Book reviews classifier in Amazon - edX Data science course PH125.9x

Gianluca Turco

April 18, 2019

Introduction

Text classification is the process of assigning tags or categories to text. It is one of the fundamental tasks in Natural Language Processing (NLP), with applications such as language translation, sentiment analysis, topic labeling, spam detection, and intent detection. Machine learning techniques can effectively be used to classify free text into a series of predefined thematic categories.

For this project, we will build a quick classifier algorithm based on Amazon book reviews. The objective is to predict with a fairly high degree of accuracy if a review is positive or negative. To this end, we will use a small subset of the Amazon review data published on the Stanford Network Analysis Project website (https://snap.stanford.edu/data/web-Amazon.html). The full dataset includes approximately 35 million Amazon reviews spanning from 1995 to 2013.

Our subset has been parsed into small text chunks and labelled appropriately, and can be retrieved in this Github repository (https://github.com/glturco/Amazon_reviews). The "Training" data contains 400 1-star book reviews labeled "Neg" (for negative) and 400 5-star book reviews labelled "Pos" (for positive). We will use the files in the "Training" folder to train our model and predict whether or not the reviews in our "Test" directory (400 reviews in all) are negative or positive. This machine learning classifier should be able to predict whether an Amazon book review - or any short text - reflects a positive or a negative customer experience with a given product.

The first part of the project will focus on cleaning the data. We will then train our algorithm, analyse the results and conclude. All the analysis will be conducted using the `dplyr`, `tm`, `kernlab`, `caret`, `splitstackshape` and `e1071` packages.

Note:

- For the sake of clarity in the context of this project, I have included in the report most of the code. This would not be the case in a real working environment.

- I was not able to install on the hardware at my disposal Tex, which is required to create a PDF output in R Markdown. I had to extract the output in HTML format and then convert it into PDF. I have left `pdf_document` as output in this file in case somebody wants to run it.

1. Data preparation

1.1 Ingestion and cleaning of the training data

First, we will download and load all the R packages needed to execute the analysis. Then, we will need to convert the plain text data into a corpus, i.e. a collection of documents which can be processed using the `tm` package. The `VCorpus` function from the `tm` package will be used to this effect.

```
#The following commands will install the necessary libraries, if needed, and load the
m.

if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(tm)) install.packages("tm", repos = "http://cran.us.r-project.org")
if(!require(kernlab)) install.packages("kernlab", repos = "http://cran.us.r-project.or
g")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(splitstackshape)) install.packages("splitstackshape", repos = "http://cra
n.us.r-project.org")
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")

### Step 1.1 Ingest your training data and clean it. ###

# This is the folder in which I have saved the data.  Replace it with yours if you wan
t to run this code.
path_train <- "C:/Users/TURCOGI/R/Capstone/My_project/Training"
path_test <- "C:/Users/TURCOGI/R/Capstone/My_project/Test"

# The VCorpus function from the tm package will create a volatile corpus from the trai
n data.
train <- VCorpus(DirSource(path_train, encoding = "UTF-8"), readerControl=list(languag
e="English"))
```

The resulting object has two object classes, `Vclass` and `Corpus`. We can inspect its first few elements:

```
class(train)
```

```
## [1] "VCorpus" "Corpus"
```

```
inspect(train[1:10])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 10
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 372
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 135
##
## [[3]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 2036
##
## [[4]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 1457
##
## [[5]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 6381
##
## [[6]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 837
##
## [[7]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 743
##
## [[8]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 546
##
## [[9]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 487
```

```
##
## [[10]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 1160
```

The text must now be cleaned using the `tm_map` function, in order to make our data usable by the classifier. In particular, we will strip unnecessary white space, convert everything to lower case (since the `tm` package is case sensitive), remove numbers, punctuation and English common words like ‘the’ (so-called ‘stopwords’).

```
# Let's clean up the data by collapsing extra whitespace to a single blank:
train <- tm_map(train, content_transformer(stripWhitespace))

# Convert all text to lower case
train <- tm_map(train, content_transformer(tolower))

# Remove numbers and punctuation
train <- tm_map(train, content_transformer(removeNumbers))
train <- tm_map(train, content_transformer(removePunctuation))

# This will remove English stopwords, i.e. English common words like 'the'
train <- tm_map(train, removeWords, stopwords("english"))
```

1.2 Create a Document-Term Matrix (DTM) for the training data

The next step is to create a Document-Term Matrix (DTM) with our `train` data. DTM is a matrix that lists all occurrences of words in the corpus. In a DTM, documents are represented by rows and the terms (or words) by columns. If a word occurs in a particular document `n` times, then the matrix entry for corresponding to that row and column is `n` , if it does not occur at all, the entry is 0.

```
### Step 1.2 Create your document term matrices for the training data. ###

train_dtm <- as.matrix(DocumentTermMatrix(train, control=list(wordLengths=c(1,Inf))))
```

This is the structure of our train data as a Document-Term Matrix:

```
str(train_dtm)
```

```
 num [1:800, 1:11461] 0 0 0 0 1 0 0 0 0 0 ...
 - attr(*, "dimnames")=List of 2
  ..$ Docs : chr [1:800] "Neg_1.0_A100NXYDA6HSKA.txt" "Neg_1.0_A106CGPJUBDG0P.txt" "Ne
g_1.0_A10KECJUJOBO9H.txt" "Neg_1.0_A10VGXGJD9JMOC.txt" ...
  ..$ Terms: chr [1:11461] "abandon" "abandonded" "abandoned" "abandoning" ...
```

1.3 Ingest, clean and create a Document-Term Matrix (DTM) for the test data

We will now repeat the steps above for the `test` data.

```
# Clean test data
test <- VCorpus(DirSource(path_test, encoding = "UTF-8"), readerControl=list(language
="English"))
test <- tm_map(test, content_transformer(stripWhitespace))
test <- tm_map(test, content_transformer(tolower))
test <- tm_map(test, content_transformer(removeNumbers))
test <- tm_map(test, content_transformer(removePunctuation))
test <- tm_map(train, removeWords, stopwords("english"))

# Create a document-term matrix  for the test data:
test_dtm <- as.matrix(DocumentTermMatrix(test, control=list(wordLengths=c(1,Inf))))
```

The code in paragraphs 2 and 3 creates two new data matrices: one `train_dtm`, containing all of the words from the “Training” folder, and a `test_dtm` matrix, containing all of the words from the “Test” folder. For most of the following parts of this project, we will be working with `train_dtm` in order to create, train, and validate our results.

1.4 Make sure `test` and `train` matrices are of identical length

We need to further manipulate our data in order to create a functioning classifier. The first step is to make sure that our datasets have the same number of columns, so that we only take overlapping words from both matrices.

```
# Clean test data
train_df <- data.frame(train_dtm[,intersect(colnames(train_dtm), colnames(test_dtm))])
test_df <- data.frame(test_dtm[,intersect(colnames(test_dtm), colnames(train_dtm))])
```

Here is the result:

```
str(train_df, list.len = 10)
```

```
'data.frame':	800 obs. of  11461 variables:
 $ abandon                                      : num  0 0 0 0 1 0 0 0 0 0 ...
 $ abandonded                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abandoned                                    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abandoning                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abbaacutes                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abberation                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abdulbahaacute                               : num  0 0 0 0 0 0 0 0 0 0 ...
 $ aberration                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ ability                                      : num  0 0 0 0 0 0 1 0 0 0 ...
 $ able                                         : num  1 0 0 0 0 0 0 0 0 0 ...
  [list output truncated]
```

```
str(test_df, list.len = 10)
```

```
'data.frame':	800 obs. of  11461 variables:
 $ abandon                                      : num  0 0 0 0 1 0 0 0 0 0 ...
 $ abandonded                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abandoned                                    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abandoning                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abbaacutes                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abberation                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ abdulbahaacute                               : num  0 0 0 0 0 0 0 0 0 0 ...
 $ aberration                                   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ ability                                      : num  0 0 0 0 0 0 1 0 0 0 ...
 $ able                                         : num  1 0 0 0 0 0 0 0 0 0 ...
  [list output truncated]
```

1.5 Adjust the labels

Our data must have a column that dictates whether the files are “Neg” (negative) or “Pos” (positive). Since we know these values for the training data, we have to separate the labels from the original filenames and append them to the “corpus” column in the data. For our testing data, we do not have these labels, so we will add dummy values instead (that will be filled later).

```
label_df <- data.frame(row.names(train_df))
colnames(label_df) <- c("filenames")
label_df<- cSplit(label_df, 'filenames', sep="_", type.convert=FALSE)
train_df$corpus<- label_df$filenames_1
test_df$corpus <- c("Neg")
```

2. Model building

We are now ready to build our classifier. It is important to note that we will not be running cross-validation of the model for the scope of this project. In a more advanced scenario, we should create folds within the data and cross-validate our model across multiple cuts of the data in order to be sure that the results are accurate. In this simple scenario, we will only run one validation and use the confusion matrix to measure the accuracy of our predictive machine learning model.

We will use the `train` dataframe for both training our model (with a radial basis function kernel) and testing it.

```
# We will start by using the training dataframes for both training and testing our mod
el:
df_train <- train_df
df_test <- train_df
df_test$corpus <- as.factor(df_test$corpus)
# We will use this kernel for training our algorithm.
df_model<-ksvm(corpus~., data= df_train, kernel="rbfdot")
# Predict:
df_pred<-predict(df_model, df_test)
```

And here is the confusion matrix, which tabulates each combination of our prediction and the actual value.

```
con_matrix<-confusionMatrix(df_pred, df_test$corpus)
print(con_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Neg Pos
       Neg 329   0
       Pos  71 400

               Accuracy : 0.9112
                 95% CI : (0.8894, 0.93)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8225
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8225
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.8493
             Prevalence : 0.5000
         Detection Rate : 0.4113
   Detection Prevalence : 0.4113
      Balanced Accuracy : 0.9113

       'Positive' Class : Neg
```

Results

The ‘Accuracy’ field gives us a quick estimate of the percentage of the files predicted correctly by the classifier, which is a satisfactory ~91%. In other words, in approximately 91% of the cases our classifier was successful in determining whether or not a file was positive or negative just based on its content.

We can now run the final prediction on the `test` data and recreate the file names. The code below runs the `predict()` model on the `test` data, and adds the results to the `results` dataframe. The original filenames are then re-attached to the row names of the `results` dataframe, clearly showing the predictions of our model next to the actual value of the data.

```
df_test <- test_df
df_pred<-predict(df_model, df_test)
results <- as.data.frame(df_pred)
rownames(results) <- rownames(test_df)
head(results,30) %>% knitr::kable()
```

	df_pred
Neg_1.0_A100NXYDA6HSKA.txt	Neg
Neg_1.0_A106CGPJUBDG0P.txt	Pos
Neg_1.0_A10KECJUJOBO9H.txt	Neg
Neg_1.0_A10VGXGJD9JMOC.txt	Neg
Neg_1.0_A11423AYVMMC4K.txt	Neg
Neg_1.0_A11DMJCWJP9FWM.txt	Neg
Neg_1.0_A11IHV8N5A8IC7.txt	Neg
Neg_1.0_A1243Z7MQ4FRCW.txt	Neg
Neg_1.0_A12A41D1UD193O.txt	Neg
Neg_1.0_A12BTR2MVK2BR5.txt	Neg
Neg_1.0_A12LKEM543ILBK.txt	Neg
Neg_1.0_A12Y90C2QKO143.txt	Neg
Neg_1.0_A12YD3OXVP3AZQ.txt	Neg
Neg_1.0_A13DKAV1SES6QH.txt	Neg
Neg_1.0_A13ML60EDB3YFZ.txt	Neg
Neg_1.0_A13WK3ZST0KK06.txt	Neg
Neg_1.0_A13XK1XROK2Y1K.txt	Neg
Neg_1.0_A144NUECKP2AS7.txt	Neg
Neg_1.0_A14A2VLURI8DGP.txt	Neg
Neg_1.0_A14YNI58YQRDGT.txt	Neg
Neg_1.0_A157GF69NHAC2H.txt	Neg
Neg_1.0_A15DR3N4TW1X60.txt	Pos
Neg_1.0_A15GDVKMYRUPRC.txt	Pos
Neg_1.0_A15HP656WUGPA7.txt	Neg
Neg_1.0_A15HZS9RVI9ZXO.txt	Neg
Neg_1.0_A15RDGDH40H4UP.txt	Neg
Neg_1.0_A15TNUM2PBS6F0.txt	Neg
Neg_1.0_A15U8WD5827Y29.txt	Neg

	df_pred
Neg_1.0_A16EM8SXS5CQ9Q.txt	Neg
Neg_1.0_A16FD1ZQX5AW7Q.txt	Neg

Conclusion

We have built a quick-start binary classifier that can categorise the sentiment of Amazon book reviews with a fairly high degree of accuracy. Such a classifier might be useful to analyse a larger volume of customer feedback for sentiments around a product or a service. More sophisticated models of the same type are are widely used, especially as a part of social media analysis, to identify businesses' strengths and weaknesses and can effectively be utilised to monitor consumers' behaviour.