Spam Emails

Nady Monir

10/27/2020

Overview

This is a summary of my analysis regarding spam emails for the dataset in the following link:

https://archive.ics.uci.edu/ml/datasets/Spambase

The idea is to separate the Spam emails from the regular (Not Spam) emails, keeping in mind that it is worse to classifying a regular as Spam and lose important information, than to classify a spam as regular email. After wrangling the data in the desired format, I have used the following machine learning algorithms:

- Generalized linear model (GLM)
- Naive Bayes
- Linear discriminant analysis (LDA)
- K Nearest Neighbours (KNN)
- Classification and regression tree (RPART)
- Random Forest (RF)
- Classification with a bagging (TREEBAG)
- An Ensemble of all the above

I separated the data into two sets, one for training and tuning, and the other for testing Let's go into it step by step

Analysis

Data Preparation

The data was presented in the sources as two files, one for Names of the fields (columns) and the descriptions associated, the other is or the data itself

and so, we will start with the names, we just selected the lines that contains the column names, and for example, such a line will be read like this:

```
dl <- tempfile()
download.file(
  "https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.names", dl)

read_lines(file = dl, skip_empty_rows = T)[34]
```

```
## [1] "\nword_freq_make:         continuous."
```

and then we removed the first character (\n), and then split by (:) choosing only the first part,

Here is the code for extracting the Names:

```
dl <- tempfile()
download.file(
  "https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.names", dl)

lines<-read_lines(file = dl, skip_empty_rows = T)[34:90]
lines<-sapply(lines,function(l){
  x<-str_sub(l,start = 2L)[1]
  str_split_fixed(x,":",2)[,1]
})
```

after that all we need to do is to add the Spam classified as a field:

```
lines<-unname(lines)
names<-c( lines,"spam")
```

Now, for the data, the source is representing every record (row) as a line with the fields are separated by comma (,), and the Spam classifier is a binary (1 for Spam, and 0 for Not Spam)

Here is the code for extracting the emails dataset, setting the column names, and changing the fields type as appropriate:

```
dl <- tempfile()
download.file(
  "https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data",
  dl)
emails<-read.delim2(dl,header = F, sep = ",", dec = ".") %>%
  set_names(., nm = names) %>% mutate(spam=ifelse(spam==1,"Spam", "Not Spam")) %>%
  mutate(spam=factor(spam, levels = c("Spam", "Not Spam"))) %>%
  mutate(capital_run_length_longest=as.double(capital_run_length_longest)) %>%
  mutate(capital_run_length_total=as.double(capital_run_length_total)) %>%
  as.tibble()
```

Data Exploration

1. There is 57 different predictor + Spam (Classifier / Target) with the following names:

```
names
```

```
##  [1] "word_freq_make"             "word_freq_address"
##  [3] "word_freq_all"              "word_freq_3d"
##  [5] "word_freq_our"              "word_freq_over"
##  [7] "word_freq_remove"           "word_freq_internet"
##  [9] "word_freq_order"            "word_freq_mail"
## [11] "word_freq_receive"          "word_freq_will"
## [13] "word_freq_people"           "word_freq_report"
## [15] "word_freq_addresses"        "word_freq_free"
## [17] "word_freq_business"         "word_freq_email"
## [19] "word_freq_you"              "word_freq_credit"
## [21] "word_freq_your"             "word_freq_font"
```

```
## [23] "word_freq_000"              "word_freq_money"
## [25] "word_freq_hp"               "word_freq_hpl"
## [27] "word_freq_george"           "word_freq_650"
## [29] "word_freq_lab"              "word_freq_labs"
## [31] "word_freq_telnet"           "word_freq_857"
## [33] "word_freq_data"             "word_freq_415"
## [35] "word_freq_85"               "word_freq_technology"
## [37] "word_freq_1999"             "word_freq_parts"
## [39] "word_freq_pm"               "word_freq_direct"
## [41] "word_freq_cs"               "word_freq_meeting"
## [43] "word_freq_original"         "word_freq_project"
## [45] "word_freq_re"               "word_freq_edu"
## [47] "word_freq_table"            "word_freq_conference"
## [49] "char_freq_;"                "char_freq_("
## [51] "char_freq_["                "char_freq_!"
## [53] "char_freq_$"                "char_freq_#"
## [55] "capital_run_length_average" "capital_run_length_longest"
## [57] "capital_run_length_total"   "spam"
```

this represents the frequency of some words and special characters, while the last three fields are for the Capital letters in the email body, we can see that there is an predictor for the frequency of the word "george", this can lead the way to personalize the spam classification in the future

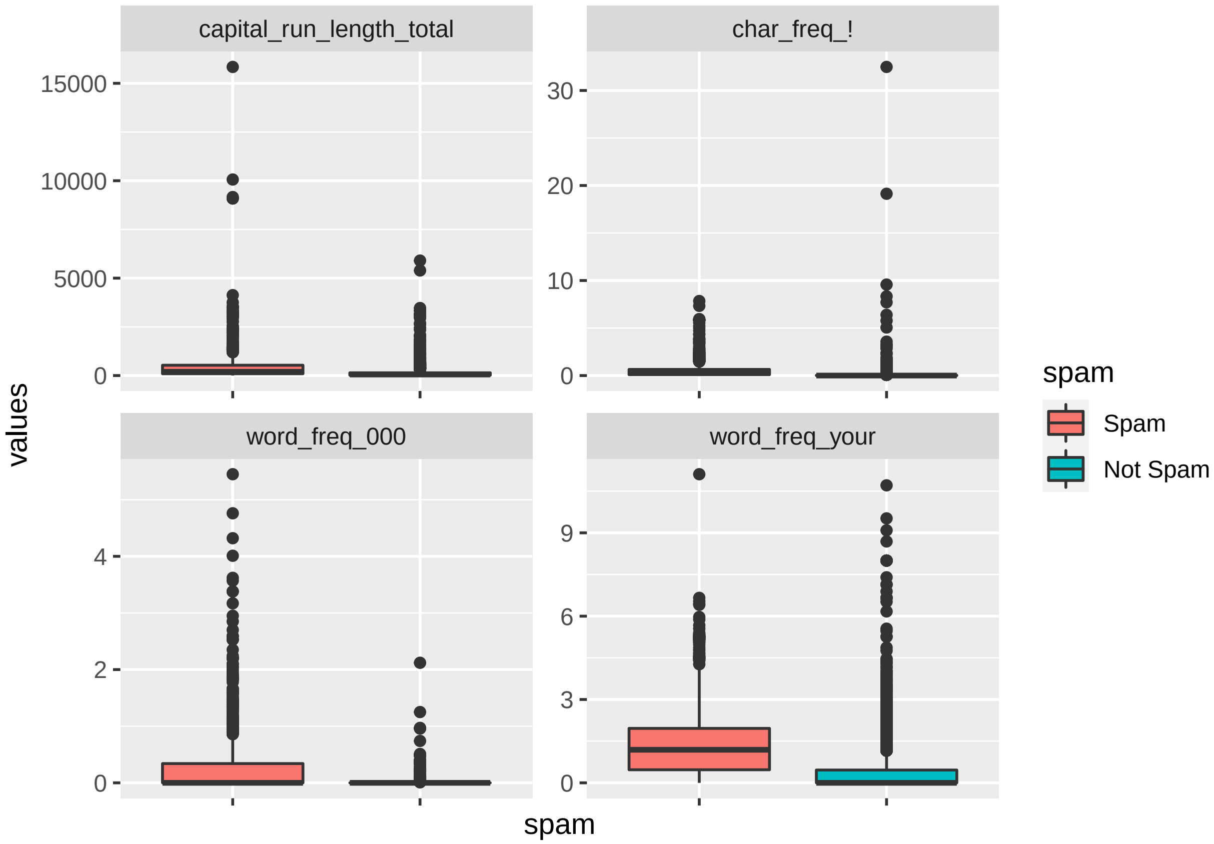
2. There is 4601 instance / row in the dataset, with the following classification:

```r
table(emails$spam)
```

```
##
##     Spam Not Spam
##     1813     2788
```

3. Some of the fields can be good predictors, as there a clear difference between the distribution for Spam / not Spam emails

```r
emails %>%
  pivot_longer(names_to = "Attr", values_to="values", cols=-spam) %>%
  filter(Attr %in% c("word_freq_000","word_freq_your",
                     "capital_run_length_total","char_freq_!")) %>%
  ggplot(aes(spam,values, fill=spam))+
  geom_boxplot()+
  facet_wrap(~Attr, scales = "free")+
  theme(axis.text.x = element_blank())
```



I have just choose some of the predictors, but remember there are 57 predictors

Training and Testing sets

Before we start trying some ML algorithms, we had to split the data into test and train datasets Here we choose the test dataset to be 20% of the original data

```
test_index<-createDataPartition(emails$spam, times = 1, p = 0.2, list = F)
train_set<-emails[-test_index,]
test_set<-emails[test_index,]
```

and then we reformat the training set into x & y, where y is the Spam classifier and x is a matrix contains the predictors:

```
x<-train_set %>% select(-spam) %>% as.matrix()
y=train_set$spam
```

Generalized linear model (GLM)

we will start with glm, as we can see, it doesn't have any tuning parameters:

```
modelLookup("glm")
```

```
##   model parameter     label forReg forClass probModel
## 1   glm parameter parameter   TRUE     TRUE      TRUE
```

Remembering that it is worse to classifying a regular as Spam and lose important information, than to classify a spam as regular email, will choose Specificity as the main metric instead of Accuracy,

we will create a small dataframe as a tracker of the performance of every algorithm we try Here is the code, the Confusion Matrix table and the performance (Specificity & Accuracy) of our model:

```r
control <- trainControl(summaryFunction = twoClassSummary)

train_glm<-train(x,y, method="glm",  metric = "Spec", trControl=control )
y_hat_glm<-predict(train_glm, newdata = test_set)
confusionMatrix(y_hat_glm, test_set$spam)$table
specificity_df<-data.frame(method="glm",
                           Spec=specificity(y_hat_glm, test_set$spam),
                           Accuracy=as.double(confusionMatrix(y_hat_glm,
                                            test_set$spam)$overall["Accuracy"]))
specificity_df
```

```
##           Reference
## Prediction Spam Not Spam
##   Spam      323       23
##   Not Spam   40      535

##   method      Spec  Accuracy
## 1    glm 0.9587814 0.9315961
```

Finally let's take a look at the 5 important fields used in this model:

```r
varImp(train_glm)$importance %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   arrange(desc(Overall)) %>% slice(1:5)
```

```
##            rowname   Overall
## 1    `char_freq_$` 100.00000
## 2 word_freq_remove  87.97299
## 3    word_freq_our  83.52989
## 4     word_freq_hp  80.60927
## 5   word_freq_free  77.21339
```

Naive Bayes

For Naive Bayes model, it has three tuning parameters, we will use the default tuning in Caret::train method:

```r
modelLookup("naive_bayes")
```

```
##         model parameter                label forReg forClass probModel
## 1 naive_bayes   laplace    Laplace Correction  FALSE     TRUE      TRUE
## 2 naive_bayes usekernel     Distribution Type  FALSE     TRUE      TRUE
## 3 naive_bayes    adjust Bandwidth Adjustment  FALSE     TRUE      TRUE
```

So here is the simple code for training, predicting and tracking the performance:

```r
train_naive_bayes<-train(x,y, method="naive_bayes",  metric = "Spec", trControl=control )
y_hat_naive_bayes<-predict(train_naive_bayes, newdata = test_set)
confusionMatrix(y_hat_naive_bayes, test_set$spam)$table
specificity_df<-rbind(specificity_df,
                      data.frame(method="naive_bayes",
                                 Spec=specificity(y_hat_naive_bayes, test_set$spam),
                                               Accuracy=confusionMatrix(y_hat_naive_bayes,
                                                     test_set$spam)$overall["Accuracy"]))
specificity_df
```

```
##           Reference
## Prediction Spam Not Spam
##   Spam      347      271
##   Not Spam   16      287
```

```
##        method      Spec  Accuracy
## 1         glm 0.9587814 0.9315961
## 2 naive_bayes 0.5143369 0.6883822
```

The Naive Bayes has very low performace in our case

Let's take a look at the important fields used in this model:

```r
varImp(train_naive_bayes)$importance %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   arrange(desc(Spam)) %>% slice(1:5)
```

```
##                      rowname      Spam  Not.Spam
## 1                char_freq_! 100.00000 100.00000
## 2 capital_run_length_longest  91.93271  91.93271
## 3 capital_run_length_average  87.09241  87.09241
## 4             word_freq_your  86.50675  86.50675
## 5                char_freq_$  83.39224  83.39224
```

Linear discriminant analysis (LDA)

For LDA model, There is no tuning parameters:

```r
modelLookup("lda")
```

```
##   model parameter     label forReg forClass probModel
## 1   lda parameter parameter  FALSE     TRUE      TRUE
```

Here is the simple code for training, predicting and tracking the performance:

```r
train_lda<-train(x,y, method="lda", metric = "Spec", trControl=control )
y_hat_lda<-predict(train_lda, newdata = test_set)
confusionMatrix(y_hat_lda, test_set$spam)$table
```

```r
specificity_df<-rbind(specificity_df,
                      data.frame(method="lda",
                                 Spec=specificity(y_hat_lda, test_set$spam),
                                          Accuracy=as.double(confusionMatrix(y_hat_lda,
                                                     test_set$spam)$overall["Accuracy"])))
specificity_df
```

```
##           Reference
## Prediction Spam Not Spam
##   Spam      287       20
##   Not Spam   76      538
```

```
##        method      Spec  Accuracy
## 1         glm 0.9587814 0.9315961
## 2 naive_bayes 0.5143369 0.6883822
## 3         lda 0.9641577 0.8957655
```

Let's take a look at the important fields used in this model:

```r
varImp(train_lda)$importance %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   arrange(desc(Spam)) %>% slice(1:5)
```

```
##                      rowname      Spam  Not.Spam
## 1                char_freq_! 100.00000 100.00000
## 2 capital_run_length_longest  91.93271  91.93271
## 3 capital_run_length_average  87.09241  87.09241
## 4             word_freq_your  86.50675  86.50675
## 5                char_freq_$  83.39224  83.39224
```

K Nearest Neighbours (KNN)

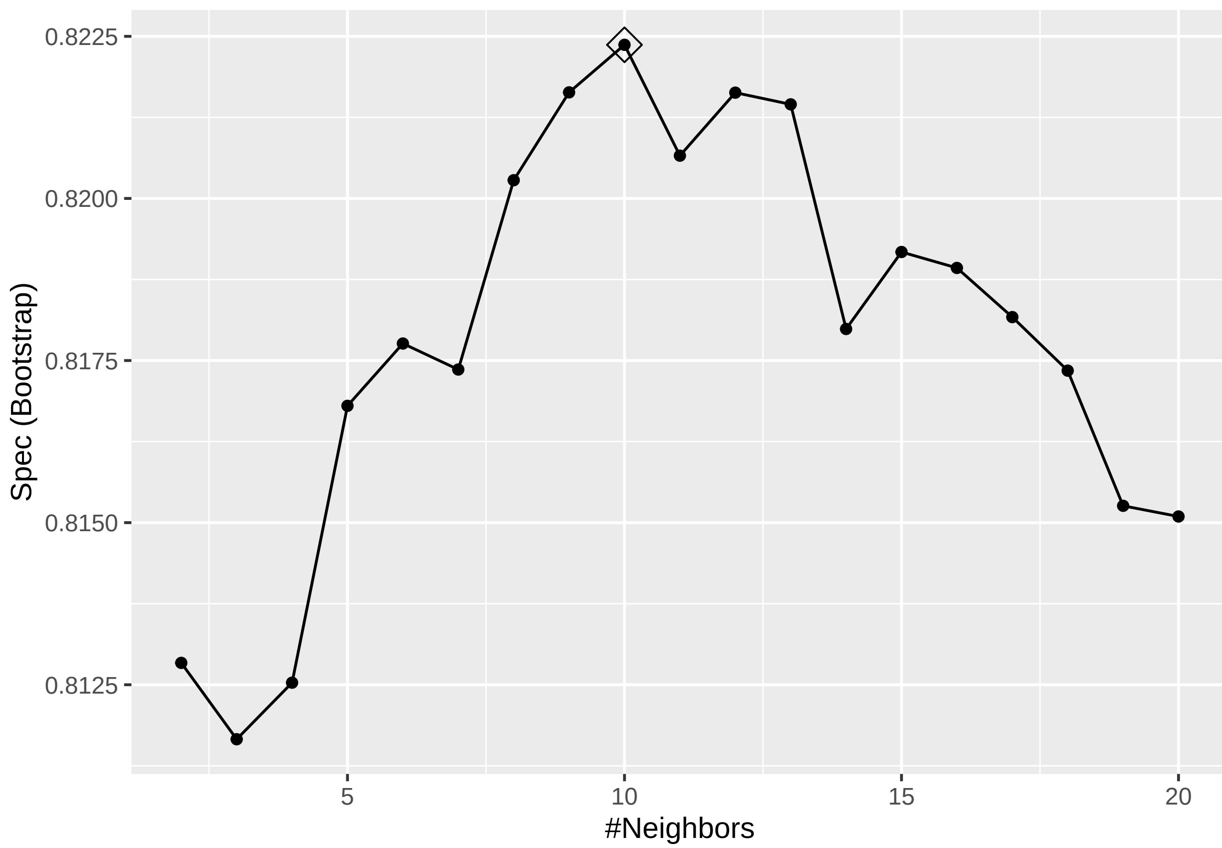
For KNN model, There is one tuning parameter, k, the number of neighbours taken into account to classify a point:

```r
modelLookup("knn")
```

```
##   model parameter      label forReg forClass probModel
## 1   knn         k #Neighbors   TRUE     TRUE      TRUE
```

We will try tuning with k from 2 to 20, and then plot the output and printing the optimum k

```r
grid <- data.frame(k = seq(2, 20))

train_knn<-train(x,y, method="knn", metric = "Spec", trControl=control , tuneGrid = grid)

ggplot(train_knn, highlight = T)
train_knn$bestTune
```




```
## [1] 10
```

and for the performance on the test set:

```
y_hat_knn<-predict(train_knn, newdata = test_set)
confusionMatrix(y_hat_knn, test_set$spam)$table

specificity_df<-rbind(specificity_df,
                      data.frame(method="knn",
                                 Spec=specificity(y_hat_knn, test_set$spam),
                                          Accuracy=as.double(confusionMatrix(y_hat_knn,
                                                  test_set$spam)$overall["Accuracy"])))
specificity_df
```

```
##           Reference
## Prediction Spam Not Spam
##   Spam      259       87
##   Not Spam  104      471
```

```
##        method      Spec  Accuracy
## 1         glm 0.9587814 0.9315961
## 2 naive_bayes 0.5143369 0.6883822
## 3         lda 0.9641577 0.8957655
## 4         knn 0.8440860 0.7926167
```

Let's take a look at the important fields used in this model:

```
varImp(train_knn)$importance %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   arrange(desc(Spam)) %>% slice(1:5)
```

```
##                      rowname      Spam  Not.Spam
## 1                char_freq_! 100.00000 100.00000
## 2 capital_run_length_longest  91.93271  91.93271
## 3 capital_run_length_average  87.09241  87.09241
## 4             word_freq_your  86.50675  86.50675
## 5                char_freq_$  83.39224  83.39224
```

Classification and regression tree (RPART)

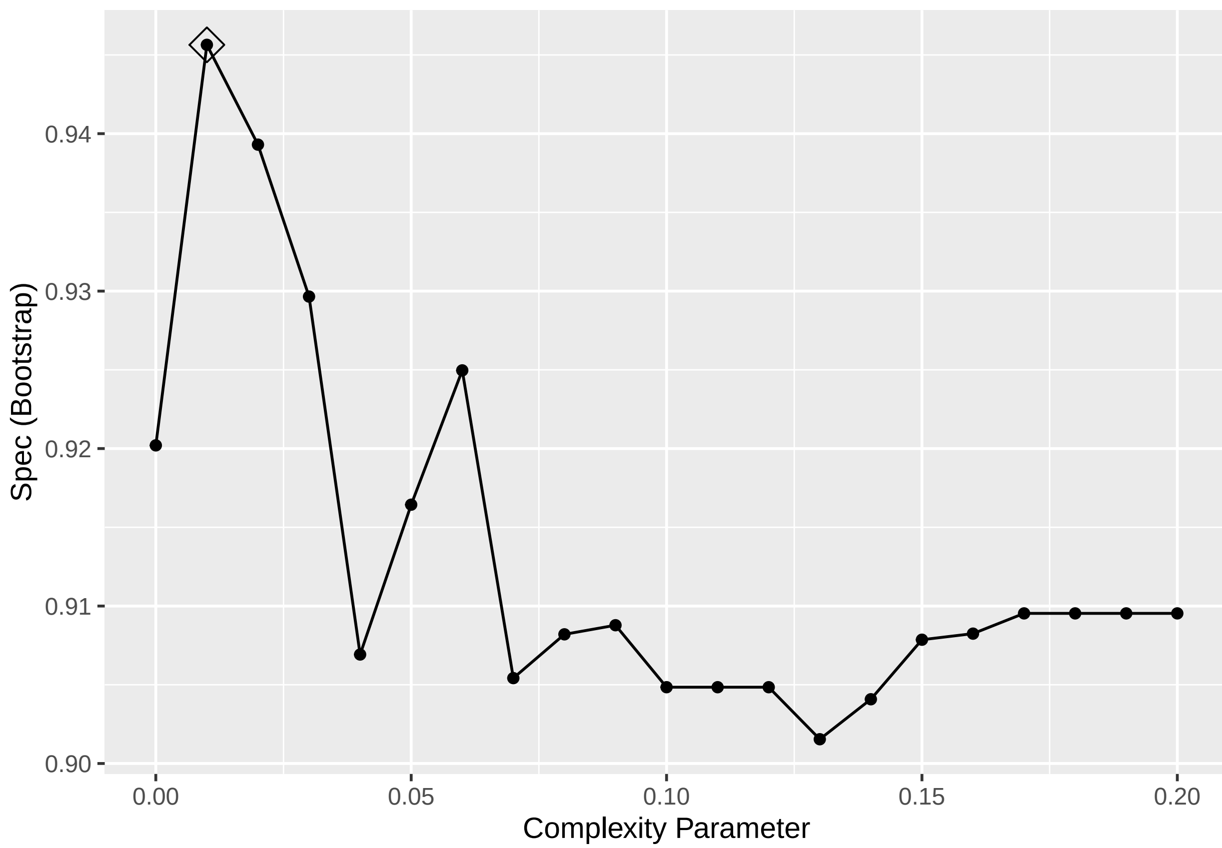
For RPART model, There is one tuning parameter, Complexity Parameter cp , it is used to prune trees to the limit the new branch van enhance the metric:

```
modelLookup("rpart")
```

```
##   model parameter                label forReg forClass probModel
## 1 rpart        cp Complexity Parameter   TRUE     TRUE      TRUE
```

We will try tuning with cp from 0 to 0.2, and then plot the output and printing the optimum cp

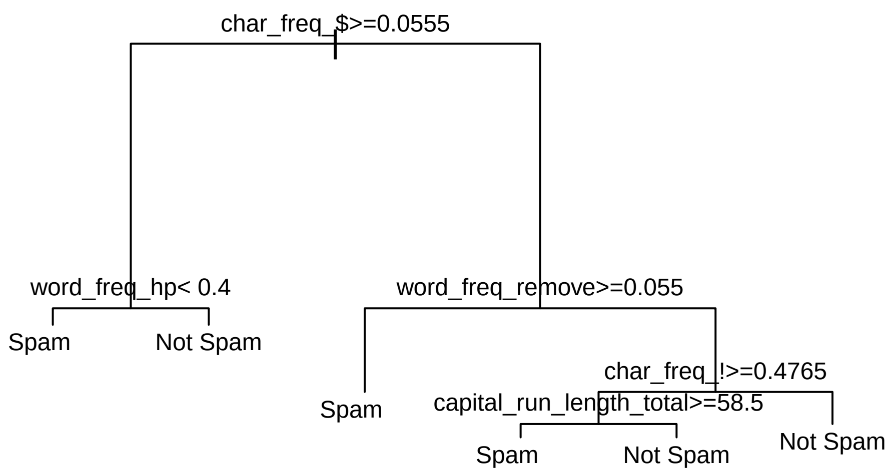
```
grid <- data.frame(cp=seq(0,0.2, length.out = 21))

train_rpart<-train(x,y, method="rpart", metric = "Spec", trControl=control ,
                   tuneGrid = grid)
ggplot(train_rpart, highlight = T)
train_rpart$bestTune
```




```
## [1] 0.01
```

let's take a look at the optimum model:

```
plot(train_rpart$finalModel,margin = 0.1)
text(train_rpart$finalModel, cex=0.75)
```



and for the performance on the test set:

```
y_hat_rpart<-predict(train_rpart, newdata = test_set)
confusionMatrix(y_hat_rpart, test_set$spam)$table

specificity_df<-rbind(specificity_df,
                      data.frame(method="rpart",
                                 Spec=specificity(y_hat_rpart, test_set$spam),
                                         Accuracy=as.double(confusionMatrix(y_hat_rpart,
                                                    test_set$spam)$overall["Accuracy"])))
specificity_df
```

```
##             Reference
## Prediction Spam Not Spam
##   Spam      290       30
##   Not Spam   73      528
```

```
##        method      Spec  Accuracy
## 1         glm 0.9587814 0.9315961
## 2 naive_bayes 0.5143369 0.6883822
## 3         lda 0.9641577 0.8957655
## 4         knn 0.8440860 0.7926167
## 5       rpart 0.9462366 0.8881650
```

Let's take a look at the important fields used in this model:

```r
varImp(train_rpart)$importance %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   arrange(desc(Overall)) %>% slice(1:5)
```

```
##            rowname   Overall
## 1      char_freq_! 100.00000
## 2   word_freq_free  83.69383
## 3 word_freq_remove  80.18533
## 4   word_freq_your  68.75806
## 5      char_freq_$  61.42393
```

Random Forest (RF)

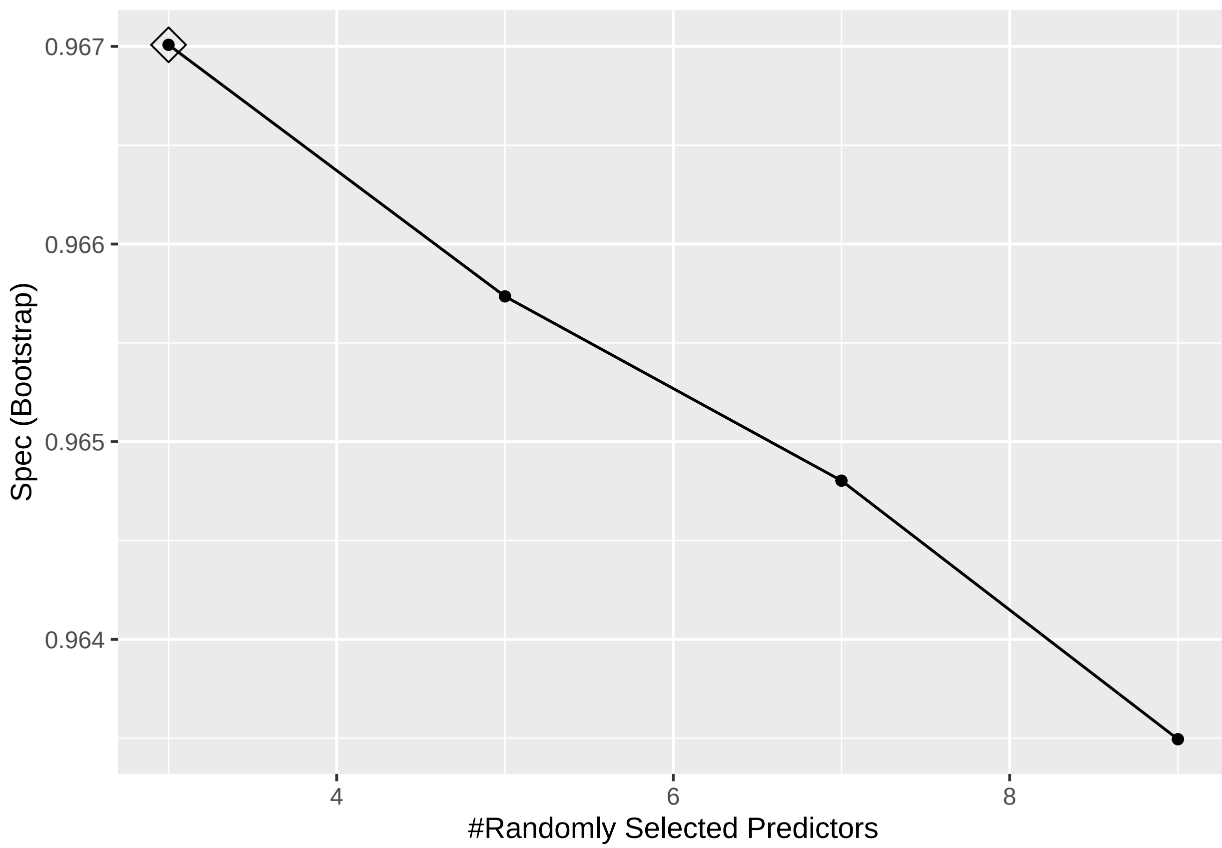
For RF model, There is one tuning parameter, mtry , it the number of the fields used in every tree:

```r
modelLookup("rf")
```

```
##   model parameter                         label forReg forClass probModel
## 1    rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
```

We will try tuning with mtry with these numbers (3, 5, 7, 9), and then plot the output and printing the optimum mtry

```r
train_rf<-train(x,y, method="rf", metric = "Spec", trControl=control ,
                tuneGrid = data.frame(mtry=c(3, 5, 7, 9)))
ggplot(train_rf, highlight = T)
train_rf$bestTune
```




```
## [1] 3
```

and for the performance on the test set:

```
y_hat_rf<-predict(train_rf, newdata = test_set)
confusionMatrix(y_hat_rf, test_set$spam)$table

specificity_df<-rbind(specificity_df,
                      data.frame(method="rf",
                                 Spec=specificity(y_hat_rf, test_set$spam),
                                        Accuracy=as.double(confusionMatrix(y_hat_rf,
                                               test_set$spam)$overall["Accuracy"])))
specificity_df
```

```
##           Reference
## Prediction Spam Not Spam
##   Spam      336       13
##   Not Spam   27      545
```

```
##        method      Spec  Accuracy
## 1         glm 0.9587814 0.9315961
## 2 naive_bayes 0.5143369 0.6883822
## 3         lda 0.9641577 0.8957655
## 4         knn 0.8440860 0.7926167
## 5       rpart 0.9462366 0.8881650
## 6          rf 0.9767025 0.9565689
```

Let's take a look at the important fields used in this model:

```r
varImp(train_rf)$importance %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   arrange(desc(Overall)) %>% slice(1:5)
```

```
##                      rowname   Overall
## 1                char_freq_! 100.00000
## 2                char_freq_$  77.92622
## 3          word_freq_remove  74.09888
## 4            word_freq_free  64.58441
## 5 capital_run_length_average  62.26511
```

Classification with a bagging (TREEBAG)

For TREEBAG model, There is no tuning parameters:

```r
modelLookup("treebag")
```

```
##     model parameter     label forReg forClass probModel
## 1 treebag parameter parameter   TRUE     TRUE      TRUE
```

Here is the training and testing code:

```r
train_treebag<-train(x,y, method="treebag",  metric = "Spec", trControl=control )

y_hat_treebag<-predict(train_treebag, newdata = test_set)
confusionMatrix(y_hat_treebag, test_set$spam)$table

specificity_df<-rbind(specificity_df,
                      data.frame(method="treebag",
                                 Spec=specificity(y_hat_treebag, test_set$spam),
                                      Accuracy=as.double(confusionMatrix(y_hat_treebag,
                                                    test_set$spam)$overall["Accuracy"])))
specificity_df
```

```
##           Reference
## Prediction Spam Not Spam
##   Spam      330       17
##   Not Spam   33      541
```

```
##        method      Spec  Accuracy
## 1         glm 0.9587814 0.9315961
## 2 naive_bayes 0.5143369 0.6883822
## 3         lda 0.9641577 0.8957655
## 4         knn 0.8440860 0.7926167
## 5       rpart 0.9462366 0.8881650
## 6          rf 0.9767025 0.9565689
## 7     treebag 0.9695341 0.9457112
```

Let's take a look at the important fields used in this model:

```
varImp(train_treebag)$importance %>%
   as.data.frame() %>%
   rownames_to_column() %>%
   arrange(desc(Overall)) %>% slice(1:5)
```

```
##            rowname   Overall
## 1      char_freq_! 100.00000
## 2   word_freq_free  81.12943
## 3 word_freq_remove  77.91572
## 4   word_freq_your  73.49917
## 5      char_freq_$  63.93209
```

Ensemble

Till now, we have tested 7 different models, with ensembling their output, we will take votes for every email (row) if it is Spam or Not, and deciding upon that vote:

```
ensemble<-data.frame(y_hat_glm,y_hat_naive_bayes,y_hat_lda,
                     y_hat_knn,y_hat_rpart,y_hat_rf,y_hat_treebag)
y_hat_ensemble <-apply(ensemble,1 , function(x)
  ifelse(mean(x=="Spam")>0.5,"Spam","Not Spam"))
y_hat_ensemble<-factor(y_hat_ensemble, levels = c("Spam","Not Spam"))
```

and here is the performace of the Ensemble:

```
confusionMatrix(y_hat_ensemble, test_set$spam)$table

specificity_df<-rbind(specificity_df,
                      data.frame(method="ensemble",
                                 Spec=specificity(y_hat_ensemble, test_set$spam),
                                 Accuracy=as.double(confusionMatrix(y_hat_ensemble,
                                          test_set$spam)$overall["Accuracy"])))
specificity_df
```

```
##           Reference
## Prediction Spam Not Spam
##   Spam      332       13
##   Not Spam   31      545
```

```
##        method      Spec  Accuracy
## 1         glm 0.9587814 0.9315961
## 2 naive_bayes 0.5143369 0.6883822
## 3         lda 0.9641577 0.8957655
## 4         knn 0.8440860 0.7926167
## 5       rpart 0.9462366 0.8881650
## 6          rf 0.9767025 0.9565689
## 7     treebag 0.9695341 0.9457112
## 8    ensemble 0.9767025 0.9522258
```

Result

From the models above, The Random Forest resulted the best performace, and the Important variables across most of the models are:

- char_freq_!
- char_freq_$
- word_freq_remove
- word_freq_free
- word_freq_your
- capital_run_length_averag
- capital_run_length_longest

The Special characters (! & $) has the most effect

That result is based on the Specificity & Accuracy

Conclusion

There is more than 250 Billion sent every day, with spam rate averaged at 14.30% worldwide, A small decision tree, or a simple glm model on a small number of variables can increase the performance of the email servers, while producing an OK result

The models presented here can be personalized, we saw a variable named "word_freq_george", something like that can be personalized for every receiver, ex: "word_freq_nady"

The limitations in this analysis is that we only have 4601 emails to work with, we would need larger data, to produce meaningful results, and we can also create different variables, if we have access to the emails themselves, but of course there is privacy issue in this regard.