E-mail Filtering within Outlook using Machine Learning Techniques

By
Xiaofei Lin (276592)

Supervised by Professor Tony White

An honor’s project report submitted to the Faculty of Computer Science in partial fulfillments of the requirements of course 95.495

School of Computer Science
Carleton University
Ottawa, Ontario

April 28, 2003

© Copyright 2003, Xiaofei Lin
Abstract

While most mail reader applications allow users to file messages into folders and filter incoming electronic mail (e-mail) by user defined rules, in practice these tasks tend to be tedious and have their limitations. Mail filtering is a method for the organization of e-mails. This honor project involves creating a mail filter in Microsoft Outlook using Machine Learning Techniques.

In this report, the built-in mail filters in Outlook are critiqued, how Machine Learning techniques could improve upon it without eliminating the benefits that already provided is described in detail. Afterwards a application used COM add-in method for Microsoft Outlook that help manage the inbox by automatically classifying e-mail based on user created folders is developed. Furthermore, we conducted performance on e-mail classification and point out some of the issues involved with creating Machine Learning based mail filter. We conclude that the newly developed COM add-in application eliminates some of the problems with the current mail filters used in Microsoft Outlook while still maintaining high classification accuracy.
Acknowledgments

I would like to thank the following people or groups for their help with this project. Without their help I would have had a difficult time completing this project.

- Professor Tony White, his time, guidance, ideas, advice and comments are very much appreciated
- Experts exchange website and Microsoft help and support group, their valuable documents and feedback always give so much help on Visual Basic programming and Outlook extension.
- Miss. Grace Li for her encouragement and suggestions.
- Miss. Heather Crooks for making this report a better written English document
# Table of Contents

1 INTRODUCTION ................................................................................................................................ 5

2 MOTIVATION AND OBJECTIVE ................................................................................................... 7
   2.1 MODERN MAIL ORGANIZERS IN OUTLOOK...................................................................................... 7
   2.2 OBJECTIVE ....................................................................................................................................... 8

3 BACKGROUND REVIEW ................................................................................................................. 9
   3.1 MACHINE LEARNING ....................................................................................................................... 9
   3.2 AIM TEXT CLASSIFIER ALGORITHM ............................................................................................. 10
   3.3 COM ADD-IN ....................................................................................................................................... 12

4 DESIGN ANALYSIS ................................................................................................................ .......... 14
   4.1 DESIGN OVERVIEW ....................................................................................................................... 14
   4.2 E-MAIL FILTER OBJECTS ............................................................................................................... 15

5 IMPLEMENTATION ................................................................................................................. ....... 17
   5.1 AIM TEXT CLASSIFIER IMPLEMENTATION .................................................................................... 17
      5.1.1 Feature Extraction from e-mail ............................................................................................ 17
      5.1.2 Training Data Storage .......................................................................................................... 19
   5.2 COM ADD-IN IMPLEMENTATIONS ................................................................................................. 21
      5.2.1 My Add-in Implementation ................................................................................................... 21
      5.2.2 Challenge...................................................................................................................... ........ 25

6 EXPERIMENTS .................................................................................................................... ............. 27
   6.1 EXPERIMENTAL DESIGN ................................................................................................................. 27
   6.2 RESULTS....................................................................................................................................... 28
      6.2.1 Classification Accuracy........................................................................................................ 28
      6.2.2 Observations................................................................................................................... ...... 30

7 CONCLUSION ..................................................................................................................... .............. 32

8 FUTURE WORK .................................................................................................................... ............ 33

9 HOW TO USE THE E-MAIL FILTER ........................................................................................... 35
   9.1 INTRODUCTION.............................................................................................................................. 35
   9.2 USER’S GUIDE.................................................................................................................................... 35
      9.2.1 Compatibility .................................................................................................................. ...... 35
      9.2.2 Installation................................................................................................................... ......... 35
      9.2.3 Program Operations ............................................................................................................. 36

10 REFERENCES ............................................................................................................................... 40
1 Introduction

As our lives become ever increasingly tied to the online world, the volume of e-mail coming into our inboxes has also been increasing steadily. To facilitate the retrieval of messages weeks, months or years after their original receipt, most mail reader applications allow users to organize their message into user-defined folders. Typically, a user may maintain a large number of separate folders, some applications permit these folders to be arranged hierarchically. The need to make this task easier is critical. One of the current solutions of most applications is to provide a simple interface for filing messages into folders. A list of folders is displayed and users are allowed to select the destination folder and move to it. Another popular solution consists of using a mail filtering program that can sort incoming mails based on user-specified rules. Such mail filters do help to relieve the organizational dilemma, but they are still a long stretch from solving the problem.

This project is to implement a simple e-mail filter based on the COM (Microsoft’s Component Object Model) add-in method [9], would act similar to an intelligent personal assistant by reducing the cognitive burden and the time required for a user to organize e-mails into folders within Outlook e-mail reader application. The e-mail filter, using a text classifier algorithm which is a topic in Machine Learning technique [4] that adapts dynamically to a user’s observed mail-filing habits, predicts the top three folders that are most likely to be appropriate for a given message and provides shortcut buttons that facilitate filing the message into one of its predicted folders. When one of the predicted folders is correct, the user’s task is greatly simplified. Rather than having to derive a best
choice from a large set of folders, the user can merely confirm one of the predictions and move the message to the folder by a single mouse click. Certainly, the predictions will not always include the desired choice. The prediction accuracy of this e-mail filter is roughly 80% to 95% even for users with as many as 20 folders. However, even though none of the predictions are correct, users are still able to use the original interface provided by Outlook to file the message into the appropriate folder.

This report consists of the following parts:
Section 2 justifies the motivation of this project and lists the objective.
Section 3 provides background knowledge on Machine Learning Technique, Text Classifier Algorithm, and COM add-in method to extend Outlook.
Section 4 illustrates the design of the e-mail filter.
Section 5 provides implementation details. This includes: Algorithm implementation and COM Add-in implementation.
Section 6 presents experimental results and analyzes the performance data.
Section 7 concludes the report
Section 8 points out the potential for future work.
Section 9 is the user’s manual.
Section 10 lists references.
2 Motivation and Objective

Most e-mail reader applications provide options allowing user to organize their messages. These organizers may still make the classifying task tedious, time consuming and have the limitations for user. This section will discuss the motivation of this project by critiquing the existing mail organizer in Microsoft Outlook and present the objective of this project.

2.1 Modern Mail Organizers in Outlook

Outlook includes a simple interface for filing messages into folders. It provides a MoveToFolder button with a pop-up dialog box that allows users to select the destination folder. It also displays a list of folders next to the message and allows the user to move a message into a folder using drag-and-drop action. Regardless of the detailed arrangement of the user interface, in practice the cognitive effort required to decide upon an appropriate folder and locate the icon or menu item representing it is substantial enough to discourage users from filing at all, particularly when faced with the prospect of repeating the exercise dozens of times per day or a large number of folders.

Outlook also provides a mail filter to allow the user to concoct a set of rules by which incoming e-mail is classified. For example, rules that allow the identification of the content of an e-mail message, and consequently, to classify and organize the e-mail in the context of the mailbox. Such tools, however, are mainly human-centered in the sense that the user is required to manually describe rules and keyword lists that can be used to
recognize the relevant features of messages. Such an approach has the following disadvantages:

- It is inadequate for a large volume of e-mails which may contain too many distinct and possibly overlapping concepts. As a consequence, the process of manually detecting the main contents of mail messages occurring in a mailbox can be time-consuming.
- It does not accommodate to requirement changes that usually occur when users need to periodically re-organize their mailbox. The contents of e-mail messages may change with a given frequency, and consequently the user is required to periodically re-organize their filter.

In general, these two e-mail organizers provided in Outlook are still not simple enough for users to filter their e-mails

2.2 Objective

The objective of this project is to develop an e-mail filter that solves the disadvantages of Microsoft Outlook 2002 listed above in section 2.1. This e-mail filter is based on Machine Learning Technique which can study user’s pre-filling habit automatically, specifically AIM text classifier algorithm, and implemented to extend Microsoft Outlook by using COM add-in. The related concepts of Machine learning technique, AIM text classifier and COM add-in will be discussed in section 3.
3 Background Review

3.1 Machine Learning

The goal is to devise an accurate mail filter, which eliminates the aforementioned problems of modern mail filters in Outlook. One way in which this might be possible is through the application of Machine Learning methods [4]. Machine Learning is a subfield of Artificial Intelligence and is usually the study of computer algorithms that improve automatically through experience. Text classification is one approach for Machine Learning, and it has been shown to perform quite well in eliminating or reducing the problems with modern mail filters in Outlook.

Machine Learning has the potential to eliminate many of the problems of current mail filters, and there has already been strong evidence that such an algorithm could accurately classify personal e-mail. SwiftFile[6] and MailCat [7] use AIM text classifier [8], which is a TF-IDF style classifier [2] to score high classification accuracy on the task of classifying e-mail into folders in Lotus Notes. David Lewis and Kimberly Knowles use a TF-IDF classifier to successfully match e-mail messages with their “parents” [11]. William Cohen tests TF-IDF and a rule-based classifier on a corpus of his own e-mail [1]. Studies which regarding the classification of e-mail documents generally show Machine Learning algorithms achieving an accuracy of 80% or better, even when the classification
task is across a wide range of classes. Thus, Machine Learning algorithms are quite likely to be capable of handling the task of filtering personal e-mail.

3.2 AIM Text Classifier Algorithm

TF-IDF \[2\] are term weighting algorithms, which are used in many approaches to ranked retrieval for the relation between language modeling algorithms. It is a Vector-based classifier and support fast re-training. As we have mentioned, TF-IDF achieves a high accuracy for classifying e-mail. Previous results show TF-IDF to be an effective Machine Learning algorithm which achieves high accuracy rates for classifying e-mail.

The text classifier of this e-mail filter uses AIM text classifier algorithm. AIM \[8\] text classifier is a TF-IDF style classifier developed at IBM’s Almaden research laboratory. The algorithm of AIM is more simple and easy to implement.

AIM represents each message \( M \) as a word-frequency vector \( F(M) \), in which each component \( F(M, w) \) represents the total number of times the word \( w \) appears in \( M \). Each folder \( F \) is represented using a weighted word-frequency vector \( W(F, w) \). Several steps are involved in computing \( W(F, w) \) is computed by summing the word-frequency vectors for each message in the folder:

\[
F(F, w) = \sum_{M \in F} F(M, w) \quad (1)
\]

This folder centroid vector is then converted to a weighted word-frequency vector using the TF-IDF principle: the weight assigned to a word is proportional to its frequency in the
folder and inversely proportional to its frequency in other folders. We define \( FF(F, w) \) to be the fractional frequency of word \( w \) among messages contained within folder \( F \), or the number of times word \( w \) occurs in folder \( F \) divided by the total number of words in \( F \):

\[
FF(F, w) = \frac{F(F, w)}{\sum_{w' \in F} F(F, w')}
\]  

(2)

The definition of term frequency \( TF(F, w) \) used by AIM is

\[
TF(F, w) = FR(F, w)/FR(A, w)
\]  

(3)

where \( A \) represents the set of all messages (the entire database of organized mail). We define the document frequency \( DF(w) \) to be the fraction of folders in which the word \( w \) appears at least once. The definition of inverse document frequency \( IDF(w) \) used by AIM is:

\[
IDF(w) = \frac{1}{DF(w)^2}
\]  

(4)

Finally, AIM combines these two formulas to define the weight for word \( w \) in folder \( F \):

\[
W(F, w) = TF(F, w) \times IDF(w)
\]  

(5)

The weight vectors for each folder are used to classify each new message. When a message \( M \) arrives to be classified, it is first converted into a word-frequency vector \( F(M) \). Then, AIM computes the similarity between \( M \) and the weighted word-frequency vectors for each folder, \( W(F) \). AIM computes the similarity between the message vector and the weighted folder vectors using a variation of cosine distance called \( SIM4 \) [2]:
Here the sums are taken only over the words that are contained within $M$. Finally, AIM takes the three folders with greatest similarity as its predictions.

3.3 COM add-in

Microsoft office 2002 applications have the extensibility to allow developers to develop solutions for them. There are a number of ways to develop the Microsoft office extension, such as VBA or COM add-ins [9]. COM add-ins provide a secure and easy way to develop and distribute the office applications. Since COM add-ins are compiled in .dll files their code is more secure than VBA code, which is distributed as source code. Also, since only one VBA project can be loaded by the application, any changes made to a distributed VBA project will overwrite the previous project. This will destroy any macros or customizations that the user has made. But, COM add-ins do not overwrite the user’s customizations and office application can load many COM add-ins, with little or no unwanted interaction between them. Furthermore, COM add-ins can be created and compiled using a number of different development environments; such as Delphi, VC++, VB5, VB6 and Microsoft Office Developer Edition.

COM add-in has two requirements [3]; first it must register itself as an Office add-in by setting values in the Office section of the registry, and second, it must implement the
IDTExtensibility2 interface. Once Office invokes the add-in Office objects can be used to perform endless actions such as customizing the user interface, creating new documents or creating special rules that evaluate incoming mail messages.

Figure 3-1 Add-ins contain special registry entries and implement the IDTExtensibility2 interface.
4 Design Analysis

4.1 Design Overview

The fundamental design principle of this e-mail filter is simplicity. It aims at providing a substantial benefit to users without demanding anything extra from them. Users should not be required to learn or perform anything new in order to use it. Furthermore, any errors made by this e-mail filter should have no negative impact on the user, and in particular the user should be able to ignore it and use the application as though the new e-mail filter is not present.

According to the basic design principle, this e-mail filter should provide three shortcut buttons for each e-mail message. The shortcut buttons allow the user to quickly move the message into one of the three folders that the text classifier predicts to be the message’s most-likely destinations. The buttons are ordered from left to right, with the leftmost representing e-mail filter’s best guess and the rightmost representing its third-best guess. When one of the three buttons is clicked, the message is immediately move to the indicated folder. The e-mail organization task will then be simply achieved by a single button click.

In order to make predictions by using a text classifier, classifiers often require a large number of training messages before they yield accurate predictions. This e-mail filter uses previously-filed messages as a training corpus to study user’s filing habit. After the
training is complete, the e-mail filter is ready to make useful predictions when a user selects the mail in Inbox to read or open.

The e-mail filter should update the training data when users are constantly changing the folder’s hierarchal structure or the content of the folder. It adapts to changing conditions by using incremental learning. Once the classifier has been trained, the classifier’s model is continuously updated by presenting the messages that have been added to or deleted from each folder to the classifier. The cost of the update is only linear in the length of the message. After updating, the classifier’s predictions are identical to those that would be obtained by training the classifier on the entire mail database.

4.2 E-mail filter objects

According to the overview, this filter works with Outlook and makes predictions using a text classifier. As such, the entire system would be consisting of three objects: Outlook application, an interface to extend Outlook and a Text Classifier. In this project, COM add-in is used as an interface between Outlook and the Text Classifier, and also to extend the Outlook user Interface. The following figure 4-1 reflects the relationship of the components and presents the logic of the whole system.
Figure 4-1 Logic view of the whole system.
5 Implementation

As discussed in the last section, the e-mail filter is implemented as an extension to Outlook. The dependencies on Outlook are displayed in the following aspects: providing mail folders, the messages inside for each folder, and the way for catching user operations through Outlook. In order to make implementations easier and compatible with Outlook, the e-mail filter is developed in Visual Basic 6.0. The development of the e-mail filter is divided into two tasks: the AIM Text Classifier implementation and COM add-in implementation.

5.1 AIM Text Classifier Implementation

5.1.1 Feature Extraction from e-mail

AIM text classifier takes a message consisting of a set of words and tries to categorize the message among a specified set of folders. Therefore, we need to parse an e-mail to a set of words in order to be fed to AIM text classifier. We would like to discuss how to extract the relevant words from a given e-mail as follows.

First, what kind of information from a message should be used to make a correct classification? If we are trying to mimic the user’s preferred method of classification, the user must classify the message according to the user observable elements of the email message. Therefore, only observed information in our learning problem is useful. We thus use only the “From:” and “Subject:” fields as well as the entire text of the message body of each message as inputs for text classifier.
Once we have done this, there are still many refinements we can make to the message which will facilitate the learning process. One such refinement is to provide a *stoplist*. Since we are classifying messages based on the existence of certain words in each message, we can simplify our problem by eliminating those words which appear with high frequency in all documents, regardless of classification. These are words such as “the”, “he”, or “it”. Common pronouns, modifiers, simple adverbs and verbs all fall into this *stoplist*. Since these words appear in almost all documents, we can ignore them without losing information. Essentially this is just noise that does not help with classification. We implement this by using the *stoplist* that is defined in snowball [10].

Another means for refining the text classification problem is to use a process known as *stemming*. The purpose of stemming is to treat words with the same root as being equivalent. For example, the words “serve”, “serves”, and “served” would all be transformed into their root “serve”. We used the Porter Stemming Algorithm [5] and made a dynamic link library with C functions. We call this dynamic link library directly through our implementation.

Finally, we note that it is important to lex the input stream correctly so that we preserve strings like a user’s e-mail address as one token. E-mail addresses are important since we recognize that people often sort messages based on the sender. Thus, we need to use a lexer that does not separate words using punctuation symbols such as [@] or [,.] or which specifically preserves e-mail address. In our implementation, we simply use a lexer that
separates token by white space. Then, ignore all the punctuation symbols which are at the beginning or end of a word. We treat the punctuation symbols in middle of a word as delimitation of the string. And change all upper case letters to lower case. Each message can then be viewed as a set of words.

5.1.2 Training Data Storage

As mentioned in Section 4, the text classifier should learn users’ filing habit through pre-filled messages in order to do classification. Defining what information should be kept as a classifying reference and determining the way to store it are the most important parts in text classifier implementation. The adaptive method would make the whole text classifier implementation easier.

According to Section 3, which described the AIM text classifier algorithm, we know the algorithm makes predications by comparing the similarities between messages and folders. From the algorithm, we realize when a message $M$ is to be classified, the classifier computes on the fly only the weights for words occurring in $M$, since these are all required by the $SIM4$ in equation 6. However, if we keep the weighted word-frequency vector $W(F, w)$ as the classifying reference database, it’s difficult to maintain it when the training data changes, since it cause a change in all the calculations. Therefore, we implemented the AIM text classifier algorithm by keeping the folder’s centroids vector $F(F, w)$ as the classifying reference database. Whenever a new message $M$ is added to a folder $F$, the database updates itself by simply adding the word-frequency vector $F (M, w)$ to it.
Similarly, when a message is removed from a folder, the word-frequency vector for that message is subtracted from the folder’s centroid vector. The database is stored in a dictionary data structure as in Table 5-1.

<table>
<thead>
<tr>
<th>Key: (string)</th>
<th>Folder1 Path</th>
<th>Folder2 Path</th>
<th>…….</th>
<th>Folder n Path</th>
<th>overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item: (dictionary)</td>
<td>Folder1 centroid vector</td>
<td>Folder2 centroid vector</td>
<td>…….</td>
<td>Folder n centroid vector</td>
<td>All messages centroid vector</td>
</tr>
</tbody>
</table>

Table 5-1 Data structure of pre-filled messages’ Database.

The keys of the database collect all paths of each user created folders that have at least one message in it. In Outlook object model, we can always get the folder through its path. The items are the corresponding centroid vectors for each user created folder. The centroid vector is also stored in a dictionary structure as in Table 5-2.

<table>
<thead>
<tr>
<th>Key: (string)</th>
<th>word 1</th>
<th>word 2</th>
<th>…….</th>
<th>word n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item: (integer)</td>
<td>Appear times</td>
<td>Appear times</td>
<td>…….</td>
<td>Appear times</td>
</tr>
</tbody>
</table>

Table 5-2 Data structure of folder’s centroid vector.

This dictionary collects all words (except words that are discarded by the data refinement technology, which are discussed in section 5.1.1) that appear in the folder at least once as the keys and the total number of times the word appears in the folder as the item. The last key-item pair in Table 5-1 with the key labeled “overall” keeps cumulative information for all pre-filled messages.
We also keep another dictionary as in Table 5-3 to store the information that will make the calculation of the similarity easier. The keys are same as Table 5-1, and the items keep the total number of words in corresponding folder.

<table>
<thead>
<tr>
<th>Key: (string)</th>
<th>Folder1 Path</th>
<th>Folder2 Path</th>
<th>…….</th>
<th>Folder n Path</th>
<th>“overall”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item: (Integer)</td>
<td>number of words in F1</td>
<td>number of words in F2</td>
<td>…….</td>
<td>Number of words in Fn</td>
<td>Number of words in all messages</td>
</tr>
</tbody>
</table>

Table 5-3 Data structure of myHelpDatabase.

We implement other vectors that are used in AIM algorithm as dictionaries with folder path or word as key and calculation result number as item, because a dictionary is easier both on iterating and retrieving item according a specific key value.

5.2 COM add-in Implementations

5.2.1 My Add-in Implementation

The easiest way to create an add-in is to create a new Visual Basic add-in project. Then, we can use the Connect add-in designer to specify properties for the add-in. These properties are written to the registry when a component is registered so it will be recognized by Outlook applications.
Since this project is an Outlook 2002 add-in, it should include the **Microsoft Outlook 10.0 Object Library** in order to access the Outlook objects directly.

After all properties for registering in Outlook are configured, it’s ready to implement the **IDTExtensibility2** interface. The Office applications use the **IDTExtensibility2**
interface to inform an add-in that it is being loaded, updated and unloaded. All COM add-ins inherit from this interface and must implement each of its five methods. The five methods are:

- **OnConnection**: This event fires whenever the COM add-in is connected. The add-in may be connected on startup, by the end user, or through automation. Our e-mail filter add-in is based on the option of connecting the Outlook at startup and it loads the Outlook application as a global variable in this function.

- **OnDisconnection**: This event fires when the COM add-in disconnected and just before it unloads from memory. The add-in performs any cleanup of resources in this event.

- **OnAddInsUpdate**: This event fires when the set of registered COM add-ins changes, In other words, whenever a COM add-in is installed or removed from the host application, this event fires. In our implementation, nothing is done in this event.

- **OnStartupComplete**: This method is called if the add-in was connected during startup, and when the Outlook application is entering a state where user-interaction should be avoided because the application is busy loading itself from memory. In our implementation, we add an Enable button in the tool bar allowing user to enable the e-mail filter or disable it. This relates the fundamental design for the e-mail filter that is described in Section 4. This allows users to ignore the e-mail filter, hence speeding up other operations within Outlook.

- **OnBeginShutdown**: This method is called if the add-in is disconnected by the host during shutdown.
Through the interface, the e-mail filter add-in also handles other events which are performing work, such as predicting the most likely folders, moving to folder, and updating changes, etc.

- **EnableButton_Click**: This method is called when user clicks the enable button to enable or disable the e-mail filter. When enabling the e-mail filter, this method will scan the user created folders to train the messages, and keep information in the database for the Text classifier algorithm class. The e-mail filter is then ready to classify the messages from the Inbox. This method also adds an update button on the tool bar to allow users to manually update the training message database. When disabling the e-mail filter, this method will free all the global variables and delete the changes made in Outlook GUI by this e-mail filter add-in.

- **New_Inspector**: This method is called when user opens an e-mail from the inbox in a new window. In this method, we will calculate the best three predictions for the e-mail and add three move buttons on the new window’s tool bar.

- **curItem_read**: This method is called when the current selected e-mail is read. At times, the user may set the Outlook to read the mail automatically when it is selected. this method will do the same thing in New_Inspector method expect it will add the three move buttons on the tool bar in main window instead of a new window.

- **MoveToButton_Click**: There are six moveTo buttons, three are used when user opens an e-mail in a new window, and other three are used when the selected e-mail is read in the main window. When one of the buttons is clicked it will move
the current e-mail to the folder that the button corresponds to and update the database incrementally.

- **UpdateButton_Click**: This method is called when a user clicks the update button. This method will scan all user created folders and update the database.

The following methods are trying to catch the user’s operations through Outlook and implement incremental updates.

- **deleteItems_ItemAdd**: This method is called when a user deletes a message. *deleteItems* keeps track of the messages in the delete folder and monitors any changes. In this method, it will update the database incrementally when it is necessary.

- **deleteFolders_folderAdd**: This method is called when user deletes a folder. It will update the database accordingly.

- **curExplorer_beforeItemPaste**: This method is called when a user drops or pastes a message to a folder, and will increment the database if needed.

- **goItems_ItemAdd**: This method is called when a user moves a message using the moveTo buttons provided by this e-mail filter add-in. *goItems* monitors the messages in the destination folder which is set when a user clicks the moveTo button for changes. This method will update the database incrementally.

### 5.2.2 Challenge

Ideally, this e-mail filter would monitor the mail database for changes. Whenever it detects that a message has been added or removed or the hierarchy of the user created
folders has been changed, it should update the text classifier’s database incrementally. The Text classifier algorithm fully supports the incremental update. During the phase of interactions with Outlook, most of the events, such as when a user deletes a message or folder, drags a message and drops to a folder, or a user moves the message by using the moveTo buttons provided by this add-in can be captured. However not all user events that cause changes to messages or folders can be captured automatically due to the diversity of user actions and time constraints of the project to gain a better knowledge of Outlook. With this in mind, the e-mail filter provides an update button to allow a user to manually update the database at any time.
6 Experiments

6.1 Experimental design

In classification tasks, performance is often measured in terms of accuracy. This e-mail filter is designed to classify each message in the Inbox by predicting the three most similar folders. If one of the three predictions is correct, the filter will make the classification task easier than before in Outlook. In this way, we say the e-mail filter is working properly. This experiment is designed to evaluate the accuracy of the e-mail filter.

In the experiment, we applied the e-mail filter to three e-mail accounts individually as three users in Outlook. Table 6-1 presents the information of the three accounts.

<table>
<thead>
<tr>
<th>Account info</th>
<th># Folders</th>
<th># E-mails</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account #1</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Account #2</td>
<td>10</td>
<td>62</td>
</tr>
<tr>
<td>Account #3</td>
<td>20</td>
<td>113</td>
</tr>
</tbody>
</table>

Table 6-1 Accounts’ information.

The messages range of the three accounts is 20 to 113, and the number of the folders is from 5 to 20. The number of folders is an important factor because, as it increases, so does the difficulty of the classification. The experiment conducted using each account’s previously-filed message as data. We randomly moved different number of messages from the data to the Inbox as un-classifying messages and the remaining messages as training data. This experiment also tested the e-mail filter without the refinements on
feature extraction technique which was discussed in section 5.1.1, to see whether the extra work was worthwhile. The experiment was repeated three times for each account.

6.2 Results

6.2.1 Classification Accuracy

Experiment results of the three accounts are displayed in table 6-2. This table lists the number of e-mails that are correctly classified by the e-mail filter with two implementations (with or without the refinement). It also provides distribution information of the training data and the number of e-mails that were classified for each experiment.

<table>
<thead>
<tr>
<th>Account #1</th>
<th># Training Data</th>
<th># E-mail in Inbox</th>
<th># correction no refinement</th>
<th># correction with refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>52</td>
<td>61</td>
<td>35</td>
<td>46</td>
</tr>
<tr>
<td>2</td>
<td>73</td>
<td>40</td>
<td>27</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>88</td>
<td>25</td>
<td>18</td>
<td>21</td>
</tr>
</tbody>
</table>

(1)

<table>
<thead>
<tr>
<th>Account #2</th>
<th># Training Data</th>
<th># E-mail in Inbox</th>
<th># correction no data refinement</th>
<th># correction with data refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>42</td>
<td>23</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>22</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>11</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

(2)
The performance of the e-mail filters is measured by accuracy, which is the number of the e-mails that are correctly classified, divide the total number of e-mails in Inbox. According to the experiment results in Table 6-2, we calculated the accuracy, which displayed in Table 6-3 for each account. We plot the results of Table 6-3 to Figure 6-1 to have an intuitive view.

<table>
<thead>
<tr>
<th>Account #1</th>
<th>Account #2</th>
<th>Account #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Experiment</td>
<td>Accuracy no refine (%)</td>
<td>Accuracy with refine (%)</td>
</tr>
<tr>
<td>1</td>
<td>57%</td>
<td>75%</td>
</tr>
<tr>
<td>2</td>
<td>68%</td>
<td>83%</td>
</tr>
<tr>
<td>3</td>
<td>72%</td>
<td>84%</td>
</tr>
<tr>
<td>average</td>
<td>66%</td>
<td>81%</td>
</tr>
</tbody>
</table>

Table 6-2 Experimental result.

Table 6-3 Analytic result.
6.2.2 Observations

As can be seen in Table 6-3, the e-mail filter performs quite well, achieving an average accuracy of 80%. By looking at individual experimental result, we can see that the performance of the e-mail filter without the refinements on feature extraction is from 55% to 80%. A performance level of 80% is likely to be regarded as helpful by most users, but as noted in Table 6-3, the 80% accuracy only achieved when user has small number of folders. The average performance level is around 68% which is a little low. While the user may still save time using the e-mail filter, the perception that the e-mail filter is frequently incorrect may lead to user dissatisfaction. With the refinements on
feature extraction, the e-mail filter’s accuracy improves to 69% to 100%. It is likely that
the refinement works well to eliminate noise and make the intended classification. The
average performance level is approximately 86% which is pretty well to help most users.

The volume of training data is an important factor that affects the accuracy of
classification. Therefore, we designed that each experiment with a different number of
training data, namely from experiment 1 to 3 in an increasing number. We can observe
from Figure 6-1 that the accuracy for the three accounts increases as the volume of
training data increases. Also from the last graph in Figure 6-1, we can see smaller
numbers of folders tended to produce better results. However, this is sometimes depends
on the raining data. For example, the average accuracy of account 1 is almost the same as
account 2, even account 1 had 10 more folders than account 2. This occurred because the
experiments for account 1 used enough training data for each folder, and the text
classifier algorithm is learning based upon words, not semantics. It learns content rather
than more abstract notions. We can imagine that much of the content there is similar, and
hence the learning process sometimes gets confused. So, enough training data plus data
with certain similar topics proved to be easier to get correctly classified.
7 Conclusion

The experimental results provided in this report provides strong evidence that mail filtering using text-based Machine Learning Technique is capable of achieving high classification accuracy while eliminating many of the annoyances of traditional mail filters. This e-mail filter’s great characteristic is simplicity. No longer does the user need to spend time creating and updating rules and they no longer restricted to organizing mail according to keyword matching. Users have nothing new to learn or do to use it.

Another great characteristic of the e-mail filter is that has virtually no adverse side effects than automatically filing messages. If its predictions are wrong, the user can bypass it effortlessly by using the original mail filtering provide by Outlook.

Furthermore, this e-mail filter performance is excellent (80% to 96%) by adding the refinement on feature extraction technology. Even though the higher performance is dependent on the training data, this e-mail filter still performs well enough to consider it a useful life-simplifying tool.
8 Future Work

Looking back at the first half term, I have struggled with the understanding of the Machine Learning algorithm and with knowing how to extend Outlook. At that time I could not imagine how I could accomplish this project. Now I’m proud to say that I have made a lot of progress and learned a lot of new knowledge, even though there are still other improvements for the e-mail filter and open researches to be continued, such as:

- The incremental learning implementation in Outlook as mentioned in section 5.2.3. When a user decides to rename folders, reorganize the folder hierarchy, or use the original moving interface provided by Outlook to move messages, the e-mail filter takes no special action to recognize this severe organizational change. Unless the user wishes to deal with a classifier which is not very well suited for the current mailbox organization, the user must manually click the update button to update the e-mail filter’s database. It would be easier for the user if the filter were able to detect all the user changes and update the database incrementally.

- While the e-mail filter was developed for electronic mail, it can easily be used to organize other types of electronic documents. The concepts behind this filter can be applied to organizing bookmarks, audio recordings, files, and other text based documents that are placed into a hierarchy of folders.
• There are various Machine Learning algorithms, such as Naïve Bayes. Additional research area is also needed to be working on is to compare the responsiveness of AIM text classifier to other types of text classifiers.
9 \hspace{1em} How To Use The E-mail Filter

9.1 Introduction

This project comprises the implementation of an add-in application for Microsoft Outlook 2002 that intends to help managing the inbox by automatically classifying email based on user folders. The project comes with the ability to predict the three most likely folders and allow a user to move mail to a folder by a single button click if one of the predictions is correct.

This manual explains how to install, run and use this e-mail filter. It also describes how the users can disable or completely uninstall it from the Outlook application. It also includes an extensive description of the GUI provided.

9.2 User’s Guide

9.2.1 Compatibility

This software has only been tested with Outlook 2002 with Microsoft Outlook object library version 10.0 and Windows 2000 or Windows XP. Other combinations may not work. The software will not work with older versions of Outlook.

9.2.2 Installation

Users can install the e-mail filter to Outlook either by the installation .exe file or through Visual Basic 6.0 to compile the code.
Through the Visual Basic 6.0:

1. Open the mailFilter.vbp project in VB6.0
2. click File->make mailFilter.dll
   
   VB will automatically register the dll in Outlook.

There is an installation setup file to allow users adding this e-mail filter without knowing anything about the source code and Visual Basic 6.0. A user can simply click on the setup.exe installation file and follow the wizard to install this add-in in Outlook.

9.2.3 Program Operations

1. STARTUP

Once the e-mail filter has been installed, the user should see an ‘EnableFilter’ button which is added on the tool bar by the e-mail filter when the user starts his/her Outlook completely. The add-in will do nothing unless the user clicks the Enable button to enable the e-mail filter. At this moment, the add-in just connected to Outlook, and it is listening to a user event, such as enable button click.

![Enable button added when Outlook started completely.](image)
2. ENABLE E-MAIL FILTER

When user clicks the enable button, user should see the add-in scan through all user folders messages:

![Figure 9-2 Waiting message.](image)

This process automatically learns the user’s pre-filled habit and keeps the information in a database, which will be used for classifying in future. It will add an Update button on the tool bar after the scanning folder process is done, and will change the ‘Enable’ button to a ‘disable’ button.

![Figure 9-3 Update button added when the e-mail filter is enabled.](image)

3. CLASSIFY E-MAIL

The add-in is designed so that the e-mails in the Inbox folder are considered as uncategorized mails. Whenever an e-mail in Inbox is read or opened, the predictions are calculated and the three MoveTo buttons are added on the tool bar for current mail.
Figure 9-4 Three MoveTo buttons are added in Outlook main window.

Figure 9-5 Three MoveTo buttons are added in a new window when open a message.

4. MOVE TO FOLDER

If one of the predictions is correct, then the classifying task is simplified by clicking the MoveTo button to move the e-mail to the corresponding folder. The move action may take more time if user moves to the folder across the e-mail account instead of local folders. In this case, a waiting message dialog similar as Figure 9-2 will appear.

However, if none of the predictions are correct, the user can ignore the predictions that are made by this add-in and can use the original move to interface in Outlook to move this e-mail to the correct folder.
5. UPDATE

Currently, the database only collects the information when the e-mail filter is first enabled. Any change in hierarchy of the folders or messages that may be moved between folders while Outlook is running has not been notified by the add-in so far. This information will only be updated when user clicks the Update button. The waiting message dialog box similar to Figure 9-2 will appear.

6. DISABLE E-MAIL FILTER

When a user finishes all classifying, or would like to do other operation in Outlook, they click the Disable button on the tool bar to disable the e-mail filter as in Figure 9-3.

7. UNINSTALL E-MAIL FILTER

To uninstall the e-mail filter, a user would go to control panel and remove it same as the way as removing any software.
10 References


3. Howto: built a office 2000 COM addin in Visual Basic ,
   http://support.microsoft.com/?kbid=238228


5. The Porter Stemming algorithm
   http://www.tartarus.org/~martin/PorterStemmer/index.html

6. Richard B. Segal; Jeffrey O. Kephart. Incremental Learning in SwiftFile. IBM Thomas J. Watson Research Center

7. Richard B. Segal; Jeffrey O. Kephart. MailCat: An Intelligent Assistant for Organizing E-mail. IBM Thomas J. Watson Research Center


11. William Cohen. Learning rules that classify e-mail. 1996