Text Prediction Model Report

Babatunde Awosanya

Sunday, October 12, 2014

Introduction

This report explains the modelling process of the predictive text model.

Data Retrieval and Data Cleaning

Data Retrieval

The data used for the predictive modelling was download at https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip. The Enlish locale was selected for use and more specifically the *en_US.blogs.txt* file. The file has about 899288 lines and due to system memory limitation, only the first 800000 lines could be accomodated for further processing.

Data Cleaning

The particular file used in this task has a lot of specialized characters and the *UTF-8* encoding was used in order to have to an extent, a more acceptable and less infilterated text data. Furthermore, to have a cleaner data, all characters which were not alpha-numeric were removed from the data. This got rid of punctuations and other unwanted characters but not numbers. A special consideration was given to numeric characters in order to preserve the sentence structures. All numerical characters were replaced with a dollar character i.e. **$** so that a section of the text file having a sentence like:

be there in 2 hours

will now be transformed into

be there in $ hours and so on.

```
wordCorpus <- readLines("en_US/textDataRaw/en_US.blogs.txt",encoding = "UTF-8",n=800000)
wordCorpus <- gsub("[^a-z0-9 ]", "", wordCorpus, ignore.case = TRUE)
wordCorpus <- gsub("[0-9]+", "$", wordCorpus, ignore.case = TRUE)
wordCorpus <- tolower(wordCorpus)

bw <- read.csv("badwords1.txt",stringsAsFactors=FALSE)[,1]
for(i in 1:length(bw)){
    wordCorpus <- gsub(bw[i],"*",wordCorpus)
}
```

The next process in the data cleaning was to change the text case to lower. Although there are many spell checking functions to do spell-checking and these use the interractive mode, due to the very large size of the data it would be timing consuming and counter-productive considering the reward it gives relative to the efforts put in. Moreso, since the data is from blog posts, an assumption about the data was made that care would have been taken to ensure very minimal typos and spelling errors.

Bad/profane words were filtered from the text. The list of bad words filtered from the text are in the file named *"badwords1.txt"*. As in the case of the numerics, the bad/profane words were not totally removed but substituted with the asterics character (*) so as to preserve the sentence structure.

The consequence of these substitutions i.e. the dollar and the asterics is that our eventual model will not predict them because they were filtered off.

Text Co-occurence

The principle adopted in this modelling task is that of text co-occurence or colocations. If a word co-occurs with another word, they will occur most of the times together. Co-occurence of texts is one major concept in the processing of natural language.

Partitioning to NGrams

Let L1 and L2 denote the first and second lemma of an ngram. 2-gram, 3-gram and 4-gram objects were used to generate frequency tables and perform statistical associations of words.

As an example of 2-gram, *"i can"* for instance, *"i"* will denote L1 and *"can"* will denote L2. For a 3-gram, say for example *"i can do"*, *"i can"* will denote L1 and *"do"* will denote "L2". For a 4-gram, say for example *"i can do anything"*, *"i can do"* will denote L1 and *"anything"* will denote L2 and so on.

The next thing was to get the frequencies for the 2,3 and 4 ngrams. This was achieved by tokenization of the data and taking the counts of unique tokens. The 2,3 and 4 tokens were created by *NGramTokenizer* function from the **RWeka** package. The counts of these unique tokens were gotten using the *TermDocumentMatrix* and *inspect* functions from the **tm** package and *rollup* function from the **slam** package. Before proceeding to use these functions, a corpus of the data was created before the table of the tokens.

Creating the Corpus

With the clean text data, we proceeded with creating the corpus. And to be double check, we clean off every whitespace in order to keep a uniform space between words.

```
library(tm)
library(RWeka)
library(slam)

wordCorpus <- Corpus(VectorSource(wordCorpus))
wordCorpus <- tm_map(wordCorpus, stripWhitespace)
```

The Term Document Table

With the corpus of the data created, we tokenize and then take the counts of the unique 2,3 and 4 tokens. Due to system memory limitation, objects created, were often saved, removed from the workspace and loaded again in order to manage memory. The codes to achieve these are provided below:

```
## 2-gram
tokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
wordCorpus2 <- rollup(TermDocumentMatrix(wordCorpus,control = list(tokenize = tokenizer)),
                      2,FUN=sum)
wordCorpus2 <- as.data.frame(inspect(wordCorpus2))
```

```
write.csv(wordCorpus2,file="wordCorpusTable2.txt",quote=FALSE)

## 3-gram
tokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 3, max = 3))
wordCorpus3 <- rollup(TermDocumentMatrix(wordCorpus,control = list(tokenize = tokenizer)),
                      2,FUN=sum)
wordCorpus3 <- as.data.frame(inspect(wordCorpus3))
write.csv(wordCorpus3,file="wordCorpusTable3.txt",quote=FALSE)

## 4-gram
tokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 4, max = 4))
wordCorpus4 <- rollup(TermDocumentMatrix(wordCorpus,control = list(tokenize = tokenizer)),
                      2,FUN=sum)
wordCorpus4 <- as.data.frame(inspect(wordCorpus4))
write.csv(wordCorpus4,file="wordCorpusTable4.txt",quote=FALSE)
```

After these, the next task was to filter the tables. There were so many of the items that had just a single occurence and those items might have been by chance. In the case of the *2-gram* table, only items that occured more than 5 times were retained. However in the case of the 3-gram and 4-gram, only items that occured more than once were retained. This helped to reduce significantly the presence of most unwanted items. The penalty to be paid here would be the inability of the model to predict some rather non-trivial examples. We had to carry out this trade-off between redundancy and speed of prediction.

```
table2 <- read.table("wordCorpusTable2.txt", sep=",",header=FALSE,skip=1)
table22 <- table2[table2$V2 > 5, ]
write.csv(table22, file="table22.txt", quote=FALSE, row.names=FALSE)

table3 <- read.table("wordCorpusTable3.txt", sep=",",header=FALSE,skip=1)
table33 <- table3[table3$V2 > 1, ]
write.csv(table33, file="table33.txt", quote=FALSE, row.names=FALSE)

table4 <- read.table("wordCorpusTable4.txt", sep=",",header=FALSE,skip=1)
table44 <- table4[table4$V5 > 1, ]
write.csv(table44, file="table44.txt", quote=FALSE, row.names=FALSE)
```

Next was to create a proper table of the objects and so the tokens were then split using space between words as splitting character. In the table created, the column which held the last character in the split were filtered off the asteric (*) and dollar ($) character so that the model would not predict bad/profane words of numbers.

```
table2ngrams <- read.table("table22.txt", sep=",",header=TRUE, stringsAsFactors=FALSE)
w1 <- sapply(strsplit(table2ngrams$V1," "),'[',1)
w2 <- sapply(strsplit(table2ngrams$V1," "),'[',2)
table2ngrams <- data.frame(w1,w2,freq=table2ngrams$V2)
table2ngrams <- table2ngrams[-c(grep("[.*\\$.]",as.character(table2ngrams$w2))),]


table3ngrams <- read.table("table33.txt", sep=",",header=TRUE, stringsAsFactors=FALSE)
w1 <- sapply(strsplit(table3ngrams$V1," "),'[',1)
w2 <- sapply(strsplit(table3ngrams$V1," "),'[',2)
w3 <- sapply(strsplit(table3ngrams$V1," "),'[',3)
table3ngrams <- data.frame(w1,w2,w3,freq=table3ngrams$V2)
table3ngrams <- table3ngrams[-c(grep("[.*\\$.]",as.character(table3ngrams$w3))),]
```

```
table4ngrams <- read.table("table44.txt", sep=",",header=TRUE, stringsAsFactors=FALSE)
w1 <- sapply(strsplit(table4ngrams$V1," "),'[',1)
w2 <- sapply(strsplit(table4ngrams$V1," "),'[',2)
w3 <- sapply(strsplit(table4ngrams$V1," "),'[',3)
w4 <- sapply(strsplit(table4ngrams$V1," "),'[',4)
table4ngrams <- data.frame(w1,w2,w3,w4,freq=table4ngrams$V2)
table4ngrams <- table4ngrams[-c(grep("[.*\\$.]",as.character(table4ngrams$w4))),]
```

The head (i.e. first 6 rows) of tables created are given below:

```
head(table2ngrams)
head(table3ngrams)
head(table4ngrams)
```

```
##    w1        w2 freq
## 77  $         a 2265
## 79  $        aa   16
## 80  $     aaron    2
## 81  $        ab    2
## 82  $ abandoned    5
## 83  $      abba    2
```

```
##    w1 w2      w3 freq
## 16  $  $       a  146
## 17  $  $   about    3
## 18  $  $   above    2
## 19  $  $ abraham    2
## 20  $  $     abu    2
## 21  $  $     abv    6
```

```
##   w1 w2 w3       w4 freq
## 3  $  $  $ academic    2
## 4  $  $  $     acts    2
## 5  $  $  $     also    2
## 6  $  $  $       am   45
## 8  $  $  $      and  125
## 9  $  $  $    april    2
```

Now for the 2-gram, *w1* will denote the L1 and *w2* will denote L2 as previously used above. Similarly for the 3-gram, *w1, w2* will denote L1 and *w3* will denote L2 and for the 4-gram, *w1, w2, w3* will denote L1 and *w4* will denote L2. Note that the dollar ($) and asterics (*) will still be in the L1's but not in the L2's since they have been filtered off.

The Prediction Task

With tables above, the prediction can now be done. The process for getting the prediction are itemized below:

For any string input,

1. Remove non alpha-numeric characters

2. Convert the string input to lowercase
3. Change numbers to dollar sign ($) and bad/profane words to asterics (*)
4. Split the string
5. Extract the last 3 elements

- Supply the last element to column *w1* of 2-gram table i.e. L1
- Supply the last 2 elements to columns *w1, w2* of the 3-gram table respectively i.e. L1
- Supply all three elements to columns *w1, w2, w3* of the 4-gram table respectively i.e. L1

6. Retrieve the elements from L2 and corresponding frequencies that match input for L1 in the three tables (these will in turn be 2 column tables i.e. L2 and frequencies)

- sort the extracted tables by their frequencies columns in decreasing order, i.e. most occuring elements coming up before less occuring ones
- For the tables only top 6 rows are extracted

7. Merge the tables by L2

- The merged table will have 4 columns with column 1 representing L2, columns 2-4 representing frequencies from the three tables

8. Return the L2 column which are the predictions

A function called ***suggest*** was written perform all these task and it's given below. This algorithm does not keep memory of more than the last immediate 3 words. Where less than 3 words are supplied to the function, say for example 1 word, it uses the 2-gram table; where two words are supplied, it uses the 2-gram and 3-gram table and where 3 or more words are supplied, it uses all the three tables.

Further Tasks

Recall that we removed all non alpha-numeric characters which included apostrophes and that would affect the display of words like *don't* which had been transformed to **dont**, *can't* which had been transformed to **cant** and so on. These are referred to as **stopwords** in the **tm** package. They are:

```
toreplace <- stopwords()[c(58:61,64:68,70:74,76,80:95,97:98,100:108)]
toreplace
```

```
##  [1] "i'm"       "you're"    "he's"      "she's"     "they're"
##  [6] "i've"      "you've"    "we've"     "they've"   "you'd"
## [11] "he'd"      "she'd"     "we'd"      "they'd"    "you'll"
## [16] "they'll"   "isn't"     "aren't"    "wasn't"    "weren't"
## [21] "hasn't"    "haven't"   "hadn't"    "doesn't"   "don't"
## [26] "didn't"    "won't"     "wouldn't"  "shan't"    "shouldn't"
## [31] "can't"     "couldn't"  "mustn't"   "that's"    "who's"
## [36] "what's"    "here's"    "there's"   "when's"    "where's"
## [41] "why's"     "how's"
```

What was done was just to ensure that the words below :

```
tofind <- gsub("'","",toreplace)
tofind
```

```
##  [1] "im"       "youre"    "hes"      "shes"     "theyre"   "ive"
##  [7] "youve"    "weve"     "theyve"   "youd"     "hed"      "shed"
## [13] "wed"      "theyd"    "youll"    "theyll"   "isnt"     "arent"
## [19] "wasnt"    "werent"   "hasnt"    "havent"   "hadnt"    "doesnt"
## [25] "dont"     "didnt"    "wont"     "wouldnt"  "shant"    "shouldnt"
## [31] "cant"     "couldnt"  "mustnt"   "thats"    "whos"     "whats"
## [37] "heres"    "theres"   "whens"    "wheres"   "whys"     "hows"
```

were replaced with their correspondents previous given above in order to ensure good output quality. This was done using the function called ***decode*** below:

```
decode <- function(x, search, replace, default = NULL) {

    # build a nested ifelse function by recursion
    decode.fun <- function(search, replace, default = NULL)
        if (length(search) == 0L) {
            function(x) if (is.null(default)) x else rep(default, length(x))
        } else {
            function(x) ifelse(x == search[1L], replace[1L],
                               decode.fun(tail(search,  -1L),
                                          tail(replace, -1L),
                                          default)(x))
        }

    return(decode.fun(search, replace, default)(x))
}
```

And the prediction function:

```
suggest <- function(x){
    x <- tolower(x)
    x <- gsub("[^a-z0-9 ]", "", x, ignore.case = TRUE)
    x <- gsub("[0-9]+", "$", x, ignore.case = TRUE)
    xx <- unlist(strsplit(x," "))
    x1.tab <- table2ngrams[table2ngrams$w1==xx[length(xx)],c("w2","freq")]
    x2.tab <- table3ngrams[table3ngrams$w1==xx[length(xx)-1]&table3ngrams$w2
                           ==xx[length(xx)],c("w3","freq")]
    x3.tab <- table4ngrams[table4ngrams$w1==xx[length(xx)-2]&table4ngrams$w2
                           ==xx[length(xx)-1]&table4ngrams$w3==xx[length(xx)],
                           c("w4","freq")]
    x1.tab <- x1.tab[with(x1.tab, order(freq,decreasing = TRUE)),]
    x2.tab <- x2.tab[with(x2.tab, order(freq,decreasing = TRUE)),]
    x3.tab <- x3.tab[with(x3.tab, order(freq,decreasing = TRUE)),]
    x1.tab <- head(x1.tab);x2.tab <- head(x2.tab);x3.tab <- head(x3.tab)
    d <- merge(x1.tab,x2.tab,by.x = "w2",by.y = "w3",all=TRUE)
    e <- merge(d,x3.tab,by.x = "w2",by.y = "w4",all=TRUE)
    decode(as.character(e[,1]),tofind,toreplace)
}
```

Model Testing

The model was tested on some examples to see how well it performs. A table of input and possible follow ups are given in the table below:

Input text	True Word	Predictions
sure wish we	could	are, can, could, did, had, have, lived, were, will, would
i picked him up this	afternoon	book, is, morning, one, time, was, week, weekend, year
he wasn't all that interested in rock	music	and, band, hill, history, music, n, star, the
families are now encouraged to	celebrate	a, apply, attend, be, bring, do, get, make, participate, take, the
i am praying that both start to feel	better	a, better, free, it, like, sad, so, that, the

Conclusion

This model was used for the quizzes, and its response time is ok in terms of returning predictions. The algorithm learned from the data used to train it and returns to quite some extent good results. For very rare word combinations, it might not predict anything because I filtered off items that occured just once. If I had left them, they would have affected response time and there is also restriction on the amount of system memory that can be allocated. In the submitted application application, I let the algorithm recognize stopping characters such as full stop (.), question mark (?) and exclamation mark (!).

Reference

1. Statistical Analysis of Corpus Data with R *Marco Baroni and Stefan Evert*
2. http://ufal.mff.cuni.cz/mlnlpr13 "Machine Learning in Natural Language Processing using R"