Universität Paderborn
Fakultät für Elektrotechnik, Informatik und Mathematik
AG Wissensbasierte Systeme

Diplomarbeit

Application of Machine Learning Techniques to Spam Filtering

Vorgelegt von:

Thorsten Timm

zur Erlangung des akademischen Grades

Diplom Wirtschaftsinformatiker

Vorgelegt bei:

Hochschuldozent Dr. habil. Benno Stein

Paderborn, 01.03.2004

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen und Hilfsmittel wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum	Unterschrift

Contents

List of Figures

List of Tables

1 Introduction

Email has become a big and continually growing factor in communication. The possibility of information exchange within seconds over all distances combined with low transaction costs makes this medium successful.

But the advantages of email communication are also attractive to direct marketers who use spam to reach a large number of potential customers. According to a study from 1997 [CL98] 10% of the emails sent to a corporate network were spam emails and it is not likely that the quota has decreased since then. The statistics of Brighmail, a developer of antispam software for large organizations, confirm this. Figure 1.1 shows that the "percentages of total internet email identified as spam" [bri04] has increased from 42% in January 2003 to 58% in December.

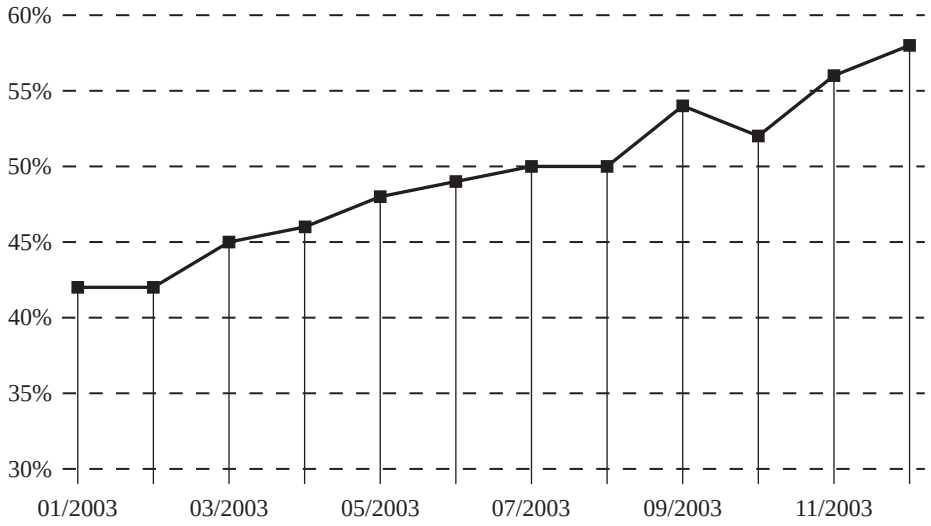

Figure 1.1: Spam statistics collected by Brightmail

The problem with spam is that it is costly to companies as well as individuals. For companies the costs are obvious. On one hand employees have to waste valuable working time by sorting out unwanted emails. On the other hand, spam is data that has to be stored and transferred within the corporate network. Storage space and bandwidth are an expense factor as well. The costs for individuals are similar to that of companies. However, low bandwidths and costs depending on connection time or amount of data transferred (e.g. receiving email by WAP) are more common in this group, which leads to even higher costs.

Another problem is that the contents of spam emails might be offensive to

the recipients (i.e. children) since they also advertise pornographic contents besides get-rich schemes and other dubious offers.

Several attempts have been made to protect the user from spam by filters. The positive effect of such filters is not only that the number of incoming unsolicited emails can be reduced, but it also takes the means of existence from the sender of such email. The spammer depends on the response rate to his emails to earn money and to cover his expenditures for collecting email addresses and the effort of sending the emails to the recipient. When the response rate decreases enough, the revenues do not cover the costs anymore and this marketing channel becomes futile.

This paper concentrates on finding spam within the daily email flow by using machine learning techniques. The next chapter explains the basic vocabulary and definitions needed within the following text. It also gives an overview of other filtering approaches that do not use machine learning. Chapter 3 describes the different filter types and presently existing spam filter approaches using these techniques. The following chapter takes a look at how these approaches might be improved followed by the description of an implementation of a spam filter in chapter 5. This filter's performance is evaluated in chapter 6 and finally, chapter 7 summarizes the results and gives an outlook on future developments in this field of research.

2 Spam Fundamentals

2.1 Definitions

Email is the short form of electronic mail. Email is "the transmission of messages over communication networks" [ema04]. There are some email systems that are confined to a company's computer network but usually the term is used for messages sent through the Internet. Emails are composed to one or several recipients using an email editor. They usually contain text but emails are also used to send other contents like images, movie files or computer programs. The sender's email composer transmits this data to a server-computer on the Internet using the standard protocol SMTP (Simple Mail Transfer Protocol). This server forwards the email to the recipient's email server which stores it. The recipient can access his emails using either the protocol POP (Post Office Protocol) or IMAP (Internet Message Access Protocol). IMAP is similar to POP but it supports additional features such as searching through email messages while they are on the server.

Emails can be divided into two types: Legitimate emails that are desired by the recipient and spam emails. This expression is a synonym for unsolicited bulk email (UBE) or unsolicited commercial email (UCE). The source for this term is a sketch by Monty Python in which a group of Vikings wants to eat in a restaurant which offers so much spam (a brand of canned ham) that other food is hard to find in the menu. There are several definitions for spam. However, the most common is:

"An electronic message is 'spam' IF: (1) the recipient's personal identity and context are irrelevant because the message is equally applicable to many other potential recipients; AND (2) the recipient has not verifiably granted deliberate, explicit, and still-revocable permission for it to be sent; AND (3) the transmission and reception of the message appears to the recipient to give a disproportionate benefit to the sender." [spa04].

As described in the previous chapter, these emails can have a negative impact on the recipient in several ways. Spam filters are used to avoid this impact. They try to identify spam emails and prevent them from getting into the user's inbox. The filter uses only certain criteria from an email to detect spam. These criteria are called either "features" or "attributes", the second term will be used in this text.

The attributes that are derived from an email to categorize it with a spam filter form the set $\mathcal{A} = \{a_1, \ldots, a_n\}$. Which attributes are derived from the

email depends on the implementation of the filter. However, they usually contain attributes of the email header (i.e. sender address, used servers, sending time) and attributes of the email body (i.e. words, HTML-tags, number of punctuation marks). The way in which words are recognized differs again. For example all character strings including or excluding special characters can be words. Additional the recognition can be case sensitive or not. The spam filter decides by these attributes whether an email should be treated as spam or not.

There are several ways in which the filter can deal with detected spam emails. In some approaches emails recognized as spam are just deleted, others send a reply to the author of this email telling him to rewrite the message to an unfiltered private address. Another option is that these messages are just labeled with a keyword in the subject line.

Which of these methods is used always depends on the error rates of the filter. There are two possible errors that spam filters can make: On the one hand they can misclassify spam emails as legitimate. The ratio of not filtered spam emails and the total number of spam emails is called the false-negative rate. On the other hand there are legitimate emails that are misclassified as spam. The corresponding error rate is called false-positive rate. Let N_L be the total number of legitimate emails and N_S be the total number of spam emails. The number of spam emails categorized as spam is $N_{S\to S}$. Likewise, $N_{S\to L}$ is the number of false-negatives, $N_{L\to L}$ the number of correctly classified legitimate emails and $N_{L\to S}$ the number of false-positives. Then the variables r_{fn} and r_{fp} shown in the calculations below are the false-negative and the false-positive rate. The labels have been adopted from [AKC+00].

$$r_{\text{fn}} = \frac{N_{S\to L}}{N_S}$$

$$r_{\text{fp}} = \frac{N_{L\to S}}{N_L}$$

It is obvious that the second kind of error should be avoided more than the first: A legitimate email that is filtered and never reaches the recipient is much worse than the case of a spam email that has to be deleted manually. For example a bill that is sent by email and that does not reach the recipient can lead to serious trouble. However, there is a correlation between the two error types. A filter can usually be configured to be more sensitive for detecting spam and therefore find more spam emails. This behavior leads to a lower false-negative rate. But this configuration usually leads to a higher false-positive rate, too.

Most filters are biased towards producing less false-positives. According to this bias, the method of how to deal with recognized spam is chosen. If false-positives are not very likely to occur, emails categorized as spam might just be deleted or moved to a certain folder. If they are more likely to occur, a method like replying to the sender or marking the email in the subject line is more appropriate.

2.2 Testing

A spam filter is usually tested in two steps. First, the filter is trained and then it is tested with other emails. Therefore two different email corpora are necessary, the training corpus and the testing corpus. These two corpora should have the same source and an equal share of spam in both of them is reasonable, too.

In most approaches, one corpus is split into two parts that form the training and the testing corpus. Some researchers also use k-fold cross-validation to make their results more reliable. This method splits the corpus into k parts. In the first run, the training is done with the first $k-1$ parts of the corpus and the last part is used for testing. The second run uses part $k-1$ as testing corpus and so on. In this way, k-fold cross-validation generates k runs and more reliable results than one run. In practice the most commonly used value for k is 10.

The simplest values calculated for comparing the performances of the different filters are the false-positive and false-negative rates described above. However, these numbers do not enable us to compare filters directly if the weighting between both error types is not clear. The only situation in which it is possible to decide between two filters based on the false-positive and the false-negative rate is when both of them are higher for one filter than for the other.

If we want to compare two filters and one of them outperforms the other one in detection of spam while the other one misclassifies less legitimate email, we need to introduce a new value. This value defined in [AKC+00] is the weighted error rate W_{Err}. To calculate it, we have to define the coefficient λ. An email categorized false-positive is λ times as costly as a false-negative. The weighted error rate is the ratio of the sum of errors made and the number of emails in the corpus where false-positives and the number of legitimate emails are weighted with λ:

$$W_{\mathrm{Err}} = \frac{\lambda \cdot N_{L \to S} + N_{S \to L}}{\lambda \cdot N_L + N_S}$$

The appropriate value of λ depends on how emails classified as spam are handled. If they are directly deleted a high value like 999 is appropriate, if they are just marked, it might be as low as one.

The research group of Anroutsopoulos continues to manipulate this parameter. They compare it to a baseline value, the weighted error rate when no spam filter is present. This means that no spam emails are recognized. The baseline weighted error rate is calculated as follows:

$$W^b_{\mathrm{Err}} = \frac{N_S}{\lambda \cdot N_L + N_S}$$

This baseline divided by the weighted error rate results in the total cost ratio TCR:

$$\mathrm{TCR} = \frac{W_{\mathrm{Err}}^{b}}{W_{\mathrm{Err}}} = \frac{N_S}{\lambda \cdot N_{L \to S} + N_{S \to L}}$$

This seems to be a good tool to evaluate the filter results intuitively because a value of less than one means that for this value of λ the performance of the spamfilter is worse than using no filter at all. A greater number means a better performance of the filter. But the problem of the total cost ratio is that it is not a function of the number of legitimate emails. Meaning that for ten falsely filtered legitimate emails out of 100 the total cost ratio is the same as for ten out of 1000. This circumstance makes it impossible to use the total cost ratio to compare filters tested on email corpora with different ratios of legitimate and spam emails.

For this reason, the weighted error rate is used in this paper to quantify the results achieved by the spam filters. The values of λ taken into account are 9, 99 and 999.

If a scientific paper uses different filter configurations, for each λ the configuration resulting in the best weighted error rate is used for the comparison. However, when different implementations are compared by these figures, it must be taken into account that a filter optimized for each λ will be able to achieve better results.

Although the weighted error rate is a good tool to compare filters, it is not a very intuitive value. For this reason, the false-positive and the false-negative rate are calculated for each filter, too. They are much easier to understand and help to compare the filters.

2.3 Filtering Spam with Conventional Methods

Spam filters can follow machine learning concepts to recognize spam. Several filters using these approaches to adjust to incoming legitimate and spam emails will be explained in chapter 3. These filters can adjust to the individual taste of a user which email is spam and which is not. Other "conventional" filters that do not adapt to the email received by the user and his reading habits are explained in the following.

2.3.1 Rule-Based Filters

Rule-based filters use information from the header or the body of the email and verify whether it meets certain rules. These rules can be either simple, e.g. "the subject line starts with 'Free' ", or more complex, e.g. "text to image

area ratio is less than 0.1".

The user can also decide whether all emails that meet a certain rule should be regarded as spam or if this decision should be made by another rule. This could be whether the number of matched rules exceeds a certain threshold. This calculation can be enhanced by assigning coefficients to rules and summing up the coefficients of rules met. This also enables the generation of positive rules like "The sender is listed in the address book" that produce negative coefficients.

These rules must be manually generated or acquired from a third party. On the one hand manual generation of such rules is a complex work that requires an adequate knowledge of constructing such rules and extensive testing. On the other hand rules acquired from a third party are not likely to meet everyone's opinion as to which emails are regarded as spam, and spammers might also adapt their emails to these filters. This enables them to prevent their emails from identification as spam, which is especially likely to happen if one rule-set is widely used. Therefore rule-based filters without any user-defined rules are not an appropriate way of filtering spam emails.

One example of a popular rule-based spam filter is SpamAssassin [spab], a server-side solution. A presentation of SpamAssassin was held at the Spam Conference 2003 [spaa] by Matt Sergeant who describes how the software works [Ser03].

SpamAssassin uses 938 rules which are rated with different scores. These are not only rules that analyze the contents of the message; SpamAssassin also uses other forms of spam detection which are integrated by a rule system. For example, it checks several freely available spam catalogues like Razor, Pyzor or DCC. A closer look at this type of spam prevention will be taken in 2.3.4. Another approach to spam detection integrated into SpamAssassin is a Naive Bayesian Classifier described in detail in 3.1.

All rules that are part of SpamAssassin are weighted by a multiplier and added up. Weights can be either positive, if a corresponding rule is biased towards spam emails, or negative, if the rule is biased towards legitimate emails. If the sum exceeds a certain threshold, the email is classified as spam. The weights are determined by a genetic algorithm which is not explained. Chapter 3.2 gives an overview of Genetic Algorithms and their application to spam filtering.

2.3.2 Blacklists

Blacklists are a drastic method of filtering spam emails. The basic concept is that a central server stores all computers that have been or might be sources of spam emails. They either collect the sources of spam emails or scan email servers for open relays. Open relays are misconfigured servers that allow spammers to send emails to any email address via this server. The spam detection software checks incoming email for its origin and determines if the server is

listed in the blacklist. Emails from listed servers are filtered.

Popular implementations of Blacklists are Spamcop [spac], the Realtime Blackhole List (RBL) offered by the Mail Abuse Prevention System (MAPS) [rbl] or the Open Relay Database [ord]. Blacklists are a controversial issue since a blocked mail server always means many false-positives. An example of this is GMX [gmx], a German email provider that was in the blacklist of Spamcop for a 10 day period in September 2003 [hei03b] and in the Open Relay Database in May 2003 [hei03a]. During these periods, customers of the blacklist providers would not receive emails sent to them from any GMX user. As a result of this incident, there have been lawsuits against operators of blacklists. An example of this is the blacklist Open Relay Blackhole Zones (ORBZ) being shut down in March 2002 following a judgment in Michigan [hei02].

2.3.3 Whitelists

This approach only lets emails reach the recipient if the sender is registered in a list of approved contacts. Unknown senders must contact the recipient first in a certain way to get access to his inbox. This technique is even more restrictive than blacklists since it does not allow the use of email as a medium to establish contact.

2.3.4 Checksum Database

The Distributed Checksum Clearinghouse (DCC) [dcc] is a distributed system that identifies spam by comparing emails to a database of registered spam emails. Whenever a new email arrives, a checksum is calculated from it. This checksum is sent to a distributed network of servers. These servers count the number of occurrences of each checksum. When the number of occurrences of one checksum exceeds a certain threshold, it is regarded as bulk email and emails with this checksum are therefore filtered.

This basic technique is backed by so-called fuzzy checksums which ignore some aspects of messages in order to find emails that are substantially identical. These fuzzy checksums allow recognizing bulk emails although the spammer inserts certain words, i.e. the recipients name into them.

The DCC is an initiative that enables users to opt-in to mailing lists. Opt-in means that the user can decide whether he wants to receive emails from a certain source or not. Therefore each user can set up a white-list of bulk email sources he wants to receive. By using whitelists, each individual can fit the filter to his own opinion of which emails should be regarded as spam.

3 Survey of Existing Approaches

3.1 Naive Bayes Classifiers

The first application of naive Bayes filtering for Spam detection was described by two groups at the AAAI-98 conference [Gra03]. The first group, of Patrick Pantel and Dekang Lin, from the University of Manitoba in Canada described their technique in [PL98], the second team from Microsoft Research published their findings in [SDHH98]. Since then several projects used this approach and modified it in different ways. The most popular seems to be Paul Graham's project with his article "A Plan for Spam" [Gra02a] and follow-up papers [Gra03, Gra02d, Gra02c, Gra02b]. His ideas have been implemented in several spam filters.

This section starts with a discussion of Bayes theorem and its application for spam identification. It then summarizes the different Bayesian filtering concepts that have been implemented. Finally, a comparison of the approaches will be given.

3.1.1 Bayes Theorem and Spam Detection

The basic approach of detecting spam with the theorem of Bayes is to calculate the spam probability of an incoming email. The email is categorized according to this probability. The conditional probabilities that are regarded are $P(\text{spam}|a_1, \ldots, a_n)$ and $P(\text{legitimate}|a_1, \ldots, a_n)$. These are the probabilities that a given email that contains the attributes $a_1, \ldots, a_n$ is a spam or a legitimate email respectively. The way in which a categorization is derived from these values differs. The usual approach is to regard the ratio of both probabilities and categorize the email as spam if a certain value is exceeded.

$P(\text{spam}|a_1, \ldots, a_n)$ and $P(\text{legitimate}|a_1, \ldots, a_n)$ have to be calculated first. To do so, it is necessary to use the formula $P(A, B) = P(A|B) \cdot P(B)$ where A and B are two events. $P(A, B)$ is the joint probability, i.e. the probability of both events in conjunction. This formula is used in the following transformations:

$$\begin{aligned} P(\text{spam}|a_1, \ldots, a_n) &= \frac{P(\text{spam}, a_1, \ldots, a_n)}{P(a_1, \ldots, a_n)} \\ &= \frac{P(a_1, \ldots, a_n|\text{spam}) \cdot P(\text{spam})}{P(a_1, \ldots, a_n)} \end{aligned} \tag{3.1}$$

$$\begin{aligned} P(\text{legitimate}|a_1,\ldots,a_n) &= \frac{P(\text{legitimate},a_1,\ldots,a_n)}{P(a_1,\ldots,a_n)} \\ &= \frac{P(a_1,\ldots,a_n|\text{legitimate})\cdot P(\text{legitimate})}{P(a_1,\ldots,a_n)} \end{aligned} \quad (3.2)$$

If the email is categorized according to the ratio of these two probabilities, the division by $P(a_1,\ldots,a_n)$ can be skipped since it is a constant in both terms.

The Naive Bayesian approach assumes the different attributes to be independent. This leads to the following transformation:

$$P(a_1,\ldots,a_n|\text{spam})\cdot P(\text{spam}) = P(a_1|\text{spam})\cdots P(a_n|\text{spam})\cdot P(\text{spam})$$

$$P(a_1,\ldots,a_n|\text{legitimate})\cdot P(\text{legitimate}) = P(a_1|\text{legitimate})\cdots P(a_n|\text{legitimate})\cdot P(\text{legitimate})$$

Most spam detection approaches presented in this chapter use this formula. However, there are differences in how the input values are calculated and how a decision is derived from the probabilities. These technical aspects are explained in the following. It is also discussed how the authors justify the assumption that the attributes are statistically independent. If the terms used in the email are used as attributes, this is not very likely to be true. For example there might be word combinations like "special offer" that occur often. If a word combination tends to appear in spam emails, each of these words would increase the spam probability of a categorized email. The total effect would be a too big bias towards spam for emails containing this word combination. More realistic values would be achieved if the whole word combination was treated as one attribute. It is the approach of Naive Bayes just to assume this independence, but it is still interesting if the authors have additional explanations for this.

3.1.2 The "Classical" Approach

This subsection covers the early Bayesian filtering approach by Patrick Pantel and Dekang Lin [PL98], which is a foundation for subsequent implementations.

Attribute Selection

Pantel and Lin derive attributes by tokenizing the message. They define a token as "a consecutive sequence of letters or digits, or a consecutive sequence of non-space, non-letter and non-digit characters". The second type is limited to a maximum length of three characters. After extracting these tokens, the words are reduced to their canonical form by the Porter Stemming Algorithm [Por80]. Additional information from the email, like header text and HTML tags, are not regarded.

Deriving Spam Probabilities

This approach uses the Naive Bayesian formula shown in 3.1.1. The probabilities $P(a_1|\text{spam}), \ldots, P(a_n|\text{spam})$ and $P(a_1|\text{legitimate}), \ldots, P(a_n|\text{legitimate})$ still have to be calculated. Therefore Pantel and Lin count the number of occurrences of each attribute a_i in spam emails $n_S(a_i)$ and in legitimate emails $n_L(a_i)$. This information is stored together with the total number of attributes in spam emails n_S and in legitimate emails n_L in a so-called "frequency table".

These numbers are used as input for the m-estimate method, which is explained in [Mit97]. It would be problematic if the conditional probabilities were calculated with the formula $P(a_i|\text{spam}) = \frac{n_S(a_i)}{n_S}$ and a word would never occur in a spam email. This case would reduce the conditional probability to zero. If the spam probability is then calculated with the formula discussed above, the result would be a spam probability of zero independent of the other attributes of the email.

The m-estimate method eliminates this effect. It adds a prior estimate p for the probability multiplied with an "equivalent sample size" m to the numerator and the equivalent sample size to the denominator. This means that m virtual emails are added to the database. Each of these m emails include every word that has been observed with the estimated probability as the "number of occurrences". This number of occurrences must be greater than zero.

When using this method for the Naive Bayesian filter, Pantel and Lin define $m = 1$ and $p = \frac{1}{k}$, where k is the number of unique words in all messages. The reason for this is that the authors assume uniform priors by selecting this value. This technique is also recommended in [Mit97]. The resulting formulas for calculating the probabilities for each word are:

$$P(a_i|\text{spam}) = \frac{n_S(a_i) + \frac{1}{k}}{n_S + 1}$$

$$P(a_i|\text{legitimate}) = \frac{n_L(a_i) + \frac{1}{k}}{n_L + 1}$$

When calculating spam probabilities, Pantel and Lin did not regard those attributes with less than four overall occurrences and attributes that do not show a tendency towards occurring in legitimate or spam emails in the training data. These attributes are determined by the following formula:

$$0.45 < \frac{P(a_i|\text{spam})}{P(a_i|\text{spam}) + P(a_i|\text{legitimate})} < 0.55$$

If the calculated spam probability is greater than the legitimate probability, the email is assumed to be spam. Otherwise it is categorized as legitimate and not filtered.

Results

Pantel and Lin used a relatively small training corpus, which consists of only 160 spam and 460 legitimate messages. The testing corpus consists of 277 spam and 346 legitimate emails. However, the testing spam emails were taken from a database on the Internet, while the training spam emails were taken from one of the authors mailbox.

The false-positive rate they achieved is 0.58% while the false-negative rate is 13.36% when they did not leave out the messages with less than 4 occurences and those that do not show a tendency in the training data. When applying these filters, they achieved a false-positive rate of 1.16% and a false-negative rate of 8.30%. These numbers are remarkable because there is a significant difference between the false-positive rate and the false-negative rate although the authors did not document a difference in the handling of both types. According to their description, they did not tend to categorize more emails as legitimate in order to avoid false-positives. The explanation for this behavior of the filter might be that the spam emails used for training and testing are from different sources.

When calculating the weighted error for the values with and without applying the filters, the version without the filter outperforms the other one for all values of λ. This is why table 3.1 shows only these figures.

λ	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{r}_{\mathrm{fp}}$	$\mathbf{r}_{\mathrm{fn}}$	$\mathbf{W}_{\mathrm{Err}}$
9	2	37	0.578%	13.357%	1.622%
99	2	37	0.578%	13.357%	0.681%
999	2	37	0.578%	13.357%	0.588%

Table 3.1: Results of the approach by Pantel and Lin

Discussion

As in all Naive Bayesian approaches, Pantel and Lin assume their attributes to be statistically independent but they do not give a further explanation for this assumption.

Another disadvantage of this approach is that there is no built-in security against false-positives. Advanced implementations use different tricks to bias the filter in order to tend to classify emails as legitimate if the output of the calculation is not significant. Nonetheless, the false-positive rate of the filter was much lower than the false-negative rate for the testing corpus. It is not clear if the filter would behave in this way when used in real communication.

3.1.3 Microsoft Research 1998

As mentioned before, the technique explained in [SDHH98] was presented at the same conference in 1998 as the paper described above. There are several technical analogies in these two approaches. However, this section concentrates on the differences, especially the introduction of so-called "hand-crafted" attributes.

Attribute Selection

The attributes used for this filter are also words used in the message body. The text does not clearly specify which characters are regarded as word separators or if some characters are ignored. All attributes that occur less than three times in the training corpus are not regarded.

The authors additionally defined so-called "hand-crafted, domain-specific features". They argue that there are many particular attributes of spam emails besides the words used in the message body. These problem-specific attributes can be divided into two groups: Phrasal and non-textual attributes.

The first group examines the message text for the appearance of particular phrases, for example "be over 21". The authors defined 35 such attributes. The second type of hand-crafted attributes are non-textual. An example mentioned in the article is the domain type of the sender. The reason for this is that for example .edu domains are rarely used to send spam emails. 20 non-textual attributes were included in the filter.

The resulting number of attributes that are derived from the message body is, according to the authors, very large. Therefore they want to use only those attributes that represent the best basis to make a decision. According to the authors, such dimensionality reduction helps to control the model variance due to estimating parameters. Another positive aspect is that the degree to which the independence assumption is violated can be reduced.

The value that is calculated to find the most significant attributes is the mutual information (MI). The mutual information measures how much information one variable X tells us about another one, Y. As mentioned above, the joint probability can be calculated as $P(X,Y) = P(X) \cdot P(Y)$ if X and Y are independent. This means that the fraction of both sides of this equation equals one for independent variables. The mutual information uses this fact. The logarithm of the fraction is calculated, weighed with the joint probability and summed up over all possible values for the variables. In this case, one variable $O(a_i)$ is one, if the attribute a_i occurs in an email and zero otherwise. The other variable states if the email is spam or not:

$$MI(a_i) = \sum_{o \in \{0;1\}, c \in \{\text{spam;legitimate}\}} P(O(a_i) = o, c) \log \frac{P(O(a_i) = o, c)}{P(O(a_i) = o) \cdot P(c)}$$

A greater value of $MI(a_i)$ means a higher entropy. Hence, the attributes with the greatest mutual information values are selected, in this case the 500 highest.

Deriving Spam Probabilities

The authors of this document also use the Bayes formula given in 3.1.1. It is not explained how they derive the probabilities for $P(a_1|\text{spam}), \ldots, P(a_n|\text{spam})$ and for $P(a_1|\text{legitimate}), \ldots, P(a_n|\text{legitimate})$ that are used in the formula.

When referring to the procedure for selecting whether an email is classified as spam or not, it is only mentioned "[...] a message is only classified as junk if the probability that it would be placed in the junk class is greater than 99.9%". This does not make clear whether both probabilities, $P(\text{spam}|a_1, \ldots, a_i)$ and $P(\text{legitimate}|a_1, \ldots, a_i)$, or only the spam probability is used.

Results

The used corpus contains 1789 email messages of which 1578 were classified as spam and 211 as legitimate. This corpus is split into a training corpus of 1538 messages and a testing corpus of 251 messages. The article does not point out how big the share of spam and legitimate messages in these two groups is. This is why it is not possible to calculate the weighted error rates and we confine ourselves to take a look at the false-positive and false-negative rates.

The results are measured in three steps: Firstly only attributes derived from the message text are used, then phrasal attributes are added and finally non-textual attributes are added. The results are summarized in table 3.2. It is particularly remarkable that the introduction of phrases only had a small effect on the rates while the non-textual information led to a notable improvement.

attributes	$\mathbf{r}_{\text{fp}}$	$\mathbf{r}_{\text{fn}}$
words	6.6%	5.7%
words and phrases	5.3%	5.7%
words, phrases and non-textual	0.0%	1.7%

Table 3.2: Results of the approach by Microsoft Research 1998, first corpus

The email corpus used for this first testing series consisted of existing email folders. The authors admit that the users from which these collections were gathered had already deleted several emails. To test the filter with a user's entire email stream, a second testing series was carried out. The testing data used in this series are all 45 spam and 177 legitimate emails sent to one user during one week. The training was done with 2593 emails from the previous year. The results achieved using words, phrases and non-textual attributes are shown in table 3.3.

λ	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{r}_{\text{fp}}$	$\mathbf{r}_{\text{fn}}$	$\mathbf{W}_{\text{Err}}$
9	3	9	1.695%	20%	2.198%
99	3	9	1.695%	20%	1.742%
999	3	9	1.695%	20%	1.700%

Table 3.3: Results of the approach by Microsoft Research 1998, second corpus

Discussion

This text does not explain how a decision is derived as to whether an email is spam or legitimate. The results of the study are good, although the reliability is doubtful since the number of emails used for training and testing was very small. The share of spam emails and legitimate emails is also not very close to reality, as the corpus consists of over five times more spam emails than legitimate emails.

A question that arises from this study is whether the improvement of the results with use of non-textual attributes can also be achieved by including the header in the generation of textual attributes. This may be the case since the bigger part of these non-textual attributes is data from the email header like the sender-domain.

3.1.4 Androutsopoulos

The approach of Ion Androutsopoulos' research group is given in [AKC+00]. It is basically a reproduction of the technique by Microsoft Research explained in 3.1.3, with a concentration on finding those values for parameters in this approach that lead to optimal filtering results. Furthermore an evaluation system for measuring the performance of spam filters is introduced. The results of this study were used in other experiments to compare Naive Bayesian spam filtering with other machine learning techniques [AKCS00, APK+00].

Attribute Selection

The selected attributes were derived from the words of the message. Attributes including other information like header-contents were not used.

These attributes were modified in several ways in order to find out which of these modifications improve the filtering results and which values of variables are the most practical. One of these modifications was ignoring the 100 most frequent English words that were added to a stop-word list. Another modification was a lemmatizer that converts words to a base form that is comparable to the stemming algorithm mentioned in 3.1.2. The implementation of this lemmatizer is not further explained. The last modification of the attribute data was the reduction to those attributes with the highest mutual information as

described in 3.1.3. The number of attributes used for filtering was 50 to 700, increasing in steps of 50.

Deriving Spam Probabilities

The decision of whether to classify an email as spam or not is based on the equations 3.1 and 3.2. In this approach the denominator is not removed, it is transformed as follows:

$$\begin{aligned} P(a_1, \ldots, a_n) &= \sum_{c \in \{\text{spam,legitimate}\}} P(c) \cdot P(a_1, \ldots, a_n | c) \\ &= \sum_{c \in \{\text{spam,legitimate}\}} P(c) \cdot P(a_1|c) \cdots P(a_n|c) \end{aligned}$$

Using this transformation, one can calculate the fraction of the probabilities $P(\text{spam}|a_1, \ldots, a_n)$ and $P(\text{legitimate}|a_1, \ldots, a_n)$. This value indicates the ratio of the probability that the categorized email is spam and that it is legitimate. In this approach, the authors estimate the proportion of the costs for categorizing a legitimate email as spam and vice versa. If the ratio of the two probabilities mentioned above is larger than this cost-ratio λ, the email is handled as spam:

$$\frac{P(\text{spam}|a_1, \ldots, a_n)}{P(\text{legitimate}|a_1, \ldots, a_n)} > \lambda \tag{3.3}$$

Androutsopoulos et al. continue to transform this inequation. They show that the numerator equals one minus the denominator. This can be shown if one uses the formulas known from 3.1.1. In the transformations below, $P(a_1|\text{legitimate}) \cdots P(a_n|\text{legitimate}) \cdot P(\text{legitimate})$ is substituted with X and $P(a_1|\text{spam}) \cdots P(a_n|\text{spam}) \cdot P(\text{spam})$ with Y for the sake of clarity:

$$\begin{aligned} P(\text{legitimate}|a_1, \ldots, a_n) &= \frac{P(a_1|\text{legitimate}) \cdots P(a_n|\text{legitimate}) \cdot P(\text{legitimate})}{\sum_{c \in \{\text{spam,legitimate}\}} P(c) \cdot P(a_1|c) \cdots P(a_n|c)} \\ &= \frac{X}{X+Y} \\ &= \frac{X}{X+Y} + \frac{Y-Y}{X+Y} \\ &= \frac{X+Y}{X+Y} - \frac{Y}{X+Y} \\ &= 1 - \frac{Y}{X+Y} \end{aligned}$$

$$
\begin{aligned}
&= 1 - \frac{P(a_1|\text{spam}) \cdots P(a_n|\text{spam}) \cdot P(\text{spam})}{\sum_{c \in \{\text{spam,legitimate}\}} P(c) \cdot P(a_1|c) \cdots P(a_n|c)} \\
&= 1 - P(\text{spam}|a_1, \ldots, a_n)
\end{aligned}
$$

We can use this to rewrite the categorization formula 3.3 as follows:

$$P(\text{spam}|a_1, \ldots, a_n) > t, \text{ with } t = \frac{\lambda}{1+\lambda}, \lambda = \frac{t}{1-t}$$

The values that were assigned to t for testing purposes were 0.999, 0.9 and 0.5. The first value, which corresponds to a value of 999 for λ (i.e. misclassifying a legitimate email as spam is 999 times as costly as misclassifying a spam email as legitimate) was an experiment for an email filter that would immediately delete emails that are categorized as spam. The second and third value which correspond to $\lambda = 9$, and, respectively, $\lambda = 1$ were used for testing the case of a filter which would reject emails that are categorized as spam but which would respond to the author of the email and ask him to resend the email to a different email address.

Results

The testing was made with a corpus called the "Ling-Spam corpus". It consists of 2412 legitimate messages from the Linguist list, a "moderated (hence, spam-free) list about the profession and science of linguistics" [AKC+00] and 481 spam messages received by one of the authors of the study. Ten-fold cross-validation was used for testing.

The testing results are shown in table 3.4. The amount of data collected by this group is quite big because the available parameters were tested with several values as mentioned above. Depending on the value of λ, the percentage of identified spam was between 82.78% for $\lambda = 1$ and using the lemmatizer and stop-list, and 63.05% for the same setting with $\lambda = 999$. Spam precision is the ratio of the number of spam emails classified as spam and the total number of messages classified as spam. Spam recall is the share of spam emails that were correctly classified. This means that it can be calculated as one minus the false-negative rate.

It is possible to calculate the weighted errors for $\lambda \in \{9; 99; 999\}$ from this data. The best values for each group are shown in table 3.5. The spam and legitimate numbers are not integers because they are average numbers from the ten-fold cross validation. The best values for $\lambda = 9$ were those achieved by the combination of the lemmatizer and the stop-list with an internal λ of 1. For $\lambda = 99$ and $\lambda = 999$ the filter using only the lemmatizer with an internal λ of 999 was superior to the other configurations.

Filter configuration	λ	n	spam-prec.	spam-rec.
bare	1	50	96.85%	81.10%
stop-list	1	50	97.13%	82.35%
lemmatizer	1	100	99.02%	82.35%
lemmatizer & stop-list	1	100	99.49%	82.78%
bare	9	200	99.46%	76.94%
stop-list	9	200	99.47%	76.11%
lemmatizer	9	100	99.45%	77.57%
lemmatizer & stop-list	9	100	99.47%	78.41%
bare	999	200	99.43%	73.82%
stop-list	999	200	99.43%	73.40%
lemmatizer	999	300	100.00%	63.67%
lemmatizer & stop-list	999	300	100.00%	63.05%

Table 3.4: Results of the approach by Androutsopoulos et al. 2000

λ	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{r}_{\mathrm{fp}}$	$\mathbf{r}_{\mathrm{fn}}$	$\mathbf{W}_{\mathrm{Err}}$
9	0.2	8.3	0.085%	17,220%	0.456%
99	0	17.5	0%	36,331%	0.073%
999	0	17.5	0%	36,331%	0.007%

Table 3.5: Results of the approach by Androutsopoulos et al.

Discussion

One of the most obvious deficiencies in this approach is that no additional information like character strings or the domain type of the sender is used, although it is based on the technique explained in 3.1.3 [SDHH98]. This paper used additional information and verified that it has a good impact on filtering results. Androutsopoulos et al. do not explain why they left out these additional attributes.

The testing results from this study are at first sight not very promising. On the one hand, the false-positive rates remain quite low. But the false-negative rates are never less than 17.22%. However, it must be taken into account that these ratios might be improved by using the additional information mentioned in the previous paragraph.

3.1.5 Paul Graham's Plan for Spam

Paul Graham describes an often implemented approach in [Gra02a].

Attribute Selection

The entire email is tokenized; the algorithm does not treat HTML, Javascript or headers differently from the message body. All alphanumeric characters, dashes, apostrophes and dollar signs are token characters while all other characters are token separators. Tokens that only consist of digits or HTML comments are not used. Each token is considered to be an attribute.

Deriving Spam Probabilities

In this approach the probability calculation is implemented differently from the approach described in 3.1.1. However, statistical independence of the attributes is still assumed; the approach is a Naive Bayes algorithm. The difference to the regular calculations is that Paul Graham first calculates the probability $P(\text{spam}|a_i)$ for each attribute a_i. These values are then combined to form one probability $P(\text{spam})$.

Paul Graham's approach uses three tables to store different numbers. The first one contains the number of occurrences in spam emails for each attribute and the second the occurrences in legitimate emails. The third table contains the probability for each attribute that an email containing it is spam. Two counters N_S and N_L count the overall number of spam and legitimate emails.

Graham's approach is based on the theorem of Bayes which states:

$$P(B_k|A) = \frac{P(A|B_k)}{\sum_{i=1}^{n} P(A|B_i) \cdot P(B_i)}$$

In this case, we want to calculate $P(\text{spam}|a_i)$. Graham makes the assumption that $P(a_i|\text{spam}) = min(1; \frac{n_S(a_i)}{N_S})$ where N_S is the number of spam emails and $n_S(a_i)$ is the number of occurrences of attribute a_i in spam emails. This means that Paul Graham calculates the probabilities with the simple formula for conditional probabilities. But he uses the number of occurrences of the attribute in spam or legitimate emails as numerator and the number of spam or legitimate emails as denominator. The problem of this approach is that the numerator can become bigger than the denominator and thereby the probability can become greater than one. Therefore Paul Graham takes the minimum of this term and one as the conditional probability. His explanation as to why he uses this unorthodox calculation is that this "adds another slight bias to protect against false-positives" [Gra02a]. The calculation of $P(a_i|\text{legitimate})$ includes another modification to bias the filter even more. The number of occurrences of each attribute in legitimate emails is doubled.

The probability that an email containing the attribute a_i is spam can be calculated using the theorem of Bayes. Paul Graham assumes that $P(\text{spam}) = P(\text{legitimate})$ which allows the following transformations:

$$
\begin{aligned}
P(\text{spam}|a_i) &= \frac{P(a_i|\text{spam}) \cdot P(\text{spam})}{P(a_i|\text{spam}) \cdot P(\text{spam}) + P(a_i|\text{legitimate}) \cdot P(\text{legitimate})} \\
&= \frac{P(a_i|\text{spam})}{P(a_i|\text{spam}) + P(a_i|\text{legitimate})} \\
&= \frac{min(1; \frac{n_S(a_i)}{N_S})}{min(1; \frac{2 \cdot n_L(a_i)}{N_L}) + min(1; \frac{n_S(a_i)}{N_S})}
\end{aligned}
$$

The last term is the one Graham uses to calculate the probability that an email containing attribute a_i is spam. This probability is calculated for each attribute whenever a new email is added to the data basis. In order to avoid the values zero and one, which would make the following calculations impossible, Graham replaces all values below 0.01 with 0.01 and all values greater than 0.99 with 0.99. These values are stored in the third table.

When a new email arrives, the first step is to find from the set of all attributes occurring in it those that are most significant. The measurement for this is the spam probability of these attributes that has to be as far away from 0.5 as possible (where words that are new to the classifier get a probability of 0.4). In this way the 15 most interesting attributes are selected, they form the set J. The spam probabilities of these 15 attributes are then combined to one probability that is used to classify the whole email. The formula used is:

$$
P(\text{spam}) = \frac{\prod_{j \in J} P(\text{spam}|a_j)}{\prod_{j \in J} P(\text{spam}|a_j) + \prod_{j \in J} (1 - P(\text{spam}|a_j))}
$$

If the calculated spam probability is greater than 0.9, the email is classified as spam. Otherwise it is considered not to be spam.

Results

Paul Graham uses spam and legitimate email corpora of about 4000 messages each for training his classifier. The results in applying the filter are described by "we now miss less than 5 per 1000 spam emails, with 0 false positives" [Gra02a]. This would mean that the false-positive rate is 0.5% and the false-negative rate is zero for this testing corpus. In a later publication [Gra03] Graham mentions that his testing corpus for legitimate email consisted of about 4000 emails and that thereby, if the next arriving email would be falsely considered to be spam, the false-negative rate would be 0.03%. In this paper he also states that four of 1750 spam emails were not recognized by his filter. This leads to the weighted error rates shown in table 3.6.

λ	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{r}_{\text{fp}}$	$\mathbf{r}_{\text{fn}}$	$\mathbf{W}_{\text{Err}}$
9	0	4	0%	0.229%	0.011%
99	0	4	0%	0.229%	0.001%
999	0	4	0%	0.229%	0.0001%

Table 3.6: Results of Paul Graham's spam filter

Discussion

This approach has several parts that are not explained or only have the explanation "add a bias to protect against false positives" or "by trial and error I chose [...]". For example the use of the word counter as numerator and the email counter as the denominator for calculating the probabilities $P(a_i|\text{spam})$ is justified by the statement that this helps to avoid false-positives. There is no further explanation for this.

Another detail that would need further explanation is that Paul Graham assumes $P(\text{spam}) = P(\text{legitimate})$. This means that the initial probability that an incoming email is spam is 0.5. However, this probability should be determined by the ratio of incoming spam emails and can be different from 0.5.

The results achieved by this filter are remarkably good. But the training corpus Paul Graham used was huge, too. It would be interesting to see how this filter would behave with smaller amounts of training emails. Maybe this would bring the filter's performance closer to the other ones.

3.2 Genetic Algorithms

David Goldberg describes Genetic Algorithms as search algorithms based on the mechanics of natural selection and natural genetics [Gol89]. They have been developed by researchers at the University of Michigan. Genetic Algorithms combine the concept of survival of the fittest among string structures with a structured, randomized information exchange. "In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure" [Gol89].

The first part of this chapter explains the concept of Genetic Algorithms according to [Mit97]. Then a study comparing Genetic Algorithms with Naive Bayesian Filters [Kat99] is described.

3.2.1 Theory of Genetic Algorithms

The search space used in Genetic Algorithms is coded as a finite length string of variables in form of an alphabet. The most common approach is to use the binary alphabet given by the set 0;1. Each string represents a possible solution

for the problem that should be solved. In our case, these solutions represent a function that assigns either the value spam or legitimate to an email depending on the attributes of this email.

The starting point of an evolutionary process by Genetic Algorithms is a random starting population, with a population being a set of solutions. Each of these solutions $s_i \in \mathcal{P}$ are evaluated by a fitness function f. This fitness function measures how well the solution can solve its task, in our case the categorization of emails. The implementation of the fitness function can differ.

When the fitness function is calculated for each member of a population, a new generation is formed. This means that a new population is generated from the old one. Three different techniques are used to specify the solutions in the new population: Selection of members of the old population that continue to exist in the new one; a crossover that combines two solutions from the old population to two new ones; and mutation which changes some random aspect of a certain fraction of the new population. This process is executed multiple times until at least one of the solution's fitness function values exceeds a threshold that has been pre-determined. The solution which has the best value is used to handle the task.

Selection

Some solutions from the old population are selected to "survive", meaning that they keep existing in the next generation. The ratio of these solutions to the size of the population $\mathcal{P}$ is given by the variable r. The solutions that are carried over into the new population are selected by random. The probability for selecting solution $s_i \in \mathcal{P}$ is given by the following formula:

$$P(s_i) = \frac{f(s_i)}{\sum\limits_{s_j \in \mathcal{P}} f(s_j)}$$

This formula implies that solutions with a higher fitness value are more likely to be selected. David Goldberg compares this selection operator to a weighted roulette wheel [Gol89]. Figure 3.1 shows such a wheel for an example population containing four solutions A, B, C and D. The greater a solution's fitness value is, the bigger is the corresponding area on the wheel. In the example, the fitness value of solution A equals 9.4% of the sum of all fitness values, solution B's 15.6% and so on. This means that the probability that the wheel will select solution A to survive is 9.4%.

The thought of the concept of selection is an analogy of Darwin's evolutionary concept "multiply, vary, let the strongest live and the weakest die" [Dar59]. The aspect of variation will be added by crossover and mutations which will be explained in the following.

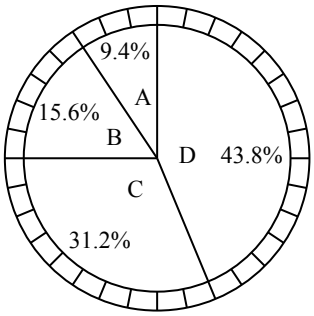

Figure 3.1: The weighted roulette wheel illustrating the selection process

Crossover

A crossover creates two new solutions by mixing attributes of two old ones. These can be selected by several methods, the basic approach is to use the probabilities $P(s_i)$ mentioned above. From each of the selected solutions, two parts are selected which are exchanged to form the two members of the next generation. In Mitchell's model, which uses bit-strings of equal length to represent solutions, each string is cut equally. The halves are then recombined in both probable ways and these new solutions are added to the new population.

Goldberg's simple crossover selected an integer position k uniformly at random between one and the string length l less one. Two new strings are created by swapping all characters between positions $k + 1$ and l inclusively [Gol89].

Mutation

Mutation is an important technique used in Genetic Algorithms because it is a way in which new patterns of behavior can be obtained. After the solutions of the new population have been generated by selection and crossover, a randomly selected fraction m of them is manipulated by mutation. In the bit-string coding of solutions described by Mitchell, one bit is selected randomly and inverted.

3.2.2 Study by Hooman Katirai

In his paper [Kat99], Hooman Katirai briefly describes the implementation of a Genetic Algorithm for spam filtering. He compares his results to those achieved by a Bayesian filter.

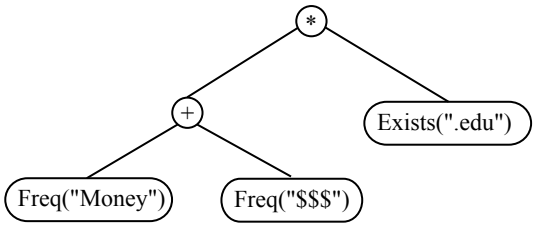

Figure 3.2: Example of a tree representing an element of the search space

Attribute Selection

The attributes used in this study are the words used in the emails. The author does not specify how a word is defined and which characters are used as separators. However, he describes that the algorithm removes all HTML tags first. This shows that the HTML tags are not used as attributes. The software also filters the 60 most common words in the English language which are considered to occur too often to be of help. Then a stemming algorithm is applied to the extracted words. The author used a variant of the Porter Stemming Algorithm [Por80] described by Frakes [Fra92] which he considers to be quite simple and fast.

Deriving a Decision

Katirai explains that a combination of word operators and numerical operators are used to build the search space. Word operators represent the frequency or the existence of a certain word in an email. Numerical operators are shown in table 3.7. The search space consists of trees that represent word operators and numbers connected by numerical operators. An example of such a tree is shown in figure 3.2. This tree sums up the frequencies of the strings “Money” and “$$$” in an email and multiplies it with one if the string “.edu” occurs or zero if it does not occur.

Type	**Symbols**
Arithmetic	$+, -, /, *$
Relational	$=, <>, >=, <$
Logical	AND, OR, NOT
Non-linear	Min, Max, ABS
Square-root	$\surd$

Table 3.7: Numerical operators

The fitness function used in this test is based on the squared errors. The error is the difference of the variable C_j indicating if email j is spam ($C_j = 1$) or not ($C_j = 0$) and the value $v_{j,i}$ calculated for this email by solution i. These squared errors are summed up for spam emails and legitimate emails

separately. Each of the sums is weighted with the reciprocal of the number of emails in the corresponding category, to calculate the average error for each category. These average errors are added up:

$$f(s_i) = \frac{1}{N_S} \cdot \sum_{j=1}^{N} C_j \cdot (v_{j,i} - C_j)^2 + \frac{1}{N_L} \cdot \sum_{j=1}^{N} (1 - C_j) \cdot (v_{j,i} - C_j)^2$$

The reason as to why this procedure is used is that the author had experienced that a simple sum of all squared errors leads to a bias in the classifier towards the email type that is more frequent in the training corpus. If for example most of the corpus consists of spam, solutions that tend to filter a high degree of emails will get a high fitness value although they are not very good at recognizing to which group an email belongs.

The ratio r of solutions that are selected to be taken over into the new population is 0 in this approach. All members of the new population are generated by crossover. For the crossover operation, one solution is selected by random and ten other solutions have to compete to be the other parent. Then one subtree is selected from each of the two solutions which are exchanged in order to form the two members of the new population.

The author explains that single-node mutation (which replaces only one node of the tree) and subtree mutation (which replaces a whole subtree) are variants of mutation that can be applied to the tree-representation. However, it is not mentioned whether only one or both concepts are used.

Results

The corpus used for testing in this approach consisted of 701 spam and 102 legitimate emails which were obtained from a single person's mailbox. This corpus was split into one used for training (671 spam and 72 legitimate emails) and one used for testing (30 spam and 30 legitimate emails).

The author made several runs of his algorithm and since the populations are randomly generated, the filter achieved different results. The best instance had a false-positive rate of 3.33% and a false-negative rate of 30%. This leads to the weighted error rates shown in 3.8

λ	$\mathbf{N_{L \to S}}$	$\mathbf{N_{S \to L}}$	$\mathbf{r}_{\text{fp}}$	$\mathbf{r}_{\text{fn}}$	$\mathbf{W}_{\text{Err}}$
9	1	9	3.33%	30%	6.0%
99	1	9	3.33%	30%	3.6%
999	1	9	3.33%	30%	3.36%

Table 3.8: Results of Hooman Katirai

Discussion

The results achieved by this approach are not very promising. Katirai concludes that it was only slightly outperformed by a Bayesian filter he implemented. But if the results are compared to the Bayesian approach by Paul Graham [Gra02a] explained in 3.1.5, they are vastly outperformed.

It is questionable how reliable the results are. The number of emails used in the testing corpus is low compared to other studies. Therefore it is doubtful that differences found in the testing results are statistically significant.

There are several points that can be improved. Many attributes are not regarded. Not all header information is used and HTML tags are removed. Another improvement might be achieved by using selection. If only crossover is used, promising solutions might be lost because they are changed too much by this operation. If they had the chance to keep existing and change only slightly by a mutation, better filters might result.

3.3 Artificial Neural Networks

Artificial neural networks (ANNs) can be used for classification. They are a method that is according to [Mit97] appropriate for problems in which instances are represented by many attribute-value pairs. The attributes derived from email are usually numerous, if the occurrence of words in the message body is defined as attributes. Therefore ANNs appear to be a suitable technique for email classification. Mitchell also mentions that ANNs usually need a long training time. This can be acceptable once, but if the network should be updated after each incoming email, such a lengthy time would not be desirable. Therefore the network should be updated only once in a while.

The first part of this section describes the general system of ANNs based on [Mit97]. If a more precise description is needed, it is provided for example by [Bis95]. The second part summarizes a study which tested the use of ANNs for filtering spam emails.

3.3.1 Theory of Artificial Neural Networks

Structure

ANNs are inspired by the structure and functionality observed in biological studies of the brain. They imitate the system of simple units that are densely connected. These units are called perceptrons. An ANN is a graph consisting of interconnected perceptrons that can be either acyclic or cyclic.

A perceptron is an object that generates an output depending on a vector of inputs $x_1, \ldots, x_n$. In mail categorization this could be one input per attribute,

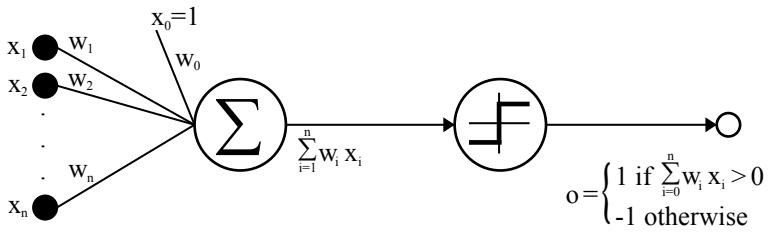

Figure 3.3: A perceptron

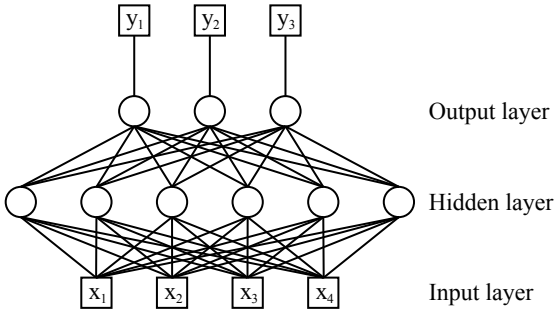

Figure 3.4: A multilayer artificial neural network

representing the number of occurrences within the email that is categorized. The perceptron weighs the inputs with a set of weights $w_0, \ldots, w_n$ and adds them. This sum is used as input for a function called the activation function. The result of this function is the output of the perceptron.

A common kind activation function is the threshold function. The output is either one if the sum of the weighed inputs exceeds a certain threshold or minus one otherwise. Figure 3.3 illustrates such a perceptron according to [Mit97]. The output $o(x_1, \ldots, x_n)$ is derived according to the following function:

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

The real valued weights $w_0, \ldots, w_n$ are determined during the training phase. The value returned by the function can be either used to derive a decision or as input for other perceptrons. When several perceptrons are combined, they can form a multilayer network as shown in figure 3.4.

Training

The ANN has to be trained using a training corpus. Mitchell explains two different training rules for ANNs, the perceptron training rule and the delta rule.

The perceptron training rule begins with random weights $w_0, \ldots, w_n$. These

weights are used to classify the elements from the training corpus one after another. After each iteration, the weights are adjusted if the output was wrong. The new value of weight w_i is calculated according to the following formula:

$$w_i = w_i^{old} + \Delta w_i$$
$$\text{with } \Delta w_i = \eta \cdot (t - o) \cdot x_i$$

The variables o and t are the output and the desired output of the perceptron. η is the learning rate, a parameter that determines how big the modification of the weight is. According to [Mit97] a typical value for it is 0.1.

The second formula calculates the difference of the old and the new weight. The difference $(t - o)$ can be either minus two or two if the output of the perceptron was wrong or zero if the output was right. If it is zero, the weights can remain. If it is two, the output was minus one while the desired value is one. This means that the products $w_i \cdot x_i$ have to be increased. If x_i is greater than zero, the weight has to be increased, if it is less than zero, the weight has to be decreased. The result is that the product $(t-o) \cdot x_i$ results in a shift into the right direction. η makes the change smaller in order to avoid a situation in which the algorithm overshoots the mark.

If the training examples are linearly separable, the perceptron rule always leads to a set of weights $\{w_0, \ldots, w_n\}$ that classifies each example correctly. If this is not the case, the delta rule, that uses the gradient descent to converge towards a best-fit approximation. It will not be explained in detail, see [Mit97] for a comprehensive coverage.

3.3.2 Survey by Rich Drewes

The study explained in [Dre02] describes an implementation of a neural network spam filter.

Attribute Selection

The author uses the words of the message body as attributes. A word is defined as a sequence of consecutive alpha characters less than 15 characters in length. The words are not treated as case-sensitive, all characters are transformed to lower case. These attributes were reduced to the m most frequent ones.

Implementation

The values $x_1, \ldots, x_m$ are calculated as the number of occurrences of an attribute in the email divided by the total number of occurring attributes. The

input vector is thus simply a representation of the presence or absence of the attributes weighed by the length of the message itself.

The text states that the network used is a three-layer network that is fully connected. This means that each node is connected to all nodes in the following and previous layer. The training was done with a backpropagation algorithm, which was used a certain number of training epochs. The author does not mention any further details about the training algorithm used.

Results

The author used a testing corpus of 1592 legitimate and 1730 spam emails. It was split into three parts, a training set of 2043 instances, a validation set of 631 instances and a test set of 649 instances. The first tests lead to the insight that the best size for the wordlist is $m = 2000$. The number of nodes in the middle layer of the ANN was also varied. The best value for this is four.

The author does not separate the two error types in the figures he gives for each part of the corpus. It is only mentioned that for all data sets the number of legitimate emails misclassified as spam was 17 while the number of spam emails not filtered was only 9. Table 3.9 was calculated on the basis of these numbers. However, it has to be taken into account that these numbers are clearly better than they would be if only the validation set and the test set were used, because the filter adopted to the instances from the training set during the training phase.

λ	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{r}_{\text{fp}}$	$\mathbf{r}_{\text{fn}}$	$\mathbf{W}_{\text{Err}}$
9	17	9	1.068%	0.520%	1.009%
99	17	9	1.068%	0.520%	1.062%
999	17	9	1.068%	0.520%	1.067%

Table 3.9: Results of the approach by Rich Drewes

Discussion

The achieved weighted error rates are rather disappointing. This effect is even strengthened if one realizes that the results have been improved by using among others the training set to calculate them. But the problem of this approach is that the results were not biased towards avoiding falsely filtered emails. The number of false-positives is nearly twice the amount of false-negatives. If such a feature could be added to the classifier, the results are likely to improve significantly.

A general disadvantage of using ANNs for spam filtering is that they need a long training time [Mit97]. This makes them less suitable for spam filters. The problem is the adjustment process if new emails arrive. If they should be taken

into account, new training runs have to be performed. Other approaches like those based on the Bayes theorem or memory based approaches discussed in 3.4 can use new emails for filtering by just adding them to a database.

3.4 Memory Based Learning

Memory based learning is also known as instance based learning. Its idea is to keep all instances that occurred in memory. A new one is categorized by measuring the similarity or the distance between it and all memorized instances. The classification is usually implemented as a variant of the k-nearest-neighbor algorithm.

3.4.1 Theory of Memory Based Learning

To determine the similarity of two instances, their distance is measured. The smaller their distance, the more similar they are. It is usually defined as the Euclidean distance in a multi-dimensional space. The dimensionality is determined by the number of attributes that are extracted. If the attributes are symbolic, such as if a certain word occurs in an email, the distance can be measured by the overlap.

It is assumed that the attributes $\mathcal{A} = \{a_1, \ldots, a_n\}$ are extracted from all emails. The vector $\vec{x_i} = \langle x_{i,a_1}, x_{i,a_2}, \ldots, x_{i,a_n} \rangle$ represents the extraction results for email i. If the (symbolic) attribute a_1 occurs in that email, x_{i,a_1} is one, otherwise it is zero. The overlap distance of two instances i and j is calculated as follows:

$$d(\vec{x_i}, \vec{x_j}) = \sum_{k=1}^{n} \delta(x_{i,a_k}, x_{j,a_k})$$

$$\text{with } \delta(x_{i,a_k}, x_{j,a_k}) = \begin{cases} 0 & \text{if } x_{i,a_k} = x_{j,a_k} \\ 1 & \text{otherwise} \end{cases}$$

The k-nearest-neighbor algorithm searches the k instances that are most similar to the one that is to be categorized, i.e. those for which the overlap distance is minimal. The class of the majority of these k neighbors is assigned to the new instance.

3.4.2 Survey by Sakkis et al.

This team published in [APK+00] the first results of their work on a memory based spam filter. In 2001 they released an article [SAP+01] that was much

more detailed and examined extensions of the k-nearest-neighbor algorithm and the impacts of changing certain parameters.

Attribute Selection

The attributes in this study are derived by splitting the email into words and lemmatizing them, this means all words are converted to their base form. The text neither mentions which characters are used as word separators nor if any header information is used. From these attributes the most valuable are selected. All words occurring in less than four messages are discarded and from those left over, the m attributes with the highest information gain are selected. The information gain was already defined in 3.1.3 but there it was called the mutual information. m is varied between 50 and 700.

Deriving a Decision

In this approach the k-nearest-neighbor algorithm is modified. First of all the algorithm does not take the k closest instances into account but those with the k smallest distances. This means that if several instances have the same distance, more than k of them will be used for the decision.

The email is not simply classified according to the majority of these neighbors. The authors calculate the probability $P(\text{spam}|a_1, \ldots, a_n)$ that an email is spam as the percentage of training instances in the k-neighborhood that belong to that category. The probability that it is a legitimate email can be calculated analogous. If the ratio of the spam probability and the legitimate probability exceed the confidence level λ, the email is categorized as spam.

Another modification of the standard distance measure algorithm the authors implemented in this study is attribute weighting. Therefore the regular equation was changed to the following:

$$d(\vec{x_i}, \vec{x_j}) = \sum_{l=1}^{n} w_l \cdot \delta(x_{i,a_l}, x_{j,a_l})$$

The coefficient w_l is called the weight assigned to attribute l. This weight can be determined in different ways. One possibility is using the information gain of an attribute. The information gain is defined differently from the variable used for attribute selection. To calculate it, the authors first introduce the entropy. The entropy $H(C)$ is the entropy of the category-denoting variable. It measures the uncertainty on the category of a randomly selected instance and is calculated as:

$$H(C) = - \sum_{c \in \{\text{spam;legitimate}\}} P(c) \cdot \log_2(P(c))$$

The uncertainty on the category for an instance with the value $o(a_l)$ for variable a_l is $H(C|O(a_l))$. $O(a_l)$ is a variable that equals one if the instance contains attribute a_l and zero otherwise. $H(C|O(a_l))$ is defined similar to $H(C)$:

$$H(C|O(a_l)) = -\sum_{c\in\{\text{spam;legitimate}\}} P(c)\cdot\log_2(P(c|O(a_l)))$$

Using these entropies, it is possible to determine the information gain $IG(a_l, c)$ from attribute a_l. It is calculated as the difference of the entropy of the category and the expected value of the entropy, when it is known, if attribute a_l occurs or not. The information gain thereby measures the reduction of the entropy when it is known if the attribute a_l occurs. If this reduction is bigger, it is more useful to know if attribute a_l occurs. $IG(a_l, c)$ is calculated as follows:

$$IG(a_l, C) = H(C) - \sum_{o\in\{0;1\}} P(O(a_l) = o)\cdot H(C|O(a_l) = o)$$

A different way to determine the weight w_l is the gain ratio which is calculated by dividing the information gain by the entropy of the attribute (also called the split information of the attribute). This step normalizes the information gain for attributes with different numbers of values:

$$GR(a_l, C) = \frac{IG(a_l, C)}{H(a_l)} = \frac{IG(a_l, C)}{-\sum\limits_{o\in\{0;1\}} P(O(a_l) = o)\cdot\log_2(P(O(a_l) = o)}$$

The distance measure was also modified to use distance weighting. This means that instances with a smaller distance were emphasized in the determination of the class. Several distance functions $f_m(d)$ are used for this. The ratio of the spam probability and the legitimate probability that determines the categorization of the email is computed differently. The contribution of each neighbor to the probabilities is determined by its distance. If c_i is one if email i is spam and zero otherwise, the email with the extraction result set $\vec{x}$ is considered to be spam if:

$$\frac{\sum\limits_{i=1}^{k} f_m(d(\vec{x}, \vec{x_i}))\cdot c_i}{\sum\limits_{i=1}^{k} f_m(d(\vec{x}, \vec{x_i}))\cdot(1 - c_i)} > \lambda$$

The four following distance measures were compared. The first one is a linear weight, while the other three represent hyperbolic weights. $d_{\max}$ is the maximum obtainable distance.

$$f_0(d) = d_{\max} - d$$

$$\text{and } f_m(d) = \frac{1}{d^n},\ m \in 1, 2, 3$$

Results

The corpus used for testing is again the "Ling-spam corpus" described in 3.1.4 with 2412 legitimate and 481 spam emails. For testing purposes, the number of attributes was varied from 50 to 700 and the neighborhood size from one to ten. It was also examined how the training corpus size and the use of attribute weighting and distance weighting affected the results.

The results for the use of attribute weighting depended on the parameter λ that defines how much more costly a falsely filtered legitimate email is than an unfiltered spam email. With $\lambda = 9$ and $\lambda = 99$ the use of information gain outperformed the other approaches. For $\lambda = 999$ the results were different depending on the number of attributes. When less than 200 attributes were used, the algorithm without attribute weighting was better than the others. When using at least 200 attributes, the best version was alternating between gain ratio and information gain. The authors conclude that information gain is the best approach.

When testing the different distance weighting formulas, $f_3(d)$ resulted in the best filter accuracy. This means that the best results are achieved when closer instances are strongly emphasized. The best value for k identified in another test series was eight for $\lambda = 1$, two for $\lambda = 9$ and four for $\lambda = 999$.

Table 3.10 shows the number of errors that occured using the best value for k for each λ and the weighted error rates. The values for $N_{S \to L}$ and $N_{L \to S}$ are not integers because the authors used ten-fold cross-validation and calculated the average.

λ	$\mathbf{N_{L \to S}}$	$\mathbf{N_{S \to L}}$	$\mathbf{r}_{\text{fp}}$	$\mathbf{r}_{\text{fn}}$	$\mathbf{W}_{\text{Err}}$
9	0.5	8.7	0.207%	18.07%	0.587%
99	0	15.4	0%	31,98%	0.064%
999	0	15.4	0%	31,98%	0.006%

Table 3.10: Results of the approach by Sakkis et al.

Discussion

The authors of this study tested several modifications of the k-nearest-neighbor algorithm and the distance measures. The results they achieved show that this testing can lead to much better results than using the regular algorithm and the Euclidean distance only.

A point that can be criticized is that the authors do not mention whether they used header information. It seems quite unlikely that it was used. This leaves

some room for improvement.

3.5 Comparison of all Discussed Approaches

Table 3.11 summarizes the weighted error rates achieved by all filters. The differences between them are quite big. If we take a look at the different filter methods first, it seems like only the artificial neural network and the memory based approach can compete with Bayesian filters. However, Paul Graham's results outperform all others by far. The more intuitive false-positive and false-negative rates confirm this. The values 0% and 0.229% achieved by Graham's filter are better than all other error rates. Although there might be reasons like the big training corpus that led to this, an implementation of this concept seems to be the most reasonable decision.

λ	**Bayes (Pant)**	**Bayes (Micr)**	**Bayes (Andr)**	**Bayes (Grah)**	**GA (Kati)**	**ANN (Drew)**	**MBL (Sakk)**
9	1.622%	2.198%	0.456%	0.011%	6.0%	1.009%	0.587%
99	0.681%	1.742%	0.073%	0.001%	3.6%	1.062%	0.064%
999	0.588%	1.700%	0.007%	0.0001%	3.36%	1.067%	0.006%

Table 3.11: Weighted error rates of discussed approaches

Another reason why Naive Bayesian filters are more suitable to the use as spam filters than artificial neural networks is that they can be easily adjusted if new emails are added to the database. If an artificial network should be adjusted, it must be retrained as mentioned in 3.3.2. This process can take much time.

4 Improving Existing Approaches

This chapter describes how the existing approaches explained in the previous chapter can be improved. There are two different starting points to find concepts for a better recognition rate: The filter method and the attribute selection.

4.1 Filter Method

It is rather complex to find a whole new filter method because the methods used in the existing approaches have been improved and redesigned over many years. Another approach is to use an established concept that has been proved to have a good performance for the problem and try to obtain better results by changing parameters. Paul Graham did this quite successfully in the design of his spam filter [Gra02a]. His results are superior to those of the other filters described because he tried to find promising values for the parameters of his implementation. The spam filter implemented for this research work should use this concept, too. A selection of Paul Graham's Bayesian filter is a good starting point from which to build a sound spam filter.

4.2 Attribute Selection

The room for improvement by utilizing attributes that have not yet been used might be bigger. There have already been several attempts to use this scope as described in the previous chapter. An example for this is using the Porter Stemming Algorithm to accelerate the learning process of the filter. The software implemented in the course of this paper should make use of similar attempts to improve the spam filter.

An opportunity is to use images embedded or linked in emails. These images have not yet been regarded. However, it is an increasing problem that more and more spam emails do not contain anything but a picture linking to a web site. In "The Spammers' Compendium" [GC04], a website maintained by John Graham-Cumming that deals with tricks that spammers use to avoid spam filters, this technique is listed at the first position. These emails do not contain sufficient attributes in order for them to be classified as spam. The only possibility for conventional spam filters is to use the HTML tags in the email and the link to the image. But images can also appear in legitimate

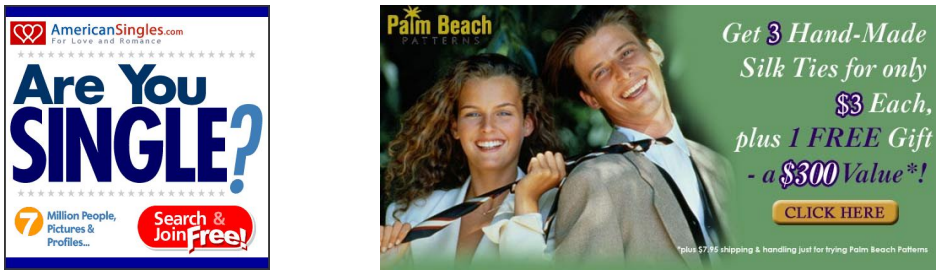

Figure 4.1: Images from spam emails

Figure 4.2: Images from legitimate emails

emails, for example holiday pictures sent to a friend. The conclusion of this is that images included or linked to within emails must be used for classification.

Using image data to classify emails requires the extraction of attributes from these pictures. The attributes extracted must be adequate to separate spam emails from legitimate emails. For this reason we have to take a closer look at images occurring in both kinds of emails. The images shown in figure 4.1 and figure 4.2 are typical examples.

The difference between both image types appears obvious to a human observer. The images from spam emails are at least partly graphics. The first one only consists of characters and small single-colored diagrams. The second image can be split vertically into two halves. The left part showing the couple is a photograph; the right half is similar to the first image, it shows letters on a monochrome background. The two pictures representing those occurring in legitimate emails are different, as both of them are photographs.

It is debatable whether or not the assumption that legitimate emails are more likely to contain photographs than spam emails is correct. There are, for example solicited emails containing images that are graphics, e.g. newsletters. Another example are emails containing comic strips or emails from colleagues containing data in the form of a graphic. However, if somebody receives such legitimate emails, this will be learned by a machine learning algorithm and the filter behavior will be adjusted. In contrast to this neutral behavior, a person not receiving such images in legitimate emails will benefit from an attribute showing whether an image is a photo or has graphical contents.

Now it is known which aspect of an image the attribute that is extract from it has to represent. It has to show whether the image tends to have a graphical or a photographical content. The question that arises is how this can be mea-

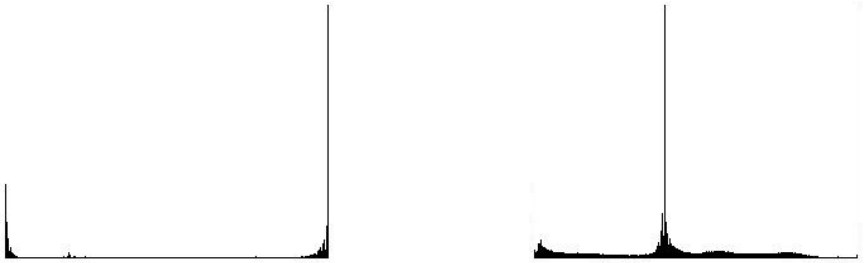

Figure 4.3: Histograms of images from spam emails

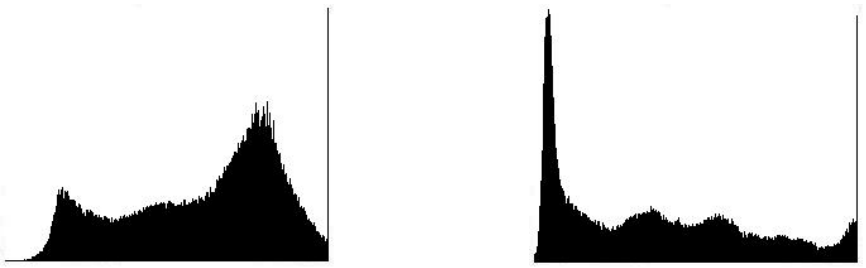

Figure 4.4: Histograms of images from legitimate emails

sured. The solution to this problem is that graphics have large, homogeneous areas of exactly the same color whereas in photographs the colors are most likely different in two neighboring points. This can be used in several ways to extract attributes from images.

The paper "Picture-Graphics Color Image Classification" [PCH+02] describes an approach for classifying images into the two groups of graphics and photographs. It uses a combination of three attributes. The first attribute is "spatial gray level dependence texture features". This tries to separate graphics from photographs by measuring the texture-density of the image. Photographs tend to be more textured than graphics which are rather smooth with large monchrome areas. However, the calculation process for this attribute is costly.

The second attribute used in [PCH+02] is the color discreteness. For this purpose, the images are first smoothed by a 4×4 averaging filter that removes noise due to half-tone. The color model used to derive this attribute is CIELUV, a rather specialized color model that provides perceptual uniformity. This means that the Euclidian distance of two colors in the color space corresponds to the difference between the colors for a human viewer. However, it is questionable whether this attribute could also be derived using a "regular" RGB color space and without using the 4×4 averaging filter. The color discreteness is calculated using the normalized histogram for each color component. For each color component (in the RGB color model red, green and blue), there are 256 possible intensities $(0 \ldots 255)$ a pixel in the image can have. $I_c(n)$ is the number of occurrences of the color value n for color c in the image. The normalized histogram for color component c is calculated by dividing the number of occurrences for one shade by the sum of pixels in the image:

$$H_c(n) = \frac{I_c(n)}{\sum_{i=0}^{255} I_c(i)}$$

To calculate the color discreteness for one color component c, the difference of all neighboring color values in the normalized histogram is calculated. These differences are summed up:

$$R_c = \sum_{i=0}^{254} |H_c(i+1) - H_c(i)|$$

As mentioned above graphic images contain areas of the same color and therefore will have sharp peaks in the histogram. The value of the color discreteness, increases if such peaks occur. It can be calculated easily for each component c of the color-model.

Figure 4.3 shows the color histograms for the red color component of two spam pictures from figure 4.1, and figure 4.4 for the legitimate pictures from figure 4.2. The resulting color discreteness values for the components red, green and blue are shown in table 4.1. It seems like this value can be used to separate graphics from photographs, as it is seen that all values for the images from legitimate emails are less than 0.1 while those values for the spam images are greater than 0.3.

Image	$\mathbf{R_R}$	$\mathbf{R_G}$	$\mathbf{R_B}$
Legitimate image 1	0.071	0.046	0.06
Legitimate image 2	0.097	0.096	0.097
Spam image 1	0.606	0.607	0.9
Spam image 2	0.403	0.382	0.463

Table 4.1: Color discreteness

The third attribute introduced in [PCH+02] is the edge feature. The authors use a "Canny edge detector" to extract edges from the picture. The underlying observation that justifies this attribute is that pictures are noisy and contain many short and broken edges while graphics have sharp and long edges. To extract an attribute from the edges, the number of connections between these edges is counted. The attribute is the number of edge pixels divided by the number of connected edges.

For a spam filter that depends on fast algorithms to filter emails quickly, the first and the last attribute do not seem to be very appropriate, as both of them are rather costly. However, this is not a big problem since, according to the authors, the used neural network (that derived a decision based on the values of all attributes) "gave significant importance to the first color discreteness feature" [PCH+02]. This shows that the most appropriate method for spam filters is to only use the discreteness attribute.

5 Implementation

This chapter describes the implementation process for a multi-user spam filter that is based on the techniques described in the previous chapters. The filter should be a Bayesian filter as this, according to chapter 3, is the best compromise between fast learning and high accuracy. This filter should be improved by using attributes derived from images which are included in the emails as described in chapter 4.

The description of the implementation begins with an overview of the technique used to integrate the filter into the email system of an organization. After this, the software behavior is shown in activity diagrams. The chapter is completed by the description of all classes and their methods in 5.3.

5.1 Integration into the Email System

There are several options of how to integrate a spam filter into an email system. It can be installed on the client computer, on the mail server itself or on a secondary server (proxy). When it is installed on the mail server or on a proxy it is appropriate to use IMAP folders to sort emails. However, when the filter is a program running on the client computer, it sorts emails that have been downloaded by the email program.

5.1.1 Integration into the Mail Server

This option works as shown in figure 5.1. The spam filter program is integrated into the mail server. The filter process can be started whenever an email arrives and the email can be moved into a certain folder whether it is considered to be spam or not. This is an advantage of this approach because the filter program

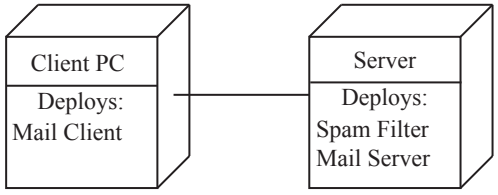

Figure 5.1: Spam filter integrated into the mail server

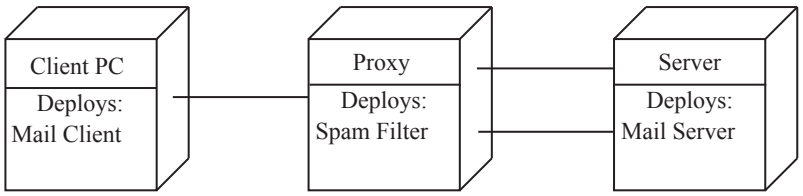

Figure 5.2: Spam filter as a separate proxy server

can access the mail server databases directly without using any email protocol. The emails can therefore be accessed fast and conveniently.

But this technique has disadvantages, too. First of all it requires either a mail server with an appropriate interface to program plugins or the implementation of a mail server from scratch. Organizations that wish to introduce a spam filter might not be willing to change their whole email system to a new mail server for this step since it implicates high costs. The new software has to be purchased, installed and customized. This involves downtimes and it requires training for the employees. Moreover, it is questionable whether the new software will be able to fulfill the needs of the organization.

5.1.2 Proxy Server

As figure 5.2 shows, a proxy server is a piece of software positioned between the mail server and the client program. However, this program can be run physically on the same computer as the mail server or the client. The proxy accepts IMAP connections from client programs on a certain port and establishes connections to the mail server. All communication between server and client is passed through and read along. Whenever the proxy recognizes a new email on the server, it is downloaded and classified. Depending on the filter result the proxy instructs the server to move the email to a certain IMAP folder.

An advantage of this approach is the high flexibility. An organization that introduces this filter program can keep the existing mail server if it is capable of the IMAP protocol. It is also possible to only introduce the filter program to parts of an organization, or to have separate proxies for organizational units that can have different configurations. The end user does not notice the change in the email system at all since the proxy is transparent. It is also possible to give the end users the possibility to switch off the spamfilter by contacting the mail server directly, for example on a different address or a different port number.

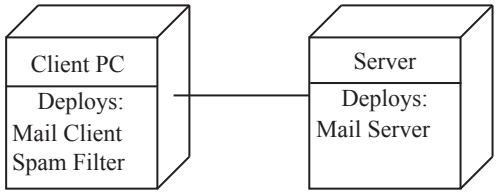

Figure 5.3: Spam filter integrated into the email client

5.1.3 Client-Side Program

The concept of a spamfilter on the client computer is shown in figure 5.3. The program waits until the client software has downloaded a new email. This email is analyzed and moved into a certain local folder if it is categorized as spam.

This approach has several disadvantages. The spam filter is limited to be used with only one mail client software. The filter's database can be lost due to system errors. If the data is saved on a server, it would be much safer because of more reliable hardware and usually more frequent backups. Another disadvantage is that the users have to download emails first before they are filtered. This step is especially costly and time consuming for them if they use slow connections like 56k modems, in particular if image attributes are used. Server-side IMAP solutions are superior to this approach because they move spam emails into certain folders. The client can view only the header information of these emails from time to time to verify that they are really spam and delete them on the server.

5.1.4 Decision

The conclusion that can be drawn from the advantages and disadvantages described above is that a proxy solution has the best characteristics. The main advantage is its flexibility as it can be used with any mail server and any client software.

5.2 Software Behavior

This section describes the processes the spam filter runs through. 5.2.1 deals with the communication sent from the client program to the mail server. The second part examines what happens when the server sends a message to the client. Finally, 5.2.3 illustrates how incoming emails are filtered. The filter process can be initiated by both types of communication previously described.

5.2.1 Client-Server Communication

Figure 5.4 shows an activity diagram that describes the method how client connections are handled. The proxy software has a server socket at a certain port that waits for clients to connect. Whenever a client connects to this port, it reads all messages transferred over this connection. After the client has established a connection, the proxy also connects to the mail server in order to forward messages from the client. The way in which messages sent by the mail server are handled is described in 5.2.2.

Each of these messages is handled by the routine shown in the diagram. Independent of the contents of the message, it is forwarded to the mail server through the primary connection. At the same time, the messages are also examined to see whether they include certain commands. To find these commands, we have to take a closer look at the IMAP protocol [Cri03]:

```
0001 login johndoe pwd4711
```

This is an example for a string sent from the user to the mail server. All command lines are prefixed with an identifier, also called tag, which is different for every command. In the example "0001" is the tag. The tag is followed by the command, in the example "login". The command is succeeded by arguments. In our case the arguments for the login command are the username "johndoe" and the password "pwd4711".

In order to understand the following steps, it is necessary to know that the spam filter needs a secondary connection to the mail server for each user that is logged in. This secondary connection is used to receive and filter new emails when they arrive. This process is described in depth in 5.2.3. To manage these connections, the proxy has to look for certain commands in the client-server communication.

The "login" command shown above is one of the instructions the proxy has to look for. In other words, the software has to check the second word of each command sent by the client for the word "login". When the user logs in, an existing secondary connection for this user is searched and if it does not exist, username and password are extracted and a new connection is created. In the activity diagram these steps are shown as "s includes login" for the check if the user logs in and $n_U = 0$ for the test if the user is already logged in. n_U is a variable counting the number of active connections for user U. The login procedure is completed by increasing n_U by 1.

The second command incoming strings are tested for is "logout". The associated string consists of the tag and the word "logout". If the user sends such a command line, the proxy checks if n_U equals one. In this case the connection that is closed is the last one using the secondary connection for this user. Therefore it can be closed, too. In any case n_U is decreased by one to indicate that the number of open connections has decreased. If the user logged out, the

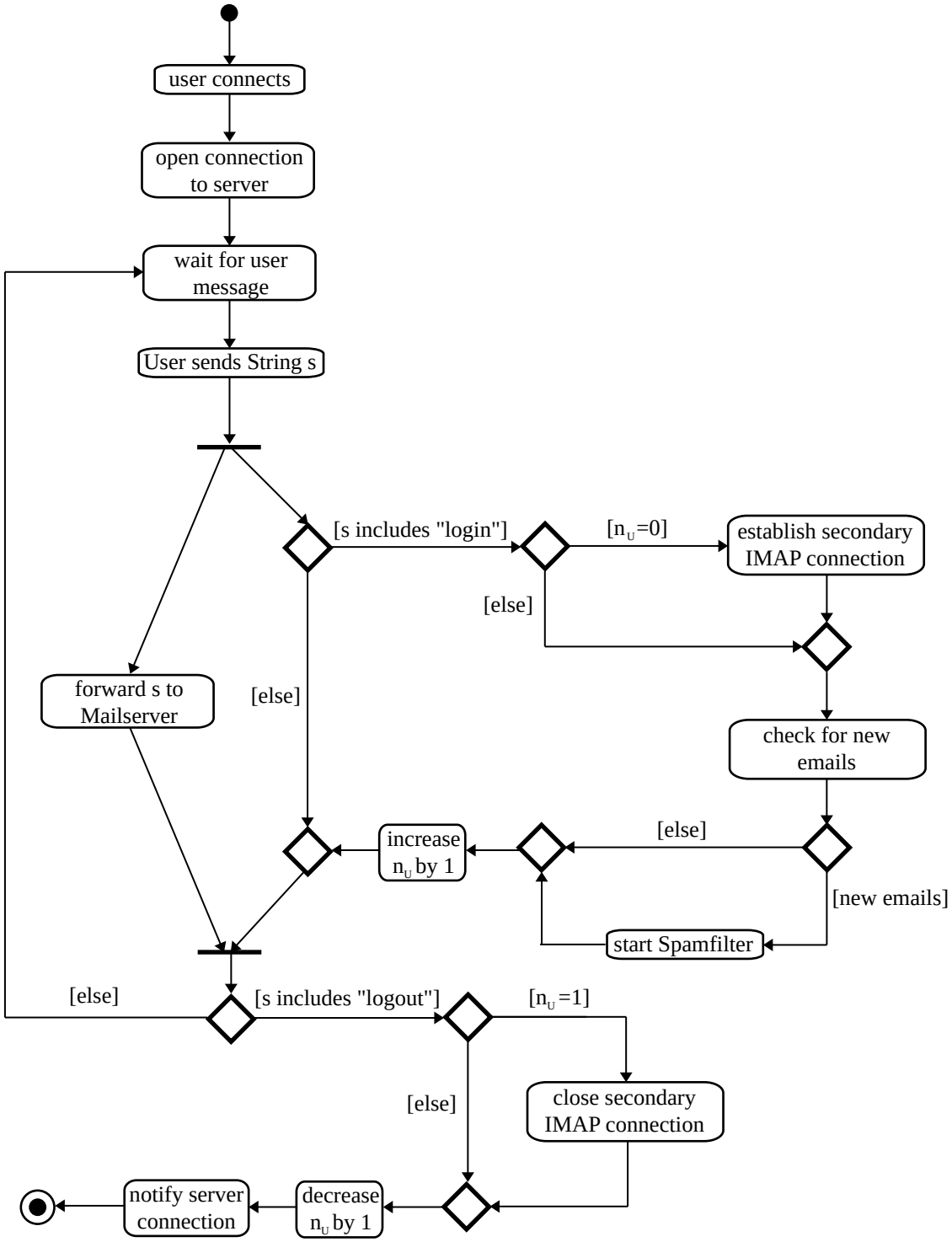

Figure 5.4: Activity diagram for user communication

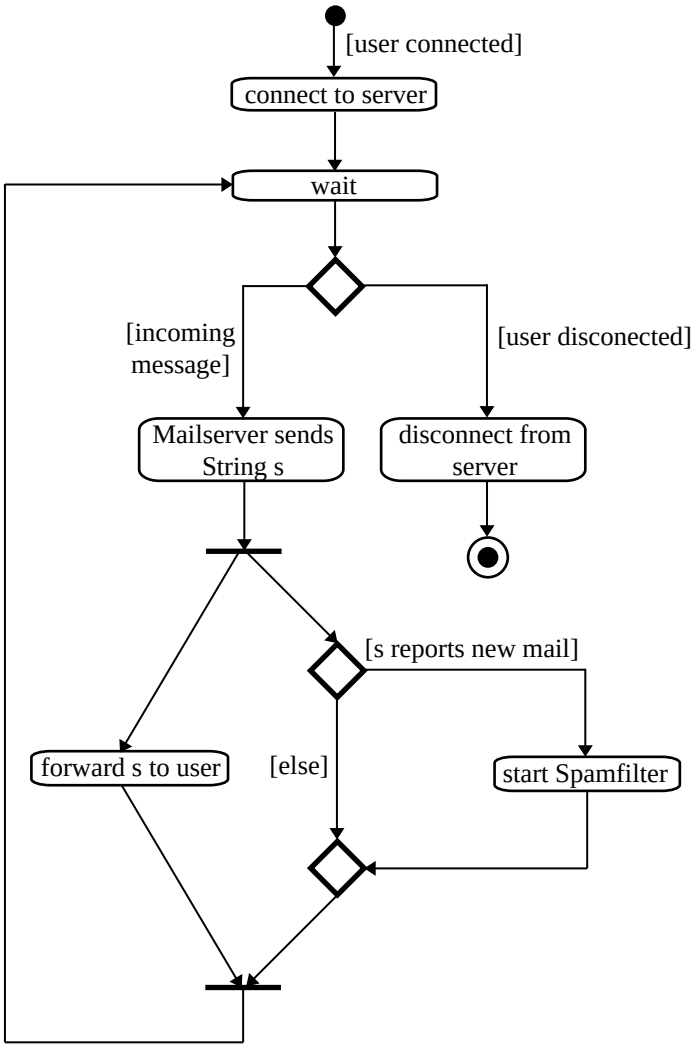

Figure 5.5: Activity diagram for server communication

server connection is notified in order to log out too, and the activity diagram is determined. If the command was not a "logout", the proxy starts over again waiting for incoming communication from the client.

The use of the connection counter n_U is necessary because some mail clients use several connections for one email account. It is not reasonable to establish a secondary connection for each of these. It makes more sense to use a common secondary connection. To manage this, we need a counter to notice when a new connection has to be established and when it can be closed.

5.2.2 Server-Client Communication

The activity diagram in figure 5.5 shows how incoming messages from the mail server are handled. The process is quite similar to that described in 5.2.1 for the opposite direction. It begins with a connection initiated by a client. When a client is connected, the proxy opens an according connection to the mail

server that is used to forward messages received from the client, and vice versa receive and forward messages from the mail server.

After establishing the connection, the spam filter goes into a state of waiting. One reason that causes it to get active again is a disconnection of the client, which also causes the proxy to close the connection to the server. The other reason for it to be reactivated is the reception of a message from the server.

Incoming messages are always forwarded to the client. Parallel to this, it is checked whether the message reports that a new email for the user has arrived. To be able to recognize this, we have to take a closer look at the IMAP protocol again, this time for data transmitted by the server [Cri03]:

```
* 22 EXPUNGE
* 23 EXISTS
* 3 RECENT
* 14 FETCH (FLAGS (\Seen \Deleted))
0001 OK NOOP completed
```

The server begins each line with a "*" if another line follows, and with the tag of the previous user command in the last string. The second line of the message shown in the example above notifies the user that 23 emails exist in the current mail folder. There are two possibilities when to check for new emails with the secondary IMAP connection. The first one is to check whenever a string combination like above occurs, with a number greater than zero for the existing messages. This would result in an unnecessary request whenever such a message is generated for a folder that is not the inbox. The other possibility is to check only when the new email information refers to the inbox. This would reduce the number of request through the secondary connection but it would require that the software keeps track when the client software changes the current folder. This would be more costly than the first option because the user communication would have to be scanned for commands switching the current folder and there are several commands that result in such changes. Therefore we use the first option and contact the mail server whenever an "exists" message with a number greater than zero occurs. In the activity diagram this is shown as "s reports new mail". After starting the spamfilter that is described in depth in 5.2.3, the proxy returns to the state of waiting for messages or a client disconnection.

5.2.3 Filter Process

Separating Spam from Legitimate Emails

The activity diagram shown in figure 5.6 illustrates how the mail filter of the proxy software works. The occasions when the filter is started have been mentioned in the preceding sections.

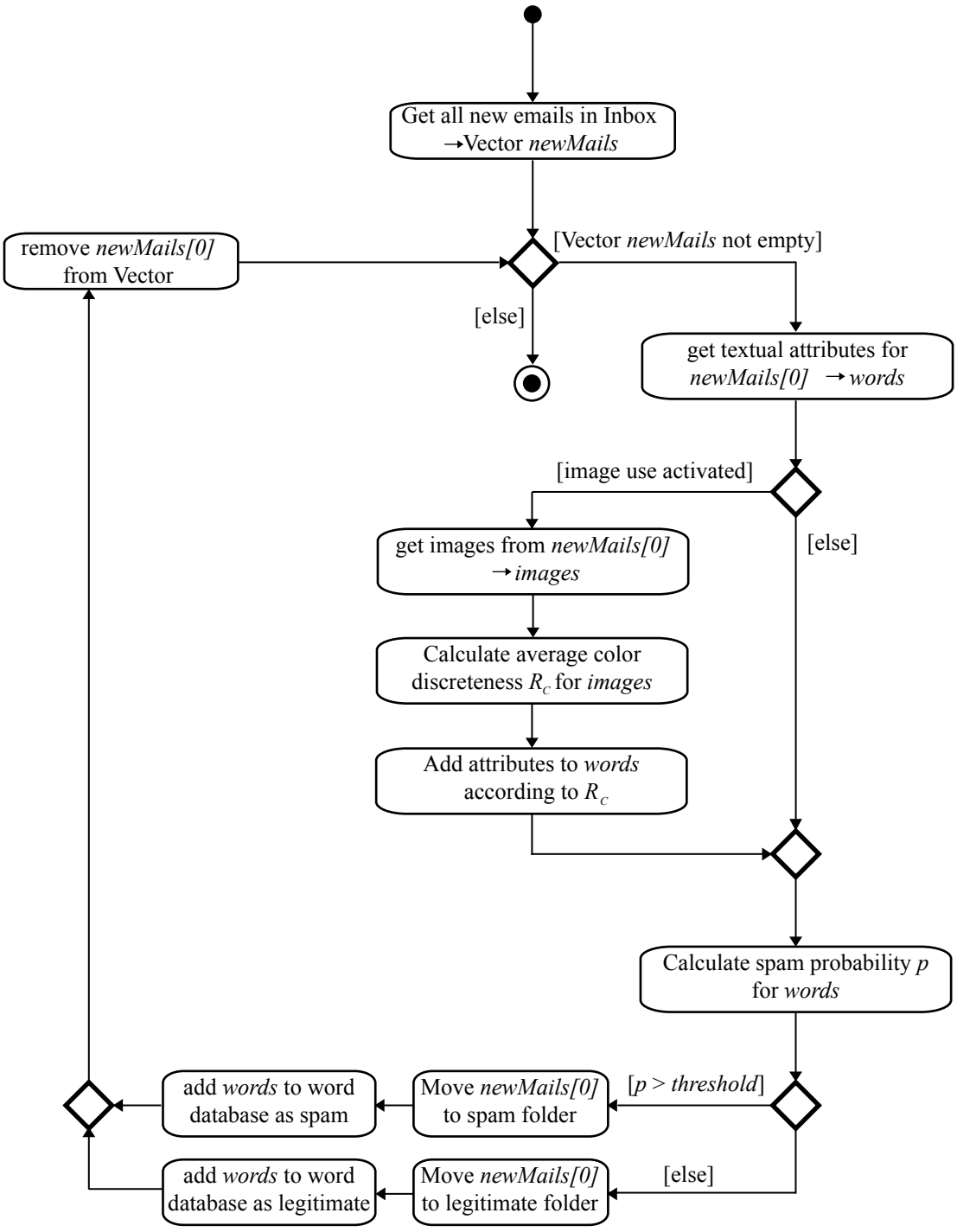

Figure 5.6: Activity diagram showing the spamfilter process

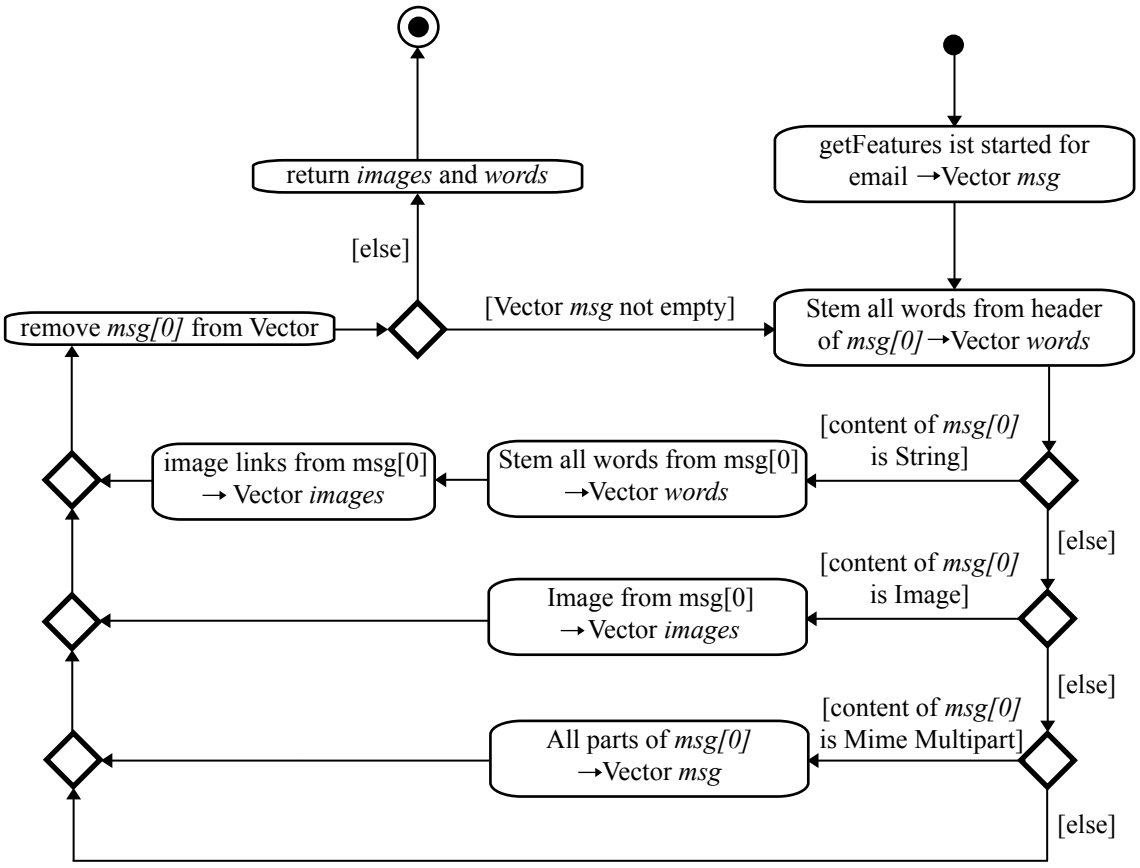

Figure 5.7: Activity diagram of feature extraction

The filter begins with retrieving all messages that are currently in the inbox. These emails are added to a vector, then they are processed one by one. The first element from the vector is taken and all textual attributes are derived from this message and collected in the vector "words". When starting the spam filter, the administrator can decide whether to use images for classification or not. If images are regarded, the next step is to get all the images embedded in the email. For these images the color discreteness is calculated as explained in 4.2. These values are averaged for all images from the email and all three color components. According to the result R_c, attributes are added to the vector words. This process will be explained later.

The attributes in the vector "words" are used to calculate the spam probability p. If this probability is greater than a certain threshold, the email is considered to be spam, otherwise to be legitimate. According to this categorization the email is moved to the legitimate folder or the spam folder and the calculated probabilities for each attribute are updated. Then the email is removed from the vector containing the new emails. If it was the last email in the vector, the algorithm is finished. Otherwise it continues with the following email.

Deriving Attributes from Emails

The activity diagram in figure 5.7 shows how attributes are derived from emails for the case that image-attributes are taken into account, too. The procedure starts with adding the email for which the attributes should be extracted to a vector. This vector is used because an email can contain several parts if it has the MIME type multipart [FB96]. That means the email contains several "virtual" emails that can be shown alternatively or combined. For example an email can consist of one part that is written in HTML and one part that is plain text. The email client is instructed to show the plain text version only if the user has disabled the display of HTML. The parts of a multipart message can also contain images or other multimedia data. Attributes have to be extracted from all parts of such messages and therefore all parts are added to the vector and processed in succession.

First all words contained in the header of the first element of the vector are stemmed using the Porter Stemming algorithm. These stemmed words are added to another vector called "words". Afterwards, the body of the message is checked for its type. If it is plain text or HTML, its contents can be extracted into one string. This string is divided into words and the words are stemmed and added to the word vector like those extracted from the header. If the message is HTML, it can also include links to pictures on the Internet which will be displayed in the email. These image links are searched within the string and added to a third vector called "images". If the content of the body is an image, it is added to "images", too. If the message has the MIME type multipart, all parts of it are added to the message vector. If the body has none of these types, meaning it contains other information, for example a movie file, it is not further considered. Finally the first element from the message vector, the element we just processed, is removed. If the vector contains additional messages, the loop is repeated with the next element, otherwise the feature extraction is finished. The word vector and the image vector are returned.

5.3 Class Concept

This section describes how the functionality explained in 5.2 is realized on the class layer. The first part gives an overview of which classes are used and how they interact. Then each class is analyzed separately.

As a foundation, two external packages have been used. The JavaMail API from Sun Microsystems [jav] provides basic functions for communicating with IMAP servers and handling emails. The aitools package, by Dr. Benno Stein and Sven Meyer zu Eißen, is used to tokenize and stem strings to obtain attributes and to manage them using efficient symbol processing algorithms. The classes used from these packages are also described in short.

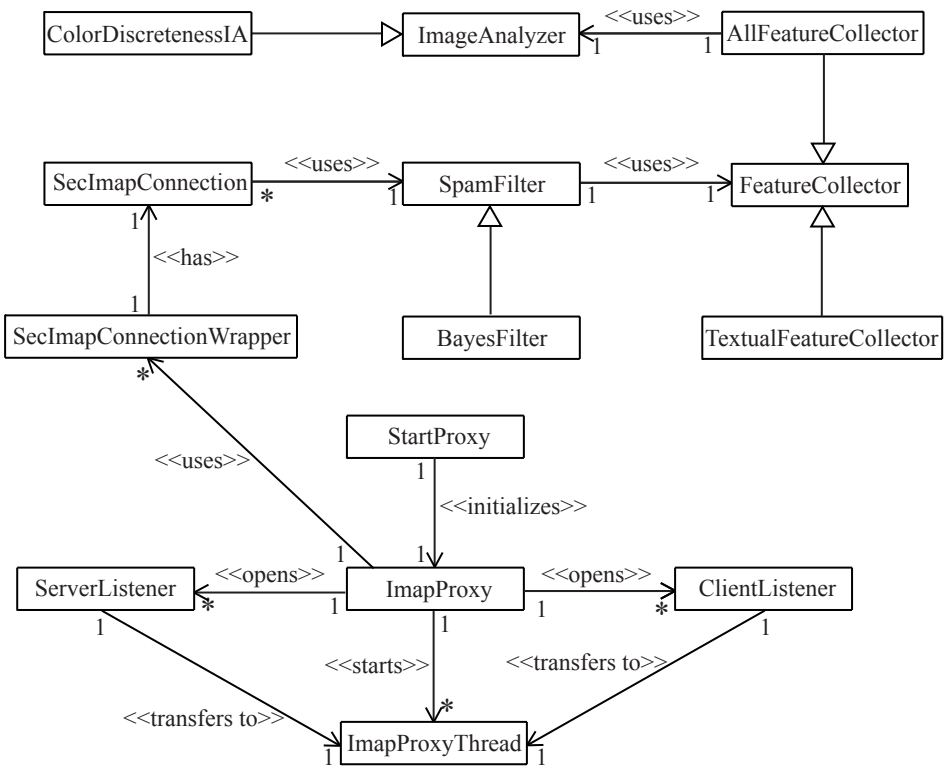

Figure 5.8: Overview of the major classes

5.3.1 Overview

Figure 5.8 gives an overview of which classes have been implemented for the spam filter and their relationships. Relations to classes from external packages are not included. Three additional classes are left out. These are extending the class SearchTerm from the JavaMail API in order to search emails that are flagged, not flagged or not deleted. These terms will be explained later.

The classes can be divided into two parts. The first part consists of all classes organizing the communication between mail server and client program. These classes are ImapProxyThread, ClientListener and ServerListener, they are positioned in the lower part of the diagram. The classes SecImapConnectionWrapper, SecImapConnection, SpamFilter, BayesFilter, FeatureCollector, TextualFeatureCollector, AllFeatureCollector, ImageAnalyzer and ColorDiscretenessIA in the upper part of the diagram use the secondary connection to the mail server for filtering spam. The class StartProxy initializes the whole program and ImapProxy organizes the collaboration. The functionality of the classes is illustrated by the three following examples.

Illustrative Example: Initialization and Login

The sequence diagram shown in figure 5.9 illustrates how the program is initialized and how a user login process is organized. With this example it is

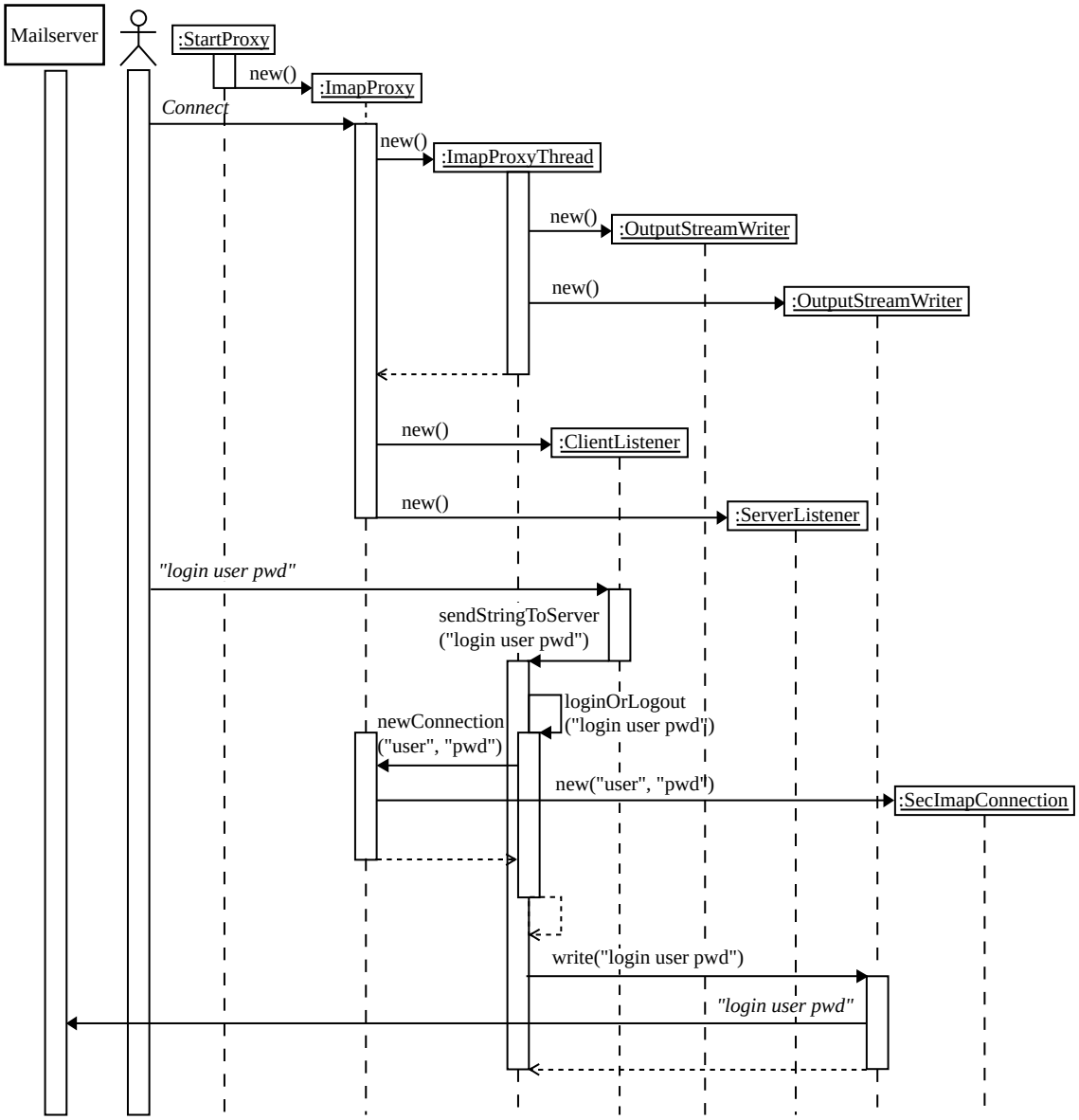

Figure 5.9: The initialization of the proxy and the login process

possible to understand what the purpose of each class is and which the most important methods are. In the following two paragraphs, the diagram will be briefly explained.

In the beginning, the class StartProxy initializes the class ImapProxy. This class makes a ServerSocket available to which a client program can connect. This causes the ImapProxy class to instantiate several classes: an ImapProxy thread; a ClientListener, using the connection initiated by the client; and a ServerListener, a connection to the mail server. The ImapProxyThread generates two objects of the type OutputStreamWriter itself. One of them is used to forward messages from the client to the server and one for the opposite direction. They are using the same sockets for the connection as the listeners.

The next step shown in the sequence diagram is the login command "login user pwd" sent by the client. The ClientListener accepts this message and calls the method sendStringToServer() in the class ImapProxy with this string as argument. This class recognizes that the string contains a login command, extracts the username and the password, and asks the ImapProxy for a SecImapConnection for this user. In the example, no previous connection for the user is existent. Therefore the ImapProxy initializes a new SecImapConnection for this user and returns it. Thereby the organizing ImapProxy class links the ImapProxyThread with the corresponding SecImapConnection. Finally, the ImapProxyThread uses the OutputStreamWriter of the mail server to forward the message.

Illustrative Example: Logout

The next process analyzed is the logout procedure shown in figure 5.10. It begins with the logout command sent by the client to the proxy. The ClientListener hands this message on to the ImapProxyThread which recognizes that it is a logout command and calls the method connectionClosed() in the ImapProxy class for this user. We make the assumption that the connection the client logs out from is the last one using this user's SecImapConnection. Therefore the ImapProxy calls the disconnect method for the SecImapConnection. After this, the ImapProxyThread sends the message to the mail server using the OutputStreamWriter.

The mail server acknowledges the logout command with the string "(...) LOGOUT completed". The ServerListener forwards this to the ImapProxyThread which first checks whether it informs that a new email has arrived and then calls the write method of the client's OutputStreamWriter to forward the message to the client.

After finishing their communication, the mail server and the client close their connection to the proxy. In our example, the client is the first one to disconnect, however the mail server could likewise be first. Since the client disconnects first, the ClientListener calls the method setConnectionNotAlive from the ImapProxyThread which closes both OutputStreamWriters and the Imap-

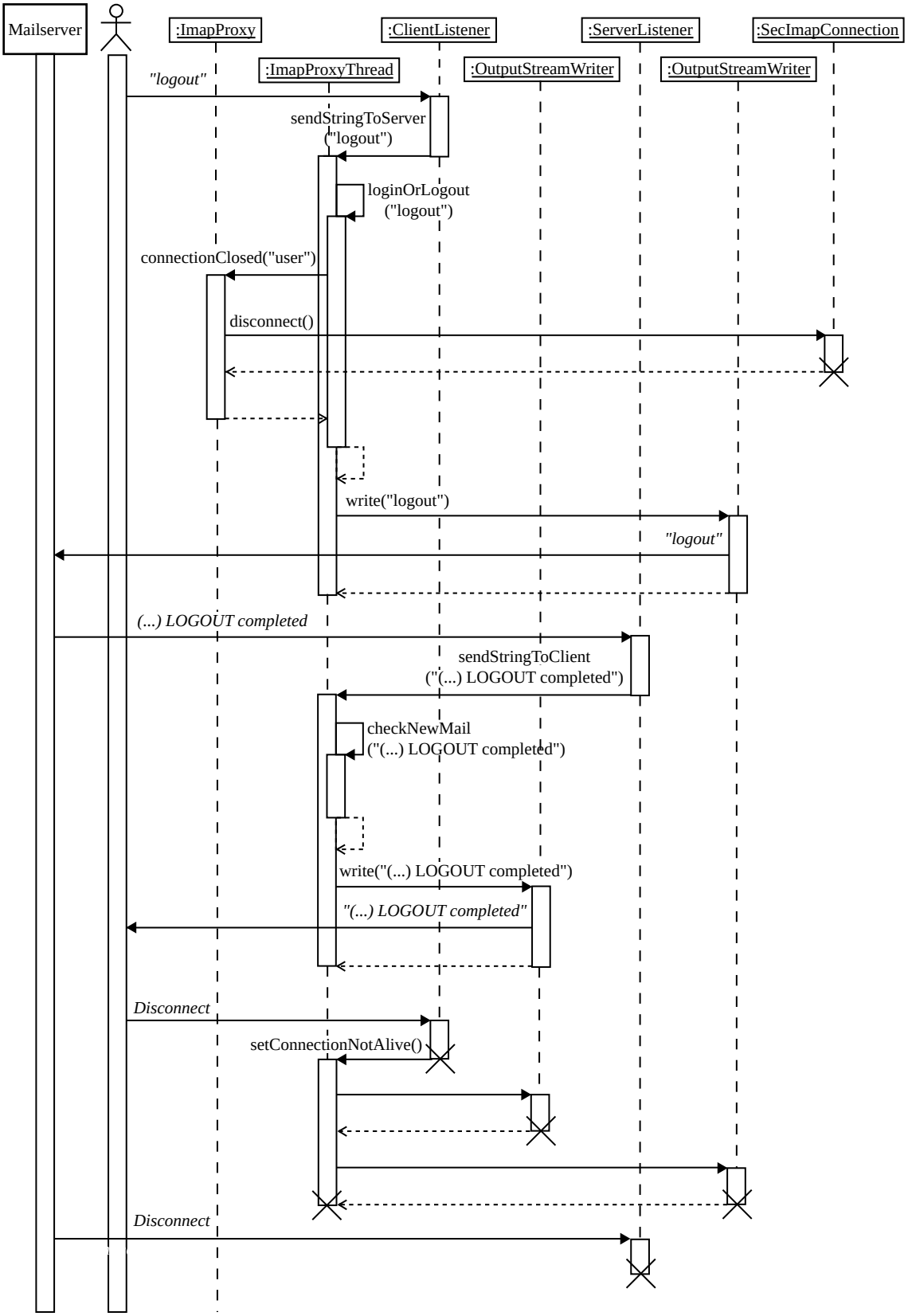

Figure 5.10: The logout process

ProxyThread.

Illustrative Example: Filtering new Emails

The last example discussed here explains the classes is shown in figure 5.11. It illustrates how incoming emails are filtered if image attributes are not considered. The process begins with the incoming message "(...) n EXISTS" from the server. This message states that there are n emails in the current folder. The ServerListener forwards this string to the ImapProxyThread by calling the method sendStringToClient(). The ImapProxyThread recognizes in the method checkNewMail() the type of the message and therefore starts with the method updateSpamFilter() from the class SecImapConnection.

The SecImapConnection now performs the filter process for all new emails. First of all, these emails are requested from the IMAPFolder inbox using an object of the type SearchNotDeleted. The messages are returned as the array "newMsg". For each of the emails in this array the proxy performs the following process: The SecImapConnection starts the method isSpam() implemented in BayesFilter. At this point any other class that extends the abstract class SpamFilter can be used. The BayesFilter uses the TextualFeatureCollector to retrieve features from the current email and uses these to run the method getSpamProbability(). If the calculated probability is greater than the threshold, the method addMail() adds the features as spam to the database and recalculates the corresponding probabilities. If the probability is less than the threshold, it will add the features as legitimate. However, we assume for our example that the email is treated as spam. The filter returns the boolean value true to the SecImapConnection which therefore moves the message to the spam folder.

The process described in the previous paragraph is performed for each email in the array "newMsg". In the example shown in figure 5.11 there is only one email in the array. The SecImapConnection finishes its activity and the ImapProxyThread calls the write function of the client's OutputStreamWriter which forwards the message from the mail server.

Figure 5.12 shows how the filter process is modified if attributes derived from images are regarded. In this case, the inheriting class of FeatureCollector used is not TextualFeatureCollector as above, but AllFeatureCollector. This class also uses ColorDiscretenessIA to derive attributes from images.

5.3.2 External Classes

Before section 5.3.3 discusses the implementation of the classes included in the package imapspamfilter, the classes used from external packages are explained. These are the aitools packages providing a data structure that is used to save and find objects efficiently and the JavaMail API. Classes from this package are used for the secondary connection to the mail server.

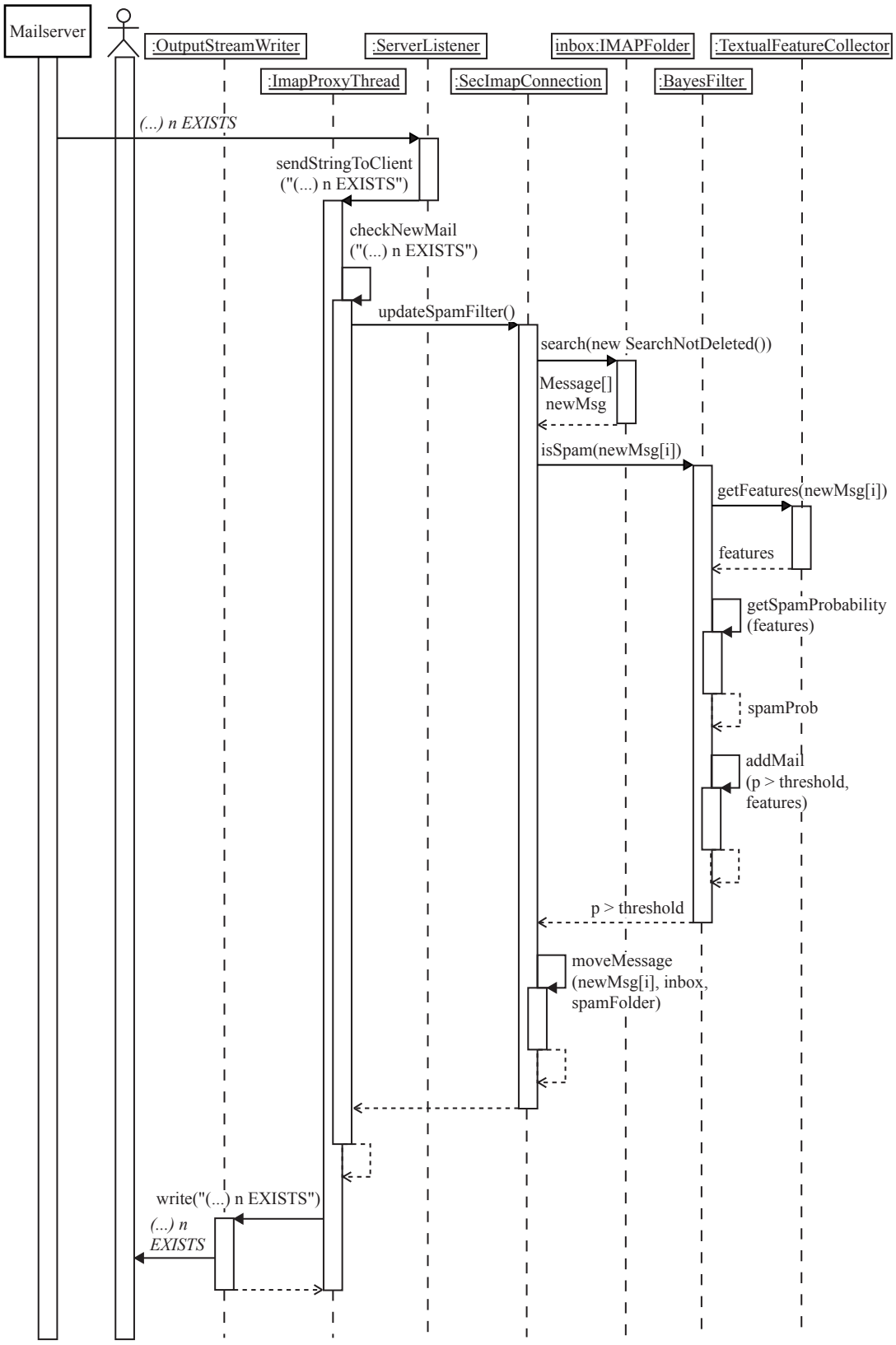

Figure 5.11: The filter process

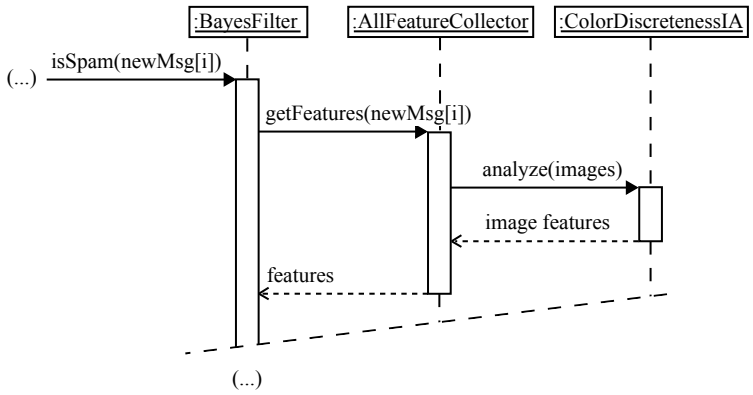

Figure 5.12: The modification of the filter Process for the use of image attributes

aitools

The major class used from this package is Symbol. It offers a data structure to save and find objects associated with a word. Another feature of the Symbol class is that it supports packages. This means that each Symbol object belongs to a certain package. In our case the names of the packages are the usernames. This allows us to build different sets of attribute names and associated integer values for each user, a basic requirement for the spam filter implementation on a per user basis.

The SymbolWrapper class implements a combination of Symbol objects and a vector called WordVector for each package. The Symbol objects contain an Integer object representing the position of the Symbol name in the WordVector. The WordVector gives the user the possibility to link back from an integer value to the corresponding Symbol. This data structure can be used to store information about the Symbol in other vectors.

The classes implemented during this research work make use of the class SymbolWrapper. They use the user name as the package name and attribute names derived from emails for Symbol objects. The Integer object included in a Symbol gives the position not only of the Symbol in the WordVector, but it also is the position of according values for the number of spam and legitimate emails containing the attribute within IntVectors and for the spam probability of this attribute within a DoubleVector. These vectors are combined in an object of the type UserDatabase for each user. This object can be found by searching for the username in the package “user-master-package”. The returned Symbol’s value is the UserDatabase object for the user.

IntVector and DoubleVector are also classes from the aitools package. They are similar to regular vectors but they offer some additional methods and they can handle primitive data types. IntVector is determined to include only integer values, DoubleVector includes double values.

OneGramIndexer is another class from the aitools package used by the proxy.

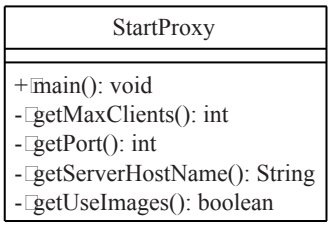

Figure 5.13: Class StartProxy

It splits strings into tokens. The standard version of this tool ignores all HTML tags, changes HTML special character codes like "ä" that represent umlauts into the corresponding character and stems all words using the Porter Stemming Algorithm [Por80]. The filter described by Paul Graham does not use these features. However they might be beneficial for the filter. This will be analyzed in chapter 6. The words are returned as a CompressedDoubleVector. This class is similar to a DoubleVector, it only leaves out all values that are zero. It consists of two vectors, one of them contains the values and the other one the corresponding offset to get the Symbol value and thereby the position in the word vector.

JavaMail API

The JavaMail API [jav] is used to establish the secondary IMAP connection to the mail server. The connection is established by a IMAPStore object, folders can be accessed through IMAPFolder objects. The package has the capability to extract parts of MIME Multipart messages. This feature is used to extract attributes from emails.

5.3.3 Class Descriptions

This section takes a closer look at the classes in the package imapspamfilter. The class variables and methods are shown in class diagrams and explained.

StartProxy

This class is used to initialize the filter. The main() method expects to be started with the arguments "-S" followed by the server host name, "-P" followed by the port to listen to, and "-M" followed by the maximum number of clients to connect with. These arguments are extracted by the three private methods in this class. The last private method getUseImages() detects whether the main() method was started with the argument "-I" which causes the filter to use image attributes. After checking all these arguments, a new ImapProxy is started.

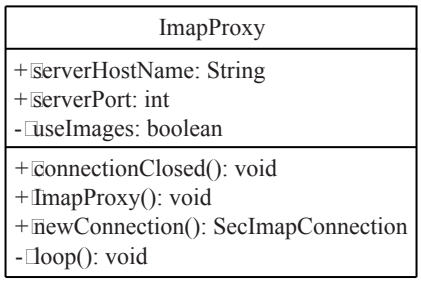

Figure 5.14: Class ImapProxy

ImapProxy

As previously mentioned, the class ImapProxy coordinates all connections of the proxy, primary connections as well as secondary. The string "serverHostName" and the integer value "serverPort" are declared public because the class SecImapConnection uses them for connecting to the mail server, too.

The constructor opens a ServerSocket which is listening to the port defined in the arguments when starting the server. The loop() method begins with the opened ServerSocket as an argument. As the name implies, this method is a loop waiting for clients to connect. Whenever this happens, it is checked whether the maximum number of connections has been reached. Otherwise a socket to the mail server is opened and a new ImapProxyThread, ClientListener and ServerListener are initialized for these connections.

The methods newConnection() and connectionClosed() organize the SecImapConnection objects for the users. The Symbol class from the aitools package is used to file a connection and the number of threads using it combined in an object of the type SecImapConnectionWrapper. The Symbol environment provides a fast data structure to get these objects subject to the username. newConnection() opens a new connection if necessary, returns it and increases the number of connections by one. connectionClosed() closes a SecImapConnection if it is no longer used and decreases the number of connections by one.

ImapProxyThread

Figure 5.15 shows the class imapProxyThread with all class variables and methods. This class implements the interface Runnable and can therefore be started as a thread. The number of class variables is quite high because some of them are set within the initialization and must be accessed when the thread is started, and because some of them are used by methods called by other classes.

The constructor assigns values to several of the class variables. Then it calls the method initializeWriters() which creates the OutputStreamWriters for mes-

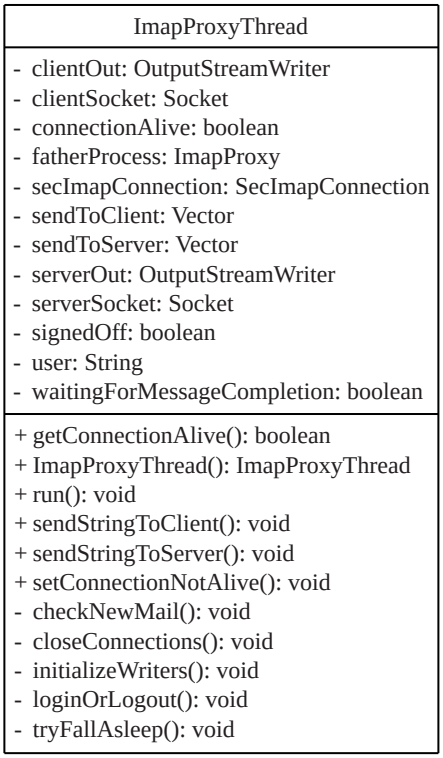

Figure 5.15: Class ImapProxyThread

sages to the mail server and to the client. After the class is initialized, the thread is started. This executes the run() method. This method checks in a loop the two vectors sendToClient and sendToServer to see if they contain any messages. If a message has to be sent to the client, the method checkNewMail() is called for this message, before it is written to the client's OutputStreamWriter. If a string for the server is available, loginOrLogout() is called for it before forwarding it using the according OutputStreamWriter. After finishing these steps, the method tryFallAsleep(), which tells the thread to wait if both vectors are empty, is called. When the thread awakes, the loop starts over. The process quits the loop when the boolean variable connectionAlive is false. When the process has left the loop, the method closeConnections(), which closes both sockets, is called. If signedOff is false, connectionClosed(), a method in ImapProxy, is run.

loginOrLogout() checks whether a string being sent to the mail server is a login or a logout command. The string is tokenized and the second token is compared with the words login and logout. If it is login, the username and the password are extracted, newConnection() from the class ImapProxy is started and signedOff is assigned the value false. If the string is a logout command, connectionClosed() is called and signedOff is assigned true. The boolean variable connectionAlive is deliberately not set false at this point because the proxy always waits for the mail server or the client to close the connection. When one of them disconnects, the other connection is closed, too.

The method checkNewMail() is similar to loginOrLogout(). It also tokenizes

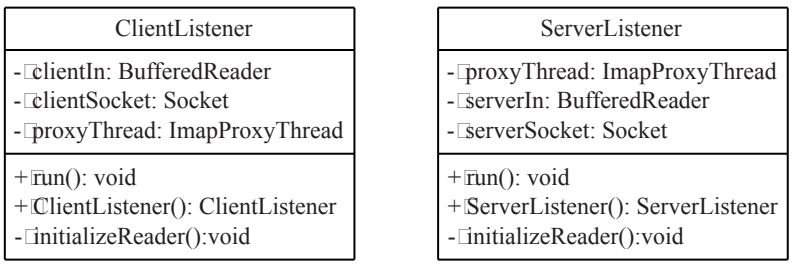

Figure 5.16: Class ClientListener and class ServerListener

strings sent by the server and waits for a "n EXISTS" notification with n greater than zero. When such a message occurs, the method waits for the end of the server message stream indicated by a leading "*" in the last string and starts updateSpamFilter() from SecImapConnection.

The method getConnectionAlive() returns the value of the boolean variable connectionAlive. setConnectionNotAlive() is called by the listeners when the user or the mail server disconnects. It assigns false to connectionAlive and starts the thread if it is waiting. The methods sendStringToClient() and sendStringToServer() add a string to the corresponding vector and notify waiting threads.

ClientListener and ServerListener

These classes will be discussed together because they have the same functionality. ClientListener receives messages from the client and forwards them to the corresponding ImapProxyThread. ServerListener does the same for messages from the server. For incoming strings they initialize a BufferedReader in the method initializeReader() called by the constructor. The classes implement the interface runnable. When the threads are started, the run() method reads from the BufferedReader. Whenever the communication partner of one of the classes disconnects, setConnectionNotAlive() from ImapProxyThread is called.

SecImapConnection and SecImapConnectionWrapper

Figure 5.17 shows a class diagram containing these two classes. SecImapConnectionWrapper is just a small class combining a SecImapConnection with a counter called numConnected. The methods implemented are get-methods for these two class variables, two classes increasing and decreasing numConnected by one, a set method for the SecImapConnection and the constructor.

SecImapConnection is more complex than the wrapper. The constructor begins with the initialization of a BayesFilter. Depending on whether images should be used to derive attributes from emails, the BayesFilter is either started with an AllFeatureCollector or a TextualFeatureCollector. The con-

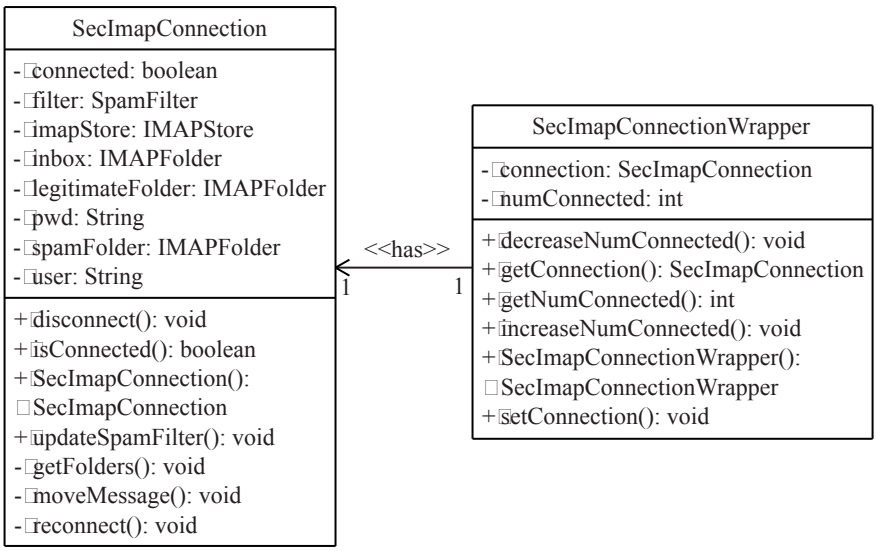

Figure 5.17: Class SecImapConnection and class SecImapConnectionWrapper

structor then opens a connection to the mail server using an IMAPStore object from the JavaMail API. Then the method getFolders() is started. When it is finished, the spamfilter is updated calling updateSpamFilter().

getFolders() opens the three folders inbox, spamfolder and legitimatefolder. If they do not exist, they are created. Then the method opens two more folders called “init-spam” and “init-legitimate”. These folders can be used to initialize a spamfilter with existing emails. The user just creates these folders as subfolders of his inbox and copies all the spam emails he has into the init-spam folder and legitimate emails into the other one. getFolders() calls the addMail() method of the BayesFilter for each of these messages and moves them into the spam and legitimate folder using moveMessage().

The method updateSpamFilter() filters newly arrived messages and modifies the attribute database if the user has moved an email from the spam folder to the legitimate folder and vice versa. It begins by checking whether the connection to the mail server is still open. If it is closed, the method reconnect(), that opens the connection again, is called. For each new email in the inbox the method isSpam() from BayesFilter decides if it is regarded as spam or not. According to this, the messages are moved to the spam folder or the legitimate folder.

updateSpamFilter() also changes the value of the “Flagged” flag, it is set for spam emails and disabled for legitimate ones. Flags are boolean variables affiliated with an email. They are used as indicators, for example to show if an email has been read. The “Flagged” flag is used to recognize when the user moves a message considered to be spam to the legitimate folder. However, this is not the perfect solution since the “Flagged” flag that usually indicates urgent emails can no longer be used. in most mail servers it is possible to

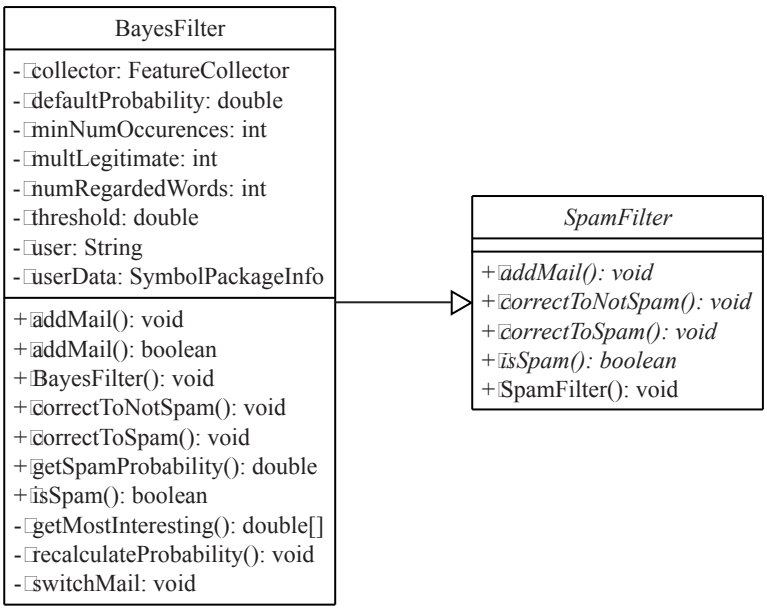

Figure 5.18: Class SpamFilter and class BayesFilter

define new flags. If the administrator of the proxy program has the rights to define flags on the mail server, a new flag called "Junk" would be the best solution. But for compatibility reasons, the standard flag used is "Flagged".

After categorizing all new emails, updateSpamFilter() checks if the user has moved emails from one folder to the other. If this is the case, either the method correctToSpam() or the method correctToNotSpam() is called and the flag value is changed.

The method isConnected() checks whether the connection to the mail server is open and returns this as a boolean value. When disconnect() is called, all folders and the ImapStore are closed.

SpamFilter and BayesFilter

The class SpamFilter is an abstract class that can be extended by several kinds of filters. These must implement four different methods: addMail() which uses a spam or legitimate email to train the filter; correctToSpam() and correctToNotSpam() which change an email's categorization from spam to legitimate and vice versa; and isSpam(), a method returning a boolean value indicating if an email is assumed to be spam.

BayesFilter extends SpamFilter. It is an implementation of the algorithm by Paul Graham described in 3.1.5. The class variables "threshold", "minNumOccurences", "defaultProbability" and "numRegardedWords" are parameters that can be used to adjust the filter.

The constructor of BayesFilter retrieves a Symbol object from the package

"user-master-package". This package includes one object for each user who has ever logged on to the proxy. This Symbol contains a UserDatabase object which contains most of the data for the user's spam database. It provides a DoubleVector with each attribute's spam probability and two IntVectors counting the number of occurrences in spam and legitimate emails for each attribute. It also contains two counters, one for the total number of spam emails and one for the total number of legitimate emails. The algorithm searches for the Symbol with the same name as the attribute. The included Integer object is used to obtain the corresponding value from the vectors.

The method isSpam() returns a boolean value indicating whether an email is spam. It begins with calling the method getFeatures() from FeatureCollector to retrieve all attributes from the email. Then the method getSpamProbability() calculates the probability that the email is spam using these attributes. If it is greater than the threshold, the attributes are added to the database by calling addMail with "true" as the second argument, otherwise with "false". Finally this boolean value is returned.

getSpamProbability() uses the method getMostInteresting() to retrieve those spam probabilities that are most different from 0.5 as defined by Paul Graham. The total spam probability that is returned is calculated from these figures also following this approach. getMostInteresting() works with an array containing double values. The size of it is defined by the class variable numRegardedWords. In the beginning it is initialized with -1 at each position. Then each attribute's probability is added to it if there is another value included that has a smaller absolute difference to 0.5 or if there is still a -1 in the array. It replaces either a -1 or, if there is none left, the value that is closest to 0.5. Finally, the array is returned.

The method addMail() increases either the number of spam emails or the number of legitimate emails in the user's UserDatabase object by one. Then it increases the number of occurrences in spam or legitimate emails for each extracted attribute by one and starts recalculateProbability() for it. This method begins with adding new values to the probability vector if the position that should be recalculated is greater than the number of values in the probability vector. The new double variables are initialized with the value of the class variable "defaultProbability". Then the value for the attribute at the requested position is calculated according to the formula defined by Paul Graham and it is added to the vector.

The methods correctToSpam() and correctToNotSpam() begin by extracting the attributes of the email. Then they call switchMail() for the offsets and the values of the attributes. The last argument of this method call is a boolean value which indicates if the email should be switched to spam or to legitimate. switchMail() works in almost the same manner as addMail(). The only difference it that it does not only add the values to one side, it also subtracts them from the other side.

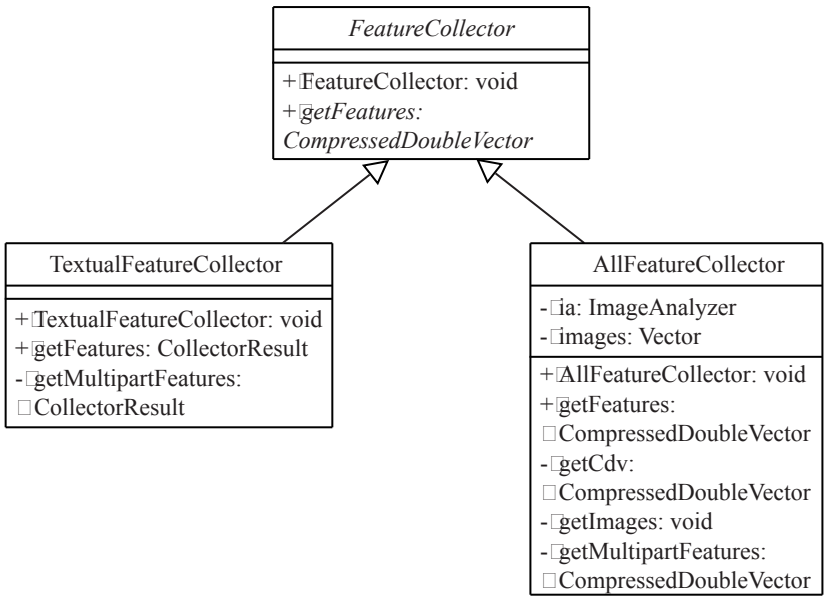

Figure 5.19: Class FeatureCollector, class TextualFeatureCollector and class AllFeatureCollector

FeatureCollector, TextualFeatureCollector and AllFeatureCollector

FeatureCollector is an abstract class defining only one method that has to be implemented in inheriting classes: getFeatures(). This method returns a CompressedDoubleVector which includes values for all attributes derived from the email.

TextualFeatureCollector is a class extending FeatureCollector. It extracts all attributes from the header and body of emails but it does not derive attributes from emails. The recursive algorithm used has already been explained in 5.2.3 and will therefore not be explained again. It is encoded in the methods getFeatures() and getMultiPartFeatures(). The class OneGramIndexer described in 5.3.2 is used to tokenize and stem the strings extracted from emails.

AllFeatureCollector extends FeatureCollector, too. It uses the same algorithm as the TextualFeatureCollector, but derives additional attributes from images and all occurring exceptions. The use of exceptions is reasonable because some spammers break certain rules for composing emails in order to interrupt spam filters. The analysis of such emails might lead to exceptions.

Images are derived either from Mime Multipart emails that contain a file or from links in HTML messages. Included emails are extracted within the method getFeatures(). Linked images are extracted by getImages(). The HTML tag below shows an embedded image link:

```
<img src="http://www.spamimages.com/img1.jpg">
```

Since most email clients also accept other notations, e.g. without quotation

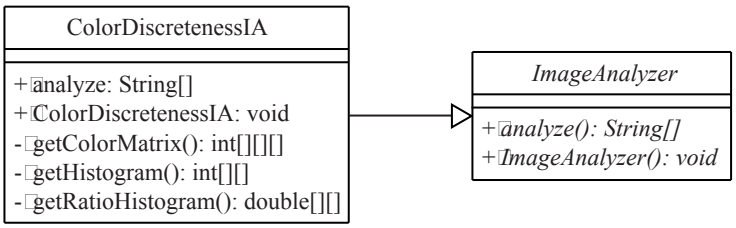

Figure 5.20: Class ImageAnalyzer and class ColorDiscretenessIA

marks or with apostrophes instead of them, the algorithm to find these addresses is quite complex. Images found in the text are downloaded from the Internet and returned.

After extracting all images from the email, the method analyze() from an ImageAnalyzer is called. It returns an array of strings that are used as image attributes. To add these attributes to the CompressedDoubleVector, the method getCdv() finds the corresponding Symbol objects and constructs a CompressedDoubleVector.

ImageAnalyzer and ColorDiscretenessIA

ImageAnalyzer is another abstract class. Inheriting classes can be used to analyze images derived from emails. The according procedure must be implemented in the method analyze() which returns an array of strings.

ColorDiscretenessIA extends ImageAnalyzer according to the calculation described in 4.2. The method getColorMatrix() converts the image into a matrix including the color values of each pixel in the RGB model. getHistogram() adds them up for each color. The values from this histogram are converted by getRatioHistogram(). It calculates which ratio of all pixels has a certain color value. The resulting histogram is used by analyze() to derive the color discreteness of each image included in an email. From these values, the weighted average is calculated; each image is weighted with its number of pixels. The advantage of a weighted average is that a spammer cannot bias the filter by including small pictures with a low discreteness in order to lower the resulting average.

The strings that are returned by this method are determined by this average discreteness. Three different strings can be returned indicating if it is smaller than the thresholds 0.5, 0.3 and 0.1. E.g. an email containing images that result in an average color discreteness of 0.2 will result in the strings “dis_sm_0.5”, and “dis_sm_0.3”. Because the algorithms deriving attributes from the message text regard the underscore in these strings as a token separator, spammers do not have the possibility to include such attributes in the body. If the average color discreteness is greater than 0.5, there are 3 more thresholds: 0.7, 0.9 and 1.2. Whenever one of them is exceeded, another string is added to the returned array. The value 0.5 is predefined as the threshold which separates graphical

images from photographs. However, the filter can adjust to other thresholds, i.e. if 0.8 is a more reasonable value, only the attributes for above 0.9 and 1.2 will achieve a spam probability low enough to influence the filter.

6 Testing

6.1 Testing Method

This chapter describes the testing of the spam filter implemented according to chapter 5. How the attributes are extracted from emails was already described in 5.3.2. Additionally to the basic version of the OneGramIndexer, modified ones have been implemented that do not stem the words, also include HTML tags and combinations of both. These additional versions are completed by an implementation of Paul Graham's extraction algorithm.

6.1.1 Testing Corpus

As explained in chapter 3, there are test corpora available on the Internet. The problem of the Ling-Spam corpus is that it is composed of spam emails received by the author on the one hand and legitimate emails from an email list on linguistics on the other hand. It is not very likely that emails from a specialized email list can embody real email traffic. The topics in such a mailing list are very likely to be quite similar. Another email corpus is called the PU1 corpus. As the Ling-Spam corpus it was published by the research group around Ion Androutsopoulos [AKCS00]. In this corpus every word is replaced by a number. All header fields, attachments and HTML tags were removed. This corpus is not useful for the tests that are made in this chapter because stemming is impossible as well as the use of HTML tags or image information.

The general problem with publishing testing corpora is that no one wants to make private emails available to everyone. This privacy problem cannot be handled since every authentic email corpus needs real, private messages. All other approaches will only be approximations of real email traffic.

This is the reason why the testing corpus used for this evaluation consists of 889 spam emails and 439 legitimate emails received by the author. This corpus is also problematic because it consists of messages in German and in English language. The major part of the legitimate emails is German but most spam emails are in English. This is a typical situation for email recipients from a non-English language area. Since spam emails are usually delivered to worldwide email addresses, everyone tends to receive spam in English. The problem is that English legitimate emails that arrive at times might be filtered. This behavior should be analyzed in this chapter, too.

For the reason above, the Ling-Spam corpus will despite all concerns be used for alternative testing.

6.1.2 Test Procedure

As mentioned above, alternate versions of OneGramIndexer have been implemented which use variations of the class Cleaner. The basic Cleaner retrieves attributes not regarding HTML tags, stems all words and uses every non-alphanumeric character as a separator (from now on it will be called "standard"). The first tested alternative is a cleaner that does not stem words ("no stem"). The second one uses HTML tags, too ("HTML") and the third one combines these two ("no stem & HTML"). The last filter used is the implementation that Paul Graham described in "A Plan for Spam" [Gra02a] ("Graham"). It does not replace umlaut substitutes (e.g. "ä" with ä), it uses HTML tags and it does not stem the words. Word separators are all characters but alphanumeric ones, dashes, apostrophes and dollar signs.

The testing is done with the filter thresholds 0.9, 0.99 and 0.999 using ten-fold cross-validation as described in 2.2. These tests are performed with and without using image attributes.

After this first series of tests, the corpus is reduced by half to see how the filter configurations react on a smaller training corpus. All tests will be done again with these emails.

As mentioned above, another test-series will be performed using the Ling-Spam corpus. Since these emails do neither contain images nor HTML, the influence of using HTML and image attributes cannot be examined.

6.2 Results

6.2.1 Tests Using the Author's Email Corpus

Tests Using the whole Corpus

Table 6.1 shows the number of false-positives and false-negatives that occurred in the test runs. It is astonishing that a large proportion of all errors are false-positives although the filter is biased towards accepting spam emails. This behavior is independent of the derived attributes and the threshold t. For the standard attribute selection process, the false-positive rate is either 1.367% or 1.39% while the false-negative rate is only 0.787% or 0.9% depending on the threshold. This means that even for a threshold of 0.999 the false-positive rate exceeds the false-negative rate. The figures of the filter using Paul Graham's attribute selection algorithm are even worse. Even with a threshold of 0.999 it only achieves a false-positive rate of 2.278% with a false-negative rate of

t	standard		no stem		HTML		no stem & HTML		based on Graham	
	$N_{L\to S}$	$N_{S\to L}$	$N_{L\to S}$	$N_{S\to L}$	$N_{L\to S}$	$N_{S\to L}$	$N_{L\to S}$	$N_{S\to L}$	$N_{L\to S}$	$N_{S\to L}$
0.9	6	7	3	7	5	5	5	5	13	4
0.99	5	8	3	8	3	5	4	5	10	4
0.999	5	8	3	8	3	6	4	5	10	5

Table 6.1: False-positives and false-negatives for ten-fold cross-validation tests with 889 spam and 439 legitimate emails without using image attributes

0.562%. This is clearly not acceptable. The algorithm using no stemming is the only filter that leads to a bias towards false-negatives (for 0.9: 0.683% and 0.787%).

This needs further explanation. To find out why the filter behaves like this, the emails that are incorrectly categorized have to be examined. The overall number of legitimate emails that are filtered in any test run is 13. Eight of these emails are written in English and four in English and German. Only one email is in German. This email and one of the mixed-language emails contains an HTML table. These tables are characteristic for spam emails, most emails that consist of images are arranged by an HTML table. The German email was only filtered by the 0.9 version of the filter not using stemming and by Graham's filter. The other email containing a table is filtered by all approaches using HTML. Two of the other emails written in English as well as in German are forwarded messages warning against computer viruses and the last one is an email containing tips on composing texts with Latex. These emails are filtered because they are not very personal. The only filter recognizing all of these emails as legitimate is the one using HTML attributes with a threshold of at least 0.99. Two of the English false-positives are also impersonal emails from a mailing list dealing with spam filters. Why just these two emails from this list are filtered is not clear because several other emails from this list are categorized as legitimate.

However, the six remaining false-positives are emails written in English by friends of the author. One of them, that was composed to a large group of people, was filtered in each test run. The others are only categorized as spam by the filter using Paul Graham's approach for attribute selection and in one case by the standard filter.

The analysis of the emails that are false-positives shows that the difficult part of filtering these emails is to recognize those legitimate emails that are English and impersonal. The problem is that many English words have only occurred in spam emails. As an example we assume that the word "money" occurred 40 times in spam emails and only one time in a legitimate email. If we assume that 800 spam emails and 395 legitimate emails have been used to train the filter (90% of each group), the resulting probability for this attribute is:

$$P(\text{spam}|\text{“money”}) = \frac{\frac{40}{800}}{\frac{2 \cdot 1}{395} + \frac{40}{800}} = 90.8\%$$

The fact that was not taken into account is that most legitimate emails that are composed in German are unlikely to include an English term like “money”. If we assume that only 10% of the legitimate emails and 90% of spam emails are English, the spam probability for this attribute is:

$$P(\text{spam}|\text{“money”}) = \frac{\frac{40}{720}}{\frac{2 \cdot 1}{40} + \frac{40}{720}} = 52.63\%$$

This shows that an attribute that is neither characteristic for spam emails nor for legitimate emails can be biased by language tendencies. Despite this fact, the majority of English legitimates is not filtered. The bias because of different ratios of languages in emails has to be taken into account in the future. Spam filters need further improvement and new techniques to work reliable in such an environment. This topic will be further discussed in chapter 7.

Only one spam email that is missed by the filters is in English. This shows that the language problem had also a bad impact on finding spam. However, the false negative rate does not exceed 0.9% which is a rather good value. The reason for this might be that German and English emails have certain attributes in common. A possible explanation are header attributes like mail servers or HTML attributes. The use of HTML seems to help recognizing spam emails. Those test runs using HTML achieve false-negative rates between 0.45% and 0.675% while the others had rates between 0.787% and 0.9%.

λ	**standard**			**Graham**		
	0.9	**0.99**	**0.999**	**0.9**	**0.99**	**0.999**
9	1.26	1.10	1.10	2.50	1.94	1.96
99	1.36	1.13	1.13	2.91	2.24	2.24
999	1.37	1.14	1.14	2.96	2.27	2.27

Table 6.2: Weighted error rates (in percent) for ten-fold cross-validation tests with 889 spam and 439 legitimate emails without using image attributes (A)

Table 6.2 shows a comparison of the weighted error rates of the standard Cleaner and the Cleaner based on Paul Graham’s approach. To make the values easier to compare, they are multiplied with 100; the table shows percentages. It is obvious that the standard Collector outperforms the other one. All of its weighted error rates are better than those achieved with Paul Graham’s approach.

In table 6.3 the weighted error rates of the three other approaches are shown. These values are even better than those achieved by the standard filter. If 0.9 is used as threshold, the test run not using stemming is the best. However,

λ	no stem			HTML			no stem & HTML		
	0.9	**0.99**	**0.999**	**0.9**	**0.99**	**0.999**	**0.9**	**0.99**	**0.999**
9	0.70	0.72	0.72	1.03	0.66	0.68	1.03	0.85	0.85
99	0.69	0.69	0.69	1.13	0.68	0.68	1.13	0.90	0.90
999	0.68	0.68	0.69	1.14	0.68	0.68	1.14	0.91	0.91

Table 6.3: Weighted error rates (in percent) for ten-fold cross-validation tests with 889 spam and 439 legitimate emails without using image attributes (B)

when 0.99 and 0.999 are used, the filter using stemming and HTML attributes outperforms all others. The next section will analyze if stemming has a more positive influence if the training corpus is smaller.

The best threshold seems to be 0.99 for each value of λ and approach used for attribute selection.

Tests Using Half of the Corpus

As mentioned above, the next step is to reduce the size of both corpora. The spam corpus is reduced to 445 emails and the legitimate corpus to 220. The results achieved during this test are not directly comparable to those above because they might be influenced by a different segmentation of the emails or by differences of the corpora resulting from the selection process. What we can analyze with this second set of test runs is how fast the filter can adjust to find spam emails using the different attribute selection processes.

t	**standard**		**no stem**		**HTML**		**no stem & HTML**		**based on Graham**	
	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$	$\mathbf{N_{L\to S}}$	$\mathbf{N_{S\to L}}$
0.9	3	2	2	2	3	0	5	1	6	1
0.99	2	2	1	2	3	0	2	1	4	1
0.999	2	2	1	2	3	1	2	1	4	1

Table 6.4: False-positives and false-negatives for ten-fold cross-validation tests with 445 spam and 220 legitimate emails without using image attributes

Table 6.4 shows the testing results. These numbers show that the difference between false-positives and false-negatives gets even worse. The false-positive rates are between 0.455% and 2.961% while the false-negative rates are between zero and 0.449%. Most of the emails that are classified falsely were also misclassified during the first testing series.

The tables 6.5 and 6.6 show the weighted error rates calculated from the figures above. If we compare it to the results from the previous test series, we can

λ	standard			Graham		
	0.9	**0.99**	**0.999**	**0.9**	**0.99**	**0.999**
9	1.20	0.83	0.82	2.27	1.53	1.53
99	1.35	0.90	0.90	2.68	1.79	1.79
999	1.36	0.91	0.91	2.72	1.81	1.81

Table 6.5: Weighted error rates (in percent) for ten-fold cross-validation tests with 445 spam and 220 legitimate emails without using image attributes (A)

λ	no stem			HTML			no stem & HTML		
	0.9	**0.99**	**0.999**	**0.9**	**0.99**	**0.999**	**0.9**	**0.99**	**0.999**
9	0.82	0.45	0.45	1.11	1.11	1.15	1.90	0.78	0.78
99	0.90	0.45	0.45	1.34	1.34	1.34	2.23	0.90	0.90
999	0.91	0.45	0.45	1.36	1.36	1.36	2.27	0.91	0.91

Table 6.6: Weighted error rates (in percent) for ten-fold cross-validation tests with 445 spam and 220 legitimate emails without using image attributes (B)

see that the advantage of using a threshold of at least 0.99 has even increased. The other difference is quite unexpected: The filter not using stemming has the best performance. This is astonishing because stemming is performed in order to get better result with a smaller training corpus. The stemming should lead to less attributes that are occurring more frequently. However, this result is not really new. Androutsopoulos [AKC+00] came to the same conclusion as described in 3.1.4.

The problem about using stemming on the attributes derived from emails in this corpus is that this algorithm is designed for the English language. Although the algorithm might successfully stem certain German words, most words will not be stemmed properly.

Image Attributes

Both test series are repeated using image attributes. But neither using them in the full corpus nor in the smaller one leads to any changes. Table 6.7 shows the probabilities achieved for the image features after scanning the whole corpora. The problem is that only nine legitimate emails in the whole corpus contain images and none of these is misclassified. For five of these, the average discreteness is less than 0.5. The reason for this is that some of the emails contain scanned articles and images that only consist of gray shades and have a low color discreteness. The number of spam emails containing images is 307. This leads to a similar effect as in the case of the different languages in both categories. Of course each attribute is dominated by spam emails.

The consequence are high probabilities even for the attributes associated with discreteness values greater than 0.5.

attribute	P (full corpus)	P (small corpus)
dis_sm_0.1	40.00%	40.00%
dis_sm_0.3	49.43%	38.55%
dis_sm_0.5	80.88%	68.88%
dis_gr_0.5	92.19%	99.00%
dis_gr_0.7	84.10%	99.00%
dis_gr_0.9	75.04%	99.00%
dis_gr_1.2	40.00%	99.00%

Table 6.7: Probabilities for image features

6.3 Tests using the Ling-Spam Corpus

Because the author's corpus leads to the language problems mentioned above, another series of test runs is performed using the Ling-Spam corpus published by the research group of Ion Androutsopoulos. The corpus consists of 2412 legitimate and 481 spam emails [SAP+01]. As mentioned before, this corpus is problematic, too. As it does not contain any HTML messages or message parts, the only variants tested are the standard version, the version without stemming and the one following Paul Graham's approach.

t	standard		no stem		Graham	
	$N_{L \to S}$	$N_{S \to L}$	$N_{L \to S}$	$N_{S \to L}$	$N_{L \to S}$	$N_{S \to L}$
0.9	0	163	0	158	0	146
0.7	0	149	0	146	0	132
0.5	0	140	0	137	0	130
0.3	0	133	1	130	0	128
0.1	1	118	1	126	1	124
0.05	3	118	3	124	1	123

Table 6.8: False-positives and false-negatives for testing with Ling-Spam

Table 6.8 shows the testing results. First of all it is remarkable that other thresholds were used. The first tests were made with the threshold 0.9 and since no false-positives occurred it made no sense to increase this value. The threshold was continuously decreased until at 0.1 and 0.05 the first false-positives occurred.

The fact that such a low threshold could be chosen for this corpus needs further explanation. The reason for this is that the legitimate emails in this corpus are, as mentioned above, very homogeneous. They are taken from a mailing list which discusses linguistic topics. Since the legitimate emails are homogeneous, they will achieve very low spam probabilities; the threshold can be decreased.

Another question that arises is why the false-negative rate is so much higher than in the tests before. The lowest false-negative rate that occurred without causing any false-positives is 26.61%. One reason for this is that many attributes that were derived in the previous test series are not present. The corpus does not contain any header information except the subject line and no HTML tags are included. Another reason why the false-negative rate exceeds the one achieved using the author's email corpus is that all spam and legitimate emails are in English language.

t	standard			no stem			Graham		
	9	**99**	**999**	**9**	**99**	**999**	**9**	**99**	**999**
0.9	0.735	0.068	0.007	0.712	0.066	0.007	0.658	0.061	0.006
0.7	0.672	0.062	0.006	0.658	0.061	0.006	0.595	0.055	0.005
0.5	0.631	0.059	0.006	0.617	0.057	0.006	0.586	0.054	0.005
0.3	0.599	0.056	0.006	0.626	0.096	0.047	0.577	0.053	0.005
0.1	0.572	0.091	0.046	0.608	0.094	0.047	0.599	0.093	0.047
0.05	0.653	0.173	0.129	0.681	0.176	0.129	0.595	0.093	0.047

Table 6.9: Weighted error rates (in percent) for testing with Ling-Spam

Table 6.9 shows the weighted error rates calculated from the data above. They are also multiplied with 100 in order to make them easier to compare. In this case, the results achieved with Paul Graham's attribute selection process dominate the other ones. At first glance this might be astonishing because it was outperformed in the other test series. But the reason why it is more accurate is that this email corpus does not tend to produce false-positives. If we take another look at tables 6.1 and 6.4, we notice that this approach is in both cases among the best at recognizing spam emails. The problem is however, that this filter configuration produces too many false-positives. When Ling-Spam is used for testing, this behavior does not occur.

6.4 Summarizing Analysis

The testing results do not allow an undisputed decision how to configure the spam filter. Especially selecting the threshold t is complicated. On the one hand, according to the tests using the author's email corpus, a reasonable value would be 0.99. On the other hand the Ling-Spam tests suggest 0.3 as threshold. However, this value appears to be quite low. The answer to this problem might be that there is no ideal threshold, it always depends on the user's emails. A good default value might be 0.9, as used by Paul Graham [Gra02a]. Another approach is letting the user decide. If the number of false-positives is too high, the user might decide to increase the threshold. This process could also be automatized. The spam filter could count the number of false-positives and false-negatives that occurred for a user and adjust the threshold according to the ratio of these values.

The selection of the best method for deriving features is complicated, too. All approaches lead to quite similar figures in the tests with the Ling-Spam corpus. Based on the first tests, the favored method is using the standard attribute deriving algorithm without stemming. Using HTML attributes only led to improvements for a large number of training messages. If the use of the filter is planned for a long period, the use of HTML is recommended.

λ	***ImapProxy***	**(Pant)**	**(Micr)**	**(Andr)**	**(Grah)**
9	*0.572%*	2.198%	1.622%	0.456%	0.011%
99	*0.053%*	1.742%	0.681%	0.073%	0.001%
999	*0.005%*	1.700%	0.588%	0.007%	0.0001%

Table 6.10: Weighted error rates for testing with the Ling-Spam corpus compared to existing Bayesian approaches

λ	***ImapProxy***	**GA (Kati)**	**ANN (Drew)**	**MBL (Sakk)**
9	*0.572%*	6.0%	1.009%	0.587%
99	*0.053%*	3.6%	1.062%	0.064%
999	*0.005%*	3.36%	1.067%	0.006%

Table 6.11: Weighted error rates for testing with the Ling-Spam corpus compared to existing other approaches

Tables 6.10 and 6.11 compare the results of this filter when using Ling-Spam to the existing approaches described in chapter 3. Depending on the value of λ, the filter is only outperformed by Paul Graham's approach which was evaluated using a much bigger corpus. The two other approaches using Ling-Spam for testing (explained in 3.1.4 and 3.4.2) achieve slightly worse results, only for $\lambda = 9$ the Bayesian filter by Ion Androutsopoulos' group is better.

7 Conclusion and Outlook

The result of this study is a spam filter that can be adjusted to the use in several environments; on a client computer for one user as well as on a server for all members of an organization. The design is modular, the major classes used for filtering are based on abstract classes. This means that the software can serve as a platform for the implementation of other approaches.

The task of filtering spam is always a struggle against spammers. They adjust to spam filters just as these filters should adjust to their emails. John Graham-Cumming, a researcher on the field of spam filters, held two talks about how spammers avoid filters on the Spam Conference 2003 and the Spam Conference 2004. He also maintains "The Spammers' Compendium" [GC04], a collection of such tricks.

These tricks include several ways of including invisible bogus texts like a piece of current news in emails. For example, such a text can be written in a very small font size or it can be hidden in a MIME part. These texts usually contain words that are not very likely to appear in spam emails. This increases the probability that the spam email is not filtered. As an additional side effect those attributes will receive a higher spam probability when the user moves the email to the spam folder. This raises the chances of legitimate messages which contain them being falsely identified as spam. This process is called poisoning. Filters must be somehow modified to recognize whether a text is visible for the user or not.

Another popular trick is to divide words into letters. This can be done by leaving blanks between each letter of a word or by including fake HTML commands. Invisible HTML tags might be a reason why using HTML attributes is not definitely an advantage. On the one hand they lead to many significant attributes like links, but on the other hand they can also be used by spammers as a trick. The only possible way out of this dilemma is to analyze HTML commands and identify fakes.

If a spam filter is adapted to these techniques used by spammers, the process that analyzes the emails will be close to a renderer. All non-visible attributes are not regarded. This leads to valuable attributes obtained from HTML tags and can lead to a new attribute indicating if the email contains invisible text or HTML commands. This attribute would be a very powerful tool against many strategies used by spammers.

Another problem that has to be handled but is never discussed is the language problem explained in 6.1.2. A possible way out of this situation is to keep

separate probability databases for each language. An incoming email must first be examined to find out in which language it is composed. Then it can be analyzed using the appropriate database. This leads to more dependable probabilities. But the approach causes a longer learning process because two filters have to be trained.

All techniques that lead to better results by solving some of these problems can be implemented as additional classes inheriting from the class SpamFilter or FeatureCollector.

A Variables

Total number of emails	N
Number of spam emails	N_S
Number of legitimate emails	N_L
Attributes	$\mathcal{A} = \{a_1, \ldots, a_n\}$
Number of unique attributes	n
Number of unique attributes in spam emails	n_S
Number of unique attributes in legitimate emails	n_L
Number of occurrences of attribute a_i in emails	$n(a_i)$
Number of occurrences of attribute a_i in spam emails	$n_S(a_i)$
Number of occurrences of attribute a_i in legitimate emails	$n_L(a_i)$
Number of spam emails categorized as spam	$N_{S\to S}$
Number of spam emails categorized as legitimate	$N_{S\to L}$
Number of legitimate emails categorized as legitimate	$N_{L\to L}$
Number of legitimate emails categorized as spam	$N_{L\to S}$
False-negative rate	$r_{\text{fn}} = \frac{N_{S\to L}}{N_S}$
False-positive rate	$r_{\text{fp}} = \frac{N_{L\to S}}{N_L}$
Weighted error rate	W_{Err}
Variable indicating whether email j is spam $C_j = \begin{cases} 1 & \text{if email } j \text{ is spam} \\ 0 & \text{otherwise} \end{cases}$	C_j
Variable for occurrence of attribute a_i in email j $O_j(a_i) = \begin{cases} 1 & \text{if attribute } a_i \text{ occurs in email } j \\ 0 & \text{otherwise} \end{cases}$	$O_j(a_i)$

Bibliography

[AKC+00] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras, and Constantine D. Spyropoulos. An Evaluation of Naive Bayesian Anti-Spam Filtering. In G. Potamias, V. Moustakis, and M. van Someren, editors, *Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML'00)*, pages 9–17, Barcelona, Spain, 2000.

[AKCS00] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, and Constantine D. Spyropoulos. An Experimental Comparison of Naive Bayesian and Keyword-Based Anti-Spam Filtering with Encrypted Personal E-mail Messages. In N.J. Belkin, P. Ingwersen, and M.-K. Leong, editors, *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'00)*, pages 160–167, Athens, Greece, 2000.

[APK+00] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach. In H. Zaragoza, P. Gallinari, and M. Rajman, editors, *Proceedings of the Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00)*, pages 1–13, Lyon, France, 2000.

[Bis95] Christoph M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[bri04] Brightmail - Spam Percentages and Spam Categories. `http://www.brightmail.com/spamstats.html`, January 2004.

[CL98] Lorrie Faith Cranor and Brian A. LaMacchia. Spam! *Communications of the ACM*, 41(8):74–83, August 1998.

[Cri03] M. Crispin. Internet Message Access Protocol - Version 4rev1. `http://rfc.net/rfc3501.html`, March 2003.

[Dar59] Charles Darwin. *On The Origin Of Species By Means Of Natural Selection, or, The Preservation Of Favoured Races In The Struggle For Life*. John Murray, 1859.

[dcc] Distributed Checksum Clearinghouse. http://www.rhyolite.com/anti-spam/dcc/.

[Dre02] Rich Drewes. An Artifical Neural Network Spam Classifier. http://www.interstice.com/drewes/cs676/spam-nn/spam-nn.html, May 2002.

[ema04] email. http://www.webopedia.com/TERM/E/e_mail.html, January 2004.

[FB96] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. http://www.rfc.net/rfc2045.html, November 1996.

[Fra92] W. Frakes. *Stemming Algorithms*. Englewood Cliffs, 1992.

[GC04] John Graham-Cumming. John Graham-Cumming: The Spammers' Compendium. http://www.jgc.org/tsc/, January 2004.

[gmx] GMX. http://www.gmx.net.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[Gra02a] Paul Graham. A Plan for Spam. http://www.paulgraham.com/spam.html, August 2002.

[Gra02b] Paul Graham. Filters vs. Blacklists. http://www.paulgraham.com/falsepositives.html, September 2002.

[Gra02c] Paul Graham. Spam is Different. http://www.paulgraham.com/spamdiff.html, August 2002.

[Gra02d] Paul Graham. Will Filters Kill Spam? http://www.paulgraham.com/wfks.html, December 2002.

[Gra03] Paul Graham. Better Bayesian Filtering. http://www.paulgraham.com/better.html, January 2003.

[hei02] Anti-Spam-Dienst ORBZ vom Netz. http://www.heise.de/newsticker/data/uma-20.03.02-000/, March 2002.

[hei03a] GMX landete auf Open-Relay-Blacklist. http://www.heise.de/newsticker/data/hob-27.05.03-000/, May 2003.

[hei03b] Spamcop nimmt GMX von der schwarzen Liste. http://www.heise.de/newsticker/data/uma-12.09.03-000/, September 2003.

[jav] JavaMail API. http://java.sun.com/products/javamail/.

[Kat99] Hooman Katirai. Filtering Junk E-Mail: A Performance Comparison between Genetic Programming & Naïve Bayes. http://members.rogers.com/hoomank/papers/katirai99filtering.pdf, September 1999.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[ord] Open relay database. `http://ordb.org/`.

[PCH+02] S. Prabhakar, H. Cheng, J.C. Handley, Z. Fan, and Y.W. Lin. Picture-Graphics Color Image Classification. In *Procedings of the International Conference on Image Processing (ICIP'02)*, volume 2, pages 785–788, Rochester, New York, September 2002.

[PL98] Patrick Pantel and Dekang Lin. SpamCop: A Spam Classification & Organization Program. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[Por80] M.F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, July 1980.

[rbl] Maps realtime blackhole list. `http://mail-abuse.org/rbl/`.

[SAP+01] Georgios Sakkis, Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Constantine Spyropoulos, and Panagiotis Stamatopoulos. A Memory-Based Approach to Anti-Spam Filtering. Technical report, National Center for Scientific Research (NCSR) "Demokritos", 2001.

[SDHH98] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[Ser03] Matt Sergeant. Internet Level Spam Detection and SpamAssassin 2.50. `http://axkit.org/docs/presentations/spam/SpamConf2003.pdf`, January 2003.

[spaa] `http://www.spamconference.org/index2003.html`.

[spab] Spamassassin. `http://www.spamassassin.org`.

[spac] Spamcop. `http://www.spamcop.net`.

[spa04] What is "spam"? `http://mail-abuse.org/standard.html`, January 2004.