Q2.Rmd

Stuti Madaan

August 15, 2016

1. Naive Bayes Approach

Reading the files in the directory paths and creataing list of files and authors to train on.

```
### Read Data ###
library(tm)
library(foreach)

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

## Reading in all authors' filepaths and author's names
author_dirs = Sys.glob('files/ReutersC50/C50train/*')

## Combining files and extracting Author names ##
file_list = NULL
labels = NULL
for(author in author_dirs)
{
  author_name = substring(author, first = 27)
  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)))
}

## Removing ',txt'from file names
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
```

Now we create our training Document-term Matrix

```
# Creating training corpus and processing
my_corpus = Corpus(VectorSource(all_docs))
names(my_corpus) = file_list

# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower)) # make everything lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # remove numbers
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation)) # remove punctuation
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace)) ## remove excess white-space
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwords("SMART"))

# Creating DTM
DTM = DocumentTermMatrix(my_corpus)
```

```
# Removing sparse items
DTM = removeSparseTerms(DTM, 0.975)
```

Using the DTM, we calculate probabilities for each word in the corpus, and then we calculate the conditional probabilities for each document. This list w will contain 2500 rows (1 for each document) and 1389 columns (one for each word) alongwith probability for each word.

```
# Now a dense matrix, smoothing factor and find probability vectors
X = as.matrix(DTM)
smooth_count = 1/nrow(X)

# Naive Bayes: the training sets for all the authors
w = list()
smooth_count = 1/nrow(X)
j = 1
for (i in seq(1,length(file_list),50) )
{
  w[[j]] = colSums(X[i:(i + 49),] + smooth_count)/sum(colSums(X[i:(i + 49),]
                                                              + smooth_count))
  j = j + 1
}
```

Reading the files in the directory paths and creataing list of files and authors to test on.

```
# Test Data Preparation similar to Training Data . Final DTM matrix to be used for modelling

readerPlain = function(fname){
  readPlain(elem = list(content = readLines(fname)),
            id = fname, language = 'en') }
## Rolling two directories together into a single corpus
author_dirs_test = Sys.glob('files/ReutersC50/C50test/*')
#author_dirs = author_dirs[1:2]
file_list_test = NULL
labels_test = NULL
for(author in author_dirs_test) {
  author_name = substring(author, first = 26)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list_test = append(file_list_test, files_to_add)
  labels_test = append(labels_test, rep(author_name, length(files_to_add)))
}

# Need a more clever regex to get better names here
all_docs_test = lapply(file_list_test, readerPlain)
names(all_docs_test) = file_list_test
names(all_docs_test) = sub('.txt', '', names(all_docs_test))

my_corpus_test = Corpus(VectorSource(all_docs_test))
names(my_corpus_test) = labels_test

# Preprocessing
my_corpus_test = tm_map(my_corpus_test, content_transformer(tolower)) # make everything lowercase
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeNumbers)) # remove numbers
my_corpus_test = tm_map(my_corpus_test, content_transformer(removePunctuation)) # remove punctuation
my_corpus_test = tm_map(my_corpus_test, content_transformer(stripWhitespace)) ## remove excess white-spa
my_corpus_test = tm_map(my_corpus_test, content_transformer(removeWords), stopwords("SMART"))
```

```
DTM_test = DocumentTermMatrix(my_corpus_test)
DTM_test # some basic summary statistics
```

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

Next, we remove any new words in the test dataset that the training dataset hasn't *seen* yet.

```
# Keeping only those words which we used in the training set.
common_words = colnames(DTM_test)[colnames(DTM_test) %in% colnames(DTM)]
DTM_test <- DTM_test[, common_words]
DTM_test
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 1389)>>
## Non-/sparse entries: 246565/3225935
## Sparsity           : 93%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Now we calculate the log-probabilities for each combination of document and author. THe author with the highest log-probability will be assigned to that document.

```
# Taking test documents and Comparing log probabilities
X_test = as.matrix(DTM_test)
# Creating empty matrix to calculate log-probabilities
Y_test = matrix(0, nrow = 2500, ncol = 50)
K = list()
j = 1
for (i in 1:2500)
{
  for (j in 1:50)
  {
    Y_test[i,j] = sum(X_test[i,]*log(w[[j]]))
  }
}

# Finding the document which corresponds to maximum log-probability (Hence the author)
# This can be done in a more readable way using for loop
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
author_predictions <- as.vector(t(as.data.frame(t(Y_test)) %>%
                      summarise_each(funs(which.max(.) ) ) ) )
# Since authors are arranged so well with one author for every fifty files
```

```
author_actual <- as.vector(rep(1:50,each=50))
```

Finally, we calculate the confusion matrix to see how many times we predicted the correct author.

```
#confusion matrix
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:NLP':
##
##     annotate
```

```
library(e1071) # Weird. ConfusionMatrix asked for this library
confMatrix <- confusionMatrix(author_predictions, author_actual)
confMatrix$overall["Accuracy"]
```

```
## Accuracy
##   0.6036
```

The accuracy of the Naive-Bayes model was found to be 60.4%.

2. Random Forests

The 2nd model we chose to utilize was Random forests.

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(caret)
library(e1071) # Weird. ConfusionMatrix asked for this library
set.seed(2)

rffit = randomForest(y = as.factor(rep(1:50,each=50)), x = as.matrix(DTM),
                     mtry = 50, ntree = 500)

predicted_price_rftree <- predict(rffit, newdata = as.matrix(DTM_test))
confMatrix_rf <- confusionMatrix(as.vector(predicted_price_rftree), author_actual)
```

```
## Warning in confusionMatrix.default(as.vector(predicted_price_rftree),
## author_actual): Levels are not in the same order for reference and data.
```

```
## Refactoring data to match.
```

```
confMatrix_rf$overall["Accuracy"]
```

```
## Accuracy
##   0.6144
```

The accuracy of the random forest model was found to be 61.7%.