Group 4
Renee Sweeney
S154 Final Project
Fall 2016
Due: Dec 2nd 2016, 2pm at 309 Evans

Team Member Name	Contribution	Signature
Abigail Chaver	1. Feature creation and filtering	
Pranay Singal	1. SVM classifier	
Kensen Tan	1. RF classifier 2. Test set predictions	
Renee Sweeney	1. K-means classifier 2. Report write-up	

Team Leader Signature: _____
Date: _____

Introduction

In this project, our goal is to classify the sender of emails found on Hillary Clinton's email server. Our high-level objective is to identify patterns in each sender's email that are distinct to them. Our primary method of analyzing this large email set is to compute a word frequency table, but we are also interested in other characteristics that may be unique to each sender. Our priority is to build an accurate classifier that performs well on a test set. In order to build our classifier, we will compare 3 modeling approaches: Random Forest, Support Vector Machine, and K-means Clustering. Since we can only estimate our test error with the supervised methods, we compare the results of RF and SVM to select our final model. With each of these methods, we use cross validation to tune the parameters in order to select the best model. We ultimately find that Random Forest performs highest in 5-fold cross validation, and use this as our best classifier.

Data Processing

The initial training data consisted of a tab separated file with two columns: the sender code (labeled 1-5) and the string of the email. With 3505 emails in the training set and another 389 as our test set, we proceeded to converting the emails into a frequency matrix.

Initially, we read in the tables with the parameter sep = "\t", because the files were tab-separated. However, this somehow misread the data, concatenating some entries, so that most of our original modeling was done on an erroneous dataset. We found this error very late. After we reread the data correctly without this parameter, we had to re-tune our models at the last minute. Given the computationally intensive nature of the modeling, this was a challenge.

After reading in the tsv, we converted the string to a character vector. We then wrote those vectors to a corpus so that we could use the R package `tm` to remove punctuation, numbers, and stop words; convert to lowercase; and stem. Next, we removed common words that were seen in every email, such as "subject", and "US", and words that were seen in 3 or fewer emails- this helps us eliminate the long tail of words that are used too infrequently to provide any sort of pattern. We visualized some of the data to get a sense of its distribution and possibly pick out other useful characteristics. We first looked at the sender frequency.

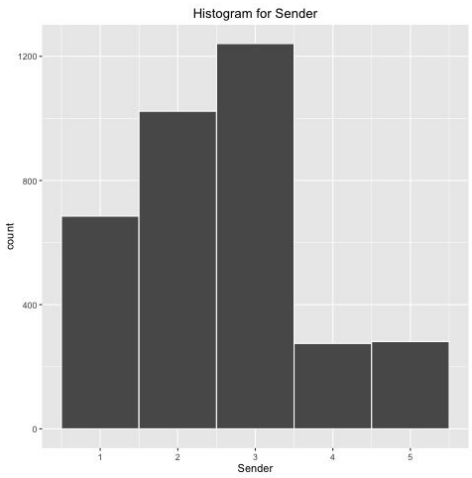


Figure 1: Sender 3 has the most emails, while 4 and 5 both have much fewer.

We also did some wordclouds to visualize the differences in word frequency.

Figure 5: Wordclouds by Sender

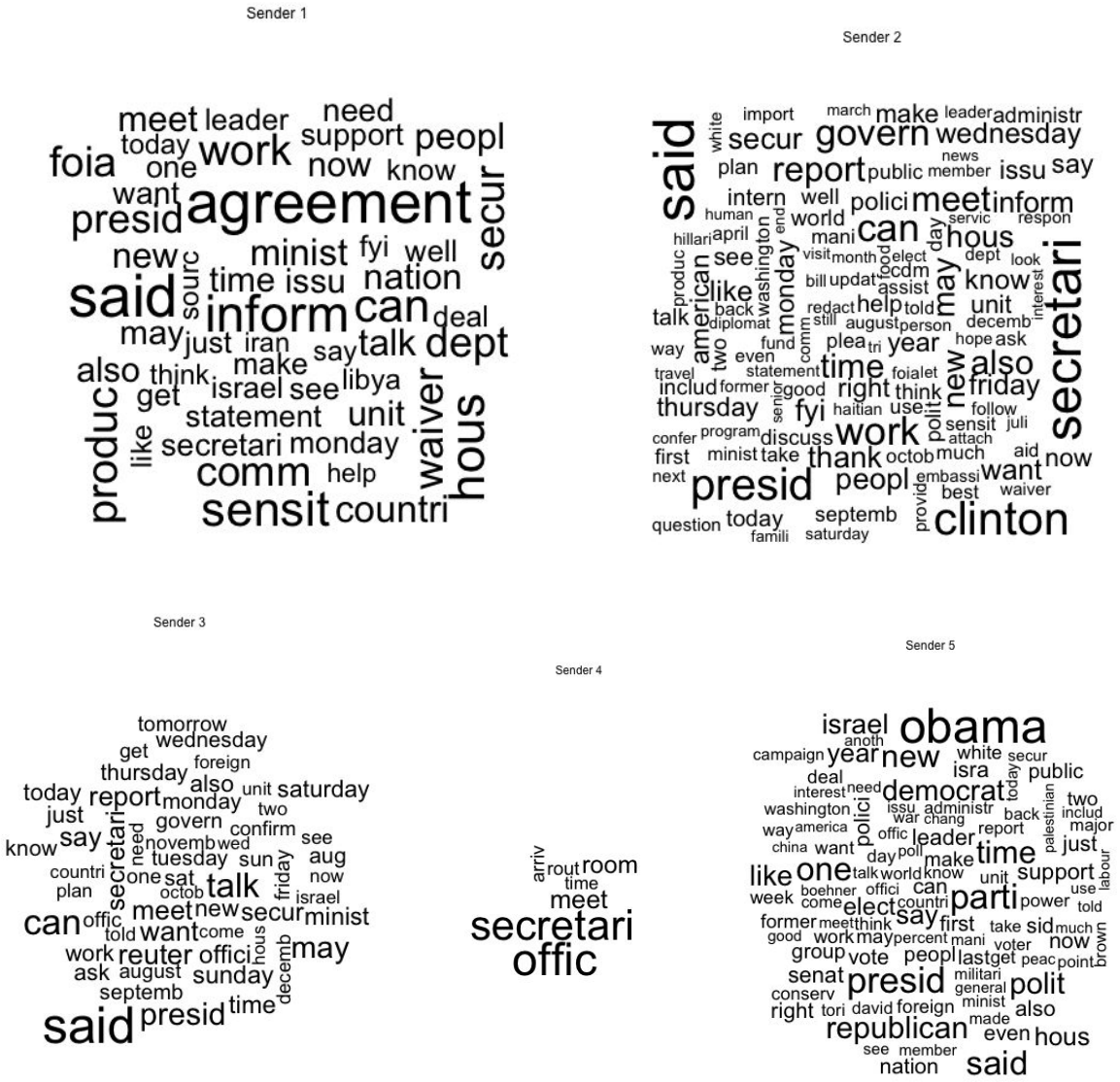


Figure 2: Filtering the wordclouds to use only words appearing more than 200 times yielded some interesting results - the senders with the highest quantity of high-frequency words (the largest wordclouds) are not the senders with the most emails.

We were curious about other qualities besides word frequency and took a quick look at the number of characters in each email.

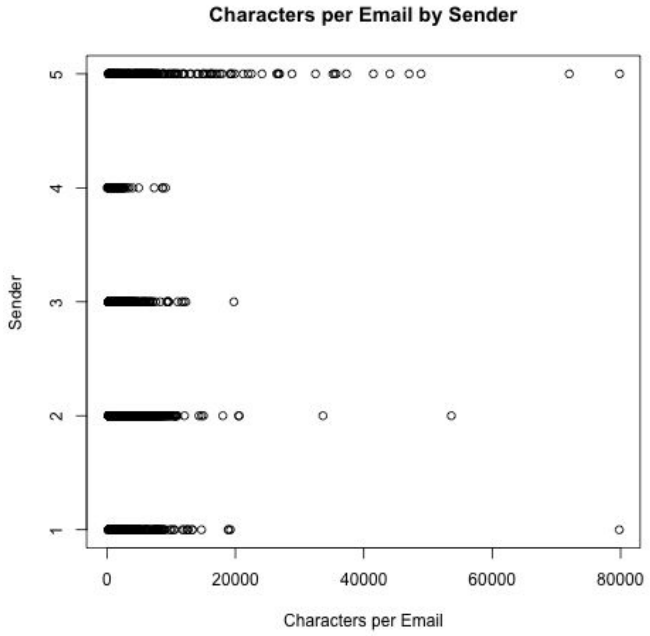


Figure 3: The distributions seem different enough that this could be a useful feature.

We also explored other possible features that could predict sender, including number of words, average word length, and rate of use for ampersands, semicolons, question marks, numerals, and hyphens. We performed simple one-way ANOVA to determine whether or not to include the extra feature in our matrix.

We found that words per email, number of hyphens divided by number of words, and number of question marks divided by number of words all yielded small p-values. Ordinarily we would be concerned with the significance issues that arise with multiple testing, but this was just a preliminary vetting. We added these three as features to our matrix. It is worth noting that this method is not ideal, as any significant higher-level interactions will be missed.

Through these steps, we produced feature matrices with the following numbers of terms:

Step	**Number of Terms**
Raw	38427
Stop Words	38327

Stemmed	27777
Frequency Selection	9789
Additional Features	9792

Unfortunately, our reduced feature matrix was still extremely large, at 9792 columns. We tried two strategies to reduce the number of columns.

The first was to use univariate ANOVA to remove features that had no statistical significance. As mentioned before, this method is not highly defensible statistically; it both creates problems with multiple testing (since we are running about 9000 hypothesis tests), and will miss any features that have significant interaction effects but not one-way effects. However, we thought it might be worth a try, so we selected only words whose ANOVA produces p-values of less than 0.05, resulting in 3599 features. Unfortunately, this data produced higher error rates than the original, probably due to lost information that could not be captured in a one-way analysis.

The second was a feature selection method called the Boruta algorithm. The Boruta algorithm avoids the problem of only catching one-way significance. This algorithm takes a sample of the data and shuffles within-column values. Then it compares the results of running RF on the "shadow" data with the original data. The procedures are repeated a number of time to time; if a real feature has a statistically evident higher importance than the best of the shadow features, it is confirmed as important.

Sadly, when we ran Boruta, nearly all of the features were rejected as unimportant. We took the small subset (10 features) that was labeled "tentative", and computed a power feature matrix that went up to cubic powers, including interactions (about 1100 features). However, the cross-validation error it yielded was pitiful, at about 50%, as it gave up an enormous amount of information.

Random Forest

The first supervised method we tried is using Random Forest. An advantage of Random Forest in this case is that it should be able to handle the potentials of multicollinearity among our long feature lists. However, one downside of that is that it takes significantly longer than SVM. Nevertheless, we implement and tune the random forest in the following way.

- With the full feature list, $\mathbf{D_0}$, we try different combos of parameters. In this case, it is the number of trees, **B**, and number of random features at each node split, **m**.
- Compare the 5-fold CV error for each combo of parameters, and run the full model on the best one.

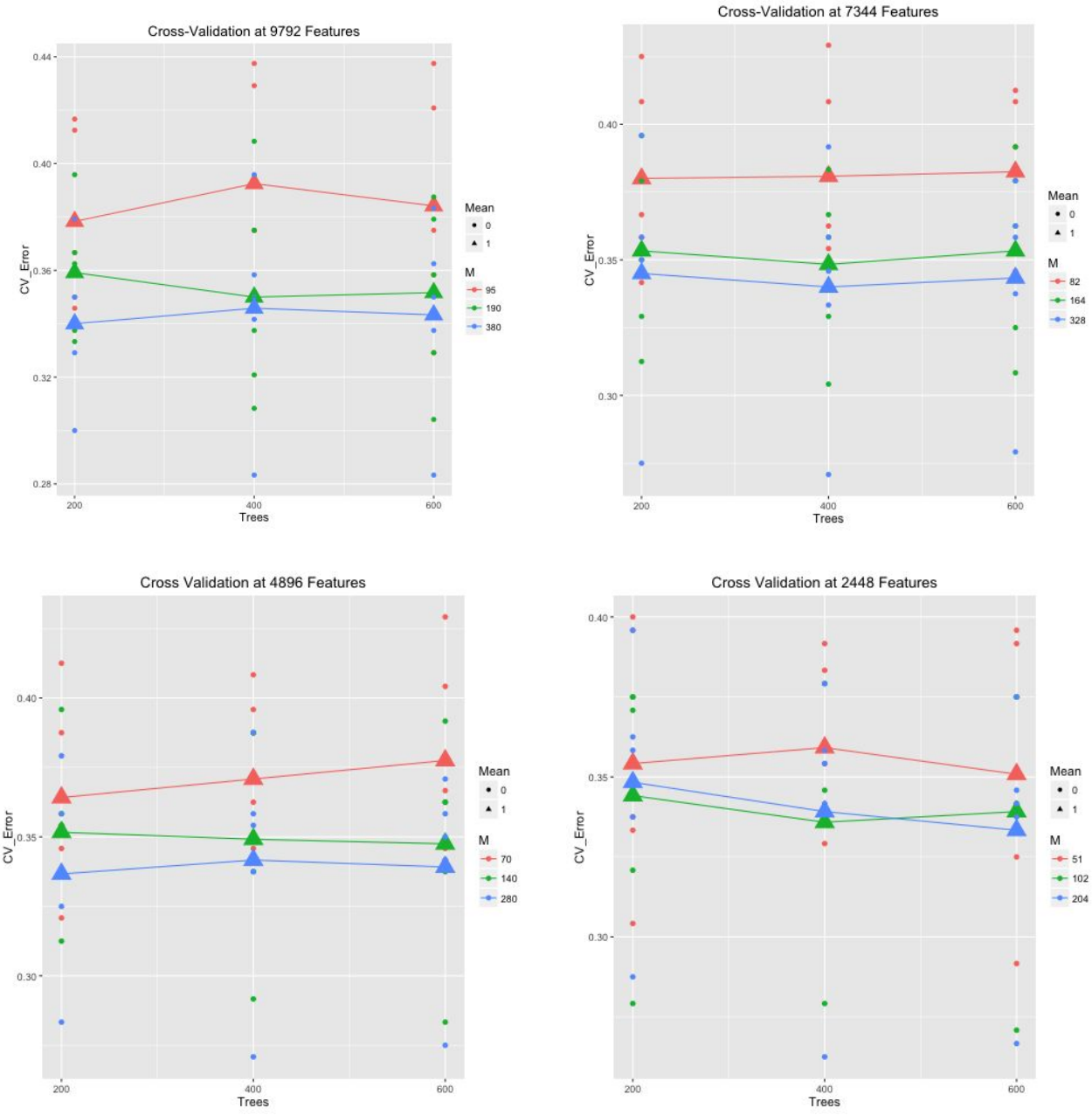
- From the previous best model, we remove **0.25*D_0** least important features, ranked by the mean decrease of Gini.
- Repeat the above process up to 3 times until the reduce featured list does not yield a lower 5-fold CV error.

We try 3 values of **B** and 3 values of **m** in each of iteration, so each iteration will have 9 combos:

B = {200, 400, 600} m = $\{\sqrt{Di},\ 2*\sqrt{Di},\ 4*\sqrt{Di}\}$

As we see, we started with the full feature set, and then we continue shrinking our full feature set from 9792 to 7344, from 7344 to 4896, and so on.

Figure 4: Cross Validation Errors Over 5 Iterations of Shrinking



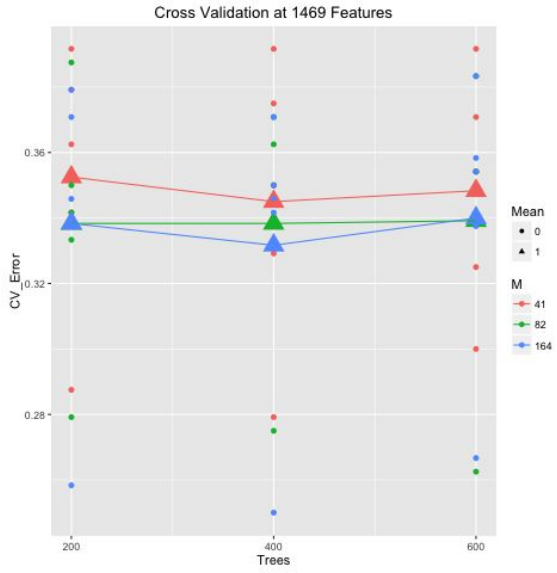


Figure 4:

The result is fairly straightforward; it seems like there are as the size of feature space decreases, the lowest CV error for each iteration reduces slightly. On top of that, it also reduces the computational time due to the shrinkage of feature space. Therefore, we want to continue cutting down feature space size to ***15%*** *of* $D_{o.}$

Surprisingly, the CV error still went down, so we favor this model the most along with its smallest feature space size. We could just stop right here, but the accuracy was still not impressive. As a result, we looked into adding another feature based on the concept of Topic Modeling using **Latent Dirichlet Allocation (LDA)**.

The mathematical explanation of LDA could be rather complex, but the idea is that LDA sees each document as a mixture of topics using probabilistic models. In our case, we can choose **k** topics among our documents and the model will try to identity each document as one of the k topics.

Class	israel	inform	report	parti	can	clinton	say	secretari	haiti	obama
(1)	0.05	0.39	0.05	0.07	0.07	0.07	0.05	0.09	0.08	0.09
(2)	0.06	0.39	0.05	0.06	0.11	0.06	0.08	0.08	0.05	0.06
(2)	0.05	0.5	0.06	0.08	0.06	0.04	0.04	0.1	0.03	0.05
(2)	0.06	0.42	0.03	0.03	0.02	0.19	0.08	0.02	0.05	0.1
(1)	0.14	0.32	0.12	0.05	0.05	0.08	0.07	0.09	0.04	0.04
(1)	0.05	0.45	0.05	0.05	0.06	0.05	0.05	0.11	0.05	0.06

For k = 10, each document has a probability being assigned to each topic. The algorithm will label each document with the topic with the highest probability. For the first six documents, the algorithm labeled all of them to topic 2 “inform”, and the senders are 1 or 2. We inspected the topics probabilistic distribution by senders and observed some patterns. For example, sender 3 is much more likely to be labeled as topic 5 than any other senders.

As a result, we decide to append it to D_5 and calculate the new CV error on the best model with this additional new feature. Our assumption was that only one additional feature would not hurt the model at all, but if it is a meaningful feature it would definitely improved the accuracy.

After appending the topic labels with **k** = 15, our CV error improved slightly, by only 0.5%. The change was very minor, but however, the model treated this feature as one of the important features with mean decrease Gini of 10.9. Anyway, we did not include this in our feature space.

Please note that the above procedures we took were based on the incorrectly imported dataset. As much as we would want to use the same procedure, which we believe to be one of the most comprehensive methods, due to the time limit we could not process that method. We had to stop our first big loop because after 12 hours nothing came out.

Therefore, we need to take short cuts and guess the parameter values that might fit.
We first filtered out all the words with less than 10 total frequencies, which cut down the number of features to around 4971. Our assumption is that these rare words would not tell us too much information, although it is also possible that we also cut away some rare words only used by a specific sender.
First Attempt: ***B*** *= {200, 400} and* ***m =*** $\sqrt{4971} \approx 70$

(B, m)	1st fold	2nd fold	3rd fold	4th fold	5th fold	average
(200,70)	0.26	0.28	0.29	0.26	0.29	0.278
(400, 70)	0.26	0.28	0.27	0.27	0.29	0.274

As we see, with 200 more trees, the 5-fold cross-validation error only decreases slightly, so the extra computations will not be worth it, especially for us. Using the right data, even the simplest model beats the accuracy of the old one. Afterward, we rank the features by mean decrease Gini from our best model and select the best 2500 for the next iteration.

Using 2500 features and m = 50,

(200, 50)	0.26	0.27	0.28	0.25	0.29	0.270

The error decreased by 0.8%; although it is not too significant, but we definitely favor this model more due to the smaller feature space size! Interestingly, looking at accuracy by class, only sender (1) and sender (4) gained a lower accuracy, whereas the other three senders improved. We wish there are times for us to try out different combos of tuning parameters, but due to time limit, we will choose the above model as our best model.

Step	**Total # of Features Use**	**Total Accuracy**	**Accuracy by Class**
RF	4971	0.722	● Sender 1 Accuracy = 0.83 ● Sender 2 Accuracy = 0.68 ● Sender 3 Accuracy = 0.68 ● Sender 4 Accuracy = 0.96 ● Sender 5 Accuracy = 0.90
RF(with feature selection)	2500	0.73	● Sender 1 Accuracy = 0.82 ● Sender 2 Accuracy = 0.69 ● Sender 3 Accuracy = 0.69, ● Sender 4 Accuracy = 0.94 ● Sender 5 Accuracy = 0.93

Support Vector Machine (SVM)

The second type of classifier that we utilized in our analysis of the emails was Support Vector Machines (SVM). The goal of SVM is to create an optimal hyperplane that predicts the class an observation is in depending on which side of the hyperplane it lies. To create the optimal hyperplane, one which maximized the margin between the support vectors and the hyperplane, we utilized the functions within the package 'e1071' in 'R'. In using the functions, we were asked to provide the type of kernel, the value of cost, the value of gamma (when using the radial kernel), and the number of degrees (when using the polynomial radial), aside from the formula, the data set, and the number of folds in doing cross validation.

In order to find the optimal hyperplane, we had to first find the best tuning parameters (value of cost and value of gamma for each kernel type) that yielded the lowest error rate using cross-validation. To find these tuning parameters, we ran multiple iterations of the 'svm' function with the following options for kernel type, cost and gamma:

kernel = {"linear", "polynomial", "radial"}

cost = {0.1, 0.5, 1, 10, 100}
gamma = {0.1, 0.5, 1, 5}

For kernel, we utilized the 3 typical options as they allowed us to use linear and non-linear approaches, accounting for scenarios where the data cannot be best divided by a linear class boundary. For cost, we wanted to have iterations in which we have a low and a high budget. Having a low cost-value, means that we want to restrict the number of slack variables, and a higher cost-value means that we allow for more violations. Similar to the cost, we fit an array of values for the tuning parameter. A higher value for gamma means two observations are considered to be in the same class if they are closer to each other, while a lower gamma value allows the radius of the area of influence to be farther and as a result more data points are captured in certain classes. With each value of the cost and gamma tuning parameters, we experience a tradeoff between the bias and variance. We can find the optimal tradeoff point and the tuning parameters by using cross-validation.

Prior to running the code, we decided to filter some of the features out. In our original data set, we had 9792 features. We noticed that in more than 50% of the features (5657 features or 57%), the word feature appeared less than 10 times. Due to the smaller occurrence of the word in comparison to the overall number of observations, we decided it would be best to filter out these features from our overall data set as they are potentially noise variables.

With the refined data set, we determined the optimal hyperplane and SVM classifier model. In our code, we utilized the tunecontrol parameter to run a 5-fold cross-validation that would reduce overfitting and provide a better estimate for the error rate. Running the code across the different permutations of kernels, the cost tuning parameter, and the gamma tuning parameter, we determined the CV error for each permutation.

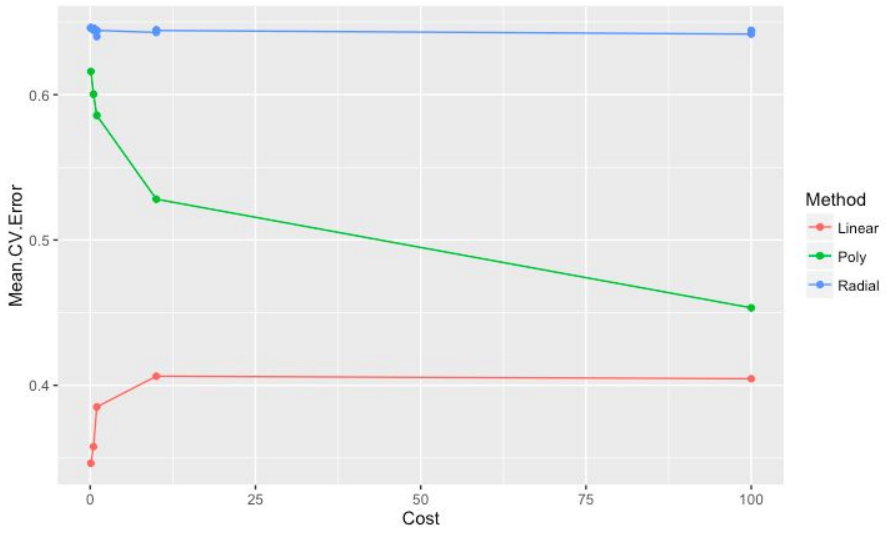


Figure 5: Linear Cross Validation outperforms the others easily, doing well at low cost values

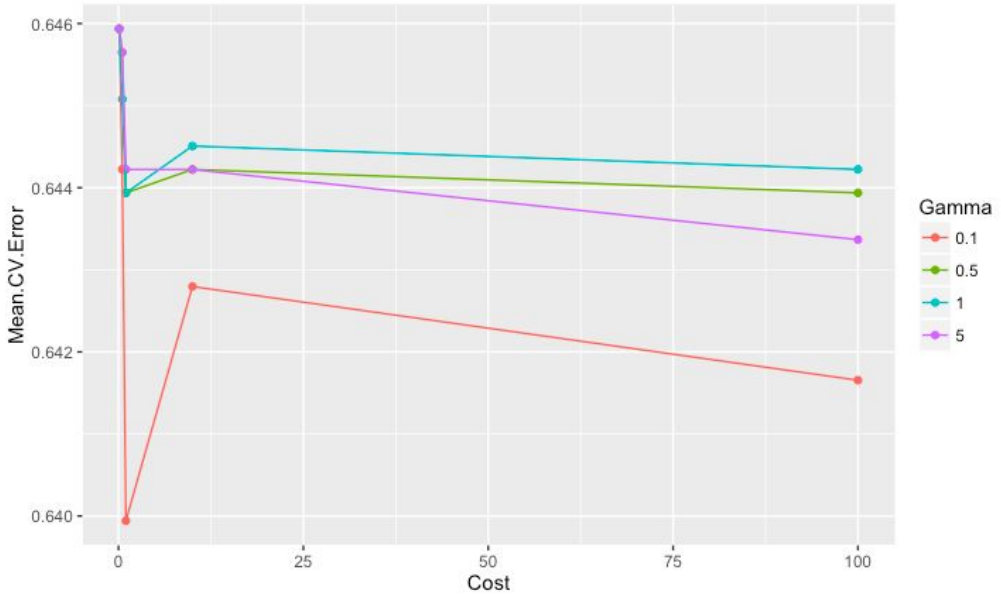


Figure 6: Radial Method generally performs poorly, but does best at gamma = 0.1.

In Figure 5 and Figure 6, we have graphed the CV errors for the linear, polynomial, and gamma kernels. As you can see, across the 3 kernels for different values of gamma and cost, the linear kernel has the lowest CV error, thus outperforming the other 2 kernels. Within the linear kernel itself, the best CV error occurred at a cost-value of 0.1, which is the optimal tuning parameter and the one being used for the best model.

In our SVM analysis, certain features placed a larger weight on the support vectors that constituted the hyperplane. The top 10 important features (in order of importance) were as follows: 1) Locat, 2) Ltd, 3) Marri, 4) Massiv, 5) Mashaban, 6) Megan, 7) Master, 8) Mcdonald, 9) Match, and 10) Medicin.

To determine the rankings for the overall feature set we followed the methodology described in the paper “Gene Selection for Cancer Classification using Support Vector Machines.” In this methodology, the coefficients for the set of features were used and a weight was determined for each feature, by dividing the mean of the coefficients by the standard deviation. The absolute value of the weight determined the ranking.

Step	**Total # of Features Use**	**Total Accuracy**	**Accuracy by Class**
SVM	4135	0.633	• Sender 1 Accuracy = 0.42 • Sender 2 Accuracy = 0.66 • Sender 3 Accuracy = 0.74 • Sender 4 Accuracy = 0.55

			• Sender 5 Accuracy = 0.72

Since the random forest model outperformed SVM, we will make our prediction based on the random forest model 2500 features, **B** = 200, and **m** = 50.

K-Means Clustering

Unsupervised methods such as k-means clustering can be useful for analyzing your feature space and seeing how data is grouped together. To create unlabeled data without too much noise, we calculated the one hundred most important features from the random forest classification and indexed our training data to include only those features. When passed through the kmeans() function with five clusters, different parameter values of nstart (1, 20, and 50) resulted in the same cluster assignments. This indicates that the global optimum is easily found, as opposed to a local optimum obtained with a low nstart value. The within-cluster sum of squares was 137630564, possibly due to high-dimensionality and several larger outliers in the "num_words" feature. Figure 7 shows that "num_words" is optimal for visualizing the clusters in two-dimensions. The clusters seem to be based on that axis alone, even though it is only the third most important feature, as opposed to "sid", the most important feature.

Although labels are not usually available for unsupervised methods, we were able to compare our cluster sizes to the number of emails assigned to each sender (label). Our algorithm output clusters of size 452, 28, 2892, 5, and 128, while there are 685, 1023, 1241, 275, and 281 data points for labels 1, 2, 3, 4, and 5, respectively. Seeing this and Figure 7 below, it appears that five outliers were clustered together, and 2892 emails with a low number of words were clustered together. Overall, the clustering method did not seem to group the data according any other feature except "num_words", which makes us question the validity of our one hundred most important features. Two features were deemed more important than "num_words" but according to our plots, they had little significance in grouping the data.

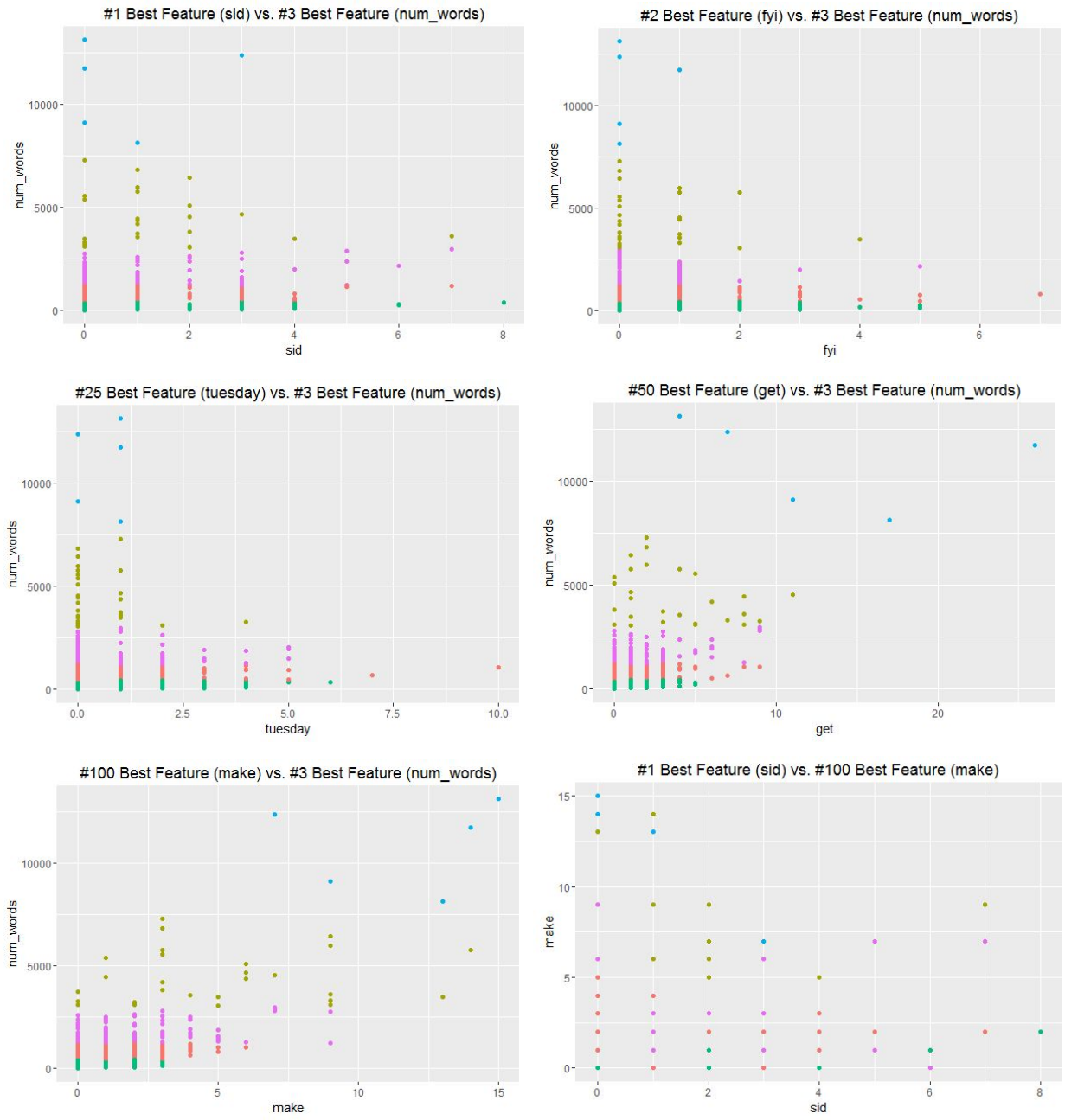


Figure 7. Various plots of clusters in two-dimensions using different features on the axes. The feature "num_words" shows the clusters most clearly, whereas in the bottom left plot, clusters are overlapping, which happens in a majority of these plots since the data is high-dimensional.

Conclusion

This project, from importing the emails, processing them, and turning them into a document term matrix, was something new for all of us. It was very interesting to apply different machine learning algorithms to a recent controversial political topic. We were able to use the core

theoretical and code-based skills we developed in class and learn even more along the way. Everything from preparing the data, filtering it, selecting features, and tuning the models was done from scratch. The challenges we faced overcame will be useful lessons for the future.

We also learn to accept the annoying part of Machine Learning which is the run time, and the frustration that not all algorithms we try will work. Alongside, we learned the basics of text mining and natural language processing, and the terminologies of sparsity and corpus. When we struggled to improve the accuracy, we discovered interesting algorithms such as Boruta and Latent Dirichlet Allocation. Although they might not have helped us a lot in this project, we believe they could be useful for us in the future. Overall, all these things simulate our interest in doing more future Machine Learning projects. Lastly, as much we wish the mistake did not happen, we learned to work as a team last minute. We are very curious about how we could further improve the accuracy with more time.

References

Awati, Kailash. "A Gentle Introduction to Text Mining Using R." *Eight to Late*., 19 Oct. 2016. Web. 02 Dec. 2016. <https://eight2late.wordpress.com/2015/05/27/a-gentle-introduction-to-text-mining-using-r/>

Chen, Edwin. “Introduction to Latent Dirichlet Allocation”. *Edwin Chen.,* 22 Aug. 2011. Web. 02 Dec 2016. <http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/>

Guyon, Vapnik, “Gene Selection for Cancer Classification using Support Vector Machines“. *Barnhill Bioinformatics*. 2000. Web. 02 Dec. 2016 <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.9598&rep=rep1&type=pdf>

Thompson, Jim. “Using the Boruta Package to Determine Feature Relevance” *Kaggle.*, 13 Feb. 2016. Web. 02 Dec 2016. <https://www.kaggle.com/jimthompson/bnp-paribas-cardif-claims-management/using-the-boruta-package-to-determine-fe/comments>