CS229 Final Project: Predicting Expected Email Response Times

Laura Cruz-Albrecht (lcruzalb), Kevin Khieu (kkhieu)

December 15, 2017

1 Introduction

Each day, countless emails are sent out, yet the time in which one receives a response can vary significantly. For our project, we sought to develop a model that predicts the expected response time for an email. Our research is motivated by the fact that current popular email frameworks lack any means for users to know how long it will probably take to receive a response. This is particularly important in academic and business settings, where emails are constantly being used as a form of communication.

This project aims to aid in this issue by applying general machine learning principles. In particular, we will extract features from email content, email metadata, and sender/recipient information, and provide this as input to various models - Linear regression, Logistic regression, Naive Bayes, SVM, and Neural Network - in order to predict the expected response time for a given email.

2 Related Works

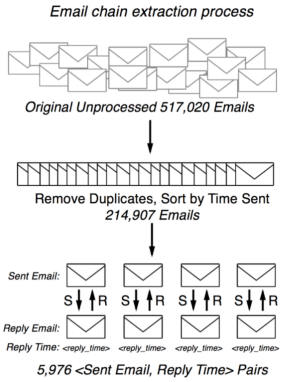
To the best of our knowledge, little prior work exists seeking to predict how long it will take for an email to receive a response. However, there is ample literature on a similar problem: whether an email will receive a reply at all, or merits being responded to. In *Applying Machine Learning Techniques for Email Reply Prediction*, Ayodele et. al. extracted features from the body and metadata of emails and then applied a scoring method to predict whether an email needed a response. In *Intelligent Email: Reply and Attachment Prediction*, Drezde et. al. worked to solve two problems, one of which was whether an email needed a response; they used a logistic regression classifier with bag of words representation of the email as well as extracted features such as sender and recipient information. Similar work was done by Drezde et. al. in *Reply Expectation Prediction for Email Management.* In *Characterizing and Predicting Enterprise Email Reply Behavior*, Yang et. al. also worked to predict whether an email merited a response, as well as how long it would take for an email to receive a response, bucketed into 3 response time buckets; using similar features such as bag of words as well as temporal and user-specific information extracted from email metadata, and models including logistic regression and AdaBoost, they attained a best of 0.72 accuracy on reply needed classification (random: 0.5), and 0.46 accuracy on reply time prediction (random: 0.33).

3 Dataset and Features

For our project we used the May 7th, 2015 version of the Enron email dataset, which is comprised of 500,000 emails generated by employees within the Enron Corporation.

3.1 Preprocessing

Before we could conduct research on email response times, we first had to preprocess the dataset to: 1) extract the email objects out of the single string that comprised each 'email' in the original dataset; and 2) harvest email `<email, response time>` pairs: emails which we identified as having received a response email, and the associated time it took for the response email to be sent. For (2), we used our chain pair extractor algorithm, and in total, we collected 5976 examples.



1. Group emails with same title into list (candidates for an email thread)
2. For each such list:
   a. sort emails by time sent
   b. For each consecutive pair of emails (e1, e2):
      i. check if e2 is a potential response to e1 (e2's recipients list includes the sender of e1 and vice versa)
      ii. if this holds for all consecutive pairs, and doesn't hold for any non-consecutive pairs (ie, (e1, e3)), then we consider each <e1, e2>, <e2, e3>... to be a chain pair
      iii. For each <ei, ej> pair we have: <sent email, response time> = <ei, ej sent time minus ei sent time>

Figure 1: Chain pair extraction approach and algorithm.

3.2 Data Characterization

We next performed some general analysis on the processed dataset. We find the histogram of emails by response time seems to follow the power law distribution, and that most emails receive a response within an hour (this imbalance will be something we will need to address during our prediction problem). Inspired by Yang et. al., we also investigated the relationship between day of week / hour of day, and response time: we find people are slower to respond on weekends and quicker to respond in the morning.

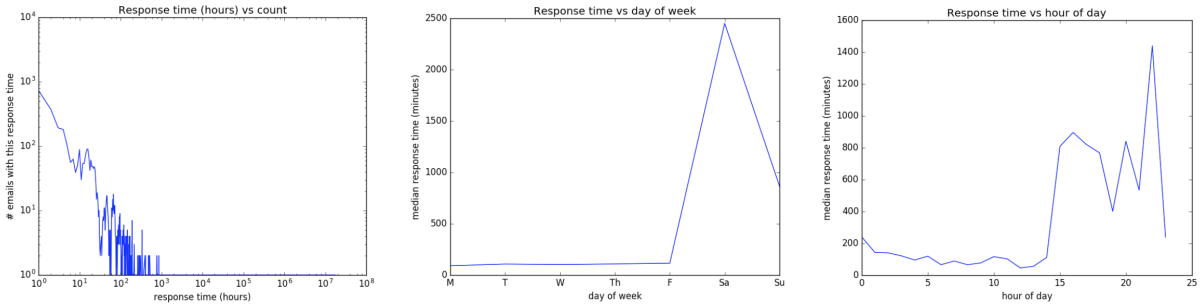


Figure 2: 1) histogram of emails in each response time category (bucketed by hour); 2) response time (minutes) vs. day of week; 3) response time (minutes) vs. hour of day.

3.3 Features

Feature extraction consisted of obtaining the following features features from the parsed dataset:

- Number of recipients in "To" field
- Number of words in email body
- Number of recipients in "CC" field
- If email is the start of a new email chain
- Time of day
- Day of week
- Number of words in subject line
- Number of '?'-marks in body / subject

In our enhanced feature set for later iterations of SVM and NN, we also appended the following:

- If question mark present in body / subject
- If keywords "response", "can", "please", "urgent", "important", "need" in body / subject

For the features involving words, we first stemmed the words using the Porter Stemmer: to do so, we leveraged source code from cs124. Additionally, for two of our machine learning experiments (Naive Bayes and Neural Network), features were created that served to indicate which words were present in the email body.

Thus, for each email we extracted a vector `<x>` of features, and had an associated label `y`. For our multiclass classification problem, `y` corresponded to the bucket 0 - 24 correlating to the response time rounded to the nearest hour. For our binary classification formulation, `y` corresponded to either 0 or 1 (response $< \frac{1}{2}$ hour or $>= \frac{1}{2}$ hour; where this cutoff was chosen as it yielded approximately equal class sizes).

4 Methods

For our problem, we applied five different machine learning models; we describe each below. The experiments and results of applying these models are then described in section 5.

4.1 Linear Regression

One model we applied towards our problem, as a baseline, was that of linear regression. In order to apply linear regression, which predicts continuous values, given that our labels were discrete values (rounded to the nearest hour), we rounded the model prediction to obtain a resulting discrete prediction. Our prediction function is: $h_\theta(x) = \theta^T x$. Linear regression seeks to then learn the parameter θ, via stochastic gradient descent, which minimizes the ordinary least-squares cost function across the training examples: $J(\theta) = \frac{1}{2}\sum_1^m (h_\theta(x^i) - y^i)^2$. To do this, we used scikit-learn's linear regression libraries.

4.2 Logistic Regression

The second model we applied was that of logistic regression, which is perhaps better suited to our prediction problem as it is one intended for classification. In logistic regression, the hypothesis function is the sigmoid function: $h_\theta(x) = \text{sigmoid}(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$. Then, gradient ascent is used to maximize the log likelihood of the labels given the training examples and learned parameters. For the multiclass version of our problem, we used the softmax function, which generalizes logistic regression, to find the predicted probability of each of the classes. We implemented this using scikit-learn's logistic regression package.

4.3 Naive Bayes

The third model we tried was the Naive Bayes Bag of Words model. To do this, we used a weighted TF-IDF version of Naive Bayes where, instead of just using word counts, we used TF-IDF scores to help determine which words were more important in determining the class of an email. To aid in this process, we leveraged scikit-learn libraries in Python.

4.4 Support Vector Machine

Another model we applied to our prediction problem was a support vector machine. The SVM seeks to define a hyperplane that separates training examples of different classes with highest accuracy while also maximizing the margin between the support vectors of different classes (where the support vectors are the vectors parallel to the hyperplane which correspond to points closest to the margin). To make a potentially non-linearly separable dataset separable by a hyperplane, SVM's will typically map the vector x to a higher dimensional space via a kernel mapping; in our case, we use the Gaussian RBF kernel, defined by $e^{-\gamma||x-z||^2}$. This was also implemented with scikit-learn's SVM package, with a specified Gaussian RBF kernel.

4.5 Neural Network

A Neural Network was the last model we used to try and tackle our problem. For this approach, we again leveraged scikit-learn's Multilayer Perceptron package to do most of the heavy lifting. We decided to use a neural network due to its ability to recognize highly complex patterns in features.

5 Experiments and Results

We performed four different experiments; two involving multiclass labels, and two with binary labels. For each experiment, we used 80% of the dataset for training / development, and withheld 20% for testing. In experiment 4, we also conduct parameter-tuning for the NN, specifically for the number of hidden neurons, number of hidden layers, and learning rate. Our performance metric for all models was accuracy (fraction of correctly classified examples), which we averaged across 10 repeated runs on random shuffles of the dataset.

5.1 Multilabel Classification: Experiments 1 and 2

Here, we used 25 labels, corresponding to email response times rounded to the nearest hour between 0 and 24. The results reported in this subsection are tested on two variations of our dataset:

- **Experiment 1**: our aggregated dataset described in section 3, excluding `<email, response time>` pairs with response times greater than 24 hours to remove outliers (this accounts for 84.6% of all pairs). This provided a training/development set of size 4,044 and test set of 1,011 emails.
- **Experiment 2**: a "balanced" dataset where we cap the number of `<email, response time>` pairs in each response time category at 50. This provided a training set of size 955 and test set of size 239.

5.1.1 Experiment 1 and 2 Results

Linear Regression: Our first baseline consisted of a Linear regression model as described in section 4, using scikit-learn's Linear regression package. In Experiment 1, we report an accuracy of 0.059, and in Experiment 2, 0.055. Thus, the model does not perform above random (this accuracy is roughly the percentage presence of each bucket), indicating the features and label do not follow a linear relationship.

Logistic Regression: We next used a logistic regression model using scikit-learn's Logistic Regression package generalized to perform Softmax regression. In Experiment 1, we obtained 0.362 prediction accuracy, but found that the most common label, 0, was always being predicted. When run on the balanced dataset instead (Experiment 2), we obtained 0.097 accuracy. Though low, on the balanced dataset the model does perform above random.

Naive Bayes: Our third approach was to use a bag of words analysis using scikit-learn's Multinomial Naive Bayes package. We created a boolean feature set for each training pair that marked which words existed in a given email's body, where the keys to this list correspond to ID's for a given word and a 1 indicates that the word exists in the email. On the aggregated dataset, we obtained 0.380 accuracy (again due to Naive Bayes always predicting 0). On the balanced data set, we obtained 0.008 accuracy, worse than random.

Neural Network: Our last experiment involved running our emails through a two hidden-layer neural network using scikit-learn's multi-layer perceptron package. In setting up our neural network, we fed in all of our features from the previous three models (the nine features + the bag of words from Naive Bayes). On the aggregated dataset, the neural network performed at 0.319 accuracy, again predicting mostly 0's. On the balanced data set, our neural network performed at 0.066.

Table 1: Multilabel classification results

(a) Experiment 1

Model	Train Acc.	Test Acc.
Linear regression	0.061	0.059
Logistic regression	0.368	0.362
Naive Bayes	0.509	0.380
Neural network	0.322	0.319

(b) Experiment 2

Model	Train Acc.	Test Acc.
Linear regression	0.055	0.055
Logistic regression	0.137	0.097
Naive Bayes	0.509	0.008
Neural network	0.100	0.066

5.2 Binary Classification: Experiments 3 and 4

We subsequently reduced our prediction problem to one of binary classification (we hypothesized that the low performance in the previous section may in part have been due to having too many labels for our models to effectively classify). We also tried using additional features to try improving performance. For this section, we binarized our original response time labels as follows: 0 if email response time $< \frac{1}{2}$ hour; 1 otherwise (we chose $\frac{1}{2}$ hour as this resulted in roughly equal class sizes; then randomly removed some examples from the second class to fully balance the classes). This provided a training set of size 2,896 and test set of size 724.

Experiment 3 In this experiment, we used the binary-labeled dataset described above, and ran the same experiments as in the multiclass experiments described previously. In addition, we also introduced the SVM

model, using scikit-learn's SVM package with a Gaussian RBF kernel. Overall, we found that linear regression and Naive Bayes performed comparably to random guessing. However, the neural network, logistic regression, and SVM did perform slightly above random: the full results are displayed in Table 3.

Experiment 4 For our final experiment, we selected the three highest-performing models from experiment 3 and added the additional features described in the end of 3.3. With this, the SVM attained an accuracy of 0.560, the NN attained 0.570, and logistic regression attained 0.576. However, SVM and NN had a train accuracy of around 0.8, indicating that those models were overfitting slightly to the train set. Thus, the additional features we chose did not seem to boost performance much. Below we also include parameter tuning results for the neural network before adding improved features.

Table 2: Neural Network Parameter Tuning Results

Hidden Layer Neurons		Number of Hidden Layers		Learning Rate	
# of Neurons	Test Accuracy	# of Layers	Test Accuracy	Rate	Test Accuracy
10	0.550	1	0.567	1E-01	0.566
20	0.563	2	0.525	1E-02	0.569
40	0.572	3	0.572	1E-03	0.569
60	0.584	4	0.570	1E-04	0.569
75	0.559	5	0.575	1E-05	0.577
90	0.527	6	0.579	1E-07	0.572
		7	0.578		

Parameter tuning results fluctated from run to run, giving us these average approximations.
Final parameters: 60 hidden layer neurons, 6 hidden layers, and learning rate of 1e-0.5

Table 3: Binary classification results

(a) Experiment 3

Model	Train Acc.	Test Acc.
Linear regression	0.536	0.526
Logistic regression	0.587	0.572
Naive Bayes	0.502	0.500
SVM	0.871	0.558
Neural network	0.777	0.590

(b) Experiment 4

Model	Train Acc.	Test Acc.
SVM	0.773	0.560
Neural network	0.828	0.570
Logistic regression	0.589	0.576

6 Conclusion

In conclusion, our investigation indicates that the problem of email response time prediction is a difficult one, in both the multi-label and binarized-label space. The low accuracies across models is understandable however: there are many social factors that cannot be detected in the metadata of an email that impact how quickly a person chooses to, or is able to, respond. However, we did find that the Neural Net, SVM, and Logisitic regression Models did perform at above-random levels, indicating that there is some degree of correlation email features and expected response time. Potential areas for future research include investigating additional features that can be used for prediction: in particular, incorporating a user's history of reply behavior, as well as the relationship between sender and recipient, could potentially improve classification.

7 References

1. Ayodele et. al. *Applying Machine Learning Techniques for Email Reply Prediction.* 2009.
2. Drezde et. al. *Intelligent Email: Reply and Attachment Prediction.* 2008.
3. Yang et. al. *Characterizing and Predicting Enterprise Email Reply Behavior.* 2017.
4. Dredze et. al. *Reply Expectation Prediction for Email Management.*

8 Contributions

- Kevin: Neural network, Naive Bayes model, feature selection, making poster, writing report.
- Laura: Dataset parsing, linear/logistic regression and SVM experiments, data analysis, writing report.