Email Classification: Spam or Ham

Evan Pfeifer, DATA 100 at UC Berkeley

November 2020

1 Introduction

In this project for DATA 100[1] at UC Berkeley, I created a logistic regression classifier that can distinguish spam (junk or commercial or bulk) emails from ham (non-spam) emails.

The goals of the project were to develop the following skills:

- Feature engineering with text data
- Using scikit-learn libraries to process data and fit models
- Validating the performance of your model and minimizing overfitting
- Generating and analyzing precision-recall curves

2 Part I: Initial Analysis

2.1 Loading the Data

The dataset consists of email messages and their labels (0 for ham, 1 for spam). For DATA 100, we were given a training dataset that contains 8348 labeled examples, and the unlabeled test set contains 1000 unlabeled examples.

	id	subject	email	spam
0	0	Subject: A&L Daily to be auctioned in bankrupt...	url: http://boingboing.net/#85534171\n date: n...	0
1	1	Subject: Wired: "Stronger ties between ISPs an...	url: http://scriptingnews.userland.com/backiss...	0
2	2	Subject: It's just too small ...	<html>\n <head>\n </head>\n <body>\n <font siz...	1
3	3	Subject: liberal defnitions\n	depends on how much over spending vs. how much...	0
4	4	Subject: RE: [ILUG] Newbie seeks advice - Suse...	hehe sorry but if you hit caps lock twice the ...	0

[1]Note that because this project is still in use for Data 100, class policy dictates that I cannot make my code publicly available. However, the code can be seen upon request by those who are interested.

Given that a part of the grading for the project was meeting a certain accuracy level on the test set, we did not have access to the labels for the test data. Instead, I split the training set once again to be 90% training and 10% validation, so that I could have a sense of model performance before making predictions on the true test set.

2.1.1 Missing Data

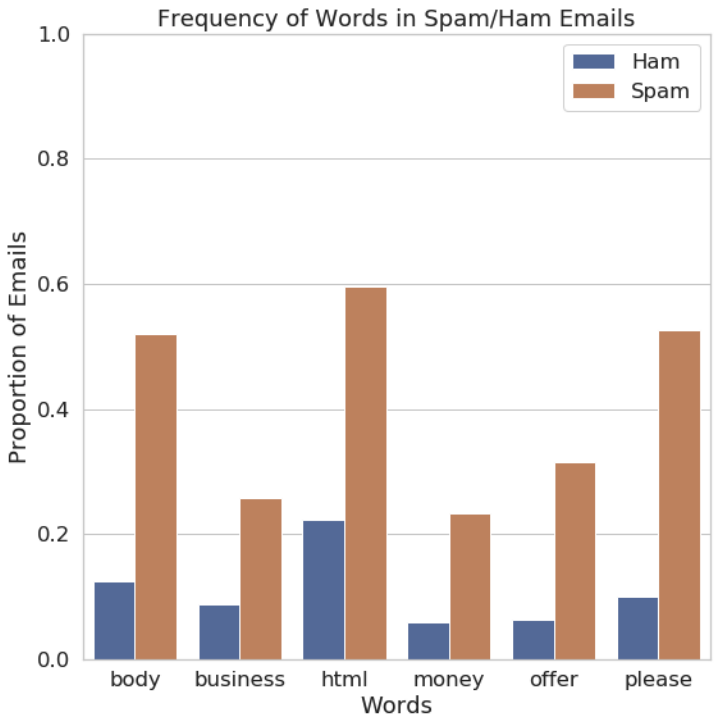
There were only 6 emails with any missing data, and all of those were only missing subject data. Missing subject lines were simply filled with empty strings.

2.2 Basic Feature Engineering

To train an logistic regression model we need a numeric feature matrix X, and a vector of corresponding binary labels y. Unfortunately, the data are text, not numbers. To address this, I created numeric features derived from the email text and use those features for logistic regression.

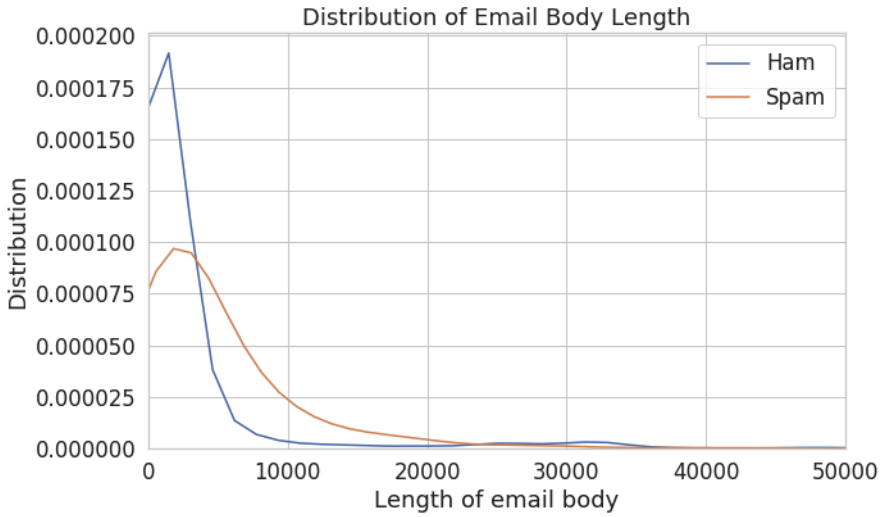
2.2.1 Basic EDA

Many of the usual suspects appear to have a higher frequency in spam emails than ham emails. I additionally included some common HTML tags, as many spam emails have raw HTML included in the body.



These words can be used as numeric features by finding whether each email contained a particular word, and assigning a value of 1 or 0 accordingly. This proves useful later on when building the logistic regression classifier.

In the class conditional density plot below, we can see that emails that are very long tend to be spam.



2.2.2 Basic Classification

I first tested two naive models. The first model was trained with the following words encoded as features: ['drug', 'bank', 'prescription', 'memo', 'private']. The training accuracy was 75.8%, but this result is less accurate than one might believe. If we have a second naive model classify every email as ham, regardless of the features, we find that it has an accuracy of 74.5%.

Presumably, our model would be used for filtering (i.e. we want to be able to filter out spam messages so they do not reach a user's inbox). Thus, there are two errors we could make:

- False positive (FP): a ham email gets flagged as spam and filtered out of the inbox.
- False negative (FN): a spam email gets mislabeled as ham and ends up in the inbox.

In particular, we should look at a few key metrics:

- Precision: measures the proportion of emails flagged as spam that are actually spam.

- Recall: measures the proportion of spam emails that were correctly flagged as spam.
- False-Alarm Rate: measures the proportion of ham emails that were incorrectly flagged as spam.

While our model trained on the five words may appear to be no better than the all-ham model, it is still preferable when we look at the aforementioned metrics:

	5-Word Model	All-Ham Model
Precision	0.642	0
Recall	0.114	0
False-alarm rate	0.022	0

We cannot even calculate the metrics for the all-ham model, as it has no true positive, and also no false positives. So, even though the training accuracy for both models was nearly the same, our 5-word model is still preferable.

3 Part II: Making an Accurate Model

3.1 Natural Language Toolkit (NLTK)

The majority of the features for the classifier were built using the NLTK library in Python. In order to find distinguishing words between spam and ham emails, I used a series of data processing steps:

1. Split training data into spam and ham sets and concatenate every email body in each set
2. Use regular expressions to only include words with English letters
3. Use NLTK to tokenize each word and remove stopwords (words with no meaning, i.e. "an," "is," "the")
4. Use NLTK to get the most common words in the spam and ham sets
5. Choose words that do not overlap with the other set to be encoded as features

I ultimately ended up using 498 words to be encoded as features. This was a much higher number than I anticipated using. I originally thought that using such a large number of words would have led to overfitting, but my validation accuracy still rose as I increased the number of words used as features, and eventually landed upon this number as being close to optimal.

3.2 Length of Email Body

In addition to the presence of certain words, I also used the length of the email body as a feature. From the class conditional density plot of the distributions of email body length, it can be seen that most emails over 5000 characters in length are spam.

3.3 Fitting the Model

I used the standard logistic regression model from scikit-learn. In addition to being easy to use, it has a built-in feature for fitting the regularization parameter, which is the only hyperparameter in logistic regression models. Scikit-learn imports were also used for testing and cross validation.

3.4 Model Accuracy

I performed many tests of the models accuracy:

- Training accuracy: 98.7%
- Cross validation accuracy (5 folds): 97.8%
- Validation set accuracy: 97.4%
- Test set accuracy: 97.3%

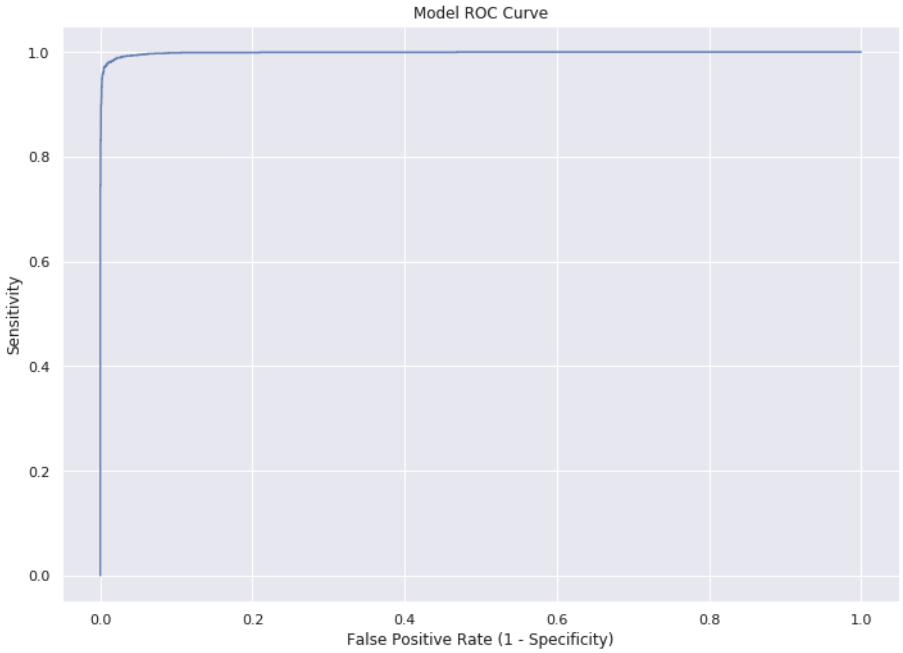
These results were even better than I had hoped. The training accuracy was very close to the cross validation score. And at a test set accuracy of 97.3%, I was well in the clear of the 88% threshold required for full points on the accuracy portion of the project.

3.5 ROC Curve

In most cases we won't be able to get 0 false positives and 0 false negatives, so we have to compromise. For example, in the case of cancer screenings, false negatives are comparatively worse than false positives — a false negative means that a patient might not discover that they have cancer until it's too late, whereas a patient can just receive another screening for a false positive.

Logistic regression calculates the probability that an example belongs to a certain class. Then, to classify an example we say that an email is spam if the classifier gives it ≥ 0.5 probability of being spam. However, we can adjust that cutoff: we can say that an email is spam only if our classifier gives it ≥ 0.7 probability of being spam, for example. This is how we can trade off false positives and false negatives.

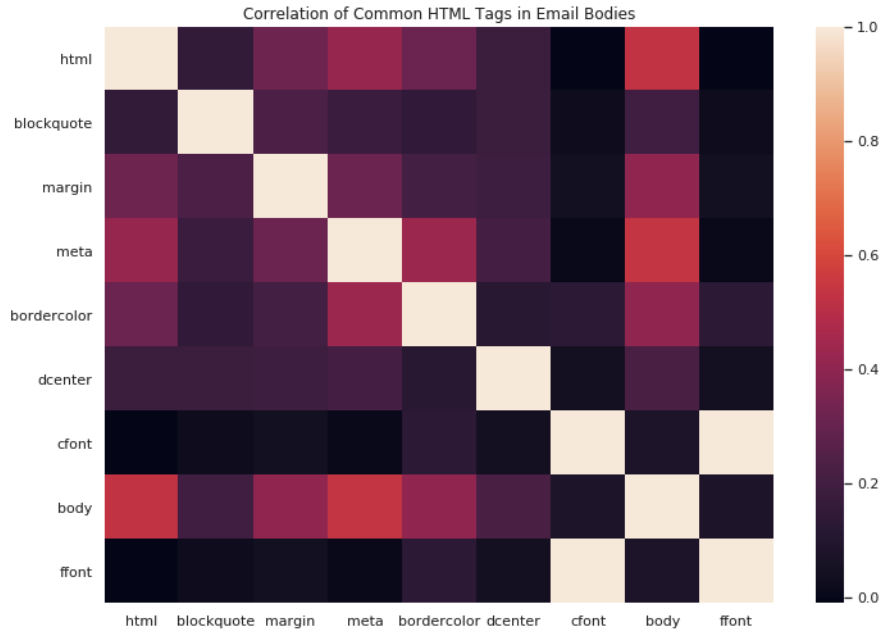
The ROC curve below shows this trade off for each possible cutoff probability, and shows promising results.



Additionally, we can find the ROC area-under-curve (AUC) score, which is 0.9987. This means that 99.87% of the time, the model is more likely to classify a spam email as ham than a ham email as spam. This is a good result, because it is much worse to have a real email get sent to the spam folder than it is for the occasional spam email to slip into a user's inbox.

4 Thoughts for the Future

While this project went well in many ways, and the model accuracy was quite high, there is always room for improvement. In particular, I am quite certain that there are redundant features being used. See the heatmap below for an example of how certain words currently being used in the model correlate quite well with each other.



In addition to removing redundancies, I could have explored making different features, such as punctuation use or the number of capital letters used. Perhaps I could extract out the text from the HTML tags to help find better words. Or, match HTML tags themselves, or even some combination of the two. This said, I am still pleased with the performance of the model I made, and definitely grew my skills in data processing, model creation and tuning, and the general data science lifecycle.