# Learning to Recognize Missing E-mail Attachments

**Technical Report TUD–KE–2009-05**
Marco Ghiglieri, Johannes Fürnkranz
Knowledge Engineering Group, Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Knowledge
Engineering

**Abstract**

Forgotten attachments of e-mail message are a common and obnoxious problem. Several E-mail readers provide plugins that attempt to tackle this problem by trying to guess whether a message needs an attachment and warn the user in case s/he does not attach a file to such a message. However, these approaches essentially only work with a fixed list of keywords, which trigger such a warning whenever they occur in a message. In this paper, we try conventional machine learning techniques, which have been previously shown to work well for related problems such as spam mail filtering, on this new problem. Our results show that they work very well, clearly outperforming simple keyword-based approaches. The software is available as plugin for the Thunderbird e-mail reader.

## 1 Introduction

E-mail messages pose numerous challenges for machine learning and text classification methods (Brutlag and Meek, 2000). They are rather short, quite heterogeneous, and originate from multiple authors. One of the earliest works on classifying E-mail messages was the approach by (Cohen, 1996), who showed that a rule learner can learn to categorize E-mail messages rather well. Similar ideas have subsequently been used in several E-mail assistants (Payne and Edwards, 1997; Rennie, 2000; Crawford et al., 2002). Interest in machine learning methods for E-mail classification grew considerably when it was recognized that they may form an adequate tool for fighting spam mails (Goodman et al., 2005; Fawcett, 2003). One of the widest used algorithms in this domain is a probabilistic algorithm introduced by Graham; 2003), which forms the basis of the trainable spam filter used in Mozilla web browsers, or is used as a component in the SpamAssassin spam filtering package.

Another obnoxious problem in E-mail processing is that users frequently forget to attach documents to their message although they actually had intended to do so. Several E-mail readers provide plugins that attempt to warn the user in cases where it seems apparent that s/he forgot to attach a file. Such plugins include Forgotten Attachment Detector[1][2] and Attachment Reminder for Gmail, Attachment Reminder[3] and Check and Send[4] for Thunderbird, or Forgotten Attachment Detector[5] for MS Outlook. All these plugins work in a similar way. Essentially, they all maintain a configurable list of keywords (such as *attach, attachment, file, ...*), and, after the user pressed "*Send*", check each outgoing message for the appearance of one of these keywords in the message. If one of the keywords appears in a message and the message does not have an attachment, a pop-up window warns the user that an attachment might be missing.

We envision a solution to this problem via machine learning techniques, which can be trained on a corpus of a user's E-mail messages. The training information is the text of the messages, and the training signal is whether the message had an attachment or not. The trained classifier will then emit a warning to the user whenever it thinks that the text of the message indicates that the user had intended to attach a document, but failed to so.

The key advantage of a machine-learning solution is, of course, that the system can adapt to different users. In fact, our results show that our best solution, a combination between Naïve Bayes and Graham's algorithm, achieves a uniformly strong performance across five different users. A possible disadvantage is that its performance in the early training phases may be inferior to a conventional keyword-based approach. Although we did not investigate this more closely, we just note that this problem could be alleviated either by a good general initialization of the classifier (e.g., one trained on several users) or with a gradual shift from a keyword-based classifier to the trainable system.

Automated recognition of missing attachments is an interesting challenge, because the above-mentioned methods all focus on classifying the contents of an E-mail into a set of topics (or E-mail folders). Even spam mail filtering works that way, because the topic of spam E-mails is typically considerably different from the topics of a user's regular E-mails. Messages with and without attachments can be found in practically all possible topics, hence the vocabularies in the attachment and no-attachment classes will be much more similar to each other than for content-based e-mail classification problems.

The result of this work is a plugin for the Thunderbird E-mail classifier that can be trained to recognize E-mail messages with missing attachments. The latest version of the plugin can be downloaded from `http://www.attachmentchecker.de`. In this paper, we describe this system and evaluate it on several corpora, also in comparison with a simple keyword-based approach such as those mentioned above. We will start with a brief formal description of the problem (Section 2) and then proceed to describe the four algorithms that we implemented in detail (Section 3). Sections 4 and 5 then describe our experiments and their results, before we conclude (Section 6).

[1] `http://mail.google.com/support/bin/answer.py?hl=en&answer=143085`

[2] `https://addons.mozilla.org/de/thunderbird/addon/5759`

[3] `https://addons.mozilla.org/de/thunderbird/addon/5759`

[4] `https://addons.mozilla.org/de/thunderbird/addon/2281`

[5] `http://www.officelabs.com/projects/forgottenattachmentdetector/`

## 2 Problem Description

Our application scenario is the following: Every time the user sends a message, the message is not immediately sent to the mail server, but first passed to a learning system (realized with a separate server process). If the message contains an attachment, the learner processes it as a new training example and then sends it off to the mail server right away. If the message does not contain an attachment, it is first passed to the classifier to ask whether it should have an attachment. If the classifier decides "no", the message is again used for training and then sent to the mail server, if the classifier says "yes", a pop-up window is generated and asks the user whether s/he would like to add an attachment or not. The message together with the user's final decision is then used for training and sent to the mail server.

Thus, from the point of view of the learner, we assume an incoming stream of E-mail messages $M_i$, which have to be classified into the attachment-class $a$ and the class $r$ of regular e-mails. After each message $M$ has been processed, the learner receives the correct label $c(M)$ for the message. Strictly speaking, in the case when the e-mail is sent off by the user without an attachment and the classifier agrees that it should not contain a message, its label may not be correct because both the user and the classifier might have overlooked that the message should contain an attachment. We currently simply ignore this problem.

At any time $t$, the learner thus has a pre-classified set of training examples $D_i = \{M_1, \ldots M_{i-1}\}$. The task of the learner is to automatically derive a classifier from the dataset $D_i$, which is able to predict a class $\tilde{c}(M)$ for a new message $M$. The learning problem is *incremental*, i.e., the classifier can be adapted after a new message has arrived instead of having to re-train the classifier after every change in the dataset $D_i$.

In our evaluation, we will also work with static datasets, in which case we simply omit the subscript $i$. Each dataset $D$ can be divided into two sets $D = D_a \cup D_r$, where $D_a$ is the set of messages that contain an attachment and $D_r$ is the set of regular messages, i.e., messages that do not contain an attachment. Each of these sets may also be considered as a large meta-document, i.e., $A = \bigcup_{M \in D_a} M$ is the hypothetical message that consists of the concatenation of all e-mail messages with an attachment, and $R = \bigcup_{M \in D_r} M$ is the concatenation of all messages without an attachment. The number of times a given word $t$ occurs in these meta-documents is $n_{t,A}$ (the number of times that the word $t$ occurs in $A$) and $n_{t,R}$ (the number of times that the word $t$ occurs in $R$). We defined a word $t$ simply as any consecutive character strings which entirely consist of letters and hyphens.

As all our learners essentially only use the frequency counts $n_{t,A}$ and $n_{t,R}$ as the basic information for deriving their predictions, their implementation in an on-line learning scenario, where we have a continuous incoming stream of messages, was quite straight-forward. Essentially, one only has to consider that the meta documents $A$ and $R$ are continuously growing and their associated term counts have to be adapted on-line.

## 3 Algorithms

In this section, we briefly recapitulate the learning algorithms with which experimented. We start with a simple keyword matcher, which is used in similar form in most publicly available attachment checking plugins (Section 3.1), and also discuss a simple method for automatically generating a list of keywords. We then proceed with the three learning methods, namely a Naïve Bayes classifier (Section 3.2), Paul Graham's spam mail filtering algorithm (Section 3.3), and a straight-forward combination of both (Section 3.4).

### 3.1 Keyword Matching

A keyword matcher maintains a list of keywords $K$, which are indicative of an E-mail that requires an attachment. The algorithm then simply walks through all words $t$ in the e-mail message $M$, and whenever one of the words also occurs in $K$, the keyword matcher predicts class $a$. If none of the words of the message occur in $K$, it predicts class $n$.

$$\tilde{c}(M) = \begin{cases} a & \text{if } \exists t \in M : t \in K \\ r & \text{if } \forall t \in M : t \notin K \end{cases} \tag{3.1}$$

We worked with two versions of this simple algorithm, which use different keyword lists $K$, one that uses a fixed, user-defined keyword list, and one that is able to automatically learn its keywords from messages with known classifications:

#### Static Keywords

This approach uses a fixed, immutable list of keywords. As we were working with German-text e-mails, we used the following keyword list

$$K = \{\text{'anhang', 'anhänge', 'datei', 'dateien', 'fotos', 'version', 'versionen', 'entwurf', 'entwürfe', 'anbei'.}\} \tag{3.2}$$

#### Dynamic Keywords

This approach adapts the list of keywords to the corpus at hand. The key idea is that a word is promoted to a keyword for the attachment class if it occurs $k$ times as frequently in class $a$ and than in class $r$, where $k$ is a user-settable parameter.

More formally, this approach tries to estimate the probabilities $P(t \mid a)$ and $P(t \mid r)$ that the word $t$ occurs in class $a$ and $r$ respectively. The values $P(t \mid a)$ and $P(t \mid r)$ can be estimated with their relative frequencies in the meta documents $A$ and $R$, i.e.,

$$P(t|a) = \frac{n_{t,A}}{|A|}, \qquad P(t|r) = \frac{n_{t,R}}{|R|}. \tag{3.3}$$

Given these estimates, the keyword list is defined as

$$K = \left\{ t \in A : \frac{P(t|a)}{P(t|r)} > k \right\} \tag{3.4}$$

The keyword list can be estimated in batch (from a given training corpus) or dynamically adapted with the stream of incoming E-mails.

### 3.2 Naïve Bayes

The Naïve Bayes classifier is known for its simplicity and computational efficiency. It also can be trained incrementally, and therefore seems to be a very good candidate for solving our problem.

Its key idea is that for an e-mail message $M$, the Naïve Bayes classifier estimates the probabilities $P(a \mid M)$ that the message should contain an attachment and $P(r \mid M) = 1 - P(a \mid M)$ that the message should not contain an attachment. Based on these probabilities, the classifier then predicts one of the two classes. Typically it will pick the one with the

larger estimated probability, i.e., it will predict $a$ if $P(a\,|\,M) > P(r\,|\,M)$ and $r$ otherwise. More generally, one can define a decision threshold $d$, and predict class $a$ if the probability $P(a\,|\,M)$ is at least $d$ times the probability $P(a\,|\,M)$.

$$\tilde{c}(M) = \begin{cases} a & \text{if } \frac{P(a|M)}{P(r|M)} > d \\ r & \text{if } \frac{P(a|M)}{P(r|M)} \leq d \end{cases} \tag{3.5}$$

For estimating the probability $P(a|M)$, the Naïve Bayes classifier uses Bayes' law:

$$P(a|M) = \frac{P(M|a) \cdot P(a)}{P(M)}, \tag{3.6}$$

where

- $P(a)$ is the a priori probability for a message to have an attachment, i.e., the probability that a message has an attachment without considering the message text
- $P(M)$ is the a priori probability for an e-mail text $M$, i.e., the probability, that a given e-mail message text $M$ can be observed independent of whether it has an attachment or not
- $P(M|a)$ is the probability that a given e-mail text $M$ would occur in the class of e-mails with an attachment

The first term, $P(a)$, can be easily estimated from the data with the percentage of messages that contain an attachment. $P(r)$ can be estimated analogously, i.e.,

$$P(a) = \frac{|D_a|}{|D|}, \qquad P(r) = \frac{|D_r|}{|D|} = 1 - P(a) \tag{3.7}$$

The denominator of equation (3.6), $P(M)$, can be considered as a normalizing constant, i.e.,

$$P(M) = P(M|a) \cdot P(a) + P(M|r) \cdot P(r). \tag{3.8}$$

The word-extraction divides e-mails into single words $t$. The assumption that the occurrence of a word in a message only depends on the class of the message (with or without attachment) but not on the presence of other words in the message, allows the following simplification

$$P(M|a) = \prod_{t \in M} P(t|a) \tag{3.9}$$

where $P(t\,|\,a)$ is the probability that a word $t$ occurs in the documents with attachments. $P(M|r)$ is computed analogously. The values $P(t\,|\,a)$ and $P(t\,|\,r)$ are estimated as described in the previous section. Combining all these, yields the following estimate $nb(M)$ for the probability $P(a|M)$ that an e-mail $M$ should have an attachment

$$nb(M) = \frac{P(a) \cdot \sum_{t \in M} P(t\,|\,a)}{P(a) \cdot \sum_{t \in M} P(t\,|\,a) + P(r) \cdot \sum_{t \in M} P(r\,|\,a)} \approx P(a|M) \tag{3.10}$$

It should be noted that the assumption underlying (3.9), the so-called *Naïve Bayes assumption*, is typically not justified. However, it can be argued that while making this assumption will yield bad probability estimates, it will have a much lesser impact on the performance of the classifier that is based on these estimates (Domingos and Pazzani, 1997).

### 3.3 Graham's Algorithm

The algorithm of Graham; 2003) has been specifically developed and tuned for dealing with the problem of spam recognition. Several of its features are specifically targeted towards this problem, such as its focus on using only a few significant words, or the use of different weights for words in spam and regular E-mail messages. The algorithm has been quite successful and forms the basis of many anti-spam packages, such as Mozilla's spam mail filter or the SpamAssassin software.

In our work, we adapted the algorithm for use of recognizing e-mails that require attachments. The key component of the algorithm are two heuristic weights that are assigned to each term in an e-mail message, one for the attachment class and one for the class of regular e-mails

$$w_{t,a} = \min\left(1, c_a \cdot \frac{n_{t,A}}{|D_a|}\right) \qquad w_{t,r} = \min\left(1, c_r \cdot \frac{n_{t,R}}{|D_r|}\right) \tag{3.11}$$

In both cases, we calculate the proportion between the numbers of words in the sets $A$ and $R$, and the numbers of e-mails in the respective classes.[1] For spam mail filtering, the weights $c_a$ and $c_r$ are set according to a ratio of 1:2, i.e., the occurrence of non-spam words are weighted twice as heavily as the occurrence of spam words. For the problem of e-mail attachment recognition, we set both weights $c_a = c_r = 1$.

The two weights (3.11) are then combined into an overall weight $w_t$, which is an indicator how often a word $t$ occurs in a e-mail with respectively in a e-mail without attachment. The higher the value or $w_t$, the larger the probability that a word is an indicator for the attachment-class.

$$w_t = \max\left(NP, \min\left(AP, \frac{w_{t,a}}{w_{t,a} + w_{t,r}}\right)\right). \tag{3.12}$$

The values AP and NP are set by the user. $NP$ specifies the lowest value, which a word can have, if it is in the noattachment-class. Analogous specifies $AP$ the highest value, which a word can have, if it is in the attachment-class. Because that $0 < NP < 1$, $0 < AP < 1$ and $NP < AP$, the value of $w_t$ is between 0 and 1.

Note that if we ignore all the minima and maxima in (3.11) and (3.12), and furthermore assume that the number of e-mail messages with and without attachments are approximately equal, then $w_t$ essentially estimates the probability that a given word occurs rather in class $a$ than in class $r$:

$$P(a \mid t) = \frac{n_{t,A}}{n_{t,A} + n_{t,R}} \approx \frac{\frac{n_{t,A}}{|D_a|}}{\frac{n_{t,A}}{|D_a|} + \frac{n_{t,R}}{|D_r|}}$$

For a given e-mail message $M$, Graham's algorithm selects the 15 words $t_i$ whose weights $w_i$ are most different from 0.5. These are then combined into a score $g(M)$ for the e-mail message $M$

$$g(M) = \frac{\prod_{i=1}^{15} w_i}{\prod_{i=1}^{15} w_i + \prod_{i=1}^{15}(1 - w_i)} \tag{3.13}$$

.

To decide if $M$ should have an attachment the score $g(M)$ is compared with a threshold score $g$.

$$\tilde{c}(M) = \begin{cases} a & \text{if } g(M) > g \\ r & \text{if } g(M) \leq g \end{cases} \tag{3.14}$$

We used much lower thresholds $g$ than typically used for spam recognition, because false positives are not as problematic in the attachment problem as they are in spam recognition.

[1] It may appear a bit strange that the numerator counts words, while the denominator counts documents, but Graham's approach is entirely heuristic.

### 3.4 Combined Algorithm

Naïve Bayes and Graham's algorithm compute different scores for recognizing attachments, as we will also see in the experimental evaluation. Thus, it makes sense to try to combine these two in the hope to get the best of both approaches.

Both algorithms are evaluated independently, and their respective scores $nb(M)$ (3.10) and $g(M)$ (3.13) are computed. Both are in the range $[0,1]$ and can be combined with a simple weighted average

$$s(M) = \frac{v \cdot nb(M) + g(M)}{v+1} \tag{3.15}$$

where $v$ is a factor that indicates the relative importance of the two different methods. Unless otherwise mentioned, we use the value $v = 2$ in our experiments, i.e., the Naive Bayes prediction is considered to be twice as important as the prediction of Graham's algorithm.

Again, a message $M$ is predicted as belonging to class $a$ if the score $s(M)$ exceeds a threshold score $s$.

$$\tilde{c}(M) = \begin{cases} a & \text{if } s(M) > s \\ r & \text{if } s(M) \leq s \end{cases} \tag{3.16}$$

## 4 Experimental Setup

We implemented the methods described in the previous section as a plugin for the Thunderbird mail reader. In order to determine the best performing configuration, we experimentally compared the algorithms on several mail collections. In this section, we describe the experimental setup, the next section will present the results from this evaluation.

### 4.1 Algorithms

We implemented all of the algorithms in the previous section, and also included a simple benchmark algorithms. In the following, we list all six algorithms and also give the parameter settings that were used (unless mentioned otherwise).

Never Attach: the default classifier, which always predicts the majority class, i.e., it always assumes that a messages will not require an attachment.

Static Keywords: the keyword-matching algorithm using the fixed set of keywords (3.2). These keywords were manually picked by the authors.

Dynamic Keywords: the keyword-matching algorithm using a list of keywords that has been automatically generated from the training data. We used the frequency threshold $k = 2$, i.e., keywords must occur at least twice as likely in class $a$ than in class $r$.

Naive Bayes: with a threshold of $d = 0.5$.

Graham's algorithm: with a threshold of $g = 0.51$

Combined algorithm: with value $\nu = 2$ (Naive Bayes twice important as Graham) and a threshold of $s = 0.34$.

### 4.2 Evaluation

We compare the performance of the algorithms using accuracy, precision and recall, which are commonly used for evaluating the performance of information retrieval and text classification algorithms (Manning et al., 2008). These performance measures can be computed from a 2x2 confusion matrix with the following entries:

- *true positive* — e-mail with attachment, which is classified correctly,
- *false positive* — e-mail without attachment, which is misclassified,
- *true negative* — e-mail without attachment, which is classified correctly and
- *false negative* — e-mail with attachment, which is misclassified.

*Accuracy* describes the proportion of correctly classified e-mails in all e-mails:

$$\text{accuracy} = \frac{\text{true positive} + \text{true negative}}{\text{true positive} + \text{true negative} + \text{false negative} + \text{false positive}}.$$

*Precision* is the fraction of correctly recognized e-mails in all e-mails that have been predicted to have an attachment:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}.$$

*Recall* is the fraction of e-mails with attachment that have been correctly predicted:

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}.$$

In order to summarize both measures into a single number, we also show results for the $F$-measure, the harmonic mean of recall and precision.

$$F = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$$

In addition, we also investigated the sensitivity of a method's parameters by trying several values and plotting the results in an ROC curve (Provost and Fawcett, 2001). A ROC curve plots the true positive rate over the false positive rate for various different versions of a classifier. These can be obtained by sorting all predictions according to the predicted score value, and then trying several (or all) possible thresholds to break the list into the classes $a$ and $r$.

We also report the AUC, the area under the ROC curve, which essentially is the probability that an arbitrarily selected message with attachment is ranked before an arbitrarily selected message without attachment. For more details on the use of ROC analysis for evaluating machine learning classifiers we refer to (Fawcett, 2006).

For estimating these values from the training data, we performed a 10-fold cross validation

### 4.3 Datasets

We could not find any suitably publicly available datasets for our experiments. For example, from a big publicly available corpus of E-mails, the Enron corpus (Klimt and Yang, 2004), all attachments had been removed prior to publication. Moreover, private e-mail-user do not want to give away private e-mails. Eventually, we found the following datasets to evaluate the algorithms:

1. dataset with 453 e-mails in German language, of which 74 e-mails have at least one attachment,
2. dataset with 531 e-mails in German language, of which 113 e-mails have at least one attachment,
3. dataset with 2245 e-mails in German language, of which 732 e-mails have at least one attachment.

For these three datasets, the full text of the e-mails was available. Only these datasets were used for full experimentation with the algorithms, i.e., the reported best-performing parameters of the algorithms were only tuned on these datasets.

Two additional datasets — so called *control datasets* — were used for evaluation only:

4. dataset with 282 e-mail in German language, of which 70 e-mail have at least one attachment,
5. dataset with 922 e-mail in German language, of which 96 e-mail have at least one attachment.

For these datasets, we did not have off-line access to the body of E-mails, but the algorithms were evaluated in the user's client browser, and only the results were recorded in order to protect the user's privacy. Thus, we were not able to tune the parameters of the algorithms or optimize their performance in any way, so that the results on these two datasets can be considered as a reasonable estimate of the algorithms performance in a real setting. The only difference to a practical setting is that, for simplicity, the evaluation is still performed via 10-fold cross-validation of the user's existing mail and not in an true on-line setting.

## 5 Results

This sections describes our results. We will first focus on predictive accuracy (Section 5.1), then look at recall and precision (Section 5.2), and finally perform an ROC analysis (Section 5.3).

### 5.1 Accuracy

Table 5.1 shows the results of the algorithms on the first three datasets. Both versions of the keyword matching algorithm are unable to significantly outperform the default *Never attach* strategy. On the first dataset, they even perform even worse, which could indicate that they pick up a systematic bias. However, one should also note this dataset only contains 7–8 documents with attachments in each of the test sets of the 10-fold cross-validation. The dynamic strategy, which automatically chooses its keywords, performs somewhat better than the fixed strategy, but both versions do not provide much leverage to the user.

| Algorithm | Dataset 1 | | Dataset 2 | | Dataset 3 | |
|---|---|---|---|---|---|---|
| *Never Attach* | 83.66% | — | 78.68% | — | 67.32% | — |
| *Static keywords* | 76.38% | (−7.28%) | 79.66% | (+0.98%) | 71.00% | (+3.68%) |
| *Dynamic keywords* (2) | 78.81% | (−4.85%) | 80.04% | (+1.36%) | 69.53% | (+2.21%) |
| *Graham* (0.51) | 87.64% | (+3.98%) | 90.21% | (+11.53%) | 85.84% | (+18.52%) |
| *Naïve Bayes* (0.5) | 87.64% | (+3.98%) | 89.64% | (+10.96%) | 88.95% | (+21.63%) |
| *Combined* (2:1,0.34) | 93.82% | (+10.16%) | 90.77% | (+12.09%) | 88.55% | (+21.23%) |

Table 5.1: Predictive accuracy of the learning algorithms (absolute and in comparison to the *Never Attach* benchmark).

Paul Graham's algorithm and the Naïve Bayes classifier, on the other hand, exhibit a similar performance and are both consistently better than the three simple benchmarks. However, their predictions are still sufficiently diverse so that the combined algorithm is able to outperform both of them, most notably in Dataset 1, where it seems to be able to considerably reduce the variance resulting from the comparably low example numbers. On the third dataset, it is a little bit behind the Naïve Bayes algorithm, but it is obvious that it has been able to correct the comparably bad performance of Graham's algorithm on this dataset.

In comparison to the *Never Attach* strategy, which does not require any learning, but simply predicts the majority class ($r$), the best algorithms achieve an improvement in accuracy of about 20% (numbers shown in brackets). These were obtained on Dataset 3, where the number and fraction of messages with attachments is considerably higher than in the other two datasets, which could indicate that the learners have learned better models from the larger datasets.

| Accuracy | Dataset 4 | | Dataset 5 | |
|---|---|---|---|---|
| *Never attach* | 75.18% | — | 89.59% | — |
| *Static keywords* | 65.96% | (−9.22%) | 82.21% | (−7.38%) |
| *Dynamic keywords* (2) | 68.79% | (−6.39%) | 84.06% | (−5.53%) |
| *Graham* (0.51) | 84.40% | (+9.22%) | 90.67% | (+1.08%) |
| *Naïve Bayes* (0.5) | 78.01% | (+2.83%) | 90.78% | (+1.19%) |
| *Combined* (2:1,0.34) | 95.04% | (+19.86%) | 96.10% | (+6.51%) |

Table 5.2: Predictive Accuracy of the learning algorithms on the control datasets (absolute and in comparison to the *Never Attach* benchmark).

To get independent results the algorithms were tested on the two control datasets. In principle, the trend of the first three datasets is confirmed. The Combined algorithm performs best, followed by Naïve Bayes and Graham's algorithm, and the three simple algorithms perform worst. Somewhat surprising is that on these datasets the trend from Dataset 1, namely that the keyword algorithms perform worse than the *Never Attach* strategy, is strongly confirmed on these datasets. This clearly indicates that something is going wrong, presumably the keywords are too unreliable to discriminate between the two classes and consequently trigger too many alarms.

For all accuracy values, it is not clear how the improvement of the detection was achieved. The increase may be caused by triggering fewer but more reliable alarms, or, conversely, by triggering more alarms at the expense of a few more false alarms. We will investigate this in more detail in the next sections.

## 5.2 Precision/Recall

Because the accuracy results do not allow conclusions about the reasons for the different results of the algorithms, we also analyzed the results with precision and recall in order to get a deeper understanding of the behavior of the algorithms. As the *No Attach* strategy never predicts class $a$, its recall is 0, and its precision is undefined. Hence we do not further consider it in this section. Figure 5.3 shows the results of the remaining five algorithms on the first three datasets.

| | Dataset 1 | | | Dataset 2 | | | Dataset 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| Algorithm | Prec. | Recall | F | Prec. | Recall | F | Prec. | Recall | F |
| *Static Keywords* | 23.81% | 20.27% | 21.90% | 52.29% | 50.44% | 51.35% | 62.54% | 27.60% | 38.30% |
| *Dynamic Keywords* (2) | 78.81% | 21.05% | 33.23% | 52.76% | 59.29% | 55.83% | 53.49% | 50.27% | 51.83% |
| *Graham* (0.51) | 70.45% | 41.89% | 52.54% | 82.80% | 68.14% | 74.76% | 95.00% | 59.70% | 73.23% |
| *Naïve Bayes* (0.5) | 62.50% | 60.81% | 61.64% | 75.00% | 76.99% | 75.98% | 85.28% | 79.92% | 82.51% |
| *Combined* (2:1,0.34) | 76.14% | 90.54% | 82.72% | 76.67% | 81.42% | 78.97% | 88.00% | 75.14% | 81.06% |

Table 5.3: Precision, recall, and F-measure of the learning algorithms

The *Static Keywords* approach shows a rather poor performance, in particular with respect to recall. This shows that the fixed list of keywords does not fire very often and does not classify many e-mails into the attachment class. However, it is also rather poor with respect to precision, which means that even if the keywords trigger an alarm, it is still quite frequently a false alarm. Only on the third dataset one has the impression that the keywords fit o.k., but still rather poor in comparison to the other approaches.

The *Dynamic Keywords* approach shows somewhat better results. Interestingly, there is not clear trend in whether the automated keyword selection tends to improve precision more than recall or the other way around. On the first dataset, it really boosted precision to the best performance of all algorithms, but maintained a very poor recall, i.e., it selected too few keywords, but those were very reliable. On the third dataset, the opposite seemed to happen, namely that the automatic keyword selection greatly improved recall, but at the expense of precision. Again, it might be that the comparably low number of the attachment documents in dataset 1 is responsible for this difference.

*Naïve Bayes* and *Graham's algorithm* perform considerably better, clearly topping the two keyword-based approaches in both recall and precision (with the exception of the precision of the dynamic keywords on dataset 1). However, it also becomes apparent that they work quite differently: Graham's algorithm is much more precise but produces a much lower number of warnings, whereas Naïve Bayes gives more alarms but is also less precise in its alarms. Even though our version of Graham's algorithm used a much lower threshold than is used for spam filtering, Graham's algorithm is still much more conservative in its predictions.

| | Dataset 4 | | | Dataset 5 | | |
|---|---|---|---|---|---|---|
| Algorithm | Prec. | Recall | F | Prec. | Recall | F |
| *Static Keywords* | 25.93% | 20.00% | 22.58% | 27.92% | 44.79% | 34.40% |
| *Dynamic Keywords* (2) | 37.14% | 37.14% | 37.14% | 33.33% | 53.13% | 40.96% |
| *Graham* (0.51) | 82.50% | 47.14% | 60.00% | 69.23% | 18.75% | 29.51% |
| *Naïve Bayes* (0.5) | 56.25% | 51.43% | 53.73% | 58.73% | 38.54% | 46.54% |
| *Combined* (2:1,0.34) | 85.00% | 97.14% | 90.67% | 84.91% | 80.21% | 82.49% |

Table 5.4: Precision and recall of the learning algorithms on the control datasets

The *Combined* algorithm always provides good precision and recall values. They are typically among the top two of all results, and no algorithm beats them in both dimensions. This can also be seen when looking at the F-measures, where it is only beaten once on dataset 3 (which we already had observed and discussed above for the accuracy results). In general, it seems that the combined algorithm compensates the weaknesses of each of its constituent algorithms, never produces significantly worse results, but sometimes a considerably better performance. For example, on Dataset 1, the $F$-measure of the combined method is 20% higher than Naïve Bayes and 30% higher than Graham's algorithm. This behavior can be explained by the fact that Graham's algorithm and Naïve Bayes classify different e-mails as attachments.

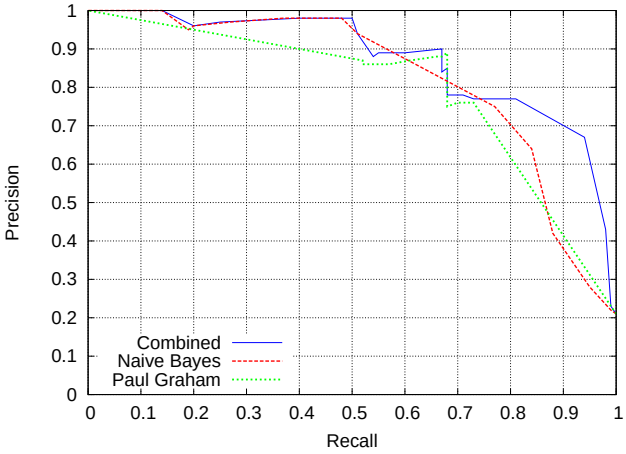


Figure 5.1: Recall/Precision diagram – Dataset 2

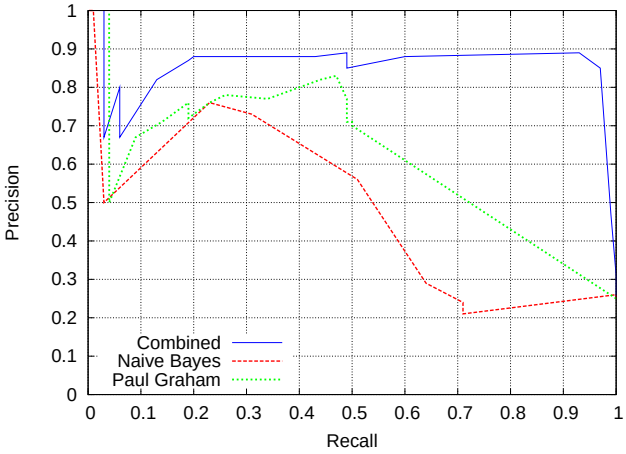


Figure 5.2: Recall/Precision diagram – Dataset 4

If one of the algorithms found an attachment-e-mail with high probability, this e-mail is reported. If the probability of the algorithms is on the borderline, the algorithm decides now with the help of the other if it is an attachment-e-mail.

All of the above findings are, again, confirmed on the control datasets, which are shown in Table 5.4.

Finally, Figures 5.1 and 5.2 show two (non-interpolated) recall and precision diagrams for datasets 2 and 4, which are generated by sorting all predictions according to the predicted score (similar to ROC curves, which are described in the next section) and computing and plotting the recall and precision values after each possible threshold. In both cases, one can see that the combined algorithm outperforms the other two in particular at higher recall levels, which indicates that the corrective effect is much higher in regions where the scores returned by Naïve Bayes and Graham's algorithm are less reliable.

## 5.3 ROC-Curve/AUC

So far, we primarily evaluated the classification performance of the methods with respect to the thresholds that we found worked best for our problems. However, it is of course also interesting to evaluate the quality of the scores $nb(M)$ (3.10), $g(M)$ (3.13) and $s(M)$ (3.15) independent of the selected thresholds. For the accuracy, recall and precision results presented in the last sections (except for the curves in Figures 5.1 and 5.2), these scores were thresholded in order to make a final prediction for attachment or no-attachment, and the presented results crucially depend on this parameter. ROC analysis allows to abstract from concrete values of this threshold.

In general, a good scoring function should give higher scores to messages with attachment than to messages without attachments. This can be most clearly evaluated with ROC curves. In an ROC curve (cf. also Section 4.2), the diagonal between the lower left corner and the upper right corner represents a random ordering of the predictions. the farther the curve moves away from the diagonal, the better is the sorting of the prediction into the two classes. The ideal case is

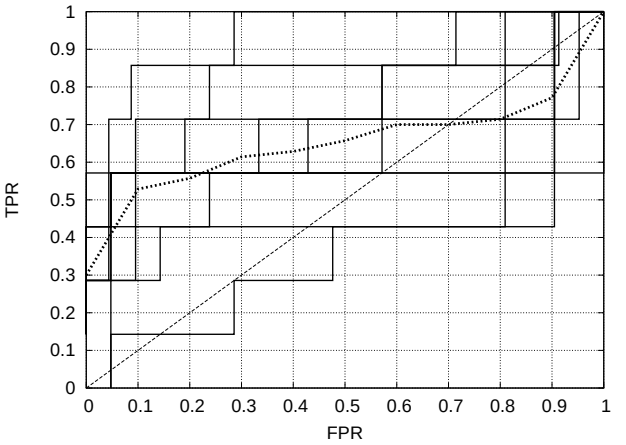


Figure 5.3: ROC-curves for the 10 folds and average curve for Naive Bayes on Dataset 4

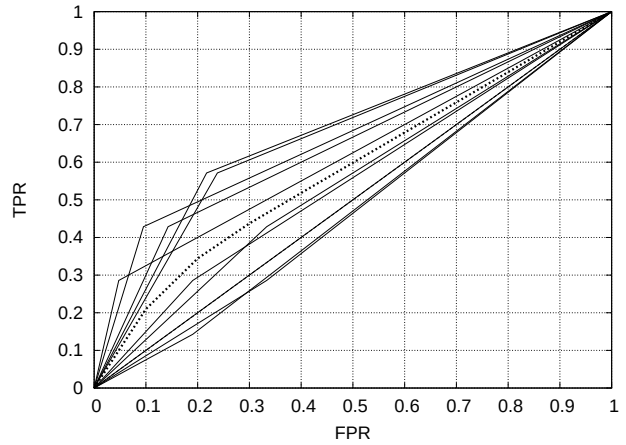


Figure 5.4: ROC-curves for the 10 folds and average curve for Dynamic Keywords on Dataset 4

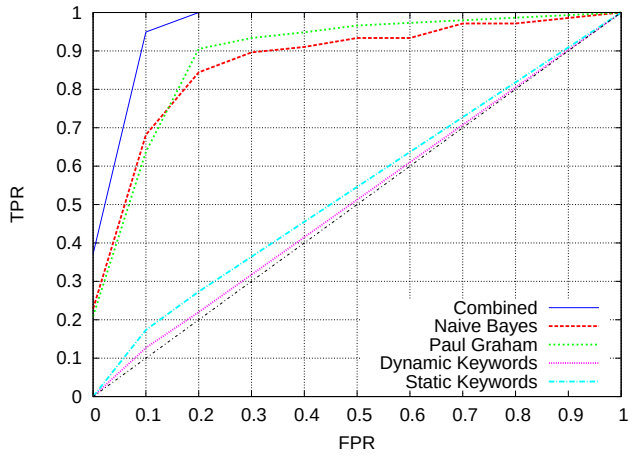


Figure 5.5: ROC-curves for Dataset 1

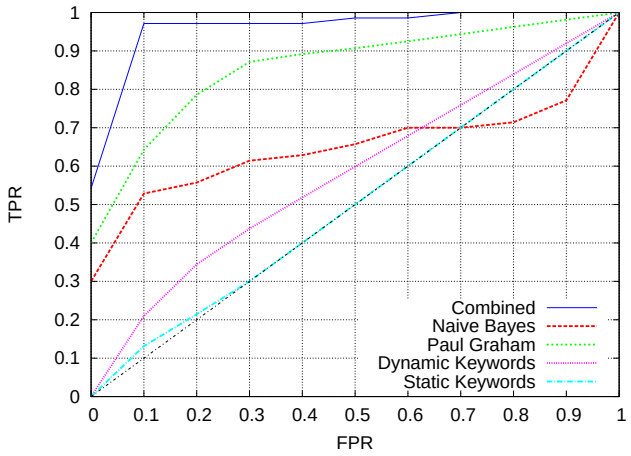


Figure 5.6: ROC-curves for Dataset 4

when all messages with attachments receive a higher score than all messages without attachments. In this case, the ROC curve starts at the lower left corner $(0,0)$, goes straight up to the upper left corner $(0,1)$, and then continues straight to the upper right corner $(1,1)$ of the ROC space.

For comparison, we also show the results of the keyword classifiers, even though these do not return any scores. Here, the ROC curve consist only of a three points. The middle point represents the true positive and false positive rates of the classifier. The curve is formed by connecting this point to $(0,0)$ and $(1,1)$.

All shown curves are average curves over all 10 folds. The averages were formed by via an 11-point vertical average (Fawcett, 2006). This means that the average curve consists of measurements for a false positive rate of 0, 0.1, ..., 0.9, 1.0, which are the average of the point values in the individual curves. Figures 5.3 and 5.4 illustrate this by showing the curves of the individual folds and (with a dotted line) the resulting average curve for the *Naïve Bayes* and the *Dynamic Keyword* classifiers. In Figure 5.3, we can see that the individual curves only have seven different levels on the *TPR*-axis. This is because each of the ten folds only contained seven of the 70 e-mails with an attachment. In Figure 5.4, we can clearly see the three-point curves of the keyword-based classifier. Each of the angles represents one of the 10 classifier points, the dotted line, again, shows the average performance.

Figures 5.5 and 5.6 show the AUC curves for Datasets 1 and 4. Essentially, they reflect the results that we have obtained so far: a very good separation with the combined algorithm is possible, the worst separation is achieved with the two keyword-based algorithms. In particular on Dataset 1 (Figure 5.5) they are both very close to a random classification. On Dataset 4, the *Dynamic Keywords* approach is a little bit better, but still has a very bad performance. All three learning algorithms exhibit a much stronger performance and are clearly above the diagonal, with the *Combined* algorithm dominating the other two.

The *Naï*ve Bayes algorithm in Figure 5.6 shows a worse than random performance in the upper left region. Essentially this means that some of the attachment e-mails have been ranked very low, i.e., the algorithm was quite certain that they do not require attachments. In that area, the predictions would be better if they were reversed. As can be seen from Figure 5.3, this did not happen in all of the folds of the cross-validation.

The area under these ROC curves is a good measure for summarizing the ranking performance of the algorithms. Essentially, it estimates the probability that an e-mail with attachment is ranked before an e-mail without attachment. The AUC values of all algorithms are shown in Table 5.5. They confirm again the order of the algorithms that we have already observed. The keyword-based algorithms are only slightly better than random, while Graham's algorithm and Naïve Bayes exhibit a reasonable performance in about the same range (with the exception of the outlier Dataset 4 for *Naïve Bayes* that we have discussed above). The *Combined* algorithm shows a very strong performance. On all five datasets, the probability that it ranks an e-mail without attachment before an e-mail with attachment was less then 5%.

| AUC | Dataset 1 | Dataset 2 | Dataset 3 | Dataset 4 | Dataset 5 |
|---|---|---|---|---|---|
| *Static Keywords* | 0.54 | 0.69 | 0.60 | 0.51 | 0.66 |
| *Dynamic Keywords* | 0.51 | 0.68 | 0.81 | 0.58 | 0.70 |
| *Graham* | 0.90 | 0.86 | 0.90 | 0.86 | 0.84 |
| *Naïve Bayes* | 0.88 | 0.90 | 0.94 | 0.66 | 0.81 |
| *Combined* | 0.97 | 0.95 | 0.97 | 0.96 | 0.97 |

Table 5.5: AUC-values on the learning algorithms

## 6 Conclusion

In this paper, we compared machine learning algorithms to the keyword-based algorithms that are currently used in practice for the problem of recognizing missing spam mails. Our results show that the learning algorithms clearly outperform the keyword-based approaches, both an approach with a static set of keywords as well as an approach that automatically adapts its keywords to the user. The two standard learning algorithms, Naïve Bayes and Graham's algorithm seem to fit well to this problem, but exhibit very different characteristics. While Naïve Bayes has a higher recall and a somewhat lower precision, Graham's algorithm, originally proposed for spam filtering, focuses much more on precision. A simple combination of the two outperformed all other algorithms. Given a randomly selected pair of messages, one with attachment and one without, it will recognize the e-mail with attachment in more than 95% of the cases.

## Bibliography


Jake D. Brutlag and Christopher Meek. Challenges of the email domain for text classification. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 103–110, Stanford, CA, 2000. Morgan Kaufmann. 2

William W. Cohen. Learning rules that classify e-mail. In M.A. Hearst and H. Hirsh, editors, *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25. AAAI Press, 1996. Technical Report SS-96-05. 2

Elisabeth Crawford, Juy Kay, and Eric McCreath. IEMS – the intelligent email sorter. In Claude Sammut and Achim G. Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning (ICML-02)*, pages 263–272, Sydney, Australia, 2002. Morgan Kaufmann. 2

Pedro Domingos and Michael J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997. 5

Tom Fawcett. "In vivo" spam filtering: A challenge problem for data mining. *SIGKDD explorations*, 5(2), December 2003. URL `http://www.hpl.hp.com/personal/Tom_Fawcett/papers/spam-KDDexp.pdf`. 2

Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. 9, 13

Joshua Goodman, David Heckerman, and Robert Rounthwaite. Stopping spam. *Scientific American*, April 2005. 2

Paul Graham. Better Bayesian filtering. In *Proceedings of the 2003 Spam Conference (`http://spamconference.org/proceedings2003.html`)*, Cambridge, MA, 2003. URL `http://www.paulgraham.com/better.html`. 2, 6

Paul Graham. A plan for spam. `http://www.paulgraham.com/spam.html`. 2, 6

Bryan Klimt and Yiming Yang. The Enron corpus: A new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning (ECML-04)*, pages 217–226, 2004. URL `http://springerlink.metapress.com/openurl.asp?genre=article{\&}issn=0302-9743{\&}volume=3201{\&}spage=217`. 9

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. URL `http://www.informationretrieval.org/`. 8

Terry R. Payne and Peter Edwards. Interface agents that learn: An investigation of learning issues in a mail agent interface. *Applied Artificial Intelligence*, 11(1):1–32, 1997. 2

Foster Provost and Tom Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231, 2001. 9

Jason D. M. Rennie. ifile: An application of machine learning to e-mail filtering. In M. Grobelnik, D. Mladenić, and N. Milic-Frayling, editors, *Proceedings KDD-2000 Workshop on Text Mining*, Boston, MA, 2000. 2