Ex4-TextMininig

Authors: **Ronit Yoari & Orel Swisa**

***Date*: June 12, 2016**

Competitions Page

Load Required Library

```
library(stringdist)
library(tm)
library(readr)
library(rpart)
library(party)
library(caret)
library(randomForest)
library(SnowballC)
```

Helper Funtions

Clean Text

This function uses text mining techniques on a received column, in order to convert the current input attributes into some smarter set of attributes that will help to predict relevance ranks.

```
cleanText <- function(col){
  train_product_title_source <- VectorSource(col)
  corpus<- Corpus(train_product_title_source)
  corpus<-tm_map(corpus,removePunctuation)
  corpus<-tm_map(corpus,tolower)
  corpus<-tm_map(corpus,removeNumbers)
  corpus<-tm_map(corpus,removeWords,stopwords("english"))
  corpus<-tm_map(corpus,stripWhitespace)
  for(j in seq(corpus))
  {
```

```
    corpus[[j]] <- gsub("/", " ", corpus[[j]])
    corpus[[j]] <- gsub("@", " ", corpus[[j]])
    corpus[[j]] <- gsub("\\|", " ", corpus[[j]])
    corpus[[j]] <- stemSentence(corpus[[j]],"english")
  }
  corpus <- tm_map(corpus, PlainTextDocument)
  return(corpus)
}
```

Stemming

This function uses stemming techniques on a received sentence, in order to transforming a sentence into its stem (normalized form).

```
stemSentence<-function(x, language){
  x<- strsplit(x, "[[:blank:]]")[[1]]
  x<- wordStem(x, language)
  paste(x, collapse=" ")
}
```

Generate Score

This function, receives a file name and performs on it several operations:

1. Load the file.
2. Clean the file with cleanText function.
3. Create 'DocumentTermMatrix' in order to use it for calculation.
4. Caculate Similarity between the query colomn to title & description.
5. On the basis of the similarity calculating -> Score colomn is generate. In the end, the file with the score colomn is returend.

```
claculateScore<-function(fileName){
  # Loading the data
  unzip(paste(fileName, ".csv.zip", sep = ""))
  sourceFile <- read_csv(paste(fileName, ".csv", sep = ""))
  File_Data  <- read.csv(paste(fileName, ".csv", sep = ""),stringsAsFactors = FALSE)
  # Preprocessing - Clean Text
  querys_Clean      <- cleanText(File_Data$query)
  title_Clean       <- cleanText(File_Data$product_title)
  description_Clean <- cleanText(File_Data$product_description)
  # Create DTM for each colomn base on dictionary=Terms(query_DTM)
  query_DTM       <- DocumentTermMatrix(querys_Clean, control=list(bounds = list(local = c(0, Inf)),
                                                                   dictionary=NULL,
                                                                   wordLengths=c(2,Inf)))
  title_DTM       <- DocumentTermMatrix(title_Clean,control=list(bounds = list(local = c(0, Inf)),
```

```
                                                                dictionary=Terms(query_DTM),
                                                                wordLengths=c(2,Inf)))
  description_DTM <- DocumentTermMatrix(description_Clean,control=list(bounds = list(local = c(0, Inf))
                                                                       dictionary=Terms(query_DTM),
                                                                       wordLengths=c(2,Inf)))
  # Caculate Similarity between: query<->title & query<->description
  query_title            <- as.matrix(query_DTM)*as.matrix(title_DTM)
  query_title_simi       <- rowSums(query_title)
  query_description      <- as.matrix(query_DTM)*as.matrix(description_DTM)
  query_description_simi <- rowSums(query_description)
  # Calculate Score
  query_DF<-data.frame(text=unlist(sapply(querys_Clean, `[`, "content")),
                       stringsAsFactors=F)
  query_length<-calculateLength(query_DF)
  for(j in seq(sourceFile$id))
  {
    if(sourceFile$product_description[[j]]==""){
      sourceFile$score[[j]] <- query_title_simi[j]/query_length[j]
    }
    else{
      sourceFile$score[[j]] <- (query_title_simi[j]+
                                  query_description_simi[j])/(2*query_length[j])
    }
  }
  return(sourceFile)
}
```

```
# Calculate Score for the train file
train <- claculateScore("train")
# Calculate Score for the train file
test <- claculateScore("test")

train$median_relevance <- factor(train$median_relevance)
```

Model creation

Let's train a classification model based on the training set.

Random Forest:

```
model1 <- randomForest(median_relevance ~ score, data=train, ntree=30)
```

Decision Tree:

```
model2 <- train(median_relevance ~ score, data = train,
                method = "rpart",
                trControl = trainControl(classProbs = F))
```

Classification for Prediction

Classifying the test.csv data and exporting the results to a submission file.

```
results <- predict(model1, newdata = test)
Newsubmission = data.frame(id=test$id, prediction = results)
write.csv(Newsubmission,"model.csv",row.names=F)
```

Results

In our solution we used:

1. Two types of Model: *Decision Tree* & *Random Forest*.
2. Two types of methodes to generate the score.

Option 1

Option Number	Model	Generate Score Type
1	*Decision Tree*	1

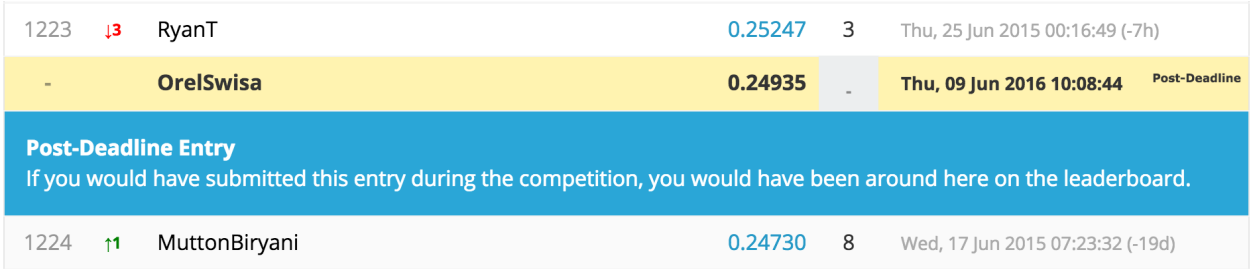

Figure 1: Result Option 1

Option 2

Option Number	Model	Generate Score Type
2	*Decision Tree*	2

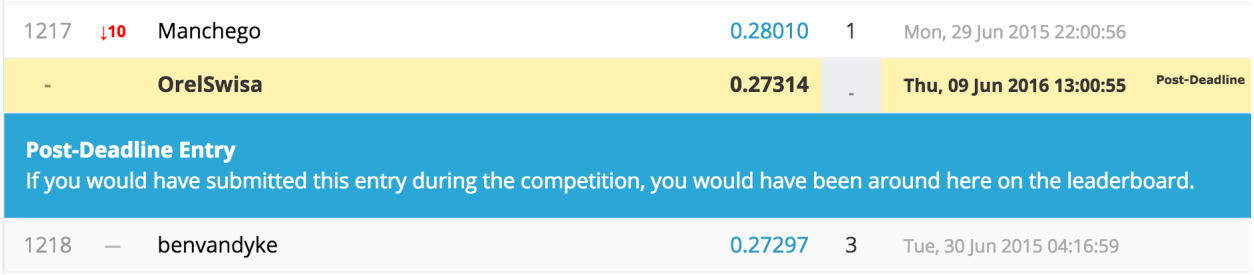

Figure 2: Result Option 2

Option 3

Option Number	Model	Generate Score Type
3	*Random Forest*	1

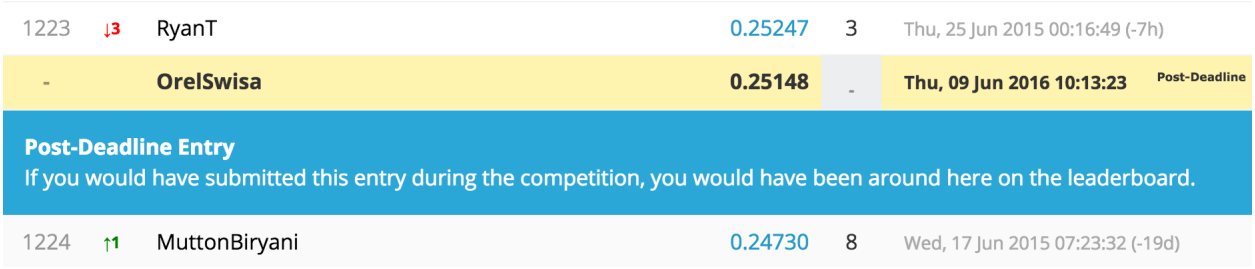

Figure 3: Result Option 3

Option 4

Option Number	Model	Generate Score Type
4	*Random Forest*	2

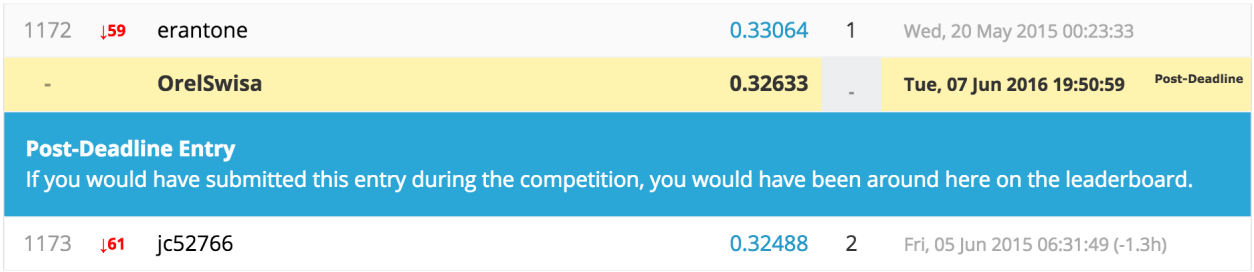

Figure 4: Result Option 4