TextMining

HilaReut

11 June 2016

Crowdflower Search Results Relevance

This task was to predict the relevance of search results from eCommerce sites.

Our first approach was to organize the data: We took the words from query,product_title,product_description fields and we remove unnecessary data like spaces, puncturation ,common words and extra and we tried to get the similarity between query-title, query-description. but unfortuntally we didn't get high score . so we though to use the familier tf/idf to improve the prediction.

We use the example from the ppt "TextMining1". We needed to think how to arrange the data so we can use this algorithm.

each word was word from the query words . each document was row in our data . we run the tf/idf for words from the query but we look for them in the words from the title/description. we took the higher score we get from the tf/idf.This improve our score .

In addition we used the feature from the Accompanied Code??

Loading the data

```
# Loading the data
unzip("train.csv.zip")
```

```
## Warning in unzip("train.csv.zip"): error 1 in extracting from zip file
```

```
train  <- read.csv("train.csv")
unzip("test.csv.zip")
```

```
## Warning in unzip("test.csv.zip"): error 1 in extracting from zip file
```

```
test  <- read.csv("test.csv")
```

Preprocessing

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

Extract your smart features

First feature - each textual attribute is truncated to its first character.

```
#Preprocess the train data
train$query_f1 <- factor(substr(train$query,1,1),)
train$product_title_f1 <- factor(tolower(substr(train$product_title,1,1)))
train$product_description_f1 <- factor(tolower(substr(train$product_description,1,1)))
```

```
#Preprocess the test data as well
test$query_f1 <- factor(substr(test$query,1,1))
test$product_title_f1 <- factor(tolower(substr(test$product_title,1,1)))
test$product_description_f1 <- factor(tolower(substr(test$product_description,1,1)))

levels(train$query_f1) <- union(levels(train$query_f1), levels(test$query_f1))
levels(train$product_title_f1) <- union(levels(train$product_title_f1), levels(test$product_title_f1))
levels(train$product_description_f1) <- union(levels(train$product_description_f1), levels(test$product
levels(test$query_f1) <- union(levels(train$query_f1), levels(test$query_f1))
levels(test$product_title_f1) <- union(levels(train$product_title_f1), levels(test$product_title_f1))
levels(test$product_description_f1) <- union(levels(train$product_description_f1), levels(test$product_
```

Second Feature

Similarity between two groups of words. 1:query-product_title; 2:query-product_description.

using set_similarity(A, B) function.

```
myfunction <- function(text){
  df <- data.frame(text, stringsAsFactors = FALSE);
  docs <- Corpus(VectorSource(df$text));
  docs <- tm_map(docs, content_transformer(tolower)); # convert all text to lower case
  as.character(docs[[1]]);
  docs <- tm_map(docs, removePunctuation); # remove Puncturation
  as.character(docs[[1]]);
  docs <- tm_map(docs, removeNumbers); # remove Numbers
  as.character(docs[[1]]);
  docs <- tm_map(docs, removeWords, stopwords("english")); # remove common words
  as.character(docs[[1]]);
  docs <- tm_map(docs, stripWhitespace); # strip white space
  as.character(docs[[1]]);
  docs <- tm_map(docs, stemDocument); # stem the document
  as.character(docs[[1]]);
  query_words = scan_tokenizer(as.character(docs[[1]]));
  return(query_words);
}

 train$myfeature <- 0;
 test$myfeature <- 0;
 train$myfeature_title <- 0;
 test$myfeature_title <-0;

for (i in 1:nrow(train)){
  words_query = myfunction(train$query[i]);
  words_product_title = myfunction(train$product_title[i]);
  words_product_desc = myfunction(train$product_description[i]);
  train$myfeature[i] = set_similarity(as.set(words_query),as.set(words_product_desc));
  train$myfeature_title[i] = set_similarity(as.set(words_query),as.set(words_product_title));

}
```

```
for (i in 1:nrow(test)){
  words_query_test = myfunction(test$query[i]);
  words_product_title_test = myfunction(test$product_title[i]);
  words_product_desc_test = myfunction(test$product_description[i]);
  test$myfeature[i] = set_similarity(as.set(words_query_test),as.set(words_product_desc_test));
  test$myfeature_title[i] = set_similarity(as.set(words_query_test),as.set(words_product_title_test));

}
levels(train$myfeature) <- union(levels(train$myfeature), levels(test$myfeature))
levels(test$myfeature) <- union(levels(train$myfeature), levels(test$myfeature))
levels(train$myfeature_title) <- union(levels(train$myfeature_title), levels(test$myfeature_title))
levels(test$myfeature_title) <- union(levels(train$myfeature_title), levels(test$myfeature_title))


###Third Feature
####tfidf algorithm .

tfidfFunction <- function(text,i){
  #print(text)
  df <- data.frame(text, stringsAsFactors = FALSE);
  docs <- Corpus(VectorSource(df$text));
  docs <- tm_map(docs, content_transformer(tolower)); # convert all text to lower case
  #print(as.character(docs[[1]]));
  docs <- tm_map(docs, removePunctuation); # remove Puncturation
  #print(as.character(docs[[1]]));
  docs <- tm_map(docs, removeNumbers); # remove Numbers
  #print(as.character(docs[[1]]));
  docs <- tm_map(docs, removeWords, stopwords("english")); # remove common words
  #print(as.character(docs[[1]]));
  docs <- tm_map(docs, stripWhitespace); # strip white space
  #print(as.character(docs[[1]]));
  docs <- tm_map(docs, stemDocument); # stem the document
  #print(as.character(docs[[1]]));
  docs <- tm_map(docs, stripWhitespace)   #Stripping unnecesary whitespace from your documents
  #print(as.character(docs[[1]]));
  docs <- tm_map(docs, PlainTextDocument) #This tells R to treat your preprocessed documents as text do
  mydtm <- DocumentTermMatrix(docs)
  #print(mydtm$dimnames$Terms);
  term_tfidf <-tapply(mydtm$v/slam::row_sums(mydtm)[mydtm$i], mydtm$j, mean) * log2(nrow(train)/slam::c
  #choose the max val for query words
  myarr = cbind(mydtm$dimnames$Terms,term_tfidf);
  maxval = 0;
  if (!is.na(train$query[i])){
  query = scan_tokenizer(train$query[i]);
  for (x in 1:length(myarr)){
    word = myarr[x];
    for (y in 1:length(query)){
     if (!is.na(word) && !(is.na(query[y]))){
       if (word == query[y]){
          if (maxval < myarr[x,2])
           maxval = myarr[x,2];
        }
      }
```

```
    }
  }
  }
  return(maxval)

}

for (i in 1:nrow(train)){
  u_t = intersect(scan_tokenizer(train$product_title[i]),scan_tokenizer(train$query[i]));
  u_d = intersect(scan_tokenizer(train$product_description[i]),scan_tokenizer(train$query[i]));
  #  print("u_t");
  if (length(u_t) > 0){
    # print(u_t);
    title = union(scan_tokenizer(train$query[i]),scan_tokenizer(train$product_title[i]));
    train$tfidf_title[i] = tfidfFunction(title,i);
  }

  else{
    train$tfidf_title[i] = 0;
  }
  if (length(u_d) >0){
    desc = union(scan_tokenizer(train$query[i]),scan_tokenizer(train$product_description[i]));
  }
  else{
    # print("empty");
    train$tfidf_desc[i] = 0;

  }

}


for (i in 1:nrow(test)){
  u_t = intersect(scan_tokenizer(test$product_title[i]),scan_tokenizer(test$query[i]));
  u_d = intersect(scan_tokenizer(test$product_description[i]),scan_tokenizer(test$query[i]));
  #  print("u_t");
  if (length(u_t) > 0){
    # print(u_t);
    title = union(scan_tokenizer(test$query[i]),scan_tokenizer(test$product_title[i]));
    test$tfidf_title[i] = tfidfFunction(title,i);
  }
  else{
    #   print("empty");
    test$tfidf_title[i] = 0;
  }
  #print("u_d");
  if (length(u_d) >0){
    #  print(u_d);
    desc = union(scan_tokenizer(test$query[i]),scan_tokenizer(test$product_description[i]));
  }
  else{
    # print("empty");
    test$tfidf_desc[i] = 0;
```

```
  }

}

levels(train$tfidf_title) <- union(levels(train$tfidf_title), levels(test$tfidf_title))
levels(train$tfidf_desc) <- union(levels(train$tfidf_desc), levels(test$tfidf_desc))
levels(test$tfidf_title) <- union(levels(train$tfidf_title), levels(test$tfidf_title))
levels(test$tfidf_desc) <- union(levels(train$tfidf_desc), levels(test$tfidf_desc))

###Model Creation

####Random Forest:

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
```

```
model <- randomForest(median_relevance ~ query_f1+product_title_f1+product_description_f1+myfeature+myf
results <- predict(model, newdata = test)
#results = round(results)
Newsubmission = data.frame(id=test$id, prediction = results)
write.csv(Newsubmission,"model.csv",row.names=F)
```