Learning to Filter Unsolicited Commercial E-Mail

Ion Androutsopoulos
Athens University of Economics and Business
and
Georgios Paliouras and Eirinaios Michelakis
National Centre for Scientific Research "Demokritos"

We present a thorough investigation on using machine learning to construct effective personalized anti-spam filters. The investigation includes four learning algorithms, Naive Bayes, Flexible Bayes, LogitBoost, and Support Vector Machines, and four datasets, constructed from the mailboxes of different users. We discuss the model and search biases of the learning algorithms, along with worst-case computational complexity figures, and observe how the latter relate to experimental measurements. We study how classification accuracy is affected when using attributes that represent sequences of tokens, as opposed to single tokens, and explore the effect of the size of the attribute and training set, all within a cost-sensitive framework. Furthermore, we describe the architecture of a fully implemented learning-based anti-spam filter, and present an analysis of its behavior in real use over a period of seven months. Information is also provided on other available learning-based anti-spam filters, and alternative filtering approaches.

Key Words: E-mail abuse, machine learning, spam, text categorization

NCSR "Demokritos" Technical Report, No. 2004/2, March 2004
revised: October 2004; minor corrections: October 2006

Address of first author: Dept. of Informatics, Athens University of Economics and Business, Patission 76, GR-104 34 Athens, Greece. Tel: +30-210-8203571, Fax: +30-210-8226105, e-mail: `ion@aueb.gr`

Address of other authors: Institute of Informatics and Telecommunications, N.C.S.R. "Demokritos", P.O. Box 60228, GR-153 10 Ag. Paraskevi, Greece. Tel: +30-210-6503158, Fax: +30-210-6532175, e-mails: {paliourg, ernani}@iit.demokritos.gr.

1. INTRODUCTION

In recent years, the increasing popularity and low cost of e-mail have attracted direct marketers. Bulk mailing software and lists of e-mail addresses harvested from Web pages, newsgroup archives, and service provider directories are readily available, allowing messages to be sent blindly to millions of recipients at essentially no cost. Resources of this kind are a temptation for amateur advertisers and opportunists. Consequently, it is increasingly common for users to receive large quantities of *unsolicited commercial e-mail* (UCE), advertising anything, from vacations to get-rich schemes. The term *spam*, that originally denoted Usenet messages cross-posted to numerous newsgroups, is now also used to refer to unsolicited, massively posted e-mail messages. Spam is actually a broader term than UCE, since spam e-mail is not necessarily commercial (e.g., spam messages from religious cults). Non-UCE spam messages, however, are rare, to the extent that the two terms can be used interchangeably. Note that messages sent by viruses are not considered spam, although they, too, can be sent blindly to large numbers of users.

Spam messages are extremely annoying to most users, as they clutter their mailboxes and prolong dial-up connections. They also waste the bandwidth and CPU time of ISPs, and often expose minors to unsuitable (e.g., pornographic) content. Following anti-spam campaigns by volunteer organizations[1], most respectable companies now seek the consent of potential customers before sending them advertisements by e-mail, usually by inviting them to tick check boxes when visiting their Web pages. Unfortunately, it does not take many irresponsible advertisers to flood the Internet with spam messages. A 1997 study reported that spam messages constituted approximately 10% of the incoming messages to a corporate network [Cranor and LaMacchia 1998]. Public comments made by AOL in 2002 indicated that of an estimated 30 million e-mail messages daily, about 30% on average was spam; and Jupiter Media Matrix predicted that the number of spam messages users receive would more than double from 2002 to 2006, exceeding 1600 messages per user per year in 2006.[2] The situation seems to be worsening, and without appropriate counter-measures, spam messages may undermine the usability of e-mail.

Legislative counter-measures are gradually being adopted in the United States, Europe, and elsewhere [Gauthronet and Drouard 2001], but they have had a very limited effect so far, as evidenced by the number of spam messages most users receive daily. Of more direct value are *anti-spam filters*, software tools that attempt to identify incoming spam messages automatically. Most commercially available filters of this type currently appear to rely on white-lists of trusted senders (e.g.,

[1]Consult `http://www.cauce.org/`, `http://www.junkemail.org/`, and `http://spam.abuse.net/`.
[2]Sources: CAUCE site, September 2002, and ZDNet, July 9th, 2002, respectively.

as listed in each user's address book), black-lists of known spammers (typically provided and updated by the filters' developers), and hand-crafted rules that block messages containing specific words or phrases (e.g., "be over 21" in the message's body) or other suspicious patterns in the header fields (e.g., empty "To:" field).

White-lists reduce the risk of accidentally blocking non-spam, hereafter called *legitimate*, messages, but they have to be combined with other techniques to discriminate between spam messages and legitimate messages from senders not in the white-list (e.g, first-time correspondents). Black-lists containing e-mail addresses or domain names are of little use, as spammers typically use fake sender addresses. Black-lists in the form of continuously updated on-line databases of IP numbers that have or could be used by spammers (e.g., open SMTP relays) are more effective.[3] However, spammers can bypass them by sending messages from new IP numbers. Furthermore, black-lists can victimize innocent users, or even entire domains, when addresses or IP numbers are used by spammers without the consent of their owners. Hand-crafted rules are also problematic: rules that are common across users (e.g., the default rules of a filter) can be studied by spammers, who can adjust their messages to avoid triggering the rules. To avoid this problem and be more effective, rules need to be tuned for individual users or groups. This is a tedious task requiring time and expertise, which has to be repeated periodically to account for changes in the content and wording of spam e-mail [Cranor and LaMacchia 1998].

The success of machine learning techniques in text categorization [Sebastiani 2002] has led researchers to explore learning algorithms in anti-spam filtering. A supervised learning algorithm [Mitchell 1997] is given a corpus of e-mail messages that have been classified manually as spam or legitimate, and builds a classifier, which is then used to classify new incoming messages to the two categories. Apart from collecting separately spam and legitimate training messages, the learning process is fully automatic, and can be repeated to tailor the filter to the incoming messages of particular users or groups, or to capture changes in the characteristics of spam messages. Viewed in this way, anti-spam filtering becomes a problem of text categorization, although the categories of anti-spam filtering (legitimate or spam) are much less homogeneous in terms of topics than those of most text categorization applications, including other e-mail and news categorization systems [Lang 1995; Cohen 1996; Payne and Edwards 1997]. A further complication is that anti-spam filtering is a cost-sensitive problem: the cost of accidentally blocking a legitimate message can be higher than letting a spam message pass the filter, and this difference must be taken into account during both training and evaluation.

In previous work, Sahami et al. [1998] experimented with an anti-spam filter based

[3]See `http://www.mail-abuse.org/`, `http://www.ordb.org`, `http://www.spamhaus.org/`, and `http://www.spews.org/` for examples of such lists.

on a Naive Bayes classifier [Mitchell 1997]. In similar anti-spam experiments, Pantel and Lin [1998] found Naive Bayes to outperform Ripper [Cohen and Singer 1999], while Gomez Hidalgo and Mana Lopez [2000] reported that C4.5 [Quinlan 1993] and PART [Frank and Witten 1998] performed on average better than Naive Bayes and k-Nearest Neighbor [Cover and Hart 1967; Aha and Albert 1991]. Drucker et al. [1999] experimented with Ripper, Rocchio's classifier [Rocchio 1971; Joachims 1997], Support Vector Machines (SVMs) [Cristianini and Shawe-Taylor 2000], and BOOSTEXTER, a version of C4.5 with boosting [Schapire and Singer 2000], with results showing that SVMs and C4.5 with boosting achieve very similar error rates, both outperforming Rocchio's classifier. A direct comparison of these previous results, however, is impossible, as they are based on different and not publicly available datasets. Furthermore, the reported figures can be misleading, as they are not formulated within a cost-sensitive framework.

To address these shortcomings, we made publicly available two mixed collections of manually classified legitimate and spam messages, Ling-Spam and PU1, that can be used as benchmarks, along with suitable cost-sensitive evaluation measures. In previous work, we used the two collections and the proposed measures in experiments with a Naive Bayes and a k-Nearest Neighbor learner, respectively; both algorithms achieved comparable results [Androutsopoulos et al. 2000a; 2000b; 2000c; Sakkis et al. 2003b]. Our benchmark corpora and measures have received a lot of attention. Carreras and Marquez [2001] used PU1 and our evaluation measures in experiments that showed the positive effect of boosting in decision-tree filters. Kolcz and Alspector [2001] employed an extension of our evaluation methodology in an exploration of SVM-based anti-spam filtering. Gomez Hidalgo [2002] used Ling-Spam and our measures to compare filters based on C4.5, Naive Bayes, PART, Rocchio, and SVMs; the results showed no clear winner, but SVMs had most frequently the best results. Schneider [2003] used Ling-Spam and PU1 to study the effect of different event models in Naive Bayes, along with different versions of attribute selection measures based on information gain.

Continuing our previous work, this paper presents what is, to the best of our knowledge, the most extensive exploration of learning approaches to anti-spam filtering to date. We use the same evaluation framework as in our previous work, but apart from PU1, we introduce three additional message collections, PU2, PU3, and PUA, each deriving from a different user. Furthermore, unlike previous work, this article is not limited to corpus experiments. We describe the architecture and functionality of a fully implemented and publicly available learning-based anti-spam filter that incorporates the key findings of our experiments. We also present an assessment of its performance in real-life, including a study of its errors.

In terms of learning algorithms, our exploration includes Naive Bayes, which despite its simplicity has proven particularly effective in previous experiments, an enhanced version of Naive Bayes, called Flexible Bayes [John and Langley 1995], that performs particularly well with continuous attributes, an SVM implementation based on Sequential Minimal Optimization [Platt 1999], and a boosting algorithm, LogitBoost [Friedman et al. 2000], with regression stumps as weak learners. We chose regression stumps, a form of decision trees of unary depth, as weak learners in the boosting experiments after witnessing unacceptable training times in initial experiments with J48 as the weak learner.[4] Drucker et al. [1999] make a similar observation, while the results of Carreras and Marquez [2001] indicate that boosting with decision stumps still leads to good results. Overall, our experiments include the learning algorithms, or variations of them, that have been reported to produce the most promising results to date. Unlike previous work, however, we shed light to the effect of several parameters, including the size of the training corpus, the size of the attribute set, and the effect of n-gram attributes.

We note that learning-based components are also beginning to appear in some e-mail clients and public domain filters; Section 6 below provides related information. Most of these components, however, are so far limited to flavors of Naive Bayes, and do not appear to have been subjected to rigorous experimental investigation. On the other hand, one of their interesting characteristics is that they employ a variety of additional attributes, that do not examine only the text of the messages. For example, there may be attributes examining whether or not the message was posted from a suspicious domain or by night, whether or not it carries attachments of particular types, colored text, or links, and attributes of this kind can provide additional clues on whether or not the message is spam. In contrast, the investigation of this paper considers only the text in the subjects and bodies of the messages, ignoring other headers, HTML markup, non-textual attributes, and attachments. This is a limitation of our work, that was introduced for a variety of practical reasons; for example, devising non-textual attributes requires a detailed study of the characteristics of spam messages and the habits of spammers, while the attributes we used can be selected in a fully automatic manner; processing attachments requires obtaining and invoking readers for the various attachment types; and processing headers is not trivial when messages originate from different e-mail servers and clients. Despite this limitation, we believe that the work of this paper is useful, in that it provides a detailed investigation of what can be achieved using only the text of the messages, thereby acting as a solid basis on which further improvements can be tried. In fact, our experimental results indicate that effective learning-based anti-spam filters can be built even when only the text of

[4]Our corpus experiments were performed using Weka version 3.1.9. Weka is available from `http://www.cs.waikato.ac.nz/ml/weka/`. J48 is Weka's implementation of C4.5.

the messages is examined, and, as we discuss in Section 6, there are still several improvements that can be made in the way the text of the messages is handled, to confront tricks that spammers employ to confuse anti-spam filters.

The remainder of this paper is organized as follows. Section 2 presents the message collections of our experiments, and how they compare to previous collections. Section 3 discusses the learning algorithms that we used and the preprocessing that was applied to the collections to turn them into the representation the learning algorithms require. Section 4 shows how the notion of cost can be embedded in usage scenarios for anti-spam filters, it defines our cost-sensitive evaluation measures, and explains how learning algorithms can be made sensitive to cost. Section 5 presents our corpus experiments and their results. Section 6 presents the architecture of our prototype filter, along with an assessment of its performance in real life. Finally, Section 7 concludes and suggests further directions.

2. BENCHMARK CORPORA FOR ANTI-SPAM FILTERING

Research on text categorization has benefited significantly from the existence of publicly available, manually categorized document collections, like the Reuters corpora, that have been used as standard benchmarks.[5] Producing similar corpora for anti-spam filtering is more complicated, because of privacy issues. Publicizing spam messages does not pose a problem, since spam messages are distributed blindly to very large numbers of recipients, and, hence, they are effectively already publicly available. Legitimate messages, however, in general cannot be released without violating the privacy of their recipients and senders.

One way to bypass privacy problems is to experiment with legitimate messages collected from freely accessible newsgroups, or mailing lists with public archives. Ling-Spam, the earliest of our benchmark corpora, follows this approach. It consists of 481 spam messages received by the first author, and 2412 legitimate messages retrieved from the archives of the Linguist list, a moderated – and, hence, spam-free – list about linguistics.[6] Ling-Spam has the disadvantage that its legitimate messages are more topic-specific than the legitimate messages most users receive. Hence, the performance of a learning-based anti-spam filter on Ling-Spam may be an over-optimistic estimate of the performance that can be achieved on the incoming messages of a real user, where topic-specific terminology may be less dominant among legitimate messages. In that sense, Ling-Spam is more appropriate to experiments that explore the effectiveness of filters that guard against spam messages sent to topic-specific mailing lists [Sakkis et al. 2003b].

The SpamAssassin public mail corpus is similar to Ling-Spam, in that its legiti-

[5] See `http://about.reuters.com/researchandstandards/corpus/`.
[6] Ling-Spam and the PU corpora are available at `http://www.iit.demokritos.gr/skel/i-config/`.

mate messages are publicly available.[7] It contains 1897 spam and 4150 legitimate messages, the latter collected from public fora or donated by users with the understanding that they may be made public. The corpus is less topic-specific than Ling-Spam, but its legitimate messages are again not indicative of the legitimate messages that would arrive at the mailbox of a *single* user, this time for the opposite reason. Many of the legitimate messages that a user receives contain terminology reflecting his/her profession, interests, etc. that is rare in spam messages (see Section 3.1 below), and part of the success of personal learning-based filters is due to the fact that they learn to identify this user-specific terminology. In a concatenation of legitimate messages from different users this user-specific terminology becomes harder to identify. Hence, the performance of a learning-based filter on the SpamAssassin corpus may be an under-estimate of the performance that a personal filter can achieve.

An alternative solution to the privacy problem is to distribute information about each message (e.g., the frequencies of particular words in each message) rather than the messages themselves. The Spambase collection follows this approach.[8] It consists of 4601 vectors providing information about 1813 spam and 2788 legitimate messages. Each vector contains the values of 57 pre-selected attributes, mostly word frequencies; vector representations will be explained further in Section 3.1. Spambase, however, is much more restrictive than Ling-Spam and the SpamAssassin corpus, because its messages are not available in raw form, and, hence, it is impossible to experiment with attributes other than those chosen by its creators.

A third approach, adopted in PU1, is to release benchmark corpora that each consists of messages received by a particular user, after replacing each token by a unique number throughout the corpus, as illustrated in Figure 1. The mapping between tokens and numbers is not released, making it extremely difficult to recover the original text.[9] This bypasses privacy problems, while producing a form of the messages that is very close, from a statistical point of view, to the original text. For example, one can compute the frequency of any (anonymized) token across the messages, the frequencies of particular sequences of tokens, or the probabilities that particular tokens appear at certain parts of the messages (e.g., the first or last paragraph). The same encoding scheme was used in PU2, PU3, and PUA.

There is an additional issue to consider when constructing benchmark corpora

[7]SpamAssassin is a sophisticated anti-spam filter that will be discussed briefly in Section 6.3. The SpamAssassin corpus was created by Justin Mason of Network Associates, and can be downloaded from `http://www.spamassassin.org/publiccorpus/`.
[8]Spambase was created by M. Hopkins, E. Reeber, G. Forman, and J. Suermondt. It is available from `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
[9]Decryption experts may be able to recover short sequences of common words based on statistical analysis, but not proper names, addresses, and similar personal details.

Fig. 1. A spam message after tokenization, and its encoded form.

Subject: unlimited cash and millions in your mail	Subject: 40 9 7 24 18 45 21
dear friend ,	10 13 1
this is one of the great money making programs	38 20 29 28 36 15 25 23 34
over the net and the only one i have earned	31 36 27 7 36 30 29 17 16 12
money with . the advantage of this program is	25 42 2 36 5 28 38 33 20
that all the money is directed to your mailing	35 6 36 25 20 11 39 45 22
address without intrusion of a third party . [...]	4 43 19 28 3 37 32 2 [...]
my best wishes to you , george	26 8 41 39 44 1 14

for learning-based anti-spam filtering. Typically, a large percentage of a user's incoming messages are from senders the user has regular correspondence with, RC messages for brevity. Regular correspondents are unlikely to send spam messages,[10] and their addresses are usually stored in the user's address book. The latter can act as a personal white-list: messages from senders in the address book can be categorized immediately as legitimate, without invoking the learning-based component of the anti-spam filter. That component, which is the focus of our experiments, will be invoked to decide only upon messages from non-regular correspondents. Hence, for the collection to be representative of the messages that the learning-based component will be faced with, it must exclude RC messages. In the PU corpora, this was achieved by retaining only the earliest five legitimate messages of each sender. It is assumed that by the time the sixth legitimate message arrives, the sender will have already been inserted in the recipient's address book, and the learning-based component will no longer need to examine messages from that sender.

Duplicate messages are another consideration. It is common for users to receive multiple copies of a spam message on the same day, as a result of duplicate entries or equivalent addresses in the spammers' lists. Duplicates can cause problems in cross-validation experiments (Section 3.1.2 below), as they can end up in both the training and test parts of the collection, leading to over-optimistic results. In PU1 and PU2, duplicate spam messages received on the same day were removed manually. In the more recent PU3 and PUA, the process was automated using a flavor of UNIX's `diff` command. Two messages were taken to be different, if they differed in at least five lines. All duplicates, regardless of when they were received, were removed, and this mechanism was applied to both spam and legitimate messages.

Table I provides more information on the composition of the PU corpora. The legitimate messages in PU1 are the English legitimate messages that the first author had received and saved over a period of 36 months, excluding self-addressed messages; this led to 1182 messages. Legitimate messages with empty bodies and

[10]A possible exception are chain letters, but our view is that they are not spam, as they are resent by different people, who are, at least partially, responsible for the waste of resources they cause.

Table I. Composition and sizes of the PU collections. The third column shows the number of legitimate messages from regular correspondents (RC), while the fourth column counts the legitimate messages that were discarded for other reasons (e.g., empty bodies or duplicates); in PU1 and PU2 both types of legitimate messages were discarded in one step. The last column shows the legitimate-to-spam ratio in the retained messages.

collection name	legitimate initially	RC messages	other legit. discarded	legitimate retained	spam retained	total retained	L:S ratio
PU1	1182	564		618	481	1099	1.28
PU2	6207	5628		579	142	721	4.01
PU3	8824	6253	258	2313	1826	4139	1.27
PUA	980	369	40	571	571	1142	1

RC messages were then removed, the latter using the white-list emulation discussed above; this left 618 legitimate messages. The spam messages in PU1 are all the spam messages the first author had received over a period of 22 months, excluding non-English messages and duplicates received on the same day; they are the same as the spam messages of Ling-Spam. PU2 was constructed in the same way, starting from the legitimate and spam messages that a colleague of the authors had received over a period of 22 months. The vast majority of his legitimate messages were RC.

PU3 and PUA contain the legitimate messages that the second author and another colleague, respectively, had saved up to the time of our experiments; unlike PU1 and PU2, non-English (e.g., Greek) legitimate messages were not removed. As neither of the two persons had saved the spam messages they had received, in both collections we used a pool of 1826 duplicate-free spam messages deriving from PU1, additional spam messages received by the first author, the SpamAssassin corpus, and spam messages donated to us.[11] In PUA, only 571 of the pool's messages were used, the same as the number of retained legitimate messages. The legitimate-to-spam ratio (Table I) of PU3 is approximately the same as that of the earlier PU1, except that PU3 is almost four times larger. In PU2, there is only one spam message for every four non-RC legitimate messages, while in PUA a message from a sender outside the white-list has equal a priori probability to be legitimate and spam.

In all PU collections, attachments, HTML tags, and header fields other than the subject were removed, and the messages were then encoded as discussed above. Unlike earlier experiments [Androutsopoulos et al. 2000c] no lemmatizer or stop-list was used, as their effect is controversial. Punctuation and other special symbols (e.g., “!”, “$”) were treated as tokens. Many of these symbols are among the best discriminating attributes in the PU corpora, because they are more common in spam messages than legitimate ones; related discussion follows in Section 3.1.

[11]The donation was from F. Sebastiani. Orasan and Krishnamurthy have recently made available a similar spam collection; see `http://clg.wlv.ac.uk/projects/junk-email/`. For a more recent attempt to create a large public spam archive, see `http://www.spamarchive.org/`.

3. LEARNING ALGORITHMS

This section describes the four learning algorithms that we examined in the experiments of this paper. Each algorithm can be viewed as searching for the most appropriate classifier in a search space that contains all the classifiers it can learn. We focus on the following characteristics of the algorithms, which bias their search:

Instance representation. In our case, the representation of the messages that is used during the training and classification stages.

Model representation. The model the algorithm uses to represent the classifiers it can learn. Classifiers that cannot be represented are not considered, and, hence, cannot be learnt. This introduces a bias towards the classifiers that the model can represent, the *model bias*. The more restrictive the model representation is, the strongest the model bias.

Search method. The search method, including classifier selection criteria, that the algorithm adopts to find a good classifier among the ones it can learn. The search method introduces its own bias, the *search bias*; for example, it may disregard whole areas of the search space, thus favoring some classifiers against other possibly more appropriate ones.

The discussion aims to provide a clear picture of the algorithms that we used, for the benefit of readers not familiar with them, but also to clarify their differences and the limitations that the choices of representations and search methods introduce as implicit biases [Mitchell 1997]. Furthermore, we examine the computational complexity of each algorithm, both at the learning and classification stages, speed being an important factor in the design of on-line anti-spam filters.

The rest of this section is structured as follows. Subsection 3.1 describes the representation of instances, which is the same in all of the learning algorithms we considered. Subsections 3.2 to 3.5 then describe the four learning algorithms, each in terms of its model and search bias. Subsection 3.6 summarizes the most important differences among the algorithms.

3.1 Instance representation

3.1.1 *Attribute vectors.* All four learning algorithms require the same instance representation. In our case, where the instances are messages, each message is transformed into a vector $\langle x_1, \ldots, x_m \rangle$, where $x_1, \ldots, x_m$ are the values of the attributes $X_1, \ldots, X_m$, much as in the vector space model in information retrieval [Salton and McGill 1983]. In the simplest case, each attribute represents a single token (e.g., “adult”), and all the attributes are Boolean: $X_i = 1$ if the message contains the token; otherwise, $X_i = 0$. All previous experiments with Ling-Spam and PU1 had adopted this representation.

The experiments of this paper explored richer vector representations. First, instead of Boolean attributes, frequency attributes were used. With frequency attributes, the value of X_i in each message d is $x_i = \frac{t_i(d)}{l(d)}$, where $t_i(d)$ is the number of occurrences in d of the token represented by X_i, and $l(d)$ is the length of d measured in token occurrences.[12] Frequency attributes are more informative than Boolean ones. For example, reporting simply that the token "$" is present in a message, as in the case of Boolean attributes, may not be enough to raise suspicions that the message is spam, especially if the message is very long; but if a very high proportion of "$" tokens is present in the message, a fact that can be represented with frequency attributes, this may be a good indicator that the message is spam. The additional information of frequency attributes usually comes at the cost of extra complexity in the learning algorithms, which is needed to allow them to cope with attributes whose values are real numbers.

In a second enhancement of the attribute vector representation, we have examined attributes that represent not only single tokens, but more generally n-grams of tokens with $n \in \{1, 2, 3\}$, i.e., sequences of tokens of length 1, 2, or 3. In that case, $t_i(d)$ is the number of occurrences in message d of the n-gram represented by X_i, while $l(d)$ remains the number of token occurrences in d. Although the effectiveness of n-gram attributes in text categorization is controversial [Sebastiani 2002], spam messages often contain particularly characteristic phrases (e.g., "to be removed"), and, thus, there is reason to expect that some n-gram attributes ($n > 1$) may discriminate particularly well between spam and legitimate messages. Unfortunately, the number of possible distinct n-grams in a corpus grows exponentially to n, which makes subsequent computations involving n-grams prohibitive, unless n is kept small; this is why values of n greater than 3 were not explored.

3.1.2 *Attribute selection.* Using every single token or n-gram as an attribute is impractical, especially when using n-grams, as it leads to vectors of very high dimensionality, slowing down significantly the learning algorithms. Hence, an attribute selection phase is commonly employed before invoking the learning algorithms, which detects the attributes that discriminate best between the categories. In our experiments, attribute selection was performed in three steps, discussed below.

Step 1: A pool of candidate attributes was formed. In experiments where attributes represented single tokens, the pool contained a candidate attribute for each token in the training corpus. In experiments where attributes represented n-grams, the pool contained a candidate attribute for each token sequence of length 1, 2, or

[12]The values of frequency attributes are similar to the TF scores commonly used in information retrieval; an IDF component could also be added, to demote tokens that are common across the messages of the training collection.

3 in the training corpus.

Step 2: Candidate attributes corresponding to tokens or n-grams that did not occur at least 4 times in the training corpus were removed from the pool. This removed a large number of very low frequency tokens or n-grams that were too rare to be useful in a general classifier, speeding up the attribute selection process and lowering memory requirements.

Step 3: The information gain $IG(X, C)$ of each remaining candidate attribute X with respect to the category-denoting random variable C was computed, using the Boolean values of the candidate attributes, as shown below; c_L and c_S denote the legitimate and spam categories, respectively. The m candidate attributes with the highest IG scores were then retained, with m depending on the experiment.

$$IG(X, C) = \sum_{x \in \{0,1\},\ c \in \{c_L, c_S\}} P(X = x \wedge C = c) \cdot \log_2 \frac{P(X = x \wedge C = c)}{P(X = x) \cdot P(C = c)} \quad (1)$$

The probabilities were estimated from the training corpus with m-estimates [Mitchell 1997], in order to avoid zero probabilities. It can be shown that $IG(X, C)$ measures the average reduction in the entropy of C given the value of X [Manning and Schutze 1999]. Intuitively, it shows how much knowing X helps us guess correctly the value of C. Yang and Pedersen [1997] have found experimentally information gain to be one of the best attribute selection measures. Schneider [2003] experimented with alternative versions of information gain attribute selection, intended to be more suitable to frequency-valued attributes, instead of using their Boolean forms. However, he observed no significant improvement of the resulting filters.

Selecting m attributes from a pool of M candidate attributes and N training vectors requires $O(MN)$ time to compute the probabilities, then $O(M)$ to compute the information gain scores, and $O(M \log M)$ to sort the candidate attributes on their information gain scores and retain the m best. Thus, the selection process is linear to the number of vectors and log-linear to the number of candidate attributes.

All of the experiments were performed using stratified ten-fold *cross-validation*. That is, each message collection was divided into ten equally large parts, maintaining the original distribution of the two classes in each part. Each experiment was repeated ten times, reserving a different part for testing at each iteration, and using the remaining parts for training. The results were then averaged over the ten iterations, producing more reliable results, and allowing the entire corpus to be exploited for both training and testing. The attribute selection process was repeated anew at each iteration of the cross-validation. When we refer to the training corpus of cross-validation experiments, we mean the nine parts that were used for training at each iteration.

For illustrative purposes, Table II lists, in non-encoded form, the n-grams that correspond to the attributes with the highest *IG* scores in the entire PU1. A large proportion of the first author's incoming legitimate messages, that were used in PU1, concern natural language research. Hence, many of the best attributes correspond to words or phrases that are common in those messages and rare in spam messages (e.g., "research", "speech", "natural language") or vice versa (e.g., "only $", "be removed"). Approximately half of the n-grams in Table II are 2-grams or 3-grams, which indicates that there are many phrases that discriminate well between the two categories.

3.2 Naive Bayes

3.2.1 *Model representation.* The Naive Bayes learner is the simplest and most-widely used algorithm that derives from Bayesian Decision Theory [Duda and Hart 1973]. From Bayes' theorem and the theorem of total probability, the probability that a message with vector $\vec{x} = \langle x_1, \ldots, x_m \rangle$ belongs in category c is:

$$P(C = c \mid \vec{X} = \vec{x}) = \frac{P(C = c) \cdot P(\vec{X} = \vec{x} \mid C = c)}{\sum_{c' \in \{c_L, c_S\}} P(C = c') \cdot P(\vec{X} = \vec{x} \mid C = c')}. \tag{2}$$

The optimal choice for $\vec{x}$ is the category c that maximizes equation (2). The denominator does not affect this choice, as it remains the same for all categories. What matters are the a priori probabilities of the categories and the a posteriori probabilities of the vectors given the category, which need to be estimated from the training data. The number of possible vectors (combinations of attribute values), however, is very large, and many of the vectors will be rare or may not even occur in the training data, making it difficult to obtain reasonable estimates of the required a posteriori probabilities. Hence, a simplifying assumption is made, leading to the Naive Bayes classifier: attributes $X_1, \ldots, X_m$ are considered conditionally independent given the category C. This allows (2) to be rewritten as (3), where now only the much fewer a posteriori probabilities $P(X_i = x_i \mid C = c)$ have to be estimated. Along with the a priori probabilities $P(C = c)$, they constitute the model that Naive Bayes uses to represent a classifier.

$$P(C = c \mid \vec{X} = \vec{x}) = \frac{P(C = c) \cdot \prod_{i=1}^{m} P(X_i = x_i \mid C = c)}{\sum_{c' \in \{c_L, c_S\}} P(C = c') \cdot \prod_{i=1}^{m} P(X_i = x_i \mid C = c')} \tag{3}$$

Although the independence assumption is in most cases over-simplistic, studies in several domains, including anti-spam filtering, have found Naive Bayes to be surprisingly effective [Langley et al. 1992; Domingos and Pazzani 1996; Pantel and Lin 1998; Sahami et al. 1998; Androutsopoulos et al. 2000a; 2000b; 2000c].

When Boolean, or more generally discrete, attributes are used, it can be shown [Duda and Hart 1973] that Naive Bayes can only address linearly separable problems. That is, the classifiers that Naive Bayes can learn can each be viewed as a

Table II. The n-grams with the highest information gain scores in PU1, for $n \in \{1,2,3\}$, ranked by decreasing score. Also shown are the a priori and conditional probabilities of the n-grams, assuming Boolean attributes. Shaded and non-shaded rows indicate n-grams that are more frequent in the spam and legitimate category, respectively.

i	n-gram	$P(X_i = 1)$	$P(X_i = 1 \mid c_L)$	$P(X_i = 1 \mid c_S)$
1	! !	0.231607	0.001612	0.527950
2	!	0.484105	0.216129	0.828157
3	$	0.257947	0.040322	0.538302
4	! ! !	0.181653	0.001612	0.414078
5	language	0.247956	0.440322	0.002070
6	money	0.163487	0.001612	0.372670
7	remove	0.146230	0.001612	0.333333
8	free	0.309718	0.104838	0.573498
9	. 00	0.141689	0.001612	0.322981
10	fax :	0.174386	0.309677	0.002070
11	university	0.219800	0.374193	0.022774
12	click	0.118982	0.001612	0.271221
13	mailing	0.169845	0.029032	0.351966
14	natural	0.156221	0.277419	0.002070
15	the internet	0.110808	0.001612	0.252587
16	removed	0.110808	0.001612	0.252587
17	research	0.281562	0.445161	0.072463
18	university of	0.154405	0.274193	0.002070
19	%	0.128973	0.009677	0.283643
20	: +	0.152588	0.270967	0.002070
21	natural language	0.144414	0.256451	0.002070
22	, 000	0.156221	0.029032	0.320910
23	#	0.126248	0.012903	0.273291
24	! you	0.099000	0.001612	0.225672
25	ion	0.138964	0.246774	0.002070
26	speech	0.138056	0.245161	0.002070
27	days	0.148955	0.027419	0.306418
28	applications	0.134423	0.238709	0.002070
29	fax : +	0.134423	0.238709	0.002070
30	development	0.133514	0.237096	0.002070
31	removed from	0.094459	0.001612	0.215320
32	never	0.094459	0.001612	0.215320
33	for you	0.166212	0.040322	0.329192
34	today	0.160762	0.037096	0.320910
35	on your	0.092643	0.001612	0.211180
36	now !	0.091734	0.001612	0.209109
37	only $	0.091734	0.001612	0.209109
38	1999	0.128973	0.229032	0.002070
39	. you	0.218891	0.079032	0.399585
40	thousands	0.090826	0.001612	0.207039
41	papers	0.127157	0.225806	0.002070
42	tel	0.125340	0.222580	0.002070
43	you can	0.265213	0.117741	0.455486
44	linguistics	0.123524	0.219354	0.002070
45	pay	0.087193	0.001612	0.198757
46	! this	0.087193	0.001612	0.198757
47	be removed	0.087193	0.001612	0.198757
48	save	0.086285	0.001612	0.196687
49	. you can	0.086285	0.001612	0.196687
50	000	0.170753	0.05	0.327122

linear discriminant, a function of the following form, where the weights w_i and w_0 can be expressed in terms of the probabilities $P(C)$ and $P(X_i \mid C)$.[13]

$$f(\vec{x}) = \sum_{i=1}^{m} w_i \cdot x_i + w_0 \qquad (4)$$

The equality $f(\vec{x}) = 0$ corresponds to a hyperplane in the vector space, which in our case aims to separate the legitimate from the spam messages of the training corpus. Hence, when using discrete attributes, an alternative view of Naive Bayes' model is that it is a hyperplane in the vector space.

In the case of continuous attributes, which is more relevant to this paper, each a posteriori probability $P(X_i = x_i \mid C = c)$ can be modeled by a normal (Gaussian) probability density function $g(x_i; \mu_{i,c}, \sigma_{i,c})$.[14] Thus, equation (3) becomes:

$$P(C = c \mid \vec{X} = \vec{x}) = \frac{P(C = c) \cdot \prod_{i=1}^{m} g(x_i; \mu_{i,c}, \sigma_{i,c})}{\sum_{c' \in \{c_L, c_S\}} P(C = c') \cdot \prod_{i=1}^{m} g(x_i; \mu_{i,c'}, \sigma_{i,c'})} \qquad (5)$$

$$g(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (6)$$

The mean and typical deviation of each density function are estimated from the training corpus. The resulting classifier is a quadratic function of the attributes, which is reduced to a linear discriminant under the assumption of homoscedacity, i.e., same variance for all categories [Duda and Hart 1973]. The latter assumption, however, is uncommon in the Naive Bayes literature, and is not adopted in the implementation of Naive Bayes that we used. Hence, for our purposes, an alternative view of Naive Bayes' probability model, when using continuous attributes, is that it is a quadratic surface in the vector space, which attempts to separate the legitimate from the spam messages of the training corpus.

3.2.2 *Search method.* In addition to its strong model bias, Naive Bayes has a very strong search bias, as it does not perform any search at all. The parameters of the model, i.e., the a priori probabilities of the categories and the a posteriori probabilities of the attribute values given the category (with continuous attributes, the mean and typical deviation of the corresponding Gaussian distributions), are calculated simply as maximum-likelihood estimates from the training data.

The strong search bias of Naive Bayes makes it very efficient during training: $O(mN)$ time is needed to estimate the a posteriori probabilities, where $m \leq M$ is

[13]A recent proof [Jaeger 2003] shows that the reverse is also true: any linear discriminant can be translated into a probability model of Naive Bayes. In practice, however, the ability of Naive Bayes' probability model to approximate any linear discriminant is restricted by the probability estimation method.

[14]This approach is different from the multinomial model [McCallum and Nigam 1998], which is appropriate only for integer frequency-valued attributes. Schneider [2003] explores alternative event models for Naive Bayes in anti-spam filtering.

the number of selected attributes and N the number of training messages. At run time, Naive Bayes uses formula (5), which requires $O(m)$ computations for each test message.

3.3 Flexible Bayes

3.3.1 *Model representation.* The Flexible Bayes learner [John and Langley 1995] provides an alternative approach to Naive Bayes for continuous attributes. Instead of using a single normal distribution, the a posteriori probabilities of each attribute X_i given the category c are modeled by the average of $L_{i,c}$ normal distributions, with different mean values but common typical deviation:

$$P(X_i = x_i \mid C = c) = \frac{1}{L_{i,c}} \cdot \sum_{l=1}^{L_{i,c}} g(x_i; \mu_{i,c,l}, \sigma_{i,c}). \tag{7}$$

More information on the choice of $L_{i,c}$, and the parameters $\mu_{i,c,l}$ and $\sigma_{i,c}$ will be provided in Section 3.3.2. From the viewpoint of model, the use of a mixture of normal distributions extends significantly the space of classifiers that can be represented: Flexible Bayes' probability model can approximate discriminants of any order, potentially much more complex than the quadratic discriminants of Naive Bayes. That is, it can separate the vector space into areas of rather arbitrary shape, and assign each area to one of the two categories.

3.3.2 *Search method.* The weaker model bias of Flexible Bayes expands considerably its search space. To make the search (i.e., parameter estimation) tractable, the following assumptions are made in [John and Langley 1995], which introduce a search bias:

(1) The number of distributions $L_{i,c}$ that is used to model X_i in equation (7) is set to the number of different values the attribute takes in the training instances of category c. Each of these values is used as the mean $\mu_{i,c,l}$ of a distribution in the corresponding category.

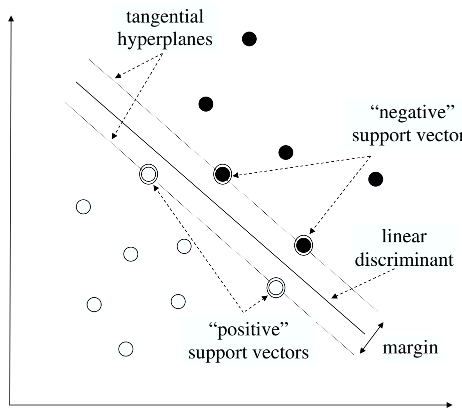
(2) All the distributions in a category c are taken to have the same typical deviation, which is estimated as $\frac{1}{\sqrt{N_c}}$, where N_c is the number of training messages in c.

Equation (7) becomes (8), where $L_{i,c}$ is now the number of different values of X_i in the training data of category c. This modeling is based on Parzen estimation [Fukunaga 1990].

$$P(X_i = x_i \mid C = c) = \frac{1}{L_{i,c}} \cdot \sum_{l=1}^{L_{i,c}} g(x_i; \mu_{i,c,l}, \frac{1}{\sqrt{N_c}}) \tag{8}$$

Despite its weaker model bias, the computational complexity of Flexible Bayes during training is the same as that of Naive Bayes, i.e., $O(mN)$, thanks to the

Fig. 2. Linear discrimination with SVMs in a linearly separable case.



simplifying approach of Parzen estimation. However, there is an increase in the number of computations at run time: in the worst case, where each attribute has a different value in each message, $O(mN)$ computations are needed for each test message, instead of $O(m)$ in the case of Naive Bayes, because of the need to sum the $L_{i,c}$ distributions in equation (8).

3.4 Support Vector Machines

3.4.1 *Model representation.* Support Vector Machines [Cristianini and Shawe-Taylor 2000] are based on the idea that every solvable classification problem can be transformed into a linearly separable one by mapping the original vector space into a new one, using non-linear mapping functions. More formally, SVMs learn generalized linear discriminant functions of the following form:

$$f(\vec{x}) = \sum_{i=1}^{m'} w_i \cdot h_i(\vec{x}) + w_0 \tag{9}$$

where m' is the dimensionality of the new vector space, and $h_i(\vec{x})$ are the non-linear functions that map the original attributes to the new ones. The higher the order of the $h_i(\vec{x})$ functions, the less linear the resulting discriminant. The type of $h_i(\vec{x})$ functions that can be used is limited indirectly by the algorithm's search method (Section 3.4.2), but the exact choice is made by the person who configures the learner for a particular application. The function $f(\vec{x})$ is not linear in the original vector space, but it is linear in the transformed one.

3.4.2 *Search method.* Given the functions $h_i(\vec{x})$, which translate the original problem into one that is more likely to be linearly separable, the search space for SVMs is the set of linear discriminant functions, or else hyperplanes, in the new vector space. The search method of SVMs aims to select the hyperplane that separates with the maximum distance the training instances (messages) of the two categories (Figure 2). This target hyperplane is found by selecting two parallel hyperplanes that are each tangent to a different category, i.e., they include at least one training instance of a different category, while providing perfect separation between all the training instances of the two categories. The training instances that lie on, and thus define, the two tangential hyperplanes are the *support vectors*. The distance between the two tangential hyperplanes is the *margin*. Once the margin has been maximized, the target hyperplane is in the middle of the margin.

More formally, let us first consider the case where the mapping functions map each $\vec{x}$ to itself (i.e., there is no transition to a new vector space) and the problem is linearly separable. The target hyperplane will satisfy equation (10), where $\vec{w}$ and w_0 can be rescaled at will, so that the instances closest to the target hyperplane, the support vectors, will satisfy $|\vec{w} \cdot \vec{x} + w_0| = 1$. Then, each of the tangential hyperplanes satisfies one of equations (11) and (12), and the distance between the two hyperplanes, i.e., the margin, is $2/||\vec{w}||$, where $|| \cdot ||$ is the Euclidean norm.

$$\sum_{i=1}^{m} w_i \cdot x_i + w_0 = \vec{w} \cdot \vec{x} + w_0 = 0 \tag{10}$$

$$\sum_{i=1}^{m} w_i \cdot x_i + w_0 = \vec{w} \cdot \vec{x} + w_0 = 1 \tag{11}$$

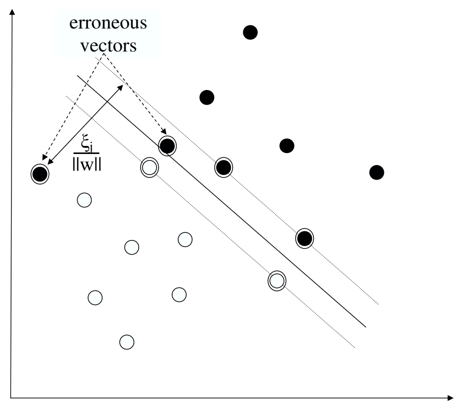
$$\sum_{i=1}^{m} w_i \cdot x_i + w_0 = \vec{w} \cdot \vec{x} + w_0 = -1 \tag{12}$$

Minimizing $||\vec{w}||^2/2$, subject to the constraint that all the training instances are classified correctly, leads to the maximum-margin solution.

In order to handle situations where no hyperplane can fully separate the training instances of the two categories and allow for misclassified training instances, instead of $||\vec{w}||^2/2$ we minimize $||\vec{w}||^2/2 + C \cdot \sum_j \xi_j$, where there is an ξ_j for each training instance $\vec{x}_j$. Intuitively, each ξ_j measures how far $\vec{x}_j$ is from where it should be; more precisely, the distance is $\xi_j/|\vec{w}|$. For training instances that are both classified correctly by the linear discriminant of equation (10) and fall outside the margin area between the two tangential hyperplanes, $\xi_j = 0$. For all the other training instances, $\xi_j/||\vec{w}||$ is the distance of the instance from the tangential hyperplane of its category (Figure 3). For training instances that are correctly classified by the linear discriminant but fall in the margin, $0 \leq \xi_j \leq 1$. C is a user-defined parameter that determines the amount of error to be tolerated. The smaller the value of C is, the less important misclassifications become in the choice of linear discriminant. Following extensive experimentation, we chose $C = 1$.

In the non-linearly separable case, the optimization problem above is difficult

Fig. 3. Linear discrimination with SVMs in a case that is not linearly separable.



to handle directly. Instead, it can be transformed [Burges 1998] into its dual Lagrangian formulation, which maximizes quantity (13) under constraints (14), where $\vec{x}_j, \vec{x}_k$ are all the training instances, and $y_j, y_k \in \{1, -1\}$ depending on the categories of $\vec{x}_j, \vec{x}_k$; the category whose tangential hyperplane satisfies equation (11) is represented by 1, and the other one by -1. The multipliers a_j, known as Langrange multipliers, are the parameters that need to be estimated by the search method.

$$L_D = \sum_j \alpha_j - \frac{1}{2} \cdot \sum_{j,k} \alpha_j \cdot \alpha_k \cdot y_j \cdot y_k \cdot \vec{x}_j \cdot \vec{x}_k \tag{13}$$

$$0 \leq a_j \leq C \text{ and } \sum_j a_j \cdot y_j = 0 \tag{14}$$

For the dual Langrangian problem to be equivalent to the primal one, a set of conditions, known as Karush-Kuhn-Tucker (KKT) conditions, need to be satisfied [Burges 1998]. Constraints (14) and the KKT conditions lead to the following constraints that guide the search towards the optimal linear discriminant:

(1) $a_j = 0$, for all the training instances $\vec{x}_j$ that are both correctly classified and fall outside the marginal area $(y_j \cdot (\vec{w} \cdot \vec{x}_j + w_0) > 1)$,

(2) $0 < a_j < C$, for the support vectors $(y_j \cdot (\vec{w} \cdot \vec{x}_j + w_0) = 1)$,

(3) $a_j = C$, for the instances in the marginal area and those that are misclassified $(y_j \cdot (\vec{w} \cdot \vec{x}_j + w_0) < 1)$.

When the mapping functions $h_i(\vec{x})$ cause a transition to a new vector space, $\vec{h}(\vec{x}) = \langle h_1(\vec{x}), \ldots, h_i(\vec{x}), \ldots, h_{m'}(\vec{x}) \rangle$ has to be used instead of $\vec{x}$ in the equations above. The most interesting property of the dual Lagrangian formulation is that the quantity to be maximized, now equation (15), is expressed in terms of a dot product between instances in the new vector space. Equation (15) can be rewritten as (16), where $K(\vec{x}_j, \vec{x}_k)$ is a function, called *kernel*, that computes the dot product.

$$L_D = \sum_j \alpha_j - \frac{1}{2} \cdot \sum_{j,k} \alpha_j \cdot \alpha_k \cdot y_j \cdot y_k \cdot \vec{h}(\vec{x}_j) \cdot \vec{h}(\vec{x}_k) \tag{15}$$

$$L_D = \sum_j \alpha_j - \frac{1}{2} \cdot \sum_{j,k} \alpha_j \cdot \alpha_k \cdot y_j \cdot y_k \cdot K(\vec{x}_j, \vec{x}_k) \tag{16}$$

In fact, we can use any function $K(\vec{x}_j, \vec{x}_k)$, provided that it is indeed a kernel, i.e., that it can be proven to compute the dot product of $\vec{x}_j, \vec{x}_k$ in some new vector space, without defining that new vector space and the mapping functions explicitly.

A range of kernel functions have been used in the literature, e.g., polynomial kernels, such as $K(\vec{x}_j, \vec{x}_k) = (x_0 + \vec{x}_j \cdot \vec{x}_k)^n$. The choice of kernel affects the model bias of the algorithm. For instance, the higher the degree n of a polynomial kernel, the higher the order of the discriminant function that can be discovered in the original vector space. However, research in text classification has shown that simple linear SVMs usually perform as well as non-linear ones [Joachims 1998]. This was also observed in our experiments, where after experimenting with polynomial kernels of various degrees, we finally decided to use a linear SVM, i.e., an SVM that uses a dot product in the original vector space as its kernel.

The solution of the dual Lagrange maximization problem requires numerical optimization, usually through a quadratic programming algorithm. The optimization algorithm we used is called Sequential Minimal Optimization, and avoids some of the computational cost of quadratic programming by breaking the problem down into smaller ones that can be solved analytically; see Platt [1999] for details.

Once the Lagrange multipliers have been estimated, the classification of an unseen instance $\vec{x}$ into category $+1$ or -1 at run time follows the sign of function (17), where $\vec{x}_j$ are the training instances with $a_j > 0$ (ideally only the support vectors). The constant w_0 can be computed from the KKT conditions.

$$f(\vec{x}) = \vec{w} \cdot \vec{h}(\vec{x}) + w_0 = \sum_j \alpha_j \cdot y_j \cdot K(\vec{x}_j, \vec{x}) + w_0, \tag{17}$$

The SMO algorithm is relatively time and memory efficient, in comparison to other optimization algorithms for SVMs. Nevertheless, this efficiency is not reflected in its worst-case computational complexity. During training, SMO iteratively identifies "good pairs of instances", which leads to a worst-case complexity that is quadratic to the number of instances, i.e., $O(mN^2)$. This is misleading, however, since the choice of instance pairs is highly optimized in the average case, emphasizing the role of the iterative optimization process in the computational cost of the algorithm. Clearly, the number of iterations is hard to quantify in terms of N, in order to be included meaningfully in the worst-case estimate. At run time, the complexity of

SMO is $O(mN)$. However, this figure corresponds to the worst case, where all the training instances receive $a_j > 0$; this is a very unusual situation, which does not seem to occur in text classification. Furthermore, with linear SVMs, i.e., if there is no transition to a new vector space, it is possible to improve this estimate to $O(m)$.

3.5 LogitBoost with regression stumps

3.5.1 *Model representation.* LogitBoost [Friedman et al. 2000] belongs in the class of boosting algorithms, which, like SVMs, learn generalized linear discriminants of the form of equation (18).

$$f(\vec{x}) = \sum_{i=1}^{m'} w_i \cdot h_i(\vec{x}) + w_0 \tag{18}$$

In boosting algorithms, however, the mapping functions $h_i(\vec{x})$ are themselves learnt from data by another learning algorithm, known as *weak learner*. Typically, the same weak learner is used to learn all the mapping functions, but at each i, the learning data are modified, leading to a different mapping function $h_i(\vec{x})$. As with SVMs, the resulting discriminant is generally non-linear in the original vector space.

A common weak learner is decision stump induction [Holte 1993], which constructs a one-level decision tree that uses a single attribute from the original attribute set to classify the instance $\vec{x}$ to one of the two categories. In the case of continuous attributes, as those used in this paper, the decision tree is a threshold function on one of the original attributes. LogitBoost requires that the weak learner perform regression, rather than classification; i.e., the weak learner $h_i(\vec{x})$ is required to approximate a function $z_i(\vec{x})$. In that case, the decision stumps become regression stumps of the following form, where $a_i(\vec{x})$ is the value of the attribute checked by the stump, and t_i is a threshold:

$$h_i(\vec{x}) = \begin{cases} r_i, & \text{if } a_i(\vec{x}) > t_i \\ q_i, & \text{if } a_i(\vec{x}) \leq t_i \end{cases}, \tag{19}$$

$$\text{where } r_i = \frac{1}{|R|} \sum_{\vec{x}_j \in R} z_i(\vec{x}_j), \quad q_i = \frac{1}{|Q|} \sum_{\vec{x}_j \in Q} z_i(\vec{x}_j),$$

$$\text{and } R = \{\vec{x}_j \mid a_i(\vec{x}_j) > t_i\}, \quad Q = \{\vec{x}_j \mid a_i(\vec{x}_j) \leq t_i\}.$$

The threshold t_i partitions the training instances in two parts, R and Q. The values r_i, q_i of the regression stump are the mean values of $z_i(\vec{x}_j)$ in the two parts.

3.5.2 *Search method.* The mapping functions $h_i(\vec{x})$ are learnt by applying iteratively (for $i = 1, \ldots, m'$) the weak learner, in our case regression stump induction, to an enhanced form of the training set, where each training instance $\vec{x}_j$ carries a weight $v_i(\vec{x}_j)$. At each iteration, the weights of the training instances are updated, and, hence, applying the weak learner leads to a different mapping function $h_i(\vec{x})$.

This iterative process is common to all boosting methods, where each $h_i(\vec{x})$ can be thought of as a weak classifier that specializes on classifying correctly training instances that the combination of the previous weak classifiers $h_k(\vec{x})$ $(k = 1, \ldots, i-1\})$ either misclassifies or places close to the classification boundary. This is similar to the behavior of SVMs, which focus on instances that are misclassified or support the tangential hyperplanes. At run time, equation (18) combines the opinions of the weak classifiers by computing a weighted sum of their confidence scores. LogitBoost sets $w_0 = 0$ and $w_i = 0.5$ (equal weights to all weak classifiers), leading to discriminants of the form:

$$f(\vec{x}) = \frac{1}{2} \cdot \sum_{i=1}^{m'} h_i(\vec{x}) \tag{20}$$

Unlike other boosting methods, LogitBoost modifies the target function $z_i(\vec{x}_j)$, in addition to the instance weights $v_i(\vec{x}_j)$, during training. At iteration i, the weight and the target value of each training instance $\vec{x}_j$ is computed as in (21) and (22), where c is one of the two categories, treated as the positive one, and $y(\vec{x}_j) = 1$ if $\vec{x}_j$ belongs in c, and $y(\vec{x}_j) = 0$ otherwise.

$$v_i(\vec{x}_j) = P(C = c \mid \vec{x}_j) \cdot (1 - P(C = c \mid \vec{x}_j)) \tag{21}$$

$$z_i(\vec{x}_j) = \frac{y(\vec{x}_j) - P(C = c \mid \vec{x}_j)}{v_i(\vec{x}_j)} \tag{22}$$

The probability $P(C = c \mid \vec{x}_j)$ is estimated by applying a sigmoid function, known as Logit transformation, to the response $f(\vec{x}_j)$ of the weak classifiers that have been learnt so far, as in (23). The sigmoid function maps $f(\vec{x}_j)$ on the $(0, 1)$ interval.

$$P(C = c \mid \vec{x}_j) = \frac{e^{f(\vec{x}_j)}}{1 + e^{f(\vec{x}_j)}} \text{ , where } f(\vec{x}_j) = \frac{1}{2} \cdot \sum_{k=1}^{i-1} h_k(\vec{x}_j). \tag{23}$$

The target function $z_i(\vec{x}_j)$ provides a weighted estimate of the error of the current combined classifier that $h_i(\vec{x})$ is intended to improve. The weight in (21) is largest when $P(C = c \mid \vec{x}_j) = 1 - P(C = c \mid \vec{x}_j) = 0.5$ (maximum uncertainty regarding the category of $\vec{x}_j$), and becomes smaller as $P(C = c \mid \vec{x}_j)$ approaches its maximum or minimum value (maximum certainty). Hence, it amplifies more the error of (22) when the current classifier is more certain of its decision (smaller denominator).

Once the target function $z_i(\vec{x}_j)$ and the weights $v_i(\vec{x}_j)$ of iteration i have been computed, the parameters of the new weak classifier $h_i(\vec{x})$, in our case the threshold t_i and the attribute α_i that the stump examines, are chosen using weighted least-square regression on the training data, with $z_i(\vec{x}_j)$ as the working response. This leads to the regression stump that models best the errors of the current classifier on the training set, a process known as fitting of residuals.

The search for the optimal stump considers only threshold values that are values of the stump's attribute in the training data. In the worst case, the attribute has a different value in each training instance. Hence, it takes $O(mN^2)$ time at each iteration to consider $O(mN)$ candidate stumps and compute the responses over the N training instances, and the overall complexity of LogitBoost during learning is $O(m'mN^2)$. At run time, computing the response of each stump takes constant time, and, thus, the complexity of LogitBoost is $O(m')$. In our experiments, we set $m' = 20$, as the LogitBoost implementation that we used was too slow during training to explore larger numbers of iterations.

3.6 Comparison of the learning methods

This section summarizes the main differences and similarities of the four learning algorithms, with respect to their model and search biases. It also compares their computational complexity at training and run time.

3.6.1 *Model representation.* As explained at the beginning of Section 3, the choice of model introduces a bias, which restricts the space of classifiers that a learning algorithm considers. The more restrictive the model representation is, the strongest the model bias. The four algorithms vary significantly in terms of their model bias. Naive Bayes has the strongest one, restricting its search, in the case of continuous attributes, to quadratic discriminants. The other three algorithms can all learn higher-order non-linear discriminants. SVMs and LogitBoost both learn generalized linear discriminants in a transformed vector space, which are generally non-linear in the original space; they use, however, very different mapping functions between the two spaces. Flexible Bayes uses a mixture of Gaussian distributions, which allows it to approximate discriminants of any order.

The common feature of the four algorithms is that they can all learn non-linear discriminants. However, the need for non-linearity has been challenged in the literature, especially for text classification [Joachims 1998; Zhang et al. 2003]. The skepticism is based on the very high dimensionality and sparseness of most text classification training sets, which can often be modeled adequately by linear discriminants. In classification problems of the latter kind, a stronger model bias that limits the search space to linear discriminants may help the search method find a good discriminant, leading to improved classification accuracy.

3.6.2 *Search method.* The method that each algorithm uses to search the space of classifiers it can learn introduces its own bias. The two Bayesian methods have the strongest search biases. With continuous attributes, Naive Bayes simply uses maximum likelihood to estimate the mean and typical deviation of the Gaussian distributions it uses, in effect performing no search. Flexible Bayes makes strong assumptions about the number and parameters of its Gaussian distributions, which,

Table III. Computational complexity of the four learning algorithms. N is the number of training messages, $m \leq M$ the number of selected attributes, and m' the number of iterations in LogitBoost.

algorithm	training	classification
Naive Bayes	$O(mN)$	$O(m)$
Flexible Bayes	$O(mN)$	$O(mN)$
SVM with SMO	$O(mN^2)$	$O(mN)$
linear SVM with SMO	$O(mN^2)$	$O(m)$
LogitBoost with regression stumps	$O(m'mN^2)$	$O(m')$

again, allow it to estimate the parameters with no search. The other two algorithms search the space of classifiers by performing optimization. They both focus on training instances that are misclassified or are close to the classification boundary, which can also be seen as introducing a strong search bias.

3.6.3 *Computational Complexity.* Table III summarizes the computational complexity estimates of the previous sections. At training, Naive Bayes and Flexible Bayes appear to be the best. At run time, Naive Bayes and LogitBoost with regression stumps have the important property that their complexities do not depend on the number of training messages, while the same is true for linear SVMs. Overall, then, Naive Bayes appears to be the best in terms of computational complexity. One should keep in mind, however, that these are worst-case estimates. In practice, the number of steps each algorithm takes may grow at more conservative rates. We return to this point when presenting the results of our corpus experiments in Section 5.

4. COST SENSITIVE CLASSIFICATION

Mistakenly treating a legitimate message as spam can be a more severe error than treating a spam message as legitimate, i.e., letting it pass the filter. This section explains how we took this unbalanced misclassification cost into account in our experiments. We first discuss different usage scenarios for anti-spam filtering, and how each can be captured in a cost matrix. We then turn to the issue of making learning algorithms sensitive to a cost matrix. Finally, we discuss why evaluation measures that are in accordance to the cost matrix are needed, and present the evaluation measures that we used.

4.1 Cost scenarios

Let $L \rightarrow S$ and $S \rightarrow L$ denote the two error types, whereby a legitimate message is classified as spam, and vice versa, respectively. Invoking a decision-theoretic notion of cost, let us assume that $L \rightarrow S$ is λ times more costly than $S \rightarrow L$, where λ depends on the usage scenario; for example, whether the filter deletes messages it classifies as spam, or simply flags them as low-priority. More precisely, we use the

Table IV. The cost matrix used in this paper.

	classified as legitimate	classified as spam
legitimate message	$c(L \rightarrow L) = 0$	$c(L \rightarrow S) = \lambda$
spam message	$c(S \rightarrow L) = 1$	$c(S \rightarrow S) = 0$

cost matrix of Table IV, where $L \rightarrow L$ and $S \rightarrow S$ denote the cases where the filter classifies correctly a legitimate or spam message, respectively. Here, cost is intended to reflect the effort that users waste to recover from failures of the filter.[15] Correctly classifying a message is assigned zero cost, since no user effort is wasted. Misclassifying a spam message ($S \rightarrow L$) is assigned unary cost, and misclassifying a legitimate message ($L \rightarrow S$) is taken to be λ times more costly.[16] This cost model assumes that the effort to recover from the misclassification of a legitimate message is always the same, regardless of its sender or subject. This is a simplification that will allow us to reach some useful initial conclusions.

In the experiments of this paper, we used two different usage scenarios. In the first one, the anti-spam filter simply flags messages it considers to be spam, without removing them from the user's mailbox, to help the user prioritize the reading of incoming messages. In this case, $\lambda = 1$ seems reasonable, since none of the two error types ($L \rightarrow S$ or $S \rightarrow L$) is graver than the other. The second scenario assumes that messages classified as spam are returned to the sender. An extra paragraph in the returned message explains that the message was blocked by a filter, and asks the original sender to repost the message, this time including a special string in the subject that instructs the filter to let the message pass. The string can be the answer to a frequently changing user-specific riddle (e.g., "the capital of France"), which spamming software cannot guess. In this scenario, λ is set to 9, reflecting the assumption that, if the sender's extra work is counted, recovering from a blocked legitimate message, i.e., reposting the message as instructed, requires as much extra effort as deleting manually approximately 9 spam messages that passed the filter.

In previous work [Androutsopoulos et al. 2000a; 2000b; 2000c], we had also considered a scenario where messages classified as spam were deleted without further action. In that scenario, the cost of recovering from an accidentally misclassified legitimate message was very high; we had used $\lambda = 999$. However, our previous results indicated that the third scenario was too difficult for both learning-based filters and filters with hand-crafted rules, and we decided not to study it further.

[15] In a more general case, cost could take into account the wasted resources of the Internet provider.
[16] Kolcz and Alspector [2001] argue for a more fine-grained specification of the cost of erroneously blocked legitimate messages. They divide the legitimate category to subcategories, such as business mail, or sensitive personal mail, and propose different costs for each subcategory.

4.2 Cost-sensitive learning

We now examine how learning algorithms can be made sensitive to a cost matrix. The optimal strategy [Duda and Hart 1973] is to classify an incoming message represented by $\vec{x}$ as spam iff the expected cost of classifying it as legitimate exceeds that of classifying it as spam, i.e., iff inequality (24) holds.

$$P(C = c_S \mid \vec{x}) \cdot c(S \rightarrow L) + P(C = c_L \mid \vec{x}) \cdot c(L \rightarrow L) > \\ P(C = c_L \mid \vec{x}) \cdot c(L \rightarrow S) + P(C = c_S \mid \vec{x}) \cdot c(S \rightarrow S) \quad (24)$$

With classifiers whose decisions are accompanied by confidence scores, the scores can be used as estimates of the probabilities in (24). Let $W_L(\vec{x})$ and $W_S(\vec{x})$ be the confidence scores of the classifier that the incoming message is legitimate or spam, respectively. Then, provided that the confidence scores are good estimates of the corresponding probabilities, inequality (24) becomes (25).

$$W_S(\vec{x}) \cdot c(S \rightarrow L) + W_L(\vec{x}) \cdot c(L \rightarrow L) > W_L(\vec{x}) \cdot c(L \rightarrow S) + W_S(\vec{x}) \cdot c(S \rightarrow S) \quad (25)$$

According to the cost matrix of Table IV, this is equivalent to $W_S(\vec{x}) > W_L(\vec{x}) \cdot \lambda$. Hence, assuming that both confidence scores are within the $(0, 1)$ interval and their sum is 1, the best strategy is to classify the message as spam iff the two equivalent criteria (26) are satisfied, where t is a classification threshold.

$$\frac{W_S(\vec{x})}{W_L(\vec{x})} > \lambda, \text{ or } W_S(\vec{x}) > t, \text{ with } t = \frac{\lambda}{1 + \lambda} \quad (26)$$

Unfortunately, many learning algorithms were designed with cost-insensitive tasks in mind. In cost-insensitive tasks, returning confidence scores that are good estimates of the corresponding probabilities is not crucial, as long as the correct category is assigned the greatest confidence score. As a result, the learned classifiers often produce unreliable estimates, e.g., based on small samples, or no estimates at all. Hence, relying on criteria (26) is not always the best strategy.

An alternative strategy is to classify each unseen instance to the category with the highest confidence score, as in a cost-insensitive setting, but to weigh the training instances according to the misclassification cost of the category they belong to. This biases the learner towards the category with the highest misclassification cost, helping it avoid the most costly errors. In our case, legitimate training messages are assigned λ times greater weights than spam ones to avoid $L \rightarrow S$ errors that cost λ times more than $S \rightarrow L$ ones. The implementations of Naive Bayes and Flexible Bayes that we used can weigh the training instances according to a formula presented by Ting [1998]. In our case, this assigns to each spam training instance a weight $w_S = \frac{N_S + N_L}{N_S + \lambda \cdot N_L}$, where N_S and N_L are the number of spam and legitimate training instances, respectively; legitimate messages are assigned a weight $w_L =$

$\lambda \cdot w_S$. The advantage of this weighting scheme is that the total weight of the instances remains $N_S + N_L$, irrespective of the value of λ.

The SVM and LogitBoost implementations that we used cannot handle weighted training instances. Instead, we used weighted resampling to simulate training instance weighting. This generates a new training set of the same size as the original one by randomly resampling the original instances according to their weights. In our case, this means that some messages, mostly legitimate ones, appear several times in the new set, and others, mostly spam ones, do not appear at all.

Preliminary experiments with $\lambda = 9$ indicated that LogitBoost performed better with criteria (26) than with weighted resampling. In contrast, the SVM performed better with weighted resampling, and Flexible Bayes performed better with weighted training instances. With all three algorithms, we chose the method that led to the best results.[17] In the case of Naive Bayes, weighting the training instances is equivalent to applying criteria (26), because all the training instances in a particular category receive the same weights. Hence, the means and typical deviations $\mu_{i,c}$ and $\sigma_{i,c}$ in each category (equation (5)) are not affected, and the a posteriori probabilities (Gaussian distributions) remain the same. The weighting affects only the a priori probability of the legitimate category multiplying it by λ, which is equivalent to using criteria (26).

4.3 Cost-sensitive evaluation

We have already considered ways to make the classifiers sensitive to the cost difference between the two types of misclassification errors. In a similar manner, this cost difference must be taken into account when evaluating the performance of anti-spam filters. In classification tasks, performance is usually measured in terms of *accuracy* (*Acc*) or *error rate* ($Err = 1 - Acc$). Let R_L and R_S be the numbers of legitimate and spam messages to be classified at run time, $|L \to S|$ and $|S \to L|$ be counters of the $L \to S$ and $S \to L$ errors at run time, and $|S \to S|$ and $|L \to L|$ be counters of the correctly classified spam and legitimate messages, respectively. Accuracy and error rate are defined as follows:

$$Acc = \frac{|L \to L| + |S \to S|}{R_L + R_S} \quad \text{and} \quad Err = \frac{|L \to S| + |S \to L|}{R_L + R_S} \ .$$

To make *Acc* and *Err* sensitive to the cost difference between the two error types, we treat, for evaluation purposes, each legitimate message in the testing corpus as if it were λ messages. Misclassifying a legitimate message counts as λ errors, while classifying it correctly counts as λ successes. Spam messages are treated as single messages. This is similar to instance weighting during training, and leads to the

[17]We also experimented with MetaCost [Domingos 1999], but found it computationally expensive, without significant gains in classification performance. This agrees with Hidalgo [2002].

following definitions of *weighted accuracy* and *weighted error rate*:

$$WAcc = \frac{\lambda \cdot |L \rightarrow L| + |S \rightarrow S|}{\lambda \cdot R_L + R_S} \ , \ WErr = 1 - WAcc = \frac{\lambda \cdot |L \rightarrow S| + |S \rightarrow L|}{\lambda \cdot R_L + R_S} \ .$$

In terms of cost, the numerator of *WErr* above is equal to the total cost incurred by using the filter on the $R_L + R_S$ messages, while the denominator is a normalizing factor equal to the incurred cost of the worst possible filter, which misclassifies all the messages. *WAcc* is simply the complement of the normalized incurred cost.

The *Acc* and *Err* scores, or their weighted versions, are often misleadingly high or low, respectively. To get a clearer picture of the classifier's performance, it is common to compare them to those of a simplistic baseline. In this paper, the baseline is the case where no filter is used: legitimate messages are (correctly) never blocked, and spam messages (mistakenly) always pass the filter. Its scores are:

$$WAcc^b = \frac{\lambda \cdot R_L}{\lambda \cdot R_L + R_S} \quad \text{and} \quad WErr^b = \frac{R_S}{\lambda \cdot R_L + R_S} \ .$$

For the benefit of readers familiar with information retrieval and extraction, the results will also be shown in terms of *recall* (R) and *precision* (P), defined below:

$$R = \frac{|S \rightarrow S|}{R_S} \quad \text{and} \quad P = \frac{|S \rightarrow S|}{|S \rightarrow S| + |L \rightarrow S|} \ .$$

Recall measures how many of the spam messages the filter managed to block (intuitively, its effectiveness), whereas precision measures how many of the messages that the filter classified as spam were indeed spam (intuitively, its safety). Despite their intuitiveness, it is difficult to compare the performance of different filters using recall and precision. Each filter yields a different pair of recall and precision scores; and without a single measure, like *WAcc*, that incorporates the cost difference between the two error types, it is difficult to decide which pair is better.[18]

Finally, in order to get a feeling of how well the learning algorithms perform against simple hand-crafted rules that look for particular keywords, the default anti-spam rules of Microsoft Outlook 2000, hereafter *hand-crafted rules*, were also tested.[19] They are 58 rules, looking for particular keywords in the body or header of the messages (e.g., "body contains ',000' and body contains $").

5. CORPUS EXPERIMENTS

We now move on to the presentation of our corpus experiments, which were performed using the PU1, PU2, PU3, and PUA collections. All the experiments were

[18]The F-measure, commonly used in information retrieval and extraction to combine recall and precision, cannot be used here, because its β factor cannot be related to λ. An alternative proposed by Hidalgo [2002] is to plot ROC curves for variable λ values.

[19]The on-line documentation of Microsoft Outlook 2000 points to a file that contains its default anti-spam rules. The results reported in this paper are for a case-insensitive version of these rules, which performed better than the original case-sensitive ones.

performed using stratified 10-fold cross-validation (Section 3.1).

The corpus experiments served several goals. First, to explore the degree to which a learning-based anti-spam filter improves, if at all, upon the baseline (no filter) and the hand-crafted rules of Section 4.3; both cost scenarios of Section 4.1 were considered. A second goal was to investigate if some of the learning algorithms lead to consistently better performance, identifying the conditions under which particular algorithms excel. A third, intertwined goal was to investigate the effect of the following parameters: the size of the attribute set, the nature of the attributes (whether they represent single tokens or sequences of tokens), and the size of the learning corpus. The findings of these experiments guided the design of our prototype filter, that will be presented in Section 6.

5.1 Experiments with 1-gram attributes

The first experiments used only 1-gram attributes, i.e., each attribute corresponded to one token. The best m attributes were retained, as in Section 3.1, with m ranging from 40 to 600 by 40. The experiments were repeated for $\lambda = 1$ and $\lambda = 9$.

Figure 4 shows the *WAcc* results for $\lambda = 1$.[20] All the learning algorithms clearly outperform both the baseline and the hand-crafted rules, even in PU2 where the baseline is very high. There is no clear winner, however, among the learning algorithms: the SVM classifier is the only one that is consistently among the two best, but the differences in *WAcc* among the four learnt classifiers are generally small, and their rankings vary across the collections. Readers wishing to implement real-life filters for this cost scenario are, therefore, advised to select among the four algorithms taking into account other factors, such as their complexities (Table III) and the availability of efficient implementations, rather than their rankings in Figure 4.

As an indication of the computational efficiency of the particular implementations that we used, Figure 5 shows the average learning and classification times per training and test message, respectively, as recorded in our experiments on PUA for $\lambda = 1$.[21] In almost all cases, there is a significant increase in both learning and classification time as more attributes are retained (notice the logarithmic vertical scale). Hence, in a real-life application one should be skeptical about using large attribute sets, especially since, as shown in Figure 4, retaining large numbers of attributes does not always lead to significant improvements in accuracy. We will explore this point further in Section 5.2.

Figure 5 implies that there is a tradeoff between training and classification time. The training time of LogitBoost was clearly the worst, by at least an order of

[20]The results of Naive Bayes on PU1 differ from those published in our earlier work, because we now use frequency-valued attributes.

[21]All the corpus experiments were performed on a computer with a 1.7 GHz Intel Pentium 4 processor and 512 MB RAM running Microsoft Windows 2000 and Java SDK v.1.4.

Fig. 4. *WAcc* results with varying number of 1-gram attributes, for $\lambda = 1$.

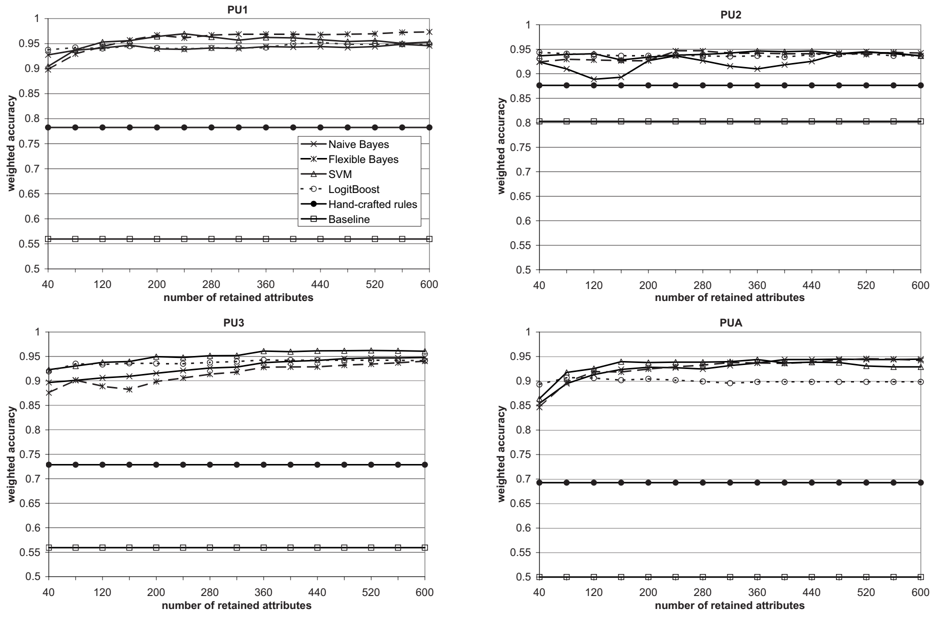
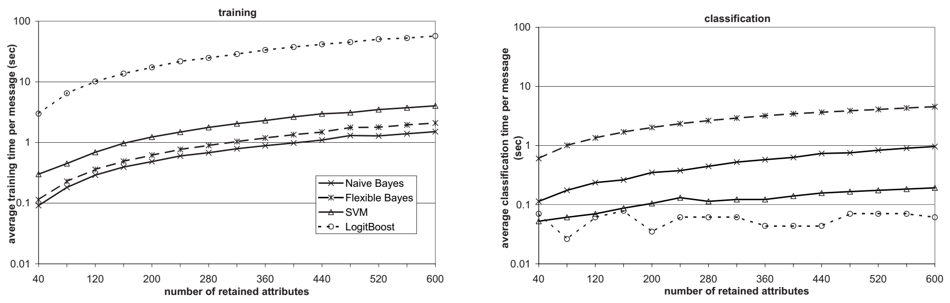


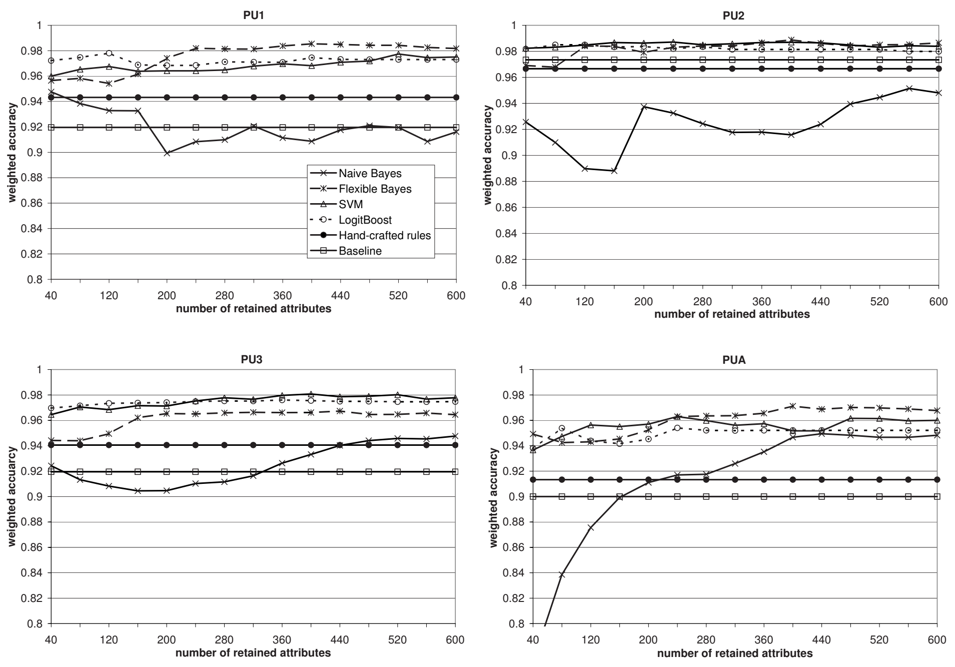
Fig. 5. Average training and classification time on PUA, with 1-gram attributes, for $\lambda = 1$.



magnitude, making it almost impractical to use. Its classification speed, however, was the best. Similarly, the training speed of SVM was the second worst, while its classification speed was the second best. The Bayesian classifiers were the fastest in terms of training time, and the worst in terms of classification time; Flexible Bayes was significantly slower than Naive Bayes in classification, because of the additional computations needed to average over its kernels (Section 3.3.2).

The recorded times appear to be compatible with, but generally less pessimistic than the worst-case complexity figures of Table III, which predicted, for varying m, constant classification time for LogitBoost, and linear time in all other cases.

Fig. 6. *WAcc* results with varying number of 1-gram attributes, for $\lambda = 9$.



However, the important differences in the efficiency of the algorithms, both during training and testing, do not seem to be fully predictable by the complexity estimates. As a result, we conclude that worst-case predictions are an insufficient guide for the choice of learning algorithm.

Moving on to the $\lambda = 9$ scenario, Figure 6 shows the corresponding *WAcc* results for 1-gram attributes. It can be seen from Figure 6 that SVM, Flexible Bayes, and LogitBoost consistently outperform both the baseline and the hand-crafted rules, even in PU2 where the baseline is extremely high due to the large legitimate-to-spam ratio and the fact that each legitimate message is counted nine times when computing *WAcc*. In contrast, the performance of Naive Bayes is much worse, very often below the baseline; this is mostly the effect of a higher number of $L \rightarrow S$ errors, which are penalized more heavily than $S \rightarrow L$ ones. Flexible Bayes performs clearly better than Naive Bayes, suggesting that the normal distribution assumption in Naive Bayes is violated.

Table V sheds more light on the experiments with 1-gram attributes: it shows the *WAcc* results that were obtained in both cost scenarios on the four collections with 600 attributes, along with the corresponding precision and recall figures. Naive Bayes achieves the best recall in all cases, at the expense of generally lower precision;

Table V. Precision, recall, and *WAcc* (%), with 600 1-gram attributes.

	$\lambda = 1$			$\lambda = 9$		
learner	precision	recall	*WAcc*	precision	recall	*WAcc*
PU1						
baseline	–	–	55.96	–	–	91.96
hand-crafted rules	95.15	53.01	78.25	95.15	53.01	94.32
Naive Bayes	89.58	99.38	94.59	89.81	98.75	91.61
Flexible Bayes	96.92	97.08	97.34	99.03	84.79	98.17
SVM	93.96	95.63	95.32	97.97	85.83	97.50
LogitBoost	95.22	93.13	94.86	98.02	81.46	97.03
PU2						
baseline	–	–	80.28	–	–	97.34
hand-crafted rules	85.33	45.39	87.62	85.33	45.39	96.65
Naive Bayes	80.77	90.00	93.66	82.96	90.71	94.80
Flexible Bayes	90.57	79.29	94.22	97.78	62.14	98.65
SVM	88.71	79.29	93.66	97.50	52.14	98.39
LogitBoost	89.46	77.86	93.66	93.82	62.86	97.99
PU3						
baseline	–	–	55.93	–	–	91.95
hand-crafted rules	96.31	40.03	72.87	96.31	40.03	94.05
Naive Bayes	93.59	94.84	94.79	93.59	94.78	94.76
Flexible Bayes	95.78	90.55	94.04	99.34	59.29	96.44
SVM	96.48	94.67	96.08	98.13	87.75	97.78
LogitBoost	94.31	92.42	94.14	98.65	78.52	97.47
PUA						
baseline	–	–	50.00	–	–	90.00
hand-crafted rules	92.97	41.68	69.26	92.97	41.68	91.33
Naive Bayes	95.11	94.04	94.47	95.11	94.04	94.82
Flexible Bayes	96.75	91.58	94.21	98.75	77.19	96.77
SVM	92.83	93.33	92.89	97.10	82.11	96.00
LogitBoost	89.62	90.88	89.82	96.75	75.79	95.21

i.e., it makes few $S \rightarrow L$ errors and more $L \rightarrow S$ ones. In the $\lambda = 1$ scenario, where the two types of error are equally severe, Naive Bayes' high recall balances its overall lower precision. In the $\lambda = 9$ scenario, however, where $L \rightarrow S$ errors are more severe than $S \rightarrow L$ ones, precision is more important than recall; yet, Naive Bayes continues to favor recall, which leads to low *WAcc* scores.

A related observation is that Naive Bayes does not appear to adapt to any significant extent to the different values of λ in the two scenarios, unlike the other learners. The precision and recall of Naive Bayes are almost the same across the two scenarios in all four datasets. This is due to a skew in the probability estimates of Naive Bayes, which causes the values of $P(C = c_S \mid \vec{x})$ to be very close to 0 or 1; hence, moving the classification threshold (Section 4.2) from $t = 0.5$ (when $\lambda = 1$) to $t = 0.9$ (when $\lambda = 9$) has a very limited effect. An alternative, which we have not explored, might be to follow Carreras and Marques [2001], and set the classification threshold experimentally, using a separate validation part of each collection, instead of using the theoretical value $t = \frac{\lambda}{1+\lambda}$ of criteria (26), which is optimal only when the estimates of $P(C|\vec{X})$ of the classifier are accurate. This approach might have allowed Naive Bayes to adapt better to the cost scenarios.

The results we obtained on PU1 are generally worse than those of Carreras and Marquez [2001], who experimented with PU1 and AdaBoost, another boosting algorithm. In the $\lambda = 1$ scenario, Carreras and Marquez reported 97.48% precision and 96.47% recall when using decision stumps as weak learners, with results reaching 98.73% precision and 97.09% recall when using decision trees as weak learners (cf. Table V). For $\lambda = 9$, their best results were 99.08% precision and 89.60% recall ($WAcc$ = 98.58%) with decision stumps, and 99.35% precision and 94.80% recall ($WAcc$ = 99.14%) with decision trees as weak learners. Apart from using a different boosting algorithm and setting the classification threshold experimentally, as discussed above, the other main differences between the setting of our experiments and that of Carreras and Marquez were that the latter performed no attribute selection (they used all the candidate 1-gram attributes of Section 3.1.2) and used hundreds of boosting iterations (their $\lambda = 1$ results above were obtained using 525 and 450 iterations, respectively), compared to the 20 iterations of our experiments with LogitBoost. As shown in Figure 5, the LogitBoost implementation we used is particularly slow during training, and, hence, we were unable to explore if using more iterations would allow LogitBoost to approach the results of Carreras and Marquez. The issue of using much larger attribute sets was explored in the experiments that will be presented in Section 5.2 below.

It is also interesting to observe (Table V) that the hand-crafted rules perform well in terms of precision. Their weakness lies in recall, where their performance is well below that of the learning algorithms. This allows the hand-crafted rules to achieve higher $WAcc$ scores in the $\lambda = 9$ scenario, where precision is more important.

5.2 Experiments with extended attribute sets

In this section we present a second set of experiments, which employed attribute sets that were extended in two ways: first, the attributes now corresponded to n-grams ($n \in \{1, 2, 3\}$), as opposed to using 1-grams only, and, second, the number of retained attributes was larger, up to 3000 attributes, compared to a maximum of 600 attributes in the previous experiments. Note that if an n_1-gram (e.g., "removed from") subsumes an n_2-gram (e.g., "removed"), with $n_1 > n_2$, both can be retained as attributes, provided that their rankings are sufficiently high.

As observed in Section 3.1 (Table II), a large number of 2-grams and 3-grams score high in terms of information gain score. Columns 2–5 of Table VI show that the majority of the 600 n-gram attributes with the highest information gain scores are 2-grams and 3-grams in all four collections. It is, thus, reasonable to expect that using n-grams instead of 1-grams may lead to improved accuracy. The motivation for also

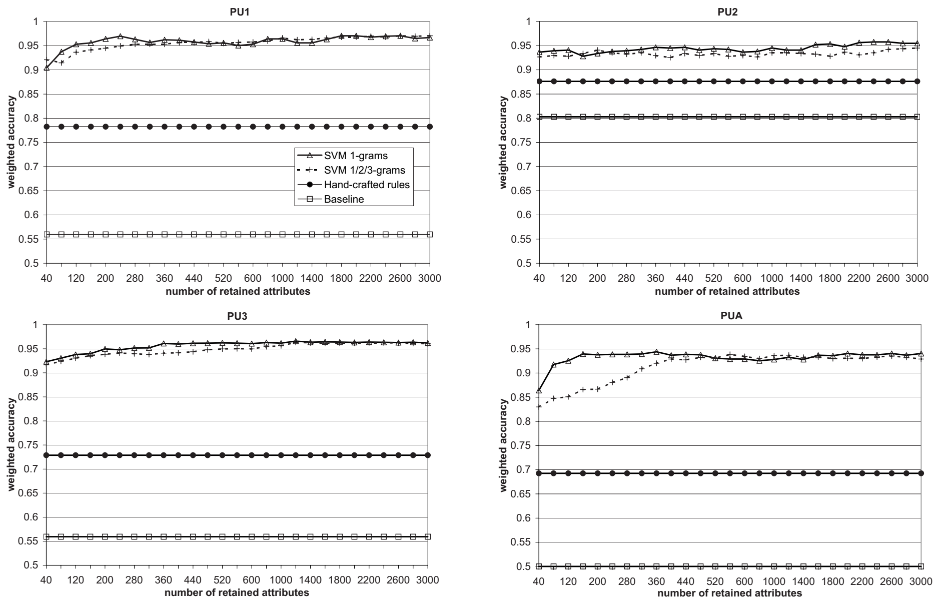
Table VI. Number of 1-grams, 2-grams, and 3-grams (Columns 2–4) among the best 600 n-grams ($n \in \{1, 2, 3\}$). Column 5 sums the previous two columns. Column 6 shows how many of the 2-grams and 3-grams in the best 600 n-grams subsume 1-grams that are also in the best 600 n-grams. Column 7 shows how many of the 3-grams in the best 600 n-grams subsume 2-grams that are also in the best 600 n-grams. All numbers are averaged over the cross-validation folds.

(1) corpus	(2) 1-grams	(3) 2-grams	(4) 3-grams	(5) 2/3-grams	(6) 2/3-grams subsuming 1-grams	(7) 3-grams subsuming 2-grams
PU1	246 (41%)	265 (44%)	88 (15%)	353 (59%)	229 (38%)	82 (14%)
PU2	154 (26%)	291 (49%)	154 (26%)	445 (74%)	262 (44%)	116 (19%)
PU3	226 (38%)	271 (45%)	101 (17%)	372 (62%)	323 (54%)	98 (16%)
PUA	170 (28%)	245 (41%)	184 (31%)	429 (72%)	352 (59%)	179 (30%)

exploring larger numbers of attributes came from the experiments of Carreras and Marquez (Section 5.1), who performed no attribute selection, and experiments by Kolcz and Alspector [2001], who retained 10,000 word attributes after attribute selection. In Section 5.1, we observed a significant increase in both learning and classification time as more attributes were retained, and we noticed that using larger attributes sets did not always lead to significant improvements in weighted accuracy. We wanted to explore the significance of the improvements when the size of the attribute set is in the order of thousands.

Figures 7 and 8 show the results with extended attribute sets. We used only the SVM learner, one of the best ones in the experiments of Section 5.1. SVMs are also known to be able to take advantage of large numbers of attributes [Joachims 1998], which was particular important in the experiments of this section. Up to 600 attributes, we used a step of 40 attributes, as in the 1-gram experiments; from 600 to 3000 attributes the step was 200. A first striking observation is that using n-grams, instead of only 1-grams, does not improve $WAcc$, despite the large numbers of 2-grams and 3-grams with high information gain scores. The only sign of improvement was observed in PUA for $\lambda = 9$ (Figure 8), but otherwise n-grams led to insignificant changes or inferior results. A comparison between columns 5 and 6 of Table VI reveals that most of the 2-grams and 3-grams among the best 600 n-grams subsume 1-grams that are also included in the best 600 n-grams. A similar phenomenon occurs with 3-grams and 2-grams: columns 4 and 7 show that the vast majority of 3-grams in the best 600 n-grams subsume 2-grams that are also in the best 600 n-grams. This can be problematic in that the subsuming n-gram (e.g., "removed from") may not be adding much to the discriminating power of the subsumed one (e.g., "removed"). In that case, the subsuming n-gram may be displacing from the attribute set other more useful candidate attributes, without

Fig. 7. *WAcc* results with extended attribute sets, for $\lambda = 1$.



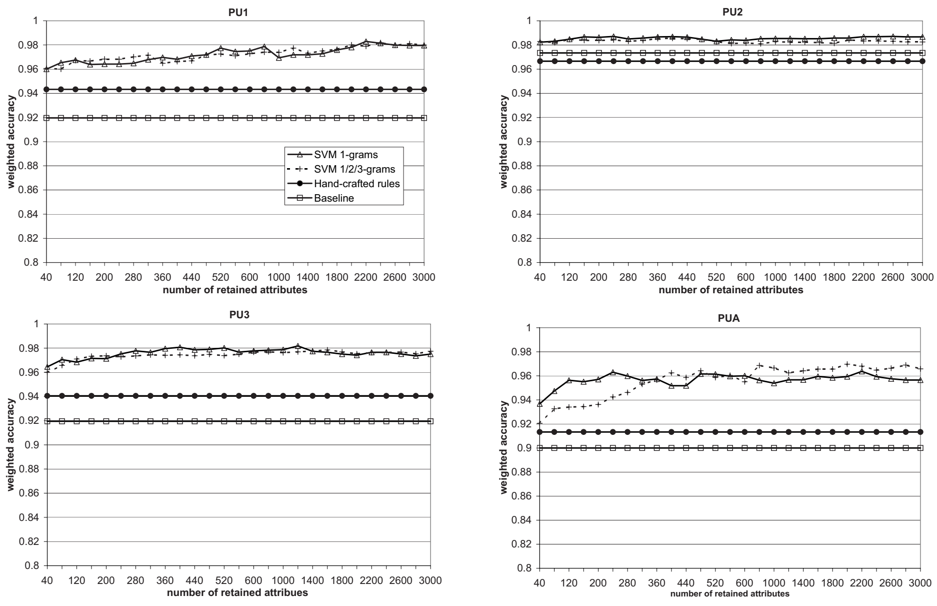
offering any significant benefit.[22] The lack of improvement, then, may be due to the redundancy that n-grams introduce to the attribute set.

As a first step towards reducing the redundancy of the attribute set, we experimented on PU2 with a simplistic extension of the attribute selection method of Section 3.1. First, the information gain score of each n-gram was computed, and the set S of the m n-grams with the highest information gain scores was formed. Any n-grams in S that subsumed other n-grams also in S were then removed, and they were replaced by the next best n-grams that had not been included in S. This was followed by a new removal and replacement phase, up to a maximum number of iterations. On PU2 with $m = 600$, this process had converged after 10 iterations to an attribute set of 201 1-grams, 292 2-grams, and 107 3-grams, averaging over the folds of the cross-validation. There was still no improvement in weighted accuracy, however, compared to the results of the 1-gram experiments of Section 5.1.

It may be the case that the attribute selection process we used is too crude. A refined version of this process that would not eliminate a subsuming n-gram when its information gain score is significantly higher that that of the subsumed one might perform better. It may also be possible to employ algorithms that assess *sets* of attributes, and add new attributes only when the discriminating power of the attribute set as a whole increases [Liu and Setiono 1996; Hall 1999]. Further

[22]With the Bayesian learners, there are also severe violations of their independence assumptions.

Fig. 8. *WAcc* results with extended attribute sets, for $\lambda = 9$.



research is needed on this issue. The existing results, however, provide no evidence that anti-spam filters can benefit from n-gram attributes.

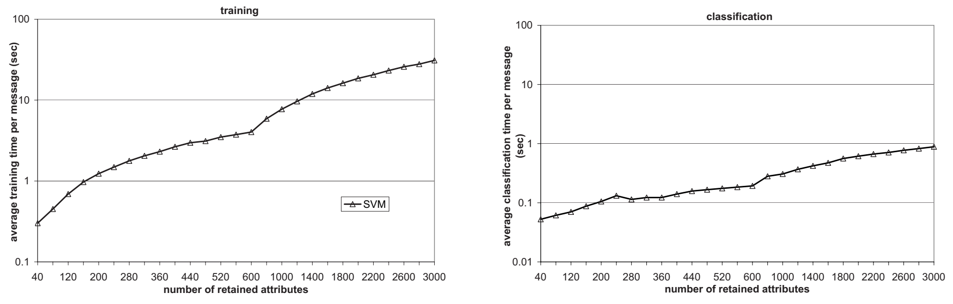
Regarding the effect of retaining large numbers of attributes, Table VII shows that in most cases the precision, recall, and weighted accuracy of the SVM learner were higher when using 3,000 attributes than with 600 attributes. The only exceptions were PU3 and PUA for $\lambda = 9$, where the increase in recall did not counterbalance the decrease of precision, leading to marginally lower weighted accuracy. As Figures 7 and 8 show, however, some of the differences in Table VII are the effect of minor fluctuations, and in most cases the weighted accuracy curves are almost horizontal after the first few hundreds of attributes, which implies that one can often obtain very similar results with much fewer attributes. The most notable exception is PU1 for $\lambda = 9$, where the SVM learner appeared to benefit significantly up to approximately 2,400 attributes. On the other extreme, in PU3 for $\lambda = 9$ there were signs of overfitting after approximately 1,400 attributes.

Figure 9 shows the average training and classification time per message we recorded on PUA for $\lambda = 1$. There is a significant increase in both learning and classification time as more attributes are retained, though one should keep in mind that the curves of Figure 9 pertain to the particular SVM implementation we used. We conclude, as in Section 5.1, that one should be skeptical about using large attribute sets, because the increase in training and classification time may not justify

Table VII. Precision, recall, and *WAcc* (%) of the SVM, using 600 and 3,000 1-gram attributes.

	$\lambda = 1$			$\lambda = 9$		
learner	precision	recall	*WAcc*	precision	recall	*WAcc*
PU1						
SVM 600 attributes	93.96	95.63	95.32	97.97	85.83	97.50
SVM 3,000 attributes	95.60	97.08	96.70	98.25	89.58	97.96
PU2						
SVM 600 attributes	88.71	79.29	93.66	97.50	52.14	98.39
SVM 3,000 attributes	90.85	87.86	95.49	98.75	56.43	98.67
PU3						
SVM 600 attributes	96.48	94.67	96.08	98.13	87.75	97.78
SVM 3,000 attributes	95.60	95.88	96.20	97.53	89.84	97.51
PUA						
SVM 600 attributes	92.83	93.33	92.89	97.10	82.11	96.00
SVM 3,000 attributes	94.56	93.86	94.04	96.67	83.33	95.65

Fig. 9. Avg. training and classification time of the SVM on PUA, with 1-gram attributes, for $\lambda = 1$.



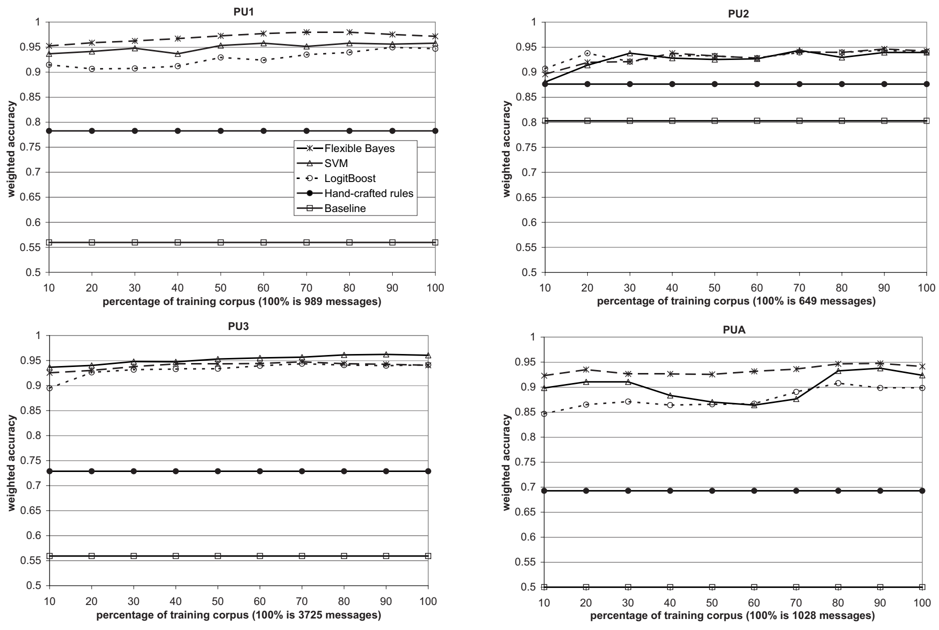
the improvement in accuracy.

5.3 The effect of the size of the training corpus

In a third set of experiments, we explored the effect of the size of the training corpus. As in the previous sections, stratified 10-fold cross-validation was used. This time, however, only the first x% messages of each of the nine training parts were used at each iteration, maintaining the original legitimate-to-spam ratio, with x ranging from 10 to 100 by 10. The experiments were performed with Flexible Bayes, LogitBoost and SVM, using 700 1-gram attributes. We excluded Naive Bayes, since the experiments of Section 5.1 suggested that Flexible Bayes performs better.

Figure 10 shows the results we obtained in the $\lambda = 1$ scenario. All learning algorithms easily outperformed both the baseline and the hand-crafted rules, even with small numbers of training messages. In PU2, the smallest of our corpora, the learning algorithms outperformed the hand-crafted rules with only 20% of training messages, which corresponds to 26 spam and 104 legitimate messages; in the other three, larger corpora, using only 10% of the training messages was enough. This

Fig. 10. *WAcc* results for varying size of training corpus, with 700 1-gram attributes, for $\lambda = 1$.



suggests that learning-based anti-spam filtering is viable in this cost scenario even with training collections that are much smaller than the corpora of our experiments.

Figure 11 shows the average training and classification time per message, on PUA for $\lambda = 1$. As in the experiments of Figure 5, there appears to be a tradeoff between training and classification time, with LogitBoost being the worst in terms of training time and the best in terms of classification time. With all learning algorithms, the average training time increases rapidly as more training messages are used, which provides additional motivation against using large training collections. (The reader is reminded that we advocate the use of personal anti-spam filters, trained on messages received by the particular users they are intended to protect, and that the filters will have to be retrained occasionally to adapt to changes in the content and wording of spam messages.) With Flexible Bayes, the average classification time also increases as more training messages are used, because each training message gives rise to an additional Gaussian distribution (Section 3.3.1). The average classification time of the SVM is largely insensitive to the number of training messages. Similarly, the classification time of LogitBoost is insensitive (modulo minor fluctuations probably due to memory management) to the number of training messages (see Table III).

Figure 12 shows the results we obtained in the $\lambda = 9$ scenario. As with $\lambda = 1$, all three learners outperform both the baseline and the hand-crafted rules, even

Fig. 11. Average training and classification time for varying size of training corpus, with 700 1-gram attributes, on PUA, for $\lambda = 1$.

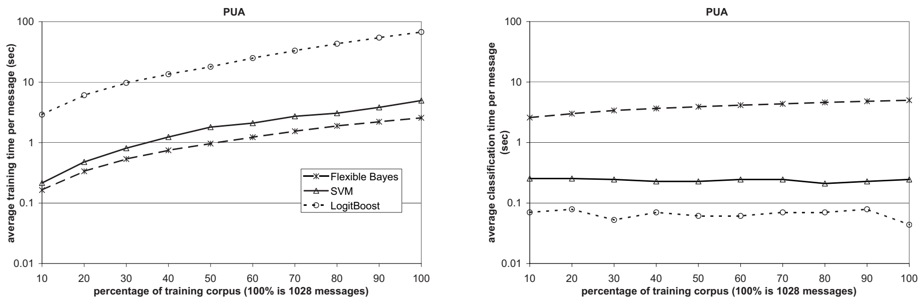
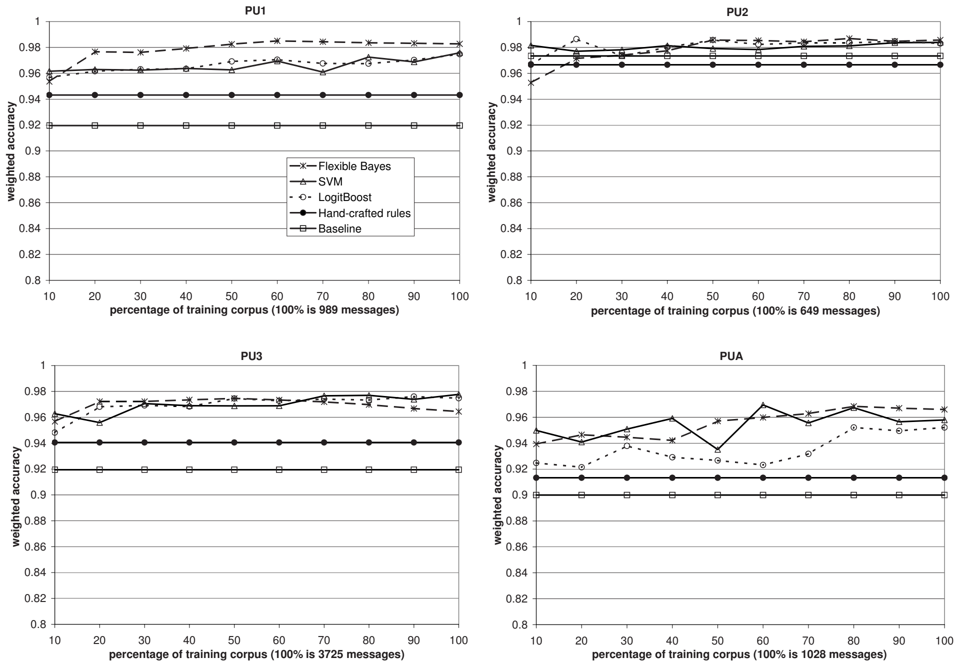


Fig. 12. *WAcc* results for varying size of training corpus, with 700 1-gram attributes, for $\lambda = 9$.



for small numbers of training messages, despite the fact that the baseline is now significantly higher, especially in PU2.

6. IMPLEMENTATION

The results of Section 5 guided the design and implementation of a fully functional learning-based anti-spam filter, named Filtron, whose source code is publicly available; see also [Michelakis et al. 2004]. Filtron uses the WEKA machine learning platform.[23] In Section 6.1 we discuss Filtron's architecture, as an example of how the techniques of the previous sections can be embedded in operational filters. The second author used Filtron in real life for seven months. In Section 6.2, we present an evaluation of Filtron's performance over that period, including a study of the messages it misclassified. Finally, Section 6.3 examines how Filtron relates to other available anti-spam filter implementations we are aware of, focusing on learning-based filters and novel alternative approaches.

6.1 Architecture

Figure 13 depicts Filtron's architecture. The training subsystem tailors the filter to the particular user it is intended to protect, by treating messages previously received by the user as training examples. Thereafter, when a new message for the user arrives, the classification subsystem is invoked to classify it.

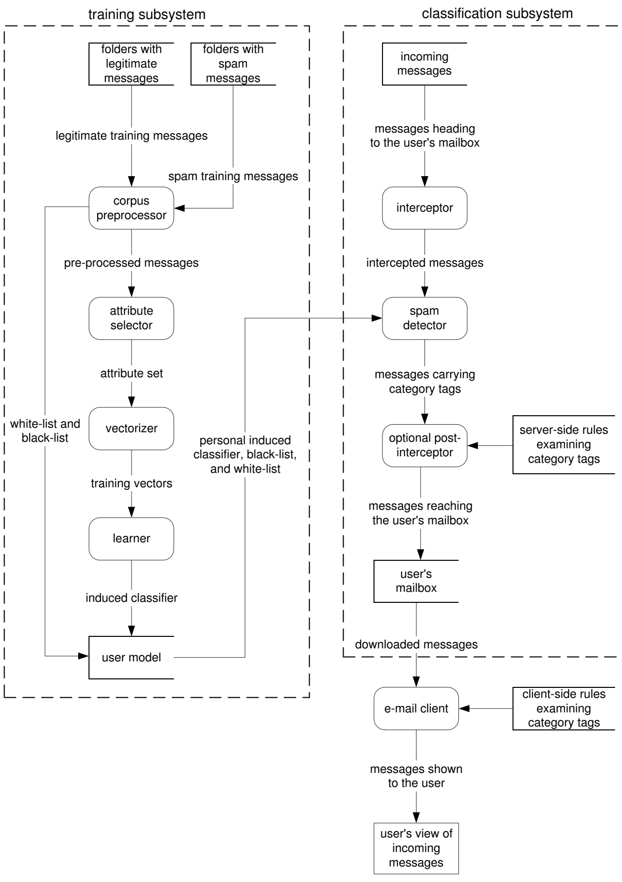
Filtron's training subsystem comes with its own collection of duplicate-free spam messages, which are used when users do not have their own collection of spam messages they have received. We update this collection regularly, and it currently contains 2559 spam messages. Spam messages are typically sent blindly, without considering the professions, interests, etc. of the recipients. Hence, training Filtron on spam messages not received by its particular user does not seem to be harmful. When using Filtron's pool of spam messages, the user can choose the number of spam messages to be included in the training collection, to approximate the legitimate-to-spam ratio of incoming messages he/she is experiencing, which can lead to more accurate classifiers. We have not investigated how well Filtron performs when trained on legitimate messages not received by its particular user, but Section 3.1 suggests that this would be worse, since many of the best attributes correspond to terminology characteristic of the user's legitimate messages.

The corpus preprocessor (Figure 13) scans the training messages, removes duplicates, attachments, and HTML tags, and retains only the first five messages from each sender, as explained in Section 2. It can also replace tokens by unique numbers to make a collection publicly available. The messages can be provided in a variety of formats, including folders of several popular email clients. The preprocessor also builds automatically a white-list with the addresses the user has received legitimate messages from, and a black-list with the addresses of the senders of all spam messages, though, as discussed in Section 1, black-lists of this form are of little use. The user can inspect and edit both lists. In future versions we will offer the option to include the addresses of the user's address book in the white-list.

The attribute selector identifies the best attributes, as in Section 3.1.2. Following the conclusions of Section 5.2, Filtron uses 1-gram attributes only. The number

[23]See Section 1. Filtron will be available from `http://www.iit.demokritos.gr/skel/i-config/`.

Fig. 13. Filtron's architecture.


training subsystem
folders with legitimate messages
folders with spam messages
legitimate training messages
spam training messages
corpus preprocessor
pre-processed messages
attribute selector
attribute set
vectorizer
training vectors
learner
induced classifier
user model
white-list and black-list
personal induced classifier, black-list, and white-list
classification subsystem
incoming messages
messages heading to the user's mailbox
interceptor
intercepted messages
spam detector
messages carrying category tags
optional post-interceptor
server-side rules examining category tags
messages reaching the user's mailbox
user's mailbox
downloaded messages
e-mail client
client-side rules examining category tags
messages shown to the user
user's view of incoming messages

of attributes to retain can be set at will. The optimal number of attributes, in terms of accuracy, training, and classification speed, will vary per user, and is hard to specify without user-specific experiments. The results of Sections 5.1 and 5.2, however, suggest that using a few hundreds of attributes is a good compromise.

Once the attributes have been selected, the vectorizer converts the training messages to vectors, as in Section 3.1. The training vectors are then passed to the learning component, that accepts as additional input the value of λ (Section 4.1). In the learning component, the user can choose any of the learning algorithms of WEKA. The default choice is the SVM implementation of Section 5, which demonstrated a good combination of accuracy, consistency, and speed. The learning component produces a user-specific classifier, and this is saved along with the user's white-list and black-list in Filtron's user model. In the case of the SVM learner, saving the classifier amounts to saving the non-zero Langrange multipliers α_j, the corresponding training vectors $\vec{x}_j$, and the threshold w_0 of equation (17). The user model is then copied to the computer where the classification subsystem will be used, if this is a different machine.

Filtron's training subsystem is implemented in TCL/TK and Java, and can be used on any computer that supports the two languages; for example, a PC where the user downloads his/her e-mail. The classification subsystem is implemented in the same languages, but resides on the user's incoming e-mail server, and is currently compatible only with SMTP servers running UNIX. The classification subsystem intercepts the user's messages before they reach his/her mailbox using Procmail, an e-mail processor that accompanies most UNIX distributions. The interceptor (Figure 13) invokes Filtron's spam detector, passing it as inputs the incoming message and the user's model. If the sender is in the user's white-list or black-list, the spam detector classifies the message as legitimate or spam, respectively, without further processing. Otherwise, it relies on the decision of the user-specific classifier. In any case, the only effect of the spam detector is that it adds the prefix "[Spam?]" to the subject of the messages it considers spam.

In Filtron's typical configuration, the user writes rules in his/her e-mail client to identify messages that carry the "[Spam?]" prefix and delete them, move them to a special folder, change their priority, ask their sender to repost them, etc. Writing such rules is easy in most modern e-mail clients. Alternatively, a Procmail-based post-interceptor (Figure 13) can be set up by the e-mail server's administrator, to intercept messages that carry the "[Spam?]" prefix before they reach the user's mailbox. The post-interceptor can be configured to perform similar actions as the rules of the e-mail client. Users that download their e-mail through slow lines may prefer this option, as it avoids downloading messages suspected to be spam.

The training subsystem can be re-invoked periodically to adjust the classifier to changes in the content and wording of spam messages, and to exploit larger training collections the user may have accumulated. A graphical interface is also provided, to help users train Filtron.

6.2 Real-use evaluation

To explore how well a learning-based filter performs in real life, the second author used Filtron for seven months. The training subsystem was applied to PU3, which contains the legitimate messages the second author had saved and what was Filtron's collection of spam messages at that time (Table I). We used the SVM learner, as it had exhibited good performance on PU3. The filter was configured for the $\lambda = 1$ scenario, where messages suspected to be spam are simply flagged to help the user prioritize the reading of incoming messages (Section 4.1). An e-mail client rule was used to move messages tagged as spam to a special folder, which was occa-

Table VIII. Real-life evaluation results of Filtron, using the SVM with 520 1-gram attributes, for $\lambda = 1$. The SVM was trained on PU3. Bracketed precision, recall, and *WAcc* scores are the corresponding scores we had obtained with 10-fold cross validation on PU3.

days used	212	
messages received	6732	(avg. 31.75 per day)
spam messages received	1623	(avg. 7.66 per day)
legitimate messages received	5109	(avg. 24.10 per day)
legitimate-to-spam ratio	3.15	
correctly classified legitimate messages ($L \rightarrow L$)	5057	
incorrectly classified legitimate messages ($L \rightarrow S$)	52	(avg. 1.72 per week)
correctly classified spam messages ($S \rightarrow S$)	1450	
incorrectly classified spam messages ($S \rightarrow L$)	173	(avg. 5.71 per week)
precision	96.54%	(PU3: 96.43%)
recall	89.34%	(PU3: 95.05%)
WAcc	96.66%	(PU3: 96.22%)

sionally inspected for misclassified legitimate messages. The number of attributes was set to 520, based on the results of Figures 4 and 7. No black-list was used, since a preliminary trial indicated that its contribution was insignificant.

Table VIII presents the overall results of the seven-month evaluation. Since no black-list was used, all the messages that were classified as spam were caught by the learnt classifier. Precision (96.54%) was very similar to the corresponding score we had obtained with 10-fold cross-validation on PU3 (96.43% for 520 attributes) in the experiments of Sections 5.1 and 5.2. Recall, however, was lower (89.34% as opposed to 95.05%). The analysis of the misclassified messages of the seven months below sheds more light on this issue. Although similar, the two *WAcc* scores are not directly comparable, since the *WAcc* of the real-use evaluation (96.66%) is computed on all the incoming messages, i.e., it includes the effect of both the white-list and the learnt classifier. In contrast, since RC messages were removed from PU3 (Section 2), the *WAcc* of the cross-validation (96.22%) in effect measures the performance of the learnt classifier alone. Thus, we would have expected the *WAcc* of the real-use evaluation to be higher than that of the cross-validation. The fact that it is not is due to the lower recall of the real-use evaluation.

Overall, Filtron's performance was satisfactory for the chosen scenario, though there is scope for improvement. The e-mail client rule that moved messages tagged as spam to the special folder left on average fewer spam messages per week (5.71) in the second author's inbox than he received in a single day (7.66). The downside was that approximately two (1.72) legitimate messages were incorrectly moved to the special folder each week. The second author developed the habit of checking that folder at the end of each week, which made the misclassifications easier to accept. He also felt that in many cases the misclassified legitimate messages were rather indifferent to him (e.g., subscription verifications, commercial newsletters, etc.), an observation that the analysis of the misclassified messages below confirms. We note that Filtron was never retrained during the seven months, because the training process had not been fully automated by that time. Retraining would have allowed the system to gradually exploit more training data, and keep its white-list updated, presumably leading to better results. Throughout the seven months, Filtron caused no noticeable delay in e-mail delivery.

We now turn to the analysis of the misclassified messages, starting from the 173 misclassified spam messages. A 30% of those were "hard spam", i.e., messages with very little or no text, containing mostly hyperlinks, messages whose text was sent as images or was hidden in attachments, messages containing tricks to confuse simple tokenizers (e.g., HTML tags within words), and messages carefully written to avoid words, phrases, and punctuation patterns that are common in spam messages. Many of these messages could have been caught by using a more elaborate preprocessor; this will be discussed further in Section 6.3. Overall, however, they are indicative of an arms race between spammers and developers of anti-spam filters. Future anti-spam filters may well have to incorporate optical character recognition to cope with messages sent as images, and they may have to follow hyperlinks to Web pages to cope with spam messages that contain only hyperlinks and no text. Additional support for the latter point comes from the observation that a further 8% of the misclassified spam messages were advertisements of pornographic sites, carefully written as friendly letters with no pornographic vocabulary, but containing hyperlinks to sites that a Web filter like the one described by Chandrinos et al. [2000] would have no difficulty classifying as adults-only content.

Another 23% of the misclassified spam messages were written in languages other than English, mostly German and Spanish. Non-English spam messages were rare at the time the PU corpora were assembled, but appear to have become more frequent thereafter. Hence, they are under-represented in PU3, which was the training collection of the real-use evaluation. (The same holds for the other PU corpora, especially PU1 and PU2 where non-English spam messages were manually removed; see Section 2.) We believe that a large number of these misclassified messages would have been caught if Filtron had been frequently retrained during the seven months. This would have allowed Filtron to select non-English words as attributes as non-English spam messages were becoming more common. Filtron's collection of spam messages now includes non-English messages.

A further 30% of the misclassified spam messages were written in BASE64 or quoted printable format, which Filtron could not handle at that time; appropriate support has now been added. Messages in these formats were very rare in the second author's legitimate messages, and, hence, the problem affected only spam messages. Filtron classified all the spam messages that were written in the two formats as legitimate, because the encoding did not allow it to detect any attribute tokens. In PU3, messages of this kind had been excluded, which partially explains the higher recall score. Another factor that contributed to the higher recall of the cross-validation is that spam messages that contained only attachments, hyperlinks, and no text were removed when PU3 was constructed, because the preprocessor converted them to empty messages. In the real-use evaluation, Filtron's preprocessor again turned spam messages of this kind to empty messages, but this time they were classified as legitimate, which lowered recall.

Another 6% of the misclassified spam messages were formal long letters advertising tremendous business opportunities or asking for financial aid. Since spam messages of this kind are very common and contain many cliche phrases, it came as a surprise that Filtron did not manage to detect so many of them. It appears that the main reason these messages were misclassified is that they contained several occurrences of the second author's name, apparently to make their messages sound more personal. This was uncommon in PU3, where the user's name was among the best legitimate-denoting attributes. Again, retraining would have allowed Filtron to diminish its confidence to this attribute as the user's name was becoming more common in spam messages.

The remaining 3% of misclassified spam messages contained very unusual content, and, hence, they were very different from the spam messages Filtron had been trained on. Interestingly enough, some of these messages were quite relevant to the second author's scientific interests (e.g., search engines for research articles, natural language processing platforms, etc.), which may be an indication of an attempt made by spammers to target particular user groups (e.g., by collecting user names from particular scientific newsgroups, or customer lists of particular vendors). Personalized messages of this kind are more difficult to detect, as their vocabulary and content is very similar to those of the user's legitimate messages, but at least appear to be more interesting to read.

Overall, then, there are indications that regular retraining and encoding improvements that have now been made would have allowed Filtron to achieve higher recall. More elaborate pre-processing and the addition of a Web filter might allow learning-based filters to perform even better, though there are also signs of an arms race between spammers and filter developers.

Turning to the 52 misclassified legitimate messages, it is comforting that 52% of them were automatically generated messages (e.g., subscription verifications, virus warnings), or commercial newsletters whose content was very similar to spam messages. A further 22% were very short messages (3-5 words) containing attachments and hyperlinks. Again, this emphasizes the need to process these elements in future anti-spam filters. Retraining would also have helped, because the senders of some of these messages were recent colleagues not in the user's white-list. The remaining

26%, were short messages (1-2 lines), with no attachments or hyperlinks, written in a very casual style that is often exploited by spammers. As in the previous category, retraining would have placed the senders of some of these messages in the user's white list. Non-textual attributes indicating that the incoming messages contained no attachments and hyperlinks might also have helped, since it is uncommon for spammers to send such sort messages with no attachments and hyperlinks.

6.3 Other implementations

The majority of commercial anti-spam filters currently appear to rely on black-lists, white-lists, and hand-crafted rules that search for particular keywords, phrases, or suspicious patterns in the headers. As noted in Section 1, black-lists have evolved to continuously updated databases of suspicious IP numbers; hence, they are more powerful than Filtron's black-lists, which simply store the addresses the user has received spam from. On the white-list side, some commercial filters send replies to senders not in the user's white-list, asking them to answer a simple question in order to rule out spamming robots.[24] This is similar to a suggestion we made in Section 4.1 for the $\lambda = 9$ scenario, but since there is no learning component to admit messages whose content looks legitimate, it may lead to a frustrating proliferation of automatic answer-seeking replies.

Following research publications (Section 1), the developers of anti-spam filters are becoming increasingly aware of the potential of machine learning. Consequently, there is a growing number of filter implementations that incorporate learning-based components, mostly Naive Bayes classifiers.[25] For example, an anti-spam filter based on a flavor of Naive Bayes was recently added to Mozilla's e-mail client.[26] The filter is trained on legitimate and spam messages of its particular user, and can be configured to perform actions such as moving incoming messages suspected to be spam to a special folder. Unlike Filtron, Mozilla's filter supports incremental learning, i.e., the user can correct the category of misclassified messages, and the probabilities of the Bayesian filter are adjusted accordingly, without retraining on the entire message collection.

POPFILE is similar to Mozilla's filter, in that it also resides on the client computer, employs Naive Bayes, and supports incremental learning. However, it is a more general filter, which apart from detecting spam messages can also classify

[24]Examples are MailFrontier's Matador (`http://www.mailfrontier.com/`) and Altebia's SpamPepper (`http://www.spampepper.com/`).
[25]Many of these were presented at the 2003 Spam Conference; see `http://spamconference.org/`.
[26]Mozilla uses a version of Naive Bayes suggested by Paul Graham; see `http://www.paulgraham.com/spam.html`. Other Naive Bayes filters we are aware of are K9, Spammunition, and Bogofilter; see `http://keir.net/k9.html`, `http://www.upserve.com/spammunition/`, and `http://bogofilter.sourceforge.net/`. For POPFILE see `http://popfile.sourceforge.net/`, and for CRM114 consult `http://crm114.sourceforge.net/`.

legitimate messages into different categories. Written in Perl, it acts as an e-mail proxy for POP3 servers, and adds tags to messages it considers spam. The tags can then trigger rules of the e-mail client, much as in Filtron. One of the most interesting characteristics of POPFILE is its elaborate preprocessor, which attempts to confront tricks spammers are beginning to use to confuse anti-spam filters; for example, inserting spaces, other characters, or HTML comments and formatting tags between letters to fool tokenizers (as in "g e t r*i*c*h f-a-s-t", "vi`<!-- 45 -->`agra", "s`<b>`e`</b>`x").[27] POPFILE shows how important preprocessing will be in future anti-spam filters, as opposed to Filtron's current simple tokenizer.

Mailfilter is part of CRM114, a system that can be used to process incoming e-mail, logs, and other data streams. Mailfilter is also based on Naive Bayes, and supports incremental learning. It resides on the e-mail server, and can be invoked through Procmail, like Filtron, or via a forward file in the user's home directory. Its attribute set, however, is very different. Each message is mapped to a set of 32-bit hash values, one hash value for each order-preserving sequence of up to 5 (not necessarily adjacent) tokens of the message. For example, the message "click here to get rich" is mapped to a set containing a hash value for each one of the sequences "click", "click here", "click to", "click here to", etc. The hashing allows the presence or absence of all possible sequences to be represented using 2^{32} Boolean attributes, one for each possible hash value. To reduce the size of the attribute set further, the hash values are folded to one million values via the modulo function, leading to one million attributes. Unlike the experiments with n-gram attributes in Section 5.2, then, Mailfilter's n-grams are not necessarily composed of adjacent tokens, they can be longer ($1 \leq n \leq 5$), and no attribute selection is applied to them; instead Mailfilter reduces dimensionality via hashing.

An interesting alternative to learning-based anti-spam filters is Vipul's Razor, a form of collaborative filtering, which relies on a network of servers that store signatures of spam messages.[28] The users of the network report to its servers spam messages they receive, and at the same time they use screening software that compares their incoming messages to the spam messages other users have reported. To make the comparison fast, the reported spam messages are stored as signatures, in the simplest case a hash value for each message. When an incoming message arrives at the mailbox of a user, the screening software computes its signature and compares it to the signatures in the network's servers. If a match is found, the message is moved to a special folder. If a spam message escapes the matching, the user can report it to the network, and this helps other users avoid it. Unfortunately,

[27]See the "Spammer's Compendium" by John Graham-Cumming at `http://www.jgc.org/tsc/`.
[28]Consult `http://razor.sourceforge.net/`, `http://www.cloudmark.com/`, and `http://pyzor.sourceforge.net/`. See also `http://www.rhyolite.com/anti-spam/dcc/` for a similar approach.

spammers can defeat simple signatures by introducing random changes to the copies of the messages they send (e.g., inserting a random string, or replicating randomly parts of the message), which causes the signatures to be different and the matching to fail. Hence, much of the work in this area is devoted to devising signatures that are insensitive to such changes.

Spam messages can be collected automatically via spam traps, e-mail accounts that are created for the sole purpose of attracting spam messages. The addresses of spam traps are included in newsgroup postings and Web pages, making it clear to humans that messages sent to them will be considered spam. The software that spammers use to harvest e-mail addresses cannot distinguish between normal and spam trap addresses, and, hence, the latter end up in their mailing lists.[29]

In the long run, the spam problem is more likely to be solved by a combination of several techniques. The SpamAssassin filter is a good example of such a combination.[30] It incorporates elaborate hand-crafted rules, white-lists, on-line black-lists, Vipul's Razor, and a Naive Bayes classifier. Users can add their own rules, white-lists, and black-lists, and they can retrain the Naive Bayes classifier, or modify a set of weights that specify how much SpamAssasin relies on each filtering technique. A genetic algorithm can also be used to generate the weights automatically.

7. CONCLUSIONS

We presented a thorough investigation on the use of machine learning to construct effective personalized anti-spam filters. The investigation considered a set of four learning algorithms, selected to include the algorithms, or variations of them, that previous work on anti-spam filtering had found most promising. Four corpora, constructed from the mailboxes of different users, were used. Continuing our effort to provide benchmarks in this area, we made the four corpora publicly available, along with a fully implemented anti-spam filter that we developed based on the findings of our corpus experiments. We also presented an assessment of that filter's performance in real life over a period of seven months.

Our investigation examined various aspects of the design of learning-based anti-spam filters. Unlike our previous work, we used numeric attributes that represent the frequencies of tokens, rather than Boolean attributes. Furthermore, we examined the role of attributes that represent the frequencies of n-grams, i.e., sequences of tokens. Our conclusion was that, although many of these sequences are good indicators of whether or not a message is spam, they rarely add significantly to the performance of a classifier that uses attributes corresponding to single tokens.

We have also discussed the model and search biases of the four learning algo-

[29]See `http://www.brightmail.com/` for a filter that relies substantially on spam traps.
[30]See `http://www.spamassassin.org/`.

rithms, i.e., the type of classifier each method can construct, and the heuristics it uses to search the space of possible classifiers, along with worst-case computational complexity estimates. Our coprus-based experiments confirmed that a combination of a strong model bias, such as learning linear discriminants, and an effective heuristic, like the optimization method of SVMs, provides good performance at relatively low computational cost. Our experiments also showed that the real-life computational cost is often lower than what the worst-case theoretical figures predict.

Another issue in anti-spam filtering is the cost of different misclassification errors. After considering alternative ways to produce cost-sensitive classifiers, we tested the performance of the four learning algorithms in two different cost scenarios: one where messages classified as spam are simply flagged, and one where they are returned to their senders for reposting, in a way that rules out responses from spamming software. With the exception of Naive Bayes, the learning algorithms adapted well to the uneven misclassification cost of the second scenario.

We also examined the effect of the size of the training and attribute set. Our corpus experiments showed learning-based anti-spam filtering is viable even with very small training collections, especially when the filter simply flags messages it suspects to be spam. We also concluded that attribute selection based on information gain leads to good performance with a few hundreds of attributes. Using much larger attribute sets may lead to further improvements in accuracy, but the improvements are usually small, and they are counter-balanced by increased computational cost.

The real-life use of our filter confirmed that machine learning can play a prominent role in anti-spam filtering. There were signs, however, of an arms race between spammers and developers of filters. Regular re-training seems necessary, but more elaborate preprocessing is also needed to address the techniques that spammers are beginning to use to fool content-based filters.

In the long run, mixtures of different filtering approaches, including collaborative filtering, are more likely to be successful, and machine learning has a central role to play in such mixed filters. Combining different learning algorithms also seems to be promising, as different classifiers often make different errors [Sakkis et al. 2001a].

ACKNOWLEDGMENTS

We thank K.V. Chandrinos, V. Karkaletsis, G. Petasis, and F. Sebastiani for the e-mail messages they provided. We also thank K.V. Chandrinos, J. Koutsias, and G. Sakkis for their involvement in the early stages of our work on anti-spam filtering. Special thanks to C.D. Spyropoulos for his encouragement and the resources he made available to us. The third author thanks P. Stamatopoulos, who was his advisor at the Univ. of Athens during his final-year project on anti-spam filtering.

REFERENCES

Aha, W. and Albert, M. 1991. Instance-based learning algorithms. *Machine Learning 6*, 37–66.

Androutsopoulos, I., Koutsias, J., Chandrinos, K., Paliouras, G., and Spyropoulos, C. 2000. An evaluation of Naive Bayesian anti-spam filtering. In *Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning.* Barcelona, Spain, 9–17.

Androutsopoulos, I., Koutsias, J., Chandrinos, K., and Spyropoulos, C. 2000. An experimental comparison of Naive Bayesian and keyword-based anti-spam filtering with encrypted personal e-mail messages. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* Athens, Greece, 160–167.

Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C., and Stamatopoulos, P. 2000. Learning to filter spam e-mail: A comparison of a Naive Bayesian and a memory-based approach. In *Proceedings of the Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases.* Lyon, France, 1–13.

Burges, C. 1998. A tutorial on Support Vector machines for pattern recognition. *Journal of data Mining and Knowledge Discovery 2,* 2, 121–167.

Carreras, X. and Marquez, L. 2001. Boosting trees for anti-spam email filtering. In *4th International Conference on Recent Advances in Natural Language Processing.* Tzigov Chark, Bulgaria, 58–64.

Chandrinos, K., Androutsopoulos, I., Paliouras, G., and Spyropoulos, C. 2000. Automatic web rating: Filtering obscene content on the web. In *Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries.* Lisbon, Portugal, 403–406.

Cohen, W. 1996. Learning rules that classify e-mail. In *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access.* Palo Alto, California, 18–25.

Cohen, W. and Singer, Y. 1999. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems 17,* 2, 141–173.

Cover, T. and Hart, P. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory 13*, 21–27.

Cranor, L. and LaMacchia, B. 1998. Spam! *Communications of the ACM 41,* 8, 74–83.

Cristianini, N. and Shawe-Taylor, J. 2000. *An introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press.

Domingos, P. 1999. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining.* ACM Press, San Diego, California, 155–164.

Domingos, P. and Pazzani, M. 1996. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In *Proceedings of the 13th International Conference on Machine Learning.* Bari, Italy, 105–112.

Drucker, H. D., Wu, D., and V., V. 1999. Support Vector Machines for spam categorization. *IEEE Transactions On Neural Networks 10,* 5, 1048–1054.

Duda, R. and Hart, P. 1973. *Bayes Decision Theory.* John Wiley, Chapter 2, 10–43.

Frank, E. and Witten, I. 1998. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference on Machine Learning.* Madison, Wisconsin, 152–160.

Friedman, J., Hastie, T., and Tibshirani, R. 2000. Additive logistic regression: A statistical view of boosting. *Annals of Statistics 28,* 2, 337–374.

Fukunaga, K. 1990. *Nonparametric Density Estimation.* Academic Press, Chapter 6, 254–299.

Gauthronet, S. and Drouard, E. 2001. Unsolicited commercial communications and data protection. Technical report, Commission of the European Communities, Internal Market DG.

Hall, M. 1999. Correlation-based feature selection for machine learning. Ph.D. thesis, Department of Computer Science, University of Waikato, New Zealand.

Hidalgo, J. G. 2002. Evaluating cost-sensitive unsolicited bulk email categorization. In *Proceedings of the 17th ACM Symposium on Applied Computing.* 615–620.

Hidalgo, J. G. and Lopez, M. M. 2000. Combining text and heuristics for cost-sensitive spam filtering. In *Proceedings of the 4th Computational Natural Language Learning Workshop.* Lisbon, Portugal, 99–102.

Holte, R. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine Learning 11,* 1, 63–91.

Jaeger, M. 2003. Probabilistic classifiers and the concepts they recognize. In *Proceedings of the 20th International Conference on Machine Learning.*

Joachims, T. 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proceedings of the 14th International Conference on Machine Learning.* Nashville, Tennessee, 143–151.

Joachims, T. 1998. Text categorization with Support Vector Machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning, Lecture Notes in Computer Science, n. 1398.* Springer Verlag, Heidelberg, Germany, 137–142.

John, G. and Langley, P. 1995. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence.* Montreal, Quebec, 338–345.

Kolcz, A. and Alspector, J. 2001. SVM-based filtering of e-mail spam with content-specific misclassification costs. In *Proceedings of the Workshop on Text Mining, IEEE International Conference on Data Mining.* San Jose, California.

Lang, K. 1995. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning.* Stanford, California, 331–339.

Langley, P., Wayne, I., and Thompson, K. 1992. An analysis of Bayesian classifiers. In *Proceedings of the 10th National Conference on Artificial Intelligence.* San Jose, California, 223–228.

Liu, H. and Setiono, R. 1996. A probabilistic approach to feature selection – a filter solution. In *Proceedings of the 13th International Conference on Machine Learning.* Bari, Italy, 319–327.

Manning, C. and Schutze, H. 1999. *Foundations of statistical natural language processing.* MIT Press.

McCallum, A. and Nigam, K. 1998. A comparison of event models for naive bayes text classification. In *Proceedings of the AAAI'98 Workshop on Learning for Text Categorization.* Madison, Wisconsin, 41–48.

Michelakis, E., Androutsopoulos, I., Paliouras, G., Sakkis, G., and Stamatopoulos, P. 2004. Filtron: a learning-based anti-spam filter. In *Proceedings of the 1st Conference on Email and Anti-Spam.* Mountain View, California.

Mitchell, T. 1997. *Machine learning.* McGraw-Hill.

Pantel, P. and Lin, D. 1998. SpamCop: a spam classification and organization program. In *Learning for Text Categorization – Papers from the AAAI Workshop.* Madison, Wisconsin, 95–98.

Payne, T. and Edwards, P. 1997. Interface agents that learn: An investigation of learning issues in a mail agent interface. *Applied Artificial Intelligence 11,* 1, 1–32.

Platt, J. 1999. Fast training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods – Support Vector Learning*, B. Scholkopf, C. Burges, and A. Smola, Eds. MIT Press, 185–208.

Quinlan, J. 1993. *C4.5: programs for machine learning.* Morgan Kaufmann.

Rocchio, J. 1971. Relevance feedback information retrieval. In *The Smart retrieval system – Experiments in automatic document processing*, G. Salton, Ed. Prentice-Hall, 313–323.

Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. 1998. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization – Papers from the AAAI Workshop.* Madison, Wisconsin, 55–62.

Sakkis, G., Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Spyropoulos, C., and Stamatopoulos, P. 2003b. A memory-based approach to anti-spam filtering for mailing lists. *Information Retrieval 6,* 1, 49–73.

Sakkis, G., Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Spyropoulos, C., and Stamatopoulos, P. 2001a. Stacking classifiers for anti-spam filtering of e-mail. In *Proceedigs of the 6th Conference on Empirical Methods in Natural Language Processing.* Carnegie Mellon, Pittsburgh, 44–50.

Salton, G. and McGill, M. 1983. *Introduction to modern information retrieval.* McGraw-Hill.

Schapire, R. and Singer, Y. 2000. BoosTexter: a boosting-based system for text categorization. *Machine Learning 39,* 2/3, 135–168.

Schneider, K.-M. 2003. A comparison of event models for Naive Bayes anti-spam e-mail filtering. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*. Budapest, Hungary, 307–314.

Sebastiani, F. 2002. Machine learning in automated text categorization. *ACM Computing Surveys 34,* 1, 1–47.

Ting, K.-M. 1998. Inducing cost-sensitive trees via instance weighting. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery, Lecture Notes in Computer Science, n. 1510*. Springer Verlag, Berlin, Germany, 139–147.

Yang, Y. and Pedersen, J. 1997. A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning*. Nashville, Tennessee, 412–420.

Zhang, J., Jin, R., Yang, Y., and Hauptmann, A. 2003. Modified logistic regression: An approximation to SVM and its applications in large-scale text categorization. In *Proceedings of the 20th International Conference on Machine Learning*. AAAI Press, 888–895.