Automatically Tagging Email by Leveraging Other Users' Folders

Yehuda Koren, Edo Liberty, Yoelle Maarek, and Roman Sandler
Yahoo! Research
Haifa 31905, ISRAEL

ABSTRACT

Most email applications devote a significant part of their real estate to organization mechanisms such as folders. Yet, we verified on the Yahoo! Mail service that 70% of email users have never defined a single folder. This implies that one of the most well known email features is underexploited. We propose here to revive the feature by providing a method for generating a lighter form of folders, or tags, benefiting even the most passive users. The method automatically associates, whenever possible, an appropriate semantic tag with a given email. This gives rise to an alternate mechanism for organizing and searching email.

We advocate a novel modeling approach that exploits the overall population of users, thereby learning from the wisdom-of-crowds how to categorize messages. Given our massive user base, it is enough to learn from a minority of the users who label certain messages in order to label that kind of messages for the general population. We design a novel cascade classification approach, which copes with the severe scalability and accuracy constraints we are facing. Significant efficiency gains are achieved by working within a low dimensional latent space, and by using a novel hierarchical classifier. Precision level is controlled by separating the task into a two-phase classification process.

We performed an extensive empirical study covering three different time periods, over 100 million messages, and thousands of candidate tags per message. The results are encouraging and compare favorably with alternative approaches. Our method successfully tags 72% of incoming email traffic. Performance-wise, the computational overhead, even on surge large traffic, is sufficiently low for our approach to be applicable in production on any large Web mail service.

1. INTRODUCTION

Mail products and services from the early Lotus Notes and Thunderbird applications to more recent Web-based email services such as Hotmail, Yahoo! Mail and Gmail, all offer organization features such as folders or labels to help users browse their email. While there has been much heated discussion on which mechanism is most suitable when organizing personal information in general, in the so-called "folders vs labels war"[1], two pre-requisites for either mechanism are that users need to (1) manually define and maintain such folders or labels and (2) manually move messages into folders or label these messages as appropriate. It turns out however that most users do not take the pain to do this. Indeed, we have verified on the Yahoo! Mail service, one of the top 3 mail services in terms of active number of users, that 70% of email users have never defined even a single folder. We can safely assume that other mail clients or services do not fare better in spite of the prime real estate space folders and labels use on the mail interface.

We propose here to revive this underexploited feature by devising a mechanism for automatically associating "tags" with email messages. Note we are not taking sides in the folders vs labels debate as each has its pros and cons as demonstrated by Civan et al in [8]. Instead we focus on a general mechanism that can be used to support the automation of either labels or folders assignment. More specifically, we want to associate with any email message a ranked list of tags when appropriate. These tags can then be directly mapped into user-customizable labels or into disjoint folders (or even a hierarchy of folders if tags are appropriately clustered).

One unique aspect of our proposed solution is that unlike previous approaches, we propose to first look "horizontally" at the overall population of folders' users, rather than immediately personalizing for each individual user's inbox. Given that Yahoo! Mail counts several hundred millions active users, even only 30% of the population gives a wealth of data for us to learn from.

In order to check whether such an approach is realistic, we first needed to verify that folder users expose enough of a common behavior for us to extrapolate to the rest of the population. So as a prologue to this work, we examined the distribution of folder names on a large sample of Yahoo! Mail users. We observed that many users share the same "head" folders for similar needs. Figure 1 shows the existence and frequency of these head folders. These include expected head folders such as "travel","jobs", "friends", "family", "bills", "shopping", folders exposing new Web habits and behaviors such as "facebook", or more surprising ones like "recipes". This preliminary study gave us the motivation for this work, namely as users do seem to have common needs, we can hope to define a classification model that would gen-

KDD '11 San-Diego, USA

[1]http://www.cloudave.com/1912/google-gmail-finally-ends-the-folder-vs-label-war-what-next-find-the-answer-here/

erate popular tags. This would allow us to automate stage 1 of the email organization (the naming of folders/labels) as well as the associated stage 2, the actual labeling of any incoming message with a popular tag when appropriate. Note that we do not attempt to support here a full classification model for rare needs. We could indeed observe that like in Web search and commerce [12], many folder users expose "extraordinary" long tail tastes and generated rare if not unique folders. Yet, we argue that providing even common tags for "ordinary" needs to this majority of users who never organize email is already extremely valuable.

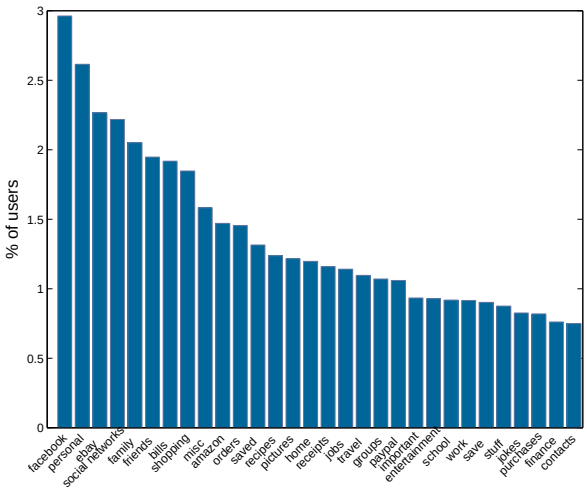

Figure 1: Most popular folders

Devising such an automated tagging system in the context of a real-life huge mail system however poses multiple challenges:

1. **Scalability:** As we are dealing with a huge number of users and even a larger number of incoming messages. Given today's latency expectations, is critical to meet tight performance requirements, and devise highly scalable techniques, raising questions such as whether all messages need to be processed? can we afford to parse and utilize each message full-text body? The resulting system must examine thousands of potential tags for billions of messages per day. Moreover, this must be performed in the time-critical email delivery pipeline.
2. **Precision:** False classification must be kept to a minimum as users are in a passive role where they are presented tags by default. Users are quick to judge when dealing with personal content they are extremely familiar with and will tolerate only minimal irrelevant results. Techniques need to be devised to control and tune precision as needed, as we strive to a close to perfect precision (even at the cost of lowering recall).

Note that there exist multiple additional challenges on the product side in order to ensure a smooth user experience and eventual adoption. This requires answering questions such as: do tags need to be exposed by default and removed upon demand? can they be renamed? how do they live side by side with existing folders? etc. We are not addressing these here and we will keep them for future work. We focus here on these scalability and precision requirements, and propose to use a two-phase cascade classification process in order to address them. For the first phase, we introduce a novel multi-class classifier, referred to as the *tag-vs-tag classifier*, whose role is to pick the most appropriate tags for each message and return them as a short ranked list. For the second phase, we use a *tag-vs-notag classifier*, which separately verifies for each of the candidate tags whether it should or should not be presented.

Using this two-phase-process allows our system to perform well on both scalability and precision requirements. In terms of scalability, the novel design of the presented tag-vs-tag classifier outputs $O(1)$ top tags for a message in time that scales logarithmically with the overall number of tags. For a typical message in our implementation, this requires less than 2 microseconds. This process is described in Sec. 4. In terms of precision, correct thresholds for the *tag-vs-notag classifiers* (Sec. 5) filter out tags that do not meet the prescribed precision requirements. Since this phase only examines $O(1)$ candidate tags, it cannot hurt our overall performance guarantee. Setting these thresholds is made significantly harder due to the fact that a single message might receive multiple valid labels. For example, "bills", "shopping" and "amazon" (tags that are derived from the popular folders shown in Figure 1) might all correctly describe different aspects of a single message. Overcoming these difficulties is explained in Sec. 5.

The key contributions of this paper are the following:

- A new "horizontal" view of the email classification challenge that allows to leverage the wisdom of crowds for tagging messages.
- A novel cascade classification model specifically designed for performance and scalability. This model selects the appropriate tag out thousands of options under harsh performance requirements.
- A new estimator for the presence of each tag within the (unlabeled) Inbox. This facilitates quality control via the automatic setting of precision cutoffs per tag.
- An optional "vertical" personalization to allow reducing the number of suggested tags while maintaining diversity for a better user experience.
- An analysis of email corpus much larger than previously reported.

2. RELATED WORK

The task of automatically assigning email messages into folders, commonly referred to as "foldering", is usually considered a special case of text categorization. Since we cannot hope to survey the entire field of text categorization we focus here only on those efforts aimed directly at email foldering. As in most domains, application of standard text based methods require special accommodations; please see, e.g., [4, 9] for good comparative studies.

All reported works classify messages of a single user (or mail-list [2]). The approaches include rule-based systems (e.g., [9, 18]), IR methods: k-NN [6] and tf-idf, and machine learning methods: naive Bayes (NB), decision trees, support vector machines(SVM), maximum entropy (MaxEnt) and neural networks (NN) [1, 4, 10, 14]. Another common approach is to consider additional properties of email messages and utilize structural features, e.g., [2, 15]. Results for

these methods are commonly given for the Enron and SRI datasets [4, 6, 7, 15].

A significant aspect differentiating our work is considering many users simultaneously. In previous works, each user's emails and foldering habits were learned separately. In this work we share information between users and learn the foldering habits of the entire population. This is crucial for being able to service passive (non-foldering) users, which are the majority of the population. This global approach, however, significantly raises the computational challenges. Instead of the few tags used by a single user, here we must consider for each message thousands of possible tags used by the entire population. Moreover, it is quite likely that multiple different tags are appropriate for a single message.

In addition, our performance requirements are much tighter, as we need to scale to a large production email system. Hence, we must resort to fastest methods, while many of the previously reported approaches would not be practical. This includes severe limitations on the message features available to us at delivery time. As a side note, the largest publicly available dataset (Enron), contains 500 times fewer messages than the corpora we employ here,

3. BASIC NOTIONS AND CONVENTIONS

This work deals with *email messages* (or simply, *messages*). Each message is viewed as a set of *features*. The used message features are unigrams and bigrams from the subject line and sender details. The features are binary (either exist or not, with no associated count or weight). Some messages are labeled with *tags*. In practice, the tags are the folder names to which users assign their messages. Hence, in our training data at most one tag is associated with any message. Typically, we deal with a few thousand different tags, assigned to many millions of messages.

We index individual tags by x and y and their number by n. Messages are indexed by i and j. The set of features associated with message i is denoted by F_i. We work with four message sets: a set of labeled training messages $\mathcal{T}$, labeled test messages $\hat{\mathcal{T}}$, unlabeled training messages $\mathcal{I}$ and unlabeled test messages $\hat{\mathcal{I}}$. Unlabeled messages are sampled from non-foldered inbox messages.

4. PHASE 1: PICK A TAG

As explained in the introduction, our process begins with efficiently searching for the most appropriate tags for each message. In this section we describe three classifiers that are built to scale to a large number of tags and messages. While the first two models are adequately fast to train offline, their online prediction performance is still too slow to be practical. The third exhibits also efficient online prediction. Common to all these methods is the need for a low dimensional representation of messages and tags.

4.1 Latent factor representation

A key component underlying our models is a mapping of both messages and tags into a joint low dimensional space of dimension ℓ, to which we refer as the latent factor representation, as follows:

- Each tag x is mapped into a vector $p_x \in \mathbb{R}^\ell$
- Each feature f is mapped into a vector $v_f \in \mathbb{R}^\ell$
- Each message i is mapped to a vector $q_i \in \mathbb{R}^\ell$, defined by the message features to be: $q_i = \frac{1}{|F_i|}\sum_{f\in F_i} v_f$

In addition, we introduce biases for tags, such that the bias of tag x is denoted by b_x. The latent space representation strives to capture the semantics of the tags and messages, such that affinities in the latent space reflect semantic relations. Working in low-dimensional space is extremely beneficial. It allows us to design efficient multiclass classifiers, with low dimensional dense representation leading to memory locality and a small number of used parameters. In our setting, we use a latent space dimensionality of $\ell = 50$, which gives a good tradeoff between time and accuracy.

Given such a latent space representation and a message i, tags are ranked by their affinity to the message defined for each tag x as

$$r_{ix} \stackrel{\text{def}}{=} b_x + q_i^T p_x = b_x + \frac{1}{|F_i|}\sum_{f\in F_i} v_f^T p_x. \quad (1)$$

Thus, the learning task is learning the latent factor representation. We will offer two alternative ways to learn such a mapping. As explained earlier, the major challenge would be making such a process computationally efficient.

4.2 A max-likelihood approach

Given a message i, the tags are assumed to follow the multinomial distribution

$$p(x|i;\Theta) = \frac{\exp(r_{ix})}{\sum_y \exp(r_{iy})} \quad (2)$$

where Θ denotes set of model parameters.

We seek to assign tags to all messages in a way that maximizes the log likelihood of the training data

$$L(\mathcal{T};\Theta) \stackrel{\text{def}}{=} \sum_{(i,x)\in\mathcal{T}} \log p(x|i;\Theta). \quad (3)$$

Note that this model is a latent space analogous to multinomial logistic regression [13] and to MaxEnt [4].

Learning proceeds by stochastic gradient ascent. Given a training example (i, x) we update each parameter θ by

$$\Delta\theta = \eta\frac{\partial \log p(x|i;\Theta)}{\partial\theta} = \eta\left(\frac{\partial r_{ix}}{\partial\theta} - \sum_y p(y|i)\frac{\partial r_{iy}}{\partial\theta}\right) \quad (4)$$

where η is the learning rate.

However, such a training scheme is too slow to be practical, as each of its update rules requires summing over the n tags. Thus, we resort to a sampling approach of the weighted sum in (4). We adopt the importance sampling trick by Bengio and Senécal [5].

Let $p(x|\mathcal{T})$ denote the empirical probability of tag x in the labeled training set $\mathcal{T}$. This is the distribution from which we sample tags into a set $\mathcal{J}$. Thus, the expensive-to-compute $p(y|i)$ probabilities appearing in (4) are replaced with the weighting scheme

$$w(x|i) = \frac{\exp(r_{ix})/p(x|\mathcal{T})}{\sum_{y\in\mathcal{J}}\exp(r_{iy})/p(y|\mathcal{T})} \quad (5)$$

Accordingly, the approximated gradient ascent step given training example (i, x) is

$$\Delta\theta = \eta\left(\frac{\partial r_{ix}}{\partial\theta} - \sum_{y\in\mathcal{J}} w(y|i)\frac{\partial r_{iy}}{\partial\theta}\right) \quad (6)$$

The set $\mathcal{J}$ is sampled with replacement based on $p(x|\mathcal{T})$. To do so, we randomly sample a (i, x) pair from the training set,

and add the tag x to $\mathcal{J}$. As mentioned in [5], the size of set $\mathcal{J}$ should grow as the training process proceeds, because at later training phases more delicate parameter adjustments need to be made. We devise a simple rule for controlling the sample size based on the fitness of the current estimate. Given a training example (i, x), we first sample a fixed fraction of the tags population (in our implementation, $n/32$ tags). Then, we keep sampling tags into $\mathcal{J}$ till satisfying:

$$\sum_{y\in\mathcal{J}} p(y|i) > c \cdot p(x|i) \Leftrightarrow \sum_{y\in\mathcal{J}} \exp(r_{iy}) > c \cdot \exp(r_{ix}) \quad (7)$$

We use $c = 2$ and a maximal sample size of n. This adaptive sampling automatically increases the sample size as parameters converge to their final values and the correct tags receive relatively high probability. We refer to this method as *FlatLikelihood*.

4.3 A normalized hinge loss approach

Here we adapt a learning to rank approach, originated by Weston et al. [19]. We try to minimize the number of misclassified messages by noticing that message i is misclassified if it is tagged x and for some other tag y we have $r_{ix} \leqslant r_{iy}$. Our error function is therefore defined by

$$E(\mathcal{T}) = \sum_{(i,x)\in\mathcal{T}} I(\exists_y : \ r_{ix} \leqslant r_{iy}) \,. \quad (8)$$

Let us denote the number of tags violating example (x, i) by

$$\#violations(x, i) = |\{y|r_{ix} \leqslant r_{iy}\}|. \quad (9)$$

Using this notation, we rewrite (8) in an equivalent form

$$E(\mathcal{T}) = \sum_{(i,x)\in\mathcal{T}} \frac{\sum_y I(r_{ix} \leqslant r_{iy})}{\#violations(x, i)} \quad (10)$$

where $0/0$ is taken as 0 [19]. Replacing the indicator function with a continuous hinge loss with a margin, to get the loss function

$$E^1(\mathcal{T}) = \sum_{(i,x)\in\mathcal{T}} \frac{\sum_y |1 - r_{ix} + r_{iy}|_+}{\#violations^1(x, i)} \quad (11)$$

where $|x|_+ = x$ if $x > 0$ or zero otherwise. In addition the margin-aware $\#violations^1(x, i)$ is defined as $|\{y|r_{ix} \leqslant r_{iy} + 1\}|$.

Minimizing (11) is costly due to the requirement to sum over all potential tags. Hence, much like in the max-likelihood case, a sampling approach is used. To make things easier, now the sum is not weighted by an expensive-to-compute probability, so we can get its unbiased estimate by uniformly drawing tags with replacement into a set $\mathcal{J}$. This way, an unbiased estimator of $E^1(\mathcal{T})$ would be

$$\hat{E}^1(\mathcal{T}) = \sum_{(i,x)\in\mathcal{T}} \frac{\sum_{y\in\mathcal{J}} |1 - r_{ix} + r_{iy}|_+}{\#violations^1(\mathcal{J}, x, i)} \quad (12)$$

where $\#violations^1(\mathcal{J}, x, i) \stackrel{def}{=} |\{y \in \mathcal{J}|r_{ix} \leqslant r_{iy} + 1\}|$.

Learning is done by a stochastic gradient descent on $\hat{E}^1$. Given a training example (i, x), we sample into $\mathcal{J}$ $n/32$ tags, and keep on sampling till $\#violations^1(\mathcal{J}, x, i) \geqslant 1$ or till hitting n samples. We refer to this method as *FlatNormalizedHinge*.

In our experimental study, both FlatLikelihood and FlatNormalizedHinge produce better results than several known baselines. However, they still suffer from poor online computational efficiency. Indeed, for each new message, we must evaluate all possible tags. Thus, the main use of these two models is in constructing a tree hierarchy of the tags, as described in the next section.

4.4 Hierarchical classifier

We stick to the principles of the latent factor models developed in the previous section. However, in order to let their prediction performance scale to a large number of tags and messages, we draw inspiration from works on language modeling, where words are modeled within a hierarchy, thereby enabling prediction of the next word out of a large vocabulary [16, 17].

The first step is embedding all tags as leaves of a binary hierarchical structure by recursively partitioning the tags into balanced groups. To this end, we represent each tag x by its latent space mapping p_x, as computed by either FlatLikelihood or FlatNormalizedHinge. Then we apply a k-means clustering (the variant by [3]) with $k = 2$ to the tags' representations. We create two equally sized clusters by sorting the tags according to their distance from the two means, and splitting at the middle point. Then, we continue recursively until reaching singletons. Hence, the n tags are identified with leafs of a binary tree (hierarchy) of depth $O(\log n)$.[2]

Each internal tree node a is associated with a hyperplane $b_a + z^T p_a = 0$ for $p_a, z \in \mathbb{R}^\ell$. For a given message i, along with its latent factor representation q_i, we traverse the tree according to half spaces q_i is in (with respect to the internal nodes' hyperplanes). More precisely, we employ the following traversal rule: at internal node a, proceed to the left child if $b_a + q_i^T p_a > 0$, otherwise proceed to the right child. Hence, given a message and a trained tree, one can traverse the root-to-leaf path in $O(\log n)$ steps.

We still need to devise ways for learning the parameters of the model, i.e., the internal nodes' hyperplanes and the messages' representation. Intuitively, a classifier trained on the formed hierarchy should encourage training examples to follow the paths leading to their true tags. More precisely, define $d_{ax} = 1$ if the leaf corresponding to x is a left offspring of a and $d_{ax} = -1$ if x is a right offspring of a.[3] If message i is tagged by x we strive to have $sign(b_a + q_i^T p_a) = d_{ax}$. If this is true for all internal nodes leading from the root to x, then message i will be correctly classified. Learning the parameters (b_a, p_a, q_i) can be based on either the max likelihood principle or on the normalized hinge loss. We first show the derivation for the normalized hinge loss.

Given a hierarchy T, and a training example (i, x), we define the set of violating nodes as

$$\#violations^1(T, x, i) = |\{a \in \text{path}_x|(b_a + q_i^T p_a)d_{ax} \leqslant 1\}|.$$

Here path_x stands for the set of internal nodes on the path from the root of the tree to x. Then, the normalized hinge loss associated with the training set, given hierarchy T is

[2]We have tried several other clustering methods. Our experiments showed that performing k-means clustering on normalized vectors (whereas tag x is mapped to $p_x/\|p_x\|$) slightly improves subsequent classifier performance.

[3]A leaf x is a left(right) offspring of a if the left(right) child of a is on the path from the root node to x.

defined as

$$\hat{E}^1(\mathcal{T};T) = \sum_{(i,x)\in\mathcal{T}} \frac{\sum_{a\in \text{path}_x} |1-(b_a + q_i^T p_a)d_{ax}|_+}{\#violations^1(T,x,i)} \quad (13)$$

Optimizing is done by stochastic gradient descent. Each training example requires traversing only a $O(\log n)$-long path, which compares favorably with the $O(n)$ tag comparisons required by the FlatNormalizedHinge (even after sampling). The impact of this structure for online prediction is even more significant at stated above. Henceforth, we dub this method as *HierNormalizedHinge*.

The derivation for maximizing the log likelihood is analogous. For a message i, at a tree node a we define the probability to proceed to the left child using the logistic function as $\sigma(a,i) = (1+\exp(-b_a - q_i^T p_a))^{-1}$. Learning is done by stochastic gradient ascent on the training set log likelihood

$$L(\mathcal{T};T) = \sum_{(i,x)\in\mathcal{T}} \sum_{a\in \text{path}_x} \frac{1}{2}(1+d_{ax})\log\sigma(a,i) + \frac{1}{2}(1-d_{ax})\log(1-\sigma(a,i)) \quad (14)$$

Henceforth, this method is referred to as *HierLikelihood*.

Hierarchical classifiers are natural for suggesting a single tag per message. In case more than one tag suggestion is required, we use an empirical confusion matrix. For each tag we record the tags it most commonly confused with. This way a single leaf outputs a list of multiple ranked tag suggestions. We elaborate more on this, and on performance evaluation in the empirical study (Sec. 6).

5. PHASE 2: TO TAG OR NOT TO TAG?

In the second phase we get a message together with a short list of suggested tags produced by the first phase. For each of the suggested tags, we decide whether we should keep it or filter it out. This is achieved by applying a binary classifier to each message and suggested tag independently.

For each tag we train a binary classifier by comparing messages labeled by the tag to unlabeled messages. Unlike in the first phase, the measure of success here is hard to evaluate. This is because each tag has a different presence within the set of unlabeled messages. For example, the tag "generic stuff" is rarely used (thus being associated with only a few labeled messages), and still a relatively large number of unlabeled messages could justifiably receive this tag. Moreover, different kinds of messages have different probabilities of being labelled. This makes the distribution of labeled messages different than the distribution of unlabeled one, e.g., travel related messages are much more likely to be tagged than promotions. Addressing this issue is the main challenge faced by the second phase. We first address the general problem and then discuss it for our case.

From this point on we view this problem as a one-tag-problem: either tag or not tag. We are required to construct a high precision classifier f for tag x. We make the crucial assumption that each message which justifies tag x is actually labeled x with probability $prob_x$ independently of the message content. In other words, the probability of a message being labeled x is independent of its content *conditioned* on x justifying tag x.

We denote by $\hat{\mathcal{T}}_x$ the set of messages labeled by x in $\hat{\mathcal{T}}$ and by $\hat{\mathcal{I}}_x$ the set of unlabeled messages in $\hat{\mathcal{I}}$ which justify tag x. In order to argue against the precision of f we must estimate the size of $\hat{\mathcal{I}}_x$. This can be performed with the help of a picky-classifier g whose precision is (almost) perfect but whose recall might be very low. We assume such g exists. Let $g(\hat{\mathcal{T}}_x)$ denote the set of messages g identifies as labeled by x in $\hat{\mathcal{T}}_x$. From our assumption, the content distribution in $\hat{\mathcal{T}}_x$ and $\hat{\mathcal{I}}_x$ is identical. This means that g should exhibit the same recall on both sets which gives $|g(\hat{\mathcal{I}}_x)|/|\hat{\mathcal{I}}_x| \approx |g(\hat{\mathcal{T}}_x)|/|\hat{\mathcal{T}}_x|$. Moreover, since g's precision is almost perfect we have that $|g(\hat{\mathcal{I}}_x)| \approx |g(\hat{\mathcal{I}})|$. We therefore get that

$$|\hat{\mathcal{I}}_x| \approx |g(\hat{\mathcal{I}})| \cdot |\hat{\mathcal{T}}_x|/|g(\hat{\mathcal{T}}_x)|. \quad (15)$$

Having approximated the size of $\hat{\mathcal{I}}_x$ we return to estimating the precision of our classifier f. Again by our independent tagging probability assumption, we have that the recall of f on $\hat{\mathcal{T}}_x$ approximates its recall on $\hat{\mathcal{I}}_x$, i.e., $|f(\hat{\mathcal{I}}_x)|/|\hat{\mathcal{I}}_x| \approx |f(\hat{\mathcal{T}}_x)|/|\hat{\mathcal{T}}_x|$. Moreover, $f(\hat{\mathcal{I}}_x)$ is exactly the set of unlabeled messages which are *correctly* classified by f. Dividing this by the overall number of labeled messages $f(\hat{\mathcal{I}})$ we get the precision of f.

$$precision(f) = \frac{|f(\hat{\mathcal{I}}_x)|}{|f(\hat{\mathcal{I}})|} \approx \frac{|f(\hat{\mathcal{T}}_x)|}{|g(\hat{\mathcal{T}}_x)|} \cdot \frac{|g(\hat{\mathcal{I}})|}{|f(\hat{\mathcal{I}})|}. \quad (16)$$

We now discuss our specific choices of classifiers. For g we use a Naïve Bayes classifier [13] trained to separate messages tagged x in $\mathcal{T}$ from all messages of $\mathcal{I}$. The reason for using the Naïve Bayes classifier here is a relative comfort in setting a uniformly sufficiently high cutoff value for achieving our close-to-perfect precision demands. Consequently, we set the cutoff to exp(10). It is worth noting that the choice of the Naïve Bayes classifier is somewhat arbitrary. Many other binary classifiers could be used with appropriate thresholds to the same effect.

For f, several binary classifiers were tried, including Naïve-Bayes, hinge loss-based, and logistic regression-based. The best empirical results were achieved with the method described in the following. We keep using the mapping of messages and tags into a low dimensional space as described in Sec. 4.1. The mapping here is not identical to the ones computed in Sec. 4.1 but, for the sake of clarity, we keep using the same symbols. Moreover, for computational reasons the mapping of messages to the latent space, q_i, is shared between all tags. An individual tag classification is carried out by thresholding on $r_{ix} = b_x + q_i^T p_x$.

We follow a logistic-regression strategy, implemented in the low dimensional latent space. For a message i, we define its probability of it being labeled x as:

$$p(x|i) = (1+\exp(-r_{ix}))^{-1} \quad (17)$$

Hence, $1-p(x|i)$ is the probability that message i is not labeled by x. We maximize the log-likelihood of the combined training set

$$\sum_{(i,x)\in\mathcal{T}} \log p(x|i) + \sum_{i\in\mathcal{I}} \sum_{x} \log(1-p(x|i)) \quad (18)$$

Learning is done by stochastic gradient ascent. Training time is potentially long due to the second summation which runs over all unclassified messages and tags. This can be controlled by limiting the relative size of $|\mathcal{I}|$ to be comparable to the number of messages belonging to a single tag. In terms of classifier accuracy such a limitation is reasonable.

We set each classifier f_x such that $f_x(i)$ returns true if $r_{ix} \geq \gamma_x$. The thresholds γ_x are chosen such that the pre-

cision of f_x is high enough according to Equation 16. More details are given in the empirical study (Sec. 6).

6. EMPIRICAL STUDY

6.1 Dataset description

We experimented with email data gathered from anonymized Yahoo! Mail users. Performance and privacy requirements limited us to analyzing only the subject and sender fields of each message. Overall we collected close to 200 million foldered messages. The messages were partitioned into three different datasets by arrival month that we refer to as T1, T2, and T3. Within each dataset, we split the messages into train and test sets by arrival time.

We limited ourselves to popular user-defined folders defined by a large enough number of users and directly mapped them into a raw list of 6,000 tags. We then ran the Flat-Likelihood model of Sec. 4.2 on the T1 corpus. The resulting embedding of the tags within the low dimensional space allowed us to identify synonymous tags, by clustering tags by their low dimensional embeddings' cosine similarity. For each of the 819 non-singleton clusters a representative tag was selected. Namely, the one most frequently used. For example, for the cluster {*facebook, fb, facebook stuff, face book,...*} the tag *facebook* was retained and all other tags/folders merged. Similarly *fun* was retained for {*fun, jokes, humor, keepers, fun stuff, funny, funnies, jokes etc, funny stuff, funnys, joke,...*}, and *recipes* for {*recipes, recipies, food, receipes, cooking, food recipes, recopies, allrecipes, all recipes,...*}. In addition, some tags were removed, based on signals indicating they were hard to classify. They typically were of three types: (1) Ambiguous tags such as {*a, b, c, d, anything, stuff, just stuff, emails, everything, everything else, misc,...*}, which in any case would not bring any value to users; (2) Personal names, which mean something else to each user, do not fit our global learning task and most of all would endanger privacy; (3) Offensive or inappropriate tags. Note also that we focused on a single language for consistency purposes so all non-English tags were removed as well. The cleaning and relabeling process reduced the number of tags from 6,000 to 2,000, and eliminated about 50% of the messages. Note that in a deployed product, in addition to the automated process, which is key to regular updates and applicability to other geos, we believe that quality control should also be conducted by a human editor in order to formally approve the final list of tags before exposing to users. We do not expect this process to be too tedious given the relatively short size and generality of the final list. It would also clearly respect the terms and conditions of most mail services as it would only be applied in the final stage on massively aggregated and thus anonymized data.

The description of the datasets is given in Table 1. A histogram of the number of messages in each folder for the T3 Train dataset is given in Fig. 2. In addition, the left column in Table 4 lists the most populated folders in the dataset by messages (as opposed to by users as depicted in Fig. 1).

In order to enable the second phase of the our method, we sampled two datasets of non-foldered messages. Each dataset was split into train and test sets. These datasets are described in Table 2.

Each message is represented as a set of 70 features at most, with an average of 15. A feature can be a unigram (single word) or a bigram (adjacent words) extracted after tokenizing the subject and sender fields, or a full sender domain. Thus, for a message from ancestry.com (a genealogy service), the full domain feature is *ancestry.com*, a sender feature is *ancestry*, and subject features include {*family, family tree, your family*}.

Corpus title	size before cleaning	size after cleaning
T1 Train	64,944,551	35,682,312
T1 Test	6,870,011	3,975,829
T2 Train	26,699,484	7,374,012
T2 Test	3,444,248	1,779,385
T3 Train	71,237,679	37,783,613
T3 Test	26,699,484	14,300,864
Total	199,895,457	100,896,015

Table 1: Three foldered messages datasets, gathered over 3 different periods, each split into train- and test-set

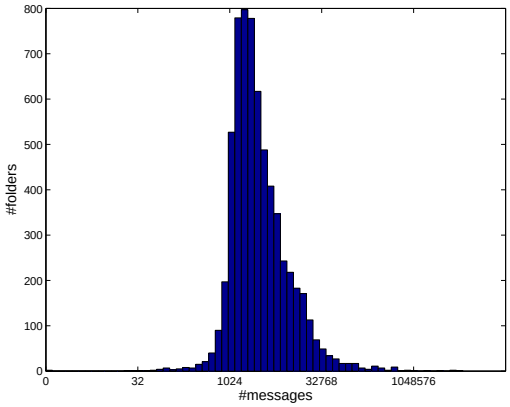

Figure 2: Histograms of number of messages per folder in the T3 train set (mode: 1,167, median: 2,438, mean: 20,773)

Feature selection was performed as follows. We computed the information gain of each feature with respect to each folder (a standard procedure in text feature selection, see e.g., [11]). Information gain tends to grow with folder size and accordingly would favor larger folders. Hence, to ensure a minimal representation for each folder, we retained the 20 features with the highest information gain per folder. Beyond this, we ranked all features by their maximal information gain value, and kept the top features. The process reduced the number of features to 2^{16}, which allowed us to represent each feature using a short integer.

6.2 Evaluating Phase 1 "tag-vs-tag" classification

Accuracy comparison: The tag-vs-tag classifier used in the first phase of our approach was evaluated on the three datasets of foldered messages. For each message, all 2000

Corpus title	size
T4 Untagged Train	8,330,355
T4 Untagged Test	8,329,492
T5 Untagged Train	20,134,623
T5 Untagged Test	20,139,110
Total	56,933,580

Table 2: Two non-foldered message datasets, gathered at different periods, each split into train- and test-set

tags were considered. We computed the accuracy rate as the fraction of messages for which we suggested the tag that was originally assigned by the user. Note that such an accuracy metric underestimates the true performance of the method. This is because a suggested tag which is different from the one the user chose might also fit the message. We compared our methods to several baseline models, which we now describe.

First, we used a nearest-neighbors approach, whose variants were used in the past for foldering messages. We view each folder as a long document containing all its messages. A standard vector space (aka tf-idf) representation is used for the resulting document corpus and for each test message. Then, we assign each message to the folder most similar to it based on cosine similarity. Results of such an approach significantly lagged behind the rest. For the T3 test set it could correctly assign 32.70% of messages into their given tags. The main issue with this naive Information Retrieval approach is that it fails to distinguish well populated folders from scarcer ones. Hence a quick fix would be to augment each cosine value with the popularity of the tag, that is, the number of messages belonging the respective folder. Our experiments showed best results when multiplying the cosine with the square root of tag popularity. Consequently, accuracy on the T3 data grew to 51.07%.

Next we applied a Naïve Bayes classifier (NB) [13], which typically performs decently in text classification and was previously used for email classification. In order for NB to produce suitable results, one needs to smooth both feature counts and folder sizes. That is, in order to estimate the probability of observing feature f at tag (or, folder) x, we count: (1) n_x – the number of messages belonging to folder x; (2) $n_{x,f}$ – the number of messages with feature f at folder x. Then we define a smoothed conditional probability

$$p(f|x) = \frac{n_{x,f} + \kappa_1}{n_x + \kappa_2} \tag{19}$$

The feature count smoother, κ_1, is standard with NB, and we set its value to 1. Of a much greater importance was the folder count smoother, κ_2, which penalizes small folders. We set its value to 10^4. The rest of NB derivation follows as usual. The fitness of tag x for message i is taken to be $p(x)\prod_{f\in F_i} p(f|x)$. The NB classifier could classify 55.47% of T3 messages into their given tags, which is an improvement over the nearest neighbors classifier.

Next we report on the accuracy of the FlatLikelihood classifier described in Sec. 4.2. Note that this classifier can be viewed as a scalable version of Logistic Regression (or the equivalent MaxEnt), allowing it to deal with thousands of labels through a latent space mapping and importance sampling. FlatLikelihood could classify 57.68% of T3 test messages to their exact given tags, exceeding the accuracy of NB.

HierLikelihood, the hierarchical version of FlatLikelihood, surprised us by meaningfully exceeding the accuracy of FlatLikelihood, thereby yielding a 61.58% accuracy rate. This shows that constraining the classifier to a strict hierarchical tag structure, did not harm but rather improved performance. The same phenomenon was observed on the T1 and T2 datasets. We speculate that the rigid hierarchical structure could serve as regularizer contributing to the generalization power of the classifier.

Finally, let us mention the normalized hinge classifiers, which yield best accuracy in our tests. The flat version (FlatNormalizedHinge) achieved an accuracy rate of 62.78%, whereas the much faster hierarchical version (HierNormalizedHinge) got a slightly lower accuracy of 62.38%. Since these margin-based classifiers share resemblance with SVM, their superior results bode well with the findings by Bekkerman et al. [4], which stated that the SVM classifier produces best results in a related email classification task.

The results we have described for T3 follow the same trend also on the T1 and T2 test sets. Full results are described in Fig. 3.

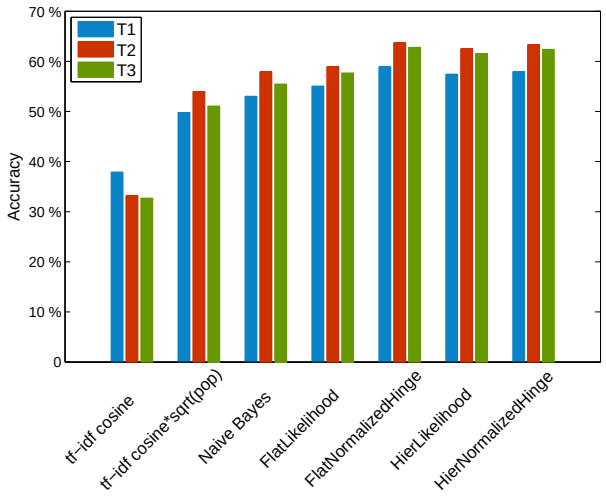

Figure 3: Accuracy of different tag-vs-tag classifiers

These classifiers can output more than a single label. Hence, we let FlatLikelihood and FlatNormalizedHinge list for each message the top-K tags of highest scores. Following the same procedure with the hierarchical classifiers would badly affect their computational speed advantage. Hence, as previously explained, we employed an approximated approach for the hierarchical classifiers. We stored for each tree leaf an ordered list of the folders most frequently leading to the leaf, and returned the top-K head elements of that list. Accordingly, recall-at-K results are depicted in Fig. 4. The approximated approach makes the hierarchical methods lag behind the flat ones as K grows. Still, for all the four methods recall@10 already exceeds 80%.

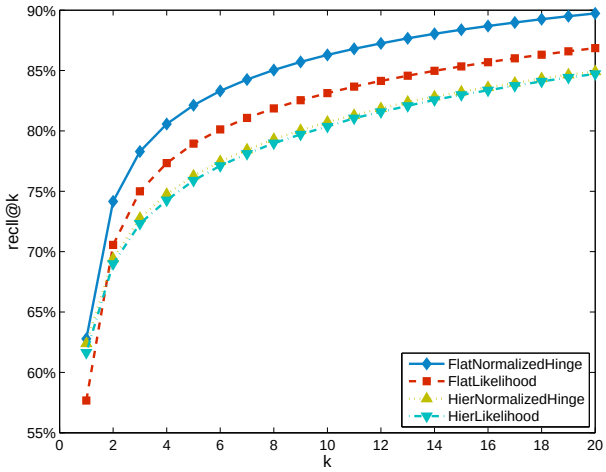

Figure 4: Recall-at-K for different tag-vs-tag classifiers on T3

Run times: We trained the classifiers with stochastic gradient descent (or ascent), while looping over the dataset with a decaying learning rate. When visiting the l-th example the decayed learning rate was set to $10^6/(5{\cdot}10^6{+}l)$. Regularization (weight decay) was found counter-productive and hence is not employed. Training stops after visiting 2^{28} examples, a point where observed accuracy of the process converges for all tried methods. We report times measured on a HP DL160 G6, 2xXeon X5650 2.67GHz system (with single threaded processes). Training times differ dramatically between the hierarchical and the flat classifiers, as depicted in Fig. 5. The fastest one, HierNormalizedHinge, takes 1017 seconds to complete training. HierLikelihood closely follows with a run time of 3440 seconds. The flat classifiers took much longer to complete: 30,679 seconds for FlatNormalizedHinge and 41,126 seconds for FlatLikelihood.

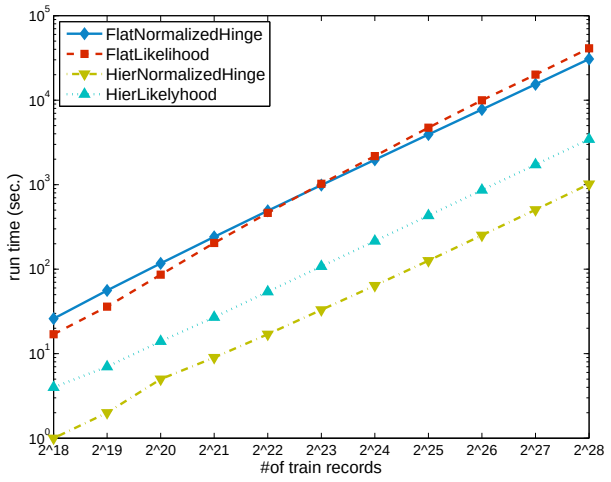

Figure 5: Training times of the tag-vs-tag classifiers

Much more important than the offline train time are the online prediction times. We measure the cumulative time required to predict the 20,056,078 records within the three test sets. Results are shown in Table 3. Here the train-free NB and the nearest neighbors classifiers, were proved to be slowest, taking 54 hours to label 1 billion messages. Assisted with their memory-local low-dimensional representation, FlatLikelihood and FlatNormalizedHinge could accomplish the same task in half the time – 27 hours, which is still way too slow for our system requirements. Note that while the FlatLikelihood and FlatNormalizedHinge differ in training, at test time they use the same linear function and hence take the same computation time. Finally, as expected, the hierarchical classifiers are significantly faster, requiring less than half an hour to label 1 billion messages. In other words, HierNormalizedHinge and HierLikelihood took less than 2 microseconds to label a message, satisfying our performance requirements. As a conclusion, HierNormalizedHinge clearly proves as our chosen method for Phase 1.

6.3 Evaluating Phase 2 "tag-vs-notag" classification

The tag-vs-notag classifier was applied to the results screened by the first phase HierNormalizedHinge. The first task at this stage consisted in evaluating the fraction of inbox messages that can be labeled with each of the tags. We followed the estimation process described in Sec. 5. The 20 most covering tags are described in the second column of Table 4.

method	prediction time	records/sec	hours/ GigaRecord
tf-idf cosine	3920 sec	5116.35	54.29
tf-idf cosine*sqrt(pop)	4642 sec	4320.57	64.29
Naive Bayes	3885 sec	5162.44	53.81
FlatLikelihood	1961 sec	10227.47	27.16
FlatNormalizedHinge	1968 sec	10191.10	27.26
HierLikelihood	35 sec	573030.8	0.48
HierNormalizedHinge	35 sec	573030.8	0.48

Table 3: Prediction running times of the various classifers on the total 20,056,078 test messages (the three test sets combined). We also report the throughput of each method in records/sec, and the time (in hours) to process 10^9 records.

Most used	Most Inbox coverage	Most suggested
facebook 13.89%	advertisement 32.19%	facebook 17.67%
groups 9.22%	facebook 26.22%	groups 13.38%
friends and family 4.55%	promotions 20.96%	stores 5.19%
ebay and paypal 4.30%	stores 15.94%	friends and family 3.19%
jobs 3.55%	groups 15.16%	jobs 1.59%
orders 3.40%	videos 13.84%	advertisement 1.17%
travel 1.96%	spam 12.16%	work 1.17%
pictures 1.95%	adult 11.62%	tagged 1.14%
recipes 1.89%	attachments 10.87%	videos 1.00%
bills 1.87%	movies 9.60%	travel 0.86%
fun 1.71%	shopping 8.19%	spiritual 0.78%
school 1.61%	stories 7.54%	ebay and paypal 0.76%
finance 1.37%	newsletters 7.38%	health and fitness 0.59%
amazon 1.23%	work 7.34%	shopping 0.55%
news 1.09%	sales 6.90%	news 0.55%
shopping 1.07%	clothes 6.60%	newsletters 0.54%
work 1.01%	friends and family 5.93%	yahoo 0.51%
stores 0.84%	array 5.33%	orders 0.49%
health and fitness 0.82%	entertainment 5.13%	friendster 0.47%
tagged 0.81%	business 5.08%	amazon 0.46%

Table 4: List of 20-top tags in terms of: (a) number of messages inserted by user population to the corresponding folders; (b) number of messages passing the tag's classifier (here shown percentages exceed 100%, as a single message can agree with multiple tags); (c) number of messages our method assigned the tag.

Note the striking difference between the tags actually employed by users (first column of the table) and the tags fitting most messages (second column of the table), which we find quite illuminating. For example, a tag like "advertisement" was found to be related to over 32% of (non-spam) inbox messages. Not surprisingly, such a tag was nowhere to be found among the top tags actually used by users. This in short tells the difference between the objectives of two phases of the system. The first phase attempts at suggesting the tags that actual users are expected to pick, and in case of multiple appropriate tags it will follow the wisdom of crowds. Hence, it will prioritize tags in a way resembling the first column in the table. On the other hand, the second phase evaluates the fit between each tag and a message, thereby favoring some rather generic tags such as those at the top of the second column of the table.

In order to assess the quality of the tag-vs-notag classifier, we computed its precision-recall curve on the T5 Untagged test set. Such a curve is necessarily approximate, where we employ the methods described in Sec. 5 for estimating precision and recall. In addition, since the second phase considers a separate classifier per tag, we averaged results over all tags. The results are depicted in Fig. 6. We decided to set the classifiers' thresholds in order to achieve an estimated

100% precision level for each of the tags. At this level, the expected average recall would be 75.5%.

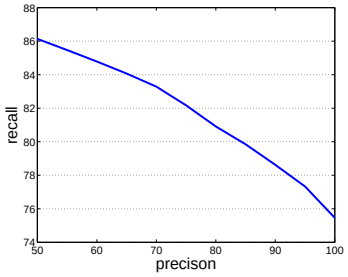

Figure 6: Precision recall curve averaged over tag-vs-notag classifiers. Since we are interested in high precision, we zoom-in on the higher precision half of the curve.

An ultimate test of our system consists of evaluating coverage as the fraction of total Inbox messages it can tag, or, the fraction of messages "surviving" the two-phase screening. When the first phase generates only the single top tag for each message, the second phase retains 67.74% of the suggested tags, leaving 67.74% of the inbox messages tagged. This fraction slowly improves when we allow the first phase suggest more tags. For example, when the first phase suggests 10 (20) tags per message, 71.26% (72.61%) of the Inbox get eventually tagged. These numbers indicate that our tagging system would be not only discoverable but clearly visible to most users. The rightmost column of Table 4 lists the tags most often suggested by the whole process on the T5 Untagged test set.

In spite of the fact that during the first phase multiple tags can be suggested, we found out that most of those besides the first one are rejected by the second phase. For example, when the first phase suggests 10 tags per message, then at the 71.26% of messages actually keeping at least one tag, an average of just 2.22 tags per message survive. Similarly, when the first phase is allowed to suggest 20 tags, then an average of 2.54 tags survive for messages actually keeping at least one tag. We view this positively, as we would not want a single message carry too many tags.

7. PERSONALIZED TAGGING

Our models, as described so far, aim at global modeling. Hence, a message is tagged regardless of the receiving user. However, at certain cases one would like to employ a more personalized tagger. For example, suggested tags can change based on localities or user demographics. In addition, we may want to promote tags already used by the user (e.g., pre-existing folders). This requires modifications to the first phase of the model. The fact that the classifiers are based on a low dimensional mapping of the messages allows their straightforward enhancement to cope with personalized tagging.

Recall that in all classifiers a message is represented as a vector q_i. We augment this vector with a user dependent vector z_u. That is, when message i is received by user u, its personalized representation would be $q_i + z_u$.

One way to define z_u, which we have not implemented yet, is as follows. Let us assume that user attributes are Boolean, given by the set $\mathrm{A}(u)$. With such Boolean attributes, one can describe gender, age group, locality, identity of existing folders, etc. We associate a distinct factor vector $y_a \in \mathbb{R}^\ell$ with each attribute, in order to describe a user through the set of attributes associated with her: $z_u = \sum_{a \in \mathrm{A}(u)} y_a$. The y_a vectors are learned during the training process in exactly the same manner the message (or feature) vectors are learned. We plan to try this approach in the near future, subject to data availability.

A simpler idea, which we indeed tried, is to define z_u in terms of already learned parameters. A user is represented through the set of messages she received. Let us denote the set of messages received by user u by $\mathrm{M}(u)$. Then the message vector q_i is replaced by

$$q_i^u = \left(q_i + \rho |\mathrm{M}(u)|^{-1} \sum_{j \in \mathrm{M}(u)} q_j \right) / (1+\rho) \qquad (20)$$

This way, all user messages are getting shrunk towards each other, by an extent controlled by the constant ρ, effectively reducing their variance. To motivate this shrinkage consider that many coexisting messages might be closely related to several tags. We strive for a consistent labeling of such similar messages. For example, it may be undesirable to assign many messages with the "stores" tag, and many other messages at the same inbox with the closely related "shopping" tag. Shrinking all messages towards each other will effectively make all decisions inter-dependent.

In order to experiment with this approach, we worked with all users receiving at least 25 messages in the T5 Untagged test set. Then, for each user we run the first phase with different values of ρ, followed by the usual run of the second phase. Results for different numbers of phase 1 suggestions are summarized in the table below.

ρ	1 - suggestion	10 suggestions	20 suggestions
0	67.74%; 9.95	71.26%; 11.20	72.61%; 11.64
0.5	66.98%; 9.38	71.03%; 10.68	72.83%; 11.23
1	61.83%; 7.16	68.16%; 8.80	71.11%; 9.57
2	53.51%; 3.80	63.43%; 6.04	68.00%; 7.13

Each entry in the table describes the fraction of inbox messages that we could tag, followed by the average number of distinct tags for a single user. We can see that when allowing phase 1 to output 20 suggestions by a non-personalized method ($\rho = 0$), 72.61% of messages get tagged yielding 11.64 different tags in an average user inbox. By increasing ρ to 2, the number of tagged messages modestly drops to 68%. At the same time the number of distinct different tags a single user receives drops to 7.13. When system designers want to limit a potential overwhelming complexity of too many co-existing tags, such a tradeoff will be desirable – significantly reducing the number of tags at the price of tagging slightly fewer messages.

8. CONCLUSIONS

We have described here a system for labeling email messages at a very large scale. The system is based on leveraging the folders of relatively few users in order to get a tagging coverage of over 70% of the Yahoo! Mail incoming messages.

Given the scale and system integration constraints we faced, we designed highly efficient classifiers that cope with thousands of considered tags per each incoming message. All our methods share the property of mapping the messages into a low dimensional latent space, thereby enabling a more compact and efficient representation. Furthermore, we borrowed two of the most successful email classification approaches (Logistic Regression and Margin-based) and showed how their training and predictions can be made faster by using sampling and hierarchical classification techniques.

The speed-up was substantial enough to satisfy our performance needs, classifying a message in less than 2 microseconds.

An end-user facing classifier, which assumes almost no user intervention or interaction, must obey very strict precision goals. Hence, after deciding on a shortlist of ranked tags per message, each suggested tag has to pass another filter deciding whether it is worth triggering or not. The major challenge here was estimating the true coverage a tag has on the population of untagged messages, which would enable us to decide at which distinct certainty level we can trigger each tag. We devised a novel method for achieving such estimations.

A classifier is as good as its underlying features. While our system inhibits us from time expensive analysis of email body, there are numerous other features that we would like to consider in the future. These include the number of recipients in To: and Cc: fields, the length of a message, the number and names of file attachments, style (html/plain) signals, and more sophisticated subject tokenization techniques.

Acknowledgments

We would like to thank the Yahoo! mail engineering team for their great help in facilitating this research and pushing it into production, with special thanks to Woody Anderson, Vishwanath Ramarao, and Andrew Sloane. In addition we thank Vinod Nair for his most valuable advice. Final thanks go to Andrei Broder for the product definition and many other insights.

9. REFERENCES

[1] D. Aberdeen, O. Pacovsky, and A. Slater. The learning behind gmail priority inbox. In *LCCC : NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds*.

[2] M. Aery and S. Chakravarthy. emailsift: Email classification based on structure and content. In *ICDM*, pages 18–25, 2005.

[3] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 1027–1035. SIAM, 2007.

[4] R. Bekkerman, A. Mccallum, and G. Huang. Automatic categorization of email into folders: Benchmark experiments on Enron and SRI corpora. In *In Technical Report, Computer Science department, IR-418*, pages 4–6.

[5] Y. Bengio and J.-S. Senécal. Quick training of probabilistic neural nets by sampling. In *Proc. 9th International Workshop on Artificial Intelligence and Statistics (AISTATS'03)*, 2003.

[6] P. Bermejo, J. A. Gámez, and J. M. Puerta. Improving the performance of naive bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. *Expert Syst. Appl.*, 38:2072–2080, March 2011.

[7] S. Chakravarthy, A. Venkatachalam, and A. Telang. A graph-based approach for multi-folder email classification. In *ICDM*, pages 78–87, 2010.

[8] A. Civan, W. Jones, P. Klasnja, and H. Bruce. Better to organize personal information by folders or by tags?: The devil is in the details. *Proc. American Society for Information Science and Technology*, 45(1):1–13, 2008.

[9] W. Cohen. Learning rules that classify e-mail. In *In Papers from the AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25. AAAI Press.

[10] Y. Diao, H. Lu, and D. Wu. A comparative study of classification based personal e-mail filtering. In *PAKDD*, pages 408–419, 2000.

[11] G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, March 2003.

[12] S. Goel, A. Broder, E. Gabrilovich, and B. Pang. Anatomy of the long tail: Ordinary people with extraordinary tastes. In *WSDM*, 2010.

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction (2nd edition)*. Springer, 2008.

[14] S. Kiritchenko and S. Matwin. Email classification with co-training. In *CASCON*, page 8, 2001.

[15] B. Klimt and Y. Yang. The Enron corpus: A new dataset for email classification research. In *In ECML*, pages 217–226, 2004.

[16] A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In *NIPS*, pages 1081–1088, 2008.

[17] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Proc. 11th International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, pages 246–252, 2005.

[18] R. B. Segal and J. O. Kephart. Incremental learning in swiftfile. In *In Proc. Seventh International Conference on Machine Learning*, pages 863–870. Morgan Kaufmann, 2000.

[19] J. Weston, S. Bengio, and N. Usunier. Large scale image annotation: Learning to rank with joint word-image embeddings. *Machine Learning Journal*, 81:21–35, 2010.