Spam Email Detection

Final Report

DS 1001 – Introduction to Data Science

Project Team Members:

Binqian Zeng (bz866)
Jui-Ting Hsu (jh3572)
Liwei Song (ls4408)
Zhaopeng Liu (zl1732)

Dec 9th, 2016

Table of Contents

1 Business Understanding

1.1 Business Problem

The business problem of this project is to alleviate a common problem with modern-day communication: spam e-mails. Spam e-mails consists of unwanted promotions, useless advertisement, and, in some cases, frauds. The objective of the project is to create a model that is able to process email messages and attempt to separate the spam emails from the normal emails (the 'hams') to improve the experience of e-mail users.

1.2 Data Mining Solution

The nature of this problem allows us to structure it as a data mining problem. Since there are labeled data, which in this scenario are emails labeled as spam or non-spam, we can structure this as a supervised learning problem. Furthermore, there are two outcomes that we are trying to predict, thus making this a binary classification problem.

2 Data Understanding

As described in the previous section, the data that will support data mining to address the business problem of interest are labeled emails. Each instance of the data is a single email containing the text of the body and an indicator stating if this email is a spam email. In our analysis, we obtained this data from the Enron-spam dataset.[1]

[1] V. Metsis, I. A. (2006, 6 19). Retrieved from http://www.aueb.gr/users/ion/data/enron-spam/

The dataset contains 6 folders. Each folder is separated into a ‘ham’ directory and a ‘spam’ directory, with a total of 33738 instances, including 16556 negative instances (ham emails) and 17182 positive instances (spam emails). In each instance the first line contains the subject of the email and the subsequent lines contains the body of the email. Since this is a binary classification, it is also important to look at the distribution of the label of the instances. There might be problems if there are significant skews in the distribution of the labels. In other words, the amount of spam and hams in the data should be on the same magnitude. In this case, the distribution between spam email and ham email are roughly evenly distributed, with 49% of the instance being negative and 51% being positive.

3 Data Preparation

3.1 Count Vectorizer

Binary count vectorizer has been widely used as it intuitively obtains features by tokenizing words. All terms or some portions of terms across all email can be combined and converted to a feature name dictionary. For each email instance, if a certain word exists in the content, the corresponding feature will have a value of 1; otherwise, it would have a value of 0. Therefore, each mail could be converted into a vector of 1s and 0s. In the actual implementation, a sparse matrix object, which only keeps the non-zero fields, is used for memory efficiency.

Two methods have been used to extract features. One intuitive method is to limit the maximum number of features (max_features) allowed in the feature name dictionary. The most frequent words are given priorities. To get a good value for max_features, we compared the AUC of a default logistic regression model using different values for max_features. According to the plot below, setting max_feature to around 750 would yield the best performance.

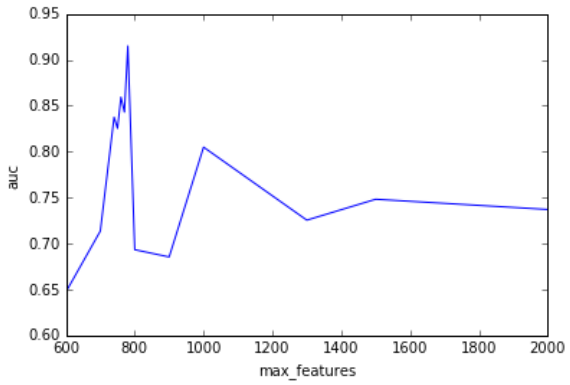

Figure 1. Finding optimal binary feature dictionary using max_features

Another method to extract features is called chi-square (Chi-2)[2], which choose features based the correlations of individual feature with the target variable. Features with higher correlation with the target variable is given priority. We performed a similar test to obtain the best *k* for the SelectKBest function. We found the optimal number of features to 101.

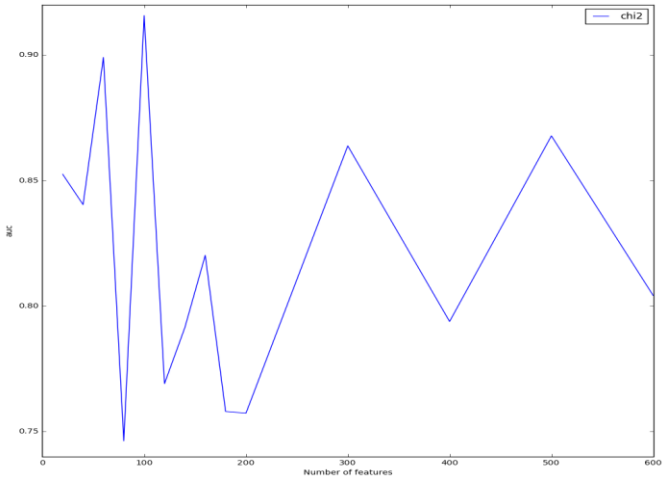

Figure 2. Finding optimal binary feature dictionary using Chi-2

[2] "Feature SelectionChi2 Feature Selection." *Feature SelectionChi2 Feature Selection*. Cambridge University Press, n.d. Web. 09 Dec. 2016.

3.2 TF-IDF Features

TF-IDF (Term Frequency – Inverse Document Frequency) has a similar word dictionary to token count, expect it added a penalty term, IDF, which penalized general term. In other words, the more frequently it appears across emails, the lower TF-IDF scores it will get. For each email instance, each feature value is a TF-IDF score instead of 1s and 0s.

A process similar to that of Count Vectorizer is performed to find the best TF-IDF feature dictionary. For max_features, the optimal value is around 400.

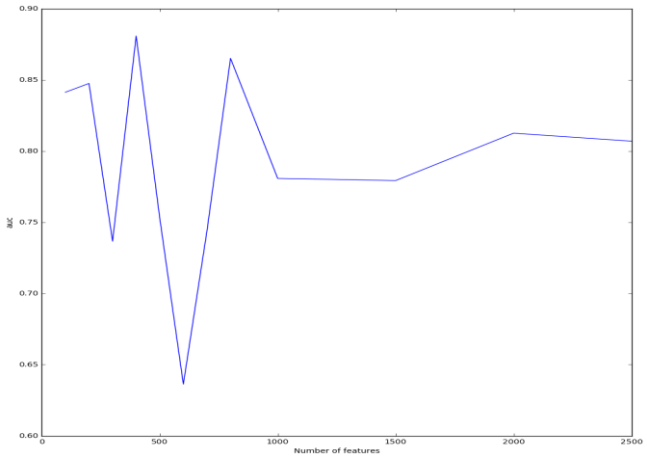

Figure 3. Finding optimal TF-IDF feature dictionary using max_features.

Similarly, we will also apply the chi-square(Chi-2) method to select features. We found the best *k* to be 187.

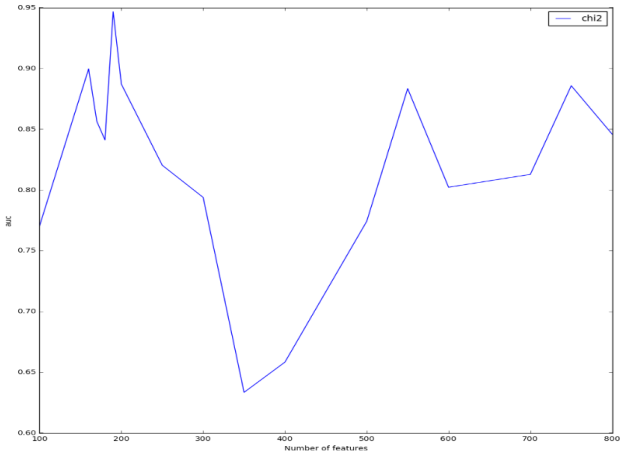

Figure 4.Finding optimal TF-IDF feature dictionary using Chi-2

3.3 Word2Vec

According to DeepLearning4J[3], Word2Vec[4] is a two-layer shallow neural net model that processes text. In a broad sense, Word2Vec transforms text data in the form of words into vector representations, or word embedding. It is similar to the auto-encoder in deep learning, but different in that Word2Vec trains words against other neighbor words from the input corpus. It constructs a vocabulary from the input data and learns the vector representation of words. The resulting word vector can then be used as the features in machine learning model training. The trained model can also be used to assess the similarity between words based on their semantic, as shown in the example from our trained Word2Vec model in figure 1.

```
w2vmodel.most_similar('money')
```

```
[('fund', 0.6492486000061035),
 ('funds', 0.5927374362945557),
 ('deposit', 0.515644907951355),
 ('dollars', 0.48747366666793823),
 ('country', 0.483626127243042),
 ('sum', 0.4824446439743042),
 ('beneficiary', 0.4502837359905243),
 ('consignment', 0.4408804178237915),
 ('abroad', 0.43630555272102356),
 ('cash', 0.4347754120826721)]
```

Figure 5. Word2Vec model demonstrating similarity.

After various experiments with the parameters and assessment using the Word2Vec accuracy metric, we ran our Word2Vec model with the following settings: 400-dimension word vectors, 20 minimum word count, 10 for context window size, and 0.001 for the down-sampling setting of frequent words.

[3] Gibson, Chris Nicholson Adam. "Word2Vec." *Word2vec: Neural Word Embeddings in Java - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM*. N.p., n.d. Web. 09 Dec. 2016. <https://deeplearning4j.org/word2vec>.
[4] "Word2Vec." *Word2Vec*. Google, n.d. Web. 09 Dec. 2016. <https://code.google.com/archive/p/word2vec/>.

4 Modeling & Evaluation

4.1 Algorithms

For the purpose of this project, we mainly proposed the following machine learning algorithms to work with:

- **Naïve Bayes**: It is one of the most common simple model used in text processing. It is also fast and scalable, and can handle sparse data, which fits our need since the processed text data are often represented in sparse matrices and are big. In our case, a Bernoulli Naïve Bayes would work well because the two classes are roughly equal in terms of presence.
- **Logistic Regression & Decision Tree**: Since we are dealing with classification problems, these two algorithms are the most common and interpretable models for classifications. Both of these algorithms also runs a lot faster than other more complex algorithms (such as random forest and support vector machines). However, these two algorithms might not provide a high accuracy as we would desire.
- **Random Forest**: Random forest is an ensemble learning method consisting of many decision trees. It yields better performance in terms of accuracy comparing with decision trees, but takes much longer time to model and it's almost uninterpretable by human. It is also prone to overfitting due to its complexity, but since we are working with a large amount of features in text mining, this problem would be alleviated.

4.2 Evaluation Metric

For the purpose of this problem, we preferred the use of AUC and the ROC curve over the other two options. The main reason for this decision is that accuracy and F-score both give equal weightings to the false positive rate and false negative rate of the prediction without further

information regarding these rates. This is definitely not the case in spam detection. The penalty of moving a regular email to the spam mailbox should be much more severe than moving a spam email to the regular inbox of a user. When classifying a regular email as a spam email (i.e. a false positive instance), we are introducing the risk of causing the user to miss information that might be important to him/her; whereas when classifying a spam email as a regular email (i.e. a false negative instance), the result is mere annoyance to the user, but no actual harm is done.

Furthermore, we would introduce cost/benefit information to the decision process when deciding the threshold of classification. As mentioned earlier, the penalty between a false positive and a false negative is not of equal weighting, in that a false positive is much more severe than a false negative. Therefore, rather than finding the best threshold by considering lowering FPR and FNR equally, we should favor towards lowering FPR by a certain factor. Involving cost/benefit information would serve this purpose. Show below is the cost/benefit matrix we are using for our decision process, with positive numbers being benefits and negative numbers representing costs:

	Predict Spam	Predict Ham
Spam Email	$V_{TP} = 10$	$V_{FN} = -10$
Ham Email	$V_{FP} = -100$	$V_{TN} = 10$

Table 1. Cost/benefits matrix.

For each FPR (false positive rate) and TPR (true positive rate) generated from the ROC curve, we will use the following formula to obtain the score of the model's performance:

$$Score = FPR \cdot V_{FP} + (1 - FPR) \cdot V_{TN} + TPR \cdot V_{TP} + (1 - TPR) \cdot V_{FN}$$

Equation 1. Calculating model performance score.

This score metric will be the evaluation metric used to choose the best final model. Under this metric, the maximum possible score is 20, with true positive rates and true negative rates of 1, and false positive and negative rates of 0.

4.3 Evaluation Framework

Since we have not only multiple algorithms, but also multiple data preprocessing technique, our evaluation framework has one more layer than the typical evaluation framework. We divided the task into three approaches, mainly different in their methods of preprocessing the text data: count vectorizer, TF-IDF, and Word2Vec. Upon our baseline with no spam filter, we will try out other algorithms to train our model. While tuning our parameter, we would assess the AUC of our model, and obtain the designated score mentioned in the previous section. This score will then be used to assert improvements of model performance.

4.4 Baseline Model

The simplest baseline model which makes sense is basically a system with no filter at all. In other words, we would classify all the emails as hams. The accuracy of such a "model" will be the distribution of spam and ham, which, in this case, around 0.5. There will be no false positive and no true positive instances. Half of the instances will be true negatives, while the other half would be false negatives. According to our evaluation metric, this model would achieve a score of 0. While it may sound trivial, this is actually a reasonable baseline, mostly attributed to the heavy penalty on false positives. In other words, a model must have a true positive rate that is at least ten times higher than its false positive rate to surpass this naïve baseline model.

4.5 Results

4.5.1 Count Vectorizer

For count vectorizer, features from both max_features and Chi-2 methods were used to fit on the four algorithms mentioned. Using features from the max_features method, we tuned the parameters for each of the models. The performances of the tuned models are shown in the plots below. Among four models, logistic regression with C = 0.1 with threshold 0.0857 obtained the highest score of **2.93**, slightly better than the baseline model.

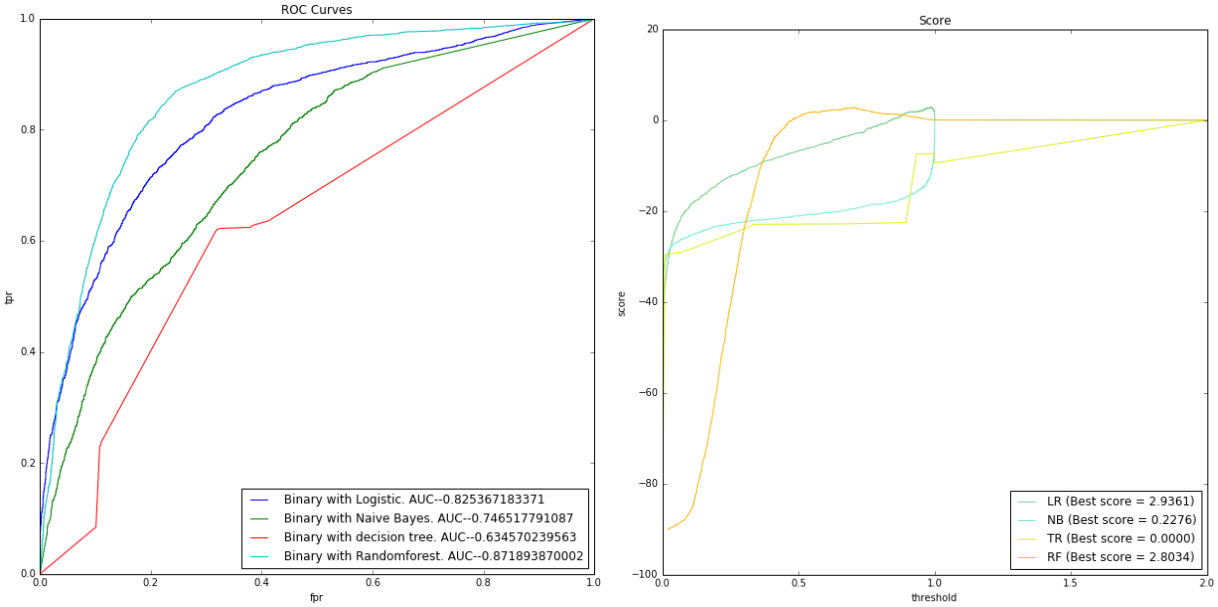

Figure 6. Performance of models on binary count with max features.

We then tried modeling on the feature dictionary from the Chi-2 selection method. Among the four tuned models, shown in the plots below, the Bernoulli Naïve Bayes model with alpha = 0.1 and threshold at 0.5 model obtained the highest score of **5.4939**, much better than the baseline model.

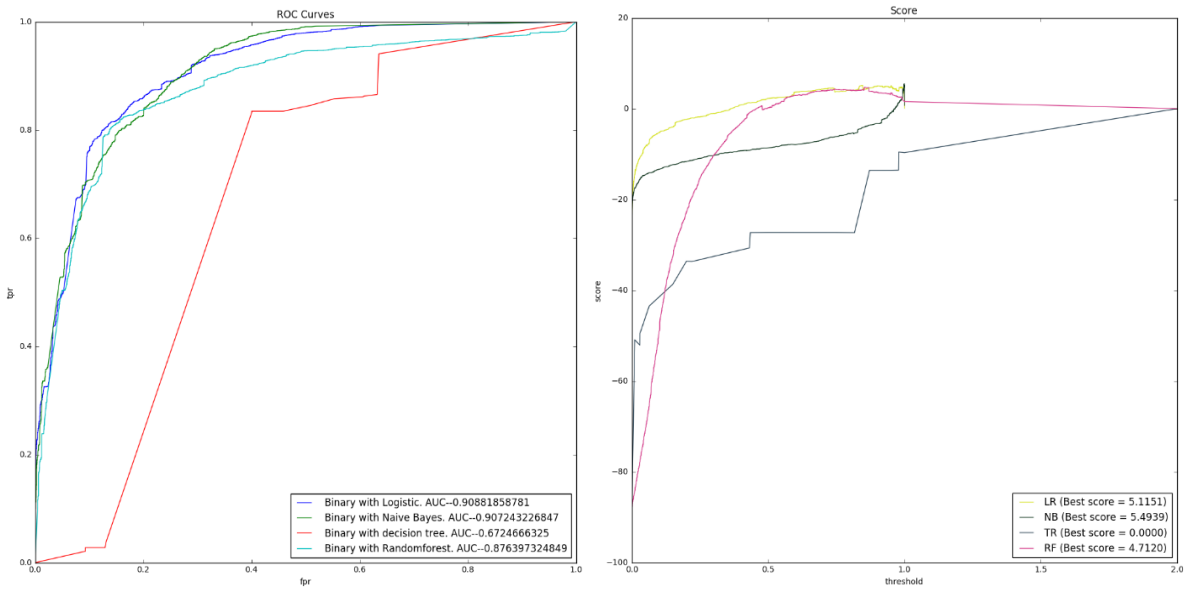

Figure 7. Performance of models on binary count with Chi-2.

4.5.2 TF-IDF

For TF-IDF features, we also used features selected by the two methods mentioned in the previous section. Shown below are plots of ROC and scores of the tuned models using features from the max_features method. Among the tuned models, random forest with 1500 trees and max features = log2(n_features) has the highest score **1.3897** when setting the threshold as 0.319, slightly better than the baseline model.

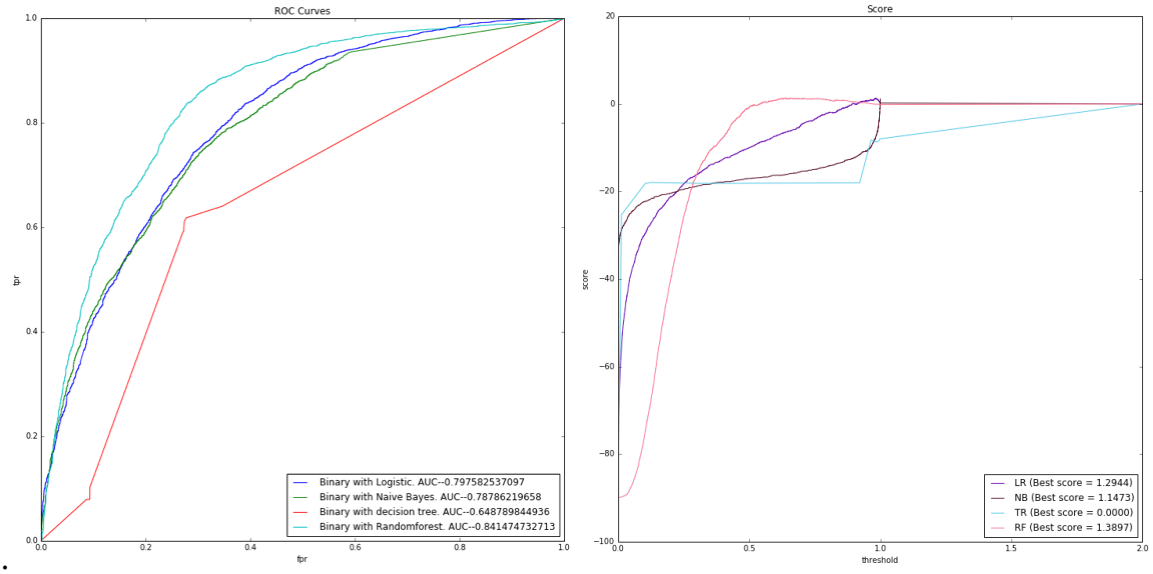

Figure 8. Performance of models on TF-IDF with max features.

Using the Chi-2 selection for TF-IDF, we tuned the models yielding the performance shown in the plots below. Among the four models, the random forest model with 500 trees and log2 features has the highest score of **9.1413**, much better than the baseline model, with threshold 0.2882.

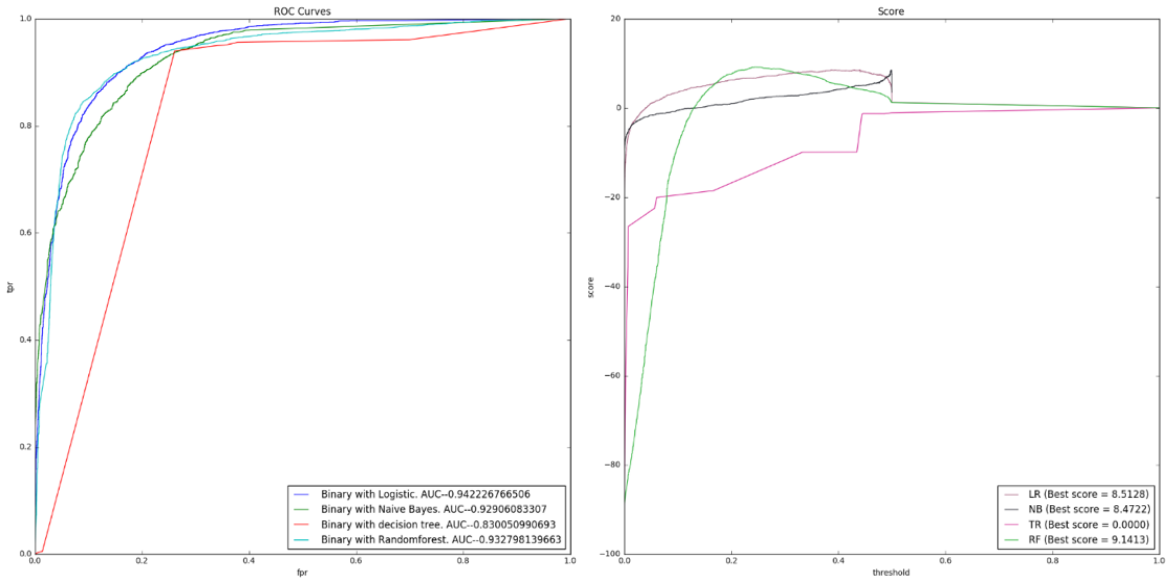

Figure 9. Performance of models on TF-IDF with Chi-2 selection.

4.5.3 Word2Vec

By applying the word vectors learned from the training data, we are able to transform the input email text data to word vectors. The same process is applied to both the training and the test data. Four different algorithms were implemented using this set of data: logistic regression, decision tree, Na ïve Bayes, and random forest. Each model was tuned using the grid search cross-validation method, and compared based on their performance on the test data using the score metric we developed. After grid searching, the best model is then fitted on the entire training data before

being assessed. Shown below is a plot depicting the comparison between performances of the tuned model for each algorithm on the score scale:

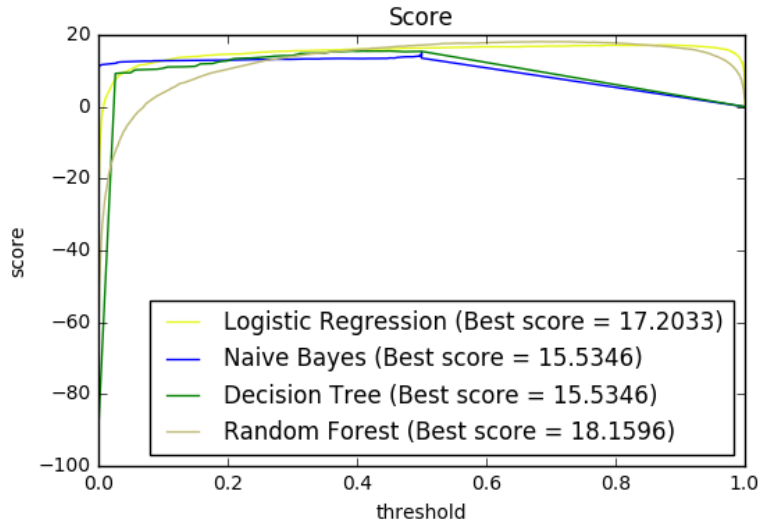

Figure 10. Score comparison of different models.

Random forest had the best performance, with a score of **18.16** on the test dataset (which has always been left out). This greatly exceeds the baseline performance of no filter and is close to the perfect score of 20. The best random forest model found by grid search had the following parameters: no limit on maximum depth, minimum 1 sample in leaf, minimum 3 samples for each split, and 1500 trees. These parameters make sense because typically in random forests models you wouldn't want to limit the tree depth (make each individual trees as complex as possible) and more trees to take the most advantage of the power of a bagging model. This model also achieved an AUC of 0.998. The discussion of other models can be found in the attached iPython notebook. The optimal threshold for classification is around **0.686**. In other words, only instances with a classified spam probability of higher than 0.686 should be classified as spam. This would greatly reduce the false positive rate. This is also the **best** model we found throughout the entire project.

5 Deployment

The core of the deployment of the result of the data mining lies in the prediction of the model we trained and tuned, which can be transferred using a pickle file. Depending on the architecture of client's system, we can either predict if each new incoming email is a spam or not, or perform the prediction in batches. Before getting to the user, emails will be delivered to our system. The emails will be pre-processed into our trained model's input data format (TF-IDF, Word2Vec, etc). The result of the prediction will go to two places: a filter that directs the email to the user's inbox or spam box, and a small database to store the instance and its prediction. Upon reading the email, users can report classification errors to the small database: either a regular email has been placed in the spam box mistakenly, or a spam mail has arrived at a user's inbox. Then, we can update our model with these new training data using active learning methods like stochastic gradient descent, thus improving our model through time.

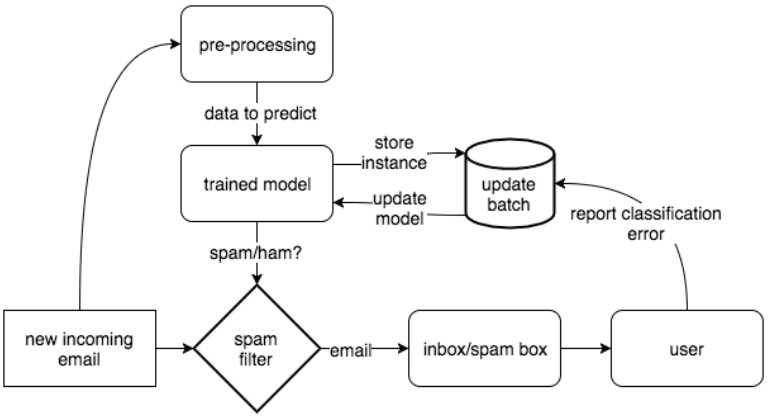

Figure 11. Deployment flowchart.

Any clients which uses this spam system must be aware of that important emails that may have been sent to spam by mistake. Since the accuracy of the system is not 100%, there is always a chance for mistake, even if it is small. Also, since we are constantly updating our model, clients should remain aware of possibilities of abusing this feature by attackers constantly feeding certain data to the system, thereby influencing the model.

The spam system is also prone to some subjectivity. Only users themselves can judge if an email is spam. The system may filter some advertisements as spam according to the dataset, but they might be useful for users. That's why we need expand the dataset continuously through updating correct results.

6 Ethics

Since this system would have access to user's email contents, there are ethical considerations regarding the privacies of users. Every user on the system must be aware that their incoming emails would be fed to this system, and that emails would be stored in the form of processed data for a certain period of time, but would be discarded after updating the model. In the long term, no email contents would exist on the system, other than contributing to the learned model.

7 References

1. V. Metsis, I. A. (2006, 6 19). Retrieved from http://www.aueb.gr/users/ion/data/enron-spam/
2. "Feature SelectionChi2 Feature Selection." *Feature SelectionChi2 Feature Selection*. Cambridge University Press, n.d. Web. 09 Dec. 2016.
3. Gibson, Chris Nicholson Adam. "Word2Vec." *Word2vec: Neural Word Embeddings in Java - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM*. N.p., n.d. Web. 09 Dec. 2016. <https://deeplearning4j.org/word2vec>.
4. "Word2Vec." *Word2Vec*. Google, n.d. Web. 09 Dec. 2016. <https://code.google.com/archive/p/word2vec/>.

Appendix I. Contributions

Jui-Ting Hsu:

- Modeling and feature engineering with Word2Vec
- Deployment on report.
- Putting report together.

Liwei Song:

- Modeling and feature engineering with both Count Vectorizer and TF-IDF using max_features selection method.
- Data preparation on report.
- Business Understanding on report.

Zhaopeng Liu:

- Modeling and feature engineering with both Count Vectorizer and TF-IDF using Chi-2 selection method.
- Data preparation on report.
- Data Understanding on report.

Binqian Zeng:

- Data cleaning from raw text
- Data understanding on report
- Ethics on report

Appendix II: Files

- DS1001 Final Project - Spam Email.ipynb

  iPython notebook performing the preprocessing of reading email from folders

- DS1001 Final Project - Word2Vec.ipynb

  iPython notebook performing data transformation and feature engineering using the Word2Vec model.

- DS1001 Final Project - Model with Word2Vec.ipynb

  iPython notebook performing modeling and analysis using feature vectors from the Word2Vec model.

- DS1001 Final Project – Selection using max features.ipynb

  iPython notebook performing data transformation, feature engineering, and modeling and analysis using the max features selection method.

- DS1001 Final Project – Selection using Chi-2.ipynb

  iPython notebook performing data transformation, feature engineering, and modeling and analysis using the Chi-2 selection method.

- utils.py

  Contains utility functions used throughout the project.

We also saved raw data, processed clean data frames, engineered features, and trained models as pickle files for future use. These will be available upon request.