Untitled

Mart Staarink

17-11-2021

Introduction

I chose to do assignment 3 in which a Shiny App must be built. The app must be able to classify a certain E-mail as either Spam or Ham (not Spam). Therefore, this is a classification problem. The dataset that was used is retrieved from the site Kaggle (link: https://www.kaggle.com/balaka18/email-spam-classification-dataset-csv). The dataset shows the count of a particular word retrieved from 5172 E-mails. Table 1 shows a small section of the dataset.

```
#seed and libraries
set.seed(1)

library(readr)
library(glmnet)
library(caret)
library(zoo)
library(randomForest)
library(ggplot2)
library(wordcloud2)
library(wordcloud)
library(caret)
library(knitr)

#load and prepare data
Data_Emails <- read_csv("C:/Users/marts/OneDrive - Universiteit Twente/Financial Engineering and Manage
names(Data_Emails)[3002] = "Spam"
Data_Emails$Spam = ifelse(Data_Emails$Spam==1,"Spam","Ham")

#make preview table
preview_data = Data_Emails[1:10,1:7]
dot.row = rep("...",10)
dot.column = rep("...",10)
spamham.column = Data_Emails[,3002]
last.word.column = Data_Emails[,3001]

preview_df = data.frame(preview_data,dot.column,last.word.column[1:10,],spamham.column[1:10,])
preview_df[9,] = dot.row
preview_df[10,1] = "Email 5172"

kable(preview_df,"pipe",col.name=c("Email Nr.","Word 1: the","Word 2: to","Word 3: ect","Word 4: and","
caption="Section of Spam/Ham Data Used")
```

Table 1: Section of Spam/Ham Data Used

Email Nr.	Word 1: the	Word 2: to	Word 3: ect	Word 4: and	Word 5: for	Word 6: of	...	Word 3001: dry	Spam/Ham
Email 1	0	0	1	0	0	0	...	0	Ham
Email 2	8	13	24	6	6	2	...	0	Ham
Email 3	0	0	1	0	0	0	...	0	Ham
Email 4	0	5	22	0	5	1	...	0	Ham
Email 5	7	6	17	1	5	2	...	0	Ham
Email 6	4	5	1	4	2	3	...	0	Spam
Email 7	5	3	1	3	2	1	...	0	Ham
Email 8	0	2	2	3	1	2	...	0	Spam
...	...	...	...	...	...	...	...	...	...
Email 5172	4	4	35	0	1	0	...	0	Ham

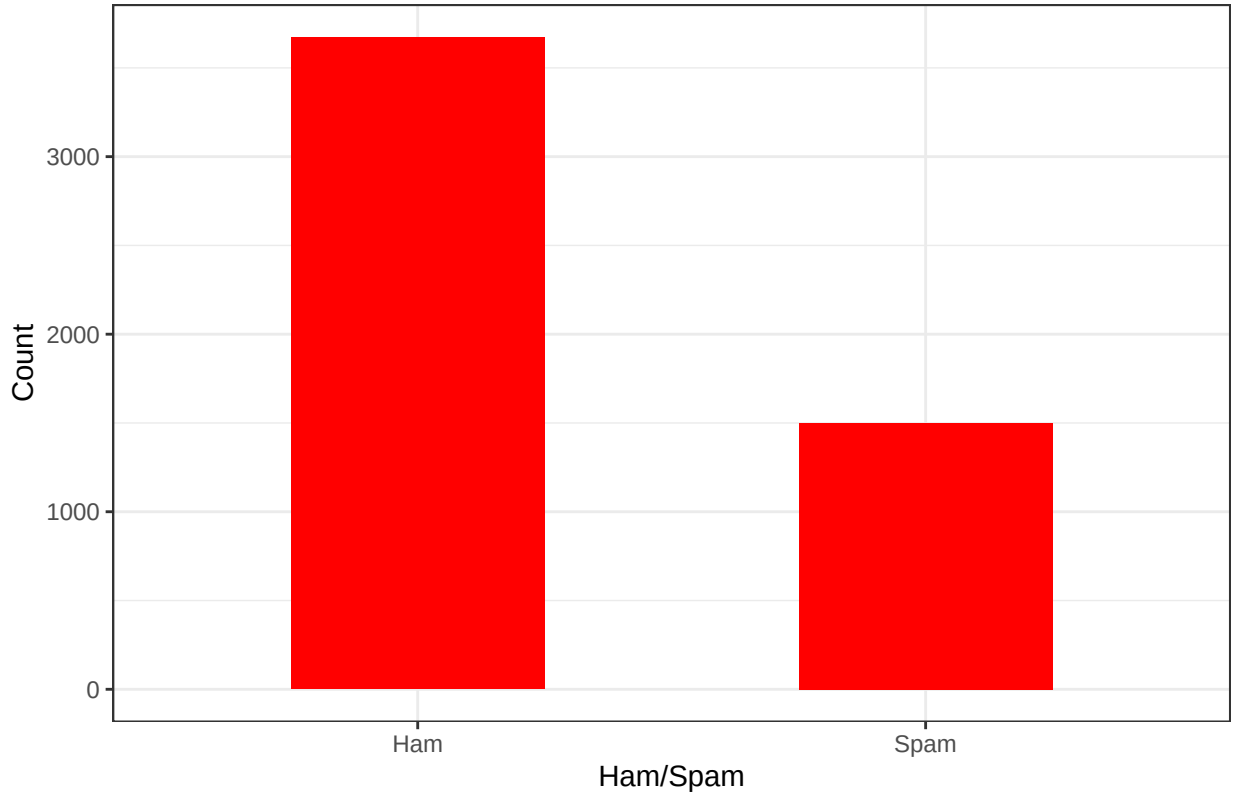
```
Data_Emails = Data_Emails[,-1] #delete first column
Data_Emails$Spam = as.factor(Data_Emails$Spam) #make factor of outcome variable

#we delete the 'enron' column
index.enron = which(colnames(Data_Emails)=="enron")
Data_Emails = Data_Emails[,-index.enron]


percentage.Spam = mean(Data_Emails$Spam=="Spam") * 100
percentage.Ham = mean(Data_Emails$Spam=="Ham") * 100
```

Data Inspection

Now, we will inspect the dataset. Figure 1 shows the division of the outcome variable Spam/Ham. 29% of the outcome variable is Spam, and 71% is Ham. This is quite an imbalanced dataset.

```
#inspect data (spam/ham histogram, frequency plots and wordclouds)
# spam/ham histogram
theme_set(theme_bw())
ggplot(aes(x=Data_Emails$Spam),data=Data_Emails) +
  geom_bar(fill="red",width=0.5) +
  labs(x="Ham/Spam") +
  labs(y="Count") +
  ggtitle("Figure 1: Spam/Ham Counts of the Dataset") +
  theme(plot.title = element_text(hjust = 0.5))
```

Figure 1: Spam/Ham Counts of the Dataset



Also a wordcloud of the 100 most frequently occuring words bigger than 4 characters for the Spam and Ham E-mails, respectively, are shown. We can see that from the first wordcloud (for Spam) that 'typical' Spam E-mail words are present in the Spam E-mails, for example: "Company", "Stock", "Price", "Invest", to name a few. The same holds for the Ham E-Mails, with less 'commercially-driven' words.

```
#wordclouds
max.nr.words = 100

spam.array = Data_Emails$Spam=="Spam"
ham.array = Data_Emails$Spam=="Ham"

Emails_Spam = Data_Emails[spam.array,]
Emails_Ham = Data_Emails[ham.array,]

lastcol = length(colnames(Emails_Spam))
words.total.Spam = sum(Emails_Spam[,-lastcol])
words.total.Ham = sum(Emails_Ham[,-lastcol])

minchar = 5
indices = which(nchar(colnames(Data_Emails[,-lastcol]))>=minchar)

Emails_Spam2 = Emails_Spam[,indices]
Emails_Ham2 = Emails_Ham[,indices]

colsums.Spam = colSums(Emails_Spam2)
colsums.Ham = colSums(Emails_Ham2)
```

```
freq.array.Spam = rep(NA,length(colsums.Spam))
freq.array.Ham = rep(NA,length(colsums.Ham))

for (i in 1:length(freq.array.Spam)){
  #Spam
  freq = colsums.Spam[i] / words.total.Spam
  freq.array.Spam[i] = freq

  #Ham
  freq = colsums.Ham[i] / words.total.Ham
  freq.array.Ham[i] = freq
}

#Spam
words.array.Spam = colnames(Emails_Spam2)
df.Spam = data.frame(words.array.Spam,freq.array.Spam)

  #Make a Spam df with the N most-often occuring words
ordered.freqs.Spam = order(df.Spam$freq.array.Spam,decreasing=TRUE)[1:max.nr.words]
df.Spam.ordered = df.Spam[ordered.freqs.Spam,]

#Ham
words.array.Ham = colnames(Emails_Ham2)
df.Ham = data.frame(words.array.Ham,freq.array.Ham)

  #Make a Ham df with the N most-often occuring words
ordered.freqs.Ham = order(df.Ham$freq.array.Ham,decreasing=TRUE)[1:max.nr.words]
df.Ham.ordered = df.Ham[ordered.freqs.Ham,]

wordcloud(words=df.Spam.ordered$words.array.Spam,freq=df.Spam.ordered$freq.array.Spam,
          max.words=100,random.order=FALSE,colors=brewer.pal(8,"Dark2"),scale=c(2.5,0.25))
```


```
wordcloud(words=df.Ham.ordered$words.array.Ham,freq=df.Ham.ordered$freq.array.Ham,
          max.words=100,random.order=FALSE,colors=brewer.pal(8,"Dark2"),scale=c(2,0.5))
```

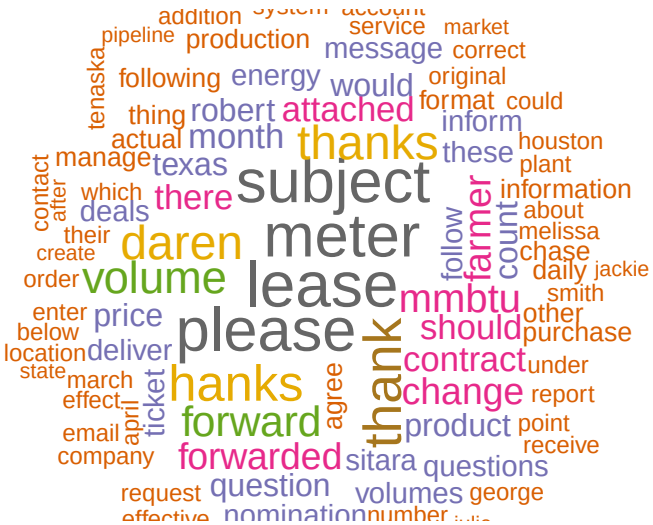
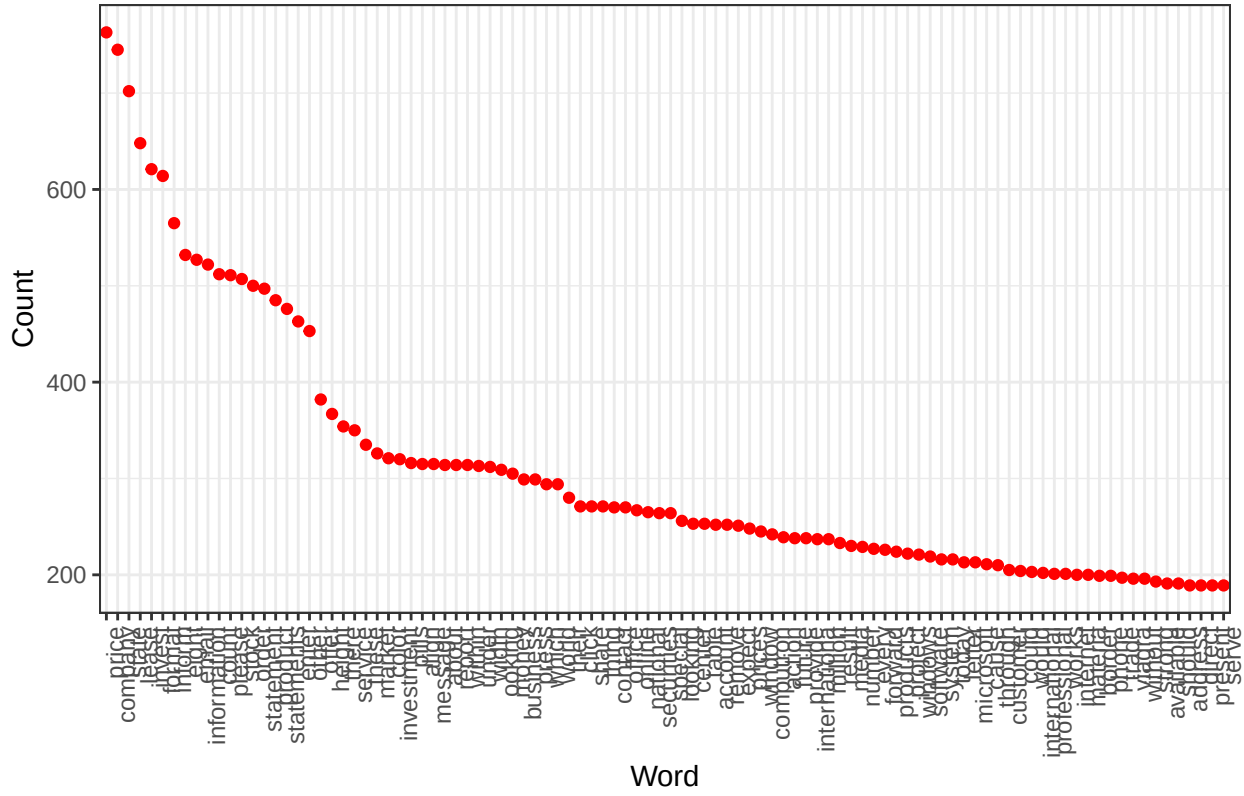


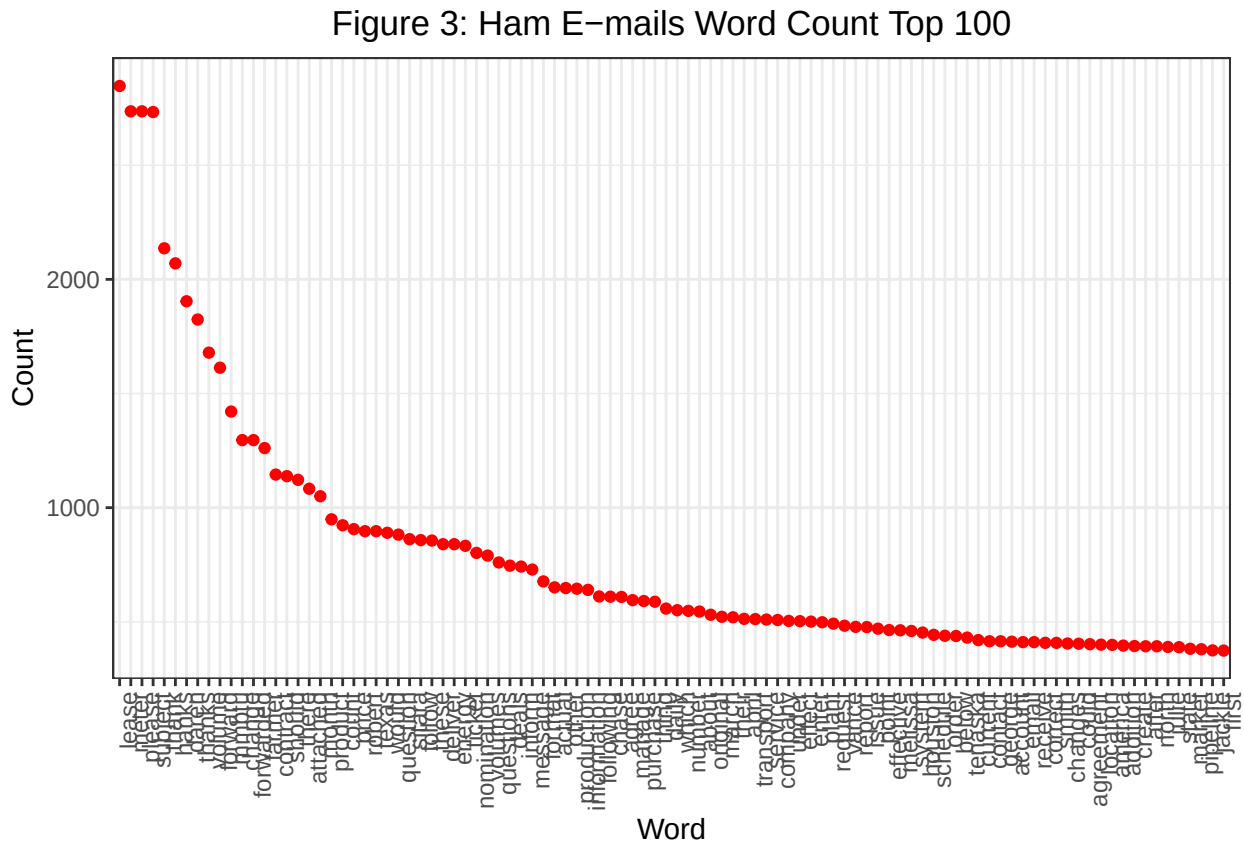
Figure 2 and 3 show the 100 most frequently encountered words in the Spam and Ham E-mails, respectively. Again the most 'commercially-driven' words are included in the Spam E-mails, while this is not the case for the Ham E-mails.

```
#frequency plots
  #Spam
counts.maxnr.Spam = colsums.Spam[ordered.freqs.Spam]
df.Spam2 = data.frame(df.Spam.ordered[,1],counts.maxnr.Spam)

ggplot(df.Spam.ordered,aes(x=df.Spam.ordered[,1],y=counts.maxnr.Spam)) +
  geom_point(color="red") +
  theme(axis.text.x=element_text(angle=90,hjust=1)) +
  aes(x=reorder(df.Spam.ordered[,1],-counts.maxnr.Spam),y=counts.maxnr.Spam) +
  ggtitle("Figure 2: Spam E-mails Word Count Top 100") +
  xlab("Word") + ylab("Count") +
  theme(plot.title = element_text(hjust = 0.5))
```

Figure 2: Spam E−mails Word Count Top 100




```
  #Ham
counts.maxnr.Ham = colsums.Ham[ordered.freqs.Ham]
df.Ham2 = data.frame(df.Ham.ordered[,1],counts.maxnr.Ham)

ggplot(df.Ham.ordered,aes(x=df.Ham.ordered[,1],y=counts.maxnr.Ham)) +
  geom_point(color="red") +
  theme(axis.text.x=element_text(angle=90,hjust=1)) +
  aes(x=reorder(df.Ham.ordered[,1],-counts.maxnr.Ham),y=counts.maxnr.Ham) +
  ggtitle("Figure 3: Ham E-mails Word Count Top 100") +
  xlab("Word") + ylab("Count") +
  theme(plot.title = element_text(hjust = 0.5))
```



Lastly, one word was found ('by accident') which is not really useful for building the model: Enron. Enron is a company which has filed for Chapter 11 bankruptcy. Therefore, this word was deleted from the dataset. A very thorough inspection of the dataset in order to find more of these words would be ideal, however, due to time constraints this is simply not possible. We can now continue to building the models.

Model Building Two models will be built: a Lasso model and a Random Forest model. For the Lasso model we must determine an optimal threshold value for which a certain probability is classified as 'Spam' or 'Ham'. For this, the F-score will be used. The F-score assesses the model's accuracy by combining the precision and recall. In this context, the precision would be the number of E-mails correctly classified as Spam (true positives) divided by the total number of E-mails classified as Spam (positives). It would answer the question: "Of all the E-mails that were classified as Spam, how many were actually Spam?". The recall would be the number of E-mails that were classified as Spam divided by the total number of E-mails that were actually Spam. It would answer the question: "Of all the E-mails that were actually Spam, how many were classified as Spam?".

$$F_\beta = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \tag{1}$$

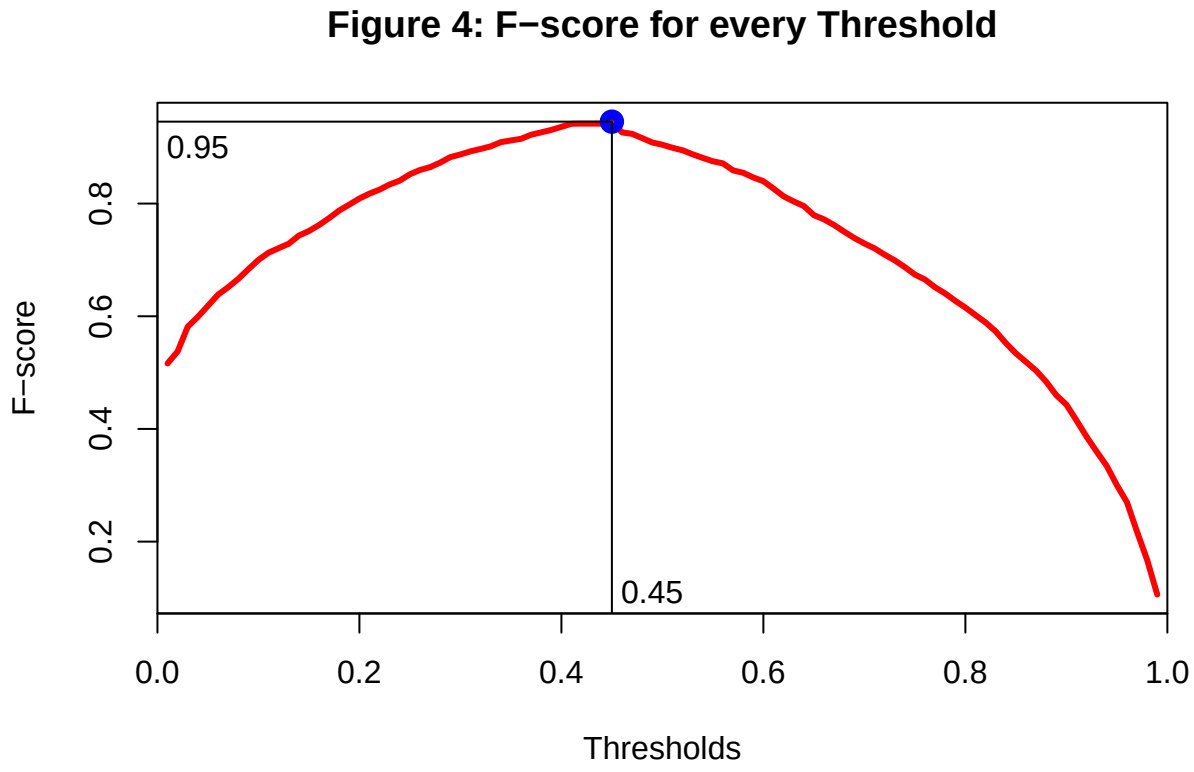
Equation 1 shows the formula for the F-score. The beta in the formula indicated the magnitude of importance assigned to the recall and precision: a beta > 1 assigns more weight to recall, a beta < 1 assigns more weight to precision. I have set beta $= 1$. Allthough an "optimal" threshold is calculated in this report, in the Shiny App, I will leave choosing the threshold to the user and merely advise the user to use the threshold as was calculated by the F-score.

Now we will build the Lasso model.

```
# LASSO
# model building (on full dataset)
```

```
xdata = model.matrix(Spam~.,Data_Emails)[,-1] #delete intercept
ydata = Data_Emails$Spam

grid = 10^seq(10,-2,length=100) #grid from 10^10 to 10^-2
cv.out = cv.glmnet(xdata,ydata,alpha=1,family="binomial",lambda=grid) #tuning lambda
bestlam = cv.out$lambda.min #the lambda resulting in the lowest mean cross-validated error

lasso.final = glmnet(xdata,ydata,alpha=1,family="binomial",lambda=grid) #full/final model,
# note that the model is built on a full grid, and not only for one lambda, this is
# (counterintuitively) faster in R (see R documentation for 'glmnet')
lasso.coefs = predict (lasso.final,type ="coefficients",s=bestlam) #coefficients for the
# final model with the lambda chosen by CV


#determine optimal threshold using F-score
thresholds = seq(0,1,by=0.01)

fscore.array = rep(NA,length(thresholds))

lasso.preds = predict(lasso.final,s=bestlam,newx=xdata,type="response")

for (i in 1:length(thresholds)){ #loop over all thresholds

  lasso.preds.yn = ifelse(lasso.preds>thresholds[i],"Yes","No")

  confusion.matrix = table(lasso.preds.yn,ydata)

  tp = confusion.matrix[4] #true positive
  fn = confusion.matrix[3] #false negative

  fp = confusion.matrix[2] #false positive
  tn = confusion.matrix[1] #true negative

  precision = (tp) / (tp + fp)
  recall = (tp) / (tp + fn)
  beta = 1
  fscore = (1+(beta^2)) * (precision*recall) / (((beta^2)*precision)+recall)

  fscore.array[i] = fscore
}
#Fscore
index.biggest.fscore = which.max(fscore.array)
max.fscore = fscore.array[index.biggest.fscore]
optimal.threshold = thresholds[index.biggest.fscore]

#plot thresholds against F-score
plot(thresholds,fscore.array,type="l",lwd=3,col="red",xlab="Thresholds",ylab="F-score",
     main="Figure 4: F-score for every Threshold",xaxs="i")
#grid(lty="solid")
points(optimal.threshold,max.fscore,lwd=5,pch=21,col="blue")
lines(c(optimal.threshold,optimal.threshold),c(0,max.fscore))
lines(c(0,optimal.threshold),c(max.fscore,max.fscore))
text(0.49,0.11,"0.45")
```

```
text(0.04,0.9,"0.95")
```




```
#finding and displaying 15 most impactful predictors (words)
absval.lasso.coefs = abs(lasso.coefs)
most.impactful.15.index = order(absval.lasso.coefs,decreasing=TRUE)[1:15]
```

```
## <sparse>[ <logic> ] : .M.sub.i.logical() maybe inefficient
```

```
lasso.coefs.15 = lasso.coefs[most.impactful.15.index]
words.15 = colnames(xdata)[most.impactful.15.index]

df2 = data.frame(words.15,lasso.coefs.15)
kable(df2,"pipe",col.name=c("Predictor (word)","Coefficient"),caption=
        "Lasso predictors and coefficients")
```

Table 2: Lasso predictors and coefficients

Predictor (word)	Coefficient
us	-1.2011892
thomas	0.8774312
dy	0.8770598
weight	0.8761487
business	-0.8058835
u	-0.7967570
www	-0.7378663
microsoft	0.6918776

Predictor (word)	Coefficient
armstrong	0.6466363
happy	0.6390796
please	-0.6335110
carthage	0.6318307
investor	0.6195798
far	0.5465657
but	-0.5459493

```
cv.out
```

```
##
## Call:  cv.glmnet(x = xdata, y = ydata, lambda = grid, alpha = 1, family = "binomial")
##
## Measure: Binomial Deviance
##
##     Lambda Index Measure       SE Nonzero
## min   0.01   100  0.4066 0.002936     164
## 1se   0.01   100  0.4066 0.002936     164
```

Figure 4 shows that the optimal threshold determined by the biggest F-score is 0.45. Table 2 shows the 10 most impactful predictors (or words) determined by the Lasso model, i.e., the 10 predictors with the biggest absolute value of their beta-coefficient. As can be seen in the output above, the Lasso has set only 164 predictors out of the 3001 to non-zero.

Now we will validate the Lasso Model. The Area Under the Curve (AUC) will be used to assess the performance of the model.

```
#CROSS-VALIDATION
K = 10 #the 'K' in K-fold CV
folds = createFolds(Data_Emails$Spam,k=K)

cv.accuracies <- matrix(NA,K,1,dimnames=list(NULL,paste(1)))
foldnr <- 0 #initialize
grid = 10^seq(10,-2,length=100) #grid for lambdas

auc.lasso.array = rep(NA,K)

test.accuracies = lapply(folds, function(x) { #begin cross-validation
  foldnr <<- foldnr + 1

  xtrain = xdata[-x, ]
  xtest = xdata[x, ]

  ytrain = ydata[-x]
  ytest = ydata[x]
  yn.test = ytest

  print(1)

  cv.out = cv.glmnet(xtrain,ytrain,alpha=1,family="binomial",lambda=grid) #tuning lambda
  bestlam = cv.out$lambda.min

  print(2)
```

```
  thresholds = seq(0,1,by=0.01)
  tp.array = rep(NA,length(thresholds))
  fp.array = rep(NA,length(thresholds))

  lasso.preds = predict(cv.out,s=bestlam,newx=xtest,type="response")

  for (i in 1:length(thresholds)){ #loop over all thresholds

    lasso.preds.yn = ifelse(lasso.preds>thresholds[i],"Yes","No")

    confusion.matrix = table(lasso.preds.yn,ytest)

    tp = confusion.matrix[4] #true positive
    fn = confusion.matrix[3] #false negative

    fp = confusion.matrix[2] #false positive
    tn = confusion.matrix[1] #true negative

    true.positive.rate = tp / (tp + fn)
    false.positive.rate = fp / (fp + tn)

    tp.array[i] = true.positive.rate
    fp.array[i] = false.positive.rate

  }
  id = order(fp.array)
  auc.lasso = sum(diff(fp.array[id])*rollmean(tp.array[id],2),na.rm=TRUE) #calculate AUC

  #store relevant results
  auc.lasso.array[foldnr] <<- auc.lasso
  cv.accuracies[foldnr,1] <<- round(auc.lasso,2)
})
```

```
## [1] 1
## [1] 2
## [1] 1
## [1] 2
## [1] 1
## [1] 2
## [1] 1
## [1] 2
## [1] 1
## [1] 2
## [1] 1
## [1] 2
## [1] 1
## [1] 2
## [1] 1
```

```
## Warning: from glmnet Fortran code (error code -99); Convergence for 99th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## [1] 2
## [1] 1
```

```
## [1] 2
## [1] 1
## [1] 2
```

```
mean.auc.lasso = mean(auc.lasso.array) #calculate mean AUC

#calculating CI
n<-K
standard.dev = sd(auc.lasso.array)
CI.lower.lasso = mean.auc.lasso - 1.96*standard.dev/sqrt(n)
CI.upper.lasso = mean.auc.lasso + 1.96*standard.dev/sqrt(n)

#put cv.accuracies in a table
folds.cv = 1:K
df.cv.acc = data.frame(folds.cv,cv.accuracies)
kable(df.cv.acc,"pipe",col.name=c("Fold","AUC"),
      caption="10-Fold Cross-Validation Results for the Lasso Model")
```

Table 3: 10-Fold Cross-Validation Results for the Lasso Model

Fold	AUC
1	0.74
2	0.74
3	0.73
4	0.72
5	0.76
6	0.73
7	0.74
8	0.69
9	0.71
10	0.74

Table x shows the Cross-Validation results for every fold, the mean AUC of the Lasso model is 0.73.

Now we will build the Random Forest model. An important parameter included in the development of a Random Forest model is the number of variables which are considered at each step (i.e., the 'split candidates'), R refers to this as the 'mtry'. The initial idea was to find the optimal parameter for the 'mtry' by using 10-Fold Cross-Validation on a grid of 'mtrys'. So, for the first iteration of the Cross-Validation, 9 folds will be used as the training data, and 1 fold will be used at the test data. Then, for each value of 'mtry' in the grid a Random Forest is fit on the training data, and tested on the test data. This results in having a test accuracy for each 'mtry' for this first iteration of the Cross-Validation. These steps are then repeated 9 more times, which will results in having 10 test accuracies for each value of 'mtry' in the grid. The optimal 'mtry' is then the 'mtry' in the grid for which the test accuracy is the highest. However, this procedure requires a too high computation time, as can be expected when having 3001 predictors. Therefore, I set the 'mtry' to 55, since convention dictates that 'mtry' must be set to the square root of the total number of predictors in the dataset: $\sqrt{(3001)} = 55$. For the Random Forest model we will use the 'majority vote' principle to classify an E-mail as Spam or Ham, therefore it is not needed to determine an optimal threshold (as was the case for the Lasso model).

```
#MODEL BUILDING (on full dataset)
names(Data_Emails) <- make.names(names(Data_Emails))

npredictors = length(colnames(Data_Emails))
```

```
#final model
rf.final = randomForest(Spam~.,data=Data_Emails,mtry=sqrt(npredictors),importance=TRUE)
#importance(rf.final)
varImpPlot(rf.final,n.var=15,main="Figure 5: Variable Importance Plot")
```

Figure 5: Variable Importance Plot

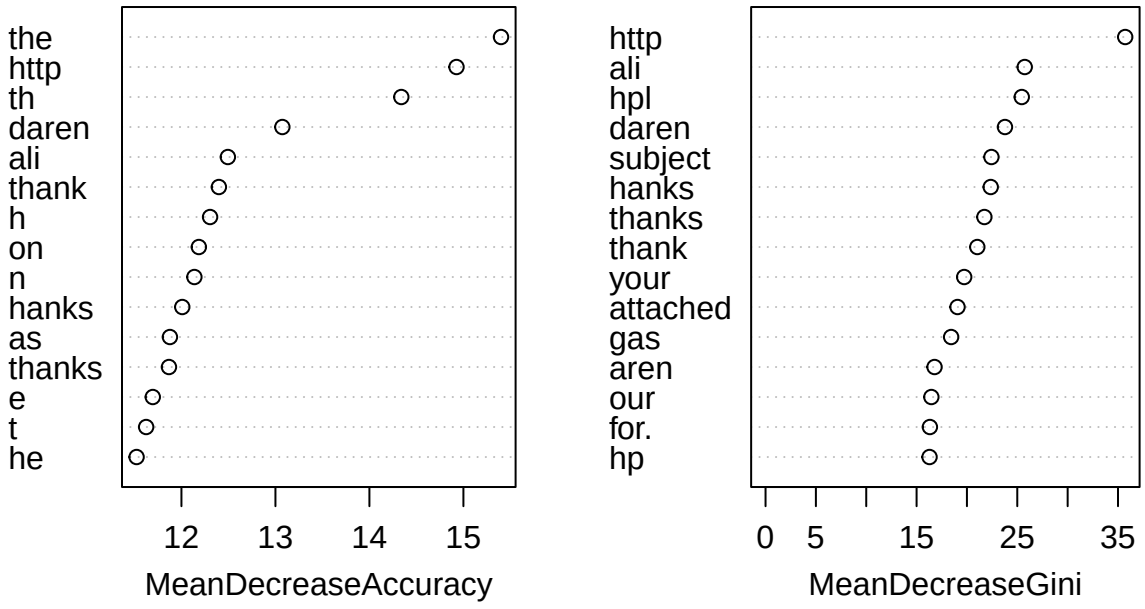


Figure 5 shows the importance of the 15 most important variables according to the Random Forest model. Based on the mean decrease in accuracy, we can see that the three most important variables in predicting whether an E-mail is Spam or Ham are: http, he, daren. The figure once again highlights the importance of removing superfluous (or even non-existing) words such as 'daren', 'h', 'da'.

Now we will validate the model and thus assess its predictive ability. This will be done by performing 5-Fold Cross-Validation and calculating the Area Under the Curve (AUC) of the ROC curve.

```
#CROSS-VALIDATION
K = 5 #the 'K' in K-fold CV
folds = createFolds(Data_Emails$Spam,k=K)

foldnr <- 0
cv.accuracies <- matrix(NA,K,1,dimnames=list(NULL,paste(1))) #to store
#CV results for each fold accuracy, and optimal mtry

auc.rf.array = rep(NA,K)

test.accuracies = lapply(folds, function(x) {
  foldnr <<- foldnr + 1

  xtrain = Data_Emails[-x,]
  xtest = Data_Emails[x,]
```

```
  yn.test = Data_Emails$Spam[x]

  rf.tree = randomForest(Spam~.,data=xtrain,mtry=sqrt(npredictors),importance=FALSE)
  yn.test = Data_Emails$Spam[x]

  thresholds = seq(0,1,by=0.01)
  tp.array = rep(NA,length(thresholds))
  fp.array = rep(NA,length(thresholds))

  rf.preds = predict(rf.tree,xtest,type="prob")[,2] #gives probabilities
  #for 'yes'

  #calculate ROC and AUC
  for (i in 1:length(thresholds)){ #loop over all thresholds

    print(i)

    rf.preds.yn = ifelse(rf.preds>thresholds[i],"Yes","No")

    confusion.matrix = table(rf.preds.yn,yn.test)

    tp = confusion.matrix[4] #true positive
    fn = confusion.matrix[3] #false negative

    fp = confusion.matrix[2] #false positive
    tn = confusion.matrix[1] #true negative

    true.positive.rate = tp / (tp + fn)
    false.positive.rate = fp / (fp + tn)

    tp.array[i] = true.positive.rate
    fp.array[i] = false.positive.rate
  }
  id = order(fp.array)
  auc.rf = sum(diff(fp.array[id])*rollmean(tp.array[id],2),na.rm=TRUE) #calculate AUC

  #store relevant results
  auc.rf.array[foldnr] <<- auc.rf
  cv.accuracies[foldnr,1] <<- round(auc.rf,3)
})
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
```

```
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
```

```
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
```

```
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
```

```
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
```

```
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
```

```
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
```

```
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
```

```
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
```

```
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
```

```
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
```

```
mean.auc.rf = mean(auc.rf.array)



#put cv.accuracies in a table
folds.cv = 1:K
df.cv.acc = data.frame(folds.cv,cv.accuracies)

#calculating CI
n<-K
standard.dev = sd(auc.rf.array)
CI.lower.rf = mean.auc.rf - 1.96*standard.dev/sqrt(n)
CI.upper.rf = mean.auc.rf + 1.96*standard.dev/sqrt(n)

foldsnrs = 1:K
df = data.frame(foldsnrs,round(cv.accuracies,3))
kable(df,"pipe",col.name=c("Fold","AUC"),caption=
        "Cross-Validation Results for finding Optimal Mtry")
```

Table 4: Cross-Validation Results for finding Optimal Mtry

Fold	AUC
1	0.857
2	0.860
3	0.829
4	0.856
5	0.832

Table x shows the Cross-Validation results for the Random Forest model for each fold. The mean AUC is 0.85.

Model Comparison

```
mod.names = c("Lasso","Random Forest")
CI.mins = c(CI.lower.lasso,CI.lower.rf)
CI.maxs = c(CI.upper.lasso,CI.upper.rf)
means.auc = c(mean.auc.lasso,mean.auc.rf)

df9 = data.frame(mod.names,means.auc,CI.mins,CI.maxs)
kable(df9,"pipe",col.name=c("Model","Mean AUC","Lower Bound (2.5%)","Upper Bound (97.5%)"),caption=
        "Model Performances Confidence Interval")
```

Table 5: Model Performances Confidence Interval

Model	Mean AUC	Lower Bound (2.5%)	Upper Bound (97.5%)
Lasso	0.7292779	0.7168700	0.7416857

Model	Mean AUC	Lower Bound (2.5%)	Upper Bound (97.5%)
Random Forest	0.8467146	0.8337623	0.8596669

Table 5 shows the mean and 95% Confidence Interval for the mean AUC for the Lasso and Random Forest model. From the table we can conclude that the Random Forest model does a significantly better job of classifying a certain E-mail as either Spam or Ham. **IN THE SHINY APP I WILL GIVE THE USER THE OPPORTUNITY TO TEST/USE BOTH MODELS???**

Shiny App ...