Beginner's Introduction to Text Mining: An App Store Reviews Exercise

Samuel Chan

18 April 2017

Foreword

This notebook is written as a workshop material for a session by Plug and Play Indonesia. The workshop is part of a series that serve as an introduction to data science and machine learning, and its intended audience are novices as well as junior professionals in the field of data science. Plug and Play is an accelerator for mobile startups, and as such, I have chosen to show application of text mining techniques and capabilities in processing app store reviews.

With the context set, this is perhaps a good time to explain the motivation of this programming exercise. I work for the company HyperGrowth, and among other things, we develop automation tools for mobile app businesses. One of our product is GrowthBot, a free automation chatbot that deliver growth metrics, app reviews, and performance scorecards to mobile marketers. Our customers use GrowthBot to better analyze and manage (*reply to*) user reviews of their apps, and the advantages are plenty: accurate market feedback, faster product iteration, and an unparalled timeliness in responding to users' needs and feedback. This notebook aims to unpack some of these concepts and show how we can utilize R and some simple text mining packages to:

- Process text retrieve from a web service endpoint
  - Converting JSON to R
- Data cleansing and content transformation techniques
  - Compare pre- and post-processed data
- Find the top 10 most frequent keywords
  - Easily find keywords that appear >100 times
- Generate a wordcloud
- Extracts sentiment
  - Visualize sentiment
  - Compare sentiment(s) between 2 apps using ggplot
- Checking corrrelations between words
- Custom stopwords, custom content transformer and the role of domain knowledge

Preparing the Data

Some initial configuration. As a safety precaution we also clear our global environment.

Load the necessary library and retrieving our data

```
library(jsonlite)
library(tm)
```

```
## Loading required package: NLP
```

```
library(SnowballC)
library(syuzhet)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:NLP':
##
##     annotate
```

```
url <- "http://bot.hypergrowth.co:3005/reviews"
raw <- readLines(url)
rd <- fromJSON(raw)[2]
rd <- rd$data
```

It's important we recognize here that R itself is developed as a statistical programming language and we're adding text mining capabilities to this language through the use of packages, developed independently by developers within the community. Every time we launch this project, we have to install those packages into our 'environment':

1. `jsonlite` - Convert JSON data into R objects
2. `tm` - A text mining package. Comprehensive documentation here
3. `SnowballC` - Word stemming
4. `syuzhet` - Extracting sentiments
5. `wordcloud` - Generate wordcloud (surprised?)
6. `ggplot2` - Data visualization based on the grammar of graphics. A ggplot2 cheatsheet, also by me

Pre-processing and Data Cleaning

Most of the time the data we get from an API isn't conveniently structured in the format that is suited for our analysis. In this example, the data we get includes a non-insignificant amount of duplication. As we will later see, we need to remove the duplicated reviews and also apply a range of transformative functions to clean up our data.

To see a list of all transformations that come with the `tm` package:

```
getTransformations()
```

```
# Remove duplicated reviews
df <- rd[!duplicated(rd$comment), ]

# Convert df$comment to a Corpus vector
```

```
docs <- Corpus(VectorSource(df$comment))

#lapply return a list. We want just the $content of each Corpus object and not the meta data
cont <- lapply(docs, function(x){x$content})

# Clean up and remove "/", "@" etc with empty spaces from our Corpus
transformer <- content_transformer(function(x, pattern)
    gsub(pattern, " ", x)
  )

docs <- tm_map(docs, transformer, "/")
docs <- tm_map(docs, transformer, "@")
docs <- tm_map(docs, transformer, "\\|")
docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeNumbers)
docs <- tm_map(docs, removeWords, stopwords("english"))
docs <- tm_map(docs, removePunctuation)
docs <- tm_map(docs, stripWhitespace)
```

Stemming vs Lemmatization

Stemming refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time. For example, *packages*, *packaging*, and *packaged* will result in *packag*. This process is crude but the overall benefit gained from stemming usually more than makes up for the downside of its special cases. Let's see stemming in action:

```
print(writeLines(as.character(docs[[10]])) + writeLines(as.character(docs[[25]])) + writeLines(as.chara
```

```
## pesanan saya mana
##  soooooo fun loved send update beautey da beast lilo stich ohana means family guest
## best shopping online deal
## numeric(0)
```

After stemming:

```
docs.s <- tm_map(docs, stemDocument, mc.cores=1)
print(writeLines(as.character(docs.s[[10]])) + writeLines(as.character(docs.s[[25]])) + writeLines(as.cl
```

```
## pesanan saya mana
##  soooooo fun love send updat beautey da beast lilo stich ohana mean famili guest
## best shop onlin deal
## numeric(0)
```

Lemmatization on the other hand, aims to doing things properly with the use of a vocabulary and grammatical context, thus aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*. A popular R package for dealing with that is `koRpus`, which makes the `TreeTagger`(Helmut Schmid, Institute for Computational Linguistics of the University of Stuttgart) functionality available to us. Among other things, determining the *lemma* of a word requires knowledge of its part of speech (POS), and POS taggers for both Bahasa Indonesia and English are available as open source projects – although that is an entirely different subject which we have to cover next time.

Exploratory text mining

Document Term Matrix

When we create a document term matrix (dtm), we're actually creating a matrix that list all occurrences of words, with each document (app review) taking a row. Naturally then, the terms that appear in the body of each app review is represented by columns. If a word occurs in a particular document once, the matrix entry corresponding to that row and column will be 1. If it appears twice, it will be recorded as 2.

```
# Create the document term matrix
dtm <- TermDocumentMatrix(docs)
mat <- as.matrix(dtm)

# sort by frequency and print the first 10
v <- sort(rowSums(mat), decreasing=TRUE)
d <- data.frame(word = names(v), freq=v)
head(d, 10)
```

```
##          word freq
## app       app  379
## dan       dan  294
## game     game  278
## saya     saya  260
## bisa     bisa  236
## good     good  219
## tidak   tidak  209
## lazada lazada  196
## ada       ada  188
## nya       nya  149
```

Notice how a DTM essentially converts a corpus of text into a mathematical object that can be analyzed using quantitative techniques of matrix algebra, allowing us to apply mathematical functions such as `colSums` and `length` etc.

Wordcloud

If up to this point, all of this seems dry and boring, we can spice things up using a wordcloud. The function itself is very straightforward and we can specify a few parameters to control the generation of our wordcloud. Probably the only one that needs a little more clarification is our `rot.per` parameter, which specifies the percentage of words we want to be plotted vertically.

```
# Generate a word cloud
set.seed(417)
wordcloud(words = d$word, freq = d$freq, min.freq = 3, max.words = 250, random.order = FALSE, rot.per =
```

It's also a good time to point out that we could, if we want something less fancy (or more structured), as easily get a list of keywords that occur say at least *100* times in the entire corpus. This is done by passing in a `dtm` to the `findFreqTerms()` function:

```
findFreqTerms(dtm, lowfreq=100)
```

```
##  [1] "ada"      "aplikasi" "app"      "bisa"     "dan"      "gak"
##  [7] "game"     "good"     "great"    "ini"      "lagi"     "lazada"
## [13] "like"     "nya"      "order"    "saya"     "sudah"    "tidak"
## [19] "update"   "yang"
```

Sentiment Analysis

When it comes to text sentiment, one of the most helpful package is the `syuzhet` package. We call `get_nrc_sentiment` on the vector that contains all our app review body, and add a new column with our sentiment score onto it. We'll call the resulting dataframe `text`.

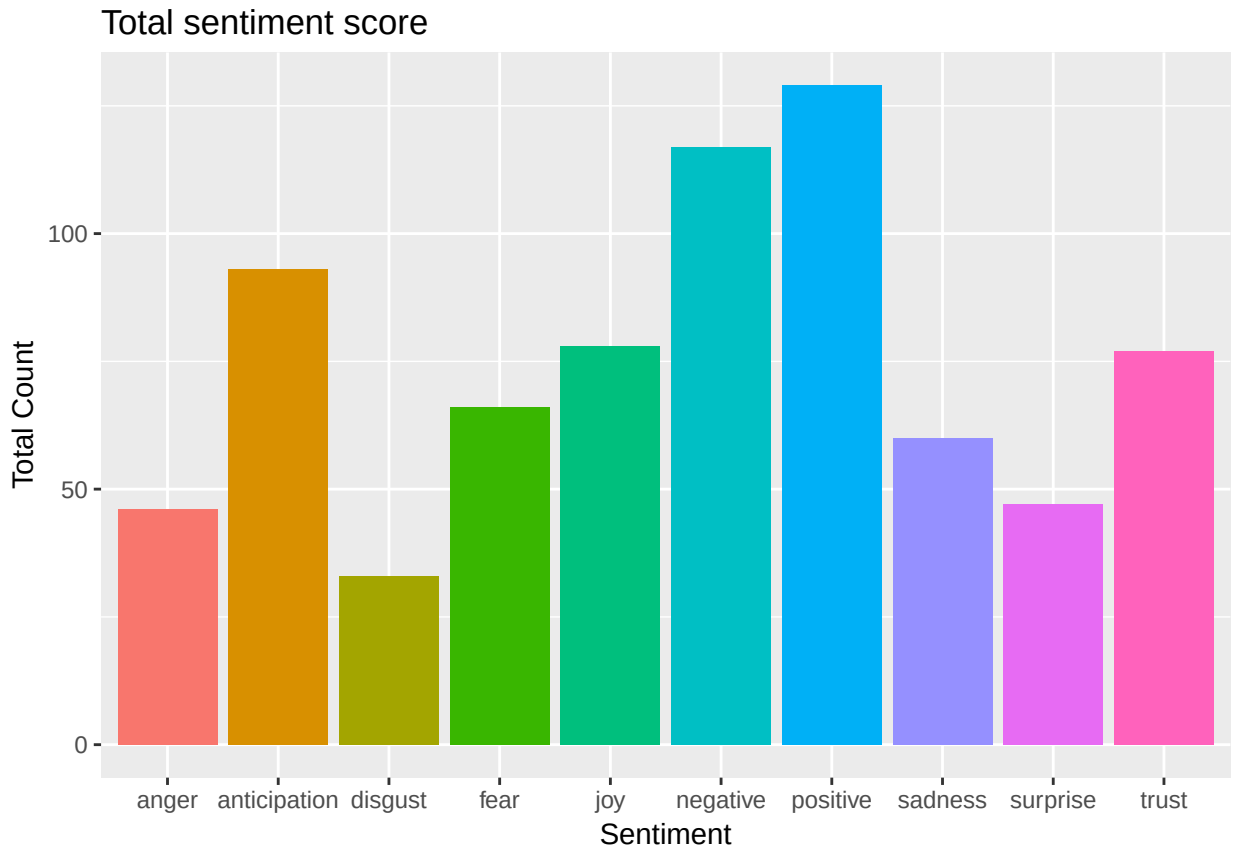
Once that is done, we sum by column each of these sentiments (col 2 to col11). We then use `ggplot` to visualize our overall sentiment score.

```
# fetch sentiment words from war pirates
wp.c <- df[df$app_id == "net.gogame.games.warpirates", 3]
sentiment <- get_nrc_sentiment(wp.c)
text <- cbind(wp.c, sentiment)

# total sentiment score by category
total.sentiment <- data.frame(colSums(text[, c(2:11)]))
```

```
names(total.sentiment) <- "count"
total.sentiment <- cbind("sentiment" = rownames(total.sentiment), total.sentiment)
rownames(total.sentiment) <- NULL

# total sentiment score of all text
ggplot(data=total.sentiment, aes(x=sentiment, y=count))+geom_bar(aes(fill=sentiment), stat="identity")+
```



If you are intrigue or interested in the inner workings of the sentiment-sorting library, I'd encourage you to dive deeper with the `syuzhet` library

```
# Get sentiment of the word hate
get_nrc_sentiment("hate")
```

```
##   anger anticipation disgust fear joy sadness surprise trust negative
## 1     1            0       1    1   0       1        0     0        1
##   positive
## 1        0
```

```
# or the word cool
get_nrc_sentiment("cool")
```

```
##   anger anticipation disgust fear joy sadness surprise trust negative
## 1     0            0       0    0   0       0        0     0        0
##   positive
## 1        1
```

We could have done the above, but instead perform the analysis on two specific subsets of our data. We'll compare the sentiments of app reviews sampled from War Pirates and Disney: Crossy Road, two mobile

games from our friends at goGame.

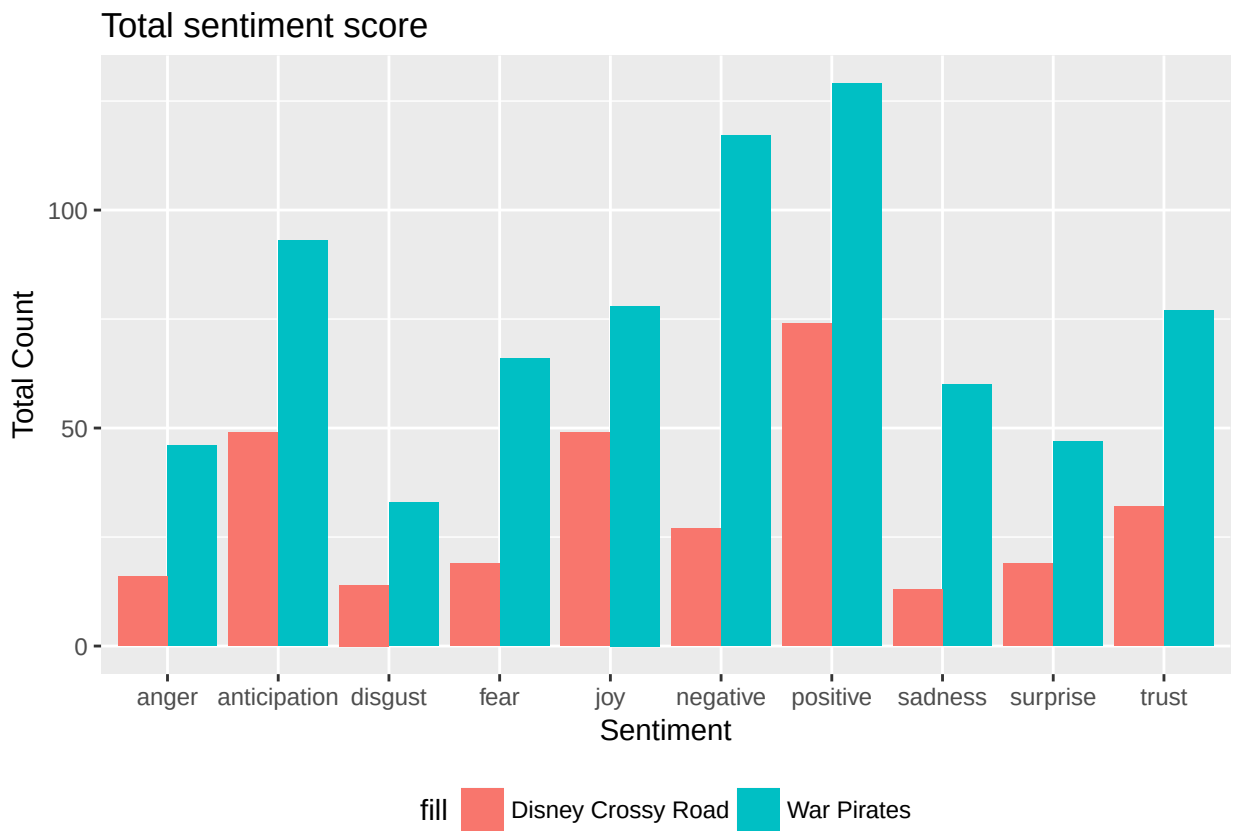
```
# fetch sentiment words from war pirates
wp.c <- df[df$app_id == "net.gogame.games.warpirates", 3]
dcr.c <- df[df$app_id == "net.gogame.disney.crossyroad", 3]

sentiment.wp <- get_nrc_sentiment(wp.c)
text.wp <- cbind(wp.c, sentiment.wp)
sentiment.dcr <- get_nrc_sentiment(dcr.c)
text.dcr <- cbind(dcr.c, sentiment.dcr)

# total sentiment score by category for War Pirates
total.sentiment.wp <- data.frame(colSums(text.wp[, c(2:11)]))
names(total.sentiment.wp) <- "count"
total.sentiment.wp <- cbind("sentiment" = rownames(total.sentiment.wp), total.sentiment.wp)
rownames(total.sentiment.wp) <- NULL

# total sentiment score by category for Disney Crossy Road
total.sentiment.dcr <- data.frame(colSums(text.dcr[, c(2:11)]))
names(total.sentiment.dcr) <- "count"
total.sentiment.dcr <- cbind("sentiment" = rownames(total.sentiment.dcr), total.sentiment.dcr)
rownames(total.sentiment.dcr) <- NULL

sentiment.df <- rbind(data.frame(fill="Disney Crossy Road", obs=total.sentiment.dcr), data.frame(fill="
# total sentiment score of all text between DCR and WP
ggplot(data=sentiment.df, aes(x=obs.sentiment, y=obs.count))+geom_bar(aes(fill=fill), stat="identity",
```



Exercise: Top 10 keywords and wordcloud comparison

Without referring to the code below, you should hopefully be able to complete a simple exercise: Create two top 10 keyword list and their respective wordcloud for each of the two apps (or any two apps of your choice, really)

```
# Create the document term matrix

docs.wp <- Corpus(VectorSource(df[df$app_id == "net.gogame.games.warpirates",3]))
docs.wp <- tm_map(docs.wp, removeWords, stopwords("english"))

docs.dcr <- Corpus(VectorSource(df[df$app_id == "net.gogame.disney.crossyroad",3]))
docs.dcr <- tm_map(docs.dcr, removeWords, stopwords("english"))

dtm.wp <- TermDocumentMatrix(docs.wp)
dtm.dcr <- TermDocumentMatrix(docs.dcr)
mat.wp <- as.matrix(dtm.wp)
mat.dcr <- as.matrix(dtm.dcr)

# convert to dataframe
sum.wp <- as.data.frame(sort(rowSums(mat.wp), decreasing = TRUE))
sum.dcr <- as.data.frame(sort(rowSums(mat.dcr), decreasing=TRUE))

# Top 20 keywords for war pirates and disney crossy road
# head(sum.wp, 20)
# head(sum.dcr,20)

freq.df <- as.data.frame(cbind(head(sum.wp,20), head(sum.dcr,20)))
freq.df
```

```
##          sort(rowSums(mat.wp), decreasing = TRUE)
## game                                           99
## loading                                        27
## like                                           22
## get                                            16
## will                                           16
## crash                                          15
## even                                           15
## fix                                            15
## game.                                          15
## keep                                           15
## love                                           15
## good                                           14
## the                                            14
## crashing                                       13
## just                                           13
## play                                           13
## really                                         13
## great                                          12
## time                                           12
## game,                                          11
##          sort(rowSums(mat.dcr), decreasing = TRUE)
## game                                            51
## loading                                         18
## like                                            16
```

```
## get                                                  16
## will                                                 10
## crash                                                 9
## even                                                  9
## fix                                                   8
## game.                                                 8
## keep                                                  8
## love                                                  7
## good                                                  7
## the                                                   6
## crashing                                              5
## just                                                  5
## play                                                  5
## really                                                5
## great                                                 5
## time                                                  5
## game,                                                 5
```

Bonus: Correlations

Another technique: We can inspect `correlations` between any interesting words. For example, if we see that the word *error* occurs in many of War Pirates review and are curious to learn what other common words co-occur with that:

```
# Assuming a search term co-occurence of 0.6
findAssocs(dtm.wp, "error",0.6)
```

```
## $error
## launching      mins   network    unable    worked     first   freezes
##       1.0       1.0       1.0       1.0       1.0       0.7       0.7
##   started
##       0.7
```

```
findAssocs(dtm.wp, "loading",0.6)
```

```
## $loading
## screen.     bar
##    0.66    0.65
```

Using association rules, we can create a very rudimentary but completely functional set of auto-completion features for customer support forms, search forms etc.

Notice that as we create our own wordcloud, we omit some other stop words (key phrases that are "noise" i.e not contributing to us understanding data better). using your domain expertise, what are some other stop words we could have selected?

```
set.seed(417)
# wordcloud(words = d$word, freq = d$freq, min.freq = 5, max.words = 250, random.order = FALSE, rot.per

wp.v <- sort(rowSums(mat.wp), decreasing=TRUE)
wp.df <- data.frame(word=names(wp.v), freq=wp.v)

dcr.v <- sort(rowSums(mat.dcr), decreasing=TRUE)
dcr.df <- data.frame(word=names(dcr.v), freq=dcr.v)

# Also: remove some other common stopwords because it's not helpful
```

```
wp.df <- wp.df[wp.df$word != "game" & wp.df$word != "will" & wp.df$word != "get", ]
dcr.df <- dcr.df[dcr.df$word != "game",]

wordcloud(words = wp.df$word, freq = wp.df$freq, min.freq = 3, max.words = 250, random.order = FALSE, r
```


```
wordcloud(words = dcr.df$word, freq = dcr.df$freq, min.freq = 2, max.words = 250, random.order = FALSE,
```

Where to go from here

Congratulations on making it this far! We've seen first hand how we can use various text mining techniques to make sense of a varied pool of app reviews data, deriving useful information from a wealth of unprocessed data. Depending on your interest and use-case, a good next step could be a more technical read of stemming and lemmatization, or expand on what we've learn in associated keywords to build a rudimentary predictive auto-completion feature. We could also study our top keywords in more depth and find patterns that could be useful in helping us determine some broad categories to classify each review into, e.g. Technical Bugs, Praise, Levels Difficulty, Payment Issues etc. The specific discpline that deals with that is known more formally as 'cluster analysis', and could go a long way to help us set automation rules when it comes to dealing with customer feedback and app reviews.

Feedback

- Reach me on Facebook
- Or my email: samuel [at] hypergrowth.co
- Learn more about GrowthBot, the chatbot that creates performance scorecards using app reviews, perform sentiment analysis, and automate your app marketing workflow: GrowthBot's Main Website