Machine Learning Algorithms for Email Spam Classification

Brian Blakely

1. Introduction

The objective of this project is to select an appropriate machine learning algorithm for the classification of spam emails. The algorithm is trained on 3000 examples and at the end is tested on 1500 held out examples. It's important to note that the data used for training and testing is already pre-processed to some extent. The classification of not spam emails is denoted by a 0 while the spam emails are denoted by a 1. The examples have already been pre-processed into a set of 30 features and their corresponding classification. Each feature corresponds to a word and denotes the proportion of all words in the email that match the given word. Meaning the algorithm will not be able to take raw emails and automatically select the best features to classify spam emails. It expects this work to already be done.

There are two metrics the algorithm is evaluated on; the accuracy and the percent of correctly classified spam at a one percent false positive rate (i.e. TPR at 1% FPR). The goal is to maximize both of these metrics, the optimization of both of these metrics will be referred to as performance. Meaning to increase the performance of an algorithm, both of the metrics either increase or increase marginally enough to outweigh the others decrease. This paper will detail the algorithm selection process and analysis, how data pre-processing affects models, the final algorithm selection for deployment, and deployment suggestion notes.

2. Data Pre-processing

Most pre-processing procedures are algorithm specific. Meaning there won't be one generic procedure that will be optimal for all the algorithms. However, there is one procedure that slightly increases the performance of all the algorithms tested. It's based on the principle that the selected dictionary of words (the set of all features) is not perfect. There are some emails that do not include any of the defined spam words, but is still spam. This is dangerous to the algorithm as it adds ambiguity to the classification, since there is no longer only clear cut cases where an email should be not spam. Obviously, the algorithm should classify an email as not spam if no spam words are present in the email. This problem is out of the scope of the algorithm as it only

considers a preselected dictionary of words. Thus, there are two solutions to the case where all features are 0 and the target is 1 :

- Just omit those cases from the training data
- Forcefully change the targets from 1 to 0

Solution 2 ended up giving better results since it isn't excluding training data. Solution 2 was used for evaluation of all models, despite it probably being bad practice to directly change raw data. Solution 2 gave roughly the same accuracy for most models, however it showed 2-3% improvements in the mean TPR at 1% FPR. Thus making it a clear choice to use.

Most pre-processing procedures ended up hurting the overall performance of most algorithms. Standard normalization packages never helped the performance and feature selection seems to be hit or miss. The feature selection techniques attempted were Lasso and RFE. Despite feature selection sometimes showing better performance in some cases, it would only be 0.1-0.2 % better than not using it. Whereas, in the worst case scenarios it would decrease performance by 2-3 % on both metrics. In the end, it gave more consistent results to not remove any of the features.

3. Algorithm Selection

The goal of this section is to find the best machine learning algorithm and the best parameters for it. The algorithm selection process involved testing many different models and comparing their evaluation metrics. At first models were chosen mostly at random in order to gauge the general performance of generic models. To evaluate the models, all of the 3000 test examples were randomly shuffled then split into a fraction of train and test examples. The accuracy and TPR at 1% FPR was calculated and recorded at each shuffle. This process was repeated k times. Typically k=20 for quick and efficient algorithms, whereas k=5 for complex and slower algorithms. Then the mean was computed and used as the evaluation of the algorithm. The train to test ratio for evaluation was ⅔ train: ⅓ test. This split was chosen since there is only a set of 3000 examples usable and it will be given a final test on a set of 1500 held out examples. Therefore, using the mentioned split on the current 3000 will result in the training data being twice as large as the test data. Which matches the split ratio for the final training.

The first algorithm tested was a support vector machine because they were used for this exact purpose in the early twenty-first century. Using Sklearn's SVM implementation with radial basis function as the kernel function. However, the best accuracy obtainable was around 89% after finding decent gamma and c variables for the function. RFE slightly increased performance on this algorithm and it was used to achieve the highest accuracy.

Multinomial naïve Bayes' was cited as being a great classifier for word distributions in many papers, however the attempts at implementation did not yield meritable results. Perhaps the information became outdated, as this field progresses fairly quickly.

Neural networks is a top candidate for email spam classification. They are very flexible and can learn many obscure patterns in data, however that strength can turn into a detriment. The idea behind picking a neural network as the spam classifier was simply based on the knowledge that Gmail's spam filter uses a neural network, TensorFlow, and their spam classification accuracy is outstanding. However, this led to the realization that the hardest part of email spam filtering is finding the best features and dynamically updating them.
First the generic mlp classifier from Sklearn was implemented for a neural network. This package gave very promising results while using hidden layer size of [100,100] and solver= adam. While testing on the first 1500 set of training examples, the best test set accuracy obtained was 97.657% with 70.930% mean TPR at 1% FPR. However, testing the algorithm on another set of 1500 held out examples showed that it was horrifically overfitting. The test set accuracy on the held out examples was 88% with 24% mean TPR at 1% FPR. The keras package from Google performed better on the held out data than Sklearn's package, however implementing keras efficiently into the algorithm was a task too ambitious.

The last algorithm evaluated was random forest classification. In short, this algorithm is just an ensemble of decision trees used to classify based on voting. The algorithm has slightly lower accuracy than neural networks, but had better TPR at 1% FPR and it had a better mean with various splits of the data compared to neural networks. Neural networks seemed to be really strong in some splits of the data, however it tended to overfit and underfit depending on the splits. Random forest classification was much more consistent. The next section will go into further detail about random forest classification and its implementation.

4. The Selected Algorithm

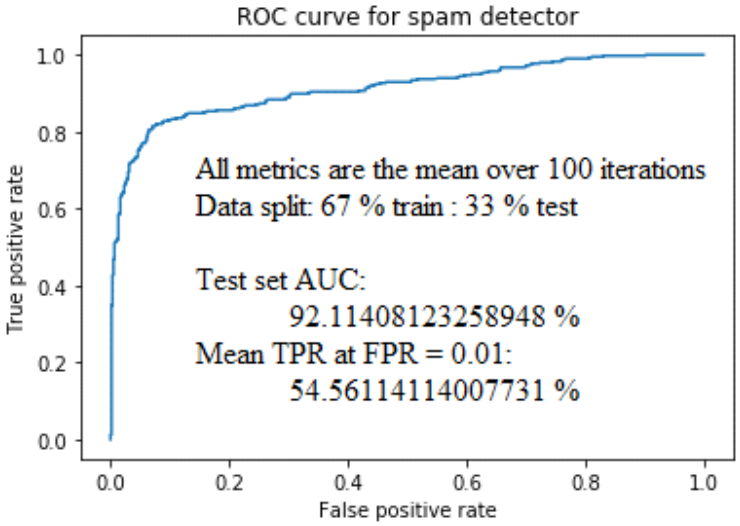
The selected algorithm ended up being a random forest classification. A nice property of random forests is that it does not make any assumptions about data distribution, thus it can handle skewed data better than other algorithms. Another reason this algorithm was chosen is because it scales well with complexity of data and its size. Implementing this algorithm meant using the appropriate parameters to help prevent overfitting while still maintaining good accuracy and a good TPR at 1% FPR. The metrics of the algorithm over 100 random different splits of size 2000 train to 1000 test is shown in figure 1. Sklearns default random forest package is used in the implementation of the algorithm with the following parameters:

- n_estimators =400
- max_depth=13
- random_state=1

These parameters are crucial to keeping both metrics relatively high. Increasing the number of estimators yields higher and more stable results. Having a high number of estimators helps prevent overfitting, however it only helps to a certain degree until it starts to decrease performance. The optimal number of estimators and max depth were chosen strictly from brute

force on the average of 20 unique random splits of data. The preprocessing the final model employs is solution 2, which is detailed in the Data Pre-processing section.

Fig. 1 TPR vs FPR of Random Forest Classifier



5. Deployment Notes

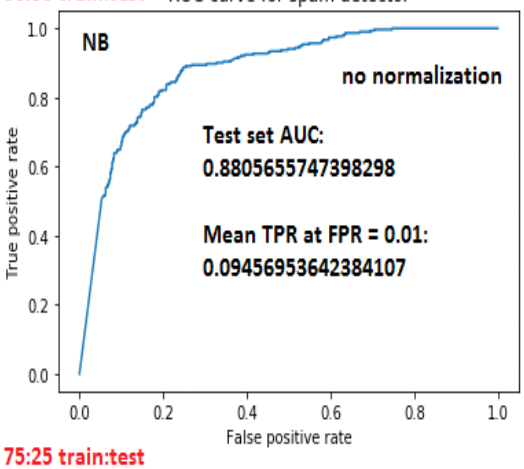
In the actual deployment of this algorithm it would be a good idea to let users be able to manually mark certain emails as spam or not spam. This is possible to achieve with a random forest classifier using incremental learning. This would let the algorithm train on emails that the user manually changes the target of then delete the email for privacy sakes. It can be tricky to implement. If space isn't an issue or if the total number of users is small then incorrectly classified emails can get appended to the training data and then have the algorithm retrain after a threshold of emails incorrectly marked gets met. For the case where all the features of an email is zero and it manually flagged as spam by the user then solution 2 wont work for retraining. This means in order to solve this issue on a higher level, the email might need to get stored until a certain threshold of emails is collected. This gives a collection of spam emails that all got through the defined spam words. One can then re-evaluate the email text data and find more features to include then append that to the existing dictionary. Setting this up in a robust fashion will allow for the spam filter to dynamically keep up with spammers as they learn what words cause their emails to get marked as spam.

In practice it would be good to evaluate the performance of this algorithm based on its TPR at 1% FPR as it's the most impactful for users. Since the algorithm could have a really high accuracy, however it could be very poor at one percent false positive rate. It's better to let a few spam emails go into the mailbox rather than a few real emails go into the spam box.

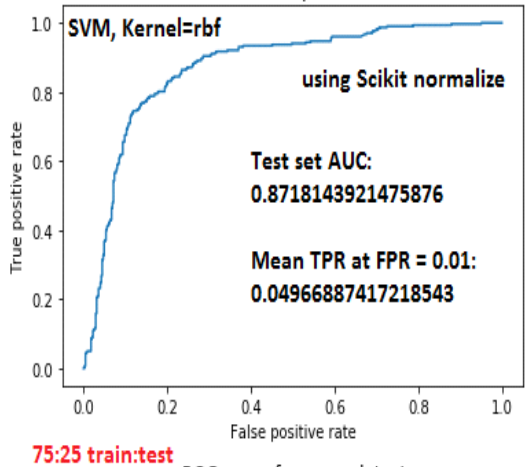
Appendix

The following image is just a compilation of graphs put together during the algorithm selection phase. This isn't necessary to read, it is just here for insight on algorithm performance. These graphs are only evaluated on the first 1500 training data, not the total 3000. The top left of the graph indicates the algorithm and some parameters, whereas the red indicates the training and test split of the 1500 examples. Best shuffleIndex denotes the best results obtained from a random shuffle of the data. The image is quite large and awkward to fit onto a single page.

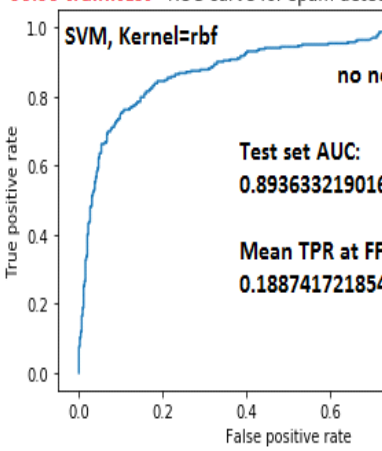
50:50 train:test



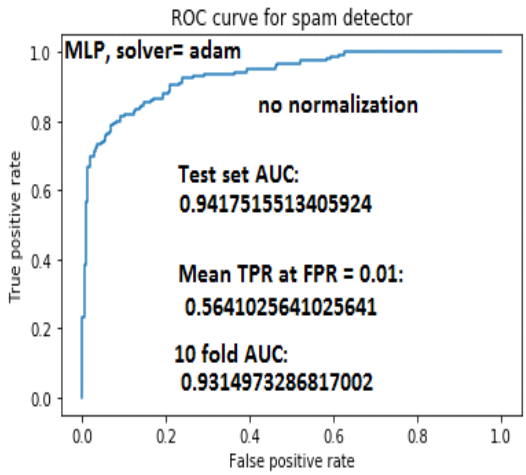
50:50 train:test



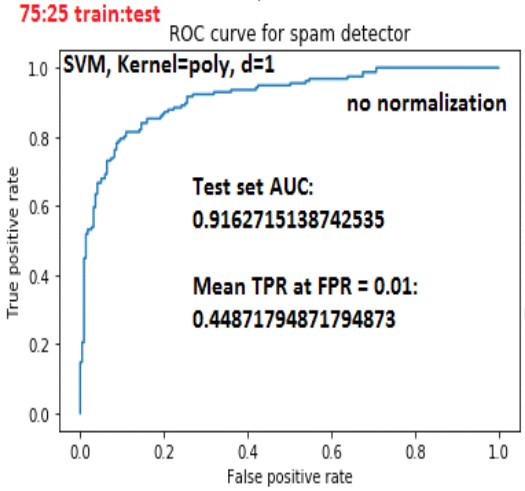
50:50 train:test



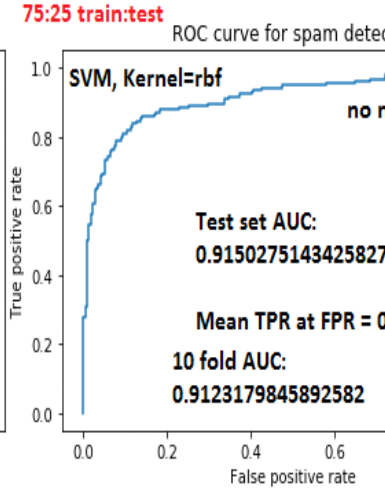
75:25 train:test



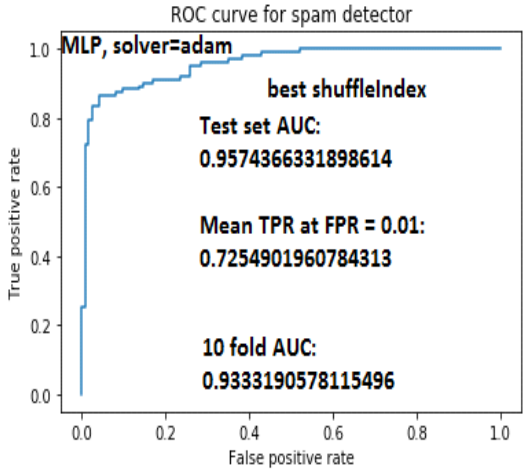
75:25 train:test



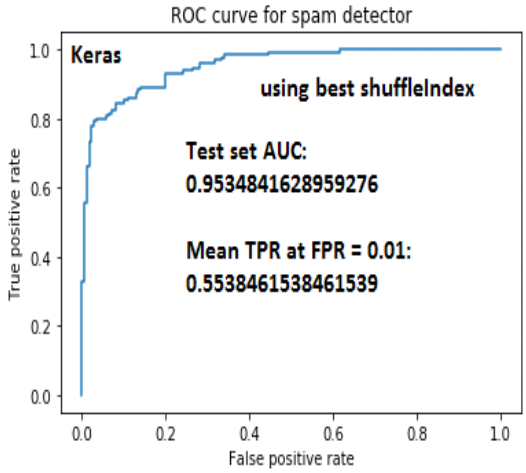
75:25 train:test



85:15 train:test



80:20 train:test



85:15 train:test