Learning at Low False Positive Rates

Wen-tau Yih
Microsoft Research
One Microsoft Way
Redmond, WA, USA
scottyih@microsoft.com

Joshua Goodman
Microsoft Research
One Microsoft Way
Redmond, WA, USA
joshuago@microsoft.com

Geoff Hulten
Microsoft
One Microsoft Way
Redmond, WA, USA
ghulten@microsoft.com

ABSTRACT

Most spam filters are configured for use at a very low false-positive rate. Typically, the filters are trained with techniques that optimize accuracy or entropy, rather than performance in this configuration. We describe two different techniques for optimizing for the low false-positive region. One method weights good data more than spam. The other method uses a two-stage technique of first finding data in the low false-positive region, and then learning using this subset. We show that with two different learning algorithms, logistic regression and Naive Bayes, we achieve substantial improvements, reducing missed spam by as much as 20% relative for logistic regression and 40% for Naive Bayes at the same low false-positive rate.

1. INTRODUCTION

In most practical spam filtering, users are much more concerned about not losing good mail (ham) than about receiving a few pieces of spam. On the other hand, most machine learning systems are trained using algorithms that assume that missing good mail and receiving spam are equally bad. After the training is performed, a threshold is set to achieve the desired low false positive rate (low missed good mail), but the actual filter optimization was performed for a different criterion, typically optimizing for accuracy or entropy. Accuracy optimizes for equal costs for false positives and false negatives; entropy optimizes for estimating probabilities correctly, across the entire range of probabilities: neither optimizes for the actual area of interest.

In the anti-spam literature, there has been relatively little research into how to optimize specifically for the low false positive situation, other than simply setting a threshold. In this paper, we explore the question of whether by explicitly changing our model or optimization criterion we can do a better job of learning for this low false positive region.

We will explore two different techniques. The first idea is to simply weight good messages (ham) as more valuable than spam data. This technique is moderately well known, and is often called stratification. However, it may appear that with probabilistic learning models, this will simply shift probabilities; indeed, Elkan [4] argued that this technique will have little effect in practice, especially for techniques like Naive Bayes. We explain what causes the improvement for Naive Bayes – regularization – and we show empirically that for spam filtering, the improvements can be large. We also show that an additional rotational effect applies for discriminative techniques like logistic regression.

The second idea is novel, more complex, and also works better. With this method, we first find data that has a very low probability of being spam according to our models. We exclude this data from our training, and retrain on only the portion of the data that falls in the low false positive region. This causes the second model to focus on this region, and improves accuracy.

We try both of these techniques on a very large corpus of hand labeled data, as well as trying them in combination. We use two different learning algorithms, logistic regression and Naive Bayes. We find that both of these techniques can substantially reduce the percentage of missed spam at a given false positive rate in the region of interest. For logistic regression, the reduction can be as much as 20%. For Naive Bayes, the reduction can be as much as 40%.

2. METHODS

In this section, we first briefly review the two learning algorithms we use, logistic regression and Naive Bayes, in part to review their optimization criteria, which in both cases do not focus on any particular section of an ROC curve, but instead try to get overall probabilities as accurate as possible.

We then describe and justify each of our two methods, both of which lead to improvements on real data.

2.1 Background

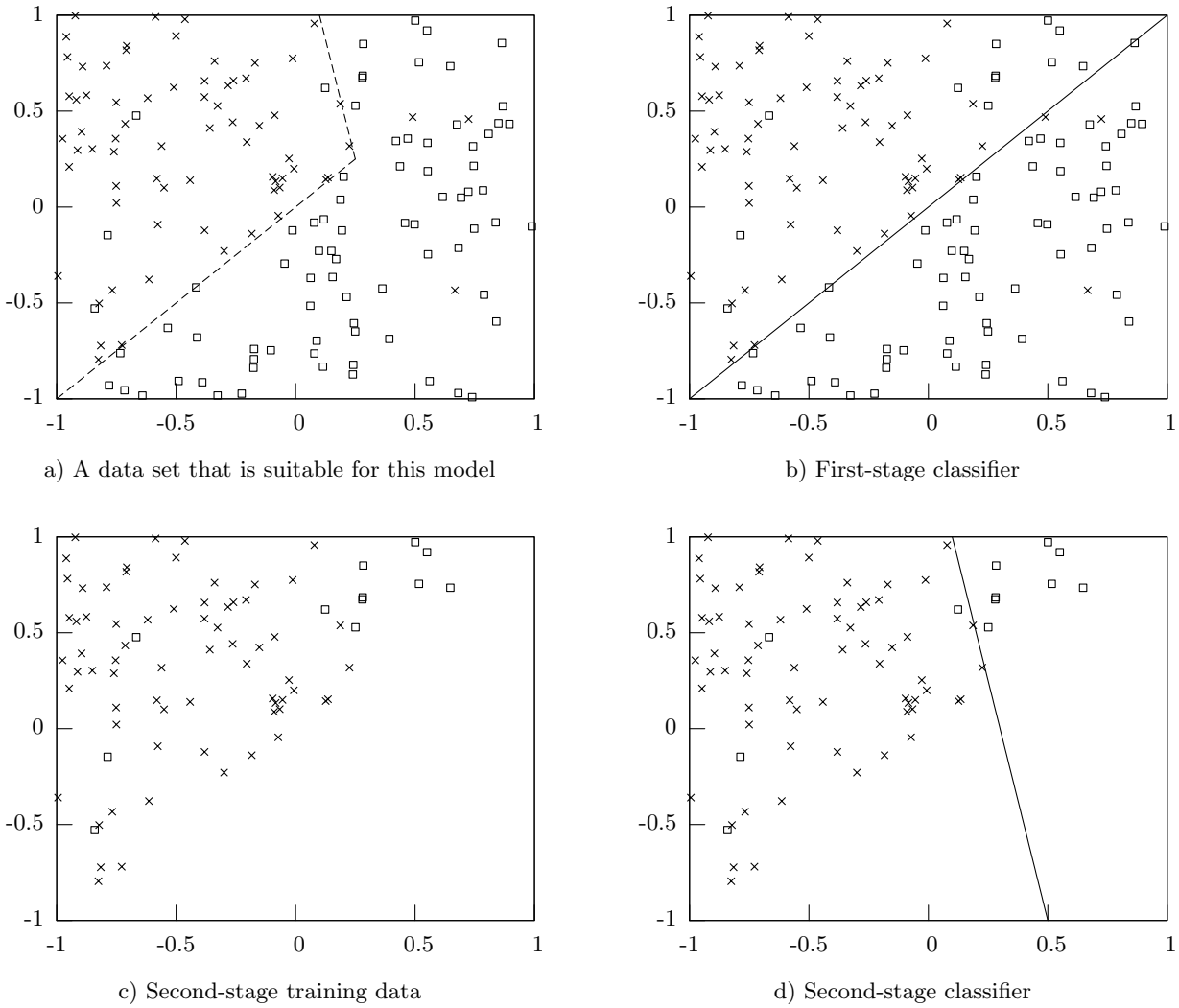
In these experiments, we use two different machine learning algorithms: logistic regression and Naive Bayes. In this background section, we quickly summarize both of these algorithms.

Logistic regression models are also called maximum entropy models in some communities, and are equivalent to a certain kind of single layer neural network. In particular, logistic regression models are of the form

$$P_{\overline{w}}(Y = 1|\overline{x}) = \frac{\exp(\overline{w} \cdot \overline{x})}{1 + \exp(\overline{w} \cdot \overline{x})} \quad (1)$$

In this equation, Y is the variable being predicted (in this case, Y takes the values 0 or 1, with 1 meaning that a message is spam). $\overline{x}$ represents the input data, such as the words in the message; for instance, it can be a vector of 1's and 0's, with a 1 indicating that a particular word is present in the message. Finally, $\overline{w}$ represents a set of weights. These weights indicate the relative weights for each word.

CEAS 2006 - Third Conference on Email and Anti-Spam, July 27-28, 2006, Mountain View, California USA



a) A data set that is suitable for this model

b) First-stage classifier

c) Second-stage training data

d) Second-stage classifier

Figure 1: Illustration of the Two-Stage Filtering Approach: □'s and ×'s represent positive and negative examples, respectively

We learn these weights in such a way as to maximize the probability of the training data. In particular, we find

$$\arg\max_{\overline{w}} \prod_{i=1}^{n} P_{\overline{w}}(Y = y_i|\overline{x}_i)$$

That is, we find a set of weights $\overline{w}$ that make the training data as likely as possible: they do as good a job as possible of predicting that each spam message is spam, and each good message is good. In practice, we almost always regularize these weights with a Gaussian prior [2]. That is, we assume that the average weight should be 0, and that very large weights are unlikely. Letting $N(w; 0, \sigma^2)$ represent the probability of a variable w being generated by a Gaussian distribution with mean 0 and variance σ^2, the actual formula we end up maximizing is:

$$\arg\max_{\overline{w}} \prod_{i=1}^{n} P_{\overline{w}}(Y = y_i|\overline{x}_i) \cdot \prod_{j=1}^{k} N(w_i; 0, \sigma^2)$$

which simply says that we try to find the weights $\overline{w}$ that overall maximize the probability of the training data, and the probability of the weights.

The training algorithm we use is Sequential Conditional Generalized Iterative Scaling (SCGIS) [8], although because logistic regression models have a global optimum, the choice of learning algorithm is typically of little importance, except for training speed considerations.

Naive Bayes is a well known algorithm, especially for spam filtering [15], so we only review it extremely briefly. Naive Bayes computes the probability of a message as a whole: given all possible good messages, what is the probability that this particular message was generated; given all possible spam messages, what is the probability that this particular message was generated. There is an assumption of conditional independence, that all words in the message were generated independently of the others, given the label (i.e., spam or good) of the message. Naive Bayes is concerned with accurately estimating the probability of all messages, and there is no focus on any particular region of an ROC curve.

2.2 Two-stage filtering

We now describe our new method, two-stage filtering, which can be used to improve either logistic regression or Naive Bayes training, or, we assume, other methods as well.

As previously mentioned, most probabilistic machine learning techniques attempt to maximize the probability of training data, when in practice, we care about low false negative rate when using a threshold that produces a low false positive rate. One way to focus on this low false positive region is to explicitly select data for training that is in the region of interest. That is, if we pick training data that is characteristic of the data we care most about, then our learned filter will be well optimized for this particular data.

One way to think about this is that we don't care so much about learning about easy good mail – we will have no trouble recognizing that. We also don't care much about learning about very hard spam – given a low false positive threshold setting, we simply won't catch those messages. We can discard both types of messages, leaving moderately hard and easy spam, and moderately hard and very hard good mail, and then train a filter to explicitly distinguish between those messages: these are the messages that will be important with a low false positive rate.

Conceptually, our learning method will proceed in two stages. First, we will learn a filter using all of our data. Using this filter, we will identify the easy good mail and the very hard spam – i.e. any messages, good or spam, assigned a low probability of being spam. We will discard these messages. Next, we will train a filter using the remaining messages.

The four sections of Figure 1 illustrates how and why such a technique might work. In Figure 1.a, we illustrate a data set for which this method might be appropriate. Notice that the data is almost, but not quite linearly separable. Figure 1.b shows the first-stage classifier we would learn. We then eliminate all of the "easy" good mail (squares in the bottom right) and hard spam (×'s in the bottom right), by picking a threshold for the first-stage classifier. In the illustration, we use a threshold of 50%, but in practice, any threshold can be picked. Depending on the threshold picked, we will end up optimizing on a different type of data, and thus optimizing for a different false positive rate. Figure 1.c illustrates the training data for the second-stage classifier. Finally, Figure 1.d shows the second-stage classifier that is learned. At test time, we use both the first-stage and second-stage classifier. We first classify a test message with the first-stage filter. If the first-stage filter classifies it as good based on the threshold we picked, we use that verdict. If the first-stage filter classifies it as being in the region of interest, we then classify it with the second-stage classifier, and use the second-stage verdict.

The preceding description is somewhat oversimplified. In practice, the learning algorithms we use are almost perfect on the training data. We would thus end up selecting almost no data for our second-stage classifier if we tested our first classifier on the data we used to train it. To solve this problem, we use cross validation. We split our training data into n chunks. We train a classifier using $n-1$ of these, and test the remaining chunk on the respective classifier. We repeat this for each of the n chunks. This helps us identify which messages are truly hard. We thus train n first-stage classifiers to select the data for the second-stage classifier.

The exact training algorithm is illustrated by Algorithm 1.

Algorithm 1 Two-stage learning

```
INPUT: training set T, threshold θ
Split T into ten equal sets, T_1, T_2, ..., T_n
for i = 1 to n do
    Train a classifier p_i using all data T except T_i
    for all t ∈ T_i do
        if p_i(t) ≥ θ then
            U = U ∪ t
        end if
    end for
end for
Train a classifier p_a using T
Train a classifier p_b using U
Return p_a, p_b
```

Notice that the classifier training algorithm is not specified in our algorithm: we can use logistic regression, Naive Bayes, or any other algorithm that returns probabilities or scores.

We spent some time to figure out the best way to combine the two classifiers in practice at test time. Algorithm 2 shows the method we found. We picked this algorithm based both on pilot experiments, trying several different techniques, and based on a theoretical justification. The algorithm is very simple: use the first-stage classifier, and if the result is below the threshold, return the probability from the first-stage classifier; otherwise, run the second-stage classifier, and return its probability.

There are two cases we need to consider. In the first case, a message with features $\overline{x}$ had value $p_a(\overline{x}) < \theta$, so in this case, $p_a(\overline{x})$ is our best estimate of its value, and we return that. But what about the case when $p_a(\overline{x}) \geq \theta$? What is our best estimate of its probability of being spam?

Since we are discussing the messages that are above the threshold based on the first-stage classifier, it is convenient to use a random variable A to represent this event.

$$A \equiv [p_a(\overline{x}) \geq \theta]$$

Notice that for a message with features $\overline{x}$ that has passed the first-stage classifier p_a, event A is true (i.e., $A = 1$).

$$\begin{aligned} P(y=1|\overline{x}) &= \frac{P(y=1,\overline{x})}{P(\overline{x})} \\ &= \frac{P(y=1,\overline{x},A)}{P(\overline{x})} \quad (2) \\ &= \frac{P(y=1,\overline{x},A)}{P(\overline{x})\cdot P(A|\overline{x})} \quad (3) \\ &= \frac{P(y=1,\overline{x},A)}{P(\overline{x},A)} \\ &= P(y=1|\overline{x},A) \end{aligned}$$

Equation 2 is true because A is always true for messages that passed the first-stage. Equation 3 is true because A is true given that the message has passed the first-stage.

This two-stage model is most suitable to data that is not already well modeled by the base classifier. It is particularly appropriate when different regions of the data have different relative distributions. For instance, as we will describe later, there are many examples of words that are more or less indicative of spam in the low false positive region than they are across the training corpus as a whole. Graphically, the synthesized data in Figure 1.a, illustrates one example

Algorithm 2 Two-stage testing

```
INPUT: classifiers p_a and p_b, threshold θ, instance t
if p_a(t) < θ then
    return p_a(t)
else
    return p_b(t);
end if
```

Algorithm 3 Training with utility

```
INPUT: training data T_r, utility values u_p, u_n
T ← φ
for all t ∈ T_r do
    if class(t) = 1 then
        for i = 1 to u_p do
            add example t to T
        end for
    else
        for i = 1 to u_n do
            add example t to T
        end for
    end if
end for
Train a classifier p using T
```

of what such data might look like when viewed in a two-dimensional representation.

2.3 Training with utility

Another different and yet simple method to focus on the low false positive rate region is to treat positive and negative training examples differently. Specifically, the *cost* of a false positive prediction (misclassifying a legitimate message as spam) should be higher. This is roughly equivalent to saying that if we are not sure about whether an email is spam or not, it's better to say that it is not.

The typical way to bias probabilistic machine learning methods is to simple use a threshold. For instance, if missing good mail has cost u_n, and receiving spam has cost u_p, then we can maximize utility by computing

$$\begin{aligned} u_n \cdot P(Y=0|\overline{x}) &> u_p \cdot P(Y=1|\overline{x}) \\ u_n \cdot P(Y=0|\overline{x}) &> u_p \cdot (1 - P(Y=0|\overline{x})) \\ (u_n + u_p) \cdot P(Y=0|\overline{x}) &> u_p \\ P(Y=0|\overline{x}) &> u_p/(u_n + u_p) \end{aligned}$$

That is, from utility theory, we can compute an appropriate threshold, assuming the model is well calibrated. For models like Naive Bayes where the calibration is particularly imperfect, we can examine held out data and set a threshold empirically.

A less common approach is to bias the training data. For example, if we assign 10 as the utility value to negative examples (good mail) and 1 to positive examples (spam), then for each negative training examples, we duplicate it 10 times to form a new set of training examples. This method is depicted in Algorithm 3. This idea dates back at least to Breiman *et al.* [1].

In practice, there are often tricks in the base learning algorithm to avoid generating artificial examples. For example, when training a logistic regression classifier using gradient descent, updates can simply be multiplied by the corresponding utility value. When training logistic regression using SCGIS [8], which examines various counts and expected probabilities, we simply multiply these values by the utility in a few places. For Naive Bayes, the training process, which just uses frequency counting, does not need to be changed. The feature frequencies can be simply corrected by multiplying by the corresponding utility values.

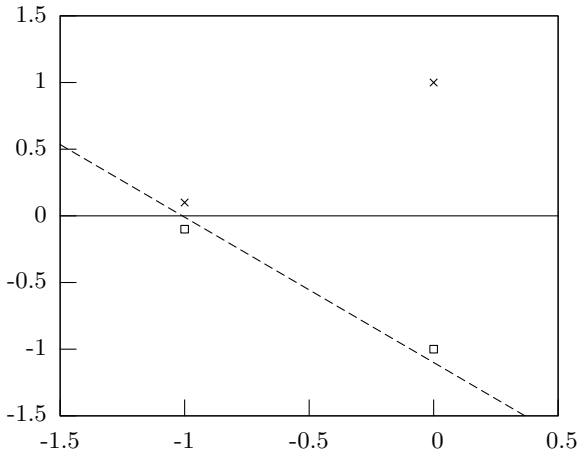


Figure 2: Example of rotation when training logistic regression with utility: □'s and ×'s represent positive and negative examples, respectively

Although this idea is well known in some communities, we are not aware of it having been used before for spam filtering. As we will describe below, although the idea is common, it is not clear in practice how well it will work, compared to simply using a different threshold setting. We have typically seen it used in cases of skewed training data, especially with non-probabilistic models that optimize accuracy, like non-probabilistic decision trees [1], or SVMs [12]: when probabilities are available, the threshold setting technique is more common.

Training with utility can have two effects; the first is to bias the learner, independent of regularization; the second is a regularization effect.

One important advantage of ROC analysis is that it only depends on the order of the assignments. Any monotonic transformation of the score function will not affect the result [6]. Therefore, ignoring regularization issues, one might guess that training with utility simply changes the prior (a monotonic transformation), and has no impact on the ROC curve.

For Naive Bayes, this intuition is correct when regularization (e.g. a Dirichlet prior) is not applied. This is because with duplicated examples, the estimated conditional probability $P(\bar{x}|Y)$ will remain the same. For logistic regression, it turns out that the learned distribution is actually changed. A simple example, as shown in Figure 2 illustrates how this can be the case. The horizontal line in the figure illustrates the logistic regression separator that is learned when the positive and negative utilities are equal. The diagonal line illustrates the separator learned when a much larger utility is placed on the negative examples. As can be seen, utility scores can actually cause a rotation in the learned separator, and rotations (as opposed to shifts) lead to different ROC

curves.

In addition, for both Naive Bayes and logistic regression, there is an effect on the regularization. Consider an example with Naive Bayes where a word A occurs 0 times in good mail, and a word B occurs once in good mail, in a corpus with 10 good messages, and assume plus-one regularization. In this case, we get $P(A|y=0) = 1/11$ and $P(B|y=0) = 2/11$: an occurrence of the word doubles the probability of being good. Now, if we weight the corpus with 10 times as much weight on good mail, we now get a value $P(A|y=0) = 1/101$ and a value $P(B|y=0) = 11/101$: an occurrence of the word increases the relative probability by a factor of 11. The utility weighting thus makes the good estimator much more sensitive to the presence of words: a single good example of a word can substantially raise the impact that that word has. Training with utility has a similar effect for regularized logistic regression, where the increased counts help overcome the prior disproportionately.

The strength of the regularization effect is somewhat surprising. Elkan [4] argued that for Bayesian techniques, there would be little impact from training with utility:

> A Bayesian learning method essentially learns a model $P(X|j)$ of each class j separately. If the frequency of a class is changed in the training set, the only change is to the estimated base rate $P(j)$ of each class. Therefore there is little reason to expect the accuracy of decision making with a Bayesian classifier to be higher with any particular base rates.

Elkan ignored regularization effects. As we will show empirically, training with utility can indeed lead to large improvements with Naive Bayes: the regularization effects are larger than what might be expected, especially in a domain like spam filtering.

Notice that the effect of training with utility may depend on the exact learning algorithm that is used. With logistic regression, we see both hyperplane rotation and shift, even without regularization effects, and we see additional effects because of regularization. For separable data trained with a hard margin SVM, utility weighting has no impact at all: there is no regularization, and since only the support vectors define the separating hyperplane, having more identical support vectors will not change the hyperplane in the hard-margin case. For soft margin SVMs, training with utility has an impact both because of regularization, and because it can cause hyperplane rotation; but the kinds of examples that will cause a hyperplane to rotate for a soft-margin SVM are rarer than they are for logistic regression.

3. EXPERIMENTAL RESULTS

In this section, we describe experimental results of applying the proposed techniques on a large corpus in a realistic setting. We first introduce the data we use, and then give ROC curves for each of the new methods applied to logistic regression and Naive Bayes. Finally, to understand the behaviors of the classifiers learned in the two-stage approach, we also examine the weight differences of features, showing that some features do change weight substantially.

3.1 Data

For a practical research topic like anti-spam, it is best to evaluate techniques in a realistic setting. Fortunately, we have access to the Hotmail Feedback Loop data, which is collected by polling over 100,000 Hotmail volunteers daily. In this feedback loop, each user is provided with a special copy of a message that was addressed to him, and is then asked to hand-label this message as Good or Spam. The original copy of the message might have been deleted, or been put in the junk folder, or might be in the user's inbox already: this is an additional copy. By asking users to label their own messages, we believe we get judgments that only they can make, across many languages, and with an up-to-date data source. Analyzing this data, we find that very roughly 3% of user labels are errors. In some cases, we can be sure that a message is labeled in error. Examples include messages labeled as good that are really virus or phishing messages, or a message from an amorous young woman that appears to be specially sent to the recipient, but is in fact sent to many thousands of people. In other cases, it is extremely difficult to tell whether a user has made an error or not. We might find for an identical message that 20% or 50% or 80% of users labeled it as spam, while the others considered it good. Unfortunately, there is often no way to know whether some of these users made a mistake or got tricked, or some users were spammed while others opted in for the same content (e.g. a newsletter).

For the experiments in this paper, the training data are messages received between July 1st, 2005 and November 30th, 2005. We randomly picked 5,000 messages from each day and the total number of messages for training is 765,000. Similarly, the testing data is taken randomly from messages received between December 1st, 2005 and December 15th, 2005. 10,000 messages were drawn from each day, which constructs a collection of 150,000 testing messages. From each message, we extracted features consisting of subject keywords and body keywords that occurred at least three times in the training set. This is a subset of the usual features we use, which also includes many proprietary features; we used this subset in this paper to make it easier to replicate our results; results trained with our full feature set show similar improvements.

3.2 Results

Note that the results reported here are quite a bit worse than our true performance in commercial applications for a number of reasons. First, the machine learning system we describe here is only one component of a larger system that also uses other techniques, such as IP blocklists, IP safelists like Bonded Sender[1], user supplied safelists, etc. Second, in order to make the experiments more replicable, we chose to use a subset of our full feature set, only subject and body features, rather than including other proprietary features. Third, in order to simplify the experiments, we used somewhat less training data – 765,000 messages – than we use commercially, although this is still almost an order of magnitude larger than any publicly available spam collection. Fourth, most "errors" at a false positive rate of 3% or below turn out not be spam filter errors, but instead are labeling errors. As we mentioned, we find that users typically make approximately 3% labeling errors, so most "false positives" at a 3% error rate or below are not errors at all. Real errors that we do find at low false positive rates are only very rarely personal mail from one user to another; more typically they are legitimate advertisements that users may be

[1]http://www.bondedsender.com

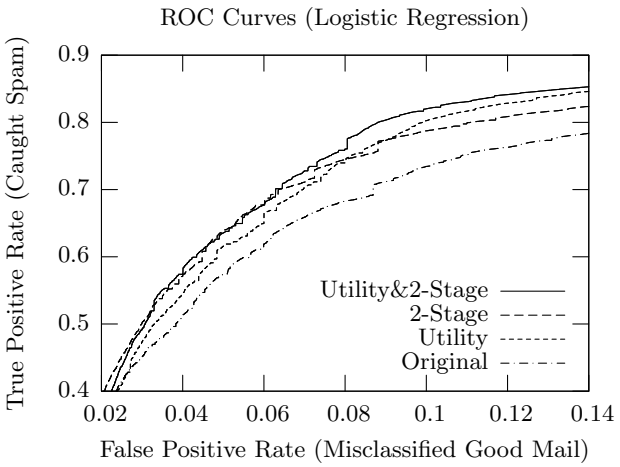


Figure 3: The performance of four different approaches with logistic regression learning

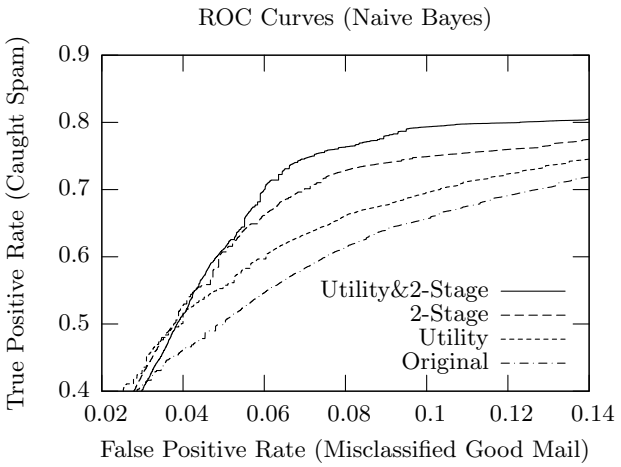


Figure 4: The performance of four different approaches with Naive Bayes learning

just as happy to have in their junk folder. The false positive rates in the results below may seem surprisingly high, but the regions displayed in the charts below do correspond to real regions of interest for this particular component of the larger system, especially at the "junk folder" threshold – that is, for messages that are put into a junk folder, rather than deleted outright. Of course, when combined with other techniques, including user-supplied safelists and global IP safelists, and with our full training set and features, actual false positive rates are substantially lower.

We evaluated our methods with two different learners – logistic regression and Naive Bayes. For each of the learning algorithms, we tested four cases: *original*, which uses just the baseline learning algorithm; *utility*, which weights negative examples (good mail) as 10 times more important than the positive examples (spam); *2-stage*, where the threshold is picked at roughly the 0.2 false positive rate; and *utility and 2-stage*, which is the method that combines both *utility* and *2-stage* methods. The performance in ROC curves is shown in Figure 3 and Figure 4, where each point of a given curve corresponds to a different threshold value.

As can be seen in the figures, both the utility and 2-stage methods are consistently better than the original approach for this data set. Also shown in Figure 3, the utility method and the 2-stage method perform fairly closely, and indeed cross. The utility method is slightly better for higher false positive rates, while the 2-stage method is better at the lowest false positive rates. Their false negative reduction rates compared to the original curve are about 18%. The combined approach works better than either method separately: its false negative reduction rate is a little more than 20% and its performance roughly matches the 2-stage method when the false positive rate is below 0.06.

Results are similar for Naive Bayes, although the improvements are even larger. As shown in Figure 4, when using Naive Bayes, the 2-stage method is consistently more effective than the utility method. Compared to the original curve, the relative false negative rate reduction for the 2-stage method is about 25% in this ROC region, while the utility method has roughly a 10% false negative reduction rate. The effect of combining these two approaches is in general better than each individual approach. When applied on Naive Bayes, the combined reduction rate can be as high as 40% relative compared to the original curve in some regions. However, when the false positive rate is lower than 0.06, the advantages of combination vanish, and the 2-stage method alone is better.

It is also interesting to compare Figure 3 and Figure 4. Here, we see that unsurprisingly, filters trained based on logistic regression are substantially better than those trained on Naive Bayes. However, the performance gap shrinks after the two-stage and utility methods are applied. Depending on which false positive rate point we pick, the combined approach on Naive Bayes can be competitive with the combined approach on logistic regression, and can outperform the original logistic regression learner.

3.3 Feature weight analysis

In the two-stage approach, the two classifiers are trained using the same feature set but with different examples. It is therefore interesting to see how the same features are treated differently by these two classifiers. In particular, we would like to find features that change weight substantially from the first stage to the second stage, to understand better why the 2-stage approach helps.

We examined the weights of the first-stage and the second-stage classifiers learned with 2-stage logistic regression, and found that some features are treated very differently. For example, words like `unsubscribe` and `click` are considered more spammy by the first-stage classifier. One possible reason might be that both spam messages and commercial newsletters often contain these words. A message without these words seems more like a legitimate message. However, to the second-stage classifier, these words may not be as informative as in the first-stage. Other features that are more helpful to distinguish spam and spam-like good mail will have higher weights now.

4. RELATED WORK

In this section, we briefly survey related literature. There is a rich literature on cost-sensitive learning, and it is beyond the scope of this paper to survey all of it: we focus on some

of the most important related work.

Training with utility, sometimes called stratification, is a well known method dating back at least to Breiman [1]. Despite being well known, as we noted in Section 2.3, Elkan [4] has argued that in at least some cases, including for Naive Bayes, this method will not work well. Our contribution then is threefold: the empirical result that this method works well for spam filtering; an explanation based on regularization for why it works better than what Elkan suggested; and the observation that the reasoning that applies to generic probabilistic techniques like Naive Bayes does not apply to discriminative probabilistic techniques like logistic regression.

The technique of training using utility has also been used in several tasks. For example, it has been used in changing the prior when building a decision tree [1]. It is also used to improve the SVM performance when handling highly-skewed data or having different misclassification costs [12, 10].

Lowd and Meek [11] used re-weighting (similar to our utility weighting) to make a spam filter adapt more quickly: they weighted recent examples more heavily than older examples. Their goal however was different from ours: faster adaptation, particularly to new spam, rather than more sensitivity to false positives; the details of which messages were re-weighted thus also substantially differed.

The two-stage filtering approach appears to be novel. Although there are many related techniques, which we will survey in this section, the idea of using multiple filters specifically to focus the training on the low false positive region is new. The 2-stage filtering technique can be thought of as a special form of decision list [13] with only two layers, which in some domains is also called as *cascade* of classifiers. While there are many previous uses of multiple filters in general and cascades in particular, they are all focused on other goals: improving accuracy; or improving all-around probability estimates, with no particular focus; or for speeding up the system.

One type of previous work used cascades primarily for the purpose of reducing training speed. This is a moderately common technique, and we cite only two examples here. Viola and Jones [16] used cascades to improve speed in an object detection task. By eliminating the least likely examples with simple classifiers, they were able to apply more complex classifiers to a relatively small number of regions. Roth and Yih [14] used a similar technique in text processing, applying a simple classifier to a large number of examples, and a more complex one to the smaller number of remaining cases. Unlike either of these (or many similar papers), our second-stage classifier uses exactly the same features as our first-stage classifier, and takes the same amount of time.

Training classifiers using different subsets of the data is also similar to Boosting [7, 5]. Our methods are different from Boosting in several ways. First, we use a threshold to select a region of interest, which specifically focuses on low false positive region; boosting methods are usually symmetric. Second, we use cross validation, so that we can actually select points near this region. Boosting in general focuses on improving accuracy at the 50/50 point, or on improving the overall probability estimate, while we focus on a particular region. Third, at test time boosting takes a linear combination of results, while we combine classifiers in a different way. In principal, our technique could be combined with boosting, using boosting in both the first stage and the second stage, in each case improving classifier accuracy.

MetaCost [3] is a technique for use in cost-sensitive domains. At first glance, it appears similar to our 2-stage learning technique. MetaCost uses bagging to get probability estimates even from non-probabilistic classifiers, and to improve the estimates from probabilistic ones. However, MetaCost does not actually use the desired utility as part of the training mechanism: it is focused on getting accurate overall probability estimates so that the standard utility-theoretic thresholding can be used. It is also much more expensive than our method at test time, since it requires using the complete set of bagged classifiers on each test instance, while we use at most two classifiers at test time.

Kolcz [9] describes a technique to do local (test document specific) feature selection for Naive Bayes, with the goal of improving spam catch rates near the zero false positive point. This method is very different than any of the techniques described here. In principal, it could be combined with either the utility or two-stage filtering techniques. We did not implement and compare to this technique in part because our goal was not to improve Naive Bayes in particular, but was rather to show that the two-stage filtering technique works well with more than one learning algorithm.

5. CONCLUSIONS

It is often assumed that because learning methods like logistic regression and Naive Bayes return probabilities, the best way to optimize them for a low false positive rate is to simply choose a low probability threshold [4]. But in practice, we find that we can do a better job by specifically attempting to model the region of interest. We have show two different techniques that reduce false negative rates in the low false positive region.

First, we can train with utility. As we have shown, this can have two different effects. Intuitively, one might think that this simply shifts a linear separator, and thus has no impact on an ROC curve. As we have shown, in the case of logistic regression, this will actually change the orientation of the separator, even without regularization. For Naive Bayes, however, it does not impact the separator orientation in the unregularized cases. In practice, both logistic regression and Naive Bayes are regularized, in which case training with utility has an impact both for Naive Bayes and for logistic regression.

Second, we have introduced a novel approach, two-stage filtering. This method specifically chooses training data from our region of interest. We have shown that this technique works well across at least two different learning algorithms.

We have shown that both two-stage filtering and training with utility can result in substantially improved spam filtering at a given low false positive rate. We can reduce missed spam by as much as 20% relative for logistic regression, and by as much as 40% relative for Naive Bayes. These substantial improvements can be very helpful in a practical system.

6. ACKNOWLEDGMENTS

We thank John Platt and Chris Meek for many helpful discussions. We are also grateful to anonymous reviewers for their valuable comments.

7. REFERENCES

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Tress*, pages 114–115. Wadsworth, 1984.

[2] S. F. Chen and R. Rosenfeld. A gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Computer Science Department, Carnegie Mellon University, 1999.

[3] P. Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 155–164, 1999.

[4] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[5] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 97–105. Morgan Kaufmann, San Francisco, CA, 1999.

[6] T. Fawcett. ROC graphs: Notes and practical considerations for researchers. Technical Report HPL-2003-4, HP Labs Tech Report, 2003.

[7] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[8] J. Goodman. Sequential conditional generalized iterative scaling. In *ACL '02*, 2002.

[9] A. Kolcz. Local sparsity control for naive Bayes with extreme misclassification costs. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 128–137, 2005.

[10] A. Kolcz and J. Alspector. SVM-based filtering of e-mail spam with content-specific misclassification costs. In *Proceedings of the TextDM'01 Workshop on Text Mining*, 2001.

[11] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *The Conference on Email and Anti-Spam (CEAS)*, 2005.

[12] K. Morik, P. Brockhausen, and T. Joachims. Combining statistical learning with a knowledge-based approach – a case study in intensive care monitoring. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 268–277, 1999.

[13] R. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[14] D. Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1257–1263, 2001.

[15] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *AAAI'98 Workshop on Learning for Text Categorization*, July 1998.

[16] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR-01*, 2001.