Prediction of User Intent to Reply to Incoming Emails

Andy Bromberg ABROMBERG@STANFORD.EDU
Kevin Shutzberg JKEVIN@STANFORD.EDU

Abstract

We aim to develop a practical solution to predict whether a user will reply to incoming emails by designing a model that draws on insights derived from classical machine learning algorithms and their basis in statistical methods and convex optimization. Our algorithm is optimized to handle the specific challenge of user email organization by incorporating heuristics about key behaviors and consequences.

1. Introduction

Email overload is an increasingly serious problem in today's society. As our inboxes become more and more saturated, it is becoming important to have methods and technologies in place to help us parse the mountain on email we face every morning.

So far, there have been three attempts at solutions to this general problem. First is the implementation of spam filters, which is a very well understood problem at this point (Pantel & Lin, 1998). Second is the use of software to categorize emails by topic, which ultimately doesn't answer the question of what a user should focus on (Dredze et al.). The third is the use of simple heuristics to tell the user what's important, but these typically incorporate just a couple features and often aren't personalized (Aberdeen et al.).

We aim to take steps towards solving this overarching problem by focusing on a component of it: we get a lot of email that isn't spam but still doesn't warant a reply. While we (hopefully) read most of the emails in our inbox, we only reply to a small fraction of those. In this project, we create a system and algorithms to isolate and highlight those emails so that a user can take action on what needs a response quickly and efficiently.

2. Dataset Representation

We gathered a corpus of received personal emails (from a single account) and labeled them with whether they had been replied to ($y_i = 1$) or not replied to ($y_i = 0$). We represented all the emails as feature vectors $X = \{x_i\}$ where each x_i represents one of the features we extracted, as follows:

Feature	**Representation**
Sender	Unique ID[1]
Number of words	Integer
Question marks in subject	TF-IDF[2] weight
Question marks in body	TF-IDF[2] weight
Links	TF-IDF[2] weight
Attachments	$\{0, 1\}$
Sent TO me (not cc'd)	$\{0, 1\}$

Table 1. Initial Feature Representations

We then discretized all of our features in order to allow them to work easily with our machine learning algorithms. For the features that were real-valued initially, we binned them by dividing them up into deciles, so our final features were as follows:

Feature	**Representation**
Sender	$\{0, 1, \dots n\}$[3]
Number of words	$\{0, 1, \dots, 10\}$
Question marks in subject	$\{0, 1, \dots, 10\}$
Question marks in body	$\{0, 1, \dots, 10\}$
Links	$\{0, 1, \dots, 10\}$
Attachments	$\{0, 1\}$
Sent TO me (not cc'd)	$\{0, 1\}$

Table 2. Final Feature Representations

[1]We assigned all senders from whom we hadn't previously received an email a sender ID of -1. This served to gather data about how we generally respond to emails from unknown senders with whom we have no history.

[2]TF-IDF weight is the term frequency-inverse document frequency, used to measure the abundance or scarcity of an item in a document in a corpus relative to its presence in the other documents in that corpus.

3. Methodology

In this section, we detail our benchmark classifiers and then the process by which we designed our own algorithm, as well as our general methodology. The results are presented in section 4 where we evaluate all of the algorithms' relative and absolute performance.

3.1. Features

The features we outlined in section 2 are accessible and calculable from the individual email documents we gathered from the email account. The sender IDs are assigned sequentially and chronologically (with the exception of one-off senders, as detailed above). The number of words is simply the count of words in the body of the email (and subsequently binned by deciles). The features requiring TF-IDF scores can be calculated (as in algorithm 1 and binned by deciles. The presence of attachments and the presence of the recipient in the *To:* field is extracted from the email metadata.

Algorithm 1 TF-IDF Calculation

```
Input: Corpus ($D$) of documents ($d_j$) with list ($T$) of terms ($t_i$)
for $t_i$ in $T$ do
  $idf(t_i, D) = \log(count(D)/count(D \text{ with } t_i))$
  for $d_j$ in $D$ do
    $tf(t_i, d_j) = count(t_i \text{ in } d_j)$
    $tfidf(t_i, d_j, D) = tf(t_i, d_j) \cdot idf(t_i, D)$
    return $tfidf(t_i, d_j, D)$
  end for
end for
```

3.2. Naïves Bayes Classifier

The Naïve Bayes Classifier is a probabilistic classifier predicated on Bayes' theorem. The usage of a Naïve Bayes Classifier assumes that each feature in the dataset is conditionally independent of each other feature (Mitchell, 2010). We are comfortable making this assumption because of our usage of frequencies as opposed to absolute counts. That is, if we were using absolute counts, the count of questions marks would certainly be conditionally independent on the length of the email. However, with our use of frequencies, the length of a given email should not affect the feature measuring question marks or links.

We chose to use Naïve Bayes as our baseline algorithm against which we could measure the performance of our other algorithms.

[3]Where n is the number of senders who sent mail to the inbox. This varied depending on our initial training set.

3.3. Support Vector Machine Classifier

The Support Vector Machine Classifier is a non-probabilistic linear classifier that creates the hyperplane in the feature space that best divides the training data (Cortes & Vapnik, 1995). As our data was not separable with either a linear or a Gaussian kernel, we had to use a regularized version of the SVM algorithm.

Initially, we tried an SVM algorithms with Gaussian and linear kernels to provide a baseline for our developed algorithms (which would also be based on SVMs). We used a base regularization parameter of $C = 100$ for all of our SVM-based algorithms.

3.4. Asymmetric Support Vector Machine

After getting some baseline results, we realized a key heuristic for this particular problem, which is that false positives are better than false negatives (i.e. there's low marginal cost for a user to read an irrelevant email but high marginal cost for a user to miss an important email) and as such, we should optimize for recall, possibly at the expense of precision and accuracy.

To do this, we changed the cost function of the SVM algorithm to penalize false negatives more than false positives. The original (regularized) SVM algorithm is formulated as:

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi i \\ \text{subject to} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, \ldots, m \\ & \xi_i \geq 0, i = 1, \ldots, m \end{aligned} \tag{1}$$

In its primal form, our asymmetrically regularized SVM is formulated as:

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C_{fn}\sum_{i:y^{(i)}=1} \xi_i + C_{fp}\sum_{i:y^{(i)}=-1} \xi_i \\ \text{subject to} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, i = 1, \ldots, m \\ & \xi_i \geq 0, i = 1, \ldots, m \end{aligned} \tag{2}$$

We let $C_{fn} = 10 \cdot C_{fp}$ to perform the penalization. We constrained $\bar{C} = \sum_i C_i/m = C = 100$ so that we have the same amount of total regularization as we did in our previous SVMs. The corresponding dual problem is:

$$\begin{array}{ll}
\underset{\alpha}{\text{minimize}} & \alpha^T Q \alpha - \sum_{i=1}^{m} \alpha_i \\
\text{subject to} & \alpha_i \geq 0, i = 1, \ldots, m \\
 & \alpha_i \leq C_i, i = 1, \ldots, m \\
\text{where} & Q_{ij} = y^{(i)} y^{(j)} K(x^{(i)}, x^{(j)}) \\
 & C_i = C_{fn}\{y^{(i)} = 1\} + C_{fp}\{y^{(i)} = -1\}
\end{array} \tag{3}$$

To perform our A-SVM with a linear kernel, we simply solved[3] the primal form of the problem on our dataset, but to perform the A-SVM with a Gaussian kernel, we had to use the dual form of the problem (to allow us to swap out the kernel). In practice, the dual problem was much more computationally complex, to the point where it was not practical to solve using our entire dataset, given our limited time and resources. Instead, we trained on a smaller portion of our dataset.

3.5. Reducing Feature Input Rank

We then moved to futher optimize our model by using PCA to reduce the rank of our feature inputs. This served to eliminate noise from the data as well as dramatically increase the speed of our algorithms.

Initially, the matrix describing our 7 feature dataset had a condition number of $\kappa = 15.21$. After dropping the three lowest singular values (which were roughly an order of magnitude below the others), we had a condition number of $\kappa = 2.63$. We then re-ran our A-SVM algorithms with Gaussian and linear kernels and the modified feature inputs.

4. Results

4.1. Naïve Bayes

Precision: 21.17%
Recall: 27.74%
Accuracy: 71.26%

		Actual	
		Positive	Negative
Predicted	Positive	119	443
	Negative	310	1748

Table 3. Naïve Bayes Confusion Matrix

[3]We used CVX to solve all of our convex optimization problems (Grant & Boyd, 2013).

4.2. SVM with Linear Kernel

Precision: 37.64%
Recall: 71.33%
Accuracy: 75.95%

		Actual	
		Positive	Negative
Predicted	Positive	306	507
	Negative	123	1684

Table 4. Support Vector Machine (Linear Kernel) Confusion Matrix

4.3. SVM with Gaussian Kernel

Precision: 71.06%
Recall: 38.93%
Accuracy: 87.40%

		Actual	
		Positive	Negative
Predicted	Positive	167	68
	Negative	262	2123

Table 5. Support Vector Machine (Gaussian Kernel) Confusion Matrix

4.4. Asymmetric SVM with Linear Kernel

Precision: 31.86%
Recall: 72.87%
Accuracy: 77.86%

		Actual	
		Positive	Negative
Predicted	Positive	231	494
	Negative	86	1809

Table 6. Asymmetric Support Vector Machine (Linear Kernel) Confusion Matrix

4.5. Asymmetric SVM with Gaussian Kernel

Precision: 37.97%
Recall: 58.68%
Accuracy: 83.40%

		Actual	
		Positive	Negative
Predicted	Positive	558	912
	Negative	393	5997

Table 7. Asymmetric Support Vector Machine (Gaussian Kernel) Confusion Matrix

4.6. Asymmetric SVM with Linear Kernel and Reduced Rank Feature Inputs

Precision: 33.47%
Recall: 75.08%
Accuracy: 78.93%

		Actual	
		Positive	Negative
Predicted	Positive	238	473
	Negative	79	1830

Table 8. Asymmetric Support Vector Machine (Linear Kernel, PCA) Confusion Matrix

4.7. Asymmetric SVM with Gaussian Kernel and Reduced Rank Feature Inputs

Precision: 25.22%
Recall: 71.90%
Accuracy: 70.80%

		Actual	
		Positive	Negative
Predicted	Positive	684	2028
	Negative	267	4811

Table 9. Asymmetric Support Vector Machine (Gaussian Kernel, PCA) Confusion Matrix

4.8. Overall Results

A graph of our results for all seven algorithms is in Figure 1.

5. Discussion

In the end, the value in this project came from optimizing our algorithms for our specific use case of classifying emails as to-be-replied-to or not. While the use of libraries like `scikit-learn` or `NLTK` to implement models like normal Naïve Bayes or SVM can get decent results, they are limited in that they only tackle a small set of specific objective functions. While these functions have widespread applications, in our case, we

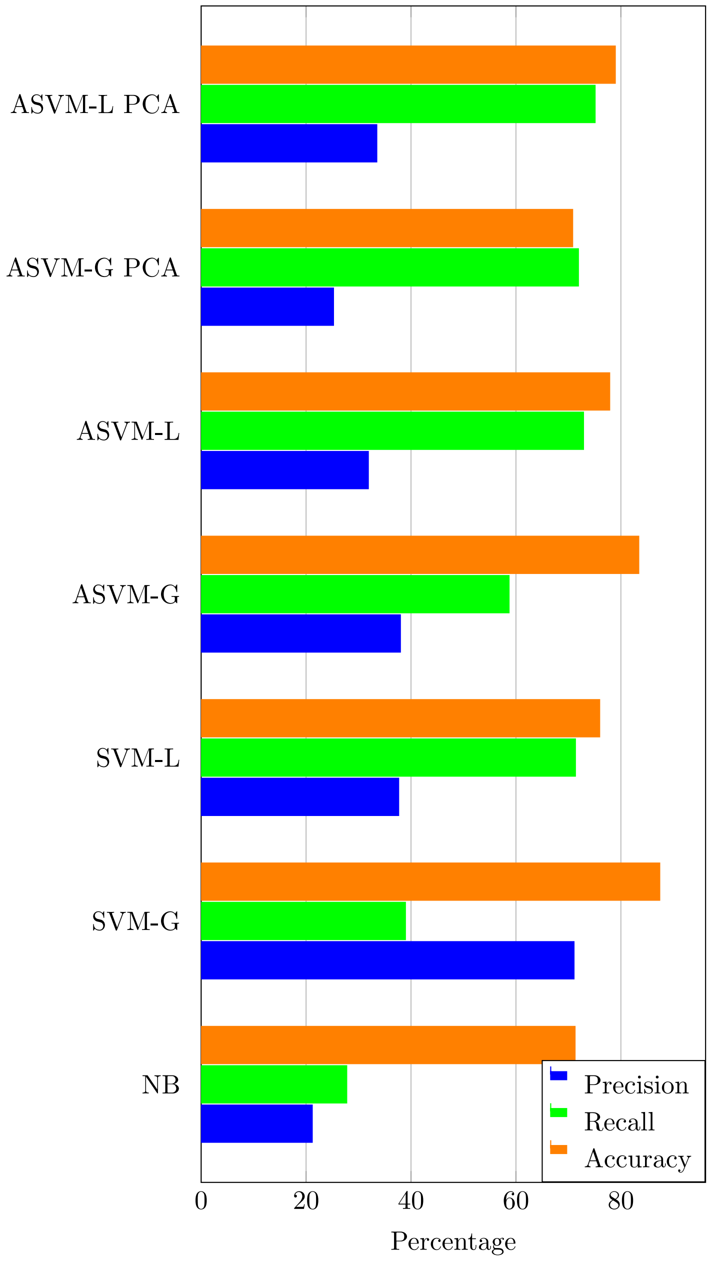

Figure 1. Results for all 7 algorithms

could dramatically improve them by specific designing an algorithm for our problem.

We crafted our own machine learning algorithms using the general Support Vector Machine paradigm and the principles of convex optimization to create an objective function matched to the task at hand. Our ability to do so came from knowledge about how the SVM algorithm functions on conceptual and mathematical levels. We were then able to implement these algorithms using CVX (Grant & Boyd, 2013).

Finally, we were able to use some unsupervised learning to condition the feature space by applying Principal Component Analysis. This marginally improved our performance on our tasks, but it also greatly improved the runtime of our algorithms due to the reduced rank of the feature inputs.

With these three steps, we hit upon mathematical and conceptual understanding of both supervised and unsupervised learning and were able to get excellent performance on our key metric (accuracy).

6. Future Work

Our primary extension to this would would be the application of asymmetric penalization (to favor false positives) to other algorithms such as Naïve Bayes. We'd love to see the results of these improvements as compared to our asymmetric Support Vector Machines.

We would also like to examine a larger dataset. This would give us a better sense for which features are commonly important. We would still segregate the larger dataset by receiver inbox, and simply train multiple instances of our model and then analyze the distribution of parameters that define our models. We could imagine using the Enron dataset for this (enr, 2009).

The heart of our algorithm for this project was our objective function. While we addressed the main issue with regard to email replies—that the marginal cost of missing an important email is much less than the marginal cost of reading an extra email—there is a lot of room to further improve the objective functions to tackle other nuances of replying to email.

Beyond just looking at the reply status of an email, we would also like to work on problems such as label or folder categorization and how to preempt that process. This has already been worked on but there are still many extensions possible to previous work (Dredze et al.; Mock).

7. Conclusion

Ultimately, we were able to attain good performance on our key metric (recall) in determining whether emails would be replied to. Our algorithm had the best performance: an asymmetrically regularized Support Vector Machine with a linear kernel and pre-processed feature inputs (which were rank-reduced by Principal Component Analysis). Our recall reached 75.08%, which is a leap beyond Naïve Bayes at 27.74%.

We consider this endeavor a success. Our models function well enough that we can imagine them being used in practice to predict whether actions would be taken on incoming emails and we were able to optimize them specifically for this application.

References

Enron email dataset, August 2009. URL `https://www.cs.cmu.edu/~enron/`.

Aberdeen, D., Pacovsky, O., and Slater, A. The learning behind gmail priority inbox. URL `http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/36955.pdf`.

Bekkerman, R., McCallum, A., and Huang, G. Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora. Technical Report IR-418, University of Massachusetts, Amherst, 2004. URL `http://management.haifa.ac.il/images/info_people/ron_bekkerman_files/email.pdf`.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

Dredze, M., Schilit, B.N., and Norvig, P. Suggesting email view filters for triage and search. URL `http://www.cs.jhu.edu/~mdredze/publications/dredze_ijcai_09.pdf`.

Grant, M. and Boyd, S. Cvx, December 2013. URL `http://cvxr.com/cvx/`.

Mitchell, Tom M. *Machine Learning*, chapter 1. McGraw Hill, 2010.

Mock, K. An experimental framework for email categorization and management. URL `http://www.math.uaa.alaska.edu/~afkjm/papers/emailcat.pdf`.

Pantel, P. and Lin, D. Spamcop: A spam classification & organization program. March 1998. URL `http://www.patrickpantel.com/download/papers/1998/aaai98.pdf`.