Part 1 should describe the feature creation, filtering and power feature creation steps. Give a list of your power features in a table (name, description). Remember power feature creation is optional. Mention how many you created (if any). Part 2 should describe should describe feature selection (optional) and classifier building steps (RF and SVM are a must to include, if you used another classifier - be sure to describe it here and give any references). Part 3 should describe clustering results and commentary. Part 4 - should describe what you did to lock your model with final # of features, tuning parameters for getting ready to predict on the test set. If you used RF or SVM, fine. If you want me to consider the results of another model - then include why, what model (you should also describe this Part 3). Also upload your results from this *new model* as predict2.txt (only one allowed per group and this is completely optional). You won't have the final test set accuracy (for either predict.txt or predict2.txt) - we will compute those.

**Describe the Data**

The training set contains 3505 emails. The class distributions are displayed below

|  |  |  |
| --- | --- | --- |
| Sender | # Emails | Frequency |
| 1 | 685 | 19.54% |
| 2 | 1023 | 29.19% |
| 3 | 1241 | 35.41% |
| 4 | 271 | 7.85% |
| 5 | 281 | 8.01% |

**Part 1**

We first used Python to read in the raw data HRC\_test.tsv. After that, we separated the email, and for each email, we separated its label and content. We then tokenized the content of each email to separate the words, digits, and punctuations. We used the nltk library to stem the words use stop-words to get rid of commonly-used words. After that, we counted the number of occurrences for all features in each email and made a dictionary in the form of {word1 : count1, word2 : count2,…}. If a email does not contain wordi then its value of counti  will be 0. We then filtered out words that occur less than 10 times.

We tried to get rid of digits and punctuations, we found that the accuracy is better with digits and punctuations, so we keep them. We also tried to use the “enchant” package in R to get rid of gibberish to reduce the number of features, but had also found it to be ineffective.

For power features, we added the number of characters of email to the feature matrix. We also added the number of unique words of each email to the feature matrix.

We also tried to use phrases as power features. Because the misclassification rate is more than 30% for sender 1 and sender 4 compared to around 15% for all other senders, we tried to look for phrases that sender 1 and sender 4 used often. We found that there’s no phrases that sender 4 used much more often than other senders, and sender 1 likes to use phrases like "benghazi libya", “wave attack”, and “ambassador stevens”. So we used indicators for these phrases. For email that contains these phrases, the indicator is 1, otherwise 0. These indicators increased the accuracy by roughly 1%, so they are not that useful. Thus, we did not add these power features to our feature matrix.

After the steps above, the dimension of our final matrix is 3505 by 7234.

|  |  |
| --- | --- |
| step | Total # of features |
| Raw | 3505 by 50943 |
| Remove Stop words | 3505 by 39712 |
| Stemming | 3505 by 30604 |
| Filter out words that occur less than 10 times | 3505 by 7232 |
| Adding power features | 3505 by 7234 |

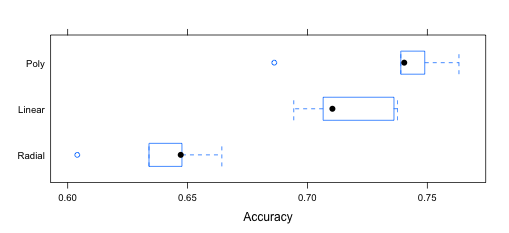
**Part 2**

Support Vector Machine:

Because we have 5 classes instead of 2 classes, we need to choose between the one-versus-one method and the one-versus-all method. Since we only have 5 classes, we use the one versus one method for SVM. That is, we fit k choose 2 pairwise classifiers. For each observation, we assign it to class that wins the most pairwise competitions.

We have to decide between linear, polynomial, and radial kernel as well as their corresponding tuning parameters cost, cost and degree, and cost and gamma.

The plot below shows boxplots of the accuracy of the three kernels. Polynomial kernel is shown to have a higher accuracy, but its minimum is quite low so it is unstable. I used 5-fold cross validation to compare polynomial kernel and linear kernel, and confirmed that linear kernel almost always outperformed polynomial kernel. For its simplicity and its relatively high accuracy, we choose to use linear kernel for SVM.



After we decide to use linear kernel, we use 5-fold cross validation to choose the tuning parameter *cost* that gives the highest accuracy. The costs we try are 0.0001, 0.001, 0.01, 0.1, 1, 10, and 100. As shown in the plot below, cost 0.01 gives the highest accuracy for linear kernel.

5-fold Cross Validation Accuracy Scores

|  |  |
| --- | --- |
| C (error penalty) | Accuracy |
| C = 100 | 0.7210 |
| C = 10 | 0.7210 |
| C = 1 | 0.7210 |
| C = 0.1 | 0.7372 |
| **C = 0.01** | **0.7708** |
| C = 0.001 | 0.7079 |
| C = 0.0001 | 0.3784 |
| C = 0.00001 | 0.5315 |

Finally, we decide to use the linear kernel with cost 0.01 for SVM, which gives 95% accuracy and the confusion matrix below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| step | Total # of features used | Total Accuracy  (xx%) | Accuracy per sender class  xx%,yy%,zz%,aa%,bb% | Top Ten features by importance |
| SVM | 2362 | 85% | Class 1: 76%  Class 2: 85%  Class 3: 94%  Class 4: 67%  Class 5: 88% | Sid  Party  Tories  Most  Labour  Leader  Xpress  Cingular  Conservatives  polls |

Confusion Matrix for Linear SVM with C = 0.01

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted 1 | Predicted 2 | Predicted 3 | Predicted 4 | Predicted 5 |
| Actual 1 | 524 | 29 | 128 | 1 | 3 |
| Actual 2 | 39 | 867 | 111 | 2 | 4 |
| Actual 3 | 35 | 36 | 1164 | 6 | 0 |
| Actual 4 | 4 | 9 | 80 | 182 | 0 |
| Actual 5 | 5 | 2 | 27 | 1 | 246 |

only 20 most important variables shown (out of 2361)

Overall

sid 0.10000

fw 0.08973

h 0.04880

b5 0.04676

X.s 0.04018

party 0.03790

his 0.03673

subject 0.03648

tories 0.03531

most 0.03238

labour 0.03221

leader 0.03196

xpress 0.03185

cingular 0.03159

conservatives 0.03148

poll 0.03126

only 0.03110

cameron 0.03059

into 0.03030

mini 0.03017

ROC: one vs all

Feature reduction: remove feature that has high correlation