Going Mini: Extreme Lightweight Spam Filters

D. Sculley
Google, Inc.
dsculley@google.com

Gordon V. Cormack
University of Waterloo
gvcormac@uwaterloo.ca

ABSTRACT

In this paper, we examine the practicality of extreme lightweight "mini" spam filters, which rely on only a small handful of inexpensive features for classification. Such filters would be cheap enough to store and serve in RAM even for email systems with very large user bases, allowing effective personalization at scale. In this paper, we propose and test a variety of both traditional and novel methods for supervised training of effective mini-filters reliant on only 2^1 through 2^6 features, rather than the more typical 2^{20} features. The best of these mini-filters are found to approach the classification performance of strong classifiers trained on the full feature set of millions of features. This is most notably the case in the real-world scenario of noisy training labels provided by non-expert humans. When mini-filters are used to augment a generic global filter, the combination is found to equal or surpass the performance of state of the art classifiers using the full feature set. These results suggest that mini-filters may be an effective approach for achieving large-scale personalized filters at low cost.

1. INTRODUCTION: WHY MINI?

In recent years, state of the art supervised spam filtering methods for separating email *spam* from *ham* (non-spam messages) have commonly employed very large large, sparse feature sets. An example of this is the success of Logistic Regression and Support Vector Machine (SVM) methods at the TREC 2007 spam filtering benchmark competition, in which top performing filters used a binary 4-*mer* feature space of all possible four-byte character strings [8].

While it is not particularly challenging for a large-scale email system to serve a single, global filter containing millions or billions of features, it may be problematic if we wish to serve a large number of *personalized* filters which vary by user. Because fast serving requires models to be stored in RAM, rather than cheaply on disk, there is significant incentive to find models for filtering that rely not on millions but on dozens of features. Furthermore, mini-filters reliant on very few features may allow extremely fast classification due to excellent memory locality of the model. Finally, mini-filters may be effective in resource-constrained environments such as hand-held mobile devices.

In this paper, we attempt to construct effective email spam filters using very small feature sets, containing between 2^1 and 2^6 features drawn from the same binary 4-*mer* feature space that gave strong performance when each of the millions of possible features were considered [8]. To our knowledge, this is the first systematic study of training and evaluating spam filters that use very few features for classification. We call such filters *mini-filters*.

Somewhat to our surprise, we found several inexpensive methods capable of training mini-filters that are competitive with filters using state of the art training methods such as linear SVM, regularizard Logistic Regression, and the popular Naive Bayes. This suggests that typical high-dimensional feature sets for email spam filtering are composed of many redundant or irrelevant features. The performance of the mini-filters is most notably competitive with filters trained on all features for the real-world setting in which filters are provided with potentially noisy training labels. Part of this effect may be ascribed to the low VC dimension of a typical mini-filter.

Finally, we investigate a natural scenario for a large scale system, in which a personalized mini-filter is used to augment the performance of a generic global filter. We find that when used together, the combined performance of a mini-filter and a generic global filter as a combined ensemble gives improvement over either method individually, and exceeds the performance levels of linear SVMs using all possible features. This result shows that personalization of a generic global filter can be effectively achieved at low cost through the addition of lightweight, personalized mini-filters, and encourages the development and application of further work in personalized mini-models for message filtering, foldering, and prioritization.

The remainder of this paper proceeds as follows. The formal problem definition and preliminaries are given in Section 2. Background in related general techniques for training models reliant on few features and specific detail for methods used in our experimental comparison are given in Section 3. The fourth section describes two new methods for training mini-models: a novel tournament-based feature selection method and a fast decision list variant for sparse data. Our data sets are described in Section 5, including the low-cost acquisition of labels from non-expert "normal" humans using crowd-sourcing methods. Section 6 contains our experimental results, and the final two sections of this paper contain a brief overview of related work in personalization for spam filtering and a concluding discussion.

CEAS 2009 - Sixth Conference on Email and Anti-Spam July 16-17, 2009, Mountain View, California USA

2. PRELIMINARIES

In this section, we briefly give a formal definition of the problem we examine in this paper and the notation used, and pay some attention to the definition of what constitutes a *feature* to ensure fair comparisons.

2.1 Problem Statement

We are interested in finding a model $f(\cdot)$, that maps messages represented as m-dimensional vectors $\mathbf{x} \in \mathbb{R}^m$ from some distribution D to predicted class labels $f(\mathbf{x}) \in \{+1, -1\}$ for *spam* and *ham* (not spam), respectively, such that the predicted class label agrees with y, the true class of the message. Furthermore, we wish $f(\cdot)$ to rely on at most t features. For example, if our classifier is a linear function $f(\mathbf{x}) = sign < \mathbf{w}, \mathbf{x} >$ (where $sign(x)$ returns $+1$ for $x \geq 0$ and -1 otherwise) dependent on a weight vector $\mathbf{w}$, we require that $||\mathbf{w}||_0 \leq t$. That is, we require that $\mathbf{w}$ have at most t non-zero values.

More generally, we say that a model $f(\cdot)$ is t-reliant if there exists a diagonal transformation matrix T with every $T_{(i,i)} \in \{0, 1\}$ and $||T||_0 \leq t$ such that $\forall \mathbf{x} \in D : f(\mathbf{x}) = f(T\mathbf{x})$. That is, if we can zero out all but a particular subset of t features without changing any predictions by our model, then our model is t-reliant. In this paper, we define a *mini-filter* as a filter that is t-reliant with $t \in [2^1, 2^6]$.

2.2 Defining Features

For fair comparison in this preliminary investigation, it is necessary to restrict the definition of a *feature* in this paper. This is because a mini-filter that uses the output of t costly spam filters as features would be t-reliant, but would be unfair to compare with a filter using only t character-level substrings. Thus, we leave the investigation of using mini-filters in systems deploying a larger number of sophisticated models to future work, and use the same feature space for all methods to ensure fair comparison.

The feature space used for all methods in this paper is the binary 4-*mer* feature space that gave best results with several learning methods at the TREC 2007 spam filtering benchmarks [8]. A 4-*mer* is a consecutive substring of four byte characters; mapping a string to the space of binary 4-*mers* requires finding all the unique (possibly overlapping) 4-*mers* in the string. For example, the 4-*mers* in the string `abc_ba` are `abc_`, `bc_b`, and `c_ba`. Each 4-*mer* found in the string is given a score of 1 in its corresponding feature, and each 4-*mer* not found is scored 0 in its corresponding feature.

An exception to this strict limitation on features is given in our final set of experiments, in which a personalized mini-filter is combined with a generic global filter. In this case, the output of the generic global filter is used as an additional input feature to the mini-filter. In these experiments, all tested methods are given the same augmented feature set.

Finally, methods reliant on a bias term (for example, a term $\mathbf{x}_0$ always set to 1) are considered to be using an extra feature compared to methods not reliant on a bias term.

3. BACKGROUND: FEATURE SELECTION

The problems of feature selection and sparse model (models relying on few features) is an established and current area of research in machine learning [14]. In this section, we review several general approaches and note their associated tradeoffs. We also give specific detail on those methods used in our experimental comparisons.

3.1 Global Feature Selection Methods

The most widely used feature selection methods, in practice, are global methods such as selecting the top t features by information gain [31], as measured with respect to features and class labels in the training set. The model is then trained only on the t features. Similar ideas include selecting the top t features using other global quality measures such as those based on rank order statistics [23] or the magnitudes of weights in a linear model [1].

Global feature selection methods have the benefit of being cheap to apply, but they do not take into account relationships among features. In the context of this paper, the main drawback is that global methods may be prone to selecting many redundant features, rather than independent features. In practice, this drawback is dealt with by letting t be relatively large (on the order of hundreds or thousands). To our knowledge, the performance of global information gain for selecting very small feature sets for email spam filtering has not been previously evaluated.

We include the global information-gain (also called mutual information) approach in our experimental comparisons in Section 6, as this is a popular and effective choice for feature selection in many domains [14]. In the case of discrete variables and discrete class labels (as we have with binary 4-*mers* and class labels of $\{+1, -1\}$), the information gain $I(f)$ of feature f is defined:

$$\sum_{X \in \{0,1\}} \sum_{Y \in \{+1,-1\}} P(\mathbf{x}_f = X, y = Y) \log \frac{P(\mathbf{x}_f = X, y = Y)}{P(\mathbf{x}_f = X) P(y = Y)}$$

Here, $P(\mathbf{x}_f = X)$ is the observed probability that the given feature f has value X in an example $\mathbf{x}$ randomly selected from D, $P(y = Y)$ is the observed probability that the randomly chosen $\mathbf{x}$ has class label Y, and $P(\mathbf{x}_f = X, y = Y)$ is the probability that a randomly selected $\mathbf{x}$ has both feature value f with value X and class label Y. Each of these quantities is traditionally estimated by counting occurrences in the training data.

3.2 Greedy Methods

The general problem of finding the t best features for a model is NP hard, as $\binom{m}{t}$ subsets must be considered. One practical strategy for finding an approximate solution to this problem is the greedy heuristic [14], for which there are several variants commonly used in machine learning.

The most generic form of this greedy approach involves the use of a *wrapper* method around a learner [17]. In the *forward-selection* wrapper approach, for each of t iterations, a unique model is trained for each of the m features using the current feature set (beginning with the empty set) augmented with the feature under consideration. At the end of each round, the model with the best classification performance is selected and its feature set is used as the base feature set in the next round. Although this wrapper approach may be used with any learning algorithm, it may involve considerable cost (for example, in conjunction with an SVM learner) and is not applied in this paper. Similarly, we do not consider wrapper methods using more expensive search heuristics such as genetic algorithms.

In this paper, we do test two greedy methods in the spirit of the forward-selection wrapper method. These are *short decision trees* and *boosting with early stopping*.

3.2.1 Short Decision Trees

A decision tree is a tree containing conditions (based on features and thresholds) at interior nodes, and prediction values at leaves [22]. For the purposes of this paper, it is sufficient to assume that the decision tree is a binary tree, and that the condition at interior nodes is a test for the presence or absence of a binary 4-*mer*.

Decision trees are commonly trained using a recursive strategy, successively partitioning the training data set based on the feature that gives the best information gain for the current partition [22]. Because of this recursive partitioning, redundant features tend not to be selected at different levels of the tree as they do not increase information gain.

Furthermore, we can make a decision tree t-reliant by limiting the depth of the tree to at most $1 + ceil(\log_2 t)$. Thus, short decision trees appear to be a good candidate for mini-filter models, and are tested in our experimental evaluation.

3.2.2 Boosting with Early Stopping

The idea behind *boosting* is that weak learners, that produce models with at least $0.5 + \epsilon$ accuracy on any distribution, can be combined into an aggregate strong classifier with accuracy as close to 1.0 as desired through a process of training new weak learners on successively more "difficult" distributions of training data and aggregating the resultant weak learners with a weighted linear combination [13].

The AdaBoost algorithm is a practical boosting algorithm, managing the weighting of the weak learners and the construction of each successive distribution of training data [13]. A commonly used weak learner in such situations is a *decision stump*, which is a decision tree with a single feature; boosted decision stumps often give strong classification performance.

Because boosting proceeds in iterative rounds, with each round focusing on a more difficult training distribution, we may stop early after the completion of any iteration. (Early stopping for boosting can also be used as a defense against over-fitting [33].) Stopping after t rounds means that the model will contain t weak learners; if these weak learners are decision stumps then the aggregate model depends on at most t features and is therefore t-reliant. In this paper, we use decision-stumps boosted with the AdaBoost algorithm stopping after t rounds for mini-filter training. As experiments in Section 6 show, this approach is extremely effective.

3.3 Sparsity-Encouraging Regularization

The problem of finding sparse models has been cast as an optimization problem involving an L_0-norm penalty[1] for regularization:

$$\min \lambda ||\mathbf{w}||_0 + \text{loss}(\mathbf{w}, \text{D})$$

over models represented as weight vectors $\mathbf{w}$, with respect to a given loss function and a given set of training data D, and with the parameter λ determining how much weight to give to the possibly conflicting goals of minimizing test error and minimizing model complexity [14].

Because this optimization problem using an L_0-norm penalty is non-convex (and NP-hard to solve in general), a standard solution is to approximate this using an L_1-norm penalty, which restores convexity and allows efficient solution [3]. However, using λ to tune training so that models contain specific numbers of features is problematic in practice. In our preliminary experiments using an L_1-norm regularization method suitable for high-dimensional data (using a trucated gradient descent method [19]), we were unable to train effective models with 2^6 or fewer features, and left further investigation of this and similar approaches to future work. (The difficulty for us was finding a suitable value for λ to produce feature sets of the desired size.)

[1]Recall that the L_0 norm is equivalent to the number of non-zero elements of a vector.

3.4 Dimensionality Reduction Methods

Dimensionality reduction methods are another area of machine learning and information retrieval that has seen an explosion of work. Several such methods revolve around the concept of reduced-rank approximation of the matrix of training data, via singular value decomposition or related methods [29], a popular example of which is the latent semantic analysis algorithm from information retrieval [11]. Graphical models for topic modeling are another rich set of techniques that are applicable for dimensionality reduction of this form, and include probabilistic latent semantic analysis [15] and author-topic models [25]. A particularly relevant example of this technique is the Author-Recipient-Topic model of McCallum *et al.* [21] for modeling topics and network relationships on a per-author, per-recipient, and per-message basis for email data.

Such methods may be applicable and effective for the task of learning personalized mini-filters; it is possible to imagine that different users prefer different mixtures of topics, for example. However, as noted in Section 2.2, applying such methods here would be effectively equivalent to creating features representing sophisticated models. This is outside the scope of this preliminary investigation into the training and effectiveness of mini-filters based on simple, cheap features.

4. ALGORITHMS

In this section, we describe two new methods for training sparse models: a fast greedy decision list algorithm suited for sparse data, and a feature selection method based on a tournament system.

4.1 Fast Decision List Training on Sparse Data

Decision lists are a hypothesis class that are well studied in the machine learning theory community [24, 2], but are relatively unrepresented in application. Formally, a decision list L of length m is a list of $m + 1$ if-then decision rules linked by else clauses as follows:

```
If $x_{L_1} > t_1$ then predict $y_1$
else if $x_{L_2} > t_2$ then predict $y_2$
else if ...
else predict $y_{m+1}$
```

where each L_i is the index of a particular feature in the feature set, each t_i is a threshold for that feature, and each y_i is a decision value in $\{-1, +1\}$ to return if rule i is satisfied. It is commonly known that a decision list on boolean features can be represented with a linear weight vector $\mathbf{w}$, with $\mathbf{w}_{L_i} = sign(y_i)2^{1-i}$, where the function $sign(y)$ returns 1 for $y \geq 0$ and -1 for $y < 0$. (To see this, observe that if rule i is the first of the t rules to be satisfied, then its weight of

Algorithm 1: DecisionList-H(S, t, c)
Given: Training set S of labeled examples with boolean features, number of iterations t, correction term c
Produce: Weight vector ($\mathbf{w}$) with new weights for classification.

```
w = 0
For i from 1 to t do:
      counts[][] = 0
      For each example (x, y) in S do:
           For each feature index j where x_j == 1 do:
                  counts[j][y] := counts[j][y] + 1
      H[j] = 0
      For each feature index j do:
           prob-j-pos = (counts[j][1] + c) / (counts[j][1] + counts[j][-1] + 2c)
           prob-j-neg = (counts[j][-1] + c) / (counts[j][1] + counts[j][-1] + 2c)
           H[j] = prob-j-pos * log(1 / prob-j-pos) + prob-j-neg * log(1 / prob-j-neg)
      Choose j with minimum value in H[j]
      Set w_i = sign(counts[j][1]) - counts[j][-1]) * 2^{1-t}
      Remove all (x, y) with x_j == 1 from S
      If |S| == 0 then break
Return w
```

Figure 1: A greedy decision list algorithm, using Laplace-smoothed entropy as the purity function.

2^{1-i} is greater than $\sum_{k=i+i}^{t} 2^1 - k$, which is the largest possible total weight for all the remaining rules.) Classification of a new example $\mathbf{x}$ may then be performed with the familiar linear classification function $f(\mathbf{x}) = sign(<\mathbf{x}, \mathbf{w}>)$.

There exist algorithms for finding the shortest consistent decision list with a set of data, if one exists to fit the data [2]. However, finding the shortest decision list to minimize classification errors is in general an NP hard problem if there is no decision list that fits the data perfectly (as is likely in the case of class label noise). Thus in practice we must resort to a greedy approximation strategy.

In this paper, we train decision lists in greedy fashion using the algorithm outlined in Figure 1. The main idea is that at each iteration, we select a feature that slices off a subset of the data that is as close as possible to being purely single-class as possible, while also being as large as possible. We define a slice S_i from S as the subset of examples $\mathbf{x}$ in S that have $\mathbf{x}_i = 1$. We measure the goodness of a slice using its entropy, $H(S_i) = \sum_{y\in\{+1,-1\}} p_{y,i}(-\log p_{y,i})$ where $p_{y,i}$ is the probability that a randomly selected example from S_i has class label y.

This approach of using entropy as the purity function for decision lists goes back to the CN2 algorithm by Clark and Niblett [6]. However, for small slices the estimates of each $p_{y,i}$ may be severely under or over estimated which may overly encourage the selection of small pure slices, rather than a large nearly-pure slice. Thus, we use a Laplacian correction term c in the estimation of these probabilities to encourage the preference of large, relatively pure slices over small, perfectly pure slices. (The value of c is found by tuning on separate tuning data.) To our knowledge, this approach has not been used, although a Laplacian correction for a similar purity function based on accuracy has been applied [5]. The inclusion of this correction term significantly improved performance over a version of this algorithm without the correction term in preliminary experiments.

Finally, we note that prior theoretical work on decision lists and applications in natural language processing have focused on lists with complex conjunctions [24] in decision nodes resulting in slow algorithms that include a beam-search step at each iteration [32] and models as expressive (and susceptible to over-fitting) as decision trees. Because we do not require complex conjunctions for our small models, and furthermore optimize for the case of binary-valued features, we are able to construct a fast training algorithm for decision lists that exploits sparsity. This algorithm is fast, as each of the t iterations may be completed in time $O(ns)$ where n is the number of examples remaining for the iteration, and s is the maximum number of non-zero features in any example. This efficient algorithm for decision list training of sparse, high-dimensional data has not previously appeared.

Algorithm 2: TWFS(S, $G()$, t)
Given: Training set S of labeled examples with boolean features, global scoring method $G()$, number of features t to select
Produce: Weight vector ($\mathbf{w}$) with new weights for classification.

```
For each feature f present in S do:
    Compute G[f] := G(f)
winners[] := 0
For each x in S do:
    Select feature f with highest G[f] and x_f ≠ 0
    winners[f] := winners[f] + 1
Return t top-scoring features f in winners[]
```

Figure 2: Tournament Winners Feature Selection Algorithm.

	`trec05p1`	`trec07p`	`ceas08-1`
TRAIN SET SIZE	10,000	10,000	3,067
TEST SET SIZE	82,189	65,419	206,207
SPAM RATE	0.57	0.67	0.80
HUMAN ERROR RATE	0.04	0.05	0.16

Table 1: Summary of the benchmark data sets.

4.2 TWFS: Tournament Winners Feature Selection

As noted in Section 3.1, selecting features by a global method such as global information gain may result in the selection of many redundant features with little additional value. This may be especially problematic in email spam settings, in which spammers send many duplicate or near-duplicate messages. In this case, the near-duplicate spams will contain many redundant features with high information gain (increasing with the number of near-duplications) that could dominate a global information gain selection process.

To combat this effect, while maintaining the low cost of a two-pass algorithm, we propose a method called Tournament Winners Feature Selection (TWFS). Unlike the decision list algorithm described above, TWFS does not iteratively partition the data, and is thus amenable to incremental updates.

The main idea of TWFS is to conduct feature selection as a tournament, with each message representing a single competition. Before the tournament begins, each feature is given a global score (described below). For each message, the features in that message are ranked by score, and the top scoring feature in the message is considered the winner of the competition. Ties are broken arbitrarily but consistently. For the tournament, features are ranked by the number of competitions they have won, and the top t of these winners are selected as features for model training. In our experiments with TWFS, we test both linear SVM and regularized Logistic Regression for training models on the features selected with TWFS.

Pseudo-code for the TWFS method is given in Figure 2. We tried two methods for the global scoring function $G()$. The first was global information gain, and the second was the magnitude of per-feature log-odds multipliers found by training a Logistic Regression model (using a simple stochastic gradient descent method). We found the log-odds multipliers to give better performance in preliminary trials compared with global information gain, and used this method in all TWFS experiments reported in Section 6.

TWFS can be considered a single-vote election strategy, and has several qualities of a desirable election method [10]. Among these, TWFS is monotonic: monotonic transformations of the global scores do not change the outcome of the tournament. Additionally, TWFS is Pareto-optimal: if every competition results in i beating j, then feature j is not selected. Furthermore, in the absence of ties and when run to exhaustion, TWFS satisfies the Condorcet criterion: if there is feature i that wins every pairwise-preference comparison with other features, then feature i will be selected. (Note that other features not winning every pairwise preference contest may also be selected.)

5. EXPERIMENTAL DATA

This section describes the benchmark data sets used for experimental evaluation, with both gold-standard training labels and noisy labels from non-expert humans. Because human-labels were not available for two data sets, we acquired them at low cost using Amazon's Mechanical Turk.

5.1 Benchmark Data Sets

We performed experiments on three large, publicly available benchmark data sets: `trec05p-1` [9], `trec07p` [8], and `ceas08-1`.[2] A summary of each data set is given in Table 1. Although these data sets contain messages sent to several recipients, the scope of these recipients is narrow enough in each data set that we feel it mimics the personalization task we aim to model in our experiments. (This intuition is further supported by the similarity in benchmark results from these corpora and the private, single-user corpora from the TREC spam filtering benchmark evaluations [9, 7, 8]). Furthermore, we use relatively small, fixed training sets rather than the online filtering scenario to mimic the setting of training on a limited number of messages, as in the typical personalization setting.

For `ceas08-1`, the training set was composed of the `train` messages, and the batch train/test filtering task is equivalent to the `pretrain-nofeedback` task in the CEAS 2008 public corpus data set. For `trec07p`, the training set is composed of the first 10,000 messages in the ordered corpus, and the batch train/test filtering task is nearly equivalent to the `delay` task from TREC 2007, with the exception that in the TREC task filters were trained and evaluated incrementally for these first 10,000 messages in addition to being evaluated without additional training on the remaining test set. The training set we use for `trec05p-1` is composed of the first 10,000 messages in the ordered corpus, for repeatability.

We used separate tuning data for all parameter tuning and initial experiments, drawn from the `spamassassin` corpus.[3] To simulate the effect of class label noise, we injected synthetic class label noise by flipping the class label of each message in the training set with probability $p = 0.1$. (The specific train/test split for this data set and the training set with synthetic class noise are available on request.)

5.2 Gathering Human Labels

Because we wished to assess the performance of mini-filters trained using not only gold-standard label feedback, but also noisy feedback from non-expert human users, it was necessary for us to collect noisy human labels for each training set. For all experiments involving human labels, the (possibly noisy) human labels were used only for training; gold-standard labels were used for test evaluation.

For the `trec05p-1` data set, following the methodology of [26], we sampled one human label per training message from the labels provided to the SpamOrHam.org project[4] organized by John Graham-Cummings. The rate of human error in these labels and for the other hand-labeled training sets are shown in Table 1, where error is with respect to the gold-standard labels provided with the data sets.

[2]The `ceas08-1` public corpus from the CEAS 2008 spam filtering challenge is available at `http://plg.uwaterloo.ca/~gvcormac/ceascorpus`

[3]Available at: `http://www.spamassassin.org`

[4]Formerly available at `spamorham.org`, derived labels available on request.

For both ceas08-1 and trec07p, it was necessary to acquire human labels as none were previously available for these data sets. For this data collection, we used human raters working on Amazon's Mechanical Turk.[5] The email messages were converted to HTML format using the MHonArc-2.6.16 toolkit,[6] with mailto links disabled and non-image attachments removed for security.

For a given message, raters were asked to provide a rating of "Spam" or "Not Spam". If the message was marked as "Spam", the raters were further asked to classify the message into one of ten possible sub-categories (including **business proposal**, **adult content**, and **miracle cures**). The sub-categorization was intended both to assess inter-annotator agreement and to make it difficult for any rater to cheat by applying an existing spam filter to the messages. The rate of pay was $0.01 US per rating (plus $0.005 US commission); ratings were on average completed in 16 seconds per message.

For the 3,067 examples in ceas08-1, we asked for three independent ratings per message, to allow measurement of inter-annotator agreement, and required all raters working on these messages to have a prior approval rating of 95%.[7] These ratings were completed in roughly two hours. To construct the final training set, we sampled one label at random from each of the given labels for each message, for consistency with the methodology of the trec05p-1 labels.

For the 10,000 examples in the trec07p data set, we asked for only a single rating per message, but required raters to have a prior approval rating of 98%. These ratings were completed in roughly four hours. Note that there was a significant gain in accuracy of the human ratings (with respect to gold-standard ratings) through the application of a more stringent prior approval rating for raters, at no additional cost per rating, although some of this difference may be due different degrees of difficulty of message evaluation in the two corpora.

The human labels (including the results of the sub-categorization) for both trec07p and ceas08-1 are freely available for future researchers on request.

6. EXPERIMENTAL RESULTS

In this section, we report results from experiments comparing the mini-filter algorithms from Sections 3 and 4 on the data sets described in Section 5. We also compare with baseline methods using all features.

6.1 Methodology

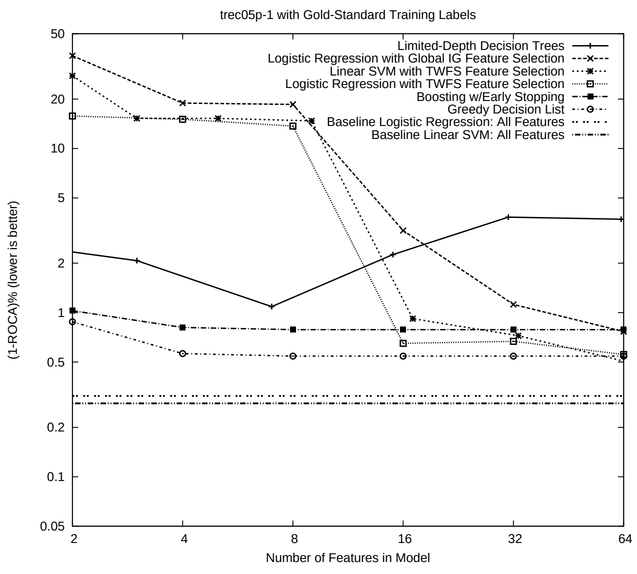
Three scenarios are considered: the case where gold-standard training feedback is given to filters for training, the case where noisy labels from human are given to the filters for training, and the case where filters are given noisy training labels for training plus the output of a generic global filter as a feature to consider for training and classification. In all scenarios, the gold-standard labels are used for test evaluation.

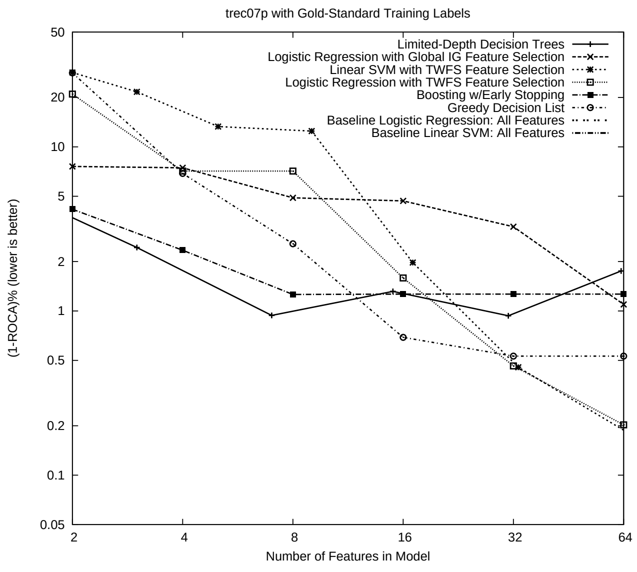
The evaluation measure we use is the $(1 - ROCA)\%$ measure (area above the ROC curve, expressed as a percent),

[5]http://www.mturk.com

[6]http://www.mhonarc.org

[7]Prior approval rating is a metric provided by the Mechanical Turk system, noting the percentage of previous tasks that the individual rater completed to the satisfaction of the task requesters.





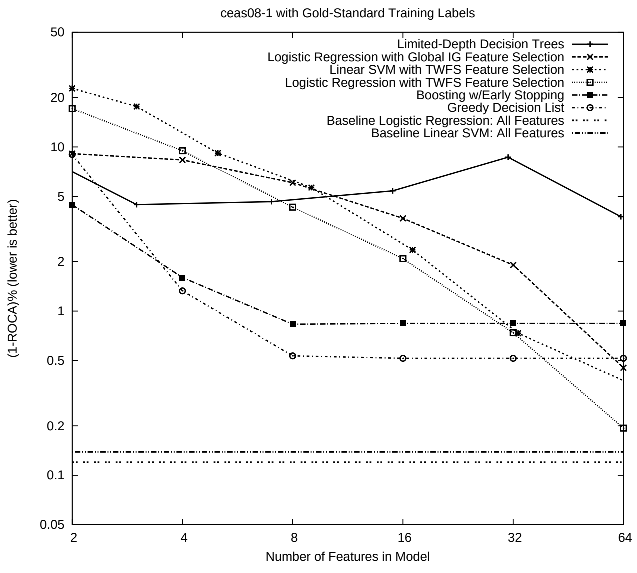


Figure 3: Results for Mini-Filters and Benchmark Filters using Gold Standard Feedback

standard from the TREC benchmark evaluations for email spam filtering [9]. This measure may be interpreted as the percent chance that a randomly selected *ham* message will be incorrectly scored as "more spammy" than a randomly selected `spam` message. A completely random classifier is expected to give a $(1 - ROCA)\%$ score of 50.0, while a perfect set of classifications would yield a score of 0.

As described in Section 2.2, we use a feature space of binary 4-mers drawn from the first 2500 characters of each message for all tests. Preliminary experiments using bag-of-words features showed qualitatively similar results.

For both short decision trees and boosting with early stopping, we used the `fest` package by Nikos Karampatziakis.[8]

We used our own implementation of the decision list algorithm given in Section 4.1, with the c parameter set to 20 by tuning on the `spamassassin` tuning data.

We tested both linear SVM and regularized Logistic Regression with both the global information gain feature selection, and the TWFS method given in Section 4.2. For linear SVM, we used a setting of the $C = 0.05$ found by tuning on `spamassassin` data; we tested both the `SVM-light` and `libsvm` packages with linear kernel, and found no significant differences between them in terms of classification performance. Results reported here are for the `SVM-light` package [16]. The Logistic Regression package we used for these experiments was `lr_trirls` [18], with default parameters. We also tested a Naive Bayes classifier in conjunction with both global information gain and TWFS, but found that the results were not competitive and do not report the results here.

6.2 Baseline Methods

We include two baseline methods to determine how well a strong classifier would perform on these data sets, trained in batch mode using all features. These baseline methods were SVM using all features and Logistic Regression using all features, both with L2-norm regularization. The regularization parameter C was found to give best results with a value of 0.05 for both methods in tuning trials. We used the `SVM-light` linear SVM, and the `liblinear` L2-norm regularized implementation of Logistic Regression [12] for these trials.

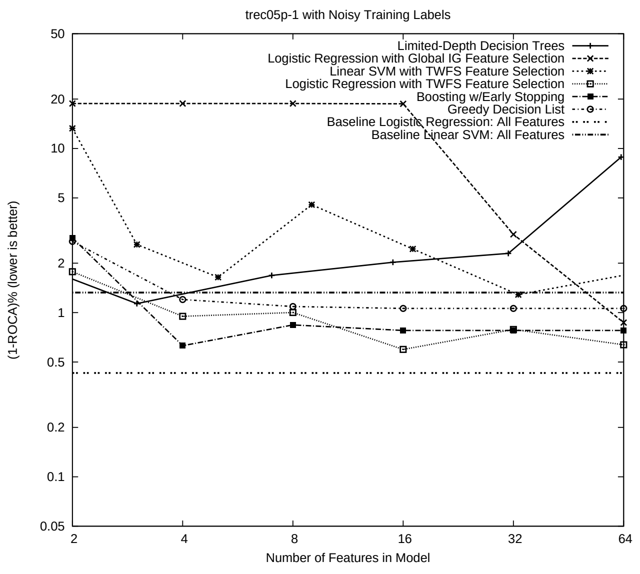
A Naive Bayes classifier was also tested on all features as an additional baseline. This classifier gave results roughly an order of magnitude worse than the Logistic Regression baseline, and is not included in the reported results.

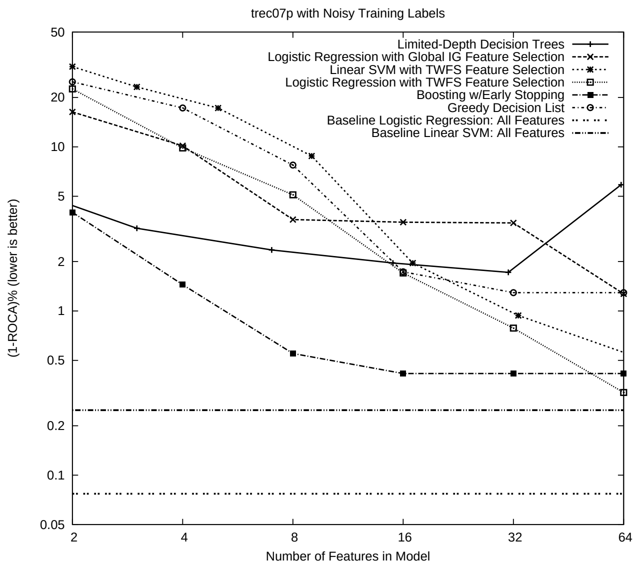
6.3 Gold-standard Label Feedback

The first scenario tested used gold-standard labels for training and testing. Results for these experiments are given in Figure 3; note the log-log scale for all graph results in this paper. (The baseline methods of Logistic Regression with all features and linear SVM with all features both gave performance levels of 0.01 on `trec07p`, which is below the range displayed on the graph.)

There are several observations to make. First, the overall performance of the mini-filters is surprisingly strong considering that they are using so few features. As a comparison point, for `trec07p`, the best performing mini-filters (TWFS using 64 features) gave what would have been median performance at TREC 2007 on the nearly identical `delay` learning task, out of 36 entrants.

[8] `http://www.cs.cornell.edu/~nk/fest/`





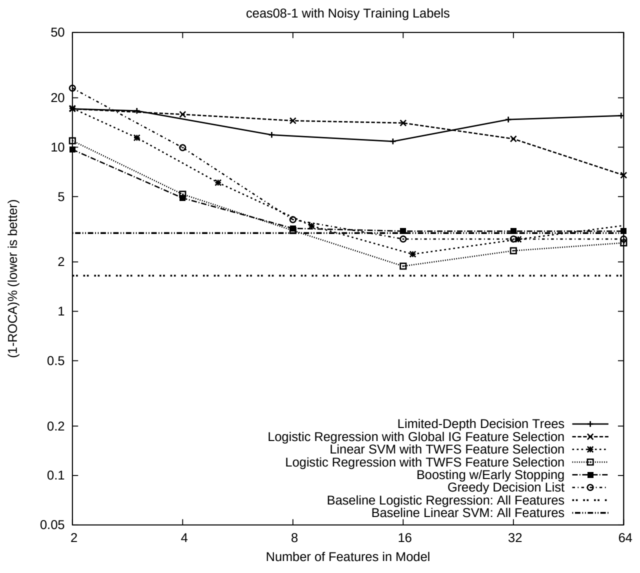


Figure 4: Results for Mini-Filters and Benchmark Filters using Noisy (Human) Training Labels

Both decision lists and boosting with early stopping achieve strong levels of performance with very few features, getting most of the benefit with only eight features in most cases and then reaching a performance plateau. Examining the models learned, it appears that both methods stopped finding useful features to add after ten or twenty iterations. In the case of the decision lists, this was because all of the examples had already been covered, and in the case of boosting the algorithm continued to add the same feature *ad infinitum* with near zero weight. (We tested up to 1000 iterations to confirm this pattern.) The greedy forward-selection methods used by both algorithms appears to be limited, paradoxically, by achieving too much success in early rounds. Bagging may be a way to gain additional benefit at $t = 32$ and $t = 64$; we will investigate this possibility in future work.

In contrast, the TWFS method gives results that are more than an order of magnitude poorer in early rounds, but continues to benefit from the addition of features through $t = 64$, and surpasses the greedy methods at these later iterations on both `trec07p` and `ceas08-1`. Global information gain lagged behind TWFS somewhat, but still gave surprisingly strong performance at $t = 64$.

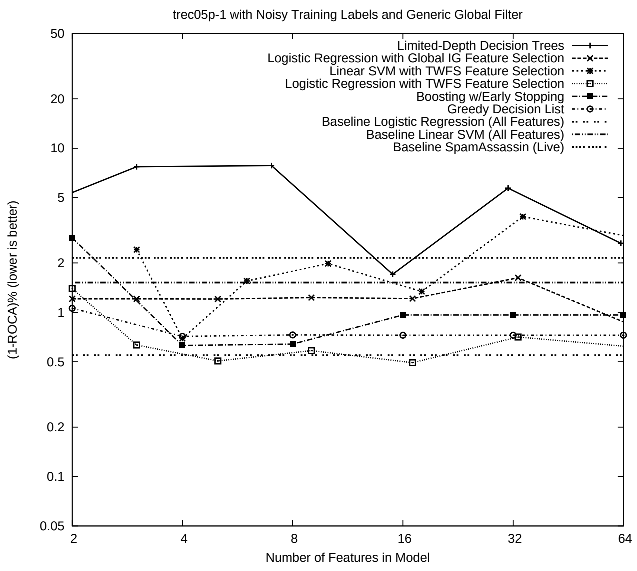
Another surprise to us was the poor performance of the limited depth decision trees, whose performance was terrible on `trec05p-1` and `ceas08-1` (although strong on `trec07p` with 8 or fewer features), often getting worse with added features. We ascribe this poor performance to overfitting of the data enabled by the expressivity of the hypothesis class, even at these limited tree depths.

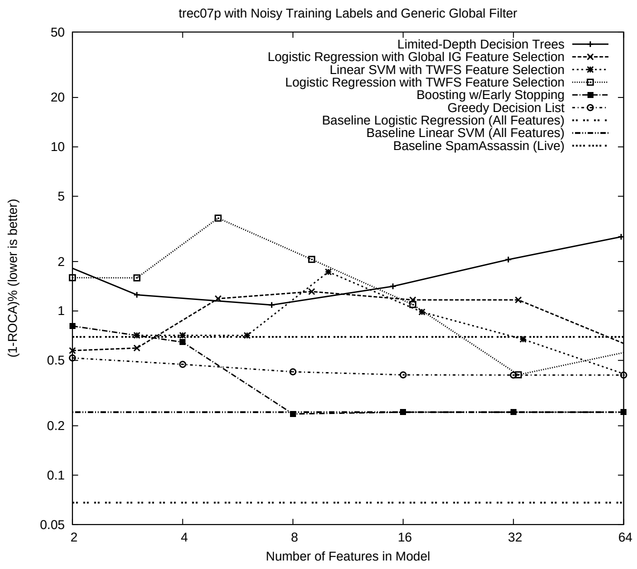
6.4 Human Label Feedback

In our second set of experiments, we used the noisy human labels for training, and evaluated on the gold-standard labels for testing. These results are shown in Figure 4.

The striking result of these experiments is that, on these more realistic training labels, the mini-filters appear to be much more noise-tolerant than the baseline methods of linear SVM and regularized Logistic Regression using all features, despite the heavy regularization employed by both of these methods with $C = 0.05$. On both `trec05p-1` and `ceas08-1`, the decision list, boosting with early stopping, and TWFS mini-filters actually exceed the performance of the linear SVM using all features, and are not far off the slightly better performance of regularized Logistic Regression. Furthermore, the performance of boosting with early stopping was actually improved by the noisy labels in `trec07p`; we conjecture that this improvement was due to the noisy labels acting as a deterrent to overly-aggressive feature weighting in the early boosting rounds seen with the gold-standard labels. In general, the noise-tolerance of the mini-filters may be explained in terms of their lower VC dimension [22]. Because the models are less expressive, they are less vulnerable to over-fitting of noise.

Note also that global information gain performs relatively more poorly than the other mini-filter methods in the presence of noisy labels, and the short decision trees continue to show poor performance.





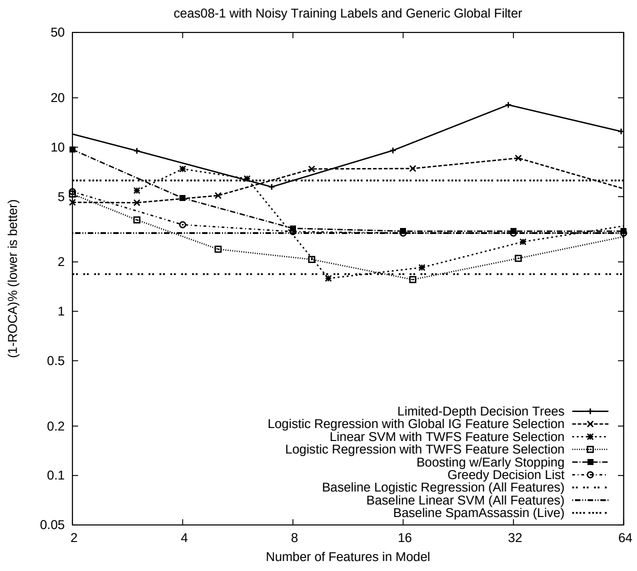


Figure 5: Results for Mini-Filters and Benchmark Filters using Noisy (Human) Training Labels, and Output of Generic Global Filter

6.5 Mini-Filters Augmenting a Generic Global Filter

Our final set of experiments was aimed at investigating one natural application of mini-filters, in which a personalized mini-filter is used to augment the accuracy of a generic, global filter.

In these experiments, the live `SpamAssassin` email filter running on the University of Waterloo email system was used as the generic global filter. This system-level filter was not trained on the training data in these tasks, and was not otherwise specially adapted to these data sets. The performance of this global filter without a mini-filter is given as an added baseline in the results reported in Figure 5.

To combine the generic global filter with the mini-filters, we tested two methods: using the output of the generic filter as an additional real-valued feature in the feature vector, and fusing the output of the mini-filter with the output of the global filter. All methods except the decision lists (which ran only binary features) gave better performance using the global output as a feature. The decision list gave better performance when its output was fused with the global filter's output, using $p = d + cg$, where p is the final score for a particular example, d is the score from the decision list, g is the score from the global filter, and c is a constant (for these experiments, set to 0.1 by tuning with a linear threshold unit on the `trec05p-1` training data).

Looking at the results of these augmented filters (Figure 5), the first thing we notice is that combination of a good mini-filter and the generic global filter gives much improved results over either filter in isolation. Indeed, for `trec05p-1` and `ceas08-1`, the best of these combinations equal the performance of regularized Logistic Regression and surpass the linear SVM trained on all features.

This is an important result: lightweight personalization in conjunction with a non-personalized global filter is able to match the performance of the heavyweight personalization given by the baseline methods.

7. RELATED WORK

The area of lightweight personalization for spam filtering has received previous attention by Chang *et al.* [4], who used a partitioned logistic regression scheme to train global and personalized components together. The personalized components in this paper were essentially equivalent to giving a unique bias variable to each user; this approach reduced misclassification of grey mail significantly. Our approach differs by allowing more fine-grained personalization while maintaining lightweight cost. Furthermore, our approach is not limited to adjusting a single bias parameter (showing more or less tolerance to grey mail), but may conceivably used for more specific learning tasks such as personalized foldering and prioritization of emails.

Heavyweight solutions to personalized spam filtering have been investigated previously as well. Segal compared a system deploying a large number of personalized filters (using a large number of features) to a single global filtering system, and found that the personalized filters (using Naive Bayes) fared poorly in comparison to the global model.[27] In this work, it was also found that these heavyweight personalized filters were able to augment the performance of the global model. Our work follows along these lines, but uses lightweight mini-filters for personalization at low cost. Furthermore, in our experiments, the personalized mini-filters tended to out-perform the generic SpamAssassin filter used a global filter, but were not always competitive with heavyweight personalized filters using all features and regularized Logistic Regression.

Weinberger *et al.* took a different approach to personalization of email spam filtering, training a single model on both global features from all models, and user-specific features drawn from the cross product of user identification features and features from a given email.[30] To avoid a combinatorial explosion in the size of the feature set, they employ a feature hashing scheme for randomized dimensionality reduction. The use of mini-filters may be seen as a more modular (and perhaps more interpretable) approach to lightweight personalization.

Finally, the effectiveness of ensemble methods for spam filtering, such as in the SpamGuru system of Segal *et al.* [28], and the filter-fusion methods of Lynam and Cormack [20], also suggest that the use of mini-filters to create personalized ensembles from a fixed set of diverse filters may be a promising area for future work.

8. CONCLUSIONS

In this paper, we set out to determine if effective mini-filters could be trained for email spam filtering, using a drastically reduced feature set. The experimental results presented suggest that several methods, including boosting with early stopping, greedy decision lists, and TWFS methods all give effective, low-cost solutions to this problem. Furthermore, the initially appealing ideas of global information gain or decision trees of limited depth are shown to be less than ideal, especially under conditions of class label noise. We believe that the appropriate use of such mini-filters is in a scenario of mass personalization, including situations in which a generic global filter may be augmented with a lightweight personalized model.

Despite these encouraging preliminary results, there is significant future work to be done. Most of the successful algorithms presented in this paper operate most naturally in a batch train/test setting, rather than an online setting. Mini-filters that could be updated incrementally would be preferred. Furthermore, it is our hope that the low cost of effective mini-filters may open new avenues for application at scale, including automated personalized foldering and prioritization systems, or deployment in other resource-constrained settings such as hand-held devices.

9. REFERENCES

[1] J. Bi, K. Bennett, M. Embrechts, C. Breneman, and M. Song. Dimensionality reduction via sparse support vector machines. *J. Mach. Learn. Res.*, 3, 2003.

[2] A. Blum. On-line algorithms in machine learning. In *In Proceedings of the Workshop on On-Line Algorithms, Dagstuhl*, 1998.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[4] M. Chang, W. Yih, and R. McCann. Personalized spam filtering for gray mail. In *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS)*, 2008.

[5] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *EWSL-91: Porceedings*

of the European Working Session on Machine Learning, 1991.

[6] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3, 1989.

[7] G. V. Cormack. TREC 2006 spam track overview. In *TREC 2006: Proceedings of the The Sixteenth Text REtrieval Conference*, 2006.

[8] G. V. Cormack. TREC 2007 spam track overview. In *TREC 2007: Proceedings of the The Sixteenth Text REtrieval Conference*, 2007.

[9] G. V. Cormack and T. R. Lynam. TREC 2005 spam track overview. In *The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings*, 2005.

[10] L. F. Cranor. *Delcared-strategy voting: an instrument for group decision-making*. Ph.D. Thesis, Washington University, 1996.

[11] S. C. Deerwester, S. T. Dumais, T. K. Landauer, and G. W. F. a nd Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[12] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9, 2008.

[13] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *In Proceedings of the Thirteenth International Conference on Machine Learning*, 1996.

[14] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3, 2003.

[15] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Mach. Learn.*, 42(1-2), 2001.

[16] T. Joachims. Making large-scale SVM learning practical. *Advances in Kernel Methods - Support Vector Learning,* B. Scholkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.

[17] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2), 1997.

[18] P. Komarek and A. W. Moore. Making logistic regression a core data mining tool with tr-irls. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, 2005.

[19] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. In *NIPS 2008: Neural Information Processing Systems*, 2008.

[20] T. Lynam, G. Cormack, and D. Cheriton. On-line spam filter fusion. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Resea rch and development in information retrieval*, pages 123–130, 2006.

[21] A. McCallum, X. Wang, and A. Corrada-Emmanuel. Topic and role discovery in social networks with experiments on enron and academic email. *Journal of Artificial Intelligence Research*, 30, 2007.

[22] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[23] M. Palatucci and A. Carlson. On the chance accuracies of large collections of classifiers. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008.

[24] R. L. Rivest. Learning decision lists. In *Machine Learning*, 1987.

[25] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 2004.

[26] D. Sculley and G. V. Cormack. Filtering spam in the presence of noisy user feedback. In *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS)*, 2008.

[27] R. Segal. Combining global and personal anti-spam filtering. In *In Proceedings of the Fourth Conference on E-mail and Anti-Spam (CEAS)*, 2007.

[28] R. Segal, J. Crawford, J. Kephart, and B. Leiba. Spamguru: An enterprise anti-spam filtering system. In *In Proceedings of the First Conference on E-mail and Anti-Spam (CEAS)*, 2004.

[29] D. Skillicorn. *Understanding Complex Datasets: Data Mining with Matrix Decompositions*. Chapman and Hall, 2007.

[30] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola. Feature hashing for large scale multitask learning, 2009.

[31] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd ed.* Morgan Kaufman, 2005.

[32] D. Yarowsky. Hierarchical decision lists for word sense disambiguation. In *Computers and the Humanities*, 1999.

[33] T. Zhang and B. Yu. Boosting with early stopping: convergence and consistency. *Annals of Statistics*, 33, 2005.