Text Mining: Predicting spam/ham labels in text messages

Stat 154 Final Project
Fall 2014

Geoff Kaufman
Chloe Lim
Dilip Ravindran
Justin Sampson

Table of Contents

1. Description of data

The data given is comprised of 4180 texts, one per line, each of which begins with one of the "spam" or "ham" labels, and the rest of the line representing the actual string of text. In total, there are 3615 ham texts and 565 spam texts (86.5% ham, 13.5% spam). The presence of the pound currency symbol and the 11-digit phone numbers as well as word spellings ("colour") and website URLS (".co.uk") indicate that the texts are from the UK. Texts range in length from one to 80 words, including strings of numbers and punctuation, and appear to come from a variety of sources with a variety of intended recipients – some of the ham seem to be sent to friends, spouses, relatives, and others are sent from companies, coworkers, etc. Much of the language used in the texts appear to be shorthand ("u" instead of "you", "cos" instead of "because"), in both spam and ham messages.

We conclude that the texts are everyday texts originating in the UK, and we treat them as if they are written in British English.

2. Feature Creation

We first parsed the data using Python's csv libraries to read in each line of the text file, which resulted in a matrix with 4180 rows (texts) and 11,046 columns (word features). We then removed all punctuation, converted the text to lowercase, and removed all stop words. Our original plan was to remove all rows which contained only stop words (of which there were about 80), however we realized that these rows may still hold significance due to their power feature values. Thus we removed these rows for word feature matrix testing but included them in the combined matrix. The resulting matrix with dimensions 4180 x 8197 was transferred to R, with labels extracted into a separate column. We used the convention of **1 for spam** and **0 for ham**.

Next, we began the process of normalizing each row so that the cells represented relative frequencies within each text and each row summed to 1. We had some issues with efficiency in this step – the sheer size of the matrix resulted in an incredibly slow runtime for the relatively simple process of dividing each cell by the total count of the row – but we eventually figured out a way to use R's "apply" function to speed up the process and give us the correct output.

Our final result was a word feature matrix representing 4180 texts that contained a total of 8197 unique word features (with the first column representing the spam/ham labels).

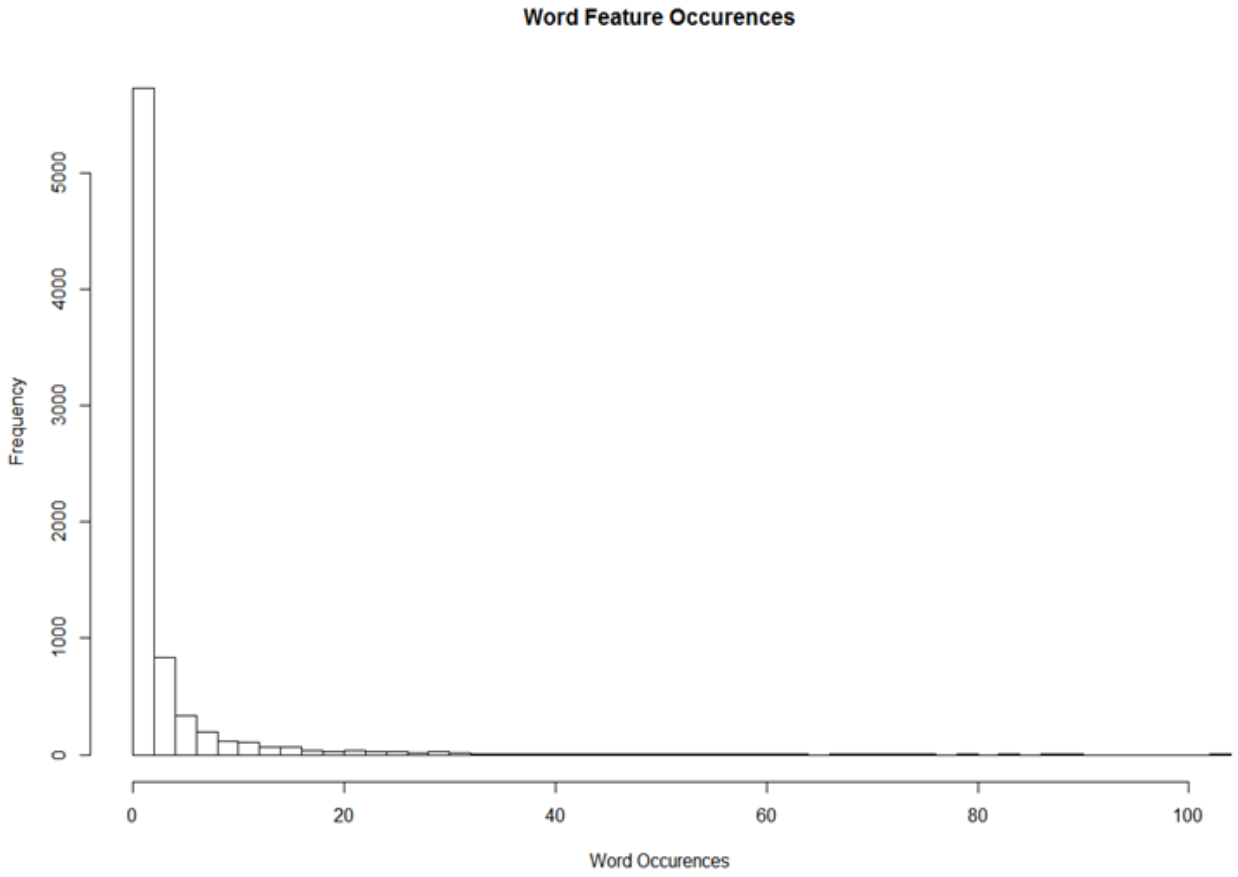
3. Feature Filtering

See the Appendix A for R code.

With the large number of features included in the full word matrix, computational costs in implementing Random Forests was a concern; having a feature size of 8000 meant that each tree of a Random Forest should be considering around $\sqrt{8000} \simeq 90$ features at each split. Furthermore, due to normalization these features are continuous, meaning at each split in each tree the algorithm must look at node impurity for a grid of values. As expected, attempting to train a Random Forest model on the unfiltered word feature matrix of this size took upwards of 5 hours (the procedure was cancelled before it could finish). Filtering was required to obtain a smaller data set that would allow us to tune our model using 10 fold cross validation in a realistic timeframe.

To reduce the number of word features, we decided to choose a threshold for the number of times a word appeared in the total training set before being considered in our model, reasoning that word features that appeared too infrequently would be less useful in classifying unseen data and simply slow down our computation.

We first converted our matrix into a simple count matrix. To do this, we simply counted the number of texts each word feature appeared in. We plotted these counts in the histogram below:

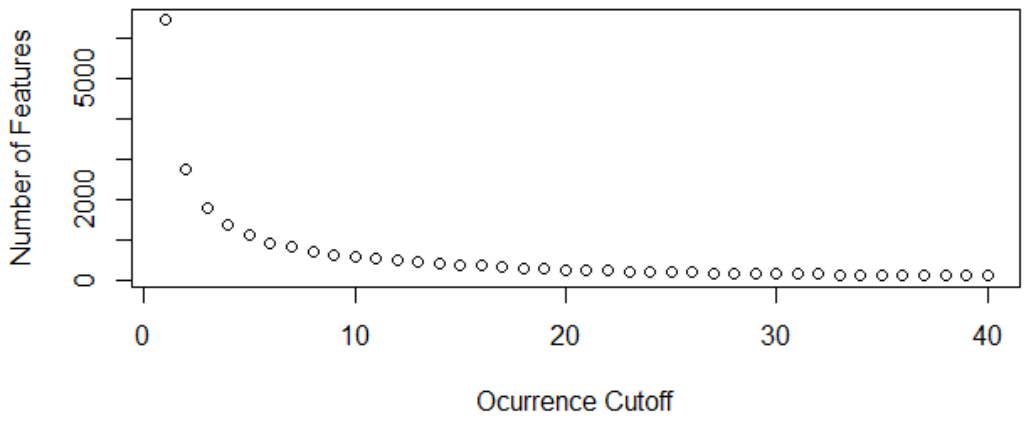


The histogram plots the number of word features (frequency) that appear x times in the corpus (where x is an integer on the x-axis)

Clearly, the distribution is heavily left-skewed, with a large number of features occurring 0 to 10 times, and very few occurring more than 20 times. The summary statistics are as follows:

Min.	**1st Qu.**	**Median**	**Mean**	**3rd Qu.**	**Max.**
1.000	1.000	1.000	4.423	3.000	406.000

We replotted this data below to show the cumulative number of features that would be included in the model for each cutoff (graph below).



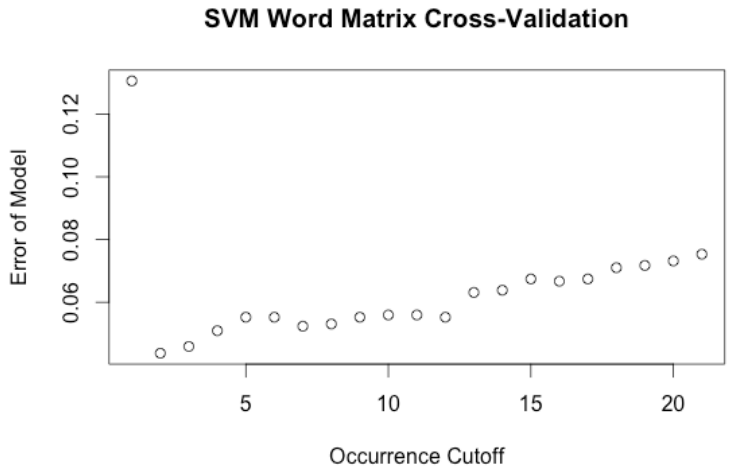
The plot shows the number of word features that occur *more than* x amount of times (for every integer x on the x-axis). Hence, it shows us how many word features our matrix would include for any frequency cutoff.

From the plot, we can see a sharp reduction in number of features from 0 to 5, eliminating about 7,000 of the total 8,000 features. At a threshold of 20, very few features remain.

Our first approach was to use 10-fold cross-validation on the training set with an SVM to estimate the best frequency threshold. We used CV to estimate the MSE for a range of candidate cutoffs and picked the cutoff that minimized error rates. We chose to use SVMs here because they are commonly used for text classification.

We tried to train SVM models in a 10-fold CV loop, but because in each iteration we had to run the "tune" function to pick the correct cost for the SVM model, which *also* required employing cross validation, the process was incredibly slow and inefficient. Eventually, we abandoned this idea, stopping the computation after getting preliminary results for thresholds 1 to 5.

In order to cut down on the computation required for this cross validation, we fixed the tuning parameter to the cost value that was picked in our first 5 runs of the original CV loop (cost = 0.001), and reran the cross validation on thresholds 0 through 20. The SVM models gave us the following test error rates for threshold values of 0 to 20:



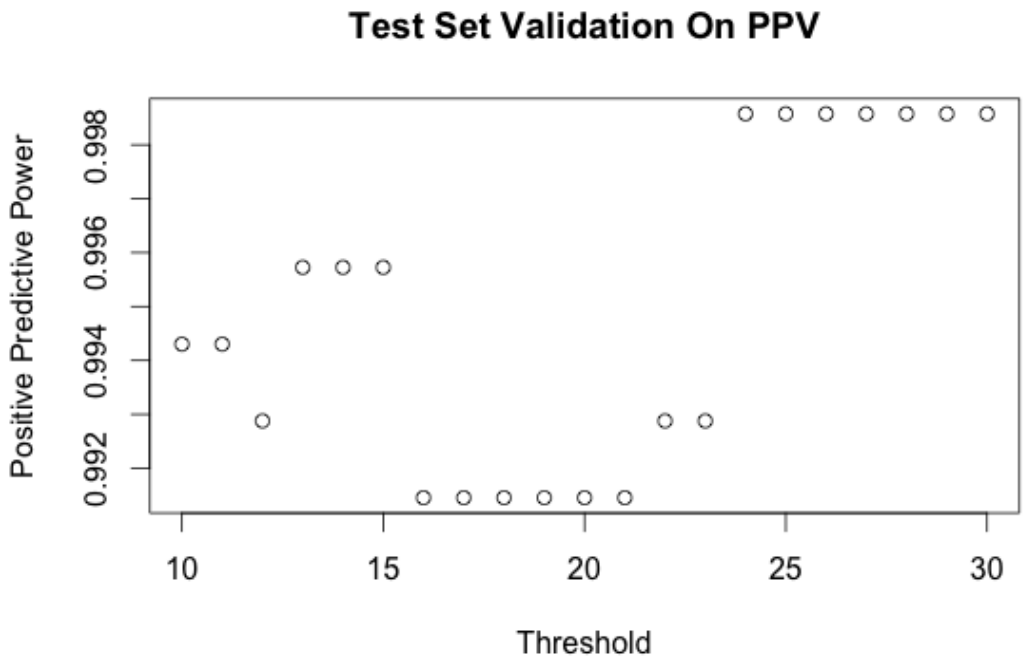
The actual error rates were as follows:

```
[1] 0.1305595 0.0437590 0.0459110 0.0509326 0.0552367 0.0552367 0.0523673 0.0530846
[9] 0.0552367 0.0559541 0.0559541 0.0552367 0.0631277 0.0638451 0.0674319 0.0667145
[17] 0.0674319 0.0710187 0.0717360 0.0731707 0.0753228
```

For the full model with no features eliminated, the error was 13.1%; removing features that occur only once resulted in a drop to an error rate of 4.4%; and for removal of any number of features more than one, there were very small increases in the error rate. Based on these results alone, it was difficult to tell which cutoff would give a good balance between a small error and better computation time.

We thus turned to a different measure of model accuracy to determine a better cutoff -- we considered the nature of the problem and the *type* of error that we care about more: false positive vs. false negative. Since in this situation misclassifying ham has more dire consequences than misclassifying spam, we decided to focus on minimizing our false negative rate (effectively, maximizing our PPV).

On the next page are the PPV rates that result from training each model on 80% of the given data and testing on the remaining 20%.




```
[1] 0.9943020 0.9943020 0.9928775 0.9957265 0.9957265 0.9957265 0.9914530 0.9914530
[9] 0.9914530 0.9914530 0.9914530 0.9914530 0.9928775 0.9928775 0.9985755 0.9985755
[17] 0.9985755 0.9985755 0.9985755 0.9985755 0.9985755 0.9985755
```

From this data, we can see that a threshold of 24 results in a maximum PPV of 99.9%, which remains constant for any higher cutoff. With this and computational considerations in mind, we choose 24 as our cutoff, resulting in a model that includes only features that appear in at least 5% of the training set spam texts -- overall, 216 of the original 8,000 features.

4. Power Feature Extraction

See Appendix B for R code.

The list of power features we included in our model (with explanations) are as follows:

1. Contains rate: text contains "/p". This is common in spam messages.
2. Contains PO Box: contains "PO Box" or similar. Common in spam.
3. Contains Long Number: contains any sequence of 5 or more digits (after removing punctuation). This includes phone numbers, zip codes, and texting numbers. We initially had individual features for phone numbers, zip codes, and texting numbers, but then decided this feature is all encompassing, still is uncommon in ham, and works on on different formats in the text messages. We use regex to in python to check this feature. This feature is common in spam.
4. Contains a pound sign: contains "£". Common in spam.
5. Contains a word with letters and numbers: e.g. "11dafg". Common in spam.

6. Contains service phrases: contains “reply stop”, “customer service”, or similar. Common in spam.
7. Number of all caps words: number of words of all capital letters after removing punctuation. e.g. “HELLO!”. Common in spam messages.
8. Contains Ampersand: contains “&”. Common in spam.
9. Contains Endearments: contains words like “babe” or “i love you”. Uncommon in spam.
10. Contains Okay: contains “ok” or “okay” or similar. Uncommon in spam.
11. Contains Swear Words: Uncommon in spam.
12. Number of Exclamation Marks: Higher for spam.
13. Number of Uppercase Letters: Higher for spam.
14. Contains Elipses: Contains “...”. Common for ham.
15. Ends With Question Mark: Text ends with “?”. Common for ham.
16. Contains Plus Sign: Common in spam.
17. Contains XXX: contains “XXX”. Common in spam.
18. Number Digits: Number of digits in text. Higher for spam.
19. Sentiment: The overall sentiment of the text. This feature is based on dictionaries of negative and positive English words. We count the number of ‘positive’ words in the message, subtract the number of ‘negative’ words, and call this the *overall sentiment* of the text. (See References for source.)
20. Contains Free: Contains “FREE” or “free!” or similar. Common in spam. Note that this feature looks for uppercase letters and punctuation and hence is not the same as the ‘free’ word feature.
21. Contains Texting Abbreviation: Contains “lol” or “ur” or similar. Common in ham.
22. Contains Website: contains “.com” or “http” or similar. Common in spam.
23. Contains Prize Words: Contains phrases like “win” and “to claim”. Common for spam.

When choosing which power features to include in our model, we used two main criteria. Firstly, the power feature had to occur often enough in the dataset (if the power feature was a boolean power feature such as ‘contains £‘); a feature that only appeared once or twice would have very little predictive power. We picked power features that occurred often (over 100 times) in the corpus. Secondly, the feature had to have a sufficient difference in how frequently it appeared in ham vs. spam messages.

To accomplish the second criterion, we looked at the distribution of potential power features across spam/ham text messages in the given training set (displayed in the table below). For boolean power features (e.g. ‘contains pound sign’), we compared the percent of spam messages that contains the feature with the percent of ham messages that contain the feature. For non-boolean power features (e.g. number of exclamation marks) we compared the average value across spam messages to the average across ham messages. We picked power features that had very different rates of occurrence or averages for the two classes and also occurred often enough in the dataset.

Power Feature	Ham	Spam	Boolean
containsRate	0.000000	0.113274	1
containsPOBox	0.000277	0.118584	1
containsLongNumber	0.000830	0.803540	1
containsPoundSign	0.001383	0.346903	1
containsNumberLetterWord	0.056708	0.778761	1
containsServicePhrases	0.001107	0.132743	1
numbersAllCapsWords	0.818257	2.959292	0
containsAmpersand	0.069710	0.196460	1
containsEndearments	0.030982	0.017699	1
containsOkay	0.017427	0.005310	1
containsSwearWord	0.042324	0.017699	1
numberExclaimations	0.180636	0.741593	0
numberUppercaseLetters	4.074965	16.037168	0
containsElipses	0.156570	0.007080	1
endwithQuestionMark	0.151591	0.007080	1
containsPlusSign	0.004149	0.122124	1
containsXXX	0.008299	0.031858	1
numberDigits	0.302905	16.100885	0
overallSentiment	0.115353	0.511504	0
containsFREE	0.000000	0.152212	1
containsTextingAbbrev	0.125588	0.169912	1
containsWebsite	0.003873	0.164602	1
numberPrizeWords	0.003596	0.539823	0

Note: For boolean features (Boolean=1), the Ham column gives rate of feature occurrence in training set ham messages and the Spam column does the same for spam messages. For non-boolean variables, the columns give the average result amongst messages of the two classes.

We hypothesize that the features with the largest differences in occurrence rates or averages between classes will be the most powerful features for classification.

We also checked the correlations between the features in the training set. A lot of these features are fairly correlated (notably, “number of digits” and “contains long number” have a correlation of 0.90). However, we decided not to remove correlated features, as they capture different characteristics of spam/ham messages (for instance “number of digits” captures the property spam messages often have of containing mixed number and letter strings. “Contains long number” captures the property that spam messages often have phone numbers or texting numbers). The correlation table was too large to include in this report, but can be viewed in our Google Drive folder (PF_correlations.PNG).

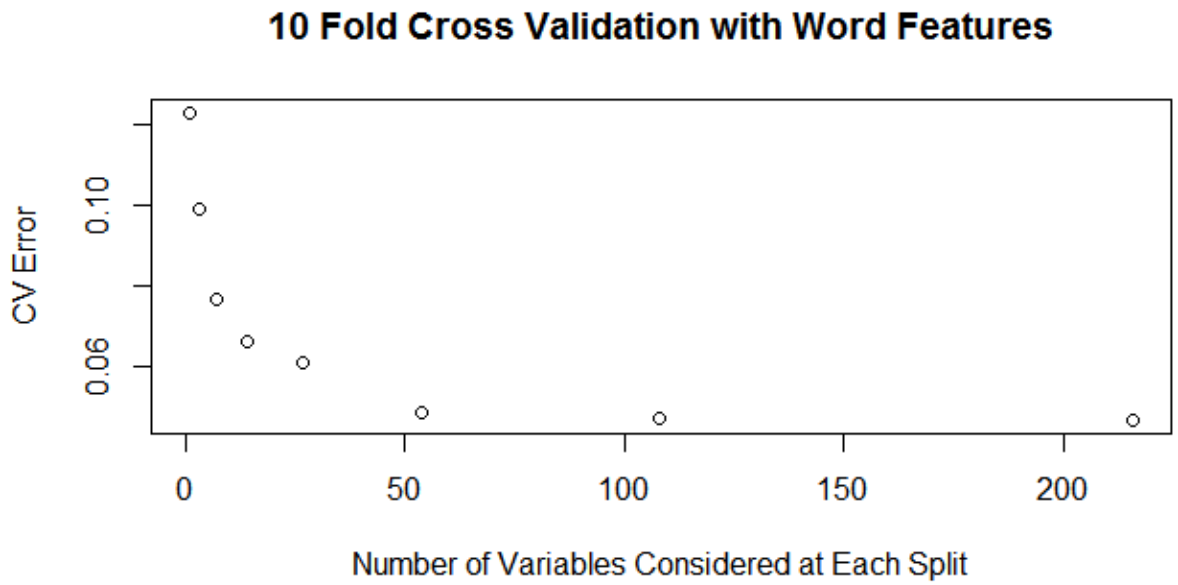
5. Word and Power Feature Combination

When combined, the word and power feature matrices produced a final matrix with 233 features and dimensions 4180 x 233.

6. Classification on filtered Word Feature matrix

See Appendix C for R code.

A Random Forest was trained on our filtered word feature matrix with the default of 500 trees, using 10-fold cross validation to determine the optimal number of features to select at each split. The results from the cross validation are displayed below:



The cross validation error evens out at about 50 variables, which is much larger than the estimated optimal number $\sqrt{216} \simeq 15$ suggested by the square root rule for Random Forest classification.

When running random forest, normally sqrt(p) is the default number of variables (parameter *mtry* in R) to choose at each node. From the graph we can see that $\sqrt{216} \simeq 15$ variables is not optimal and that error rates decline in the number of variables used at each node. We chose *mtry=50* as optimal as it achieves close to the lowest error rate. It is not worth using more variables as this is computationally costlier and does not seem to improve error rates.

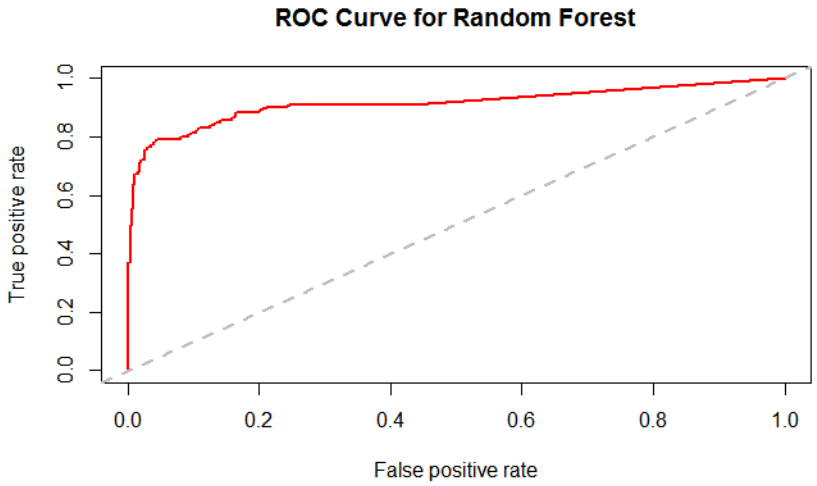
A potential explanation for the optimal number being higher than sqrt(p) is that many of the features are not frequently occurring or powerful features. Hence these sparse

features will not split the data and so by choosing a higher number of features, random forest will avoid using weak features.

Using the optimal number of features per split, 50, we trained random forest on 80% of the training data (the same 80% used to cross validate the number of features)and predicted the class of the remaining 20%. We achieved a 5.9% error rate and obtained this confusion matrix and ROC curve (with similar results on many trials):

predictions	0	1
0	706	38
1	11	81

Accuracy = .941, **PPV** = 0.88, **NPV** = 0.949, **Sensitivity** = 0.68, **Specificity** = 0.985



These results show that the word features are fairly weak spam predictors. The accuracy, specificity, and NPV of this test are sufficient, but the sensitivity is not even nearly enough to make this a useful spam classifier.

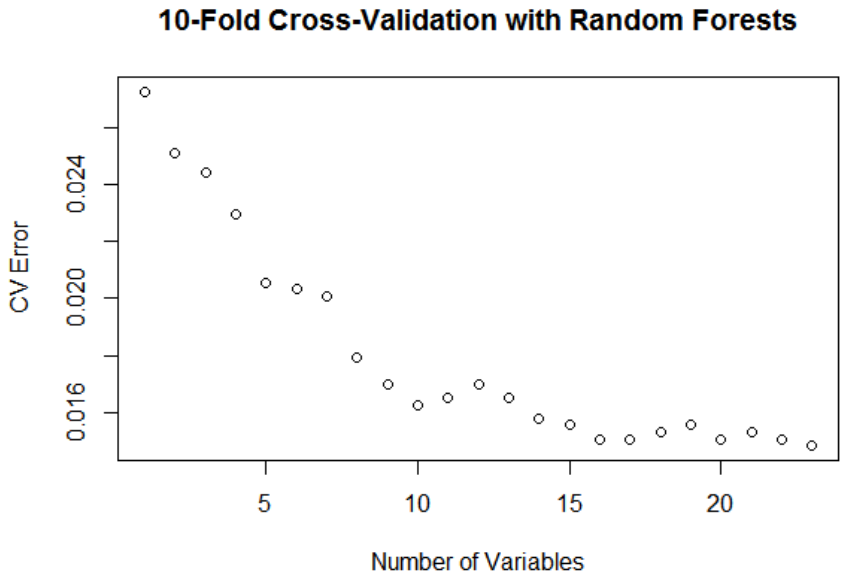
7. Verification

In the in class competition, the word feature-only model was used to predict a new set of unlabeled text messages. The test error for this prediction was 5.5%, similar to the validation error above.

8. Classification on Power Feature matrix

See Appendix D for R code.

The process of part 6 was repeated, this time on the matrix consisting solely of 23 power features. We performed 10-fold cross validation using Random Forests on 80% of the training data to select the number of variables to draw at each node when growing the tree. Here are the results:



Here is the data for the graph:

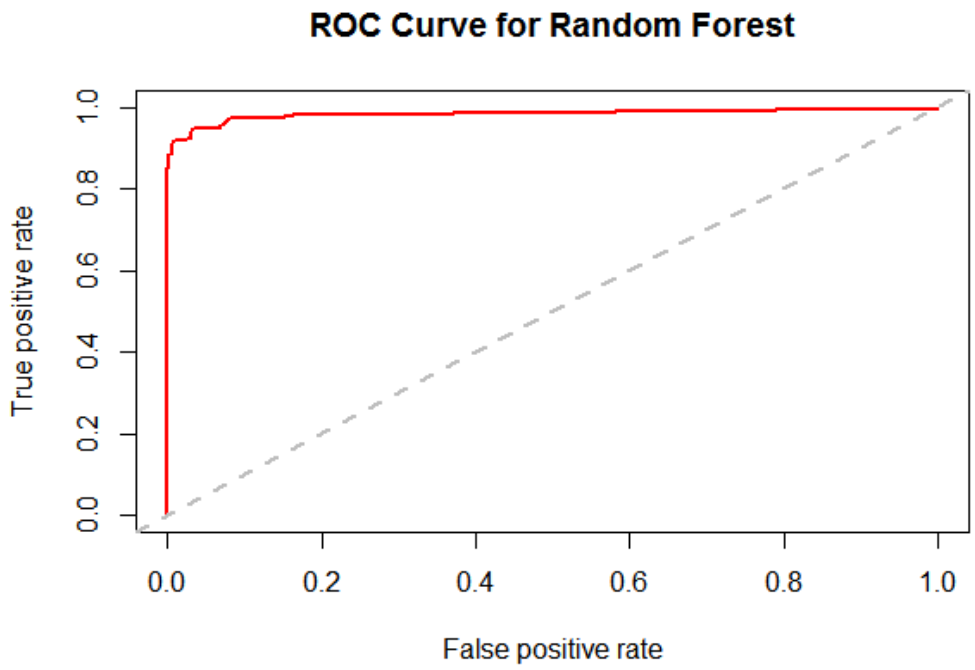
# variables	cv error rate
23	0.01483254
22	0.01507177
21	0.01531100
20	0.01507177
19	0.01555024
18	0.01531100
17	0.01507177
16	0.01507177
15	0.01555024
14	0.01578947
13	0.01650718
12	0.01698565
11	0.01650718
10	0.01626794
9	0.01698565
8	0.01794258
7	0.02009569
6	0.02033493
5	0.02057416
4	0.02296651
3	0.02440191
2	0.02511962
1	0.02727273

We chose *mtry=16* from this cross validation as it achieves the lowest error with the least variables, balancing accuracy and computational costs.

Using the optimal number of features, 16, we trained random forest on 80% of the training data and predicted the class of the remaining 20%. We achieved a 2% error rate and obtained this confusion matrix and ROC curve
:

predictions	**0**	**1**
0	711	12
1	5	108

Accuracy = .980, **PPV** = 0.955, **NPV** = 0.983, **Sensitivity** = 0.90, **Specificity** = 0.993



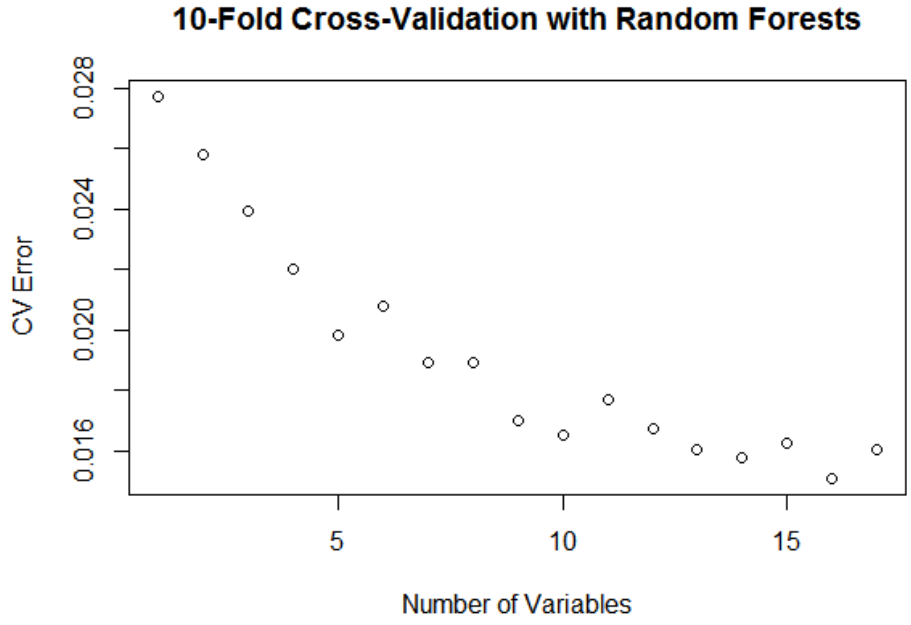
These results are noticeably better than those obtained in part 6 with the word-only matrix. All of the prediction metrics (accuracy, PPV, NPV, etc) have improved, especially the sensitivity and PPV, seeing as the word features already had a decent specificity. The results demonstrate that the power features are much more effective in targeting spam than the raw word features. This is to be expected as most of the power features were built to detect spam rather than ham. Power features are more characteristic of spam since spam is defined more by its form and special characters than ham messages are.

However, there was still room for improvement. Looking at the importances of each of the features (below), we can see that some features were not at all deemed important in classification.

	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
containsRate	3.001985e-03	5.538166e-04	2.676916e-03	2.4567310
containsPOBox	2.080485e-04	-1.077140e-06	1.790225e-04	0.1371387
containsLongNumber	5.151316e-02	1.467664e-01	6.421313e-02	300.2776742
containsPoundSign	4.188185e-03	-7.233242e-05	3.622834e-03	2.0719716
containsNumberLetterWord	4.208984e-03	6.385019e-03	4.499988e-03	28.1132305
containsServicePhrases	3.148847e-03	3.353771e-03	3.173249e-03	8.2292259
numbersAllCapsWords	3.411826e-03	9.897603e-03	4.273738e-03	12.5349532
containsAmpersand	3.096670e-04	-9.627211e-04	1.416355e-04	2.8539535
containsEndearments	1.133641e-05	9.214985e-08	9.805898e-06	0.4808506
containsOkay	1.687794e-05	0.000000e+00	1.457806e-05	0.2266424
containsSwearWord	5.658241e-06	2.286255e-04	3.559560e-05	0.8117401
numberExclaimations	7.358695e-04	3.349869e-03	1.079989e-03	5.1373276
numberUppercaseLetters	1.155898e-02	7.032598e-02	1.937297e-02	40.6627197
containsElipses	1.102059e-03	1.454865e-02	2.886846e-03	7.7641937
endwithQuestionMark	-8.548875e-05	3.580175e-02	4.720872e-03	2.3752927
containsPlusSign	1.800031e-09	1.228070e-05	1.640647e-06	0.2322543
containsXXX	2.747254e-04	1.570959e-04	2.594055e-04	0.9102968
numberDigits	3.611145e-02	2.154929e-01	5.986789e-02	312.3267856
overallSentiment	7.892335e-04	1.523075e-03	8.884873e-04	10.7715950
containsFREE	1.394359e-02	4.766828e-03	1.271176e-02	11.1863788
containsPrizeWords	3.023743e-03	2.326240e-03	2.927640e-03	4.1925072
containsTextingAbbrev	7.595593e-04	-3.192081e-04	6.152343e-04	2.3770397
containsWebsite	2.554203e-03	4.876703e-03	2.860749e-03	6.1462375

From the table, it is clear that "containsPOBox", "containsEndearments", "containsOkay", "containsSwearWord", "containsPlusSign", and "containsXXX" appear to have little predictive power (relative to the other features. They have less than 1/300 the importance of "numberDigits" and "containsLongNumber"). Hence, we eliminated these variables from our model and tried again with only 17 variables.

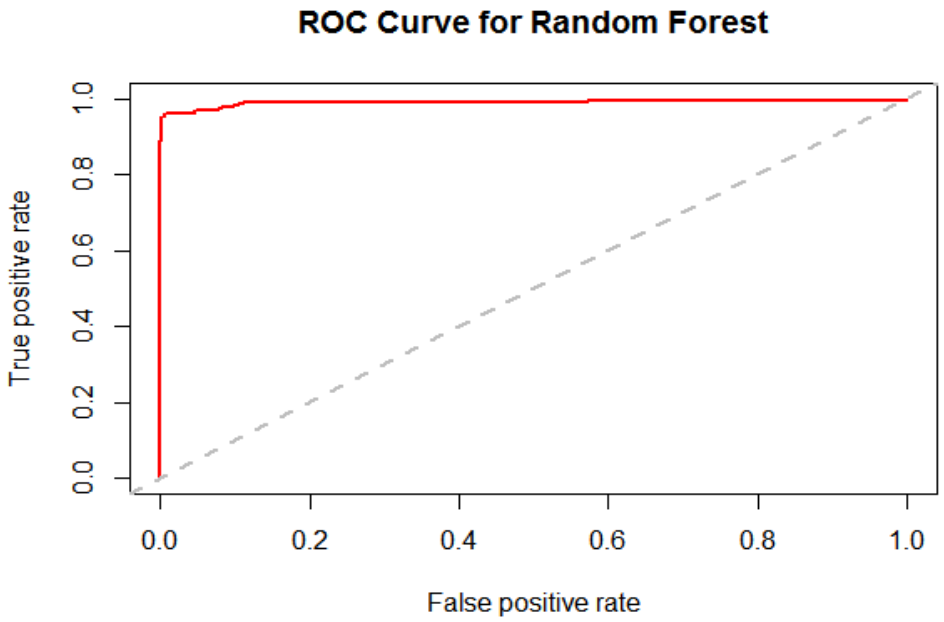
Running 10-fold cross validation, we obtained the following graph of error rates (below). We chose 10 as the optimal number of predictors random forest chose at each node. This is because the error rate at 10 is low enough and increasing the *mtry* further does not reduce error rates significantly and would result in ineffective (at classifying data) nodes in the tree and higher computational costs.



When training random forest with *mtry=10* on 80% of the training data and predicting on the remaining, we got improved error rates (averaging about 1.2%). We obtained the confusion matrices and ROC curves below:

predictions	**0**	**1**
0	708	6
1	3	119

Accuracy = .989, **PPV** = 0.975, **NPV** = 0.991, **Sensitivity** = 0.925, **Specificity** = 0.995



From our improved error rates (as well as sensitivity, specificity, PPV, and NPV), it is clear that eliminating 'unimportant' features was beneficial to our model. Our new variable importances still showed that some variables were significantly more powerful than others, and given more time, we think that tweaking the power features to only included extremely powerful ones would be best.

Feature importances:

	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
containsRate	0.0090267566	0.0008547923	0.0079483483	5.010918
containsLongNumber	0.0488945622	0.1263252072	0.0590870131	260.916582
containsPoundSign	0.0150073347	0.0005529918	0.0130946767	9.642641
containsNumberLetterWord	0.0030408468	0.0074741897	0.0036203620	40.133435
containsServicePhrases	0.0053902415	0.0057698791	0.0054406186	11.159536
numbersAllCapsWords	0.0022917816	0.0080286730	0.0030386616	13.937870
containsAmpersand	0.0005124128	-0.0003084909	0.0004038121	4.322662
numberExclaimations	0.0002974216	0.0038605617	0.0007667468	5.709159
numberUppercaseLetters	0.0037937993	0.0714772244	0.0127113613	48.899973
containsElipses	0.0006856497	0.0143862501	0.0024993070	6.135807
endwithQuestionMark	0.0002789074	0.0290727667	0.0040616284	4.138460
numberDigits	0.0412398534	0.2215232115	0.0646747154	305.908922
overallSentiment	0.0019001386	0.0015291226	0.0018499952	10.276969
containsFREE	0.0085247811	0.0012337815	0.0075601935	5.159503
containsTextingAbbrev	0.0002567236	0.0001756692	0.0002441975	2.390995
containsWebsite	0.0063192394	0.0108397182	0.0069122449	11.673611
numberPrizeWords	0.0021385998	0.0029808781	0.0022463636	5.620350

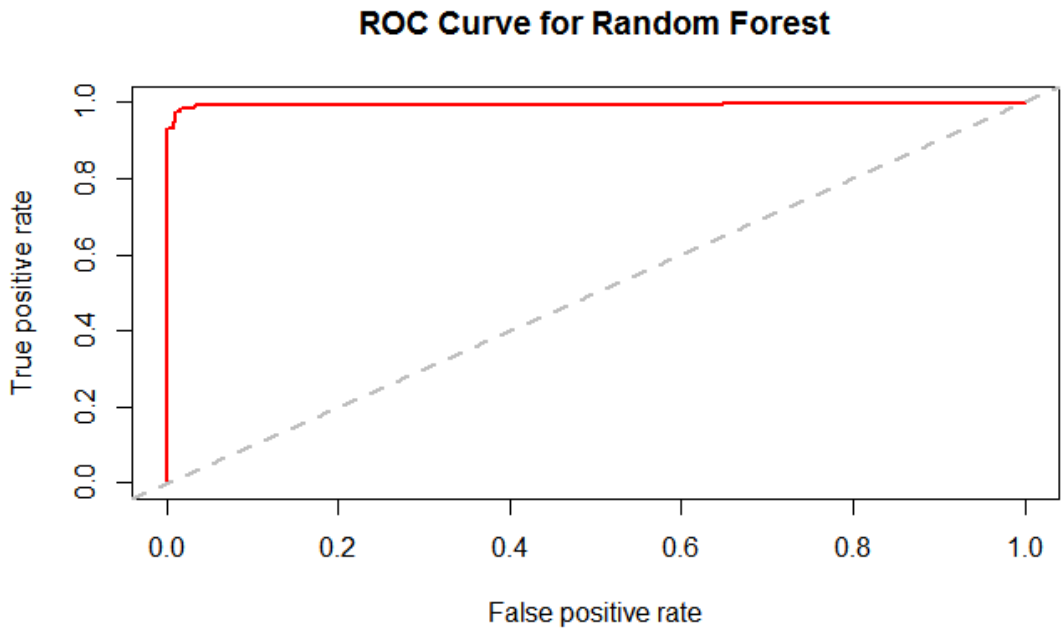
9. Classification on Combined Feature matrix
See Appendix E for R code.

Finally, a Random Forest classifier was trained on the combined word and power feature matrix.

Once again, using 10-fold cross validation, we chose to look at 50 variables for each split, the same number that was obtained by performing cross validation on the word-only model.

predictions	0	1
0	716	12
1	1	107

Accuracy = .984, **PPV** = 0.991, **NPV** = 0.984, **Sensitivity** = 0.899, **Specificity** = 0.999



Combining the two models, we originally expected the results to improve upon those of the individual models. However, the sensitivity decreased slightly. This was likely due to the fact that the power features were drowned out by the bevy of word features. Random Forests are resistant to overfitting, but one pitfall of the model may be that as the number of "trash" features in a model increases, more effective features will be trained on less frequently.

On the other hand, there was a significant improvement in the PPV of the power features from adding word features.

Comparison of results

Out of the three models, only the two containing power features achieve comparably low error rates, Between the two models, one containing only power features and one containing all features, there is a clear trade-off between sensitivity and PPV.

Bearing in mind our initial assertion that it is more important to correctly label ham messages than spam, we assert that, because PPV is the more important metric of the two, it continues to be our most important metric in making decisions regarding our model. Thus, we chose the combined feature matrix to predict on the test set, as it had the highest PPV while still maintaining a significant sensitivity.

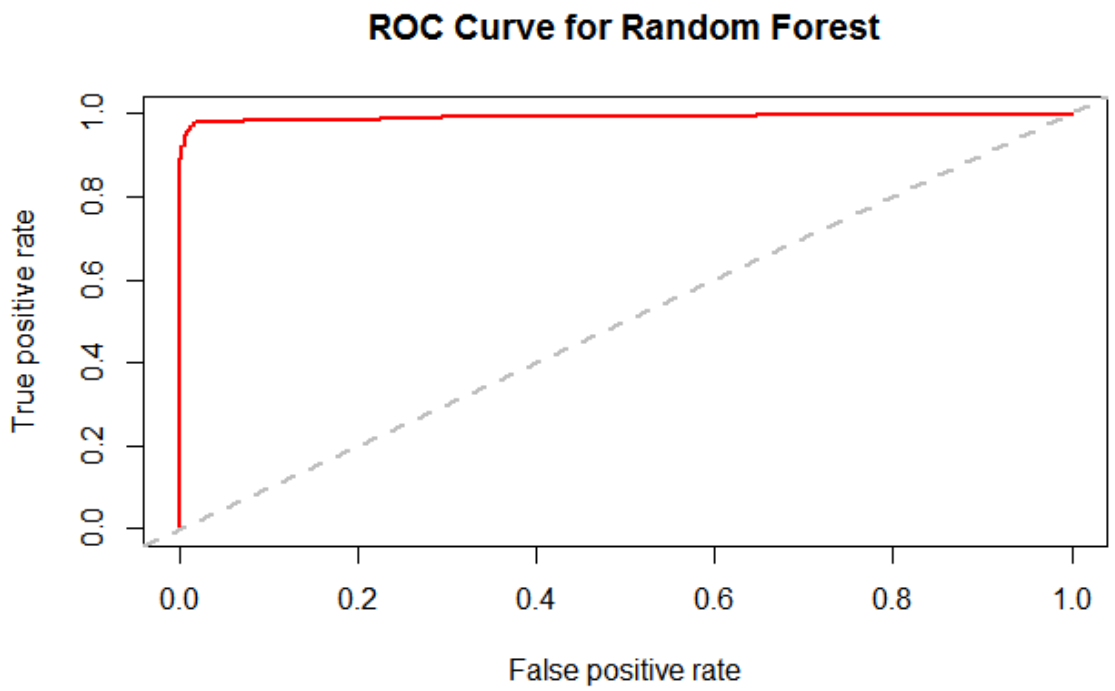
10. Final model and Results of Validation Set

See Appendix F for R code.

Running the combined word and power feature classifier on the test set of 1394 text messages resulted in the following table.

predictions	0	1
0	1208	16
1	4	166

Accuracy = .986, **PPV** = 0.986, **NPV** = 0.987, **Sensitivity** = 0.912, **Specificity** = 0.997



These results show that the final classifier is fairly effective, identifying 91% of the spam messages while misclassifying only 4 of the 1300 ham messages.

The PPV is the important statistic here, and a higher PPV would be desired, since in this case 1.4% (~20) of the spam labels were actually ham. In an actual application of spam targeting, this value would need to be much lower.

Testing these models informed us on the difference between power features and word features. We came to the conclusion that selected power features are much more powerful than raw word features. Automated spam prediction, informed purely by the words that appear in messages, is clearly less effective than a classifier that looks at some of the higher-level features of messages.

That being said, the raw word features seemed to have a beneficial effect on the PPV of the models. We would expect raw word features to have a better effect on predicting ham, since there will be a large variety of words in a given set of text messages, and raw word features can explain that variance.

11. References

Bing Liu, Minqing Hu and Junsheng Cheng. "Opinion Observer: Analyzing and Comparing Opinions on the Web." Proceedings of the 14th International World Wide Web conference (WWW-2005), May 10-14, 2005, Chiba, Japan.

12. Appendix A: R Code for Feature Filtering

```
df <- read.csv("C:/Users/Geoff Kaufman/Favorites/Dropbox/Stat 154 Final
Project/dfg.csv", header=FALSE)
unique <- read.csv("C:/Users/Geoff Kaufman/Favorites/Dropbox/Stat 154 Final
Project/uniqueg.csv", header=FALSE)
PF <- read.csv("C:/Users/Geoff Kaufman/Favorites/Dropbox/Stat 154 Final
Project/matrix_selectedPF.csv", header=TRUE, stringsAsFactors=FALSE)
#create list of unique words
unique = lapply(unique, as.character)
unique = unlist(unique)
#assign list as column names of the dataframe
colnames(df) = unique[1:(length(unique)-2)]
#extract labels and remove them from the word matrix/unique vector
label = ifelse(df[,24] > 0, 1, 0)
summary(label)
```

```
df = df[,c(-1,-24)]
unique = unique[c(-1,-24)]

#pfs is the power feature matrix sans labels
PFs = PF[,c(-1,-2)]

##################################################Normalization/Filterin
g

#create columns which show simply whether or not a word appeared
#in a sentence, as opposed to a count. For use in calculating frequency
#and eliminating colinearity
occurs = apply(df,2, function(x) x > 0)

#colinearity: eliminate columns(words) that appear together 100% of the time
duplicates=which(duplicated(occurs, MARGIN = 2))
occurs=occurs[,-c(duplicates)]
df = df[,-c(duplicates)]

#creat frequncy counts
num_occurance = apply(occurs,2, sum)

#function to remove words that are below a given frequency threshold
remove_words = function(threshold, dataframe){
boolean_vec = num_occurance < threshold

 return (dataframe[,-(which(boolean_vec == TRUE))])
}

#normalize
row_sums = apply(df,1, sum)
perc = df / row_sums

######################### Test different  thresholds
acc = c()                          # a threshold of removing words that
for (j in 10:30){                      # occur less than a certain amount of
 data = remove_words(j,perc)             # times. The data is then split into
                                # training (.8) and test (.2) sets
 train_data = data[(1:(0.8*nrow(data))),]    # of the data and labels to accompany.
 train_label = label[(1:(0.8*nrow(data)))]   # An svm model is then tuned over
                                # a range of cost parameters and the
 test_data = data[-(1:(0.8*nrow(data))),]    # best model is chosen. The model is then
 test_label = label[-(1:(0.8*nrow(data)))]   # run on the training data using the predict
                                # function. In different iteration of the code
 model_data = data.frame(x=train_data,       # either, accuracy or PPV,
```

```
  y=as.factor(train_label))                  # is appended to a list for the user to chose
  tune.mod = tune(svm, y~., data = model_data,#  the most appropriate word count
threshold.
  kernel = 'linear', ranges = list(cost= (0.001, 0.01,0.1,1,5)))
  bestmod = tune.mod$best.model              #
                                  #
  test_dat = data.frame(x =test_data,        #
       y = as.factor(test_label))          #
                                  #
  y = predict(bestmod, newdata=test_dat)     #
  acc = append(acc,(table(y,test_label)[4]/   #
  sum(table(y,test_label)[3]+table(y,test_label)[4])))
}


#choose cutoff of 24
words = remove_words(24, perc)
```

13. Appendix B: R Code for Power Feature Extraction

The creation of the power feature matrices was all done in Python.

The only power feature extraction code done in R was the code to create a table of correlations for all power features (given power feature matrix saved as a csv). This is below:

```
#load PF matrix csv
data = read.csv(filename.csv)
#cut out class label and row label columns
data=data[,-c(1,2)]
#function to produce table. I found on a blog.
corstarsl <- function(x){
  library(Hmisc)
  x <- as.matrix(x)
  R <- rcorr(x)$r
  p <- rcorr(x)$P

  ## define notions for significance levels; spacing is important.
  mystars <- ifelse(p < .001, "***", ifelse(p < .01, "** ", ifelse(p < .05, "* ", " ")))

  ## trunctuate the matrix that holds the correlations to two decimal
  R <- format(round(cbind(rep(-1.11, ncol(x)), R), 2))[,-1]

  ## build a new matrix that includes the correlations with their apropriate stars
```

```
  Rnew <- matrix(paste(R, mystars, sep=""), ncol=ncol(x))
  diag(Rnew) <- paste(diag(R), " ", sep="")
  rownames(Rnew) <- colnames(x)
  colnames(Rnew) <- paste(colnames(x), "", sep="")

  ## remove upper triangle
  Rnew <- as.matrix(Rnew)
  Rnew[upper.tri(Rnew, diag = TRUE)] <- ""
  Rnew <- as.data.frame(Rnew)

  ## remove last column and return the matrix (which is now a data frame)
  Rnew <- cbind(Rnew[1:length(Rnew)-1])
  return(Rnew)
}
library(xtable)
x=xtable(corstarsl(data))
#save table to file
print.xtable(x, type="latex", file="corr.tex")
print.xtable(x, type="html", size="small",file="corr_small.html")
```

14. Appendix C: R Code for Classification on Word Feature matrix

```
########################################## Training, Word Features Only

#remove rows with all stopwords, as power features are not involved
hasNan = apply(words,1,function(x) any(is.nan(x)))
words1 = words[!hasNan,]

#build dataframe
data = data.frame(y=as.factor(label), words1)

#partition train/test
index = seq(1,ceiling(0.8*nrow(data)))
training = data[index,]
test = data[-index,]

#cross validate on number of features at each split
cv.mtry = rfcv(training[,-1], training$y, cv.fold=10)
plot(cv.mtry2$n.var,cv.mtry2$error.cv)

#train the random forest with optimal mtry=50
model = randomForest(y~., data=training, mtry=50)
```

```
#predict new class values
ypred = predict(model, type="class", newdata=test, na.action=na.omit)

#calculate test error
sum(ifelse(ypred!=test$y, 1, 0), na.rm=TRUE)/length(ypred)

#build confusion matrix
predictions = ypred
table(predictions, test$y)

#plot roc curve using ROCR lib
rf.pr = predict(model,type="prob",newdata=test)[,2]
rf.pred = prediction(rf.pr, test$y)
rf.perf = performance(rf.pred,"tpr","fpr")
plot(rf.perf,main="ROC Curve for Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")
```

15. Appendix D: R Code for Classification on Power Feature matrix

Code for fitting Random Forests and making predictions. Here, we do 10-fold CV on 80% of the training set to pick the best *mtry* for Random Forest (and plot results). Using this best *mtry*, we make predictions on the remaining 20% of the training set given. We draw an ROC curve and print a confusion matrix and error rate.
Code:

```
#load PF matrix from file
data = read.csv("filename.csv")
#remove row labels
data=data[,-1]
library(randomForest)
#shuffle data
data = data[sample(nrow(data)),]
#pick training set (80%) and test set (20%)
train = data[1:floor(nrow(data)*0.8),]
test = data[(floor(nrow(data)*0.8)+1):nrow(data),]
#ten fold CV to find best mtry (number of variables Random Forest draws at each node).
#Try every mtry from 1 to p
#CV is done only on the training set (80% of given training set)
rf.cv = rfcv(train[,-1],as.factor(train[,1]),cv.fold=10,scale="non.log",step=-1)
#graph CV results
with(rf.cv, plot(n.var, error.cv, xlab="Number of Variables", ylab="CV Error",
main="10-Fold Cross-Validation with Random Forests"))
#print error rates for different mtry
data.frame(rf.cv$error.cv)
```

```
#fit model on training set (80%)
#use best mtry found from CV (10)
rf.model = randomForest(x=train[,-1],y=as.factor(train[,1]),mtry=10,importance=TRUE)
#print feature importances
rf.model$importance

#draw ROC curve for model
slibrary(ROCR)
rf.pr = predict(rf.model,type="prob",newdata=test[,-1])[,2]
rf.pred = prediction(rf.pr, test[,1])
rf.perf = performance(rf.pred,"tpr","fpr")
plot(rf.perf,main="ROC Curve for Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")

#make predictions on test set (20% of training)
#print confusion matrix and error rate
preds = predict(rf.model,type="class",newdata=test[,-1])
table(preds,test[,1])
sum(ifelse(preds==test[,1],0,1))/length(preds)
```

16. Appendix E: R Code for Classification on Combined Feature matrix

```
################################### Word + PF combo
PF <- read.csv("C:/Users/Geoff Kaufman/Favorites/Dropbox/Stat 154 Final
Project/matrix_selectedPF.csv", header=TRUE, stringsAsFactors=FALSE)
#set NA values to zero. stopword rows are not removed since power features may exist
words[is.na(words)]=0

#build data frame, repeat steps above for new data frame
data2 = data.frame(y=as.factor(label), words2, PFs)


index2 = seq(1,ceiling(0.8*nrow(data2)))
training2 = data2[index2,]
test2 = data2[-index2,]

cv.mtry2 = rfcv(training2[,-1], training2$y, cv.fold=10)
plot(cv.mtry2$n.var,cv.mtry2$error.cv)

model2 = randomForest(y~., data=training2, mtry=50)
ypred2 = predict(model2, type="class", newdata=test2, na.action=na.omit)
sum(ifelse(ypred2!=test2$y, 1, 0), na.rm=TRUE)/length(ypred2)
```

```
predictions = ypred2
table(predictions, test2$y)


rf.pr2 = predict(model2,type="prob",newdata=test2)[,2]
rf.pred2 = prediction(rf.pr2, test2$y)
rf.perf2 = performance(rf.pred2,"tpr","fpr")
plot(rf.perf2,main="ROC Curve for Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")
```

17. Appendix F: R Code for Final model and Results of Validation Set