# Enhanced Email Spam Filtering through Combining Similarity Graphs

Anirban Dasgupta Maxim Gurevich Kunal Punera

Yahoo! Research
701 First Ave.
Sunnyvale, CA 94089
{anirban, maximg, kpunera}@yahoo-inc.com

## ABSTRACT

Over the last decade *Email Spam* has evolved from being just an irritant to users to being truly dangerous. This has led web-mail providers and academic researchers to dedicate considerable resources towards tackling this problem [9, 21, 22, 24, 26]. However, we argue that some aspects of the spam filtering problem are not handled appropriately in existing work. Principal among these are adversarial spammer efforts – spammers routinely tune their spam emails to bypass spam-filters, and contaminate ground truth via fake HAM/SPAM votes – and the scale and sparsity of the problem, which essentially precludes learning with a very large set of parameters.

In this paper we propose an approach that learns to filter spam by striking a balance between generalizing HAM/SPAM votes across users and emails (to alleviate sparsity) and learning local models for each user (to limit effect of adversarial votes); votes are shared only amongst users and emails that are "similar" to one another. Moreover, we define user-user and email-email similarities using spam-resilient features that are extremely difficult for spammers to fake. We give a methodology that learns to combine multiple features into similarity values while directly optimizing the objective of better spam filtering. A useful side effect of this methodology is that the number of parameters that need to be estimated is very small: this helps us use off-the-shelf learning algorithms to achieve good accuracy while preventing over-training to the adversarial noise in the data. Finally, our approach gives a systematic way to incorporate existing spam-fighting technologies such as IP blacklists, keyword based classifiers, etc into one framework. Experiments on a real-world email dataset show that our approach leads to significant improvements compared to two state-of-the-art baselines.


*WSDM'11,* February 9–12, 2011, Hong Kong, China.


## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

## General Terms

Algorithms, Experimentation

## Keywords

Email spam, personalization, collaborative filtering, similarity graphs, indexing, top-k retrieval

## 1. INTRODUCTION

Over the last decade unsolicited bulk email, generally referred to as email-spam, has ballooned in volume, pretty much in lock-step with increasing Web usage among the general public. Many recent reports have put the fraction of all emails that can be considered spam as higher than 90% [8, 17]. The motivation for much of this spam activity is undoubtedly commercial; though sometimes legitimate, it often involves selling illegal products or services [15], spreading malware like viruses and spyware that result in information thefts [15], and propagation of scams and frauds [5, 11]. Email spam has evolved from being just an irritant to a danger to web users in the last decade, and hence, stopping spam messages from appearing in users' inbox has become an extremely critical task. Both free and paid mail providers have dedicated considerable resources towards it, and as a result less than 0.1% of attempted spam reaches users' inbox [15]. However, this still translates into tens to hundreds of spam messages being encountered by a email user every year. Moreover, the economics of spam marketing on the Web are such that "conversion rates" on the order of $10^{-7}$ are enough to sustain spammers [15], implying that there is a lot of room for improving spam filters.

In this paper we tackle the email spam filtering problem: determining whether an email sent to a user is legitimate (also called Ham) or Spam. We consider the problem from the perspective of an email service provider, in that we can observe email spam patterns across the system, and not just for an individual user. A lot of research effort has been devoted to this problem both in the industry and academia; however, as we argue next, there are many aspects of this problem that makes the problem hard to tackle from a standard machine learning point of view. Below, we list some of these aspects and discuss how they motivate our proposed approach.

Adversarial Content.

Email spam filtering can be naturally formulated as a binary classification task [22, 24] in which emails are represented using text and other, possibly email specific, features, and functions are learnt from these features to the ground truth - known spam and known legitimate emails. In published research, filtering accuracy in excess of 0.9 F1-score, has been achieved using highly discriminative features along with state-of-the-art learning algorithms like Data-Compression models [6], Logistic Regression [13], and Support Vector Machines [23] just to mention a few. These approaches, though, all rely on the availability of a representative training corpus with labeled examples of spam and legitimate emails. However, spammers are known to quickly adapt their email messages to circumvent these filters [12], thereby altering the distribution from which test data is drawn. Hence, these labeled training corpora become quickly outdated and are thus expensive to maintain. There has been some work on modeling spam detection formally as an adversarial classification game [10], but these results are still preliminary.

Sparse User Feedback.

Because of the privacy issues that are intrinsic to the email domain, it is not always appropriate to employ human editorial help in labeling emails as ham or spam. Barring this, the only way of obtaining a labeled dataset is to automatically construct one from user feedback. Most web-based email services provide a functionality by which the user can give feedback: users can vote Spam on spam emails that were delivered to their inbox, and vote Ham on legitimate emails that were erroneously put into their Bulk-mail folder. The email providers can then adjust their spam filtering approaches taking these votes into account. However, analysis of voting logs of a major web-based email provider shows that while the votes collected per month number in the millions, they are provided by a very small fraction of email users and on a very small fraction of emails received. This is largely a result of the fact that users tend to vote only when existing spam filters make an error, which happens infrequently. The sparsity of the voting data is made worse by the fact that new users join the system all the time and that almost all incoming emails are different from those previously received, and have no attached vote history. This sparsity problem can be somewhat alleviated by using attributes of users and emails to learn a general characterization of spam using voting feedback; this is an active area of research [9, 26]. In this work we adopt a similar approach while attempting to avoid the problems caused by adversarial votes.

Adversarial Ham/Spam Votes.

It has been observed [21] that as email providers have started using user feedback in terms of Ham/Spam votes to train their spam filters, spammers have in turn begun entering adversarial votes at a massive scale: spammers vote Ham on their own spam emails and Spam on other legitimate emails. The massive scale at which these votes are entered is made possible by large-scale fake account creation [18] as well as phishing activity [5, 11] by spammers. Training spam filters, which generalize using user/email attributes, without taking into account the quality of these votes can result in reduced performance even with strong learning algorithms; we empirically show this in our experiments. One way to handle this is to learn a spam filter separately for each user [26], though this runs into the voting sparsity issues mentioned above.

Our Solution: Personalization for Spam Filtering.

The solution we propose in this paper is to strike a balance between generalizing Ham/Spam votes across users and emails (to alleviate the impact of sparsity) and learning local models for each user/email (to limit the effect of adversarial votes). Our approach uses a vote by a user $u$ on an email $m$ to learn a filter for only those other users and emails that are "similar" to $u$ and $m$ respectively. For instance, if two users have engaged in bidirectional email exchanges with each other, then they are considered similar, and the votes of one user will affect the spam filtering of the other user in proportion to the frequency of their communication. While it might be relatively easy for a spammer to send an email to a user, it is much harder for the spammer to coax the user to reply to it. Hence, in our model a spammer will have great difficulty faking high similarity to a legitimate user, and will not be able to affect the spam filtering for the legitimate user. We call such features *spam-resilient*. Another example of a spam-resilient feature is defining similarity between two users as the Jaccard coefficient of their recipient list, since in most cases the spammer does not know which email IDs a legitimate user corresponds with it.

Analogously, we also define a parallel set of features for computing email-email similarities. For instance, two emails are considered similar if they share the IP address from which they were sent, and then the votes on one email can be transferred safely to the other, once again, in proportion to this degree of similarity. Other spam-resilient email-email similarity features are pointing to the same URLs, or having largely similar content.

Many of the spam-resilient features we devise are naturally represented as weights on the edges of the user-user or email-email graph. For example, the frequency of bidirectional exchanges between two users can be modeled as the weight on the edge between the nodes (users) in a graph. In this work, we give a machine learning formulation that learns to combine multiple spam-resilient features encoded as graphs of user-user and email-email similarities while directly optimizing the objective of better spam filtering. The number of parameters to be estimated is $k_u + k_e$, where $k_u$ ($k_e$) is just the number of user-user (email-email) features. This is much lower than the number of parameters that need to be estimated in case of a naive encoding of these features into per-user (per-email) vectors. In addition we can use our learning framework to systematically incorporate arbitrary knowledge from existing spam-fighting technologies such as IP blacklists, keyword based classifiers, etc. As we shall see, formulating the spam-filtering problem in this setting helps us use off-the-shelf learning algorithms to achieve good accuracy while preventing over-training to the adversarial noise in the data.

To the best of our knowledge, our technique is the first in parameterizing user-user and email-email similarities via spam-resilient features and in giving a learning approach to estimate the parameters. There is a lot of past work on using *collaborative filtering* approaches for personalizing recommendations in other domains [16], however, due to many of the aspects of the spam filtering problem mentioned above, these works are not expected to perform well. We give a detailed discussion of the differences between our approach and collaborative filtering in Section 2, and in Section 4 we give an empirical comparison of the two approaches.

Summary of Contributions.
1. We give a learning based framework within which we can use user-user and email-email similarities to generalize extremely sparse Ham/Spam votes.
2. We construct a set of spam-resilient features based on user-user interactions and email-email comparisons, and give a principled way to combine these features using only a small number of parameters.
3. Our approach gives a way to fold in knowledge encoded in existing spam-fighting techniques in a principled fashion.
4. We compare our approach with two state-of-the-art baselines on a real-world email dataset and our proposed approach leads to huge improvements in the operating region of the spam-filtering task.

Organization.
In Section 2 we review some of the past work related to the problem we are tackling in this paper. In Section 3 we first discuss the properties of a good solution and formulate them into an objective function. Then we give an algorithm to optimize this objective function and describe how this algorithm can be implemented in a real-world setting. In Section 4 we describe the empirical evaluation of our proposed approach against two state-of-the-art baselines.

# 2. RELATED WORK

Personalized spam filtering has been studied before in [7] and [26]. In [7], the authors propose personalized spam filtering as a solution to "graymail" (i.e. bulk-mail) classification and use a *partitioned logistic regression* model to first estimate a content based baseline probability of spamminess and then learn a user-specific modification to that for each email. Weinberger et al. [26] formalize the task of personalization as a multi-task learning problem and proposes the use of hashing to efficiently do personalization. These techniques do not deal with user-level similarities in doing personalization, but they serve as excellent motivations to the necessity of clever methods to implement personalization in the anti-spam domain – since the sparsity of feedback by average user defeats naive personalization.

In a closely related work, Chu and Park [19] tackle the problem of handling missing values by expressing both users and items as feature vectors $u_a$ and $m_b$ and then learn a weight for each pair $(a, b)$ of user and item features. Applying this technique naively in our setting would require us to blow up the feature dimension in order to represent the user-user (and item-item) interaction features. It is possible to augment their model by putting in constraints that ensure that number of parameters is the same as the number of sources of similarity, and the resulting modification would bring the two models closer. Pazzani [20] has a method for combining different types of models learnt from the item-item similarity, user-user similarity and user-item ratings – this approach learns independent models and then combines them. We believe that our approach better captures the interaction between the user-user and item-item similarities. Basilico and Hofman [4] presents a framework to learn joint kernel functions over user-item features – their combined kernel over user-item features is an outer product of user kernels and item kernels. Our work deals with a more structured definition of interactions between the kernels, and as a result our model requires a much smaller number of parameters to be learnt.

There have also been work in exploring more involved models for matrix factorization that incorporate the presence of features. Two common issues that plague both collaborative filtering systems, as well our case, are those of sparsity and cold start. Sparsity of data usually leads to overfitting in naive matrix-factorization models, since the resulting number of parameters is often of the same order as the underlying data points. In collaborative filtering research sparsity is usually handled by adding in regularization parameters – the simplest case being a $\ell_2$ regularization that assumes Gaussian prior. The work by Agarwal and Chen [2] develops a novel regularization technique for matrix factorization in which the prior is based on regression done on the user-item joint features. We deal with sparsity by constraining the parameters we actually use. Since we only learn parameters $\alpha_s$ and $\beta_s$, overfitting is less of an issue for us. The common way of dealing with cold-start problem is by having separate user features and item features. Our technique deals with the cold start problem by computing similarities of the new user to all existing users and then weighting the rating of these users on similar content.

Another set of techniques for combining user-user similarities with user specific features in classification are the graph regularization methods [1]. The basic idea here is to exploit the user-user similarities in developing a regularization term by positing that similar users should have similar predictions. This intrinsically places the similarities and the user(item)-specific features in different footings. As far as we know, there is no work that considers both user-user and item-item similarity notions for regularization.

# 3. OUR APPROACH

In this section we first motivate our formulation for the spam-filtering problem and then give an algorithm to solve it. We end the section with a concrete instantiation of the problem, and describe the details of implementing it in a real-world setting.

## 3.1 Desiderata for a Solution

Based on our discussion in the introduction of this paper, we would like our solution to the spam filtering problem to have the following comprehensive set of properties:
P1: The solution must avoid the use of editorially maintained labeled data. As we discussed earlier, this data exposes severe privacy issues, is easily outdated, and expensive to maintain. A sustainable spam filtering strategy must rely on labels that can be derived from the Ham/Spam votes provided by users.
P2: Many users and almost all emails that the spam filter is likely to encounter after training have never been seen before. The system should handle these cases appropriately.
P3: The same email is sometimes considered both spam and ham by different sets of users – marketing emails from companies are typical examples. The spam filter must try to learn these user specific preferences.
P4: The spam filter must be resilient against the spammer controlling votes by some (unknown set of) users.
P5: The voting dataset is extremely sparse and riddled with adversarial noise. The learning algorithm must avoid overtraining on the noise.
P6: There exists a lot of domain knowledge encoded in spam filters (based on IP blacklists, keyword based classifiers, etc) currently deployed by various web mail providers. Our solu-

tion must be able to incorporate arbitrary knowledge from these technologies.
P7: The technique should be amenable to scalable implementation, if necessary by using principled heuristics.

In the next section we present a formulation that attempts to satisfy all of these properties.

## 3.2 Problem Formulation

Let $\mathcal{M}$ denote the set of emails and $\mathcal{U}$ denotes the set of users. The aim of our spam filter is to be able to give a vote prediction for each incoming email $m$ addressed to a user $u$. In order to avoid editorially maintained training data, to satisfy property P1, the system is given access to a set $R$ of HAM/SPAM votes provided by users on emails. The goal then is to use the entries in $R$ to predict the votes for users and incoming email pairs that are not in $R$; this is similar to the task considered in standard *collaborative filtering*. The naive collaborative filtering approach does not have notions of user-user similarity (or email-email similarity) extrinsic to what it can derive from the voting matrix. An obvious problem with such an approach is that the emails that the system is asked to rate after training might be different (in content) from any that it has encountered before. The user for whom it is making the prediction might have had few, if any, votes in $R$ (e.g., be a new user). This in fact is a severe case of the what is known as the *cold start* problem in collaborative filtering – where the system tries to use existing models to predict votes for new users and new items. Given the ever-changing nature of spam and the constant influx of new users, dealing with cold-start is thus the chief issue to consider.

### 3.2.1 Collaborative Filtering with User/Email Features

In order to handle the cold-start issue (property P2), our spam filter constructs a number of features to represent a particular email, for example, content based tokens, information about sender's IP or user account, information about URLs or images contained in the email and so on. These features (and appropriate weights for them) are then used to share votes across "similar" emails. Similarly, the spam filter construct a number of features for each user based on their logged-in behavior, their preference for certain topics in search engine queries, and learn suitable measure of similarity between users. These features are then used to diffuse votes from one user's profile to another "similar" user's profile. A common task in deriving both user-user and email-email similarities is learning to weigh the evidence presented by each feature; to see why this task is critical consider the following two examples.

**Example.** If two emails $m_1$ and $m_2$ have been sent from the same user's email account, and $m_1$ has received a large number of SPAM votes, then it might be reasonable to assume that $m_2$ is spam as well. On the other hand, if the two emails were sent from different user accounts on the same domain, say `hotmail.com`, then we might not want to share the votes across the two emails. Hence, the feature shared-sender-account contributes more to similarity of emails than the feature shared-sender-domain for the purposes of sharing votes.

**Example.** Consider two users $u_1$ and $u_2$ who log in from the same computer repeatedly. If $u_1$ votes some email $m$ as SPAM, then the spam filter might assume with some degree of confidence that $u_2$ will also consider $m$ as spam. The system's confidence in sharing votes may not be very high if the property shared by the two users was broad demographic information like zip-code. Therefore, for users as well different features impact sharing of votes by different amounts.

Hence, our spam filtering system represents users and emails by features and for each feature learn a weight that signifies the confidence with which a vote can be shared among users or emails that are similar according to that feature. Note that this mechanism also leads to personalized spam filtering as is desired in property P3.

### 3.2.2 Resilience to Adversarial Votes

As we explained above building personalized spam filters using votes of only similar users leads to personalization to user preferences. A second critical reason to use personalization is to protect spam filters from adversarial HAM/SPAM votes cast by spammers. It has been observed that as the spammers have realized that web email providers are using user votes for spam filtering, they have started adding adversarial votes in large numbers [21]. The spammers automatically create and operate a large number of user accounts expressly for the purpose of voting HAM on their own spam emails and voting SPAM on legitimate emails [18]. Hence, it is critical that a spam filtering system be able to avoid using votes made by spammer accounts to filter spam for the legitimate accounts (property P4).

One way to accomplish this is to learn the weights for user features to ensure that spammers and legitimate users have low similarity to each other. Moreover, features used for similarity computations must be constructed so as to be *spam-resilient*. As mentioned before, by this we mean that the spammers should find it very difficult to "fake" similarity to a legitimate user in terms of these features. It is clear not all features are equally spam-resilient – some features are a lot easier for spammers to manipulate than others. On the other hand, manipulating each type of information requires a different investment on the part of the spammers. In our formulation, we thus look for defining user features from diverse sources, each with a different resilience to spam.

For instance, much of the information that a user provides at registration time is easily manipulated since there is no way for a email provider to check that the information entered by spammers is accurate. Even some information that web portals can automatically derive about users can be faked, e.g. interests in topics derived from search engine queries issued or web pages visited. On the other hand, our spam filter has a system-wide view of user activity, and it can construct features that are a lot harder for spammers to fake. As we discussed earlier, user features constructed from bidirectional user-user communication, overlaps in recipient lists, email features constructed from information on past behavior of the sending IP, are examples of features that are more difficult for spammers to fake. Our spam filter combines a number of such spam-resilient features in order to determine user-user and email-email similarities – described in details in Section 3.

### 3.2.3 Feasibility of Sharing Votes Across Similar Users

The voting data (HAM/SPAM votes) is already very sparse; extremely few users vote and emails that are voted on are typically mistakes made by existing spam filters, and hence are relatively rare. In such a scenario, if we restrict use of votes for personalization of spam filters of only similar users,

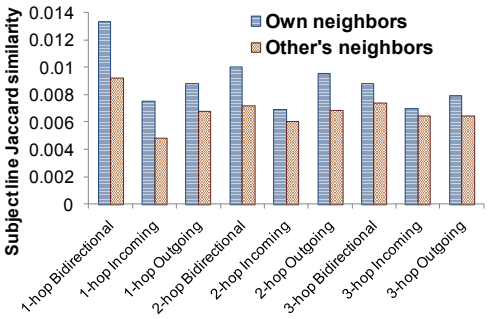


**Figure 1: Average similarity of email subjects voted on by a user to emails voted on by its neighbors, compared to neighbors of a random user.**

then we risk making the training data even sparser. In fact, in our work we construct spam-resilient user features solely in terms of email-communication between users. That is, two users who are "closer" in the communication graph have higher similarity than more far-away users. To be able to share votes between users under such definition of similarity the emails received by a user should be similar to emails voted on by other users in the communication-neighborhood. We validated that this indeed is the case by measuring average similarity between subjects of emails voted on by user $u$, and subjects of emails voted on by $u$'s neighbors at different distances in communication graphs along incoming, outgoing, and bidirectional links (see Section 4.2 for description of our dataset). Figure 1 shows the average Jaccard coefficient between subjects of email voted on by a user and of emails voted on by its neighbors, and compares it to the similarity to emails voted on by neighbors of a random user. The figure shows that there is positive correlation between neighbors' email subjects, and that some graphs induce higher correlation than the others. The results agree with our expectation that bidirectional links induce highest correlations since they are least prone to adversarial manipulation, while the incoming links are less reliable, as users have no control over their incoming email. Also, as expected, the correlation drops with distance. The results of this experiment show that using votes of only similar users for personalizing spam filters is feasible.

### 3.2.4 Learning Weights for the Similarity Measures

In our approach, we start with a set of similarity matrices $\{U^s\}$ satisfying $U^s \in \mathcal{U} \times \mathcal{U} \rightarrow \Re^+$, that denote similarities between any two users $(u, v)$. Similarly, we have a set of similarity matrices $\{M^t\}$, where each $M^t \in \mathcal{M} \times \mathcal{M} \rightarrow \Re^+$ denotes similarities between any two emails $(m, n)$. As mentioned above, the similarity values between two users $(u, v)$ could either come from comparing feature vectors for $u$ and $v$ – e.g. their behavior patterns when logged in into the webmail client – or it could come from interactions between $u$ and $v$, being present in each other's address-books, bidirectional email exchanges etc. Such user-user interactions cannot be represented as individual per-user features in a principled way without blowing up the feature space, or by resorting to heuristic and domain-specific feature construction from each such dataset. Another way of handling such interaction based similarities is by resorting to graph based regularization techniques [1], which are often computationally expensive. It is here that our method provides a principled, yet scalable, way of unifying the different notions of similarity into one that is suitable for the spam-filtering problem. We learn two vectors $\vec{\alpha}$ and $\vec{\beta}$ such that the resulting user-user similarity is given by

$$U = \sum_s \alpha_s U^s,$$

and email-email similarity is given by

$$M = \sum_t \beta_t M^t.$$

As we can see our formulation involves learning just $|\{U^s\}|+|\{M^t\}|$ number of parameters; one for each feature that contributes to user-user or email-email similarity. This is in contrast to many existing works that learn millions of parameter values [26] and reduces the risk that our spam filter will tune itself to the adversarial noise in voting logs (property P5).

### 3.2.5 Incorporating Existing Spam-fighting Technologies

Web mail services have been providing mail spam solutions for many years now and their technologies have a lot of domain knowledge encoded into them. Examples of these techniques are IP black and whitelists, rate based throttling methods, various rule and keyword based classifiers, etc. We want our solution to be able to incorporate arbitrary knowledge from these technologies in a principled fashion (property P6).

Within our formulation we can incorporate specific email filters by including "oracle" users who have votes over all emails. For each such "oracle" user $u_*$, we define a separate similarity matrix $U^*$ such that $U^*_{uu_*} = 1$ for all $u$ and zero everywhere else; "oracle" users are equally similar to all other users. The oracle user $u_*$ is also considered to be different from $u$ in every other similarity aspect: $U^s_{uu_*} = 0$ for $s \neq *$.

**Example.** An approach that is entirely orthogonal to *collaborative filtering* is to deal with this spam-filtering problem by treating it as a text-classification problem [22, 23, 26]. This approach constructs elaborate text features from each email and learns to classify these emails as spam using state-of-the-art learning methods like Support Vector Machines. Our approach can incorporate any such filter by including a "text filtering oracle" that has votes on every email. The amount of influence this "oracle" user has on vote predictions for other users is then learned using our standard weight estimation approach.

### 3.2.6 Kernel-based View of our Formulation

Another way of looking at our formulation is that we employ domain specific knowledge to construct similarity kernels between users and emails. In the kernel learning framework, we would first define a kernel $\mathcal{K}((u, m), (u', m'))$ over user-email pairs, and then the prediction for $(v, n)$ would be done as

$$\sum_{(u_i, m_i) \in S} w_i \mathcal{K}((u_i, m_i), (v, n)).$$

The weights $w_i$ would then be learnt using the training examples. The choice of the kernel here is often a matter of art. Our approach can in fact be seen as using a definition of general similarity instead of kernel [3], while constraining the similarity matrix to be a linear combination of similarities that are known to be significant from domain knowledge and then learning the best set of weights in order to combine these similarity matrices.

**Algorithm 1** Learning Algorithm

**Input:** The set of votes $R$, the set of user-user similarity matrices $\{U^s\}$ and the set of email-email similarity matrices $\{M^t\}$, error tolerance $\tau$.
**Output:** Coefficients $\alpha_s$ and $\beta_t$.
Initialize $\vec{\alpha}$ and $\vec{\beta}$.

**repeat**
Fix $\vec{\beta}$ and find $\vec{\alpha}$ that optimizes equation (3).
Do the same for $\vec{\beta}$.
**until** reduction in error over consecutive steps is $< \tau$.

**Algorithm 2** Prediction

**Input:** The set of votes $R$, the set of user-user similarity matrices $\{U^s\}$ and the set of email-email similarity matrices $\{M^t\}$, and coefficients $\alpha_s$ and $\beta_t$. Also, a new $(u, m)$ tuple.
**Output:** A prediction for $(u, m)$.
$U = \sum_s \alpha_s U^s$ and $M = \sum_t \beta_t M^t$.
$p = \frac{\sum_{(u,m)\in S} R_{um}\mathbf{C}(U_{vu}, M_{mn})}{\sum_{(u,m)\in S}\mathbf{C}(U_{vu}, M_{mn})}$. Return $p$ as the prediction.

## 3.3 Algorithm

We now give a formal description of our approach. Let $S$ denote the set of rated user-mail pairs that we see in the training period – $S_u$ is the set of users and $S_m$ is the set of emails in $S$. The label on the pair $(u, m)$ will be referred to as $R_{um}$ – it will be convenient to think of $R$ as a matrix of size $S_u \times S_m$ with missing entries. The vote given by $u$ on email $m$ could either be $-1$, indicating SPAM, or $+1$ which means that the message was marked as HAM by the user. Let us also define a general composition function $\mathbf{C}(\cdot, \cdot)$, which outputs the similarity between user-email pairs: we consider specific choices for $\mathbf{C}(\cdot, \cdot)$ later in this paper.

Our approach learns a set of weights $\vec{\alpha}$ and $\vec{\beta}$ such that $U = \sum_s \alpha_s U^s$ and $U = \sum_s \beta_s M^s$. Given these matrices $U$ and $M$, for any given user-mail pair $(v, n)$, our learner predicts the vote $P_{vn}$ to be

$$P_{vn} = \frac{\sum_{(u,m)\in S} R_{um}\mathbf{C}(U_{vu}, M_{mn})}{\sum_{(u,m)\in S}\mathbf{C}(U_{vu}, M_{mn})}. \quad (1)$$

Replacing the definitions of $U$ and $V$, we get

$$P_{vn} = \frac{\sum_{(u,m)\in S} R_{um}\mathbf{C}(\sum_s \alpha_s U^s_{vu}, \sum \beta_t M^t_{mn})}{\sum_{(u,m)\in S}\mathbf{C}(\sum_s \alpha_s U^s_{vu}, \sum \beta_t M^t_{nm})}.$$

In order to learn, we thus have to minimize the following function $F$ over all the training tuples $S$:

$$\min_{\vec{\alpha},\vec{\beta}} \sum_{(v,n)\in S} F_{vn}(\vec{\alpha}, \vec{\beta}) \text{ where} \quad (2)$$

$$F_{vn}(\vec{\alpha}, \vec{\beta}) = \left(R_{vn} - \frac{\sum_{(u,m)\in S} R_{um}\mathbf{C}(U_{vu}, M_{mn})}{\sum_{(u,m)\in S}\mathbf{C}(U_{vu}, M_{mn})}\right)^2$$

$$U = \sum_s \alpha_s U^s \text{ and } M = \sum_t \beta_t M^t.$$

The above formulation is not convex because of the normalization. However, without the normalization, the predictions are dominated by the $(u, m)$ pairs that have high similarity support. Thus, instead of converting this to a convex problem, we describe below an iterative way of solving this, that reduces the error at every step by solving a linear regression problem.

Since we are defining "similarities" between users and emails, it might also seem that we should constrain the $\alpha_s$ and the $\beta_t$ parameters to be non-negative. However, negative values of $\alpha_s$ and $\beta_t$ make sense when the features that are being used are correlated – furthermore, using such constraints would turn the above problem into a quadratic programming problem, which is hard to solve. Hence, in practice, we do not put in a non-negativity constraint on $\vec{\alpha}$ and $\vec{\beta}$, and instead do careful feature construction – we do not observe any negative values in our experiments.

The above formulation has been given in terms of an abstract composition function $\mathbf{C}(\cdot, \cdot)$. The two different instantiations of $\mathbf{C}(\cdot, \cdot)$ we consider naturally corresponds to sum and product formulations of combining the user-user and email-email similarities. The sum formulation captures the setting where two user-email pairs can be consider similar even if either the two users or the two emails are similar to each other. In the product formulation high similarity between user-email pairs requires that both the two users and the two emails be similar to each other. As we show later, while the sum function is easier to implement, the product function better captures the interaction between the user-user and email-email similarities, allowing reaching more accurate predictions.

We define the Sum-Similarity problem as above, replacing $F_{vn}$ in Equation 2 by

$$F^{\sigma}_{vn}(\vec{\alpha}, \vec{\beta}) = \left(R_{vn} - \frac{\sum_{(u,m)\in S} R_{um}(U_{vu} + M_{mn})}{\sum_{(u,m)\in S}(U_{vu} + M_{mn})}\right)^2,$$

and for the Product-Similarity replacing $F_{vn}$ in Equation 2 by

$$F^{\pi}_{vn}(\vec{\alpha}, \vec{\beta}) = \left(R_{vn} - \frac{\sum_{(u,m)\in S} R_{um}U_{vu}M_{mn}}{\sum_{(u,m)\in S} U_{vu}M_{mn}}\right)^2.$$

In both the cases, in order to optimize the functions, we employ the technique of alternating least squares over the vectors $\vec{\alpha}$ and $\vec{\beta}$. Holding $\vec{\beta}$ fixed, the problem above becomes equivalent to a least square problem, and we find $\vec{\alpha}$ to optimize the value of this. Similarly, we then hold $\vec{\alpha}$ to be fixed and determine the optimal $\vec{\beta}$ for this choice. For instance, to optimize for $F^{\pi}$, i.e. the Product-similarity problem, when we fix $\vec{\beta}$, the least square problem for finding $\vec{\alpha}$ becomes

$$\min_{\vec{\alpha}} \|R - X\vec{\alpha}\|_2^2$$

$$X_{(v,n),s} = \frac{\sum_{um\in S} R_{um}U^s_{vu}M_{mn}}{\sum_{um\in S} U_{vu}M_{mn}}. \quad (3)$$

Similarly, for the sum-similarity:

$$\min_{\vec{\alpha}} \|R' - X\vec{\alpha}\|_2^2$$

$$R'_{vn} = R_{vn} - \sum_{um\in S} R_{um}M_{mn}$$

$$X_{(v,n),s} = \frac{\sum_{um\in S} R_{um}U^s_{vu}}{\sum_{um\in S}(U_{vu} + M_{mn})}. \quad (4)$$

Also note that, normally, we would have to add in two regularization terms $\|\vec{\alpha}\|_2$ and $\|\vec{\beta}\|_2^2$ to prevent overfitting. Our problem is also in the sparse domain. Yet, because the number the parameters we learn – the $\alpha_s$ and $\beta_s$ are much smaller compared to the number of training examples, we need not worry about overfitting in our case.

## 3.4 Scalable Implementation

Recall that the dimensions of similarity matrices are determined as the number of users and emails served by the system. In a real system handling millions of users and billions of email messages, maintaining such a large matrices is prohibitively expensive. One approach is to maintain a limited set of users and emails so that similarity computation can be implemented efficiently at run-time. However, since email communication graph evolves rapidly, this set will have to be updated frequently. We thus propose a different approach, based on widely deployed free text search infrastructure.

Observe, that the vote prediction process for a user-email pair $(v, n)$, shown in Equation 1, computes a weighted average of all votes in the dataset, where the weight of each vote $R_{um}$ is the combination of similarities between the $v$ and $u$ and between $n$ and $m$. For the vast majority of past votes both of these similarities, and consequently contributions to the prediction, are low. Thus it is sufficient to retrieve only the votes most similar to $(v, n)$. Observe that this resembles an information retrieval problem in a vector space model, where the relevance of a document $d$ to a query $q$ is measured by the following scoring function:

$$Score(d, q) \;=\; \sum_t d_t \cdot q_t,$$

where $d_t$ (resp. $q_t$) denotes the weight of term $t$ in document $d$ (resp. query $q$). Note that if we replace documents by votes and queries by user-email features, and use the sum composition function, we can define the relevance score of a vote $R_{um}$ to a user-email pair $(v, n)$ as follows:

$$Score(R_{um}, (v, n)) \;=\; U_{uv} + M_{mn} = \sum_s \alpha_s U^s_{vu} + \sum_s \beta_s M^s_{nm}.$$

We then index all votes in an inverted index, and employ existing early termination techniques for efficient retrieval of the top-$k$ most relevant votes to a given user-email pair, i.e., $k$ votes that have highest $Score(\cdot, (v, n))$. Thus, for the update step in the sum-similarity problem, instead of solving for Equation 4, we solve instead the following program. Define $S^k_{vn}$ to be the top-k set of $(u, m)$ pairs with the highest similarities with $(v, n)$ under the current $\vec{\alpha}$, $\vec{\beta}$ parameters. Then, we update $\vec{\alpha}$ by solving

$$\begin{aligned}
&\min_{\vec{\alpha}} \|R' - X\vec{\alpha}\|_2^2 \\
&R'_{vn} = R_{vn} - \frac{\sum_{um \in S^k_{vn}} R_{um} M_{mn}}{\sum_{um \in S^k_{vn}} U_{vu} M_{mn}} \\
&X_{(v,n),s} = \frac{\sum_{um \in S^k_{vn}} R_{um} U^s_{vu}}{\sum_{um \in S^k_{vn}} U_{vu} M_{mn}}.
\end{aligned} \tag{5}$$

A similar program needs to be solved for updating $\vec{\beta}$. Note that we use the same top-k technique for our final prediction on the test instances also. We currently have no guarantees that we can derive on the convergence of our heuristic. Our experiments seem to suggest that this top-k indexing based descent converges fairly quickly.

Note that this implementation is efficient only when similarity matrices are sufficiently sparse. This is not a substantial problem in our setting as most user-user and email-email similarity features are naturally sparse. For example, a user has positive similarity to a relatively small set of neighbors in the communication graph; an email is similar to a small number of other email sent by the same sender. All such features can be efficiently processed by inverted index retrieval algorithms.

Handling other similarity composition functions other than sum in our top-k indexing based method is trickier, as they cannot be represented in a vector space model. Although it may be possible to develop specialized retrieval algorithms for such functions, we use a simple heuristic solution: we first retrieve the top-$k$ votes according to the sum composition function, and then compute scores of these $k$ results using the desired function. In this case $k$ has to be large enough so that the votes with highest sores under the desired composition function are among top-$k$ votes under the sum function.

The above technique allows us to dramatically reduce the overhead of computing predicted votes. Instead of considering millions or even billions of historical votes for each prediction, only $k$ votes are considered. In our experimental evaluation (Section 4.4) we show that as low value as $k = 100$ can be used with an index of 1.6 million votes.

# 4. EXPERIMENTS

In this section we evaluate the spam filtering performance of our approach on a real-world email dataset. The data was suitably anonymized before the experiments with all personally identifiable information replaced by unique tokens. We start by describing the implementation details of our approach. Then, we detail the experimental setup: the datasets used, the baselines compared against, and the evaluation measures used in these comparisons. Finally we present and discuss the spam filtering performance of our proposed approach and the baselines.

## 4.1 Our Approach

User and Mail Features.

Recall that our approach takes as input user-user and email-email similarity matrices, one for each feature, and then learns weights to combine these evidences so as to better predict spam votes. If some similarity features are noisy, our learning framework will then assign them low or zero weight without effecting spam-filtering accuracy. This allows experimenting with large number of potential similarity features. Here, we describe the specific features we used in our experiments.

We identified the email communication graph as the most informative and adversary-resilient source of information for constructing user-user similarity features. From the historical log of email communication we constructed a directed graph where users are represented by nodes and emails by edges. It is nontrivial to summarize information contained in a graph into a small number of pairwise features. We used domain knowledge to condense user neighborhood information into 9 similarity features: combinations of three different hop distances – 1, 2, and 3 – and edges of different types – incoming, outgoing, and bidirectional. For example, for a feature outgoing-2-hop, user $u$ has positive similarity to all its 2-hop neighbors along outgoing edges in the communication graph, i.e., to all users whom he sent emails, and to whom these users sent emails.

For each email voted on, we collected the following attributes: (1) sender's email address, (2) IP address from which the email was sent, (3) subject line, (4) URLs extracted from email body, (5) voter's userid. From these summaries we generated 7 email-email similarity features.

LUCENE-BASED IMPLEMENTATION.

For retrieving votes we use the inverted index implementation described in Section 3.4. We index the summaries of voted emails in the training set using Apache Lucene search engine [1]. Lucene allows to efficiently retrieve, given user and email features, the top-$k$ emails under the sum similarity composition function. That is, for a user-email pair $(u, m)$, a Lucene query contains all the features of $m$ and all the users that $u$ has positive similarity to. The weights of query terms are set according to feature weights $\alpha_s$ and $\beta_s$.

As Lucene does not support retrieval under the product composition function, we implement the product composition function by first retrieving top-$k$ results using the sum composition function, and then computing scores of these results using the product function. The effect of $k$ on prediction quality is evaluated in Section 4.4. These experiments show that $k = 100$, which we used in our experiments, is sufficient for obtaining high prediction accuracy.

For learning $\alpha_s$ and $\beta_s$ coefficients we used a higher value of $k = 4000$ to better approximate learning with complete similarity matrices. The weights were learned using a Support Vector Machine formulation to directly optimize the Area under the ROC curve of the predicted votes (see Joachims [14]).

## 4.2 Experimental Setup

DATASET.

The data we used was extracted from logs of voting activities of a random set of active users. The emails voted on are used to construct the features that are used by our approach as well as the baselines. Note that the voting occurs on emails that are delivered to user inboxes and spam folders after applying existing spam filter. Thus, typically our dataset consist emails that were incorrectly classified by that filter. The base dataset is composed of votes, collected over the period of 10 days, of 253K users – 10K random active seed users and their 1, 2, and 3 hop neighbors in the communication graph. The communication graph was constructed from emails sent during the preceding month. The users were selected this way to have a realistically dense communication graph while avoiding undue biases. The training set was constructed from all the votes in the dataset during the first 8 days, and contained 1.6M votes. The test set contained votes of the seed users during the last two days. Since the user-user similarity features we used rely on the historical communication graph among the users, we removed isolated users from our test set. The resulting test set contained 6K votes.

EVALUATION MEASURES.

We evaluate our approach, and compare it to the baseline methods, using two standard metrics of classification accuracy: ROC curve and PRECISION/RECALL. The ROC – Receiver Operating Characteristic – curve plots how the amount of false positives and true positives change as the classification threshold is set to different values. Since our application is spam filtering, we consider spam as the positive class. Hence, true positives rate counts the fraction of all SPAM-voted emails found by our approach at a certain threshold, while the false positives rate gives the fraction of HAM-voted emails found. The Area under the ROC curve (AUC) is often used to quantify the overall quality of a classifier; AUC is between 0 and 1 with 0.5 indicating random performance.

[1] lucene.apache.org

An alternative method we use to report our results is in terms of PRECISION and RECALL values. Once again, we consider spam as the positive class. Hence, at a certain threshold, PRECISION captures the fraction of emails classified as spam by the filter that are truly spam, and RECALL captures fraction all true spam emails that are found by the filter. Typically, there is tension between PRECISION and RECALL and in the spam filtering application high PRECISION is critical since we cannot tolerate high false positives (HAM-voted emails classified as Spam). Hence, we report performance in terms of RECALL after fixing PRECISION at high operating points.

COMPARED APPROACHES AND BASELINES.

In this section we compare the performance our proposed approach with state-of-the-art baseline approaches from existing work.

OURAPPROACH represents the similarity-based approach that was proposed in this work. The algorithm is described in Section 3 and implementation in Section 4.1. We used the product formulation of our objective function for OURAPPROACH.

SVM represents the past works that model the spam filtering problem as a binary classification problem [23, 26]. For this baseline we use the Support Vector Machine formulation (implemented in SVMPref[2]) for directly optimizing the AUC of the predicted votes [14]. Each email is represented using various features like 1) the text tokens from the email's subject, 2) URLs within the text, 3) IP and userid from of email author, etc. We did not use tokens from the full-text content of the emails because it is often unavailable at run-time and can often have significant implications for user privacy. These features are identical to the email-email similarity features used by OURAPPROACH. The training error penalty parameter of the SVMPref classifier was tuned using a validation set and eventually set to 10.

OURAPPROACH+SVM is an instantiation of OURAPPROACH with the predictions of a text-based classification algorithm SVM added as an "oracle" user as described in Section 3.2.5.

CF is a collaborative filtering approach that does not use any notion of similarities between users and between emails. Since this baseline has no notion of user-user and email-email connections, it does not let votes on one email effect the spam classification of another. Hence, in order to get reasonable results we coarsened the user-email data to user-sender_autonomous system (asn) level. We selected sender_asn as the appropriate level for coarsening since it gave better results that sender_IP and near-duplicate email_subjects. The matrix factorization was performed by the algorithm described in [25].

## 4.3 Spam Filtering Performance

Figure 2 depicts the ROC plots corresponding to the above approaches. The corresponding AUC values are presented in the legend of the plot as well. As we can see the OURAPPROACH outperforms both SVM and CF in terms of spam vote predictions. SVM has slightly better performance at the very low false positive rate region, but as we can see combining SVM with OURAPPROACH into the hybrid OURAPPROACH+SVM we create a system that outperforms its components in all regions of the plot. This demonstrates the

[2] www.cs.cornell.edu/People/tj/svm_light/svm_perf.html

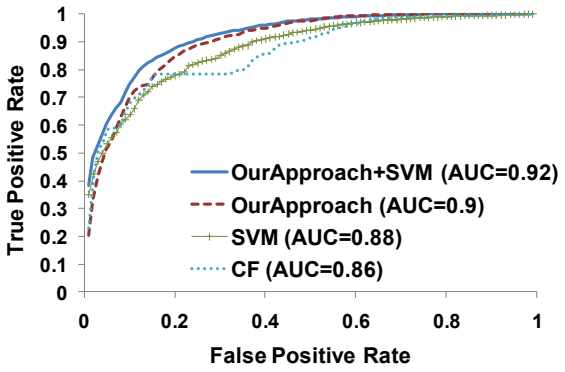


**Figure 2: The ROC curves for the performance of our technique and of the baselines in detecting spam emails.**

| | Recall at Precision of | | |
|---|---|---|---|
| | 99% | 95% | 90% |
| OurApproach+SVM | 0.437 | 0.832 | 0.948 |
| OurApproach | 0.153 | 0.742 | 0.939 |
| SVM | 0.361 | 0.695 | 0.873 |
| CF | 0 | 0.728 | 0.781 |

**Table 1: Recall of different techniques at Precision of 99%, 95%, and 90%.**

overall effectiveness of the approach proposed in this paper that can not only use new user-user and email-email similarity features but can also incorporate predictions of existing spam classifiers in a principled fashion.

In Table 1 we present numbers that give an indication of the system performance at the operating regions of spam filters. As we can see when we want extremely low to moderate number of false positives the hybrid OurApproach+SVM performs around 15-20%- better than SVM. Moreover, at the expense of only 5% false positives, the OurApproach+SVM system captures more than 80% of all truly spam emails. Recall that our system is tested on emails that have been erroneously filtered via existing spam filters and that users have then voted on. In this setting trading off 80% detection for 5% false positives is very competitive. The performance of CF is clearly inferior, as it is unable to even reach 99% precision.

Using Confidence of Predictions.

Recall that our algorithm outputs, in addition to the predicted vote, the confidence in the prediction, and this allows us to further increase accuracy by setting a confidence threshold below which the prediction is set to "do not know". Figures 3 and 4 show the accuracy of predictions made by OurApproach when we only consider the subset of cases where the prediction was made with high confidence. As we can see in Figure 3 the more confident OurApproach is the lower is the error (on the top 10% most confident predictions the AUC is close to 0.99).

Predicted votes have low confidence when the voted email has unique set of features, that do not appear in emails in the training set. This can happen when (1) the email is genuinely new and/or has never been voted on before, or (2) a similar email was voted on before but was not included in the training set. Although our training set contains a relative large set of 1.6M votes, it is still orders of magnitude smaller than the set of votes available at major email providers. We thus speculate that our High Confidence results reflect the accuracy that can be achieved with larger training sets. Moreover,

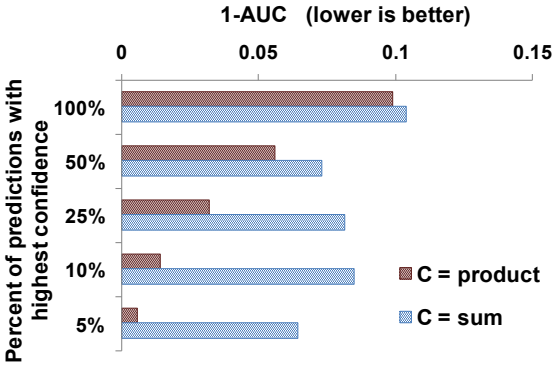


**Figure 3: Prediction error (1-AUC) with sum and product similarity composition functions on subset of prediction with highest confidence.**

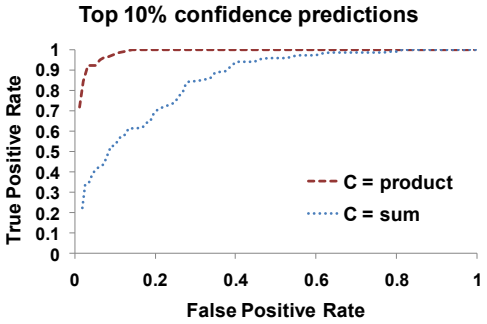


**Figure 4: The ROC curves with sum and product similarity composition functions on the subset of 10% of predictions with highest confidence.**

since a system such as this is likely to be deployed only in low false positive settings, these values can be regarded as the performance of our approach at the operating point.

The main conclusions from these results are the following:
1. Our proposed approach (OurApproach) outperforms baselines at the operating region of spam filtering task, and can be combined with existing, non-personalized, system to further improve classification accuracy (OurApproach+SVM).
2. Similarity based measures (OurApproach and OurApproach+SVM) outperform those that only rely on the voting records (CF) and those that simply use the email characteristics for classification (SVM).
3. The prediction confidence estimates output by OurApproach are high correlated with prediction accuracy and can be used to tune the operating point of our proposed system.

## 4.4 Algorithm Parameters

We next evaluate the effect of the similarity composition function on the prediction accuracy. Figure 3 shows the AUC values of our approach when using the product and the sum composition functions, on subsets of predictions with highest confidence, as a function of subset size. The figure shows that while the basic variants achieve similar accuracy, the product composition function achieves significantly higher accuracy as prediction confidence grows. This is easy to explain since the confidence computed by the product function is high only when *both* email-email and user-user similarities are high, while for the sum function it is enough that only one of these similarities is high. Therefore the confidence computed with the sum function is not as reliable as that with the product function. We expect the gap between the

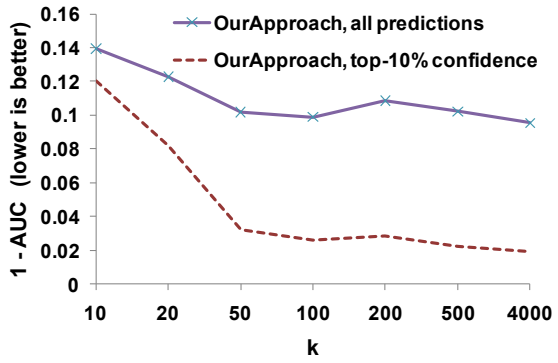


**Figure 5: Prediction accuracy as a function of $k$.**

accuracy of the sum and the product functions to increase with growing size of the training set, as OurApproach will be more likely to find votes with high email-email and user-user similarities to a given user-email.

We next evaluate the effect of $k$, used for top-$k$ retrieval of votes with similar user-email features. Recall that Lucene (and most other standard retrieval systems) support top-$k$ retrieval under linear (i.e., the sum) composition function only. Although the retrieved votes are then scored using the product composition function, $k$ has to be large enough so that the most similar candidates under the product function are among top-$k$ under the sum function. Figure 3 shows the AUC as a function of $k$. It shows that too low $k$ degrades prediction accuracy while when $k = 100$ the accuracy is nearly maximized, reaching AUC of 0.97 (on the top 10% most confident predictions). Further increasing $k$ yields negligible improvement.

# 5. SUMMARY

In this work we explored personalized spam classification using spam-resilient user-user and email-email similarity features. Using such features, derived from past user activity, has two advantages in our setting: it helps generalize extremely sparse spam votes, and it makes it harder for spammers to game the system since they have no access to the private activity information of users. We show that our approach outperforms existing solutions based on text classification and collaborative filtering, and, moreover, can be naturally combined with them yielding further improvements in classification accuracy.

# 6. REFERENCES


[1] J. Abernethy, O. Chapelle, and C. Castillo. Web spam identification through content and hyperlinks. In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 41–44. ACM, 2008.

[2] D. Agarwal and B. Chen. Regression-based latent factor models. In *KDD*, pages 19–28. ACM, 2009.

[3] M.-F. Balcan, A. Blum, and N. Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1-2):89–112, 2008.

[4] J. Basilico and T. Hofmann. A joint framework for collaborative and content filtering. In *SIGIR*, pages 550–551. ACM, 2004.

[5] A. Bergholz, J. De Beer, S. Glahn, M.-F. Moens, G. Paaß, and S. Strobel. New filtering approaches for phishing email. *J. Comput. Secur.*, 18(1):7–35, 2010.

[6] A. Bratko, B. Filipič, G. V. Cormack, T. R. Lynam, and B. Zupan. Spam filtering using statistical data compression models. *Journal of Machine Learning Research*, 7:2673–2698, 2006.

[7] 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)*. Citeseer, 2008.

[8] T. Claburn. Spam made up 94% of all e-mail in december. Technical report, Information Week, `http://www.informationweek.com/news/internet/showArticle.jhtml?articleID=197001430`, 2007.

[9] G. V. Cormack and J.-M. M. da Cruz. On the relative age of spam and ham training samples for email filtering. In *SIGIR*, pages 744–745, 2009.

[10] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *KDD*, pages 99–108, New York, NY, USA, 2004. ACM.

[11] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *CHI*, pages 581–590, New York, NY, USA, 2006. ACM.

[12] J. Goodman, G. V. Cormack, and D. Heckerman. Spam and the ongoing battle for the inbox. *Commun. ACM*, 50(2):24–33, 2007.

[13] J. Goodman and W. tau Yih. Online discriminative spam filter training. In *CEAS*, 2006.

[14] T. Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 377–384, New York, NY, USA, 2005. ACM.

[15] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 3–14, New York, NY, USA, 2008. ACM.

[16] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[17] C. Kreibich, C. Kanich, K. Levchenko, and B. Enright. Spamcraft: An inside look at spam campaign orchestration. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[18] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas – understanding captcha-solving from an economic context. In *Proceedings of the USENIX Security Symposium*, August 2010.

[19] S. Park and W. Chu. Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender systems*, pages 21–28. ACM, 2009.

[20] M. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5):393–408, 1999.

[21] V. Ramarao and M. Risher. Hadoop and the fight against shape-shifting spam. Bay Area Hadoop Summit - `http://www.slideshare.net/hadoopusergroup/mail-antispam`, May 2010.

[22] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[23] D. Sculley and G. M. Wachman. Relaxed online svms for spam filtering. In *SIGIR*, pages 415–422, New York, NY, USA, 2007. ACM.

[24] A. K. Seewald. An evaluation of naive bayes variants in content-based learning for spam filtering. *Intelligent Data Analysis*, 11(5):497–524, 2007.

[25] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20*, pages 1593–1600. MIT Press, 2007.

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