Week 11

Dan Brooks

April 6, 2016

Sentiment Analysis

Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. Sentiment analysis is widely applied to:

- Reviews
- Social Media
- Marketing
- Customer Service
- And Many More

Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. The attitude may be his or her judgment or evaluation (see appraisal theory), affective state (that is to say, the emotional state of the author when writing), or the intended emotional communication (that is to say, the emotional effect the author wishes to have on the reader).

Spam vs Non-Spam ("Ham")

This assignment will take the idea of sentiment analaysis and apply it to a "spam filter", It will take e-mails that are labeld as spam or ham and train a model to pick whether future e-mails are spam or ham nbased off of key words that are present in the training sets.

The following packages are required for this assigment:

- tm
- plyr
- class
- foreign
- gmodels
- RTextTools
- data.table
- R.utils

Reading and Manupulating the files

There is a need to read in all of the files that will be used in the training and testing of the model. The process is as follows:

1. Read in all the necesarry e-mails/text
2. create a Corpus (takes all the individual documents and combines them into one)

3. Clean the Corpus
    - Remove whitespace
    - Remove stop words (the, and, as etc.)
    - Remove Punctuation
4. Create Term Document Matrix (Takes all words in Corpus and creates a frquency matrix)

Read Files

```
#Read in the files and create the term document matrix
generateTDM <- function(types, pathname){
  s.dir <- sprintf("%s/%s", pathname, types)
  s.cor <- Corpus(DirSource(directory = s.dir, encoding="latin1"))
  s.cor.cl <- cleanCorpus(s.cor)
  s.tdm <- TermDocumentMatrix(s.cor.cl)

  s.tdm <- removeSparseTerms(s.tdm, 0.9)
  return(list(name = types, tdm = s.tdm))
}
```

Clean Files

```
#clean the corpus to get rid of all the unneccessary information
cleanCorpus <- function(corpus){
corpus.tmp <- tm_map(corpus, removePunctuation)
corpus.tmp <- tm_map(corpus.tmp, stripWhitespace)
corpus.tmp <- tm_map(corpus.tmp, tolower)
corpus.tmp <- tm_map(corpus.tmp, removeWords, stopwords("english"))
corpus.tmp <- tm_map(corpus.tmp, PlainTextDocument)

return(corpus.tmp)
}
```

Creating the Matrix

The term document matrix is created for both the spam and ham files. We assign the matricies different names, so we can manipulate each matrix individually. The last rown of the matrix does describe the email as either *spam* of *ham*. This is to allow the model to check the correctness later on.

```
bindTypeToTDM <- function(tdm){
  s.mat <- t(data.matrix(tdm[["tdm"]]))
  s.df <- as.data.frame(s.mat, stringsAsFactors = FALSE)

  s.df <- cbind(s.df, rep(tdm[["name"]], nrow(s.df)))
  colnames(s.df)[ncol(s.df)] <- "types"

  return(s.df)
}
```

Create the Term Document Matrix (TDM)

Creates the Term Document Matrix for all of the e-mails ine both the spam and ham files and combines the two matricies into one total matrix. The *NA* values will be filled with 0's to show that they did not appear in the given e-mail set. This matrix will have the overall frquency for all the terms taht appear in both the spam and ham e-mails.

```
tdm <- lapply(types, generateTDM, path = pathname)
```

```
typeTDM <- lapply(tdm, bindTypeToTDM)
tdm.stack <- do.call(rbind.fill, typeTDM)
tdm.stack[is.na(tdm.stack)] <- 0
```

Training and Hold-Out Data

After the Term Document Matrix is created, there is a need to test the model. That is done by parsing out the term document matrix into two parts:

1. Training Data Set (*train.idx*)

The training Data set, is the data set that will be used to train the model. The TDM is fed into the model and the model either creates a tree, or groups the data into sets/groups. The model is learning what type of data belongs to what groups and can use that to categorize future data.

2. Hold-Out Data Set (*test.idx*)

The hold out data is the data that will be used to test the model. This data is fed into the model and used to determine the accuracy. The hold out data will be placed into whatever groups the model determined by the training data set.

```
train.idx <- sample(nrow(tdm.stack), ceiling(nrow(tdm.stack) * 0.7))
test.idx <- (1:nrow(tdm.stack)) [-train.idx]
```

K- Nearest Neighbor (KNN) Model

The KNN model uses euclidean distance to figure out where the hold out data belongs in the trainin set. Each groups *spam/ham* has a list of points (determined by the TDM), Those points can be ploted on a graph (n-dimensional space). The new data that is brought in looks to see where their points line up with the two groups (euclidean distance). Whichever group it is closer to, that is where the model will place the data.

```
tdm.type <- tdm.stack[, "types"]
tdm.stack.nl <- tdm.stack[,!colnames(tdm.stack) %in% "types"]
knn.prediction <- knn(tdm.stack.nl[train.idx, ], tdm.stack.nl[test.idx, ], tdm.type[train.idx])
test <- CrossTable(x = tdm.type[test.idx], y = knn.prediction, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
```

```
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  794
##
##
##                     | knn.prediction
## tdm.type[test.idx] |  hard_ham |      spam | Row Total |
## -------------------|-----------|-----------|-----------|
##           hard_ham |        50 |        20 |        70 |
##                    |     0.714 |     0.286 |     0.088 |
##                    |     1.000 |     0.027 |           |
##                    |     0.063 |     0.025 |           |
## -------------------|-----------|-----------|-----------|
##               spam |         0 |       724 |       724 |
##                    |     0.000 |     1.000 |     0.912 |
##                    |     0.000 |     0.973 |           |
##                    |     0.000 |     0.912 |           |
## -------------------|-----------|-----------|-----------|
##       Column Total |        50 |       744 |       794 |
##                    |     0.063 |     0.937 |           |
## -------------------|-----------|-----------|-----------|
##
##
```

Suppoart Vector Machines (SVM) Model

- This model employs spacial representation of the data
- It tries to fit vectors between the document features that best separate the document into the various groups
- Selects the vectors in a way that they maximize the space between the groups
- Check to see which side of the vectors the features of unlabled documents come to lie and estimates the categorical membership accordingly

```
svm_model <- train_model(container, "SVM")
svm_out <- classify_model(container, svm_model)
test <- CrossTable(x = tdm.type[testSize], y = svm_out[,1], prop.chisq = FALSE )
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
```

```
##
##
## Total Observations in Table:  794
##
##
##                     | svm_out[, 1]
## tdm.type[testSize] |  hard_ham |      spam | Row Total |
## -------------------|-----------|-----------|-----------|
##           hard_ham |        66 |         4 |        70 |
##                    |     0.943 |     0.057 |     0.088 |
##                    |     1.000 |     0.005 |           |
##                    |     0.083 |     0.005 |           |
## -------------------|-----------|-----------|-----------|
##               spam |         0 |       724 |       724 |
##                    |     0.000 |     1.000 |     0.912 |
##                    |     0.000 |     0.995 |           |
##                    |     0.000 |     0.912 |           |
## -------------------|-----------|-----------|-----------|
##       Column Total |        66 |       728 |       794 |
##                    |     0.083 |     0.917 |           |
## -------------------|-----------|-----------|-----------|
##
##
```

Random Forest (Tree) Model

Creates Multiple decision tress and takes the most frequnetlt predicted membership category of many decision trees as the classification that is most likely to be accurate.

```
tree_model <- train_model(container, "TREE")
tree_out <- classify_model(container, tree_model)
test <- CrossTable(x = tdm.type[testSize], y = tree_out[,1], prop.chisq = FALSE )
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  794
##
##
##                     | tree_out[, 1]
## tdm.type[testSize] |  hard_ham |      spam | Row Total |
## -------------------|-----------|-----------|-----------|
##           hard_ham |        65 |         5 |        70 |
##                    |     0.929 |     0.071 |     0.088 |
```

```
##                     |     1.000 |     0.007 |           |
##                     |     0.082 |     0.006 |           |
## --------------------|-----------|-----------|-----------|
##                spam |         0 |       724 |       724 |
##                     |     0.000 |     1.000 |     0.912 |
##                     |     0.000 |     0.993 |           |
##                     |     0.000 |     0.912 |           |
## --------------------|-----------|-----------|-----------|
##       Column Total |        65 |       729 |       794 |
##                     |     0.082 |     0.918 |           |
## --------------------|-----------|-----------|-----------|
##
##
```

Maximum Entropy (MAXENT) Model

- analogous to the multinomian logit function
- generalizes the logit model to place documents into categories based off of probabilites.

```
maxent_model <- train_model(container, "MAXENT")
maxent_out <- classify_model(container, maxent_model)
test <- CrossTable(x = tdm.type[testSize], y = maxent_out[,1], prop.chisq = FALSE )
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  794
##
##
##                     | maxent_out[, 1]
## tdm.type[testSize] |  hard_ham |      spam | Row Total |
## --------------------|-----------|-----------|-----------|
##           hard_ham |        68 |         2 |        70 |
##                     |     0.971 |     0.029 |     0.088 |
##                     |     0.986 |     0.003 |           |
##                     |     0.086 |     0.003 |           |
## --------------------|-----------|-----------|-----------|
##                spam |         1 |       723 |       724 |
##                     |     0.001 |     0.999 |     0.912 |
##                     |     0.014 |     0.997 |           |
##                     |     0.001 |     0.911 |           |
## --------------------|-----------|-----------|-----------|
##       Column Total |        69 |       725 |       794 |
##                     |     0.087 |     0.913 |           |
```

```
## -------------------|-----------|-----------|-----------|
##
##
```

Accuracy of Model Measurement

There are three ways to which you can measure the accuracy of your model:

1. Precision - fraction of retrieved instances that are relevant
2. Recall - is the fraction of relevant instances that are retrieved
3. F-Measure - Combies precision and recall to give and overall ratio of the model

Classification Context

Relevant/Retrieved	Correct	Not Correct
Selected	TP	FP
Not Selected	FN	TN

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F - Measure = \frac{2 * precision * recall}{precision + recall}$$

```
##     Model Precision    Recall F-Measure
## 1:    KNN 0.7142857 1.0000000 0.8333333
## 2:    SVM 0.9428571 1.0000000 0.9705882
## 3:   TREE 0.9285714 1.0000000 0.9629630
## 4: MAXENT 0.9714286 0.9855072 0.9784173
```