Classifying Scientific Abstracts

Connor Segneri

10/1/2017

Contents

Project Description

The purpose for this project is to perform a classification analysis on textual data. The text that is used in this project is a collection of abstracts for scientific papers. These papers are for three subjects:

- Machine Learning
- Biology
- Psychology

The specific purpose of this project is to predict the field of study of scientific papers by analyzing their abstracts.

Data Preparation

First, the data is read into one massive data frame.

```
library(readr)

# read in machine learning articles
machLearnText = c()
for (i in 1:314) {
  fileName = paste(i,".txt",sep='')
  filePath = paste("SentenceCorpus/unlabeled_articles/arxiv_unlabeled/", fileName, sep='')
  if (file.exists(filePath)) {
    file = read_file(filePath)
    file = iconv(file, "WINDOWS-1252", "UTF-8")
    machLearnText = c(machLearnText, file)
  }
}
machLearnData = data.frame(
  text = machLearnText,
  domain = rep("Machine Learning", length(machLearnText)))
```

```
# read in psychology articles
psychText = c()
for (i in 1:345) {
  fileName = paste(i,".txt",sep='')
  filePath = paste("SentenceCorpus/unlabeled_articles/jdm_unlabeled/", fileName, sep='')
  if (file.exists(filePath)) {
    file = read_file(filePath)
    file = iconv(file, "WINDOWS-1252", "UTF-8")
    psychText = c(psychText, file)
  }
}
psychData = data.frame(
  text = psychText,
  domain = rep("Psychology", length(psychText)))

# read in biology articles
bioText = c()
for (i in 1:1657) {
  fileName = paste(i,".txt",sep='')
  filePath = paste("SentenceCorpus/unlabeled_articles/plos_unlabeled/", fileName, sep='')
  if (file.exists(filePath)) {
    file = read_file(filePath)
    file = iconv(file, "WINDOWS-1252", "UTF-8")
    bioText = c(bioText, file)
  }
}
bioData = data.frame(
  text = bioText,
  domain = rep("Biology", length(bioText)))

# combine the data into one data frame
articleData = rbind(machLearnData, psychData, bioData)
```

The first 10 rows of the data frame can be seen below.

```
dim(articleData)
```

```
## [1] 900   2
```

```
articleData[1:10,]
```

```
##
## 1
```

```
## 3  Machine Learning
## 4  Machine Learning
## 5  Machine Learning
## 6  Machine Learning
## 7  Machine Learning
## 8  Machine Learning
## 9  Machine Learning
## 10 Machine Learning
```

The domain column contains all of the field of studies, and the text column contains the raw text.

After converting the columns to the proper data types, the data frame is split into a test and training set. The `createDataPartition` function is used because it keeps the proportions for the domain equivalent across the random training and test splits.

```
# convert the domain label into a factor, and the text into characters
articleData$domain = as.factor(articleData$domain)
articleData$text = as.character(articleData$text)

# split the data into training and test datasets
library(caret)
set.seed(1)
indexes = createDataPartition(articleData$domain, times = 1, p = 0.7, list = FALSE)
articleTrain = articleData[indexes,]
articleTest = articleData[-indexes,]
```

The equivalent proportions can be seen below.

```
table(articleTrain$domain)
```

```
##
## Machine Learning       Psychology          Biology
##              210              210              210
```

```
table(articleTest$domain)
```

```
##
## Machine Learning       Psychology          Biology
##               90               90               90
```

The data is tokenized next. This means that all of the words are individually counted for each document in the corpus. The stop words are also removed. Stop words are words like “and”, “but”, “for”, and “if”, that are very common in the English language and hold no computational value. The individual terms are stemmed as well. Stemming just means that similar words are grouped together by transforming them into their roots. For example, the words “ran”, “running”, and “run” will all be transformed into the root, “run”, and that root will be given a frequency of tree.

```
library(quanteda)
```

```
## Warning: package 'quanteda' was built under R version 3.4.2
```

```
articleTrainTokens = tokens(articleTrain$text, what = "word",
                            remove_numbers=TRUE, remove_punct=TRUE,
                            remove_symbols=TRUE, remove_hyphens=TRUE)
articleTrainTokens = tokens_tolower(articleTrainTokens)

# remove stop words
articleTrainTokens = tokens_select(articleTrainTokens, stopwords(), selection = "remove")
```

```
# stem similar words into single terms
articleTrainTokens = tokens_wordstem(articleTrainTokens, language = "english")
```

Now that the data has been cleaned, a document frequency matrix can be made from the data. A document frequency matrix is a matrix that has every unique word in the corpus as the columns, and every document in the corpus as the rows. The values themselves represent the frequency of that unique word in that specific document. A truncated version of the document frequency matrix can be seen below.

```
articleDfm = dfm(articleTrainTokens, tolower = FALSE)
articleDfm[1:10,1:10]
```

```
## Document-feature matrix of: 10 documents, 10 features (52% sparse).
## 10 x 10 sparse Matrix of class "dfmSparse"
##         features
## docs     abstract paper address problem distribut learn communic
##   text1         1     3       3       2        11     7        4
##   text2         1     0       0       0         3     3        0
##   text3         1     2       0       2         4     0        0
##   text4         1     3       1       7         0    12        0
##   text5         1     3       0       3         5    11        0
##   text6         1     3       0       0         0     0        0
##   text7         1     4       0       1         1     4        0
##   text8         1     1       0      10         2     0        0
##   text9         1     2       0       3         0     4        0
##   text10        1     3       0       7         0     6        0
##         features
## docs     constraint motiv signal
##   text1           4     2      3
##   text2           0     0      0
##   text3           0     0      0
##   text4           0     0      0
##   text5           0     0      0
##   text6           0     1      0
##   text7           0     0      0
##   text8           0     1      0
##   text9           0     0      0
##   text10          0     0      0
```

Document Frequency Decision Tree Classifier

Now a decision tree can be used to classify the domains based off of the document frequency matrix. This will be done using the `rpart` to create the tree. Stratified cross validation is also used to ensure the best model is found. Cross validation is when the training data is split into k sets, one of the sets is used as test data, while the others are used as training, and the decision tree classification is performed on every different combination of the test/training splits on these sets. Stratified just means that the proportions of classes are maintained throughout the splits.

The `doSNOW` library is also used to speed up the computation time by multi-threading the computations.

```
# combine the dfm with the domain classifiers
articleDfmMatrix = as.matrix(articleDfm)
articleDfm = cbind(domain = articleTrain$domain, as.data.frame(articleDfmMatrix))
```

```
# make the column names(terms) readable by all r algorithms
names(articleDfm) = make.names(names(articleDfm))

# create stratified folds for cross validation
set.seed(1)
cvFolds = createMultiFolds(articleTrain$domain, k = 10, times = 3)
cvCntrl = trainControl(method = "repeatedcv", number = 10, repeats = 3,
                       index = cvFolds)

# multi thread using 3 cores
library(doSNOW)
```

```
## Warning: package 'doSNOW' was built under R version 3.4.2
```

```
startTime = Sys.time()
cl = makeCluster(3, type = "SOCK")
registerDoSNOW(cl)

# train a predictive classification model using a single decision tree (rpart)
rpartModel = train(domain ~ ., data = articleDfm, method = "rpart",
               trControl = cvCntrl, tuneLength = 7)

# stop the cluster and timer
stopCluster(cl)
totalTime = Sys.time() - startTime
totalTime
```

```
## Time difference of 14.17953 mins
```

The model can be seen below. The accuracy of this model is about 89%.

```
rpartModel
```

```
## CART
##
##   630 samples
## 11126 predictors
##     3 classes: 'Machine Learning', 'Psychology', 'Biology'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 567, 567, 567, 567, 567, 567, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.00952381  0.8867725  0.8301587
##   0.01666667  0.8793651  0.8190476
##   0.01904762  0.8677249  0.8015873
##   0.03452381  0.8386243  0.7579365
##   0.08333333  0.7962963  0.6944444
##   0.33571429  0.6798942  0.5198413
##   0.35952381  0.4798942  0.2198413
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.00952381.
```

TF.IDF Decision Tree Classifier

Now term frequency inverse document frequency is going to be used to form a decision tree classifier instead of just raw term frequency. This process is done much in the same way as above, and can be seen below.

```
# create the function for term frequency
tf = function(row) {
  row / sum(row)
}

# create the function for inverse docuemnt frequency
idf = function(col) {
  corpSize = length(col)
  docCount = length(which(col > 0))
  log10(corpSize / docCount)
}

# create the tf.idf function
tf.idf = function(tf, idf) {
  tf * idf
}

# find the term frequency
articleTrainTf = apply(articleDfmMatrix, 1, tf)

# find the inverse document frequency
articleTrainIdf = apply(articleDfmMatrix, 2, idf)

# find the tf.idf
articleTrainTfIdf = apply(articleTrainTf, 2, tf.idf, idf = articleTrainIdf)

# transpose the matrix so the terms are the columns again
articleTrainTfIdf = t(articleTrainTfIdf)

# check for incomplete cases and remove them
incomplete = which(!complete.cases(articleTrainTfIdf))
articleTrainTfIdf[incomplete,] = rep(0.0, ncol(articleTrainTfIdf))

# add the labels onto the data frame
articleTrainTfIdf = cbind(domain = articleTrain$domain, data.frame(articleTrainTfIdf))
names(articleTrainTfIdf) = make.names(names(articleTrainTfIdf))

# create stratified folds for cross validation
set.seed(1)
cvFolds = createMultiFolds(articleTrain$domain, k = 10, times = 3)
cvCntrl = trainControl(method = "repeatedcv", number = 10, repeats = 3,
                       index = cvFolds)

# multi thread using 3 cores
library(doSNOW)
startTime = Sys.time()
cl = makeCluster(3, type = "SOCK")
```

```
registerDoSNOW(cl)

# train a predictive classification model using a single decision tree (rpart)
rpartModelTFIDF = train(domain ~ ., data = articleTrainTfIdf, method = "rpart",
                  trControl = cvCntrl, tuneLength = 7)

# stop the cluster and timer
stopCluster(cl)
totalTime = Sys.time() - startTime
totalTime
```

```
## Time difference of 31.23724 mins
```

The model can be seen below. The accuracy is about 87%.

```
rpartModelTFIDF
```

```
## CART
##
##   630 samples
## 11126 predictors
##     3 classes: 'Machine Learning', 'Psychology', 'Biology'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 567, 567, 567, 567, 567, 567, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01428571  0.8719577  0.8079365
##   0.01666667  0.8682540  0.8023810
##   0.01904762  0.8656085  0.7984127
##   0.05000000  0.8555556  0.7833333
##   0.09285714  0.7814815  0.6722222
##   0.33333333  0.7089947  0.5634921
##   0.36904762  0.4566138  0.1849206
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01428571.
```

Comparing the Results / Predictions

The accuracy from both models show that the decision tree that was formed from just the raw term frequency actually had a higher accuracy than the TF.IDF decision tree. Both of these models are going to be used on the training set that was split from the data at the beginning of this project.

In Addition

For the 2016, Presidential election campaign for Hillary Clinton, "Researchers estimated that Google could help her win the nomination and could deliver between 2.6 and 10.4 million general election votes to Clinton

via SEME [Search Engine Manipulation Effect]" (Search Engine Manipulation Effect, Wikipedia). If you worked at Google and knew this before the election, how would you react? How would your reaction be different from Christian worldview?

If I worked at Google and knew this before the election, I would not help Hillary win. Manipulating what people see seems like a tyrannical thing to do. The information that people receive off a searches should remain impartial and unbiased. From a Christian worldview, my reaction would be the same.

References

D. (2017, June 05). Introduction to Text Analytics with R - Part 1: Overview. Retrieved October 02, 2017, from https://www.youtube.com/watch?v=4vuw0AsHeGw&index=1&list=PL8eNk_zTBST8olxIRFoo0YeXxEOkYdoxi