FEATURE

COLLABORATIVE SPAM FILTERING WITH THE HASHING TRICK

Josh Attenberg
Polytechnic Institute of NYU, USA

Kilian Weinberger, Alex Smola, Anirban Dasgupta, Martin Zinkevich
Yahoo! Research, USA

User feedback is vital to the quality of the collaborative spam filters frequently used in open membership email systems such as *Yahoo Mail* or *Gmail*. Users occasionally designate emails as spam or non-spam (often termed as ham), and these labels are subsequently used to train the spam filter. Although the majority of users provide very little data, as a collective the amount of training data is very large (many millions of emails per day). Unfortunately, there is substantial deviation in users' notions of what constitutes spam and ham. Additionally, the open membership policy of these systems makes it vulnerable to users with malicious intent – spammers who wish to see their emails accepted by any spam filtration system can create accounts and use these to give malicious feedback to 'train' the spam filter in giving their emails a free pass. When combined, these realities make it extremely difficult to assemble a single, global spam classifier.

The aforementioned problems could be avoided entirely if we could create a completely separate classifier for each user based solely on that user's feedback. Unfortunately, few users provide the magnitude of feedback required for this approach (many not providing any feedback at all). The number of emails labelled by an individual user approximates a power law distribution. Purely individualized classifiers offer the possibility of excellent performance to a few users with many labelled emails, at the expense of the great many users whose classifiers will become unreliable due to a lack of training data.

This article illustrates a simple and effective technique that is able to balance the wide coverage provided by a global spam filter with the flexibility provided by personalized filters. By using ideas from multi-task learning, we build a hybrid method that combines both global and personalized filters. By training both the collection of personal and global classifiers simultaneously we are able to accommodate the idiosyncrasies of each user, as well as provide a global classifier for users that label few emails. In fact, as is well known in multi-task learning [2], in addition to improving the experience of users who label many examples, this multi-task learning approach actually mitigates the impact of malicious users on the global filter. By offering specific consideration to the intents of the most active and unusual users, a global classifier is created that focuses on the truly common aspects of the classification problem. The end result is improved classifier performance for everyone, including users who label relatively few emails.

With large-scale open membership email systems such as *Yahoo Mail*, one of the main hurdles to a hybrid personal/global spam filter is the enormous amount of memory required to store individual classifiers for every user. We circumvent this obstacle with the use of the hashing trick [1, 5]. The hashing trick allows a fixed amount of memory to store all of the parameters for all the personal classifiers and a global classifier by mapping all personal and global features into a single low-dimensional feature space, in a way which bounds the required memory independently of the input. In this space, a single parameter vector, w, is trained which captures both global spam activity and the individual aspects of all active users. Feature hashing provides an extremely simple means of dimensionality reduction, eliminating the large word-to-dimension dictionary data structure typically needed for text-based classification, providing substantial savings in both complexity and available system memory.

1. HASHING-TRICK

The standard way to represent instances (i.e. emails) in text classification is the so-called bag-of-words approach. This method assumes the existence of a dictionary that contains all possible words and represents an email as a very large vector, $\vec{x}$, with as many entries as there are words in the dictionary. For a specific email, the i^{th} entry in the vector contains the number of occurrences of word i in the email. Naturally, this method lends itself to very sparse representations of instances and examples – as the great majority of words do not appear in any specific text, almost all entries in the data vectors are zero. However, when building a classifier one often has to maintain information on all words in the entire corpus (e.g. in a weight vector), and this can become unmanageable in large corpora.

The hashing trick is a dimensionality reduction technique used to give traditional learning algorithms a foothold in high dimensional input spaces (i.e. in settings with large dictionaries), by reducing the memory footprint of learning, and reducing the influence of noisy features.

The main idea behind the hashing trick is simple and intuitive: instead of generating bag-of-word feature vectors through a dictionary that maps tokens to word indices, one uses a hash function that hashes words directly into a feature vector of size b. The hash function

h : {*Strings*} → [1..b] operates directly on strings and should be approximately uniform[1].

In [5] we propose using a second independent hash function ξ : {*Strings*} → {-1, 1}, that determines whether the particular hashed dimension of a token should be incremented or decremented. This causes the hashed feature vectors to be unbiased, since the expectation of the noise for any entry is zero. The algorithm below shows a pseudo-code implementation of the hashing trick that generates a hashed bag-of-words feature vector for an email:

```
hashingtrick([string] email)
x⃗ = 0⃗
for word in email do
i = h(word)
x⃗_i = x⃗_i + ξ(word)
end for
return x⃗
```

The key point behind this hashing is that every hashed feature effectively represents an infinite number of unhashed features. It is the mathematical equivalent of a group of homographs (e.g. lie and lie) or homophones (e.g. you're and your) – words with different meanings that look or sound alike. It is important to realize that having two meanings of the same feature is no more and no less of an issue than a homograph or homophone: if a computer can guess the meaning of the feature, or more importantly, the impact of the feature on the label of the message, it will change its decision based upon the feature. If not, then it will try to make its decision based on the rest of the email. The wonderful thing about hashing is that instead of trying to cram a lot of different conflicting meanings into short words as humans do, we are trying to randomly spread the meanings evenly into over a million different features in our hashed language. So, although a word like 'antidisestablishmentarianism' might accidentally run into 'the', our hashing function is a lot less likely to make two meaningful words homographs in our hashed language than those already put there by human beings.

Of course there are so many features in our hashed language, that in the context of spam detection most features won't mean anything at all.

In the context of email spam filtering, the hashing trick by itself has several great advantages over the traditional dictionary-based bag-of-words method: 1. It considers even low-frequency tokens that might traditionally be ignored to keep the dictionary manageable – this is especially useful in view of attacks by spammers using rare variants of words (e.g. misspellings like 'viogra'). 2. Hashing the terms makes a classifier agnostic to changes in the set of terms used, and if the spam classifier is used in an online setting, the hashing trick equips the classifier with a dictionary of effectively infinite size, which helps it adapt naturally to changes in the language of spam and ham. 3. By associating many raw words in its 'infinite' dictionary (most of which never occur) with a single parameter, the meaning of this parameter changes depending upon which words are common, rare, or absent from the corpus.

So, how large a language can this hashing trick handle? As we show in the next section, if it allows us to 'square' the number of unhashed features we may possibly see, it could help us handle personalization.

2. PERSONALIZATION

As the hashing trick frees up a lot of memory, the number of parameters a spam classifier can manage increases. In fact, we can train multiple classifiers which 'share' the same parameter space [5]. For a set of users, U, and a dictionary size d, our goal is to train one global classifier, $\vec{w}_0$, that is shared amongst all users and one local classifier, $\vec{w}_u$, for each user $u \in U$. In a system with $|U|$ users, we need $|U| + 1$ classifiers. When an email arrives, it is classified by the combination of the recipient's local classifier and the global classifier $\vec{w}_0 + \vec{w}_u$ – we call this the hybrid classifier. Traditionally, this goal would be very hard to achieve, as each classifier $\vec{w}_u$ has d parameters, and hence the total number of parameters we need to store becomes $(|U| + 1)d$. Systems like *Yahoo Mail* handle billions of emails for hundreds of millions of users per day. With millions of users and millions of words, storing all vectors would require hundreds of terabytes of parameters. Further, to load the appropriate classifier for any given user in time when an email arrives would be prohibitively expensive.

The hashing trick provides a convenient solution to the aforementioned complexity, allowing us to perform personalized and global spam filtration in a single hashed bag-of-words representation. Instead of training $|U| + 1$ classifiers, we train a single classifier with a very large feature space. For each email, we create a personalized bag of words by concatenating the recipient's user id to each word of the email[2], and add to this the traditional global bag of words. All the elements in these bags are hashed into one of b buckets to form a b-dimensional representation of the email, which is then fed into the classifier. Effectively, this process allows $|U| + 1$ classifiers to share a b-dimensional parameter space nicely [5]. It is important to point out that

[1] For the experiments in this paper we used a public domain implementation of a hash function from http://burtleburtle.net/bob/hash/doobs.html.

[2] We use the ° symbol to indicate string concatenation.

the one classifier $\vec{x}$ – over b hashed features – is trained *after* hashing. Because b will be much smaller than d x $|U|$, there will be many hash collisions.

However, because of the sparsity and high redundancy of each email, we can show that the theoretical number of possible collisions does not really matter for most of the emails.

Moreover, because the classifier is aware of any collisions before the weights are learned, the classifier is not likely to put weights of high magnitude on features with an ambiguous meaning.

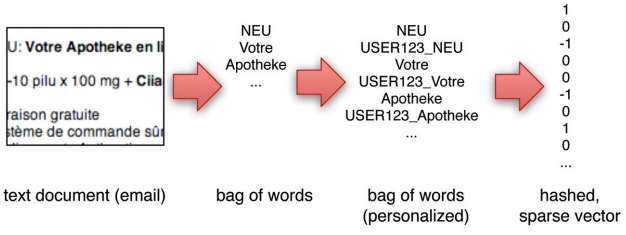

Figure 1: Global/personal hybrid spam filtering with feature hashing.

Intuitively, the weights on the individualized tokens (i.e. those that are concatenated with the recipient's id) indicate the personal eccentricities of the particular users. Imagine for example that user 'barney' likes emails containing the word 'viagra', whereas the majority of users do not. The personalized hashing trick will learn that 'viagra' itself is a spam indicative word, whereas 'viagra_barney' is not. The entire process is illustrated in Figure 1. See the algorithm below for details on a pseudo-code implementation of the personalized hashing trick. Note that with the personalized hashing trick, using a hash function $h : \{Strings\} \rightarrow [1..b]$, we only require b parameters independent of how many users or words appear in our system.

personalized_hashingtrick(string *userid*, [string] *email*)
$\vec{x} = \vec{0}$
for *word* **in** *email* **do**
$i = h(word)$
$\vec{x}_i = \vec{x}_i + \xi(word)$
$j = h(word\ userid)$
$\vec{x}_j = \vec{x}_j + \xi(\text{word} \circ \text{userid})$
end for
return $\vec{x}$

3. EXPERIMENTAL SET-UP AND RESULTS

To assess the validity of our proposed techniques, we conducted a series of experiments on the freely distributed

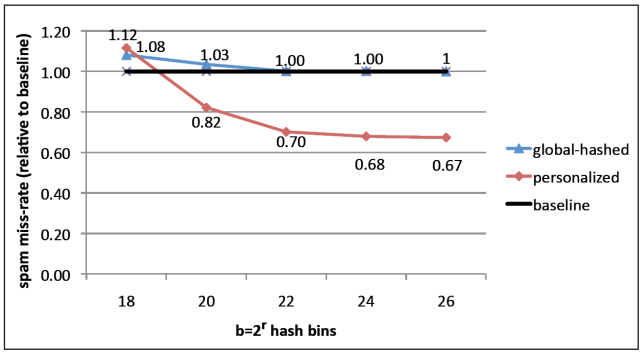

Figure 2: The results of the global and hybrid classifiers applied to a large-scale real-world data set of 3.2 million emails.

trec07p benchmark data set, and on a large-scale proprietary data set representing the realities of an open-membership email system. The trec data set contains 75,419 labelled and chronologically ordered emails taken from a single email server over four months in 2007 and compiled for trec spam filtering competitions [3]. Our proprietary data was collected over 14 days and contains n = 3.2 million anonymized emails from $|U|$ = 400,000 anonymized users. Here the first ten days are used for training, and the last four days are used for experimental validation. Emails are either spam (positive) or ham (non-spam, negative). All spam filter experiments utilize the Vowpal Wabbit (VW) [4] linear classifier trained with stochastic gradient descent on a squared loss. Note that the hashing trick is independent of the classification scheme used; the hashing trick could apply equally well with many learning-based spam filtration solutions. To analyse the performance of our classification scheme we evaluate the spam catch rate (SCR, the percentage of spam emails detected) of our classifier at a fixed 1% ham misclassification rate (HMR, the percentage of good emails erroneously labelled as spam). We note that the proprietary nature of the latter data set precludes publishing of exact performance numbers. Instead we compare the performance to a baseline classifier, a global classifier hashed onto $b = 2^{26}$ dimensions. Since 2^{26} is far larger than the actual number of terms used, d = 40M, we believe this is representative of full-text classification without feature hashing.

4. THE VALIDITY OF HASHING IN EMAIL SPAM FILTERING

To measure the performance of the hashing trick and the influence of aggressive dimensionality reduction on classifier quality, we compare global classifier performance to that of our baseline classifier when hashing onto spaces of dimension $b = \{2^{18}, 2^{20}, 2^{22}, 2^{24}, 2^{26}\}$ on our proprietary data

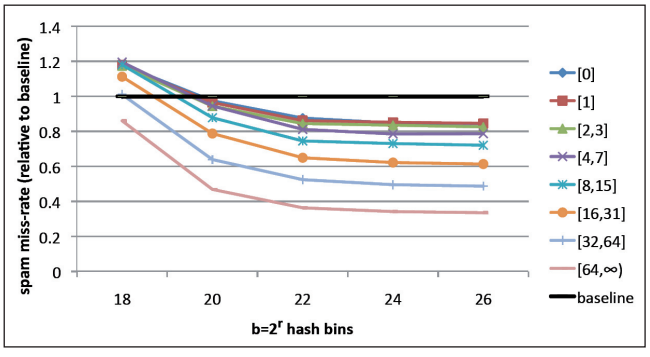

Figure 3: The amount of spam left in users' inboxes, relative to the baseline. The users are binned by the amount of training data they provide.

set. The results of this experiment are displayed as the blue line in Figure 2. Note that using 2^{18} bins results in only an 8% reduction in classifier performance, despite large numbers of hash collisions. Increasing b to 2^{20} improves the performance to within 3% of the baseline. Given that our data set has 40M unique tokens, this means that using a weight vector of 0.6% of the size of the full data results in approximately the same performance as a classifier using all dimensions.

Previously, we have proposed using hybrid global/personal spam filtering via feature hashing as a means for effectively mitigating the effects of differing opinions of spam and ham amongst a population of email users. We now seek to verify the efficacy of these techniques in a realistic setting. On our proprietary data set, we examine the techniques illustrated in Section 2 and display the results as the red line in Figure 2. Considering that our hybrid technique results from the cross product of $|U| = 400$K users and $d = 40$M tokens, a total of 16 trillion possible features, it is understandable that noise induced by collisions in the hash table adversely affects classifier performance when b is small. As the number of hash bins grows to 2^{22}, personalization already offers a 30% spam reduction over the baseline, despite aggressive hashing.

In any open email system, the number of emails labelled as either spam or non-spam varies greatly among users. Overall, the labelling distribution approximates a power law distribution. With this in mind, one possible explanation for the improved performance of the hybrid classifier in Figure 2 could be that we are heavily benefiting those few users with a rich set of personally labelled examples, while the masses of email users – those with few labelled examples – actually suffer. In fact, many users do not appear at all during training time and are only present in our test set. For these users, personalized features are mapped into hash buckets with weights set exclusively by other examples, resulting in some interference being added to the global spam prediction.

In Section 2, we hypothesized that using a hybrid spam classifier could mitigate the idiosyncrasies of the most active spam labellers, thereby creating a more general classifier for the remaining users, benefiting everyone. To validate this claim, we segregate users according to the number of training labels provided in our proprietary data. As before, a hybrid classifier is trained with $b = \{2^{18}, 2^{20}, 2^{22}, 2^{24}, 2^{26}\}$ bins. The results of this experiment are seen in Figure 3.

Note that for small b, it does indeed appear that the most active users benefit at the expense of those with few labelled examples. However, as b increases, therefore reducing the noise due to hash collisions, users with no or very few examples in the training set also benefit from the added personalization. This improvement can be explained if we recall the subjective nature of spam and ham – users do not always agree, especially in the case of business emails or newsletters. Additionally, spammers may have infiltrated the data set with malicious labels. The hybrid classifier absorbs these peculiarities with the personal component, freeing the global component to truly reflect a common definition of spam and ham and leading to better overall generalization, which benefits all users.

5. MITIGATING THE ACTIONS OF MALICIOUS USERS WITH HYBRID HASHING

In order to simulate the influence of deliberate noise in a controlled setting we performed additional experiments on the trec data set. We chose some percentage, *mal*, of 'malicious' users uniformly at random from the pool of email receivers, and set their email labels at random. Note that having malicious users label randomly is actually a harder case than having them label them adversarially in a consistent fashion – as then the personalized spam filter could potentially learn and invert their preferences.

Figure 4 presents a comparison of global and hybrid spam filters under varying loads of malicious activity and different sized hash tables. Here we set $mal \in \{0\%, 20\%, 40\%\}$. Note that malicious activity does indeed harm the overall spam filter performance for a fixed classifier configuration. The random nature of our induced malicious activity leads to a 'background noise' occurring in many bins of our hash table, increasing the harmful nature of collisions. Both global and hybrid classifiers can mitigate this impact somewhat if the number of hash bins b is increased. In short, with malicious users, both global (dashed line) and hybrid (solid line) classifiers require more hash bins to achieve near-optimum performance. Since the hybrid classifier has more tokens, the number of hash collisions is also correspondingly larger. Given a large enough number of hash bins, the hybrid classifier clearly outperforms the single global classifier under the malicious

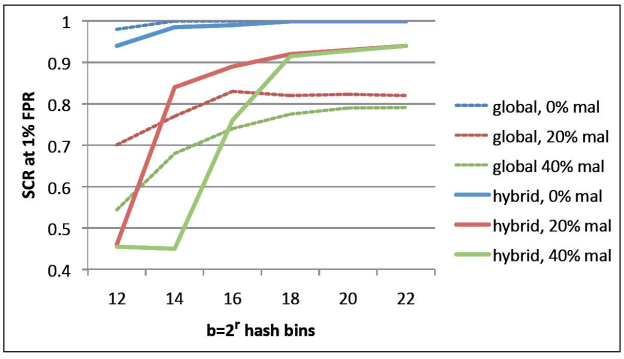

Figure 4: The influence of the number of hash bins on global and hybrid classifier performance with varying percentages of malicious users.

settings. We do not include the results of a pure local approach, as the performance is abysmal for many users due to a lack of training data.

6. CONCLUSION

This work demonstrates the hashing trick as an effective method for collaborative spam filtering. It allows spam filtering without the necessity of a memory-consuming dictionary and strictly bounds the overall memory required by the classifier. Further, the hashing trick allows the compression of many (thousands of) classifiers into a single, finite-sized weight vector. This allows us to run personalized and global classification together with very little additional computational overhead. We provide strong empirical evidence that the resulting classifier is more robust against noise and absorbs individual preferences that are common in the context of open-membership spam classification.

REFERENCES

[1] Attenberg, J.; Weinberger, K.; Dasgupta, A.; Smola. A.; Zinkevich, M. Collaborative email-spam filtering with the hashing trick. Proceedings of the Sixth Conference on Email and Spam, CEAS 2009, 2009.

[2] Caruana, R. Algorithms and applications for multitask learning. Proc. Intl. Conf. Machine Learning, pp.87–95. Morgan Kaufmann, 1996.

[3] Cormack, G. TREC 2007 spam track overview. The Sixteenth Text REtrieval Conference (TREC 2007) Proceedings, 2007.

[4] Langford, J.; Li, L.; Strehl, A. Vowpal Wabbit online learning project. http://hunch.net/?p=309, 2007.

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