TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

Final Year Project Report

On

CORVID – Intelligent Email Processor

[Code No: EG777CT]

Submitted By

Lekhnath Bhusal (060BCT515)	Exam Roll No: 24214
Madhav Sigdel (060BCT516)	Exam Roll No: 24215
Mahesh Raj Regmee (060BCT517)	Exam Roll No: 24216
Prajwal Shrestha (060BCT528)	Exam Roll No: 24226

Lalitpur, Nepal

February 2008

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

Final Year Project Report

On

CORVID – Intelligent Email Processor

[Code No: EG777CT]

Submitted By

Lekhnath Bhusal (060BCT515)	Exam Roll No: 24214
Madhav Sigdel (060BCT516)	Exam Roll No: 24215
Mahesh Raj Regmee (060BCT517)	Exam Roll No: 24216
Prajwal Shrestha (060BCT528)	Exam Roll No: 24226

SUBMITTED TO THE

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR BACHELOR'S DEGREE IN COMPUTER ENGINEERING

Lalitpur, Nepal

February 2008

Acknowledgement

Secret behind every successful project is proper planning, sufficient preliminary studies and smart implementation of planned activities. Continuous help from various persons was vital in achieving the goals to come up with solid project outcomes. In this regard, the authors would like to express their vote of thanks to following people/organizational unit.

1. Department of Electronics and Computer Engineering.
2. Project Supervisors **Er. Jayaram Timsina** and **Er. Ram Govinda Aryal** for the support and encouragement in proceeding with the project.
3. Project Co-coordinator, Asst. Professor, **Rajendra Lal Rajbhandari**
4. **Shankar Raj Uprety**, Senior Engineering manager, D2Hawkeye Services and **Subash Devkota**, Software Engineer D2Hawkeye Services, Bishalnagar for their helpful suggestion and supervision.
5. D2Hawkeye Services, Bishalnagar for its guidelines and support for the project.
6. Our seniors for their valuable materials and suggestions.
7. Our friends and family for their direct or indirect support in project study.

Thank You

CORVID Team

February 6, 2008

Abstract

CORVID is a research based project on email mining for email filtration, classification and summarization. It is built as an extension to Mozilla Thunderbird. CORVID can act intelligently to help Thunderbird users manage their inbox through personalization and learning. CORVID tracks user's behavior to incoming mails and user's feedback over CORVID actions and updates knowledgebase. Based on this knowledgebase, CORVID filters and classifies the emails. It generates summary of email threads using machine learning and linguistic techniques for summarization.

Keywords - Mozilla, XPCOM Component, Naïve Bayesian, Associative Rule, Machine Learning

List of Acronyms

ANN	Artificial Neural Network
CAR	Classification Association Rule
CBA	Classification Based on Association
CMAR	Classification Based on Multiple Association Rules
COMAR	Cohesion and Multiple Association Rule
CSS	Cascading Style Sheet
DLL	Dynamic Link Library
DTD	Data Type Definition
EFPTree	Extended Pattern Tree
FPTree	Frequent Pattern Tree
GUID	Globally Unique Identifier
HTML	Hyper Text Markup Language
NP	Noun Phrase
POS	Part Of Speech
TFIDF	Term Frequency Inverse Document Frequency
XML	Extensible Markup Language
XPCOM	Cross Platform Component Object Model
XPIDL	Cross Platform Interface Definition Language
XUL	XML User interface

List of Figures

Table of Contents

1 Introduction

1.1 Background

Electronic mail has not only turned into a vital tool for our work but also into an important means of interpersonal communication. Each minute, millions of plain text or enriched messages are being sent and received around the globe. Some of them are read with extra care and, at the same time, many of them are deleted with obvious disinterest. Even the necessary emails are in large number and it is difficult to identify important information from a sequence of emails. Email service is provided in different mechanisms such as web-based or mail client. Web based mail service can be accessed from anywhere through the internet however they only provide generic services and require internet accessibility. User specific services are not directly available in the servers. For such services and offline email achieving email client is required.

There have been different research in the field of Email filtering and different algorithms have been devised. Bayesian classification has been the mostly used one. Different mail service providers provide anti-spam techniques. Web mail providers such as Gmail, Yahoo have applied static mail filtering algorithms and appropriate actions such as deleting the mail, moving a folder etc. can be defined. Mail clients like Mozilla Thunderbird and Microsoft Outlook Express have made filtering techniques very robust. Mailwasher Pro has also incorporated Bayesian learning for spam mail detection. POPFile is another similar application which sorts mail in different categories such as (family, friends, co-worker, spam etc.) with Bayesian filtering.

1.2 Problem Statement

Most of the existing applications found so far use static filters and spammers have already devised techniques to override such filters. Also these do not address changing requirement of users. Likewise, some use Naïve Bayesian for classification which considers single words as features Also summarization feature is not present which can help in extracting important sentences from a long email.

1.3 Objectives

We propose to develop a system that can learn user preferences for filtering and categorization of email from user activities pertaining to an email message. Along with this, it is required to generate relevant summary of email threads. To achieve this, we have set the following objectives:

1. Develop a module for email filtering
2. Develop email classification module
3. Develop email summarization module
4. Integrate the above modules as an extension to Mozilla Thunderbird

1.4 Scope

The project CORVID has been undertaken to fulfill the requirement of Major Project for Bachelor’s Degree in Computer Engineering and has to be completed within the end of final semester. CORVID is a research based project on email mining for email filtration, classification and summarization. Of the several algorithms for filtration, classification and summarization only single algorithms for each are implemented.

CORVID is built as an extension to Mozilla Thunderbird. It can act intelligently to help Thunderbird users manage their inbox through personalization and learning. Mozilla Thunderbird users can use CORVID for managing their inbox through personalization. They can use the CORVID’s feature such as spam/ ham detection of an email, classification of incoming emails to folders and to extract important parts as summary of an email.

1.5 Feasibility Analysis

1.5.1 Operational Feasibility

Mozilla Thunderbird is a popular mail client and CORVID is an extension to it with added intelligence in filtering, email classification and support for email summarization. Since, this functionality can be easily incorporated in the existing system the project will be useful to the users. Users will benefit from the intelligence added to Thunderbird. The confidence of CORVID system will go on increasing as it is used and can suggest things intelligently. The confidence level will be less in the early usage of the system.

1.5.2 Technical Feasibility

CORVID is an extension to Thunderbird; therefore we require good knowledge of the design and implementation of it. Thunderbird is open source and its documentation is readily available. Also we can get support from the forums of Thunderbird. The help of supervisors, access to information in the internet and different books, research papers and articles will suffice.

1.5.3 Economic Feasibility

The development of CORVID requires Mozilla Thunderbird, Gecko Engine and Microsoft Visual C++ Express. The tools required for the development and extension of Thunderbird is also easily available, therefore there is no cost for the tools. Further the project is an academic project. Therefore costs of manpower are not involved. This makes the project economically feasible.

1.5.4 Schedule Feasibility

The time period of 4 months is a short time for this project which requires extensive research and experiments. But the target is to provide basic functionalities and develop it as an extension to Mozilla Thunderbird. Thus basic requirements can be fulfilled within the time frame.

1.5.5 Risk Analysis

CORVID is a project incorporating Artificial Intelligence and hence requires extensive research. We are a group of four members and have time limit of around 4 months which is very less for this sort of project. However, different algorithms have been devised for email filtering and email classification. Due to time constraint, we studied and implemented these algorithms instead of developing new algorithms and build a basic framework for developing such system. Hence implementation of these algorithms may possess some risk in performance level.

2 Literature Review

2.1 General

2.1.1 Email

E-mail, short for electronic mail, is a store and forward method of composing, sending, storing and receiving messages over electronic communication systems. It is often abbreviated to e-mail or simply mail. The term e-mail as a noun or verb applies both to the Internet e-mail system based on the Simple Mail Transfer Protocol (SMTP), X.400 systems and to intranet systems allowing users within one organization to e-mail each other. Often these workgroup collaboration organizations may use the Internet protocols or X.400 protocols for internal e-mail service.

2.1.2 E-mail spam

E-mail is often used to deliver bulk unsolicited messages known as spam, bulk or junk e-mails. It is a subset of spam that involves sending nearly identical messages to numerous recipients by e-mail. A common synonym for spam is unsolicited bulk e-mail (UBE). Some definitions of spam specifically include the aspects of email that is unsolicited and sent in bulk. The spam emails mainly contain commercial, financial, or other leisure advertisements. Filter programs exist to detect whether an email is spam or not, which can automatically delete some or most of these, depending on the situation.

2.1.3 Mail Server

A mail server or a mail exchanger (in the context of the Domain Name System) is a computer program or software agent that transfers electronic mail messages from one computer to another. It receives messages from another MTA (mail transfer agent) (relaying), a mail submission agent (MSA) that itself got the mail from a mail user agent (MUA), or directly from an MUA, thus acting as an MSA itself. The delivery of e-mail to a user's mailbox takes place via mail delivery agent (MDA).

2.1.4 Email Client

An e-mail client is a front end computer program used to manage e-mail. Mozilla Thunderbird, Outlook Express, Evolution Mail are some of the popular email clients.

Email Clients use POP3 or IMAP to retrieve messages and use SMTP to send messages. Fewer Internet Service Providers (ISPs) support IMAP.

2.2 Mozilla Applications

2.2.1 Mozilla Foundation

The Mozilla Foundation is a non-profit organization that exists to support and provide leadership for the open source Mozilla project. The organization sets the policies that govern development, operates key infrastructure and controls trademarks and other intellectual property. It owns a taxable subsidiary called the Mozilla Corporation, which employs several Mozilla developers and coordinates releases of the Mozilla Firefox web browser and the Mozilla Thunderbird email client.

2.2.2 Mozilla Thunderbird

Mozilla Thunderbird is a free, open source, cross-platform e-mail and news client developed by the Mozilla Foundation. The project strategy is modeled after Mozilla Firefox, a project aimed at creating a web browser. Thunderbird aims to be simple, safe and easy to use email client. It can manage multiple e-mail, newsgroup and RSS accounts and supports multiple identities within accounts. It has features such as intelligent spam filters, quick message search, and customizable views. Thunderbird incorporates a Bayesian spam filter, a white list based on the included address book, and can also understand classifications by server-based filters such as SpamAssassin. Its pluggable interface lets developers freely build extensions to make it ever more useful

2.2.3 Mozilla Architecture

The biggest advantage Mozilla has for a developer is that Mozilla-based applications are cross-platform, which means that these programs work the same on Windows as they do on Unix or the Mac OS. It's possible to have applications run across different platforms because Mozilla acts as an interpretation layer between the operating system and the application. As long as Mozilla runs on a given computer, most Mozilla-based applications also run on that computer, regardless of what operating system it uses. XPFE, Mozilla's cross-platform front end, was designed to solve this problem by enabling engineers to create one interface that would work on any operating system. a front end is more than the look and feel of an application, since it also includes the functionality and structure of that application. XPFE uses a number of existing web standards, such as Cascading Style Sheets, JavaScript, and XML (the XML component is a new language called XUL, the XML-based User-interface Language). In its most simple form, XPFE can be thought of as the union of each technology. JavaScript creates the functionality for a Mozilla-based application, Cascading Style Sheets format the look and feel, and XUL creates the application's structure. Instead of using platform-specific code in languages like C or C++ to create an application, XPFE uses well-understood web standards that are platform-independent by design. Because the framework of XPFE is platform-independent, so are the applications created with it.

2.2.4 Extension to Mozilla Application

Extensions add new functionality to Mozilla applications such as Firefox and Thunderbird. They can add anything from a toolbar button to a completely new feature. They allow the application to be customized to fit the personal needs of each user if they need additional features, while keeping the applications small to download. It is the most convenient way to distribute and add new customized feature to the existing application.

Some Popular Firefox Extensions are:

- ForecastFox
- ChatZilla
- Adblock Plus

Some Popular Thunderbird Extensions are:

- Enigmail
- FoxyTunes

2.2.4.1 Extension Architecture

The general architecture of creating extensions is as shown below. The extension is generally packaged as xpi or zippy file that manages the transfer and registry of all components -- the chrome files in the JARs, the executables, the default user information, and the libraries. It contains the XUL files, overlays, event handlers in JavaScript, XPCOM components and XPCOM interface in between.

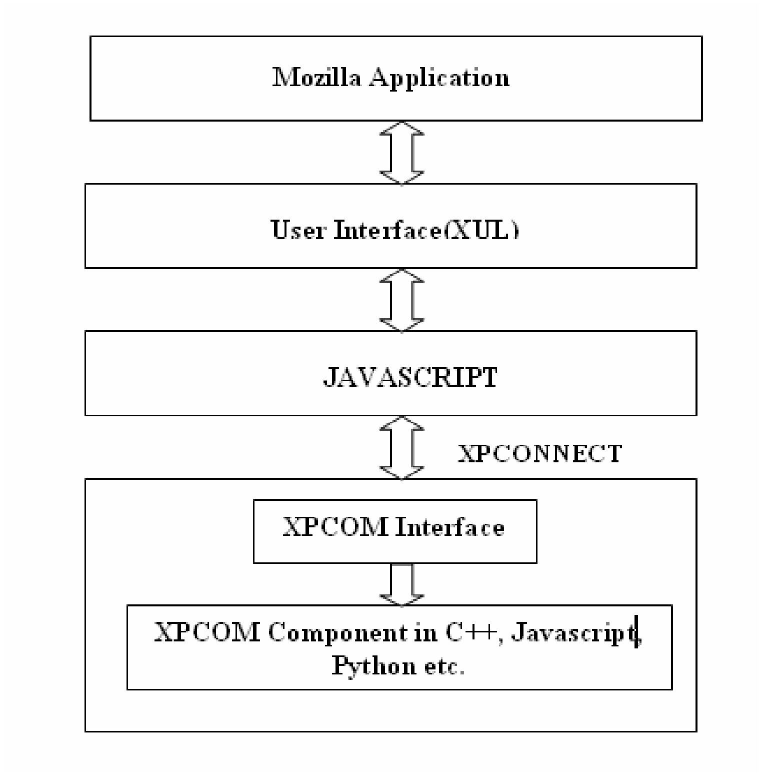


Figure 1: Mozilla Extension Architecture

2.2.4.2 XUL

XUL is Mozilla's XML-based user interface language that allows building feature rich cross-platform applications that can run connected to or disconnected from the Internet. These applications are easily customized with alternative text, graphics, and layout so they can be readily branded or localized for various markets. Web developers already familiar with Dynamic HTML (DHTML) will learn XUL quickly and can start building applications right away.

- Powerful widget-based markup language - artifacts such as windows, labels and buttons instead of pages, heading levels, and hypertext links.
- Based on existing standards - XUL is an XML language based on W3C standard XML 1.0
- Platform portability - XUL designed to be platform-neutral, it delivers on the promise of write-once, run-anywhere.
- Separation of presentation from application logic
- Easy customization, localization, or branding

2.2.4.3 Overlays

An overlay is a separate file in which additional XUL content can be defined and loaded at runtime. Overlays are often used to define things like menus that appear in different components or parts of the application. If you are creating a large application or a UI with many elements as a part of your design, the files can easily become large. The size in itself does not render it ineffective, but it does make the job of the developer a little difficult when tracking down and changing features. The best way to overcome this size problem is to use overlays. Another reason to use overlays is to extract information from a certain logical portion of the UI and contain it in a file of its own. This extraction and containment promotes modularization and reusability.

2.2.4.4 JavaScript

JavaScript is a small, lightweight, object-oriented, cross-platform scripting language. It adds the functionality to Mozilla applications. This is the main language for event handling in Mozilla applications. We can also create XPCOM Components in Javascript.

2.2.4.5 XPCOM

A component is a reusable or modular piece of code that implements a clearly defined interface. XPCOM, which stands for Cross Platform Component Object Model, is a framework for writing cross-platform, modular software. XPCOM components can be written in C, C++, and JavaScript, and they can be used from C, C++, JavaScript, Python, Java, and Perl. XPCOM itself provides a set of core components and classes, e.g. file and memory management, threads, basic data structures (strings, arrays, and variants), etc. In other words, XPCOM is used to allow JavaScript, or potentially any other scripting language, to access and utilize C and C++ libraries.

XPCOM Component can exist as a singleton service or an object instance. A singleton service is an object instance that is created only once and then used by other code. An object instance is an object that is instantiated once or many times. Components are written as classes that typically have member variables and methods. The basic purpose of a component is to implement a clearly defined set of APIs that exist in a public interface. The interface exists separately so that the implementation is abstracted away, and it can be changed without affecting the interface or breaking binary compatibility. When interfaces are deployed in a production environment, they are frozen, which means they are held in an immutable state-theoretically for as long as the application exists.

2.2.4.6 XPConnect

XPConnect is a technology which enables simple interoperation between XPCOM and JavaScript. XPConnect allows JavaScript objects to transparently access and manipulate XPCOM objects. It also enables JavaScript objects to present XPCOM compliant interfaces to be called by XPCOM objects. Together, XPCOM and XPConnect enable developers to create XUL applications that require the raw processing power of compiled languages (C/C++) or access to the underlying operating system.

2.2.4.7 XPIDL

All XPCOM interfaces are defined with the Interface Definition Language (IDL). IDL provides a language-neutral way to describe the public methods and properties of a component. Mozilla actually uses a modified, cross-platform version of IDL called XPIDL to compile interface source files. An IDL compiler is a tool that creates a binary distribution file called a *type library* from an interface description source file. Since support for many different platforms is a requirement for Mozilla, a modified version of the libIDL compiler from the Gnome project is used. This variant is called the XPIDL compiler and is primarily used to compile Mozilla's own dialect of IDL, conveniently called XPIDL. The XPIDL compiler generates XPCOM interface information, headers for XPCOM objects, and XPT type libraries from which objects may be accessed dynamically through XPConnect. It can also generate HTML files for documentation and Java class stubs. Another feature of the XPIDL compiler is the option to generate C++ code stubs. This feature creates nearly all the declaratory C++ code you need when you start a new project, which makes XPIDL useful as a coding wizard that helps you get started.

2.2.4.8 Cross Platform Type library

The key to the component architecture of XPCOM is the presence of binary-independent interface files that are used uniformly across platforms, languages, and programming environments. These interface files are compiled into `.xpt` files by the XPIDL compiler. The Mozilla `components` subdirectory is where type libraries and modules are typically stored. If you create a cross-platform type library for your component, you must place it in this directory for it to be accessible to XPCOM.

2.2.4.9 XPInstall

XPInstall, Mozilla's Cross Platform Install facility, provides a standard way of packaging XUL application components with an install script that Mozilla can download and execute. XPInstall enables users to effortlessly install new XUL. Extensions are in a form of installable bundle which can be downloaded and installed by a user, or provided pre-packaged with an application or by an external program. Extensions use a directory structure which can provide chrome, components, and other files to extend the functionality of a XUL program. Every extension must provide an install.rdf file which contains metadata about the extension, such as its unique ID, version, author, and compatibility information. After the extension files and install.rdf have been prepared, there are several ways to prepare an extension for installation: ZIP the extension directory into a user-installable XPI (xpinstall) file, unpack the extension directly into the user's application or profile directory, or register the extension in the Windows registry.

2.3 Email Filtration

Spam email filtering is the task of automatically filtering out unwanted electronic mail. Spam email also commonly refers to unsolicited commercial emails or junk emails. Automatic IR methods are well suited for this problem, since spam messages are distinguished from legitimate email messages because of their particular form, vocabulary and particular word patterns, which can be found in the headers and body of the messages.

Filtration problem can be viewed as a special case of text categorization (TC) with two classes. However, since one class is the opposite of other, it can be viewed as the task of identifying a single class. Another important issue is the relative importance of two types of possible misclassifications. While an automated filter missing a small percentage of spam messages is acceptable to most users, it is unlikely that it is acceptable to miss small percentage of legitimate messages as spam, especially when spam messages are automatically removed. This issue suggests the consideration of the cost of misclassifications for learning and evaluation of the filter systems.

There are many popular algorithms that are implemented for email filtration. In this section we are going to discuss some of these algorithms.

2.3.1 Boosting Trees

AdaBoost algorithm with confidence rated predictions is well suited algorithm for email filtering problems. It is able to efficiently manage a large feature sets (tens of thousands) efficiently, so no feature filtering is needed. Here, we will describe generalized AdaBoost algorithm with confidence rated predictions restricting to the case of binary classifications.

"The purpose of boosting is to find a highly accurate classification rule by combining many weak rules, each of which can be only moderately accurate." [Xavier Carreras and Luis Marquez, 2001]. It is assumed that weak rules are available from weak learners as weak hypotheses. The boosting algorithm obtains weak rules by calling the weak learner in a series of T rounds. These weak hypotheses are then linearly combined into a single hypothesis called combined hypothesis.

Here is the boosting algorithm:

```
AdaBoost(in: $S = \{(x_i, y_i)\}_{i=1}^{m}$)
    for $i = 1$ to $m$
        $D_1(i) \leftarrow 1/m$
    for $t = 1$ to $T$
        $h_t \leftarrow$ WeakLearner$(S, D_t)$
        Choose $\alpha_t \in \mathbb{R}$
        for $i = 1$ to $m$
            $D_{t+1}(i) \leftarrow \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
            # $Z_t$ is chosen so that $D_{t+1}$ will be a distribution
    return the combined hypothesis: $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$
```

Let $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ be the set of m training examples, each instance x_i belongs to instance space $\mathcal{X}$ and $y_i \in \{-1, +1\}$ is the class or label associated with $x_{i.}$ The goal of learning is to produce a function of the form $f : \mathcal{X} \to \mathbb{R}$, such that, for any example x, the sign of f(x) is the predicted class (-1 or +1) and the magnitude | f(x)| is interpreted as the level of confidence in the prediction. Such a function can be used for classifying or ranking new unseen examples. Above pseudo code for boosting algorithm maintains a vector of weights as a distribution D over examples. The goal of WeakLearner algorithm is to find a weak hypothesis with moderately low error with respect to these weights. Initially, the distribution D_1 is uniform, but the boosting algorithm exponentially updates the weights on each round forcing the weak learner to concentrate on the examples which are hardest to predict by the preceding hypotheses.

More precisely, let D_t be the prediction at round t, and h_t: $\mathcal{X} \to \mathbb{R}$ the weak rule acquired according to $D_{t.}$ In this setting $h_t(x)$ also makes real valued confidence-rated predictions. A real value α_t is then chosen and the distribution D_t is updated. The choice of α_t will be determined by the type of weak learner, discussed shortly after.

The final hypothesis f computes its predictions using the weighted votes of weak hypotheses. It is proven that the maximum error of AdaBoost algorithm is at most $\prod_{t=1}^{T} Z_t$, where Z_t is the normalization factor computed in round t. This upper bound is used as guidance for computing α_t and designing weak learner algorithm.

2.3.1.1 Learning Weak Rules

There are three variants of AdaBoost algorithms defined. Here, we focus on that which achieved the best result in Text Categorization domain.

According to this setting, weak hypotheses are simple rules with real valued predictions. Such simple rules test the value of a Boolean predicate and make a prediction based on that value. The predicates used to refer to the presence of a certain word in the text. For example "the word money appears in the message". Formally based on a given predicate P, our interest lies on the weak hypothesis which makes predictions of the form:

$$h(x) = \begin{cases} c_0 & \text{if } p \text{ holds in } x \\ c_1 & \text{otherwise} \end{cases}$$

Where, c_0 and c_1 are real numbers.

For a given predicate P, the values of c_0, c_1 are calculated as follows:

Let X_1 be the subset of examples for which the predicate P holds and let X_0 the subset of examples for which predicate P does not hold. Let, $[\![\pi]\!]$, for any predicate π, be 1 if π holds and 0 otherwise. Given the current distribution D_t, the following real numbers are calculated for $j \in \{0,1\}$, and for $b \in \{+1,-1\}$

$$W_b^j = \sum_{i=1}^{m} D_t(i)[\![x_i \in X_j \wedge y_i = b]\!].$$

That is, W_b^j is the weight, with respect to the distribution D_t , for the training examples in partition X_j which are of classes b.

Z_t is minimized for a particular predicate by choosing

$$c_j = \frac{1}{2}\ln\left(\frac{W_{+1}^j}{W_{-1}^j}\right).$$

And by choosing $\alpha_t = 1$, these settings imply that

$$Z_t = 2\sum_{j\in\{0,1\}} \sqrt{W_{+1}^j W_{-1}^j}.$$

Thus, the predicate P chosen is that for which the value of Z_t is smallest.

Very small or zero values for the parameters W_b^j cause c_j predictions to be large or infinite in magnitude. In practice, such large predictions may cause numerical problems to the algorithm, and seem to increase the tendency to overfit. The following smoothed values for c_j have been considered:

$$c_j = \frac{1}{2} \ln \left(\frac{W_{+1}^j + \epsilon}{W_{-1}^j + \epsilon} \right)$$

It is important to see that the so far presented weak rules can be directly seen as decision trees with a single internal node (which tests the value of a Boolean predicate) and two leaf nodes that give the real-valued predictions for the two possible outcomes of the test. These extremely simple decision trees are sometimes called decision stumps. In turn, the Boolean predicates can be seen as binary features. Thus, the already described criterion for finding the best weak rule, or the best feature can be seen as a natural splitting criterion and used for performing decision-tree induction.

The experiment [Xavier Carreras and Luis Marquez, 2001] shows that, above a certain number of rounds, all TreeBoost versions achieve consistent good results, and that there is no overfitting in the process. Deeper the weak rules, the smaller the number of rounds needed to achieve good performance. This is not surprising, sinve deeper weak rules handle much more information.

2.3.2 Bayesian Approach

Many commercial products are now available that require users to hand-build a rule set to detect junk assume that their users are savvy enough to be able to construct robust rules. Moreover, as the nature of junk email changes over time, these rule sets must be constantly tuned and refined by the user. This is a time-consuming and often tedious process which can be notoriously error-prone.

A junk mail filtering system should be able to automatically adapt to the changes in the characteristics of junk mail over time. Moreover, by having a system that can learn directly from data in a user's mail repository, such a junk filter can be personalized to the particular characteristics of a user's legitimate (and junk) mail.

This, in turn, can lead to the construction of much more accurate junk filters for each user.

Methods have recently been suggested to automatically learn rules for text classification, however are not employed specifically for filtering junk emails. Moreover, these rule based systems are of limited use in filtering problems. This is due to the fact that such logical rules make rigid binary decisions as to whether to classify a message as junk. These rules generally provide no sense of continuous degree of confidence with which classification is made.

As to the nature of problem of filtering, we require first a classification scheme that provides probability for classification decision and second, a quantification for difference of cost between two types of errors in this task.

In order to build probabilistic classifiers to detect junk emails, formalism of Bayesian networks is used. A Bayesian network is a directed acyclic graph that compactly represents a probability distribution (Pearl 1988). A Bayesian classifier is simply a Bayesian network that is applied to classification task. It contains a node C for class variable and a node X_i for each of the features. Given a specific instance x, the Bayesian network allows us to compute the probability, $P(C = c_k \mid \mathbf{X} = \mathbf{x})$ for each possible class c_k. This is done by Bayes theorem, giving us

$$P(C=c_k|X=x) = P(X=x|C=c_k)*P(C=c_k)/P(X=x)$$

The critical quantity in this equation is $P(X=x|C=c_k)$, which is often impractical to compute without imposing independence assumption. The oldest and most restrictive forms of such assumptions is embodied in Naïve Bayesian classifier which assumes each feature X_i is conditionally independent of each of other features. With this assumption, for given class variable C, this yields:

$$P(\mathbf{X} = \mathbf{x} \mid C = c_k) = \prod_i P(X_i = x_i \mid C = c_k).$$

Recently much more work are being done which allow limited dependency.

In the context of text classification, specially junk E-mail filtering, it becomes necessary to represent mail messages as feature vectors so as to make such Bayesian classification methods directly applicable. To this end, the Vector Space model is used in which each dimension of this space is defined as corresponding to a given word in the entire corpus of messages seen. Each individual message can then be represented as a binary vector denoting which words are present and absent in the message. With this representation, it becomes straight forward to learn a probabilistic classifier to detect junk mail given pre-classified set of training messages.

In considering specific problem of junk E-mail filtering, however, it is important to note that there are many particular features of E-mail beside just the individual words in the text of a message that provide evidence as to whether a message is junk or not. For example, particular phrases, such as “Free Money”, or over-emphasized punctuation, such as “!!!!!!!”, are indicative of junk E-mail.

It is straight-forward to incorporate such additional problem-specific features for junk mail classification into the Bayesian classifiers described above by simply adding additional variables denoting the presence or absence of these features into the vector for each message. In this way, various types of evidence about messages can be uniformly incorporated into the classification models and learning algorithms employed need not be modified.

2.3.3 Maximum Entropy Model

A purely statistical approach expresses the differences among messages in terms of the likelihood of certain events. The probabilities are usually estimated automatically to maximize the likelihood of generating the observations in a training corpus. Hence a statistical model is easy to build and can adapt to new domains quickly. However, the mathematical model used in a purely statistical system often lacks deep understanding of the problem domain; it simply estimates parameters from training corpus. So a model performs well on one corpus may work badly on another one with quite different characteristics. A rule based approach, on the other hand, expresses the domain knowledge in terms of a set of heuristic rules, often constructed by human experts in a compact and comprehensive way. This approach has the advantage of expressing complex domain knowledge usually hard to be obtained in a purely statistical system. For instance, heuristics such as “the message contains some java-scripts for form validation”, may be used in filtering our junk messages which contain a HTML register form. However, building a rule based system often involves acquiring and maintaining a huge set of rules with an extremely higher cost compared to the purely statistical approach. And such system is hard to scale up.

A better way may be combining the advantages of both approaches into a single statistical model, a Maximum Entropy (ME) hybrid approach to junk mail filtering task, utilizing a Maximum Entropy probabilistic model. The performance of Maximum Entropy Model has been shown to be comparable to that of other statistical modeling methods or even better.

The goal of the ME principle is that, given a set of features, a set of functions f1 . . . fm (measuring the contribution of each feature to the model) and a set of constrains, we have to find the probability distribution that satisfies the constrains and minimizes the relative entropy, with respect to the distribution p0. In general, a conditional Maximum Entropy model is an exponential (log-linear) model has the form

$$p(a|b) = \frac{1}{Z(b)} \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$$

Where p(a|b) denotes the probability of predicting an outcome a in the given context b with constraint or "feature" functions f_j (a, b). Here k is the number of features and

$$Z(b) = \sum_a \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$$

is a normalization factor to ensure that $\sum_a p(a|b) = 1$, the parameters α_j can be derived from an iterative algorithm called Generalized Iterative Scaling (Darroch and Ratcliff, 1972). ME model represents evidence with binary functions known as contextual predicates in the form:

$$f_{cp,a'}(a,b) = \begin{cases} 1 & \text{if } a = a' \text{ and } cp(b) = true \\ 0 & \text{otherwise} \end{cases}$$

where cp is the contextual predicate which maps a pair of outcome a and context b to {true, false}. By pooling evidence into a "bag of features", ME framework allows a virtually unrestricted ability to represent problem-specific knowledge in the form of contextual predicates. The features in a Maximum Entropy model need not to be statistically independent, therefore is easy to incorporate overlapping and interdependent features. Thus ME models often yield better probability estimates for the observed distribution when statistical evidence from many sources must be combined freely.

2.3.3.1 Feature Selection

Two kinds of features are used in this model: one is single word (term) feature contained in messages, the other is domain specific feature. The former captures the characteristics of the messages to be classified, which is commonly used in Text Categorization, while the later reflects some domain specific properties of the task in hand.

When extracting term feature, one of the standard conventions of the Text Categorization literature is to stem words to their morphological base forms, e.g. both "processing" and "processed" are reduced to root word "process". However, preliminary experiments showed that non-stemming version of classifier performs slightly better than stemmed one. Another factor that we found helps improve classification accuracy is tokenizing scheme. Certain punctuations like exclamation points as part of a term since spammers tend to use phrases like "click here!","FREE!!!"; also special terms such as url, ip address are not splitted and treated as single terms. Words occur in the To, From, Subject lines are treated specially.

On the task of filtering junk mail, domain special features are also precious and should not be treated with ignorance. Junk mails differ from legitimate mails not only in the boast (or offending) words they use, but also in their strange behaviors in various stages of dispatching.

When filtering junk mail, it is important to consider some particular features in the header field of a mail which give strong evidence whether a mail is junk or not. For instance, junk mails are seldom sent through normal email client such as Outlook Express or Mutt. Instead, spammers prefer to use some group mail sending software specially designed for sending mails to a large amount of recipients. This can be detected by looking at the X-Mailer field in the header. If a mail has an X-Mailer field indicating some group sending software or does not have X-Mailer field at all, it is very likely to be a spam. In addition, a good many non-textual features commonly found in spam messages can serve as good indicators to rule out spams. For example, junk mails often do not have user name in their From, To fields or simply use "Dear User" as user name. This can be identified by matching header fields with a pre-defined rule set. Also if all the words in Subject field are capitals, it probably indicates a spam. Entire HTML message containing only one relaying command trying to load an outside url (often spam site) when reading is a new kind of trick played by spammers.

For Text Categorization task, the performance rests heavily on the features selected. A χ2-test is used for feature selection. χ2-test is found to be an effective feature selection method for Text Categorization task (Yang and Pedersen, 1997). The χ2-test measures is defined to be:

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

Where A is the number of times feature f and category c co-occur. B is the number of times of f occurs without c. C is the number of times c occurs without f. D is the number of times neither c or f occurs and N is the total number of documents (messages).

Once a feature set is defined, it is straightforward to incorporate the features into the Maximum Entropy model in a "bag of features" manner. All features will have context predicates in the form:

$$cp_f(b) = \begin{cases} true & \text{if message } b \text{ contains feature } f \\ false & \text{otherwise} \end{cases}$$

Therefore all features have the form

$$f(a,b) = \begin{cases} true & \text{if } cp_f(b) = true \\ false & \text{otherwise} \end{cases}$$

Here, a is the possible category {spam, legitimate} of message b.

"The enhanced model (ME-enhanced) clearly outperformed the baseline model (Naïve Bayes) and standard ME model. Naive Bayes classifier has a higher precision than the two ME models when training data set is small." [ZHANG Le and YAO Tian-shun, 2003].

2.4 Classification

Classification is the process of finding a set of models (or functions) that describe and distinguish data classes and concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown.

Classification is a two-step process.

1. Build classification model using training data. Every object of the data must be pre classified, i.e. its class label must be known. The model can be built using one of the numerous classification algorithms discussed later. It can be represented in the form of classification rules, decision trees, mathematical formulae, or neural networks. Figure below gives an example of classification rules.
2. The model generated in the preceding step is tested by assigning class labels to data objects in a test dataset. The test data is different from the training data. Every element of test data is also pre-classified in advance. The accuracy of the classification model is determined by comparing true class labels in the testing set with those assigned by the model.

2.4.1 Text Classification

Data mining techniques are used to extract knowledge from relational, otherwise called structured, data where each tuple contains a set of attribute-value pairs. Different or modified techniques must be applied to text, which is generally not structured according to attribute-value pairs, or has only some structured elements. Examples of unstructured text are an article body, an e-mail message body, and subject fields.

Text Mining is concerned with extracting of useful knowledge from structured or semis-structured data. Text classification is one of the functionalities of Text mining that can be defined as follows.

Text classification is the supervised learning task of assigning natural language text documents to one or more pre-defined classes (also called categories or topics) according to their content.

Main categories of methods used for text classification are Naive Bayes, Nearest Neighbor, neural networks, Support Vector Machines, regression, decision trees, TF-IDF style classifiers and associative classifiers.

2.4.2 Related Works

2.4.2.1 Naive Bayes Classifiers

Naive Bayesian classifiers are recognized to be among the best for classifying text due to their simplicity, efficiency and updatability. One major drawback is the unrealistic independence assumption among individual terms which is the main idea behind Naive Bayes classifiers. Regardless of this they often produce satisfactory results. Bayes classifiers are probabilistic in nature, i.e. they not only perform a classification but also predict a degree of confidence in the class assigned. This is important when the decision about accepting or rejecting a predicted class must be made. For example, misclassifying a legitimate message into a junk message is much more expensive than the opposite misclassification. To avoid the more expensive case, the decision can be made to “trust” a classifier only if it assigns class “junk” with a very high probability.

The probability of a document d_i to belong to topic C_j is calculated using Bayes’ rule as follows.

$$P(C_j \mid d_i) = (P(d_i \mid C_j) * P(C_j))/P(d_i)$$

The classification result can be either a single topic with maximum predicted probability or the complete probability distribution. Note, that in the first case relative and not absolute probabilities are important, and the document probability P(di) can be ignored because it is the same for all topics. Otherwise, it is calculated as

$$P(d_i) = \sum_{k=1}^{|C|} P(C_k) * P(d_i|C_k)$$

Where, |C| is the total number of topics. The topic probability C_j is estimated by:

$P(Cj) = n_d(C_j)/|D|$

Where $n_d(C_j)$is the number of documents of class Cj and |D| is the total number of documents in the database.

The calculation of P(d_i | C_j) differs depending on which one of two event models is used.

The first one is multi-variate Bernoulli model that treats each document as a set of terms, where each term can appear at most once and all duplicates are eliminated. By this it captures only the presence or absence of a term and ignores its frequency in a document.

In this model, every message is represented as a feature vector of length V, where V is the total number of different terms in the entire message corpus, or vocabulary size. Each element of the vector corresponds to a particular term in the corpus, and is set to one if the corresponding term is present in the message or to zero otherwise. This model assumes that all the terms in a document occur independently from each other and

$$P(d_i|C_j) = \prod_{t=1}^{|\mathcal{V}|} P(w_t|C_j) = \prod_{t=1}^{|\mathcal{V}|} (B_{it}P(w_t|C_j) + (1 - B_{it})(1 - P(w_t|C_j)))$$

Where B_{it} = 1 if wt is present in the document d_i or 0 otherwise.

Under this model,

$$P(w_t|C_j) = \frac{1 + n_d(C_j, w_t)}{2 + n_d(C_j)}$$

Where n_d(C_j ,wt) is the number of documents of class C_j containing the term wt and n_d(C_j) is the total number of documents of class Cj .

The second model is called multinomial and unlike Bernoulli it retains the frequency information, i.e. every element of the feature vector contains a count of the corresponding term in the document. This model assumes not only independence of each term from the other terms but also independence from the position in the document. The probability of a document given its class, P(d_i | C_j), in this case is simply the multinomial distribution

$$P(d_i|C_j) = P(|d_i|) * |d_i|! * \prod_{t=1}^{\mathcal{V}} \frac{P(w_t|C_j)^{N_{it}}}{N_{it}!}$$

Where | d_i | is the length of the document, N_{it} is the count of the number of times word wt occurs in the document d_i. Terms P(|d_i |) and | d_i |! can be disregarded if only relative probabilities are needed.

The probability of each word given the class in this case is:

$$P(w_t|C_j) = \frac{1 + n_w(C_j, w_t)}{|V| + n_w(C_j)}$$

Where $n_w(C_j ,wt)$ is the number of occurrences of word wt in all documents of class C_j and $n_w(C_j)$ is the total number of different words in the documents of class C_j. Under both models, the order of the terms is lost.

2.4.2.2 Associative Classifiers

Associative classification is a combination of two data mining problems, association and classification, that uses association rules to predict a class label. It is based on the suggestion that several documents containing the same frequent term association belong to the same class. The method is gaining popularity due to several reasons. First, the classifier can handle feature spaces of tens of thousands dimensions, while decision tree classifiers are limited to several hundreds attributes only. Second, the classifier is able to consider terms together as a group, which relaxes the independence assumption methods like Naive Bayes are based on. The method also solves understandability problem by generating simple rules. Associative classifiers have been reported to be more accurate than decision tree techniques. They also become more efficient with the invention of new methods for frequent pattern mining [HPY00] that is an underlying operation for classification rule generation.

Under associative classification, the model building stage consists of two steps. First, all rules satisfying some user-specified parameters are generated using association rule mining. The amount of rules generated might be of the order of several millions, and many of them are either redundant, i.e. never used, or noisy, i.e not discriminative among the classes. Second, the redundant and noise rules are pruned. A significant number of rules can be pruned at this stage and the rules left are interesting ones that form a model used to classify new data. This model is called a classifier. Each classifier must have a default rule which is applied when no other classifier rule can be used.

Association rule mining used by associative classifiers for rule generation, was introduced in [AIS93] for market basket analysis. The problem it solves is finding all frequently occurring patterns (also called sets of items or itemsets) in large databases. Let I = $\{i_1, i_2, \ldots, i_m\}$ be a set of items. Let D be a transaction database containing |D| task-relevant transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$.
An association rule r is an implication of the form:

$$A \Rightarrow B,$$

Where $A \subset \mathcal{I}$, $B \subset \mathcal{I}$, and $A \cap B = \emptyset$. A is called an antecedent of r and B is a consequent of r.
A transaction T is said to contain an itemset A if and only if $A \subseteq T$. Two measures that are most often used to measure the interestingness of association rules are support and confidence.
"Support sup of rule $A \Rightarrow B$ in a transaction set D is the percentage of transactions in D containing both A and B."[Julia Itskevitch, 2001].

$$sup(A \Rightarrow B) = P(A \cup B) = \frac{count(A \cup B)}{|D|} = \frac{count(AB)}{|D|}$$

A rule $A \Rightarrow B$ is called frequent or large if sup ($A \Rightarrow B$) $\geq$ min sup, where min sup is a user-specified minimum support threshold. Support reflects usefulness of a rule.
"Confidence conf of rule $A \Rightarrow B$ in a transaction set D is the percentage of transactions in D containing A that also contain B."[Julia Itskevitch, 2001].

$$conf(A \Rightarrow B) = P(B|A) = \frac{P(A \cup B)}{P(A)} = \frac{count(A \cup B)}{count(A)} = \frac{count(AB)}{count(A)}$$

A rule $A \Rightarrow B$ is called confident if conf ($A \Rightarrow B$)$\geq$ min conf, where min conf is a user specified minimum confidence threshold. Confidence reflects certainty of a rule.
A rule is called strong if it satisfies both min sup and min conf.

Sub-Pattern and Super-Pattern

Sub-pattern/super-pattern of a pattern p is any other pattern p_{sub}/p_{sup} obtained by removing/adding one or several items from p. If only one item is removed from/added to p then it is called immediate subpattern/ super-pattern of p.

Sub-Rule and Super-Rule

Sub-rule/super-rule of a rule r is any other rule r_{sub}/r_{sup} whose antecedent is a sub-pattern/super-pattern of the r's antecedent. Immediate sub-rule and super-rule is defined similarly. Notice that the definition does not require the same class label for a sub-rule and its super-rule.

Association Rule Mining

Association Rule Mining is the process of generating all strong rules given user specified min sup and min conf.

Association Rule Mining is a two-step process.

1. Find all frequent patterns. Frequent patterns (also called large patterns) are patterns that satisfy user specified minimum support min sup, where pattern support is the percentage of transactions in the database containing the pattern.
2. Generate association rules that satisfy min sup and min conf.

If a set of items, $\mathcal{I}$ contains both text and non-text e-mail message attributes, the following can be an example of a frequent pattern:

{book, return, pickup, sender = adm@troy.lib.sfu.cag } [sup = 3%]

An example of an association rule is:

{book, return, pickup $\Rightarrow$ sender = adm@troy.lib.sfu.cag } [sup = 3%, conf = 80%]

Further we discuss two associative classifiers, Classification Based on Association (CBA) and Classification based on Multiple Association Rules (CMAR).

Classification Based on Association (CBA)

To make association rules suitable for the classification task, CBA method focuses on a special subset of association rules called class association rules (CARs). CARs are association rules with a consequent limited to class label values only. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of all items in D and Y to be a set of all class labels.

A class association rule r is an implication of the form

$$A \Rightarrow C$$

Where $A \subset \mathcal{I}, C \subset \mathcal{Y}$.Antecedent of a CAR is also called condset and a rule itself is called ruleitem. Ruleitem is frequent if the corresponding rule is frequent and accurate if the rule is confident.

To find all strong association rules, CBA uses an Apriori -like algorithm to generate frequent ruleitems. Apriori-like algorithms are based on the anti-monotone Apriori property that states: if any length k pattern does not satisfy min sup, then none of its length (k +1) super-patterns will satisfy it.

All the frequent rules are generated performing multiple database scans. Let k-ruleitem denote a rule whose condset has k items. In the first pass all frequent 1-ruleitems are found. These 1-ruleitems are used to generate candidate 2-ruleitems. The next scan of the data is performed to count which of the candidate 2-ruleitems satisfy min sup. The algorithm iterates in this fashion, starting each subsequent pass with the seed set of ruleitems found to be frequent in the previous pass. For each frequent ruleitem, the confidence of the corresponding rule is calculated and the rule is added to the set of all rules if the confidence satisfies min conf. The algorithm terminates at the mth iteration if the set of frequent (m-1)-itemset is empty.

To reduce the number of rules generated, the algorithm performs two types of rule pruning. The first type is pessimistic error rate based pruning method in decision tree algorithm and is performed at each iteration of rule generation. According to this pruning technique, a rule is pruned if its pessimistic error rate is higher than that of any of its immediate sub-rules. The second type of pruning is defined in as database coverage pruning. It is applied after all the rules have been generated. Before the pruning, all rules are ranked according to the following criteria, which we call confidence-support ranking criteria.

Given two rules, r_i and r_j. r_i is ranked higher than r_j if

1. $conf(r_i) > conf(r_j)$, or
2. $conf(r_i) = conf(r_j)$, but $sup(r_i) > sup(r_j)$, or
3. $conf(r_i) = conf(r_j)$ and $sup(r_i) = sup(r_j)$, but r_i is generated before r_j.

All the rules are first sorted in decreasing order of their ranks. The sorting guarantees that only the highest rank rules will be selected into the classifier. In the process of database coverage pruning the algorithm iterates through each rule starting from the first one and for each iteration step it finds all messages covered by the current rule. (Rule r is a covering rule for transaction T or transaction T is covered by a rule r if T contains all items in the antecedent of r.)

If among all the messages covered by the rule at least one is classified correctly by the rule, the rule is selected into the classifier and all the messages it covers are removed from the database; otherwise the rule is pruned. The rule selection stops when either all of them are considered or no messages are left in the database. The pruning guarantees that every rule selected classifies correctly at least one message. A default rule is also selected. It has an empty ruleset and predicts a class label, which is a majority class among the message left in the database.

In classifying an unseen case, the case is assigned a class predicted by the first rule covering the case, and thus classified by the rule with maximum possible rank. The default rule is used in case when there are no covering rules.

Classification Based on Multiple Association Rules (CMAR)

CMAR classifier extends CBA by using more than one rule to classify a new case. The efficiency is also improved by applying FP-growth-like algorithms to rule generation.Unlike Apriori that needs as many scans of the database as the length of the longest pattern generated plus one, FP-growth needs only two scans. The issue that is not addressed in CBA is rule redundancy. “A rule r is redundant if any of its sub-rules rsub has higher rank than r” [Julia Itskevitch, 2001].

In the case where several rules are used to classify a single document, redundant rules can also be selected together with their more highly ranked subrules, but this is not desirable since these rules do not provide any new information. CMAR eliminates redundant rules by defining a new data structure, Compressed Rule tree.

CMAR applies yet another type of pruning based on correlation analysis. It is needed to prune noise rules that were not pruned as redundant just because they did not happen to have a sub-rule of higher rank. The method uses a χ^2 test to measure the strength of a dependency between a condset of a rule and its class label. Smaller the value of the statistic, smaller is the dependency. Because a good rule is the rule where its condset is correlated with the class it predicts, good rules have high χ^2 values.

The algorithm prunes all the rules that have χ^2 < 3:84, that corresponds to 95% significance level of the statistics.

The procedure of pruning the rules using database coverage pruning is slightly changed here from that in CBA method. A threshold called database coverage is specified, and a transaction is removed from the database only after it is covered by database coverage number of rules. This increases the number of rules selected into the classifier, and minimizes the number of times the default rule is consulted.

CMAR reasons the use of more than one rule for the classification by the following example. Assume that some transaction is covered by the following rules (support is given in absolute counts):

1. condset1 $\Rightarrow$ C1 (sup = 20, conf=0.95)
2. condset2 $\Rightarrow$C2 (sup = 40, conf=0.94)
3. condset3 $\Rightarrow$ C2 (sup = 50, conf=0.93)
4. condset4 $\Rightarrow$ C3 (sup = 30, conf=0.80)

Rules 2 and 3 have negligibly smaller confidence than rule 1 but much higher support. If CBA method is used they will be classified into C1, even though intuitively they should be classified into C2.

On the other hand, if all covering rules are taken into account, the correct class can be lost among misclassifying rules. To guide the choice of the rules, CMAR defines two thresholds, coverage threshold and confidence difference threshold. The former has the same value as database coverage described above, since it is assumed that training and test data have similar distribution. First maximum coverage threshold rules are selected. If after that any covering rules are left, they are included one by one until the difference of confidences between the very first rule and the current rule does not exceed the confidence difference threshold.

2.4.2.3 Hierarchical Text Classification

A hierarchical topic organization helps to navigate and search information more easily. A hierarchy could be any directed acyclic graph, but we limit the discussion to trees only.

Hierarchical Classification is a process of classification classes are organized by their specificities into an ‘is-a’ hierarchy, also called taxonomy of classes.

Note, that a document class can be a non-leaf node in the hierarchy. Figure below gives an example of a topic hierarchy that e-mail messages can be organized into. The root node is usually empty or can be used to insert documents that could not be classified confidently into any other topic.

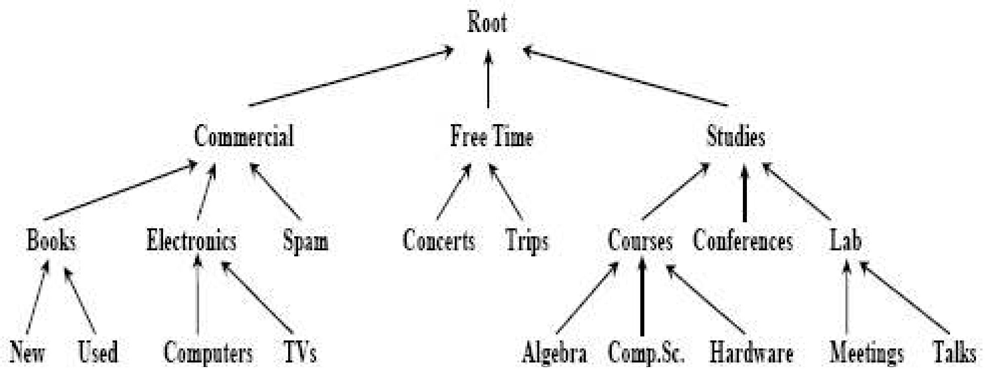


Figure 2 Topic Hierarchy

The tasks of feature selection and classifier construction become more difficult for the case of hierarchical classification because separation of feature terms from noise terms and discriminative rules from noise rules now depends on the current location in the topic hierarchy. For example, for the hierarchy in above figure terms “schedule", “exam", “room" and “grade" may be good feature terms at Courses level but become noise terms for the children of the Courses node. From the implementation point of view, topics at the lower level of the taxonomy have many terms in common with their “parent" topics, and this shared jargon complicates automatic topic separation.

There are two major approaches to handle class taxonomies in classification. The first approach is local and another one is global. Local approach breaks the problem into several sub problems while global builds a single classifier.

2.4.2.4 Clustering Techniques

The above approaches provide a *supervised classification* framework: message folders pre-exist and the main objective is to detect the most likely folder for an incoming message. To our knowledge, few approaches deal with *unsupervised classification*, i.e. the automatic construction of subject-based folders starting from a set of incoming messages. Among these we mention *Scatter/Gather* that uses a complex intermixing of iterative partitional clustering and an agglomerative scheme. The *Athena* system provides a clustering algorithm to produce only cluster digests for topic discovery, and performs a message partitioning on the basis of such digests using a bayesian classifier. Other approaches to organization of text messages, mainly based on collaborative filtering, have been adopted in the definition of agents for accessing usenet news .The main problem with such approaches is that they require interaction with the user in order to learn a suitable organization of messages.

Giuseppe Manco, Elio Masciari, Massimo Ruffolo and Andrea Tagarelli on "Towards An Adaptive Mail Classifier" introduced a mail classification system capable of automatically organizing e-mail messages stored in a mail server, and to provide a web-based e-mail client capable of both suggesting the discovered classification rules and automatically organizing the incoming messages The organization service is mainly based on clustering algorithms. The paper shows how to extract structured and unstructured information from e-mail messages, and adopt a representation model for such information, next proposes a clustering algorithm to accomplish the task of organizing such information, and studied the corresponding accuracy of the proposed system. We here by describe clustering techniques based on that paper.

To obtain a suitable representation of the message collection, we need to provide information about frequencies of index terms in the document. In particular we need to compute two basic measures about frequency: the *term frequency* and the *inverse document frequency*. To evaluate the relevance of a term for content representation, term frequency measures the number of occurrences of an index term. In our particular case, we define the *Message frequency*, denoted by $\varphi(w_j, m_i)$, as the number of occurrences of term w_j within messagem_i. Index terms with high frequency represent at best the message content, especially in long-sized messages. On the other hand, for short-sized messages, the Message Frequency information is likely to be negligible, or even misleading.

In order to significantly represent message contents, it is useful to compare the message frequency of an index term with the number of occurrences within the overall set of messages. Indeed, a term occurring in many different messages is not as discriminator as other terms that appearing in a few messages. This suggests to define the relevance of a term by a combination of its message frequency and an inverse function of the number of messages in which the term occurs. More precisely, we define the *Collection frequency*, denoted by $\psi(w_j, \mathcal{C})$ as the number of messages within a given collection $\mathcal{C} = \{m_{i_1}, \ldots, m_{i_h}\}$containing w_j .In correspondence to collection frequencies, we can associate a further preprocessing step, namely the removal of *implicit stopwords*. By implicit stopword we mean a term w exhibiting a collection frequency not included in a fixed interval $[l, u]$ (i.e., either $\psi(w, \mathcal{C}) > u$or $\psi(w, \mathcal{C}) < l$).

As a result, we can define the (normalized) *term-message* matrix, i.e. the set $\{w_1, \ldots w_N\}$ of the feature vectors, defined as follows:

$$\mathbf{w}_{ji} = \begin{cases} \frac{\varphi(w_j, m_i) \cdot \log(N/\psi(w_j, \mathcal{C}))}{\sqrt{\sum_p [\varphi(w_p, m_i) \cdot \log(N/\psi(w_j, \mathcal{C}))]^2}} & \text{if } l \leq \psi(w_j, \mathcal{C}) \leq u \\ 0 & \text{otherwise} \end{cases}$$

Clustering of Messages

To our purposes, the most interesting problem to deal with is the automatic identification of *interesting* and *important* messages, and their organization in *homogeneous* groups of messages. The most common technique to define interestingness and importance, as well as homogeneity, can be formalized as follows: i) build lists of relevant keywords or phrases from messages; ii) define matching criteria for messages based on such keywords; iii) exploit suitable clustering and categorization schemes to detect and affix labels to each message. The above observations imply the definition of a knowledge extraction process composed of several steps, necessary to prepare and mine raw messages.

For each message extract and store information concerning *Sender*, *Recipients*, *Date/Time*, *Subject*, *Attachment filename*, and *Content* of the message. From the above fields derive a feature vector $x = (y, w)$ that is mainly composed of structured and unstructured information.

Structured information (denoted by y) is obtained from the four main fields, and comprises the followings features:

Categorical	Numeric
Sender domain (e.g. yahoo.com)	Message length
Most frequent recipient radix domain (e.g., gov, com, edu)	Nr. of recipients
Weekday	Nr. of messages received from the same sender
Time period (e.g., early morning, afternoon, evening)	
Attachment file extension (e.g., jpg, ps, xls)	

Unstructured information (denoted by w) is mainly obtained from the Subject and Content fields. In principle, both fields contain unstructured text, and hence need to be processed in a more suitable form.

Formally, we can state the problem as follows: given a set M = {m_1, . . . ,m_N} of messages, find a suitable partition P = {C_1, . . . , C_k} of M in k groups (where k is a parameter to be determined), such that each group contains an homogeneous subset of messages, and has an associated label that characterizes the subset. The statement requires a rigorous definition of the notions of homogeneity and labeling. The notion of homogeneity can be measured by exploiting the feature vectors defined above. Practically, a similarity measure $s(x_i, x_j)$ as a real number, as follows:

$$s(\mathbf{x}_i, \mathbf{x}_j) \quad = \quad \alpha s_1(\mathbf{y}_i^n, \mathbf{y}_j^n) + \eta s_2(\mathbf{y}_i^c, \mathbf{y}_j^c) + \gamma s_3(\mathbf{w}_i, \mathbf{w}_j)$$

Where s_1 defines the similarity of the structured parts of the message composed by numerical features, s_2 defines the similarity of the structured parts of the messages composed by categorical features and s_3 takes into account the unstructured part of the message.

Precisely, a term w is called *characteristic* (w.r.t. a given threshold λ) for a cluster C_j if

1. $\psi(w, C_j) \geq \lambda$.
2. $\psi(w, C_i) < \lambda$, for $i \neq j$.

The intuition around the above definition is the following. First of all, the removal of implicit stopwords, states that the only candidate representatives are terms with significant frequencies. However, in order to correctly characterize the cluster, a term must be peculiar of that cluster: that is, the term must occur with a high frequency within the cluster, and is likely to occur with a lower frequency in other clusters. In principle, the above conditions can be difficult to achieve for a given value of λ. To this purpose, a possible simplification could be that of choosing the three most frequent terms in the cluster that do not appear in other clusters.

2.5 Summarization

Summarization is the creation of shortened version of a text by a computer program. The product of this procedure still contains the most important points of the original text. Broadly, summarization can be done by two approaches: extraction and abstraction. Extraction techniques merely copy the information deemed most important by the system to the summary (for example, key clauses, sentences or paragraphs), while abstraction involves paraphrasing sections of the source document. In general, abstraction can condense a text more strongly than extraction, but the programs that can do this are harder to develop as they require the use of natural language generation technology. Large researches are going on in summarization of text documents, email and even multimedia.

There are various softwares developed to summarize single text documents. Such software typically performs best on well-authored, formal documents. E-mail messages, however, are typically neither well-authored, nor formal. As a result, using the same softwares for email summarization gives poor summary of e-mail messages. Some of the issues in email summarization methods are:

1. Exploiting Email Structure to Improve Summarization
2. Summarizing Email Conversations with Clue Words
3. Combining Linguistic and Machine Learning Techniques for Email Summarization

2.5.1 Exploiting Email Structure to Improve Summarization

This method exploits two aspects of e-mail, thread reply chains and commonly-found features to generate summaries. In this method, email messages are preprocessed using heuristics to remove e-mail signatures, header fields, and quoted text from parent messages. A heuristics-based approach is used in identifying and reporting names, dates, and companies found in e-mail messages as these contain the important information in general.

E-mail threads provide valuable context for summarizing email messages, and allow summarization systems to exploit the structure of e-mail not found in other documents. E-mail threads are groups of replies that, directly or indirectly, are responses to an initial e-mail message. Threads are useful because they have the potential to organize groups of messages around a single topic. Ideally, e-mail threads can reduce the perceived volume of mail in users' inboxes, enhance awareness of others' contributions on a topic, and minimize lost messages by clustering related e-mail. If an enclosing e-mail thread exists, this algorithm processes the e-mail message's ancestors to provide additional context for summarizing the email message.

E-mail messages, especially in the enterprise, tend to center around people and events. Commonly-found features provide clues to the user about the subject matter of e-mail messages. Since much of corporate email centers around collaboration, the system reports names of people and companies, and dates mentioned in e-mail messages. Reporting commonly-found features is intended to be a first-order approximation of the more general goal of reporting summary.

2.5.2 Summarizing Email Conversations with Clue Words

This approach uses clue words to measure the importance of sentences in conversation summarization. A clue word from a node is a word (modulo stemming) that appears also in its parent node(s) and/or child node(s) in the quotation graph. It is important to note that a clue word takes into account simultaneously the content and structure of the quotation graph. The assumption is that if the words reoccur between parent and child nodes, they are more likely to be relevant and important to the conversation. The frequency of clue word appearing in a fragment is used to determine the clue score. Based on clue words and their scores, summary of an email is produced.

2.5.3 Combining Linguistic and Machine Learning

This technique combines linguistic and machine learning techniques to extract the gist or summary of email messages. It is based on the assumption that noun phrases carry the important information and machine learning technique can be applied to determine important noun phrases. The sentences containing the important noun phrases can provide relevant summary of an email. The basic steps of their method are:

- Extraction of candidate noun phrases from the email message. NPs carry the most contextual information about the document, a well-supported hypothesis (Smeaton, 1999 and Wacholder, 1998).
- Linguistic filtering of the candidate NPs, such as removing common words and unimportant modifiers
- Induction of a model that classifies the filtered NPs as important or not. This step is accomplished with extracting features of the candidate NPs and ranking of the NPs with the aid of machine learning algorithms.

The features used for the NPs are:

- Feature associated with the head of the NP
  - **Head TFIDF**: the TFIDF measure of the head of the candidate NP.
  - **Head first occurrence**: The position of the first occurrence of the head in text (the number of words that precede the first occurrence of the head divided by the total number of words in the document).
- Feature associated with the whole NP
  - **NP TFIDF:** TFIDF measure of the whole NP.
  - **NP first occurrence**: The position of the first occurrence of the noun phrase in the document.
  - **NP length words**: Noun phrase length measured in number of words, normalized by dividing it with the total number of words in the candidate NP list.
  - **NP length chars**: Noun phrase length measured in number of characters, normalized by dividing with the total number of characters in the candidate NPs list.
  - **Sentence position**: Position of the noun phrase in the sentence: the number of words that precede the noun phrase, divided by sentence length. For noun phrases in the subject line (which are usually short

and will be affected by this measure), we consider the maximum length of sentence in document as the normalization factor.

- **Paragraph position**: Position of noun phrase in paragraph, same as sent pos, but at the paragraph level
- Feature that considers all constituents of the NP equally weighted
    - Sum of TFIDF of each constituent of NP

3 System Analysis

3.1 Overview

CORVID is an extension to Mozilla Thunderbird that adds the functionality of Email filtering and classification through learning of users actions to emails and also provides the facility of extraction of important information from long sequence of email replies. It applies statistical classification techniques. It uses machine linguistic and learning technique for summarization.

3.2 System Architecture

CORVID consists of 3 independently functioning modules for email filtration, email classification and email summarization. These modules use Knowledge base which will be created during training of the modules during development. The knowledge base will be updated through personalization. These modules will be integrated into Mozilla Thunderbird from where each feature can be used for user email in Thunderbird. Email filtration, classification and summarization modules are developed in C++. XPCOM Component of the engines is called from Javascript. User Interface is added in Mozilla Thunderbird using overlays in XUL.

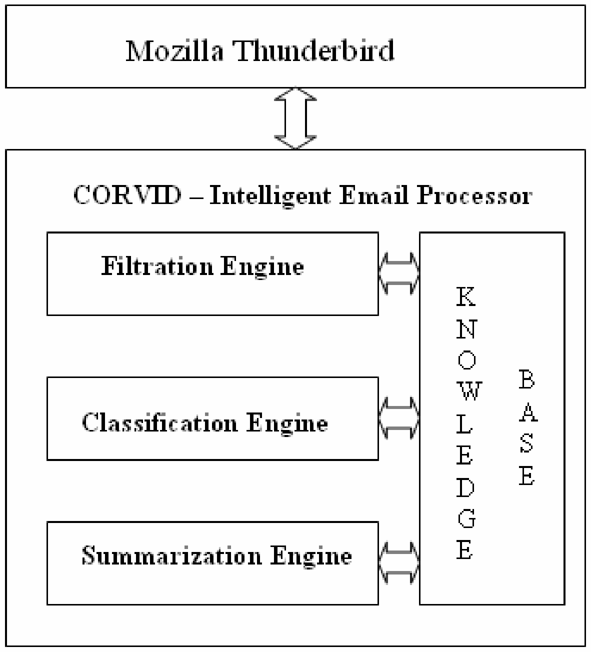


Figure 3: CORVID System Architecture

3.3 Context Level DFD

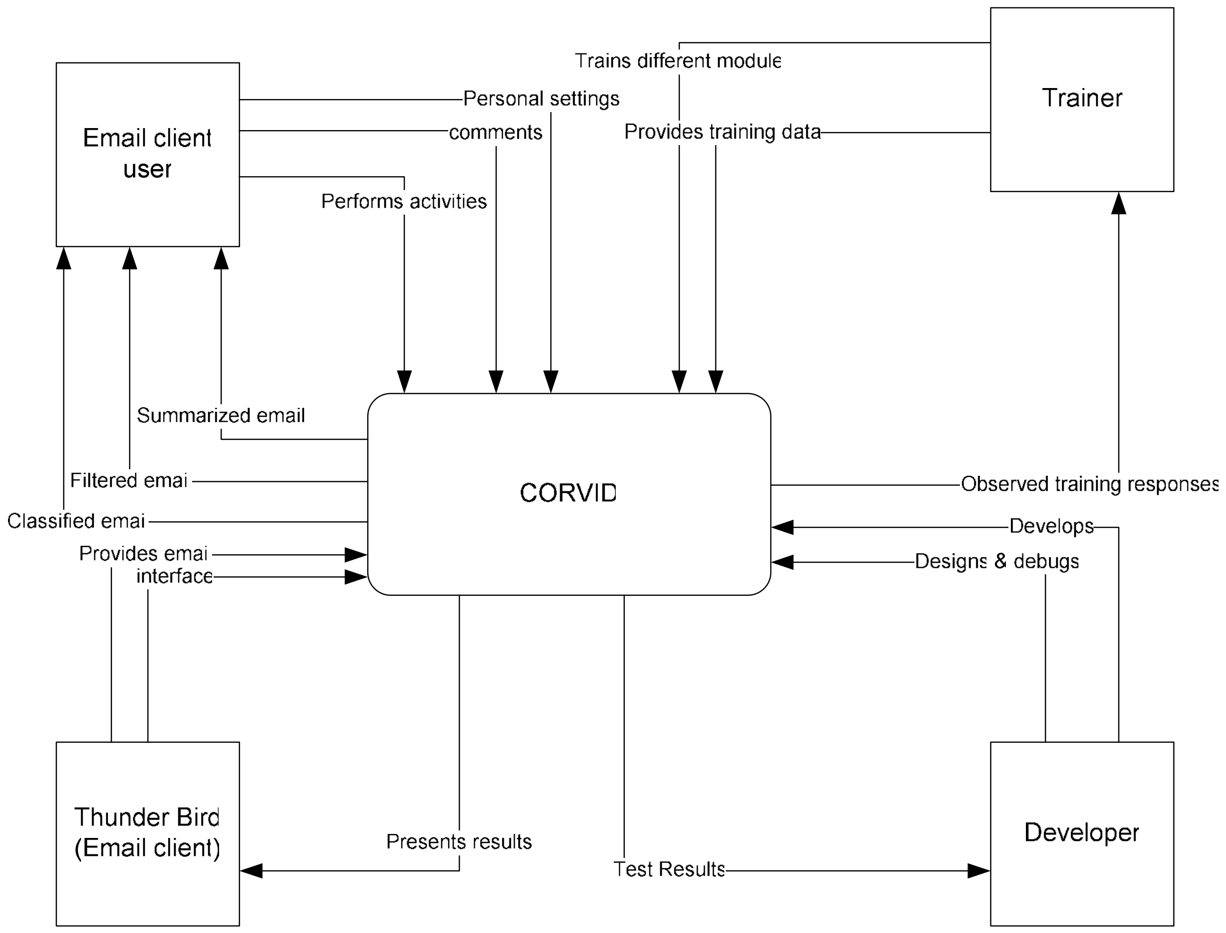


Figure 4: Context Level DFD of CORVID

3.4 Use Case Diagram

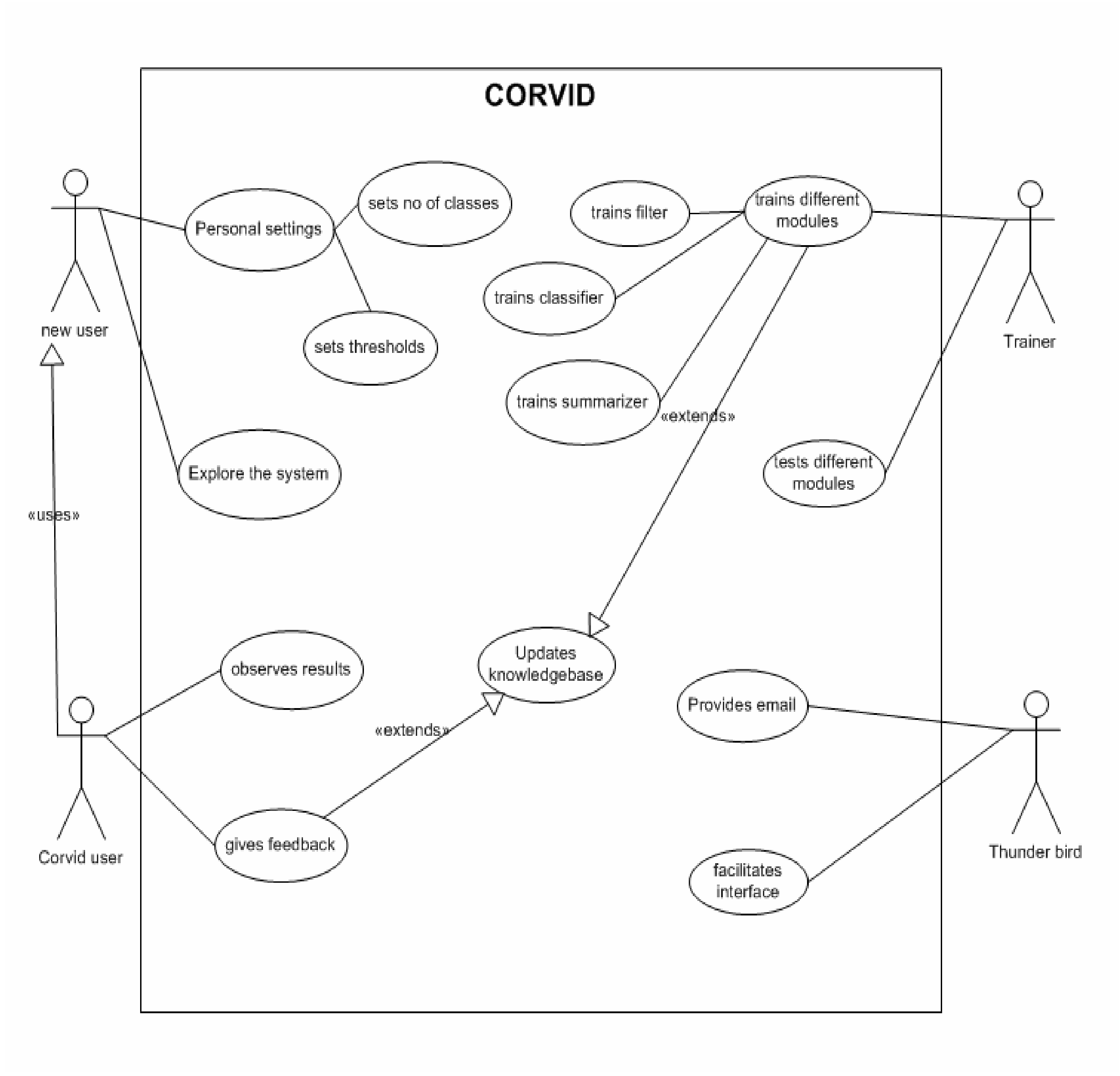


Figure 5: CORVID Use Case Diagram

4 System Design

4.1 Classification

Independent of the algorithm used, the email classification involves first build a model using a training corpora, test the model using test data, then use that model to classifying other emails(as discussed in chapter 2). We are using Cohesion and Multiple Association Rules (COMAR)[Julia Itskevitch, 1997]. The outline of the process involved is outlined in the following figure:

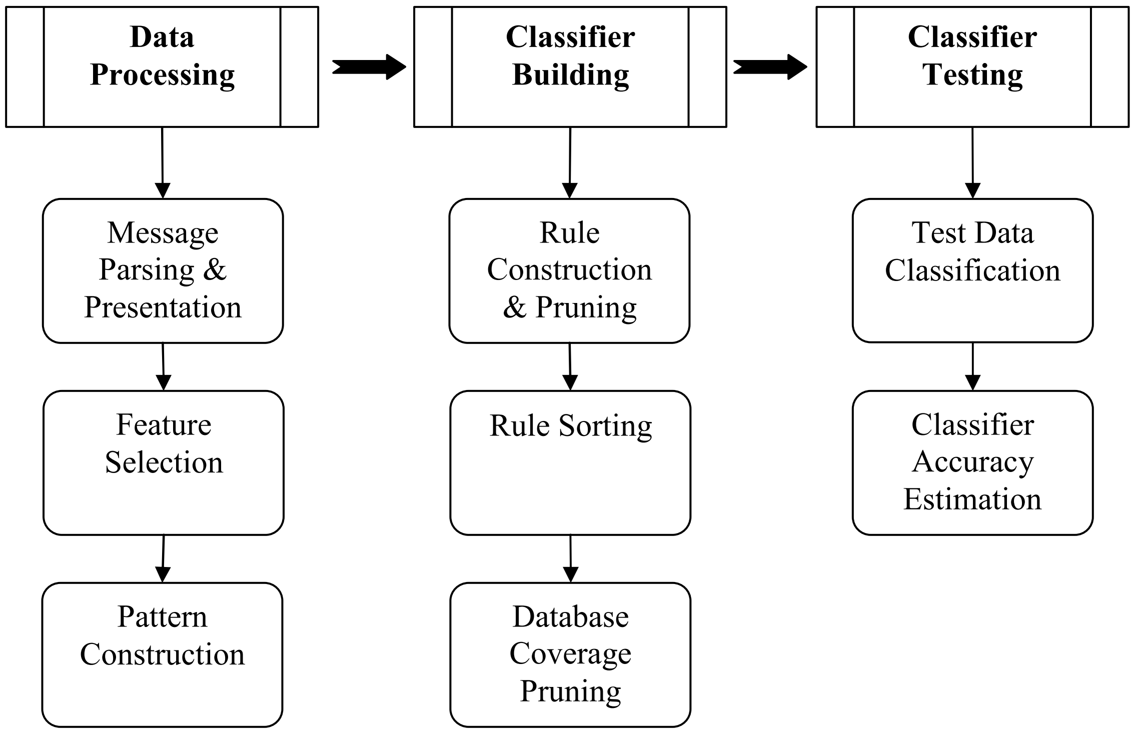


Figure 6: Classification Framework

4.1.1 Data Processing

This involves processing the email message to make it suitable for model building step. There are three steps in data processing:

- Message Parsing and Presentation
- Feature Selection
- Pattern construction

4.1.1.1 Message Parsing and Presentation

Message parsing and presentation is carried out in following steps in sequence:

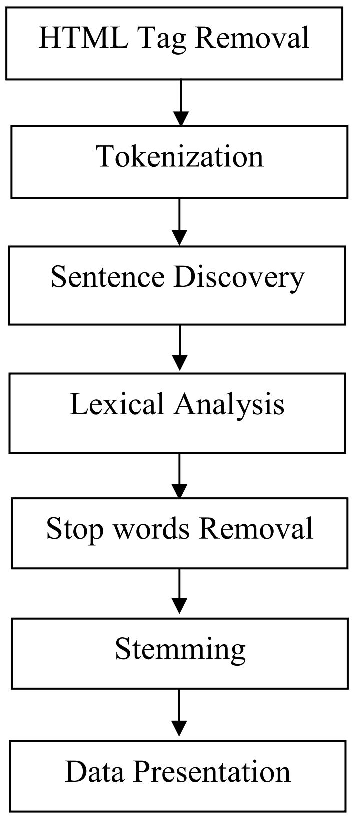


Figure 7: Message Parsing & Presentation (Pre-processing)

HTML Tag Removal

HTML Parsing is done to remove the HTML tags that may be present in an email which do not contain necessary information for email mining. This is done by matching the regular expression of the HTML tags and removing it. The library used for regular expression matching is DEELX (DEELX is a regular expression engine in C++, Perl compatible and is a research project of RegExLab).

Tokenization

Tokenization step separates message body or subject in separate tokens. We are considering the following symbols as sentence separators:

“,;.?\n:”

They if occur in the message they are considered as separate tokens.

Sentence discovery
The tokenized message is nothing, but the same message with tokens just separated out and stored in order in which they occur in the message. Now, using the list of sentence separators we apply the following algorithm [Julia Itskevitch, 1997] to separate sentences.

```
Input: (1) e-mail message m; (2) a list of sentence separators L; (3) average sentence length parameter
Output: a complete set S of sentences in m.
Method:
// p is a set of all words read after the last sentence separator;
// n is the number of new line symbols read after the last sentence separator;
 $p = \mathcal{S} = \emptyset;\ n = 0;$
 while not end of m do {
        read the next token w in m;
        if $w \in \mathcal{L}$ then
            if $|p|/(n+1) \geq l$ then then { // Do not split into shorter sentences.
                $\mathcal{S} = \mathcal{S} \cup p;$
            else {
                split p into n + 1 sentences $S_{n+1;}$
                S = S U$S_{n+1}$;
            }
        $p = \emptyset;\ n = 0;$
        }
        else if w = new line symbol then
            n = n + 1;
        else // If this is a regular word.
        p = p U w;
}
```

Now, the sentences contain tokens grouped in the form of sentences and no token contains sentence separator character or new line.

Lexical Analysis
The sentences discovered are just tokens grouped into sentences. Each of unique tokens adds one dimension to feature space for further processing. So, we need to reduce noise tokens and redundant tokens. Following steps are applied in sequence:

- Eliminate non literal characters from each end of the token, if still non literal letter is present in the token, that token is discarded.
- Discard the token that starts with a digit

Stop words removal

In text classification Ciff's law states that a highly frequent term and very low frequent term carry no information. We discard highly frequent terms in the form of stop words. The highly frequent words that occur equally in almost all messages: prepositions, conjunctions, articles, pronouns, etc. These words are removed from each sentence.

Stemming

In text categorization, the root form of the word and its derivatives carry the same meaning, for example "computer", "computing", and "computers" all belong to the same category determined by the root word "compute". So, we need to stem the derivatives to their root form so that feature space is significantly reduced. We use Porter stemming algorithm (Porter, 2000) for stemming.

Data presentation

Each unique, stemmed, non-stop word from the sentence of message is mapped to a unique integer value. Each different item in the body receives an id from a pool of integer that is initialized to 1 at the beginning. A different pool is created for item in the subject. An id is assigned to each pool. The id of pool is appended before the id of the item to identify where the item appeared.

This transforms the string operation to integer operation, significantly reducing the computational time.

4.1.1.2 Feature selection

Each unique term that appears in the message corpora is a feature, i.e. each unique term adds one dimension to feature space for further processing. Low frequent terms are not important in case of classification. There are lots of ways to remove these non-informative features. We are utilizing the confidence of term t for class c defined as

$$conf(t \Rightarrow c_i) = \frac{count(tc_i)}{count(t)}$$

If a term does not satisfy the minimum value of confidence for any class, it is removed from the feature space.

4.1.1.3 Pattern Construction

A pattern represents a group of terms; in our settings it differs from a sentence in that it represents an association of terms that occur frequently in message corpora. In that case, it represents the contextual dependency of terms in a sentence.

FP-growth Frequent Pattern Growth [HPY00] is an efficient algorithm discovering complete set of frequent patterns. It is at least an order of magnitude faster than the Apriori algorithm (discussed in chapter 2). FP-growth takes a test-only approach and avoids the costly generation of a large number of candidate item sets by adopting a pattern fragment growth method. All necessary database data is stored in a very compact structure called FP-tree (Frequent Pattern Tree).

Formal definition of FPA frequent pattern tree is an extended prefix-tree structure storing crucial quantitative information about frequent patterns [Julia Itskevitch, 1997].

The following algorithm describes the process of FP-tree construction:

```
Input: (1) a transaction database D; (2) minimum support threshold min sup.
Output: FP-tree corresponding to D and satisfying min sup.

Method:
    1) Scan D once and collect the set of frequent items F and their supports.
    Sort F in support descending order as L, the list of frequent items. If several items have the same
    support, and their names are numbers, sort the items in ascending order of their names.
    2) Create the root R of a new FP-tree and label it as “null".
    Create frequent-item header table with |F| entries. Set all head of node-link pointers to null.
    3) For each transaction $T \in D$ do { // Read D the second time.
           Select only frequent items of T into a record P;
           Sort P in the order of L.
           Call insert -tree(P;R);
    }
```

```
Procedure insert-tree(P, R){
    1)  Let P = [p|P- p], where p is the first element of P, and P - p is the remaining list.
    2)  if R has a child N such that N.item-name=p then
    3)  N:count = N:count + 1;
    4)  else {
    5)  create a new node N;
    6)  N:count = 1;
    7)  N:item - name = p;
    8)  N:parent = R;
    // Make node-link of p point to the first item with the same item-name.
    // H is a frequent item header table.
    9)  N.node-link=H(p).head;
    10) H(p):head = N;
    11) }
    // Increase by 1 count of item p in frequent-item header table H.
    12) H(p):count = H(p):count + 1;
    13) if $P - p \neq \emptyset$ then
    14) Call insert-tree(P - p;N) recursively.
    }
```

Once the frequent pattern tree is constructed from the transaction database, we mine the frequent patterns using frequent pattern growth (FP-growth) algorithm.

The FP-growth method takes the divide-and-conquer approach to frequent pattern mining. The method recursively searches for shorter frequent patterns and then _nds longer patterns by concatenating a frequent item with a shorter frequent suffix pattern.

“FP-tree not only allows compact data representation but also facilitates an efficient frequent pattern mining” [Julia Itskevitch, 1997, page 43].

The FP-growth algorithm is as follows:

Input: (1) FP-tree Tree for database D; (2) minimum support threshold min sup;

Output: a complete set of frequent patterns F.

Method: Call FP-growth (Tree, null), where null is the initial frequent suffix.

```
Procedure FP-growth($Tree$, $\alpha$) {
1)  $\mathcal{F} = \emptyset$;
2)  if $Tree$ contains a single path $P$ then {
3)      for each combination $\beta$ of the nodes in $P$ do {
4)          generate pattern $p = \beta \cup \alpha$;
5)          sup($p$) = minimum support of nodes in $\beta$;
6)          $\mathcal{F} = \mathcal{F} \cup p$;
7)      }
8)  }
9)  else for each $a_i$ in the header of $Tree$ starting from the least frequent do {
10)     generate pattern $\beta = a_i \cup \alpha$;
11)     $sup(\beta) = sup(a_i)$;
12)     $\mathcal{F} = \mathcal{F} \cup \beta$;
13)     construct $\beta$'s conditional pattern base;
14)     construct $\beta$'s conditional FP-tree $Tree_\beta$;
15)     if $Tree_\beta \neq \emptyset$ then
16)         call FP-growth($Tree_\beta$, $\beta$);
17) }
}
```

4.1.2 Classifier Building

We develop Cohesion and Multiple Association Rules (COMAR) [Julia Itskevitch,1997] classification model by training corpora of email. It involves the following steps (as shown in figure)

- Rule Construction and Pruning
- Rule Ranking
- Database Coverage Pruning

4.1.2.1 Rule Construction

Rule construction and pruning is similar to Pattern construction and pruning (discussed previously) but with following modifications:

• FP-growth method is extended to perform frequent class association rule (CAR) generation. We call the new method EFP-growth (Extended Frequent Pattern Growth) and the corresponding data structure EFP-tree (Extended Frequent Pattern Tree).

• A rule cohesion (rc) measure is defined and used for rule ranking and pruning. The rc measure is slightly different from pc (phrase/pattern cohesion) in that it incorporates topic information.

• CR-tree structure is used to prune those redundant rules that could not be pruned during frequent rule generation. Every node of CR-tree contains not just one pattern and its rank but the pattern distribution among different topics and a corresponding rank for each ruleitem.

The following are major modifications to the FP-tree that make it suitable for class association rule mining:

1. Count value of every node is replaced in the EFP-tree with the node class distribution, where each element of the distribution stores a number of transactions of the current class containing a pattern from this node to the root. The corresponding count value of each node in the FP-tree can be obtained by summing up all the elements in the distribution.
2. Similarly, every item count value in the frequent-item header table of the FP-tree is now replaced with item class distribution.
3. Every pattern inserted into the tree is sorted in ascending frequency order to make the deep rule pruning possible.

Class Support supc of ruleitem $A \Rightarrow C$ for class C in a transaction set D is the percentage of transactions of class C in D containing A.

$$sup_c(A \Rightarrow C) = P(A \cup C|C) = \frac{count(A \cup C)}{|C|} = \frac{count(AC)}{|C|}$$

A ruleitem is class frequent for class C if its supc ≥ min supc and supc ≥ minα.
This definition allows different absolute support for different topics. This minimizes the impact of unbalanced training sets, but for some under represented topics this might create the situation where a ruleitem having very smallabsolute count still satisfies min supc. This is not desirable because, first, the number of rules generated will explode exponentially and, second, a lot of noisy terms might be introduced that will negatively affect the accuracy of the classifier. That is why another threshold minα is specified. An item is considered non-frequent if its count for class C is less than absolute minimum count, even if it satisfies min supc. Requiring a pattern

inserted to be not just a frequent pattern but a frequent ruleitem significantly decreases the size and number of recursive EFP-tree's built.

The following algorithm [Julia Itskevitch, 1997] describes the process of EFP-tree construction.

```
Input: (1) a transaction database D; (2) minimum class support threshold min supc.(3) obsolute
minimum count minα;
Output: EFP-tree corresponding to D and satisfying min supc and minα.
Method:
    (1)  F = ∅; // A set of all items that are frequent for at least one class.
    Scan D once and fill out array M, where M[i, j] gives the count of item of class cj; also fill out
    array C where Cj is the total number of messages
    (2)  for each item ti in M do {
    (3)  if at least for one j M[i, j] ≥ minα and M[i, j]/C[j] ≥ min supc then
    (4)  F = F U ti;
    (5)  }
    (6)  Sort F in support ascending order as L, the list of frequent items, where the support is global
         among all classes.If several items have the same support, and their names are numbers, so the
         items in ascending order of their names.
    (7)  Create the root R of a new EFP-tree and label it as "null".
    Create frequent-item header table with |F| entries. Set all head of
    node-link pointers to null.
    (8)  for each transaction T ∈ D do { // Read D the second time.
    (9)  $c_T$ = class of transaction T;
    (10) for each item tj ∈ T do {
    // M[i,j] corresponds to item tj and class cT ;
    (11) if M[i, j] ≥ minα and M[i, j]/C[j] ≥ min supc then
    (12) select tj into a record P;
    (13) }
    (14) Sort P in the order of L.
    (15) Call insert tree(P, $c_T$ ,R);
    (16) }

Procedure insert tree(P, c, R) {
            1)  Let P = [p|P − p], where p is the first element of P, and P − p is the remaining list.
            2)  if R has a child N such that N.item-name=p then
```

```
3)  N.count(c) = N.count(c) + 1;
4)  else {
5)  create a new node N;
6)  for all classes ci do
7)  if ci = c then N.count(ci) = 1 else N.count(ci) = 0;
8)  N.item − name = p;
9)  N.parent = R;
// Make node-link of p point to the first item with the same item-name.
10) N.node-link=H(p).head;
11) H(p).head = N;
12) }
// Increase by 1 count of item p in frequent-item header table H.
13) H(p).count = H(p).count + 1;
14) if P − p ≠ ∅ then
15) Call insert tree(P − p, c,N) recursively.
16) }
```

Almost all of the properties of FP-tree that make it an efficient data structure for frequent pattern mining equally apply to the case of mining strong CARs using EFP-tree. What is different is that now any frequent ruleset built using a_i as a frequent suffix has not only the same support of its condset as a_i, but also the same class distribution.

The algorithm for mining all strong CARs using EFP-tree is given below.

Input:

(1) EFP-tree Tree for database D;

(2) minimum class support threshold min supc;

(3) minimum confidence min conf; (4) absolute minimum count minα;

Output: A complete set of strong CARs S.

Method: Call EFP-growth(Tree, null), where null is the initial frequent suffix.

```
Procedure EFP-growth(T ree, α) {
1) S = ∅;
2) if Tree contains a single path P then
3) for each combination β of the nodes in P do {
4) generate pattern p = β U α;
5) for each class Ci do {
6) sup(p ⇒ Ci) = minimum support of p ⇒ Ci among nodes in β;
7) conf(p ⇒ Ci) = sup(p ⇒ Ci) / sup(p);
8) if count(p ⇒ Ci) ≥ minα and
sup(p ⇒Ci) ≥ min supc and
conf(p ⇒ Ci) ≥ min conf then
9) S = S U (p ⇒ Ci);
10) }
11) }
12) }
3) else for each a_i in the header of T ree starting from the most frequent do {
14) generate pattern β = a_i U α;
15) for each class C_i do {
16) sup(β ⇒C_i) = sup(a_i ⇒ C_i);
17) conf(β ⇒ Ci) = sup(β ⇒ Ci)/sup(β);
18) if count(β ⇒ Ci) ≥ minα and
sup(β ⇒ Ci) ≥ min supc and
conf(β ⇒ Ci) ≥ min conf then
19) S = S U (β ⇒ Ci);
20) }
21) construct β's conditional pattern base;
22) construct β's conditional EFP-tree Treeβ;
23) if T reeβ ≠ ∅ then
24) call EFP-growth(T reeβ, β);
25) }
}
```

Rule Cohesion (rc): For a rule t1, . . . , tn ⇒ c of length n rule cohesion (rc) is a ranking measure defined as

$$rc(t_1, \ldots, t_n, c) = \frac{C(t_1, \ldots, t_n, c)}{\sqrt[n]{C(t_1) * \ldots * C(t_n) * C(c)}}$$

Where C(t1, . . . , tn, c) is a number of transactions where the rule terms and the topic occur together, C(ti), i = 1, . . . , n, is a number of transactions containing ti, and C(c) is a count of messages classified to topic c in a training set.
It is proved [Julia Itskevitch, 1997] that rule cohesion measure enjoys the anti-monotone property, i.e. for each rule r its super rules have rule cohesion value less than or equal to that of r. This, property enables deep pruning while generating the rules.

Redundant rules are pruned partially using rc measure.

4.1.2.2 Rule Ranking

rc(rule cohesion) is used as the primary criterion for rule ranking.Given two CARs, r_i and r_j, r_i is ranked higher than r_j if,

1. rc(ri) > rc(rj), or
2. rc(ri) = rc(rj), but sup(ri) > sup(rj), or
3. rc(ri) = rc(rj), sup(ri) = sup(rj), but length(ri) > length(rj), or
4. rc(ri) = rc(rj), sup(ri) = sup(rj), length(ri) = length(rj), but ri is generated earlier than rj.

Those rules that are redundant according to these criteria are pruned .For efficiency; this pruning is performed during rule generation based on the rc measure.

4.1.2.3 Database Coverage Pruning

After the rules are ranked and sorted in decreasing rank order, the highest ranked rules are selected into the COMAR classifier, while the others are pruned using database coverage pruning. Database coverage pruning ensures that every rule selected into the classifier classifies correctly at least one message and that each training message is covered by several rules (the number is specified by the database coverage threshold parameter) of the highest possible rank (as discussed in chapter 2). The following algorithm formalizes the process of database coverage pruning.

Input:

(1) a transaction database D; (2) a set of non-redundant rules R; (3) coverage threshold V;

Output: (1) a set of rules Rc used by COMAR classifier; (2) a default rule rdef.

Method:

```
1) rdef = (∅⇒ Cm), where Cm is the majority class in D;
2) R = Sort(R); // Sort the rules according to their rank.
3) for each message mi ⊆ D cover count cci = 0;
4) initialize Rc to empty set: Rc = ∅;
5) do {
6) for each rule ri ⊆ R do {
7) find a set Dcov ⊆ D of messages covered by ri;
8) if at least one m ⊆ Dcov is correctly classified by ri then {
9) Rc = Rc U ri;
10) for each message mj ⊆ Dcov do {
11) update cover count of mj : ccj = ccj + 1;
12) if ccj = V then
13) remove mj from D: D = D \ mj ;
14) }
15) }
16) }
17) } until end of R or D = ∅;
```

After database coverage pruning, all the rules selected as classifier are grouped in the classes they classify to.

4.1.3 Classifying Messages using COMAR Classifier

As was pointed out in [W. Li., 2002], a single rule, even that one with the maximum rank, does not provide enough information for an accurate classification. On the other hand, the whole set of covering rules might be too large and contain misleading rules. When selecting multiple classifying rules for message m, we first find all rules covering m and then select several strongest rules from all the covering rules.

4.1.4 Classification Class Diagram

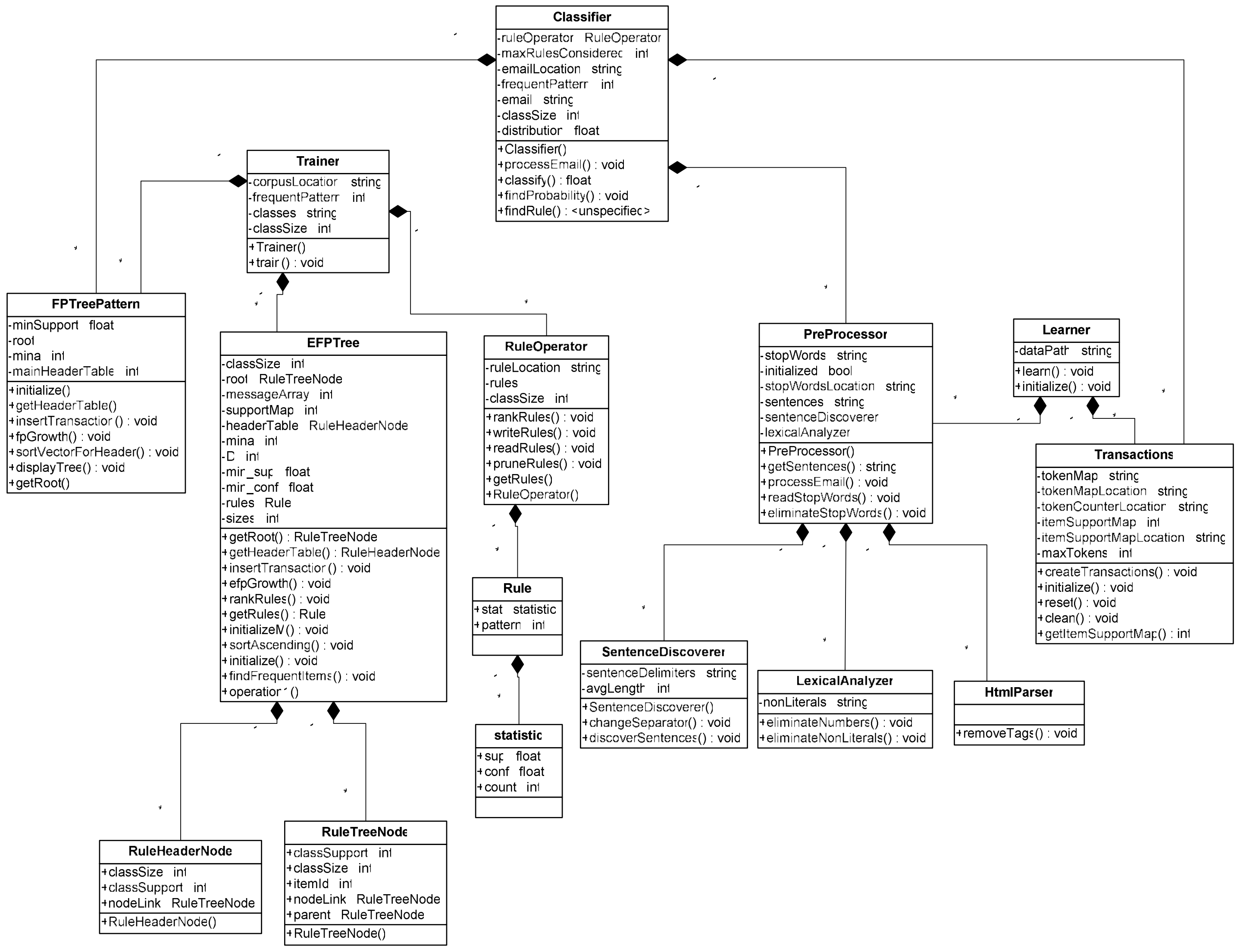


Figure 8: Classification Class Diagram

4.2 Filtration

Filtration is a special case of classification in that the process of filtration is similar to that of classification. Input to filtration module is a message and output from it is a probability value with which the message can be classified as spam. The filtration process, just like classification, is: build a filter model, train it from a training set of emails, test using a training set, and use that model to filter other incoming emails (as shown in figure below). Though, the process steps are similar in nature to those of classification, their operation are different in nature. In fact difference is inherent to the problem of classification and filtration: filtration has to deal specially with content of the message and some headers. Some headers carry more information for filtration whereas none for classification and some items appear identical for classification for filtration which is treated differently in filtration.

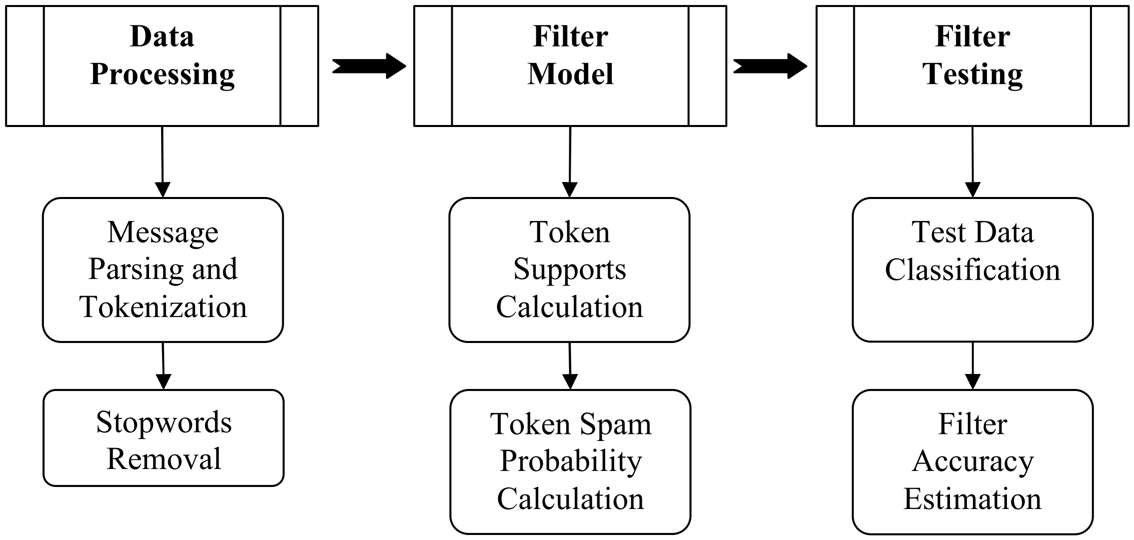

Data Processing
Filter Model
Filter Testing
Message Parsing and Tokenization
Stopwords Removal
Token Supports Calculation
Token Spam Probability Calculation
Test Data Classification
Filter Accuracy Estimation

Figure 9: Filter Design Framework

4.2.1 Data Processing

Text email is processed so as to make it useful for the filter model.

4.2.1.1 Message Parsing and Tokenization

Email in the form of plain text is parsed to separate headers and body part. Each email header or body content is tokenized. All the space characters (space, new line, and tab) are considered as token separator. All Special characters are considered part of the token except that the special characters can not occur at the ends of tokens. Also, tokens are not stemmed, since different form of a word carry different value for spam probability. Header name is appended at the beginning of token to distinguish the same token occurring in different headers.

4.2.1.2 Stop words removal

This process is identical to that of classification.

4.2.2 Filter Model

Filter model is a set of tokens, each token mapped to probability of that token to be in spam. The input to this sub module is a set of messages processed to give a set of tokens and output is the filter model. This module has following components:

4.2.2.1 Token Supports Calculation

Token supports are two maps: one mapping a token to number of spam messages in which that token appears and the other mapping to that of ham messages. In this case, the frequency of a term in the same message is not considered.

4.2.2.2 Token spam probability calculation

Token supports are used to calculate probability of each token to be in spam message. Each token is then mapped to corresponding probability value. The probability calculation pseudo code is as follows [Paul Graham, 2002]:

```
(let ((g (* 2 (or (gethash word good) 0)))
    (b (or (gethash word bad) 0)))
  (unless (< (+ g b) 5)
   (max .01
       (min .99 (float (/ (min 1 (/ b nbad))
                      (+ (min 1 (/ g ngood))
                        (min 1 (/ b nbad)))))))))
```

Where word is the token whose probability we're calculating, good and bad are the hash tables created in the first step, and ngood and nbad are the number of nonspam and spam messages respectively.

4.2.3 Test Data Classification

Incoming message is processed to obtain the frequent tokens. For each token, spam probability is obtained from the model. Then user specified number of interesting tokens (the tokens that have probability value distant from 0.5) is selected. Using these individual probabilities of selected tokens, combined probability is calculated using the formula given by Paul Graham as follows:

```
(let ((prod (apply #'* probs)))
  (/ prod (+ prod (apply #'* (mapcar #'(lambda (x)
                                          (- 1 x))
                                      probs)))))
```

4.2.4 Filtration Class Diagram

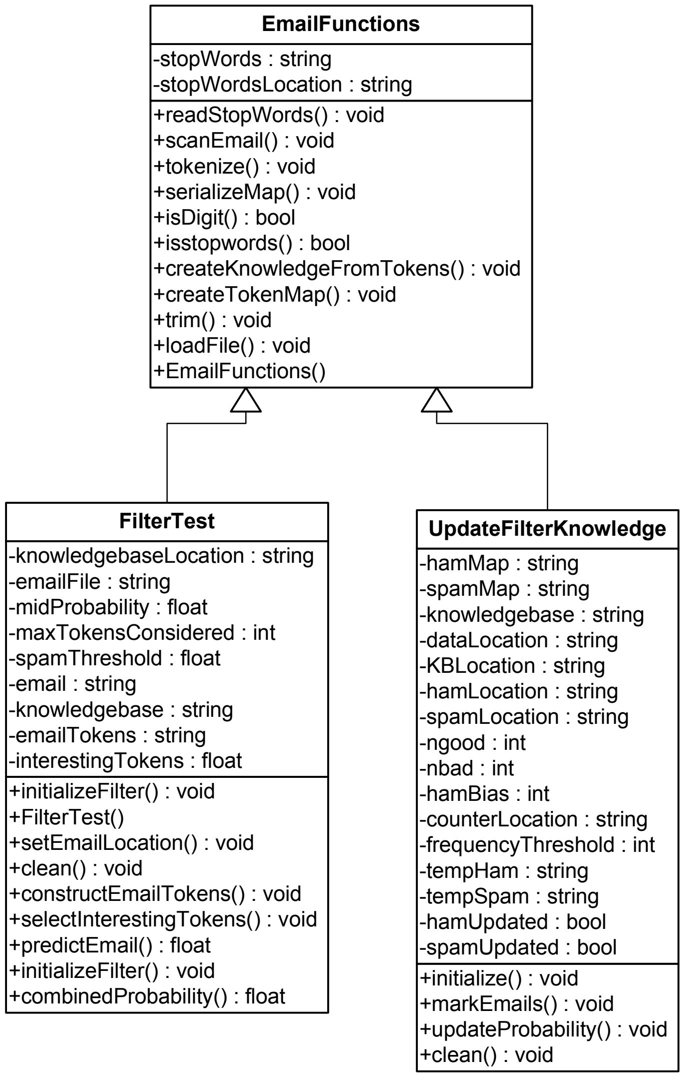


Figure 10: Filtration Class Diagram

4.3 Summarization

The input to the summarization module is a single email message. The architecture of this module as presented in Figure above consists of four distinct functional components email preprocessing, Noun Phrase extraction and Filtering unit, Machine Learning unit and presentation. Machine Learning unit consists of a feature selection module and a classifier to train statistical features of noun phrases in an email summary. This module requires a training set consisting statistical features of important sentences in an email.

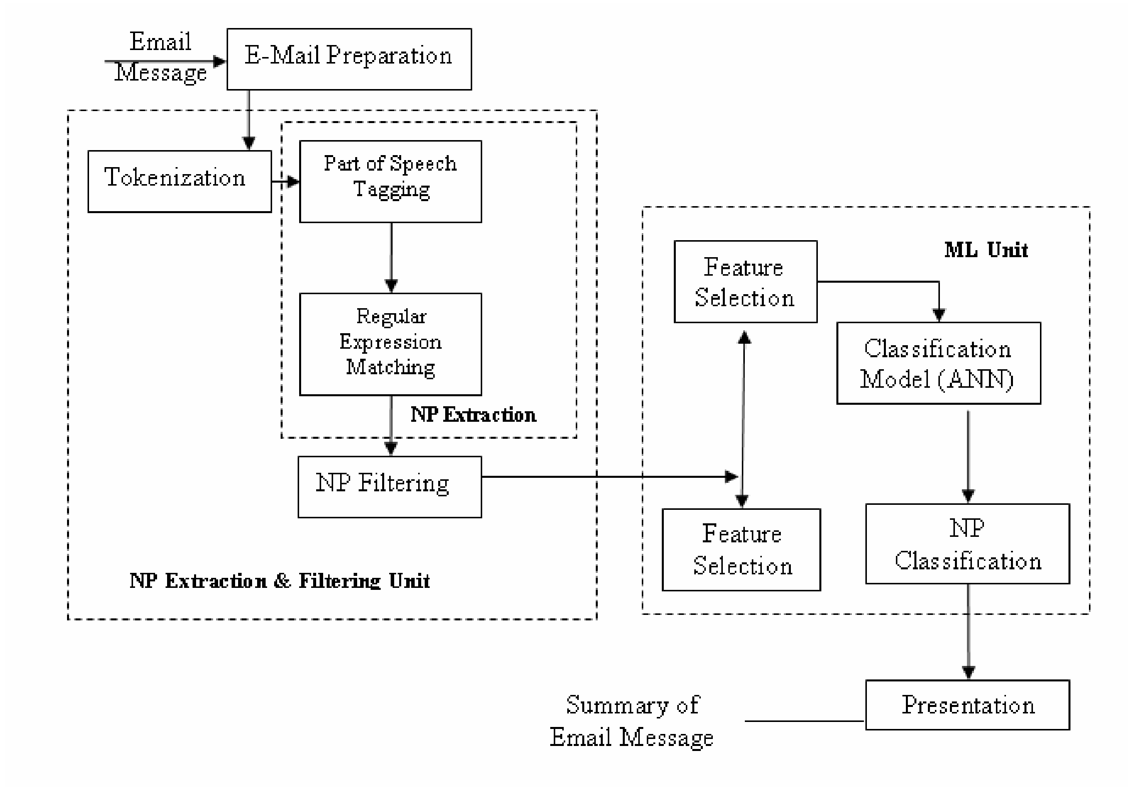


Figure 11: Email Summarization Framework

The steps are as follows

4.3.1 HTML Parsing

This is same as HTML parsing described in email classification.

4.3.2 Email Preprocessing

The plain text email obtained after HTML parsing may contain unnecessary spaces and punctuation marks which are removed and the whole email is converted to lowercase.

4.3.3 Tokenization

In this process, tokens are collected from an email. Tokens are the combination of alphabets and numbers separated by token separator. We have considered white spaces, tabs and line feeds as token separator.

4.3.4 POS tagging

POS of each tokens obtained is to be determined. For this, we have used Mark Watson's POS tagging library. This works as follows.

- Each token is binary searched in the dictionary that contains the most frequently appearing words (mostly verbs, adjectives, preposition etc.) with their respective POS.
- If the token is not present in the dictionary, certain transformational rules are applied to determine the POS.
    - DT, {VBD | VBP} --> DT, NN
    - convert a noun to a number (CD) if "." appears in the word
    - convert a noun to a past participle if words[i] ends with "ed"
    - convert any type to adverb if it ends in "ly"
    - convert a common noun (NN or NNS) to a adjective if it ends with "al"
    - convert a noun to a verb if the preceding work is "would"
    - if a word has been categorized as a common noun and it ends with "s", then set its type to plural common noun (NNS)
    - convert a common noun to a present participle verb (i.e., a gerund)

4.3.5 NP Extraction

Noun Phrases are extracted by matching the regular expression of Noun Phrase with POS tagged email. The regular expression for NP is

(Adjective | Noun) (Noun Preposition)? (Adjective | Noun)* Noun*

4.3.6 Feature Extraction

Features such as TFIDF, frequency count, Position etc are determined for each NP. The TFIDF (Term Frequency Inverse Document Frequency) is calculated as below:

The *term frequency* in the given document is simply the number of times a given term appears in that document. This count is usually normalized to prevent a bias towards longer documents (which may have a higher term frequency regardless of the actual importance of that term in the document) to give a measure of the importance of the term t_i within the particular document d_j.

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

where $n_{i,j}$ is the number of occurrences of the considered term in document d_j, and the denominator is the number of occurrences of all terms in document d_j.

The *inverse document frequency* is a measure of the general importance of the term (obtained by dividing the number of all documents by the number of documents containing the term, and then taking the logarithm of that quotient).

$$\mathrm{idf_i} = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

With

$|D|$: total number of documents in the corpus

$|\{d_j : t_i \in d_j\}|$: number of documents where the term t_i appears (that is $n_{i,j} \neq 0$).

Then

$$\mathrm{tfidf_{i,j}} = \mathrm{tf_{i,j}} \cdot \mathrm{idf_i}$$

4.3.7 NP Ranking

The set of extracted features of each NPs are then given as the input set to the ANN and then the output is compared and sorted to rank the NPs(Noun Phrases). We used lwneuralnet-0.8 as ANN in our development.

4.3.8 Sentence Extraction

The sentence is discovered and extracted for each of the important noun phrase ranked above. The sentence is discovered by finding the full stops or new line following the desired noun phrase.

4.3.9 Summary Presentation

The extracted sentence are then ordered and presented as summary as per required.

4.3.10 Summarization Class Diagram

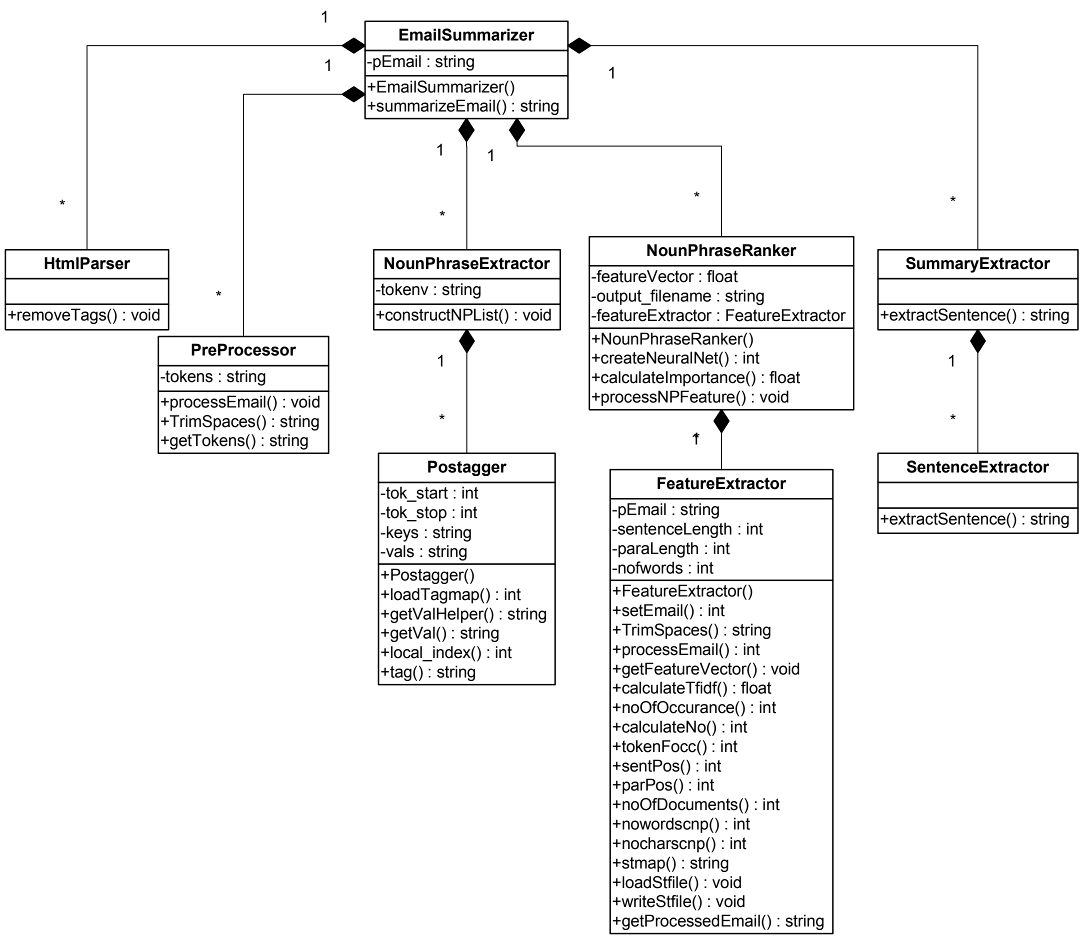


Figure 12: Summarization Class Diagram

4.4 Extension Implementation

4.4.1 Extension Folder Structure

```
corvidioe@mozilla.org/
          chrome.manifest
          install.rdf
          install.js

          content/
              xul and js files

          components/
              IFiltration.xpt
              IClassification.xpt
              ISummarization.xpt
              IFiltration.dll
              IClassification.dll
              ISummarization.dll

          defaults/
               preferences/
                    defaults.js

          data/
               filtration/*
               classification/*
               summarization/*

          locale/
              en-US/
                    dtd files

          skin/
               overlay.css
```

chrome.manifest – determine the packages and overlays the extension provides.
install.rdf - file is used by the Extension Manager when installing the extension, used to provide extension metadata such as extension's ID, version, description, author, contributors etc
install.js – extension install script in Javascript
content directory – consists of overlays, XUL and Javascripts for event handling

components directory – consists of the cross platform binary file and corresponding dll of the C++ XPCOM Component
defaults directory – consists of default setting files including preferences
data directory – consists of the knowledge bases for email filtration, classification and summarization
locale directory – consists of the DTD files for internationalization and localization
skin - apply a specific look to the extension component

4.4.2 Extension Packaging

```
corvidioe@mozilla.org/
          chrome.manifest
          install.rdf
          install.js

          chrome/
               content/

               components/

               defaults/

               data/

               locale/
                    en-US/

               skin/
```

The contents inside the chrome folder is zipped with extension .jar and the whole package is zipped with extension .xpi also called zippy file. The file install.js contains the script for installation of the extension.

4.4.3 Creating XPCOM Component

To achieve the raw processing power of C++, the main engines for filtration, classification and summarization are developed in C++. But these have to interface with the Mozilla thunderbird mail. This is done by creating the XPCOM component.

The steps in creating XPCOM Components are explained below with an example on Filtration Component.

1. `IFiltration.idl` is created in XPIDL and the GUID is generated for the interface.

```
#include "nsISupports.idl"
[scriptable, uuid(660bd8b8-b07a-11dc-8314-0800200c9a66)]
interface IFiltration : nsISupports
{
        short InitFilter(in string dataPath);
        short InitUpdate(in string dataPath);
        float PredictEmail(in string email);
        long UpdateKnowledge(in string email, in string status);
};
```

   Generate the interface header and typelib files out of the interface definition file using the xpidl utility that comes with Gecko SDK.

   - `xpidl -m header -I_DIR_ IFiltration.idl` will create the `IFiltration.h` header
   - `xpidl -m typelib -I_DIR_ IFiltration.idl` will create the `IFiltration.xpt` cross platform binary file
2. The interface header file `IFiltration.h` contains templates for building our component header and implementation files.
3. Create Filtration component header extending IFiltration
   - Create a GUID for your component.
   - Define component name, contract ID, and GUID
   - Create component implementation file - Filtration`Component.cpp and add implementation code.`
4. Create module definitions file - Filtration`ComponentModule.cpp`.
5. In the C++ Project, select output mode to dll.
6. Copy the dll and xpt files into extension components directory and create the component's instance Javascript.

5 Development Scenario

5.1 Tools and Environment

Gecko SDK, which is a collection of libraries and tools that features the XPCOM component framework. Also consists of XPIDL compiler to create the typelib and header files for XPCOM Component.

Microsoft Visual C++ Express - IDE from Microsoft to develop and test filtration, classification and summarization engines

Mozilla Thunderbird – CORVID is an extension to Mozilla Thunderbird.

XUL Explorer - tool for experimenting with XUL snippets

`guidgen` utility - GUID Generator i.e. Globally Unique Identifier

DOM Inspector - used to inspect and edit the live DOM of any web document or XUL application

Other tools – Textpad, Mozilla Firefox

5.2 Methodology

Our project, CORVID, is a research oriented project, therefore traditional approaches like Waterfall model is not suited for the development process. We have followed agile software methodology, but we have not strictly adapted to the principles of agile method.

Some of the practices we have followed during our project are listed below.

- Pair Programming
- Stand-Up Meetings
- Code Formatting & Standards
- Method Comments
- Code Ownership

6 Further Works

The algorithms implemented are bounded by performance and efficiency. Both performance and efficiency can be altered to some extent to achieve better reliability at the cost of efficiency. Nevertheless, there is still much work to be done for optimization of all the modules of CORVID.

The filtration module is totally based on the knowledge base of the system. Adding domain specific knowledge such as black-list and white- list email list will enhance the output of the engine. Further, reliability can be increased by giving priority to the personalized data in the knowledge base than the initial data in knowledge.

The classification involves phrase extraction and rule generation steps. The rules created are core part in determining the class of the email. The number of rules can grow exponentially with the growing number of incoming mails. This leads to addition of large number of noisy rules in the knowledge. These rules misguide the classification process and also reduce the efficiency. Thus efficient rule pruning when done in the process of rule generation can reduce such unwanted rules.

The noun phrase plays important part in summarization process. The noisy and redundant noun phrase extracted creates more dimensions in feature space. So, appropriate noun phrase filtering technique employed at the beginning of the summarization process can significantly enhance the performance. Also, specific information such as address, company name, dates and time are more important in email and better be considered separately.

7 Concluding Words

The project CORVID aims at automating the task of user inbox management by implementing appropriate algorithms for email mining. We are very much hopeful our work will help users in managing their inbox efficiently.

The product mainly focuses on email filtration, summarization and classification tasks. During the process of development, we studied many research works carried out in their respective fields. Based on the results and conclusions provided by such papers we selected algorithms that best addressed our project scenario. We carefully carried out trade off decisions in every step of these algorithms. Thus, the results obtained are reliable to some degree.

Despite our great effort, there is still a lot of work and research to be done in the field of email mining.

8 References

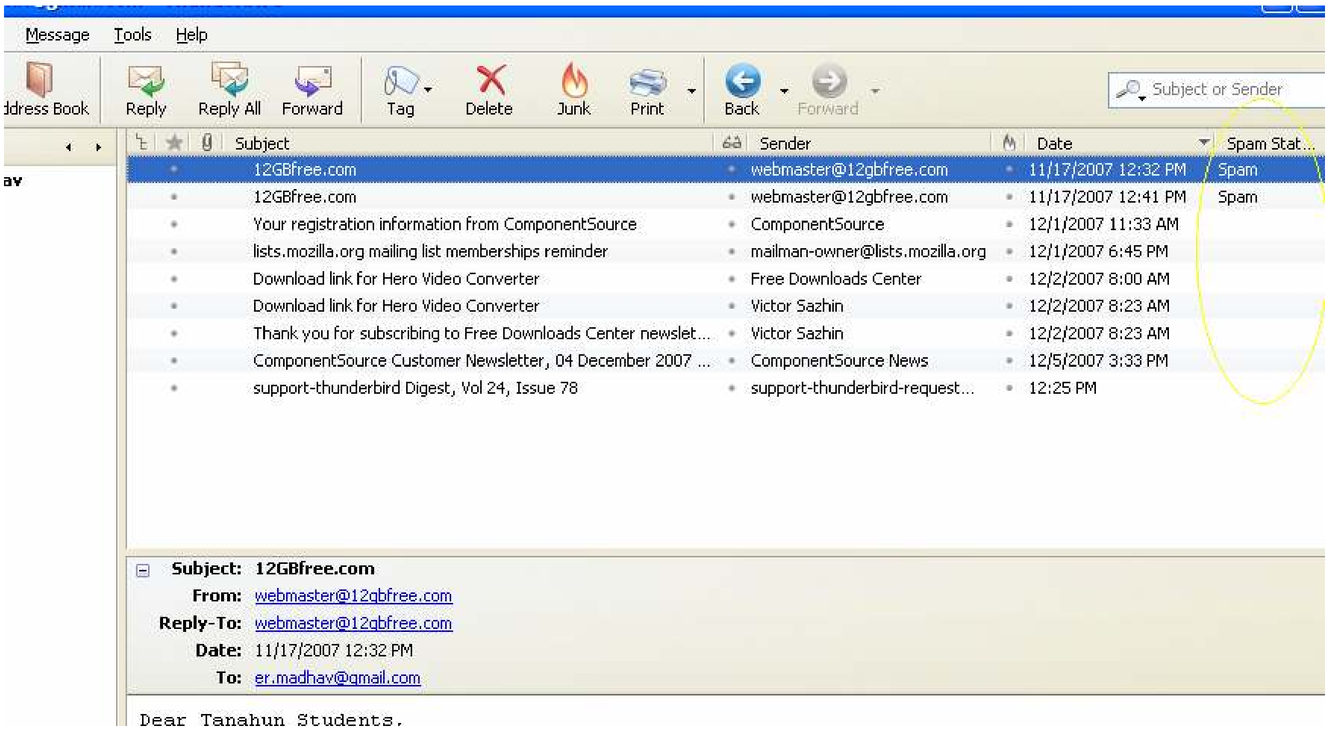
1. Ruth Bergman, Martin Griss, Carl Staelin, A Personal Email Assistant, HP Laboratories Palo Alto, HPL-2002-236, August 22nd , 2002
2. John Graham-Cumming, Three years of Spam Mutation. In the Proceedings of The Spammer's Compendium, January 2003
3. Marti Hearst, Applied Natural Language Processing, In the Proceedings of Presentation on Natural Language Processing for Email, October 2004
4. Jason D. M. Rennie, ifile: An Application of Machine Learning to Email Filtering. At the Artificial Intelligence Lab Massachusetts Institute of Technology, Cambridge.
5. Xavier Carreas and Llujes Marquez, Boosting Trees for Anti-Spam Filtering, TALP Research Center, September 2001.
6. Victor R. Carvalho & William W. Cohen, Learning to Extract Signature and Reply Lines from Email, in CEAS 2004
7. Mehran Sahami, Susan Dumais, David Heckerman and Eric Horvitz, A Bayesian Approach to Filtering Junk Mail. In the Computer Science Department Stanford University.
8. Gary Boone, Concept Features in Re-Agent, an Intelligent Email Agent, Georgia Institute of Technology.
9. Richard Cole, Gerd Stumme, CEM- A Conceptual Email Manager. School of Information Technology, Griffith University Gold Coast Campus
10. Giuseppe Manco, Elio Masciari, Massimo Ruffolo, Andrea Tagarelli, Towards An Adaptive Mail Classifier
11. Julia Itskevitch, Belorussian, Automatic Email Classification using Association Rules. State Polytechnic Academy, 1997
12. Shaun Abushar and Naoki Hirata, Filtering with Intelligent Software Agents
13. Smaranda Muresan, Evelyne Tzoukermann, Judith L. Klavans, Combining Linguistic and Machine Learning Techniques for Email Summarization
14. Xiaodong Zhou, Email Mining and Summarization
15. [Li01] W. Li. Classification based on multiple association rules. Master's thesis, Simon Fraser University, 2001.

16. M.F.Porter,An algorithm for suffix stripping,1980

17. Paul Graham, A Plan for Spam, 2002

18. Paul Graham, Better Bayesian Filtering, 2003

9 Appendices

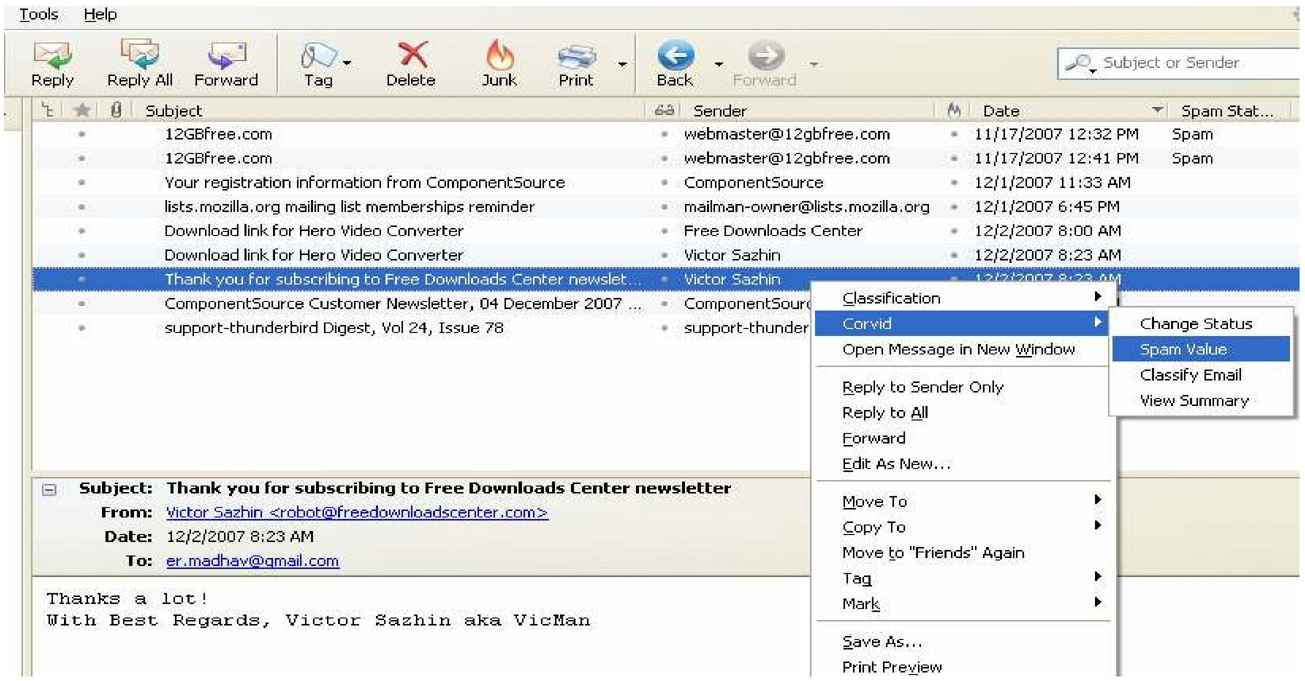


Subject	Sender	Date	Spam Stat...
12GBfree.com	webmaster@12gbfree.com	11/17/2007 12:32 PM	Spam
12GBfree.com	webmaster@12gbfree.com	11/17/2007 12:41 PM	Spam
Your registration information from ComponentSource	ComponentSource	12/1/2007 11:33 AM	
lists.mozilla.org mailing list memberships reminder	mailman-owner@lists.mozilla.org	12/1/2007 6:45 PM	
Download link for Hero Video Converter	Free Downloads Center	12/2/2007 8:00 AM	
Download link for Hero Video Converter	Victor Sazhin	12/2/2007 8:23 AM	
Thank you for subscribing to Free Downloads Center newslet...	Victor Sazhin	12/2/2007 8:23 AM	
ComponentSource Customer Newsletter, 04 December 2007 ...	ComponentSource News	12/5/2007 3:33 PM	
support-thunderbird Digest, Vol 24, Issue 78	support-thunderbird-request...	12:25 PM	

Subject: 12GBfree.com
From: webmaster@12gbfree.com
Reply-To: webmaster@12gbfree.com
Date: 11/17/2007 12:32 PM
To: er.madhav@gmail.com

```
Dear Tanahun Students.
```

Figure 13: Snapshot showing CORVID email filtering in Thunderbird



Subject	Sender	Date	Spam Stat...
12GBfree.com	webmaster@12gbfree.com	11/17/2007 12:32 PM	Spam
12GBfree.com	webmaster@12gbfree.com	11/17/2007 12:41 PM	Spam
Your registration information from ComponentSource	ComponentSource	12/1/2007 11:33 AM	
lists.mozilla.org mailing list memberships reminder	mailman-owner@lists.mozilla.org	12/1/2007 6:45 PM	
Download link for Hero Video Converter	Free Downloads Center	12/2/2007 8:00 AM	
Download link for Hero Video Converter	Victor Sazhin	12/2/2007 8:23 AM	
Thank you for subscribing to Free Downloads Center newslet...	Victor Sazhin	12/2/2007 8:23 AM	
ComponentSource Customer Newsletter, 04 December 2007 ...	ComponentSourc		
support-thunderbird Digest, Vol 24, Issue 78	support-thunder		

Subject: Thank you for subscribing to Free Downloads Center newsletter
From: Victor Sazhin <robot@freedownloadscenter.com>
Date: 12/2/2007 8:23 AM
To: er.madhav@gmail.com

```
Thanks a lot!
With Best Regards, Victor Sazhin aka VicMan
```

Figure 14: Snapshot showing CORVID features