Spam Emails - PSTAT 131 Final Report

Jalen Souksamlane, Cindy Wong, Angel Chen

March 08, 2020

Introduction

Certain emails are automatically considered spam by algorithms in email systems. These algorithms track patterns of certain keywords, which allows them to classify spam and non-spam emails. This dataset, spambase.data, allows us to explore the connection between the frequency of keywords and whether an email is classified as spam. If such a connection exists, it would be more convenient for email users because their spam emails will be filtered out before reaching them.

Our question: Is there a relationship between the predictors (frequency of keywords/characters, length of sequences, number of capital letters) and whether an email is considered spam or not? If so, which predictors affect the response?

Data

There are 57 predictor variables in our dataset and they are all numeric variables since the predictors are mainly frequency of words/characters and length. The response variable is "spam". It is binary; 0 indicates non-spam email and 1 indicates spam email.

- Predictor variables
  - WORD = percentage of words in the email that match WORD. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string. E.g. "remove" is the percentage of words in the email that matches the word "remove".
  - CHAR = percentage of characters in the email that match CHAR. E.g. "dollar.sign" is the percentage of characters in the email that matches "$".
  - capital_run_length_average = average length of uninterrupted sequences of capital letters
  - capital_run_length_longest = length of longest uninterrupted sequence of capital letters
  - capital_run_length_total = total number of capital letters in the e-mail

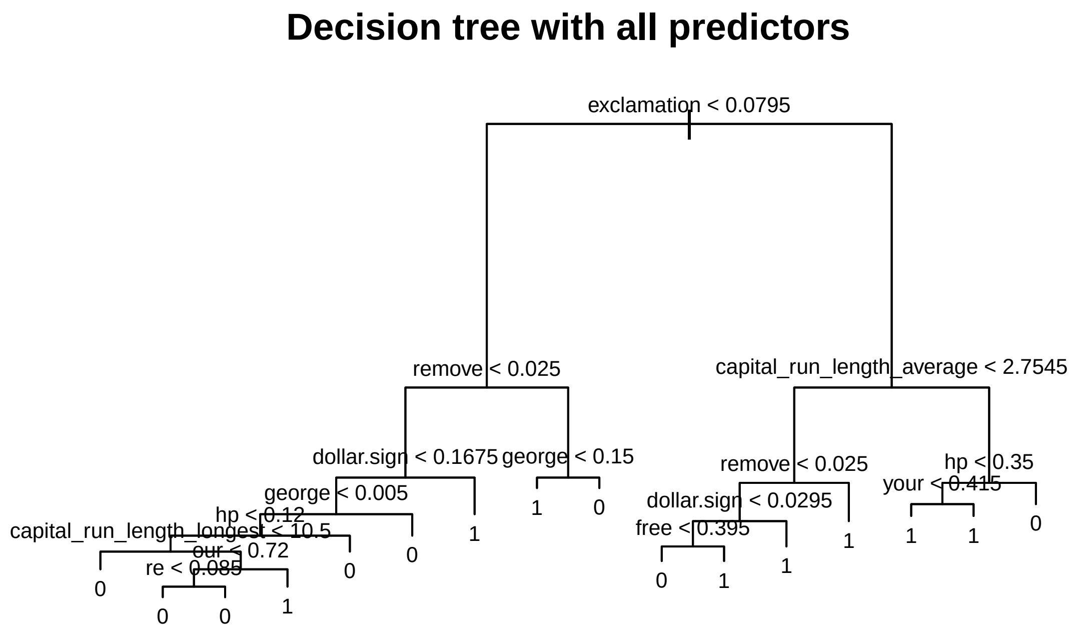
In total, spambase.data has 4601 rows and 58 columns.

Exploratory Analysis

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

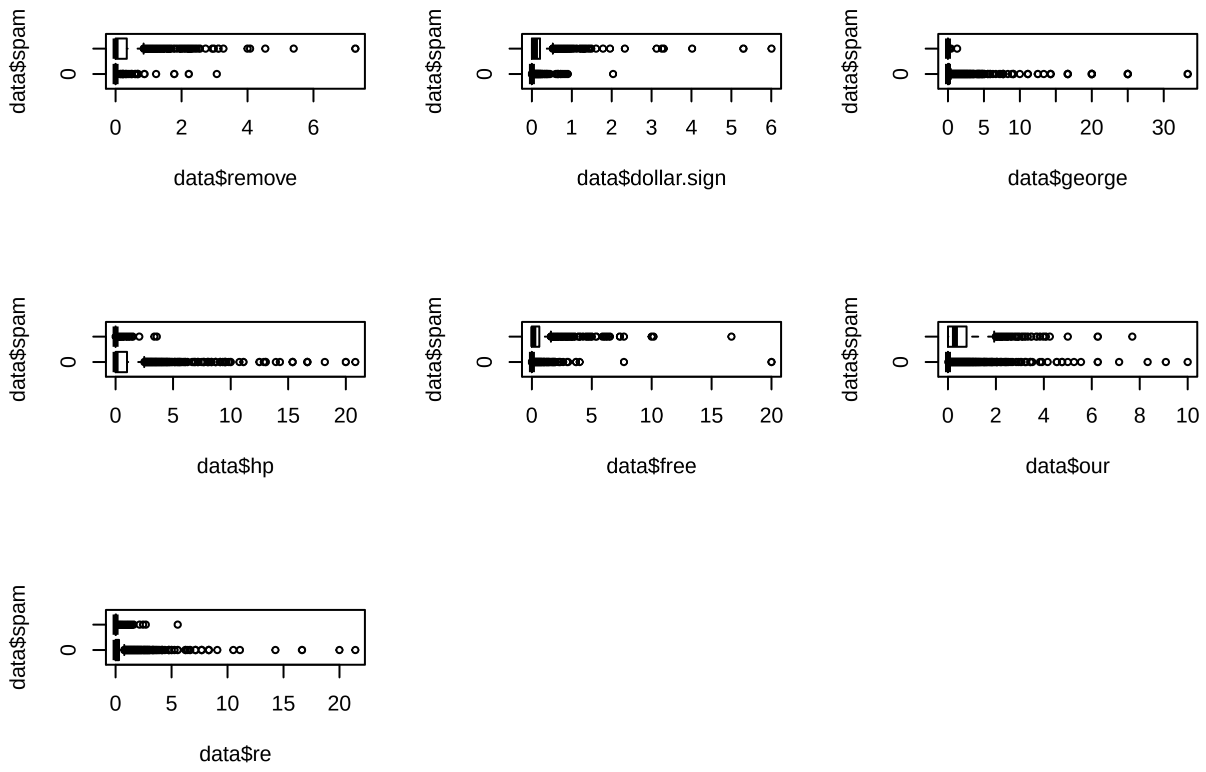
```
##
## Call:
## glm(formula = spam ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0476  -0.1908   0.0000   0.1105   4.0799
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -1.482e+00  1.977e-01  -7.498 6.50e-14 ***
## make                     -7.219e-01  3.414e-01  -2.114 0.034507 *
## address                  -1.915e-01  1.050e-01  -1.823 0.068234 .
## all                       1.046e-01  1.446e-01   0.723 0.469503
## X3d                       2.701e+00  2.884e+00   0.936 0.349147
## our                       5.770e-01  1.443e-01   3.999 6.35e-05 ***
## over                      1.319e+00  3.585e-01   3.681 0.000232 ***
## remove                    2.266e+00  4.463e-01   5.078 3.81e-07 ***
## internet                  5.082e-01  2.411e-01   2.108 0.035028 *
## order                     4.119e-01  3.394e-01   1.214 0.224904
## mail                      1.833e-01  1.082e-01   1.695 0.090105 .
## receive                  -2.628e-02  3.738e-01  -0.070 0.943952
## will                     -3.024e-01  1.119e-01  -2.702 0.006902 **
## people                   -2.296e-01  3.020e-01  -0.760 0.447165
## report                    3.236e-02  1.540e-01   0.210 0.833603
## addresses                 2.997e+00  1.153e+00   2.600 0.009325 **
## free                      9.247e-01  1.920e-01   4.816 1.46e-06 ***
## business                  1.268e+00  3.629e-01   3.495 0.000475 ***
## email                     1.166e-01  1.392e-01   0.838 0.402151
## you                       7.716e-02  4.708e-02   1.639 0.101224
## credit                    1.985e+00  8.908e-01   2.228 0.025851 *
## your                      2.192e-01  7.399e-02   2.963 0.003049 **
## font                      1.339e-01  2.204e-01   0.608 0.543433
## X000                      1.808e+00  5.641e-01   3.206 0.001348 **
## money                     1.644e+00  5.017e-01   3.276 0.001052 **
## hp                       -1.670e+00  3.498e-01  -4.775 1.80e-06 ***
## hpl                      -8.791e-01  5.001e-01  -1.758 0.078758 .
## george                   -6.698e+00  1.889e+00  -3.545 0.000392 ***
## X650                      3.953e-01  2.054e-01   1.925 0.054262 .
## lab                      -2.699e+00  1.897e+00  -1.423 0.154669
## labs                     -1.003e+00  5.489e-01  -1.827 0.067724 .
## telnet                   -8.983e+00  4.408e+00  -2.038 0.041572 *
## X857                      5.130e-01  3.191e+00   0.161 0.872256
## data                     -7.993e-01  4.239e-01  -1.885 0.059372 .
## X415                      6.616e-01  1.534e+00   0.431 0.666150
## X85                      -4.152e+00  1.661e+00  -2.499 0.012455 *
## technology                1.172e+00  4.171e-01   2.810 0.004958 **
## X1999                    -2.166e-01  2.744e-01  -0.789 0.429898
## parts                    -6.363e-01  5.249e-01  -1.212 0.225466
## pm                       -7.138e-01  4.625e-01  -1.543 0.122716
## direct                   -1.898e-01  4.772e-01  -0.398 0.690866
## cs                       -4.741e+02  2.249e+04  -0.021 0.983184
## meeting                  -2.313e+00  9.477e-01  -2.441 0.014647 *
## original                 -3.575e+00  1.887e+00  -1.894 0.058233 .
## project                  -1.105e+00  6.039e-01  -1.830 0.067241 .
## re                       -8.117e-01  1.926e-01  -4.214 2.51e-05 ***
## edu                      -2.219e+00  4.874e-01  -4.552 5.32e-06 ***
## table                    -2.995e+00  3.164e+00  -0.946 0.343938
## conference               -3.622e+00  2.159e+00  -1.678 0.093443 .
## semi.colon               -1.175e+00  5.716e-01  -2.055 0.039890 *
## parenthesis              -4.246e-01  3.913e-01  -1.085 0.277951
## bracket                  -1.829e+00  1.603e+00  -1.141 0.253880
## exclamation               2.278e-01  7.066e-02   3.224 0.001262 **
## dollar.sign               4.230e+00  8.496e-01   4.979 6.38e-07 ***
## pound                        1.498e+00  1.772e+00   0.845 0.397921
## capital_run_length_average  8.863e-02  5.979e-02   1.482 0.138281
## capital_run_length_longest  1.139e-02  3.751e-03   3.037 0.002393 **
## capital_run_length_total    3.423e-04  2.407e-04   1.422 0.154942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3719.0  on 2759  degrees of freedom
## Residual deviance: 1060.7  on 2702  degrees of freedom
## AIC: 1176.7
##
## Number of Fisher Scoring iterations: 23
```

Doing some exploratory analysis on our data, we found that extremely significant predictors include our, over, remove, free, business, hp, george, re, edu, and dollar.sign.



Doing a decision tree on all the predictors reveals that exclamation, remove, capital_run_length_average, dollar.sign, george, hp, your, free, capital_run_length_longest, our, and re are significant as well.

The 7 predictors that are significant in both logistic regression and the decision tree are remove, dollar.sign, george, hp, free, our, and re. We take a closer look by plotting boxplots.



It seems that there are many outliers across all the boxplots. In spam emails, there are higher average frequencies of remove, dollar.sign, free, our. In non-spam emails, there are higher average frequencies of george, hp, and re.

Methods

We plan to use Logistic Regression, Decision Trees, Bagging, Random Forests, Boosting, Support Vector Machines, and k-NN to classify our data. We will use cross-validation to tune any parameters in our model. To begin, we split our data as such: the training set (60%), validation set (20%), and test set (20%).

Model Building (Discussion)

Logistic Regression

```
error_log
```

```
## [1] 0.075
```

```
error_aic
```

```
## [1] 0.07391304
```

For logistic regression, we started out with all 57 predictors in our model, which gave us a warning message that there was complete separation. We chose to continue on without doing anything because we could not find any suitable solutions. Then, we used AIC to shrink our full logistic regression model with the step() function. When we did backward selection, it resulted in a logistic regression model with 45 predictors. Our full model yielded a validation error rate of 7.5% and our reduced model (from AIC) yielded a validation error rate of 7.4%.

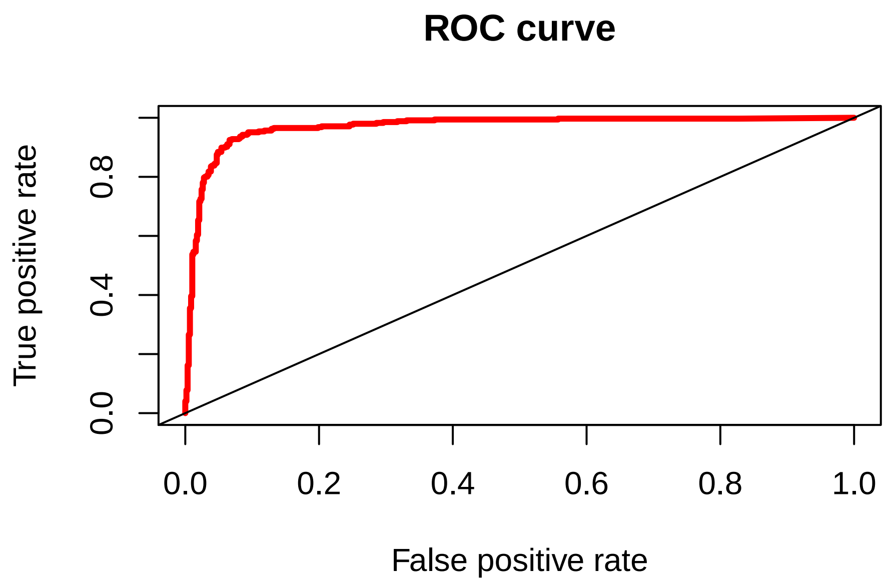


Figure 1: The ROC curve for logistic regression hugs the top left corner, which is what we want

Decision Tree

```
##          truth
## tree_pred   0   1
##         0 536  44
##         1  38 302
```

```
best_cv
```

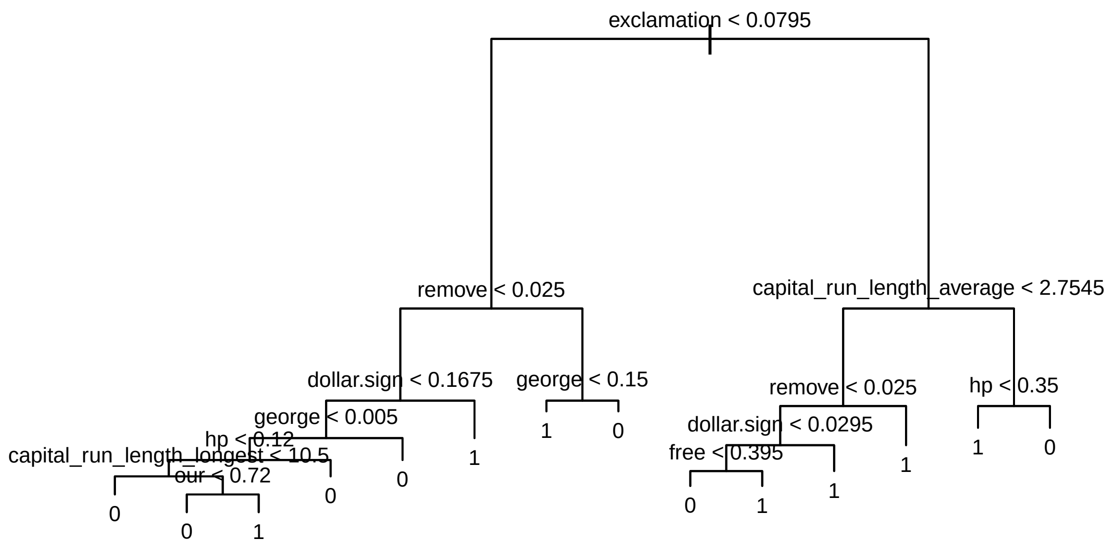
```
## [1] 14
```

```
error_tree
```

```
## [1] 0.08913043
```

Earlier, we fitted a decision and plotted it. Then, we used 10-fold cross validation to select the best tree size, which turned out to be 14 terminal nodes. Once we have the best tree size, we pruned the tree to achieve the optimal size. By doing a 10-fold CV on our decision tree, our final model yielded a validation error rate of 8.9%.

Pruned tree of size 14



Bagging

```
##          truth
## bag_pred   0   1
##        0 553  31
##        1  21 315
```

```
error_bag
```

```
## [1] 0.05652174
```

We moved onto the bagging method. We tried bagging with 500 trees and plotted the data, and it seemed to work better than the decision trees. It resulted in a validation error rate of around 5.7%, which is an improvement over the validation error rate from doing a 10-fold CV on decision tree.

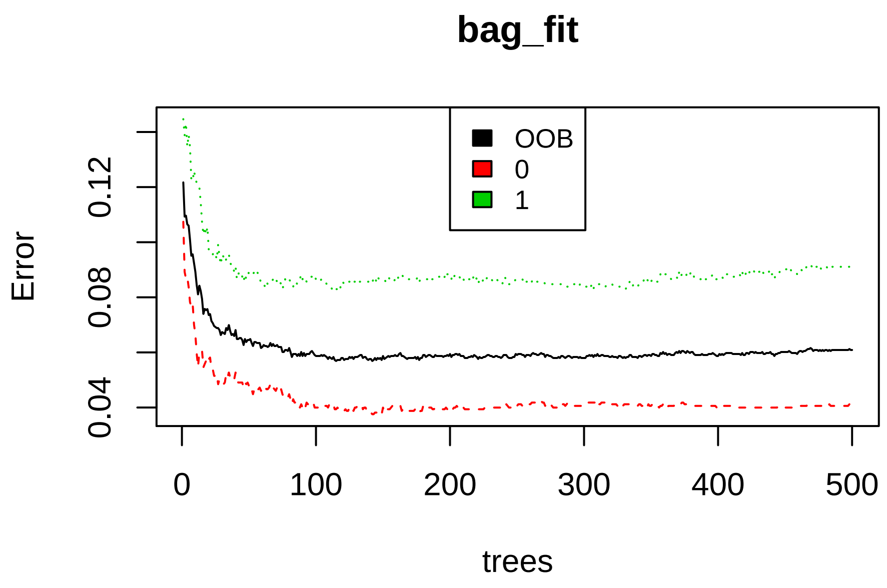


Figure 2: In bagging, classification error decreases with more trees and the OOB error rate is around 6%

Random Forest

```
##             truth
## forest_pred   0   1
##           0 553  26
##           1  21 320
```

```
error_rf
```

```
## [1] 0.05108696
```

Then, we did a random forest with 500 trees, so that we can compare to the bagging method. Because we are dealing with classification, we use the square root of 57 as the number of variables we use for splitting. This method results in a validation error rate of 5.1%, which is a slight improvement over the validation error rate from bagging with 500 trees.

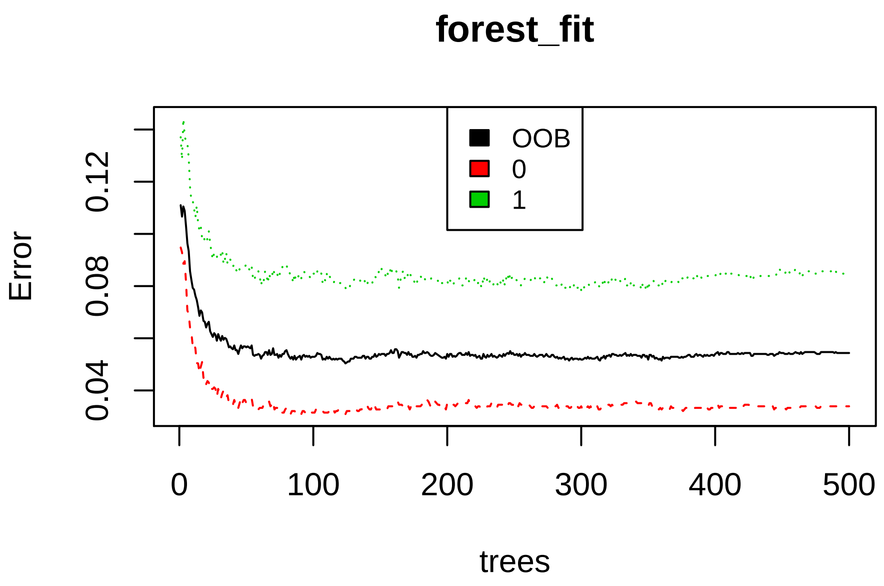
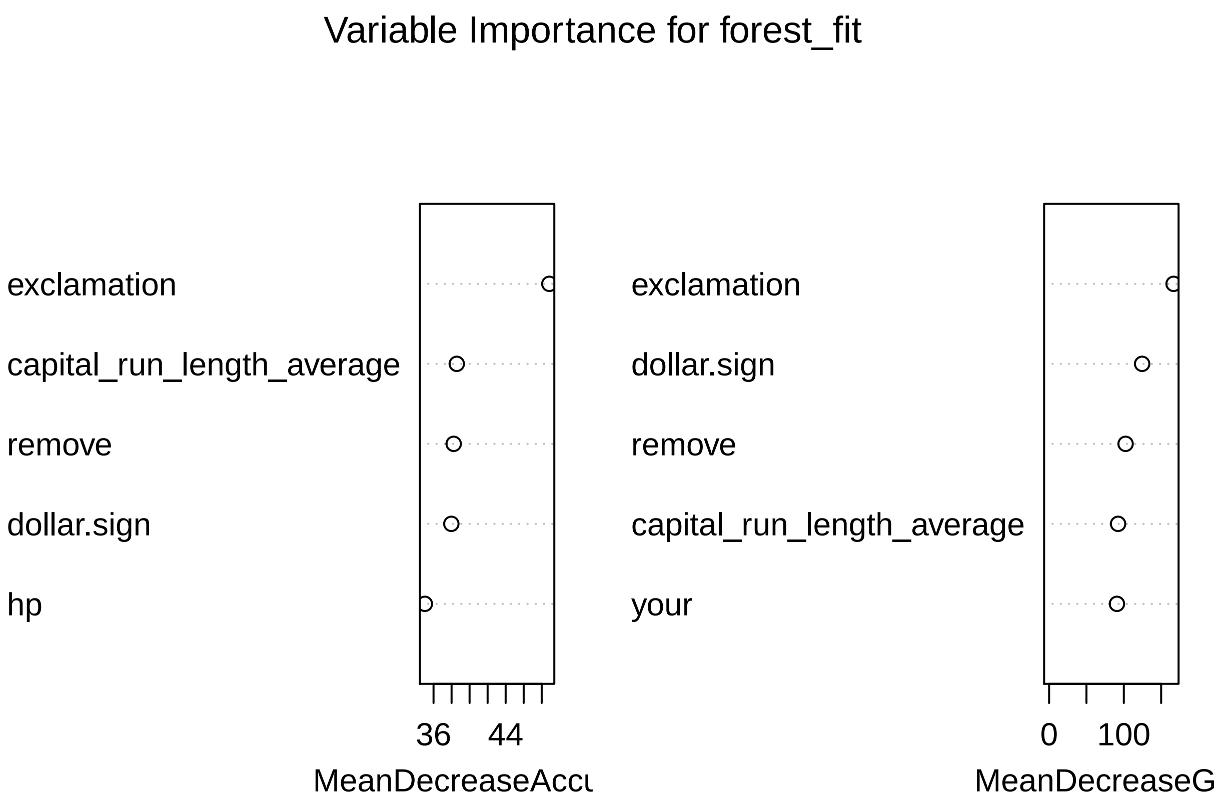
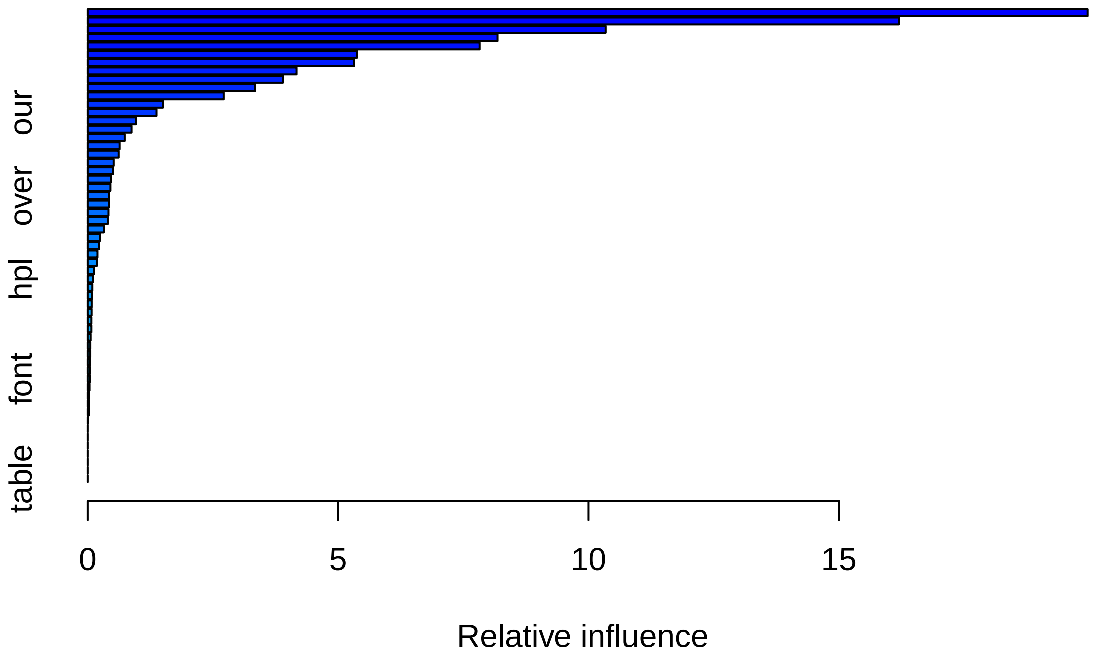


Figure 3: In random forest, classification error decreases with more trees and the OOB estimate of the error rate is around 5.4%



Across all of the trees in the random forest, exclamation is the most important variable in terms of model accuracy and Gini index. The predictors, capital_run_length_average and dollar.sign are important, as well.

Boosting



```
##                                                   var   rel.inf
## exclamation                               exclamation 19.967502
## dollar.sign                               dollar.sign 16.200538
## remove                                         remove 10.343266
## your                                             your  8.184165
## hp                                                 hp  7.827049
## capital_run_length_average capital_run_length_average  5.380529
```

This summary output for the boosting method shows us that once again, exclamation is the most important predictor.

```
##           truth
## boost_pred   0   1
##          0 545  21
##          1  29 325
```

```
error_boost
```

```
## [1] 0.05434783
```

Boosting with 500 trees resulted in a validation error rate of about 5.4%.

Support Vector Machines (Linear)

For support vector machines with all predictors, the code did not run due to the large number of predictors. Therefore we decided to proceed with the 7 most significant variables we found earlier. These 7 variables were significant in both logistic regression and the decision tree.

We try a linear kernel first.

```
##
## Call:
## svm(formula = spam ~ remove + dollar.sign + george + hp + free +
##     our + re, data = train, kernel = "linear", cost = 0.1, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  1044
##
##  ( 523 521 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.1282609
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2771739 0.02015497
## 2 1e-02 0.1692029 0.01691253
## 3 1e-01 0.1423913 0.01547120
## 4 1e+00 0.1311594 0.01697709
## 5 5e+00 0.1300725 0.01503126
## 6 1e+01 0.1300725 0.01550886
## 7 1e+02 0.1282609 0.01379140
```

To start off, we scaled the data and chose a random cost of 0.1. Then to find the best cost, we used cross-validation with the tune() function. When cost = 100, the error rate is the lowest.

```
##
## Call:
## best.tune(method = svm, train.x = spam ~ remove + dollar.sign + george +
##     hp + free + our + re, data = train, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
##
## Number of Support Vectors:  893
##
##  ( 441 452 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

We can see that from the model with the best cost, the number of support vectors is lower compared to the first model. The first model have 1044 support vectors while the best model have 893.

```
##        truth
## predict   0   1
##       0 551  83
##       1  23 263
```

```
error_svm_lin
```

```
## [1] 0.1152174
```

The error rate for the linear model is 11.5%

Support Vector Machines (Radial)

We try a radial kernel next.

```
##
## Call:
## svm(formula = spam ~ remove + dollar.sign + george + hp + free +
##     our + re, data = train, kernel = "radial", gamma = 1, cost = 1,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  971
##
##  ( 507 464 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.1043478
##
## - Detailed performance results:
##      cost gamma     error dispersion
## 1  1e-01   0.5 0.1177536 0.02254608
## 2  1e+00   0.5 0.1086957 0.01791352
## 3  1e+01   0.5 0.1043478 0.01830019
## 4  1e+02   0.5 0.1068841 0.01941780
## 5  1e+03   0.5 0.1076087 0.02259778
## 6  1e-01   1.0 0.1253623 0.01631106
## 7  1e+00   1.0 0.1101449 0.01856921
## 8  1e+01   1.0 0.1076087 0.01816017
## 9  1e+02   1.0 0.1054348 0.02059878
## 10 1e+03   1.0 0.1083333 0.02223338
## 11 1e-01   2.0 0.1398551 0.01841144
## 12 1e+00   2.0 0.1123188 0.01638244
## 13 1e+01   2.0 0.1094203 0.01789723
## 14 1e+02   2.0 0.1115942 0.01645351
## 15 1e+03   2.0 0.1181159 0.01895789
## 16 1e-01   3.0 0.1572464 0.01817222
## 17 1e+00   3.0 0.1112319 0.01584383
## 18 1e+01   3.0 0.1119565 0.01711826
## 19 1e+02   3.0 0.1170290 0.01902318
## 20 1e+03   3.0 0.1202899 0.01983031
## 21 1e-01   4.0 0.1713768 0.02266223
## 22 1e+00   4.0 0.1130435 0.01662987
## 23 1e+01   4.0 0.1130435 0.01773348
## 24 1e+02   4.0 0.1210145 0.01856921
## 25 1e+03   4.0 0.1242754 0.02233810
```

After scaling the data, we choose a random cost and gamma value (both are 1). Then we used the tune() function again to choose the best cost and gamma through cost validation. When cost = 10 and gamma = 0.5, the error rate is the lowest.

```
##
## Call:
## best.tune(method = svm, train.x = spam ~ remove + dollar.sign + george +
##     hp + free + our + re, data = train, ranges = list(cost = c(0.1,
##     1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
```

```
## Number of Support Vectors:  793
##
##  ( 405 388 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Once again, the number of support vectors has decreased. We went from 971 to 793 support vectors.

```
##        truth
## predict   0   1
##       0 530  54
##       1  44 292
```

```
error_svm_rad
```

```
## [1] 0.1065217
```

The error rate for the radial model is 10.7%, which is better than the linear model's error rate.

K-Nearest Neighbors

```
##          observed
## predicted   0   1
##         0 541  56
##         1  33 290
```

```
error_knn
```

```
## [1] 0.09673913
```

For k-Nearest Neighbors, we first created a response vector and design matrix with the training and validation sets. To choose the best value for k for our model, we used Leave-one-out Cross Validation. In order to do this, we created a loop to find the validation error for all values of k from 1 to 50. Leave-one-out cross validation found k=5 to be the model with the lowest error rate (9.7%). However, this validation error rate was not the lowest out of all other methods.

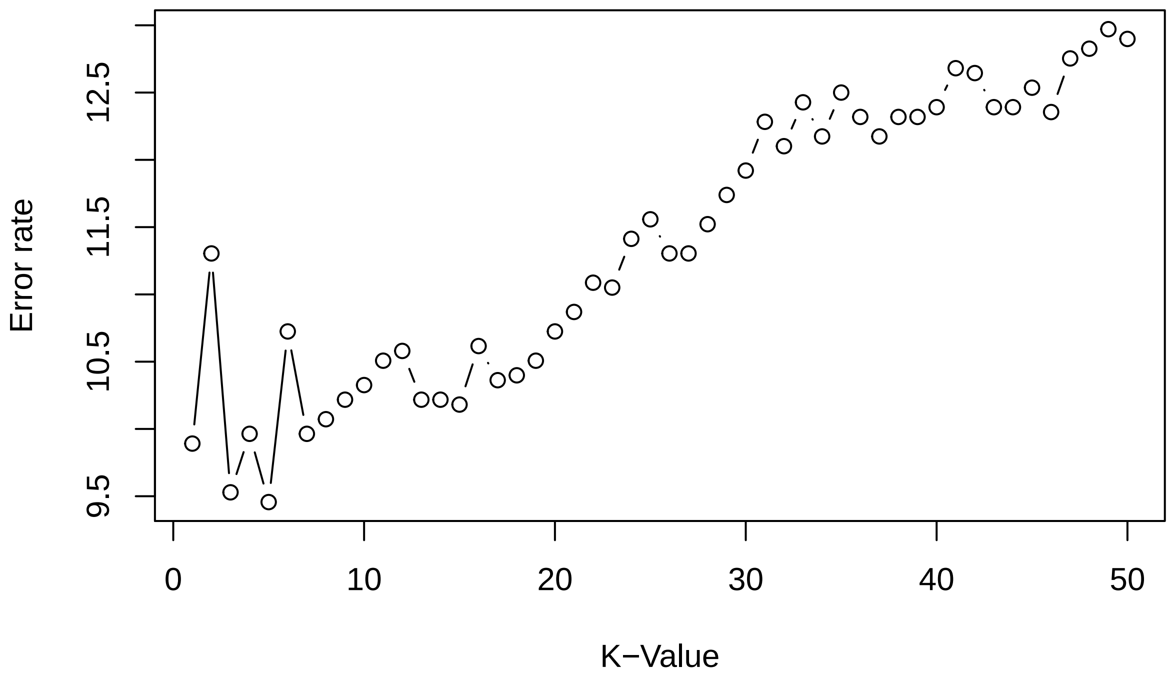


Figure 4: Classification error varies with different numbers of neighbors. Five neighbors gives us the lowest error rate.

Comparing Validation Errors

```
##                                 Method Error_Rate
## 1    Logistic Regression (FullModel) 0.07500000
## 2 Logistic Regression (ReducedModel) 0.07391304
## 3                      Decision Tree 0.08913043
## 4                            Bagging 0.05652174
## 5                      Random Forest 0.05108696
## 6                           Boosting 0.05434783
## 7                        SVM(Linear) 0.11521739
## 8                        SVM(Radial) 0.10652174
## 9                                KNN 0.09673913
```

By comparing the validation error rate of all methods, our final model is Random Forest, it yields a validation error rate of 5.1%, which is the lowest among the methods we have used.

Conclusions

Since we got the lowest validation error rate with random forest, we chose this method as our final model.

```
##             truth
## forest_pred   0   1
##           0 545  30
##           1  18 328
```

```
## [1] 0.05211726
```

The test error rate for our random forest is 5.2%, which is a small increase from the validation error rate of 5.1%.

Study Limitation

Even if we were able to determine the relationship and correlation between certain keywords and spam type, it is inevitable to have false positives. Keywords do not suffice to be the sole factor in determining whether an email is spam or not.

Potential research direction

Instead of single keywords, would different combinations of keywords included in an email yield better prediction in determining if the email is spam or not? What are the other possible key factors that could improve accuracy?

References

Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt of Hewlett-Packard Labs (1999). Spambase Data Set. http://archive.ics.uci.edu/ml/datasets/Spambase

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Appendix

```r
#reading in data
names <- c('make','address','all','3d','our','over','remove','internet','order','mail','receive', 'will
data <- read.table('~/spambase.data', sep = ',', col.names = names)
```

```r
data$spam <- as.factor(data$spam)

#splitting into training, validation, and test sets
set.seed(1)
RNGkind(sample.kind="Rejection")
sample1 <- sample(1:nrow(data), 0.6*nrow(data))
train <- data[sample1,]
forty_percent <- data[-sample1,]

sample2 <- sample(1:nrow(forty_percent), 0.5*nrow(forty_percent))
validation <- forty_percent[sample2,]
test <- forty_percent[-sample2,]

#logistic regression on all predictors
set.seed(1)
log_fit <- glm(spam ~ ., data = train, family = "binomial")
summary (log_fit)
```

```r
#fitting a decision tree
set.seed(1)
library(tree)
tree_fit <- tree(spam ~. , data=train)
plot(tree_fit)
text(tree_fit, pretty=0, cex = 0.7)
title("Decision tree with all predictors")
```

```r
#making boxplots
par(mfrow=c(3,3))
boxplot(data$remove~data$spam, horizontal = TRUE)
boxplot(data$dollar.sign~data$spam, horizontal = TRUE)
boxplot(data$george~data$spam, horizontal = TRUE)
boxplot(data$hp~data$spam, horizontal = TRUE)
boxplot(data$free~data$spam, horizontal = TRUE)
boxplot(data$our~data$spam, horizontal = TRUE)
boxplot(data$re~data$spam, horizontal = TRUE)
```

```r
#using AIC to shrink our logistic regression model
log_aic <- step(log_fit, direction = "backward")
summary(log_aic)

prob.training = predict(log_fit, newdata = validation, type="response")
round(prob.training, digits=2)


#validation error rate for full logistic model
predictedlabels <- ifelse (prob.training> 0.5, 1,0)
table(predict=predictedlabels, truth=validation$spam)
error_log <- mean(predictedlabels != validation$spam)

prob.training = predict(log_aic, newdata = validation, type="response")
round(prob.training, digits=2)

#validation error rate for reduced logistic model
predictedlabels <- ifelse (prob.training> 0.5, 1,0)
```

```r
table(predict=predictedlabels, truth=validation$spam)
error_aic <- mean(predictedlabels != validation$spam)
```

```r
library(ROCR)
pred <- prediction(prob.training, validation$spam)
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf, col=2, lwd=3, main ="ROC curve")
abline(0,1)
```

```r
#using 10-fold CV to select best tree size
tree_cv <- cv.tree(tree_fit, FUN=prune.misclass, K=10)

#best size
best_cv <- min(tree_cv$size[tree_cv$dev==min(tree_cv$dev)])

#pruning tree to optimal size
tree_prune <- prune.misclass(tree_fit, best=best_cv)

#validation error rate for pruned tree
tree_pred <- predict(tree_prune, newdata = validation, type = "class")
table(tree_pred, truth = validation$spam)
error_tree <- mean(tree_pred != validation$spam)
```

```r
plot(tree_prune)
text(tree_prune, pretty=0, cex = 0.7)
title("Pruned tree of size 14")
```

```r
#bagging with 500 trees
set.seed(1)
library(randomForest)
bag_fit <- randomForest(spam~., data=train, mtry=57, importance=TRUE)

#validation error rate for bagging
bag_pred <- predict(bag_fit, newdata = validation)
table(bag_pred, truth = validation$spam)
error_bag <- mean(bag_pred != validation$spam)
```

```r
plot(bag_fit)
legend("top", colnames(bag_fit$err.rate), col=1:4, cex=0.8, fill=1:4)
```

```r
#growing a random forest
set.seed(1)
forest_fit <- randomForest(spam~., data=train, mtry=sqrt(57), importance=TRUE)

#validation error rate for random forest
forest_pred <- predict(forest_fit, newdata = validation)
table(forest_pred, truth = validation$spam)
error_rf <- mean(forest_pred != validation$spam)
```

```r
plot(forest_fit)
legend("top", colnames(forest_fit$err.rate), col=1:4, cex=0.8, fill=1:4)
```

```r
varImpPlot(forest_fit, sort = TRUE, main = "Variable Importance for forest_fit", n.var=5)
```

```r
#boosting
set.seed(1)
```

```r
library(gbm)
boost_fit <- gbm(ifelse(train$spam=="1", 1, 0)~., data=train, distribution="bernoulli",
n.trees=500, interaction.depth = 4)
head(summary(boost_fit))
```

```r
#validation error rate for boosting
boost_prob <- predict(boost_fit, newdata=validation, n.trees=500, type="response")
boost_pred <- ifelse(boost_prob > 0.5, 1, 0)

table(boost_pred, truth = validation$spam)
error_boost <- mean(boost_pred != validation$spam)
```

```r
#linear kernel
set.seed(1)
library(e1071)
svmfit <- svm(spam ~ remove+dollar.sign+george+hp+free+our+re, data=train, kernel="linear", cost=0.1, s
summary(svmfit)

#find best cost with CV
tune.out = tune(svm, spam ~ remove+dollar.sign+george+hp+free+our+re, data=train,kernel="linear",
ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune.out)
```

```r
#best model
bestmod=tune.out$best.model
summary(bestmod)
```

```r
#validation error rate for svm (linear)
ypred = predict(bestmod, newdata=validation)
table(predict=ypred, truth=validation$spam)
error_svm_lin <- mean(ypred != validation$spam)
```

```r
#radial kernel
set.seed(1)
library(e1071)
svmfit = svm(spam ~ remove+dollar.sign+george+hp+free+our+re, data=train, kernel="radial",
gamma=1, cost =1, scale = TRUE)
summary(svmfit)

#find best cost and gamma with CV
tune.out = tune(svm, spam ~ remove+dollar.sign+george+hp+free+our+re, data=train,
kernel="radial", ranges=list(cost=c(0.1,1,10,100,1000), gamma=c(0.5,1,2,3,4)))
summary(tune.out)
```

```r
#best model
bestmod = tune.out$best.model
summary(bestmod)
```

```r
#validation error rate for svm (radial)
ypred = predict(bestmod, newdata=validation)
table(predict=ypred, truth=validation$spam)
error_svm_rad <- mean(ypred != validation$spam)
```

```r
plot(allK,error.rate,type='b',xlab='K-Value',ylab='Error rate',)
```

```r
Method <- c('Logistic Regression (FullModel)', 'Logistic Regression (ReducedModel)','Decision Tree', 'Ba
'Random Forest', 'Boosting', 'SVM(Linear)', 'SVM(Radial)', 'KNN')
Error_Rate <- c(error_log,error_aic,error_tree,error_bag,
error_rf,error_boost,error_svm_lin,error_svm_rad,error_knn)
data.frame(Method,Error_Rate)
```

```r
#test error rate for random forest
forest_pred <- predict(forest_fit, newdata = test)
table(forest_pred, truth = test$spam)
mean(forest_pred != test$spam)
```