

# Homework 2 Question 2 Random Forest

*Bejan Sadeghian*

*August 18, 2015*

## Task

Use a model to determine the author of a Reuters' article based on a corpus of 2500 documents from 50 authors.

## My Method

For the second method, I used random forest. The reason I chose random forest was because in our train and test dataset (the corpus DTM) we have 50 observations per author and depending on the sparse data removal amount chosen upwards of 1000s of variables (words). This of course poses a problem for us because we have more predictors than observations. I felt random forest would be among the best solutions (PCA would be another) since it allows for us to iterate through with different variables each time we make a tree so we could have a situation where we have more observations than variables.

Again my main metric for optimization was prediction accuracy (named 'Accuracy.Score' in my code). For the random forest mtry I first decided on a ntree value then ran the code multiple times with different mtrys to find the best accuracy score. That turned out to me 3 for me. For the random forest ntree parameter I ran a CV to find where MSE was lowest quickest. I landed on ntree = 100 because the curve for MSE mostly remained the same from ntree = 100 up to 3000, so for the sake of computation expense I went with the minimum value here.

Something I did that was unique to the dataset here as opposed to the Naive Bayes method is I trained my Random Forest model with a matrix where items used less than 5% of the time were removed then used everything in the test set except the words used 1% of the time. I of course still did the intersection because that is required for the model to not be presented with new variables when predicting.

I ended up getting around 50% prediction accuracy with random forest here. It was not quite as good as I would have hoped for given Naive Bayes came out with 62% but it was respectable. I tried using LASSO and logistic regression however those prediction accuracies came to below 10%; however, I'm sure had I spent more time with those methods I could have improved the scores. This was an iterative learning process for me with Random Forest being the last model I tried.

Finally, I tabulated all of the predictions and compared them against the actual authors and found which authors were most commonly mixed up with a different author. These authors are different from the Naive Bayes set of authors primarily because of the differing models (though there are quite a few authors showing up on both methods' lists). Todd Nissen, Jonathan Brit, Carren Schuettler, Benjamin Kang Lim, Alexander Smith, Jane Macartney, and David Lawder. Also, be aware that Heather Scofield and Todd Nissen are both listed twice, it is because the number of their predictions that were correct was the same as a incorrect prediction for one of their documents. I kept them listed here because they are still considered a commonly mistaken author to me.

# Notes

I made similar notes in my Naive Bayes file for question two, please be aware there are two markdown files for question two, I did not want to combine the two methods into one markdown file because it would be a lot of knitting time with a lot of room for error. Again, I prefer the Naive Bayes method because it has around a better prediction accuracy by around 10%.

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(stringr)  
library(plyr)  
library(randomForest)
```

```
## randomForest 4.6-10  
## Type rfNews() to see new features/changes/bug fixes.
```

```

set.seed(1)
#-----
#Creating the train set
#-----
## Rolling two directories together into a single corpus
author_dirs = Sys.glob('../STA380 Homework2/ReutersC50/C50train/*')
file_list = NULL
labels = NULL
authors_list = NULL
files_to_add = NULL

readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
               id=fname, language='en') }

for(i in author_dirs){
  author = i
  author_name = substring(author, first=41)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  labels = append(labels, rep(author_name, length(files_to_add)))
}

for(j in author_dirs){
  authors_list = append(authors_list, substring(j, first=41))
}

#Pieces of this code was provided by Dr. James Scott's Naive Bayes R file.

all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
my_corpus.train = Corpus(VectorSource(all_docs))
names(my_corpus.train) = labels

# Preprocessing
my_corpus.train = tm_map(my_corpus.train, content_transformer(tolower))
my_corpus.train = tm_map(my_corpus.train, content_transformer(removeNumbers))
my_corpus.train = tm_map(my_corpus.train, content_transformer(removePunctuation))
my_corpus.train = tm_map(my_corpus.train, content_transformer(stripWhitespace))
my_corpus.train = tm_map(my_corpus.train, content_transformer(removeWords), stopwords('SM
ART'))

#Creating a DTM and a list of all terms used
DTM.train = DocumentTermMatrix(my_corpus.train, control= list(weighting = function(x) wei
ghtTfIdf(x, normalize = FALSE)))
DTM.train

```

```
## <<DocumentTermMatrix (documents: 2500, terms: 31423)>>
## Non-/sparse entries: 425955/78131545
## Sparsity           : 99%
## Maximal term length: 36
## Weighting           : term frequency - inverse document frequency (tf-idf)
```

```
DTM.train = removeSparseTerms(DTM.train, 0.95) #Removing anything uses less than 5% of the
time considering the size of this DTM
DTM.train
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 641)>>
## Non-/sparse entries: 180911/1421589
## Sparsity           : 89%
## Maximal term length: 18
## Weighting           : term frequency - inverse document frequency (tf-idf)
```

```
X.train = as.matrix(DTM.train)

#-----
#Model Fit (Random Forest)
#-----
matrix.DTM.train = as.matrix(DTM.train) #training DF
names = as.array(unique(labels))
df <- data.frame(matrix(ncol = 50, nrow = 2500)) #Initializing output DF
df[,] = 0
colnames(df) = names
location = 1
for(i in seq(1:50)){
  df[location:(location+49),i] = 1
  location = location + 50
}
response = labels
temp = cbind(response, as.data.frame(matrix.DTM.train))
rf.model = randomForest(response~., data=temp, mtry = 3, ntree=100, proximity=TRUE, importance=TRUE)

#-----
###Test Set Creation
#-----
author_dirs = Sys.glob('../STA380 Homework2/ReutersC50/C50test/*')
author_dirs = author_dirs[1:length(author_dirs)]
file_list = NULL
labels = NULL

files_to_add = NULL

for(i in author_dirs){
  author = i
```

```

author_name = substring(author, first=40)
files_to_add = Sys.glob(paste0(author, '/*.txt'))
file_list = append(file_list, files_to_add)
labels = append(labels, rep(author_name, length(files_to_add)))
names_of_files = str_sub(files_to_add, -16)
}

```

*#Pieces of this code was provided by Dr. James Scott's Naive Bayes R file.*

```

all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
my_corpus.test = Corpus(VectorSource(all_docs))
names(my_corpus.test) = labels

```

*# Preprocessing*

```

my_corpus.test = tm_map(my_corpus.test, content_transformer(tolower))
my_corpus.test = tm_map(my_corpus.test, content_transformer(removeNumbers))
my_corpus.test = tm_map(my_corpus.test, content_transformer(removePunctuation))
my_corpus.test = tm_map(my_corpus.test, content_transformer(stripWhitespace))
my_corpus.test = tm_map(my_corpus.test, content_transformer(removeWords), stopwords('SMART'))

```

*#Creating the DTM and a list of all of the terms used*

```

DTM.test = DocumentTermMatrix(my_corpus.test, control= list(weighting = function(x) weigh
tTfIdf(x, normalize = FALSE)))
DTM.test

```

```

## <<DocumentTermMatrix (documents: 2500, terms: 32264)>>
## Non-/sparse entries: 432766/80227234
## Sparsity           : 99%
## Maximal term length: 45
## Weighting          : term frequency - inverse document frequency (tf-idf)

```

```

DTM.test = removeSparseTerms(DTM.test, 0.99) #Removing anything uses less than 5% of the time considering the size of this DTM
DTM.test

```

```

## <<DocumentTermMatrix (documents: 2500, terms: 3122)>>
## Non-/sparse entries: 318288/7486712
## Sparsity           : 96%
## Maximal term length: 18
## Weighting          : term frequency - inverse document frequency (tf-idf)

```

```

X_test = as.matrix(DTM.test)

#Combining missing data from the DF
Combined = data.frame(X_test[,intersect(colnames(X_test),colnames(X.train))])
temp.table = read.table(textConnection(''), col.names=colnames(X.train),colClasses='integer')
X_eval = rbind.fill(Combined, temp.table)
X_eval[is.na(X_eval)] = 0
X_eval = X_eval[,sort(names(X_eval))]
X_eval = as.matrix(X_eval)

#-----
#Out of bag cross validation
#-----

df.DTM.test = as.data.frame(X_eval) #training DF
df.final = data.frame(Document=rep(0,2500), Predict=rep(0,2500), Prob=rep(0,2500), Actual=rep(0,2500), Match=rep(0,2500)) #empty dataframe for results
df.final[,4] = labels

#Make the prediction
predicted = predict(rf.model, df.DTM.test, type='response')
df.final[,2] = as.data.frame(predicted)

for(index in seq(1:length(file_list))){
  doc = file_list[index]
  df.final[index,1] = file_list[index]
  df[index,4] = ifelse(str_sub(substring(substring(doc, first=28), first=1, last=(nchar(substring(doc, first=28))-16)), -1) == '/', substring(substring(doc, first=28), first=1, last=(nchar(substring(doc, first=28))-17)), substring(substring(doc, first=28), first=1, last=(nchar(substring(doc, first=28))-16)))
  df.final[index,5] = ifelse(df.final[index,4] == df.final[index,2], 1, 0)
  index = index + 1
}

#-----
#Calculate Accuracy
#-----
Accuracy.Score = sum(df.final[[5]])/length(df.final[[5]])
print(Accuracy.Score)

```

```
## [1] 0.5004
```

```
#-----  
#Cross Tabulation  
#-----  
crosstabulate = xtabs(~Actual + Predict, data=df.final)  
cross.df = as.data.frame.matrix(crosstabulate)  
rownames = rownames(crosstabulate)  
colnames = rownames(crosstabulate)  
for(i in seq(1,50)){  
  if(colnames[[i]] != rownames[which(cross.df[[i]]==max(cross.df[[i]]))]){  
    print(paste0(rownames[which(cross.df[[i]]==max(cross.df[[i]]))], ' is commonly mistake  
n for ',colnames[[i]]))  
  }  
}
```

```
## [1] "ToddNissen is commonly mistaken for DavidLawder"  
## [1] "JonathanBirt is commonly mistaken for EdnaFernandes"  
## [1] "DarrenSchuettler is commonly mistaken for HeatherScoffield"  
## [2] "HeatherScoffield is commonly mistaken for HeatherScoffield"  
## [1] "BenjaminKangLim is commonly mistaken for JaneMacartney"  
## [1] "AlexanderSmith is commonly mistaken for JoeOrtiz"  
## [1] "JaneMacartney is commonly mistaken for ScottHillis"  
## [1] "DavidLawder is commonly mistaken for ToddNissen"  
## [2] "ToddNissen is commonly mistaken for ToddNissen"
```