Unstructured Data Lab: Twitter Sentiment Analysis with MySQL and OpenStack

Dr. Simon Caton

November 6, 2018

Libraries we're going to use:

```r
library(tidytext)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(reshape)
```

```
##
## Attaching package: 'reshape'
```

```
## The following object is masked from 'package:dplyr':
##
##     rename
```

```r
library(ggplot2)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following objects are masked from 'package:reshape':
##
##     expand, smiths
```

```r
library(RMySQL)
```

```
## Loading required package: DBI
```

Analysing Tweets via tidytext

We'll start with the assumption that you have already collected a series of tweets using the Getting 100k+ Tweets lab. For the purposes of this example, I'll be using tweets related to bitcoin, but you could use anything really. In fact, the text analysis we are going to look at, will work for any text content: You just need to structure your data as a data.frame, where each observation corresponds to one document/tweet/post/review etc.

This example is based on http://tidytextmining.com/sentiment.html. Please refer to this book, for more information as well as a methods of analyzing text data.

One of the fundamental problems of sentiment analysis is that many libraries and approaches will give provide a subtly different result, and sometimes a very different result. Therefore it is usually good practice (not, sadly, that you will see many researchers do this) is to use multiple means to analyse the text data for sentiment.

Getting the Tweets from the DB

Just insert your database details into the snippet below, and it will pull all of your tweets from the database. Keep in mind that if you have a really large number of tweets that this can take a while!

```
con <- dbConnect(MySQL(),
                 user="username", password="password",
                 dbname="databaseName", host="87.44.4.something")
on.exit(dbDisconnect(con))

rs <- dbSendQuery(con, "select tweet_text, created_at from tweets;")

data <- fetch(rs, n=Inf)
```

You may see a warning message about NAs but this is nothing to worry about. You can explore the content / structure of the database we made if you want to extend beyond what we are doing here. For the purposes of this lab, I'm only interested in the tweet text, and the timestamp corresponding to when it was tweeted – but the database has an awful lot more to offer than this!

To be specific, the function *dbSendQuery* execute the SQL query, and returns a result set object. When then request the results from MySQL using the *fetch* function, the parameter n specifies the number of results to return, setting it to either -1 or Inf (infinite) means return all results.

My Twitter database has only been up and running for a few hours (I have 11865 Tweets so far), so I don't have too many Tweets to play with at the moment:

```
summary(data)
```

```
##   tweet_text          created_at
##  Length:11865       Length:11865
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

So, in the twitter lab, we didn't specify to twitter that we wanted only English tweets (just extend the query a little if you want this), however, we can also filter for English only tweets now as follows:

```
library(textcat)
library(cld2)
library(cld3)
library(tidyverse)

data <- data %>% mutate(textcat = textcat(x = tweet_text),
  cld2 = cld2::detect_language(text = tweet_text, plain_text = FALSE),
  cld3 = cld3::detect_language(text = tweet_text)) %>%
  select(tweet_text, textcat, cld2, cld3, created_at) %>%
  filter(cld2 == "en" & cld3 == "en")

summary(data)
```

```
##   tweet_text           textcat               cld2
##  Length:9218        Length:9218        Length:9218
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##      cld3            created_at
##  Length:9218        Length:9218
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

We actually ran three language detection mechanisms here, however, textcat tends to be a little unreliable cld2 and cld3 are based on Google's language detection technologies so usually fairly reliable. Mutate is a dpylr function that adds new columns. The same code could be executed as follows if the above is too convoluted:

```
data$textcat <- textcat(x = tweet_text)
data$cld2 <- cld2::detect_language(text = tweet_text, plain_text = FALSE)
data$cld3 <- cld3::detect_language(text = tweet_text)

data <- data[data$cld2 == "en" & data$cld3 == "en", ]
```

Preparing the text data

For the purposes of this exercise, I'm only interested in tweets that are not retweets, and the date they were tweeted. Obviously, as we are going to look at the affect of tweets, it makes no sense for them to be in different languages (hence filtering by language above). Similarly, removing retweets makes it less likely to have duplicated content. In Twitter vernacular, a retweet is specified by RT at the start of the Tweet, i.e. the text starts with the characters "RT".

```
data$RT <- startsWith(data$tweet_text, "RT")
```

Defines a logical column concerning whether the Tweet is a retweet or not, now if we remove retweets:

```
data <- data[!data$RT, ]
```

Note that after removing non-english Tweets and retweets I've lost about half of my Tweets – such are the challenges of Social Media research; you'll get used to it. Note that for the purposes of the DW project (the 100k minimum requirement is preprocessed not post-processed).

For reference, this leaves me with 5225 tweets; with time, my OpenStack VM will capture more and more Tweets, in fact if your query keyword(s) are well determined (make sure you specify them in the report) you can expect about 10-20k Tweets per day.

Now to convert the data.frame into the structure needed. In this example, I'm interested in analyzing the sentiment by date, such that I would be able to link it as a fact to a date dimension. For that I need to work on the date column a little bit.

Although I am using date here, you could use any aspect of your text as a means to essentially group content – I'm grouping by date, but you can just as well use products, locations, users, etc.

We need to correctly represent our date column now:

```
f <- "%Y-%m-%d %H:%M:%S"
data$created_at <- as.Date(data$created_at, format=f)
```

If the granularity of date was insufficient, I could also leverage the *as.timeDate* and manipulate the date string to suit my required granularity; e.g. remove the seconds if my finest granularity is minutes, or remove both seconds and minutes if I'm interested in hours. See the feature engineering of Name/Title etc. in R Lab 3 for examples of doing this. ALternatively, use the lubridate library, which is really nice for working with dates.

As my database has only been collected Tweets for a few hours I only have Tweets from today… for the purposes of demonstration, I'm going to cheat here and manipulate my data (you are not allowed to do this in the project!): I'm going to take the first half of the data set, and change the date stamp to yesterday.

```
data$created_at[1:round(nrow(data)/2)] <-
  data$created_at[1:round(nrow(data)/2)] - 1
```

One of the beautiful things about R is that almost any data structure can use operated on with arthimetic operators: a date - 1 removes one day from the the date – simple, elegant. Just to check, let's see how many dates I have:

```
unique(data$created_at)
```

```
## [1] "2018-11-05" "2018-11-06"
```

I have 2 different dates, obviously for more meaningful analysis I would need to collect tweets over a longer period of time. For use later, let's record the number of tweets we have for each date.

```
noTweets <- table(data$created_at)
```

Now to segment the tweets by date:

```
text_df <- data_frame(date = data$created_at, text = data$tweet_text)
```

and tokenize them (note that this tokenization process removes characters like \xab\xa0 – these are emoticons – and we don't want those)

```
text_df <- text_df %>% unnest_tokens(word, text)
```

Load the dictionaries

```
afin <- get_sentiments("afinn")
bing <- get_sentiments("bing")
nrc <- get_sentiments("nrc")
```

And analyse

```
afinSent <- text_df %>%
  inner_join(afin) %>%
  group_by(index = date) %>%
  summarise(sentiment = sum(score))
```

```
## Joining, by = "word"
```

```
bingSent <- text_df %>%
  inner_join(bing) %>%
  count(index = date, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

```
## Joining, by = "word"
```

```
nrcSent <- text_df %>%
  inner_join(nrc) %>%
  count(index = date, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

```
## Joining, by = "word"
```

Ok so because we may have more tweets for one date, than the other, we should probably control for this. (We don't in this case – I set half of them to yesterday, but in a real situation we would need to control for having more content on some days than others)

```
for(i in 1:length(noTweets)) {
  afinSent$sentiment[i] <- afinSent$sentiment[i] / noTweets[[i]] * 100
  bingSent$sentiment[i] <- bingSent$sentiment[i] / noTweets[[i]] * 100
  nrcSent[i, 2:12] <- nrcSent[i, 2:12] / noTweets[[i]] * 100
}
```

The multiple by 100 here is pretty meaningless, and could be left out. For the purposes of this example, we'll consider only sentiment, however, in other examples there may be cause to use some of the other measures of nrc:

```
str(nrcSent)
```

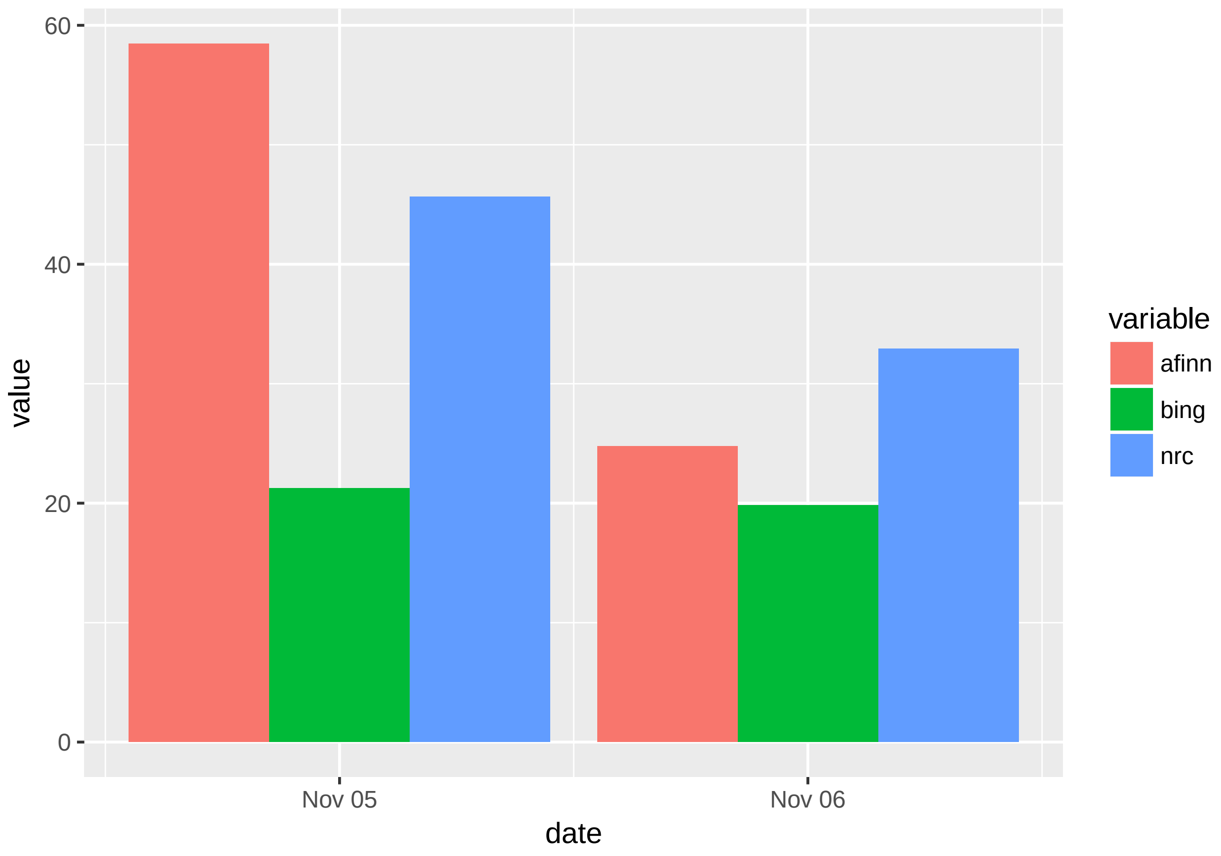
```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2 obs. of  12 variables:
##  $ index       : Date, format: "2018-11-05" "2018-11-06"
##  $ anger       : num  23.8 30.6
##  $ anticipation: num  49.5 42
##  $ disgust     : num  7.08 10.91
##  $ fear        : num  24.3 28.2
##  $ joy         : num  29.1 22.5
##  $ negative    : num  36.7 39.4
##  $ positive    : num  82.3 72.3
##  $ sadness     : num  17.6 20
##  $ surprise    : num  16.1 15.2
##  $ trust       : num  51.2 51.6
##  $ sentiment   : num  45.6 32.9
```

```
sentimentTweets <- data.frame(afinSent)
sentimentTweets <- merge(sentimentTweets, bingSent[,c("index", "sentiment")], by="index")
sentimentTweets <- merge(sentimentTweets, nrcSent[, c("index", "sentiment")], by="index")

names(sentimentTweets) <- c("date", "afinn", "bing", "nrc")
```

Ok so, we have 3 measures of sentiment. I'd rather not put the labels "positive" or "negative" on either of the days. Rather note that one day seems to be more positive than another (but remember that we faked the days, so we could also say that earlier is more/less positive than later) – purist notions of sentiment as "high" or "low" are methodologically flawed: they are relative quantities not absolute values:

```
tweetData <- melt(sentimentTweets, id="date")
ggplot(tweetData, aes(x=date, y=value, fill=variable)) +
  geom_bar(stat='identity', position='dodge')
```



For bing this difference is slightly negligible. At this point I could decide which of the sentiment measures I want, or create some sort of composite measure of all three. However, note that because afinn has a higher range of values; it scores words between -5 and 5 vs. bing and nrc noting only positive and negative affect; afinn would somewhat crowd out the other two if care was not taken into how such a composite measure was defined. We are now however, going somewhat beyond the scope of this module, but we would want to standardise the sentiment quantities (this will have been covered in stats or if you weren't paying attention, will be covered in data mining again).

We could at this point write the results back to the MySQL DB, drop a csv of the results to the staging arena, link R directly to SQL Server see: https://db.rstudio.com/databases/microsoft-sql-server/, or whatever else you prefer to do.

Twitter's REST API

If you didn't want to use OpenStack, you could actually just use the REST API and ping it at regular intervals with a cron job or scheduled task easily enough. You would do that as follows:

```r
library(twitteR)
api_key <- "<your api key>"
api_secret <- "<your api secret>"
access_token <- "<your acess token>"
access_token_secret <- "<your access token secret>"

setup_twitter_oauth(api_key,api_secret,access_token,access_token_secret)

searchTerm <- "whatever you want to search Twitter for"
```

```r
# set n to the number of Tweets you want
n <-

#note there are many other arguments for the twitteR library
#use ?searchTwitter to view the help

tweets <- searchTwitter(searchTerm, n)

tweets <- do.call("rbind", lapply(tweets, as.data.frame))

# add in any preprocessing / cleaning you want here
# e.g.

tweets$text <- sapply(df$text,function(row) iconv(row, "latin1", "ASCII", sub=""))

# add in any filtering you want here
# e.g.

tweets <- tweets[!tweets$isRetweet, ]
tweets <- tweets[tweets$language == "en", ]
tweets <- tweets[, c("text", "created")]

# add in any transformation you want here
# e.g. the date manipulation
# strictly speaking sentiment analysis also counts as transformation

noTweets <- table(df$created)
text_df <- data_frame(date = df$created, text = df$text)
text_df <- text_df %>% unnest_tokens(word, text)

afin <- get_sentiments("afinn")
bing <- get_sentiments("bing")
nrc <- get_sentiments("nrc")

afinSent <- text_df %>%
  inner_join(afin) %>%
  group_by(index = date) %>%
  summarise(sentiment = sum(score))

bingSent <- text_df %>%
  inner_join(bing) %>%
  count(index = date, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

nrcSent <- text_df %>%
  inner_join(nrc) %>%
  count(index = date, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

for(i in 1:length(noTweets)) {
  afinSent$sentiment[i] <- afinSent$sentiment[i] / noTweets[[i]] * 100
  bingSent$sentiment[i] <- bingSent$sentiment[i] / noTweets[[i]] * 100
  nrcSent[i, 2:12] <- nrcSent[i, 2:12] / noTweets[[i]] * 100
}

sentimentTweets <- data.frame(afinSent)
sentimentTweets <- merge(sentimentTweets, bingSent[,c("index", "sentiment")], by="index")
sentimentTweets <- merge(sentimentTweets, nrcSent[, c("index", "sentiment")], by="index")

names(sentimentTweets) <- c("date", "afinn", "bing", "nrc")

write.csv(sentimentTweets, file="whatever you want to call it.csv", row.names = F)
```

For other platforms, we would do something similar. Just replace the extraction part, and adjust the cleaning and preprocessing to suit your needs.