Email Filtering by Response Required

Christopher Knight

December 9, 2014

Abstract

This project explores the feasibility of applying machine learning to answer the following question: how likely is it that one will have to read and respond to an email that has just arrived? To answer this question, a data set was derived from several month's worth of a software engineers work email and served as the input to both a multinomial naïve Bayes classifier as well as an SVM classifier. Specifically, the project explores the details of which email features provide the most useful information in terms of predicting the output.

Neither of the two models studied was absolutely better than the other. With that being said, satisfactory prediction performance was never achieved from any of the features with either of the models due to the classification depending on far more information than what was available in the immediately preceeding email.

1 Introduction

Very frequently I am interrupted by the notification of a new email arriving. Should I break away from my current task to read it or not? Ideally, I would only accept the disturbance to my work if said email requires an urgent reply. Each of the following sections in this paper outlines the steps I took to answer this question. My goal was to model various features of my Qualcomm email data set using multinomial naïve Bayes and SVM classifiers to determine which feature is most effective at predicting whether an email requires a response from me.

In section 2, I detail the data collection process and the format/content of the resultant data set. In section 3 I discuss the two models I use and the reasons why they were chosen. In section 4, I fit the models to the data and present the results with respect to what features are the most valuable. In section 5 the results are discussed and conclusions presented. Lastly, in section 6 I discuss further directions that this project could take.

2 Source Data

Considering the data set for this problem wasn't in a usable form at the start of the project, a significant amount of time went into data extraction and manipulation. The result is a Python script that is capable of accessing email via the Microsoft Outlook MAPI COM interface. The program iterates through all the received email in the local Microsoft Outlook account within a specified time period and populates various data structures that allow easy access to the source data for the features[1] with which I am interested in experiementing. Running the script results in a feature matrix written to disk for each of the features being examined as well as the classification matrix. The classification matrix has the emails to which I replied classified as the positive class. Refer to Figure 1 for the list of top level features.

Each feature matrix contains a row for each email that I received.[2] Each column is a sub-

[1]When I say 'feature' here I am referring to an aspect of the email. Each of these features almost surely results in more than 1 column worth of data in the feature matrix itself

[2]I implicitly filtered out a large set of the negative class by only examining the subset of received email that has a reasonable chance of requiring a reply (i.e. email sent to massive email lists to which I am subscribed are not included).

component of the feature.

Raw Features	Ref. Name
'Body' term frequency	*body*
'Subject' term frequency	*subject*
'To' names	*to*
'CC' names	*cc*
'From' name	*from*

Derived Features	Ref. Name
'Body' TF-IDF	*bodytfidf*
'Subject' TF-IDF	*subjecttfidf*
My fraction of 'To'	*toprop*
My fraction of 'CC'	*ccprop*

Figure 1: Tables of the raw and derived features I used for modeling and the name by which I will refer to them.

I gathered 3 different time periods worth of data: 10 days (269 emails with 36 positives), 60 days (2207 emails with 195 positives), 120 days (4094 emails with 426 positives). All of the periods end at the same point in time. Additionally, all features that depended on indexing text fields (*subject*, *body*, *bodytfidf*, *subjecttfidf*) were treated with the Porter2 stemming algorithm before any frequencies were computed.[2]

2.1 Raw Features

The *body* feature was sourced from the **BODY** of each email. A body dictionary was constructed by splitting up the text and removing various punctuation.[3] Column i for each email is the number of times token at index i in the dictionary occured in the email. Similarly, the *subject* feature was constructed the same way from the **SUBJECT** field and had indices that referred to its own associated subject token dictionary.

The last three raw features I extracted were sourced from the **TO**, **CC** and **FROM** fields. Column i for each email in these feature matrices was a binary feature corresponding to whether or not the person at index i in the associated person dictionary was present in the respective field of the email.

[3]Note that I explicitly only looked at the body content of the top level email. Quoted material from prior emails in the thread was discarded.

2.2 Derived Features

The first two derived features that I constructed changed the feature vector from being raw frequencies of the term to be the TF-IDF metric of the term.[6] I theorized that this would improve the algorithms ability to train on the more "important" words in each email rather than the dead words.

The last two derived features capture the percentage of the email's audience made up by me. The idea is that, for example, if I'm one of only 2 people receiving the email, then I'm more likely to reply than if I were 1 of 50 people receiving the email. If I'm not explicity in the *TO* or *CC* fields, the associated *toprop* or *ccprop* feature will be zero. If I'm present in either of those fields, this feature is the inverse of the number of recipients in the field. See equation 1 below.

$$Field_{prop} = \frac{1\{\text{'Chris Knight'} \in Field\}}{|Field|} \quad (1)$$

3 Models

Multinomial naïve Bayes was selected as the initial model due to its very simple implementation, lack of complex, empirically-determined model parameters, and reasonable performance on text-classification problems. SVM was selected so as to possibly fit a far more complex decision boundary (when using a non-linear kernels), thus giving a richer model. Additionally, a large amount of email history is available and SVM is better able to take advantage of the additional data points in terms of providing better predictive accuracy.

The metrics used to explore the performance of these various features are the classification errors and the F1 score[4]. The F1 score attempts to provide a good measure of a models accuracy by taking into account the models *precision* as

[4]An F1 score of 1 indicates the model perfectly classifies the given data. The F1 score is very sensitive to the number of false positives and false negatives relative to the true number of positives and negatives, respectively, giving a powerful indicator of model success even in imbalanced data sets.

well as *recall*.[5] See equation 2.

$$F1 = 2\frac{(precision \cdot recall)}{precision + recall} \quad (2)$$

I'll be focusing almost exclusively on the F1 score as it is a better indicator of success when dealing with data sets as imbalanced as mine.

4 Results

In this section, I train multinomial naïve Bayes as well as SVM on each of the aforementioned features and present the results. For both of the below models, I ran k-fold cross validation with $k = 10$ on all 3 timeframes. The error metrics reported for each timeframe are averaged over all 10 folds.

4.1 Naïve Bayes

Multinomial naïve Bayes was run against each feature to determine each feature's value. To start off, let us only examine the *body* feature in the 120 day data set. The classification error was 9.35% on the training set and 13.96% on the test set. At first glance those values would appear reasonable, if not a bit high. However, the confusion matrices presented in Figure 2 for the same scenario tell us that our classifier was doing a mediocre job fitting the training set and a downright terrible job predicting the test set. For the rest of this paper, I will be focused on the F1 score.

Training Set	Predict = 0	Predict = 1
Truth = 0	3242.5	56.0
Truth = 1	288.2	94.3

Test Set	Predict = 0	Predict = 1
Truth = 0	347.5	19.0
Truth = 1	38.1	4.4

Figure 2: Training and testing confusion matrices that resulted from training multinomial naïve Bayes on *body*. Averaged over all runs of 10-fold CV.

The full results of the MNB runs can be seen in figure 3. MNB behaves similarly on the *body*, *subject*, and *subjecttfidf* features with the most valuable model obtained coming from either the *subjecttfidf* or *bodytfidf* features. The *to*, *cc*, *from*, *toprop*, and *ccprop* features were practically useless as they resulted in a model with training and test F1 scores below $\tilde{0}.2$ for all data sizes.

Interestingly, the only feature on which MNB was able to decently model the training set was *bodytfidf*. All the other features had F1 scores below 0.5 on the 60 and 120 day training sets. This indicates a serious bias problem with our model with respect to features other than *bodytfidf* as we can't even properly model our training data. That being said, *bodytfidf* was capable of modeling the training data, but failed to generalize and still performed poorly on the test data.

4.2 SVM

Next, I ran SVM against each feature. I used the C-SVM implementation of SVM present in LIBSVM for Matlab.[1] According to the SVM guide associated with the LIBSVM authors, the imbalance in my data set can be compensated for by using different weight parameters for each class.[3] I ran an empirical exploration of the weights for my positive class and significantly improved my results by weighting the positive class 5 times more than the negative class. That is, I used $C^+ = 5$ and $C^- = 1$ in the below formulation of C-SVM (equation 3). The full SVM results can be seen in figure 4.

$$f(x) = \frac{1}{2}w^T w + C^+ \sum_{y_i=1} \xi_i + C^- \sum_{y_i=-1} \xi_i \quad (3)$$

Overall, SVM is more versatile and is capable of obtaining at least a mediocre model on the majority of the features. The only feature that completely failed was *ccprop*. Features *body* and *bodytfidf* extremely overfit the training data and had the next most lackluster performance after *ccprop*. The rest of the features performed comparitively well with *subject*, *to*, and *subjecttfidf* being the best with obtaining stable F1 scores on the test set between 0.3 and 0.4 for both the 60 day and 120 day data sets.

5 Conclusions

The overall problem that plagued this project was the high bias error associated with the majority of the model-feature pairings which lead to significant underfitting and inability to adequately fit even the training data. Furthermore, the few models that were able to fit the training data well (MNB run on *bodytfidf* and C-SVM run on *body* and *bodytfidf*), failed to generalize and ended up significantly overfitting the training data.

I believe that these problems all stem from the fact that I didn't have what I would consider "complete" data for this problem. Specifically, the following aspects of the problem were not captured appropriately in the features: (1) Only 1 person within a role or team needs to provide a response. Responses from those "equivalent" people should be included in the positive class. (2) Responses can be sent over different media. In my job it is common to interact with someone via instant message after receiving a critical email rather than emailing a reply. Alternatively, I might just go to their office rather than sending a reply. (3) Multiple emails might be exchanged on the same thread between when an email was sent and when I send my reply. In this case, the features that triggered the reply are not in the email that occured immediately prior, but several emails ago, which is not a scenario that can be captured with my current approach.

Additionally, it's important to note that as people switch projects and change what they are working on over time, a model trained on various features 6 months ago might do a very poor job generalizing to the current email flow. This can be seen even in my data sets where some features improved performance when moving from 10 days of history to 60 days but either stayed the same or in fact got worse when moving up to 120 days of history.[5]

[5]This change could also be attributed to the curse of dimensionality as the dimensionality of the text features increased dramatically as I increased the timeframe.

6 Further Study

I think the largest gains in this problem space will come from improved feature selection that would better capture the aspects discussed above. Along another line of thought, if I had more time I would have liked to explore the possible improvements to the MNB model outline by Rennie et al.[4]

References

[1] Chang, C.-C., and Lin, C.-J. *LIBSVM: A Library for Support Vector Machines*. Department of Computer Science, National Taiwan University, 2001.

[2] Chaput, M. stemming 1.0. https://pypi.python.org/pypi/stemming/1.0, Feb. 2010. Python implementation of the porter2 stemming algorithm.

[3] Hsu, C.-W., Chang, C.-C., and Lin, C.-J. *A Practical Guide to Support Vector Classification*. Department of Computer Science, National Taiwan University, Taipei, Taiwan, 2003.

[4] Rennie, J. D. M., Shih, L., Teevan, J., and Karger, D. R. Tackling the poor assumptions of naive bayes text classifiers.

[5] Wikipedia. F1 score. http://en.wikipedia.org/wiki/F1_score. Wikipedia page for the F1 score.

[6] Wikipedia. tf-idf. http://en.wikipedia.org/wiki/Tf%E2%80%93idf. Wikipedia page for the TF-IDF metric.

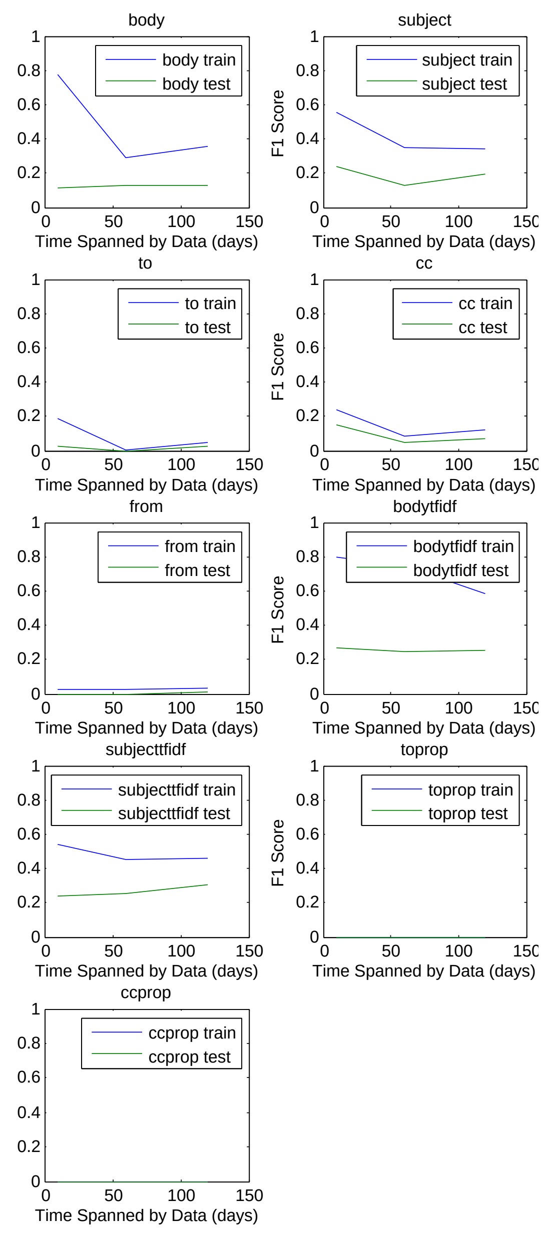

Figure 3: F1 score of MNB on each feature with no scaling of the data.

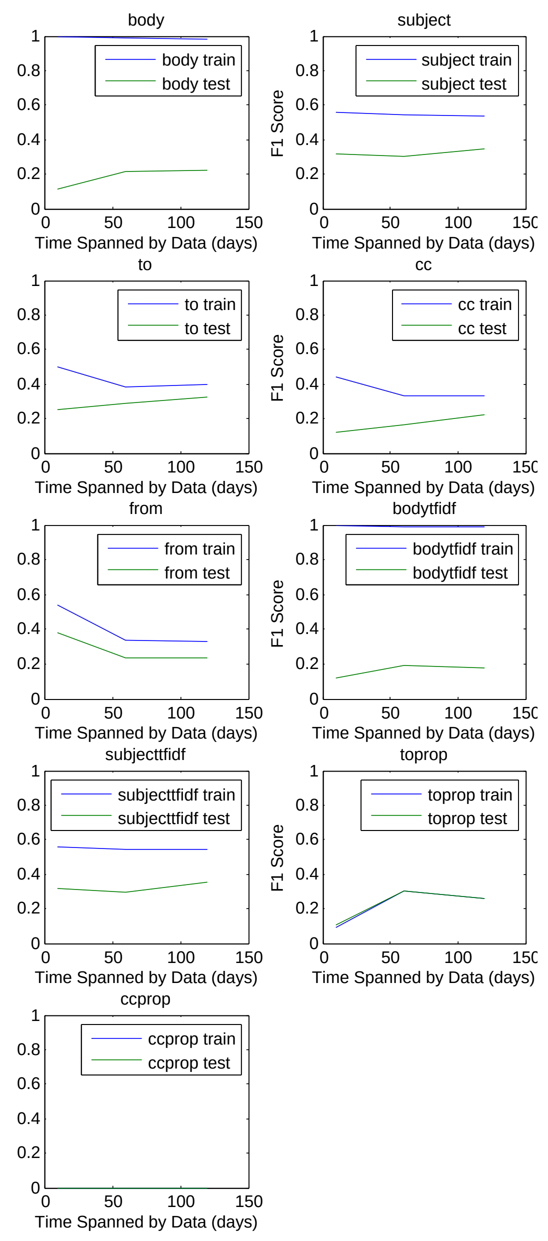

Figure 4: F1 score of C-SVM on each feature with no scaling of the data.