machine Learning Research 9 (2008): 1871-1874.

[33] Generalized Suffix Tree library, https://github.com/abahgat/suffixtree

[34] Ukkonen, E. (1995). On-line construction of suffix trees. Algorithmica, 14(3), 249-260.

[35] Almeida, Tiago A., and Akebo Yamakami. "Content-based spam filtering." Neural Networks (IJCNN), The 2010 International Joint Conference on. IEEE, 2010.

[36] Platt, John. "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods." *Advances in large margin classifiers* 10.3 (1999): 61-74.

[37] Drish, Joseph. "Obtaining calibrated probability estimates from support vector machines." *Final Project for CSE 254: Seminar on Learning Algorithms*. 2001.

[38] Jones, Sydney, and Susannah Fox. *Generations online in 2009*. Washington, DC: Pew Internet & American Life Project, 2009.

[39] Geisser, Seymour (1993). *Predictive Inference*. New York, NY: Chapman and Hall. ISBN 0-412-03471-9.

Curriculum Vitae

Full name: Ming Yang

Universities attended:

- September 2002 – University of New Brunswick – BSCCS (Bachelor of Science in Computer Science)
- Winter 2008 – Present – University of New Brunswick – MCS (Master of Computer Science)

Employment History:

- August 2010 – Present – Computer of Generated Solution, Saint John, New Brunswick – Java developer and software consultant
- October 2006 – October 2007 - Computer of Generated Solution, Saint John, New Brunswick – Courseware developer