Research On HRC's Released Emails

Final Project of Statistics 154, Professor Rabbee
University of California, Berkeley, Fall 2016

Group 2:
Boyue Sun
Junyuan Gao
Tongxin Xia
Haonian Wang

Part Ⅰ. Introduction

In this research project, our training set, HRC_train.tsv, contains full information of 3505 emails from top 5 contacts among the full email dataset. Our goal for this project is to create multiple kinds of classifiers to classify emails according to their senders and class variable, using words, phrases and features obtained from each email.

Firstly, we have a glance on the original training set, HRC_train.tsv. In the training set, each email is begin with a number that indicates its sender. However, regardless of the sender identification, it's not hard for us to directly detect which person does a certain email belong to because of some significant patterns. Even though, given extensive number of texts, it is both time-consuming and inefficient for human to find the pattern of each sender manually. Hence, in this report, we try to build matrices to record the detailed patterns as well as their weight for each email sender. With the matrices, supervised classifiers like SVM and Random Forest and unsupervised classifiers like K-means are used to produce the classification.

Our general guideline for this project is:
(1) Data cleaning, feature creation and filtering;
(2) Building supervised classifier using Random Forest and SVM;
(3) Building unsupervised classifier using K-means algorithm;
(4) Compare and select from above steps to confirm a final classifier model, produce predicted labels by applying the final classifier model onto the test set HRC_test.tsv;
(5) Making inferences, conclusion and figuring out further improvements.

Within those steps, we are able to produce a classification model which has a reasonable accuracy on classify the original emails.

Part Ⅱ. Feature Creation, Filtering and Creation

In the beginning of this part, we import the training set into R and then do the data cleaning part. We are able to reach a pure text corpus of data after data cleaning. At first, we use functions gsub() to replace '/' and numbers by the space ' ', and Corpus() to transfer each email into a string vector, which lead to total 3505 string vectors. Next we need to produce a large word dictionary from the string vectors. However, the string vectors we obtained from the first step are not feasible since there are lots of messy information like extra spaces, lower or upper case and common words(e.g. "is", "and", etc.) which will make the further calculation and feature selection inaccurate.

To avoid that, we use tm_map() to preprocess the corpus file:
(1) Removing the extra spaces, alphabet characters(i.e. comma, period, parentheses, brackets, etc.) and replace them into single spaces;
(2) Converting uppercase to lowercase in order to make words like"President" and "president" to be considered as a single unique word in our dictionary;
(3) Remove common words(i.e. different terms of a same word like "do" and "did") and stop words in order to pre-reduce the dimension of the word matrix.

In the end, we merge the original corpus profile into a profiled consent vocabulary which contains only 362 features.

After the above word processing part, we use function DocumentTermMatrix() to generate a word frequency weight matrix in order to produce the stemming process. Within this process, we can detect and extract some prefix terms, suffix terms and roots in our vocabulary, which significantly improve the amount of features. Those "stemming features" and terms obtained from above process together form a word frequency weight matrix. However, when inspecting some sample terms of the resulting frequency weight matrix, an extremely significant sparsity of level 100% come out to become a problem since lots of useless 0 terms will cost lots of time in further computing.

One reasonable reason of the phenomenon is that our stemming process produces too detailed terms which. To fix that, we reduce the dimension of the matrix(which is a nice way of remove the sparsity) by using function removeSparseTerms() with a sparsity threshold level of 0.99. After this process, we result in a final vocabulary with feature size is 358.

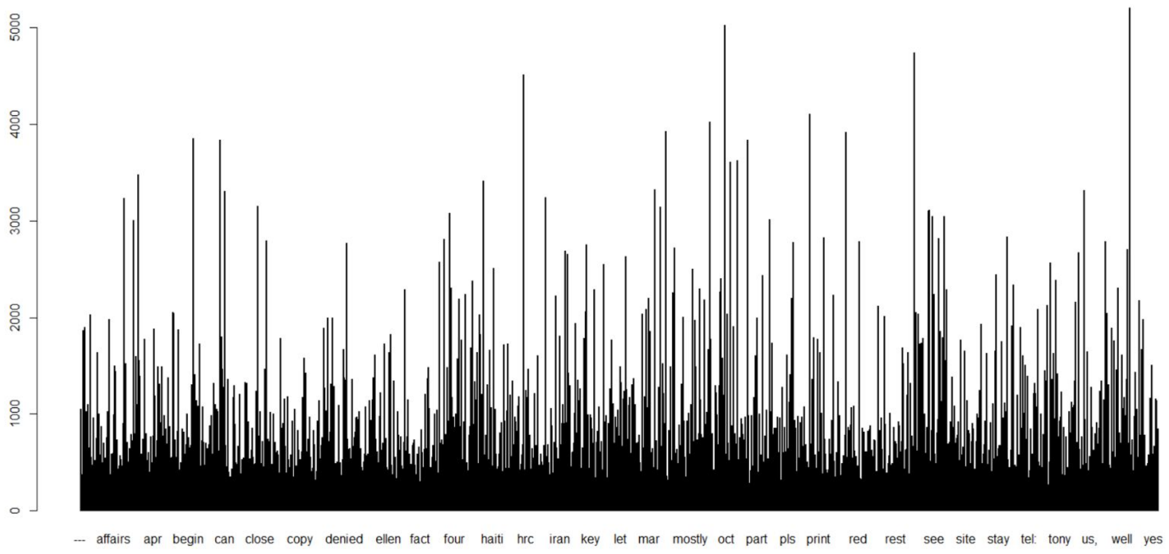

Figure 1: frequency barplot of features. Y-axis: frequency; X-axis: features

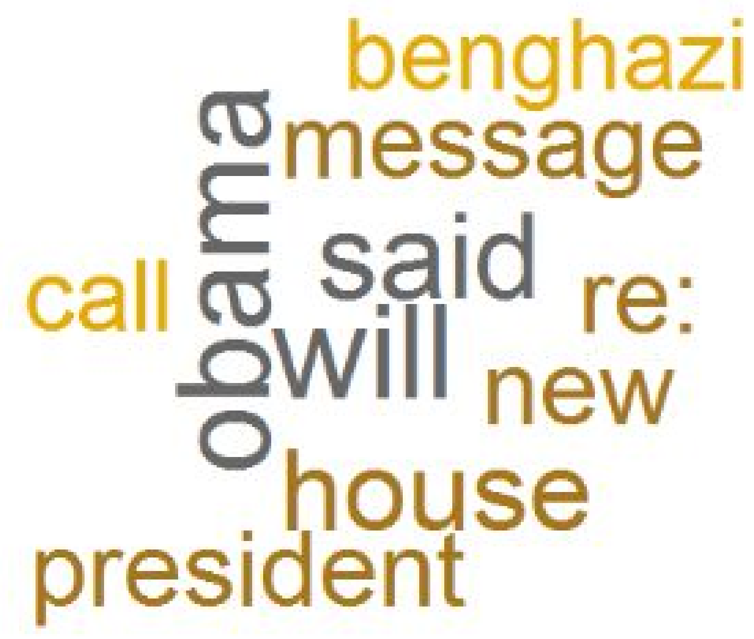

Figure 2: Word Cloud Plot of top 10 features; the size of word corresponds to its frequency

Now let us have a glance on our final features. From figure 1, we can easily observe that there are quite a large amount of features turn out to be much more significant than others. We just choose the features that have top 40 frequencies to become our "high frequency features" and make a word cloud plot of the top 10 features in order to intuitively observe its magnitude and difference in frequencies between them. At the end, we generalized a form showing the change of amount of features during the process in form 1.

Step	Total # of features
Raw	1907
Remove Stop words	362
Stemming	60955
After reducing sparsity	358

Form 1

Part Ⅲ. Build Up Supervised Classifiers using Random Forest and SVM

1.Random Forest

To apply the Random Forest Algorithm, we tune 2 parameters ntree and mtry and try to fit a best model using 5-fold CV. At first, we start at mtry=5 and ntree=500, which is a pair of common starting tunning parameters in experiments. By trying and modifying a large amount of parameters, we finally set mtry=9 and ntree=1000 as our final result and have the below result:

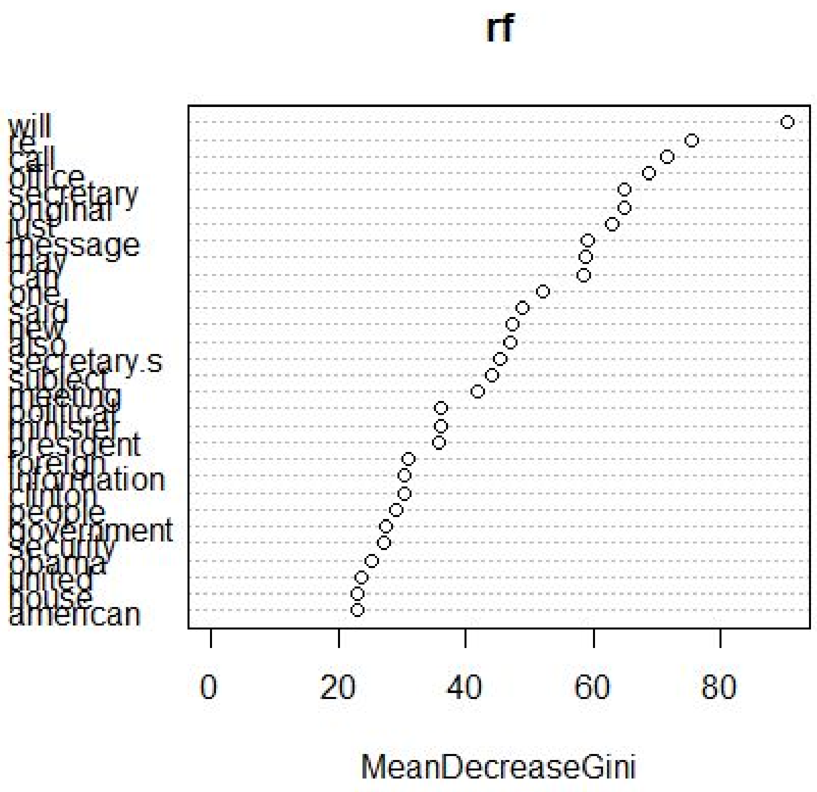

Figure 3: Variable Importance Plot

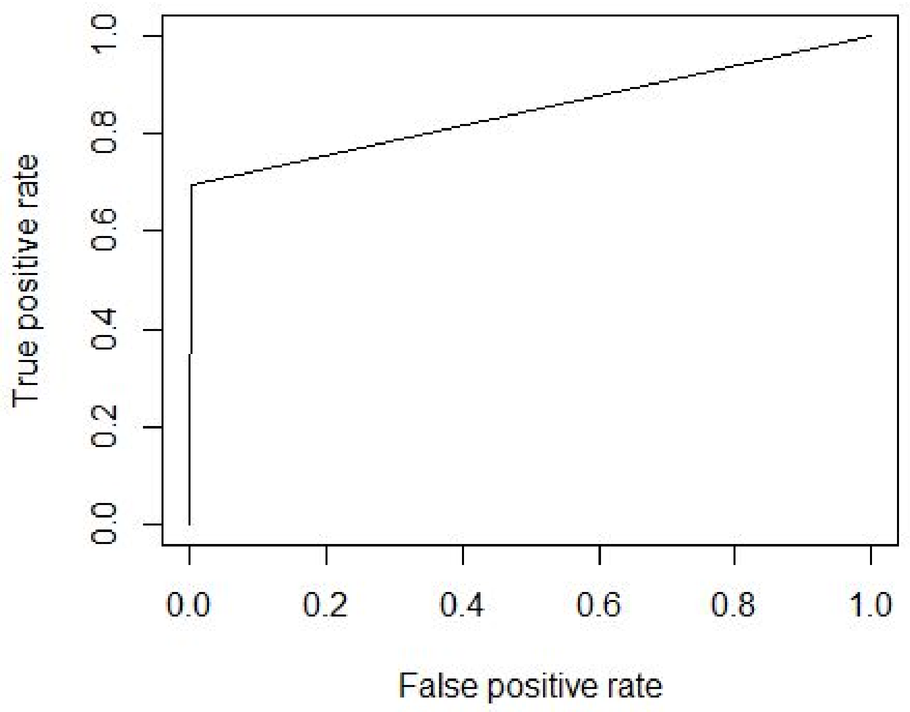

Figure 4: ROC curve of RF model

The above figures, figure 3 and 4 have shown some interesting facts. Figure 3 indicates the change of percentage in Gini Index(which is the difference between new model and old one) by removing certain feature. Thus, features like will, re. and call that goes far in x-axis is considered to be very “powerful” features in random forest model. From figure 4, we observe that when true positive rate is between 0 and approximately 0.7, the false positive rate is nearly 0, which means this model is plausible for most of the time. Moreover, we also figure out that the AUC value is 0.8463261, which is really close to 1 and shows that our model is a good classifier.

When we use our model to apply on a 5-fold CV, we receive the following result:

Step	Total # of features used	Total Accuracy(xx%)	Accuracy per sender class(from 1 to 5)
Random Forest	358	45.90585%	49.36%, 50.64%, 50.07%, 49.93%, 52.07%

2.SVM

Firstly, we make sure that choosing radial basis function kernels is better than linear or polynomial kernels in this case. Theoretically speaking, we choose radial basis function kernels mainly because of the following two reasons:

(1) Our word feature matrix after reduction contains more than 10,000 words, thus we first excluded the polynomial kernel as it sum all the variable then take the power, so in our case the parameter "n" at most equal to 2, and it has has probability that the "n" equal to 1 which is same as the linear method.

(2) However for more than 10,000 words, linear kernel still cost too much computation. Radial basis function has a significant computational advantage since we need only compute K(xi, xi') for all n(n+1)/2 distinct pairs i, i' where n is the total number of features. Thus we use radial basis function.

Besides, during the tuning process, we use the tune() in the kernlab package instead of the ksvm() or best.tune() in the e1071 package since best.tune() and ksvm() takes far more time to produce results and even we obtained the best fit parameters from best.tune(), R could not give out the best model for SVM. Therefore, tune() is the function that we choose to use after consideration.

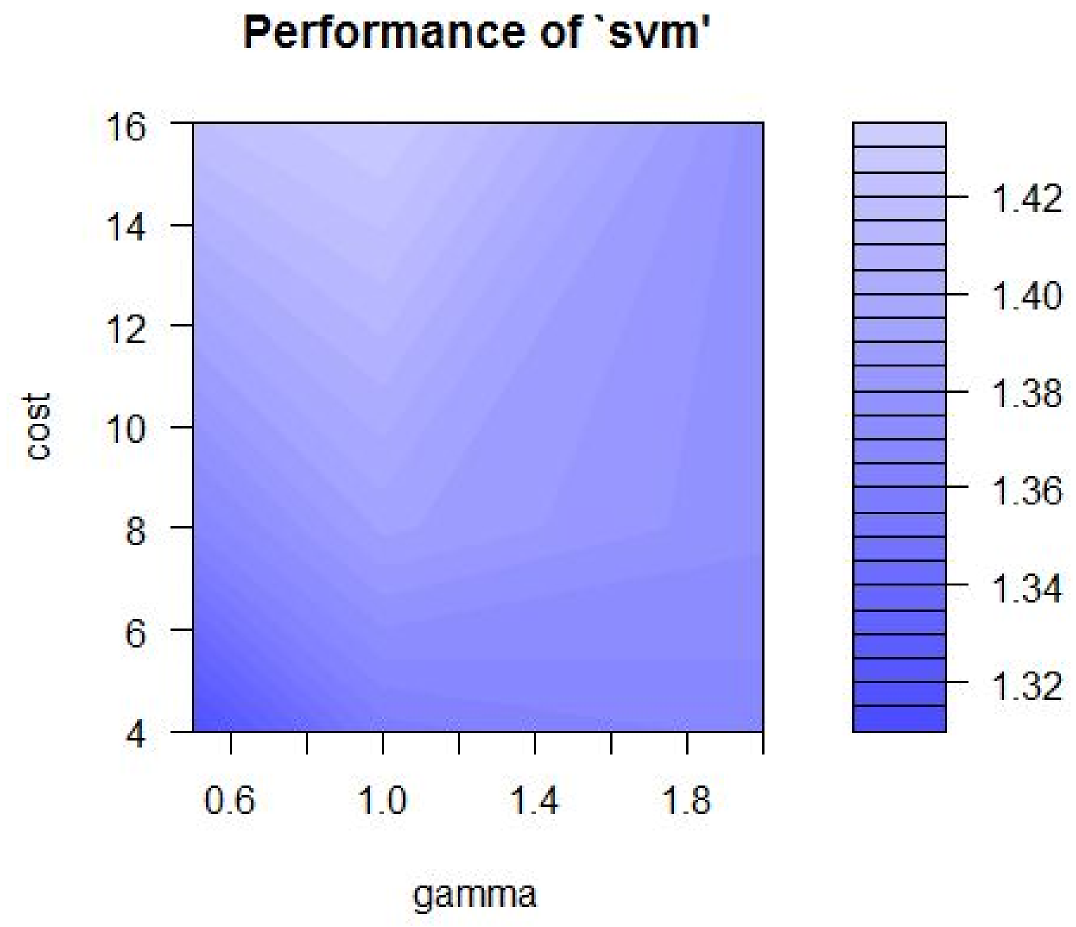

Figure 5: Performance of SVM models for different c and gamma

We use 5-fold cross-validation to tune the cost parameter in the SVM algorithm. Starting with c=4 and keep gamma fixed, we find that as c increases, error rate increases slightly. Similarly, as c decreases, error rate also decreases. We test several c-values and find that c=4 results in the least error. Then we make c fixed to derive the best gamma. An interesting phenomenon is that for different c-value, the variation trend of error when gamma increases is different. We can find this interesting trend through figure 5, where the deeper color indicates smaller error. Finally, we generally consider gamma and c-value together, and choose c=4 and gamma=0.5 as our best SVM model. The following form is a general statistics about our best SVM model trained in 5-fold CV:

Step	Total # of features used	Total Accuracy(xx%)	Accuracy per sender class(from 1 to 5)
SVM	358	50.41369%	49.36%, 50.64%, 50.07%, 49.93%, 52.07%

Part IV. Build Up Unsupervised Classifiers using K-means using top 100 RF features

In this part, we need to build unsupervised classifier on the training set. We just simply use K-means clustering algorithm to cluster emails using top 100 word features from Random Forest classifier in the step above. After calculating by using k-means algorithm, we cluster the accuracy, which is 0.06932953.

Part V. Compare and select the final classifier model

Based on above results, we can easily exclude K-means since its accuracy is much less than others. That might because we only use top 100 RF features out of all features but the features that we abandoned are also powerful. Then, comparing between Random Forest and SVM, we finally chose SVM as our model since it has a conspicuously higher accuracy than the accuracy of RF as well as a consistent performance through each folds.

Therefore, we use SVM for next step's prediction.

Part VI. Final Result and Improvement

Final Results: We use the SVM model with c=4 and gamma=0.5 as the optimal model to predict the test set.

Improvements: After doing all the process of this project, and we discussed each step of the requirement ask, we found that there is a part can be improved in our word feature. For the word feature, since we raise the top 40 frequency, so maybe we left some important words whose frequency are not in top 40. Therefore, we took this restriction then we lose some important information. after that, we thought we can add more common word on the common english word list to help us to filter some common words. Moreover,one thing that could also be a potential improvement is the power feature. Since for each email sender, there are actually patterns among his vocabulary which could not significantly appear on the frequency weight matrix. If we consider power features, we might receive a higher accuracy than the existing value.

Also, if we can see the accuracy rate from the test dataset, we can change the values of c & gamma until we got the highest accuracy rate of the test dataset. So, our predictions will be more accurate.