Sentiment analysis

Miguel-Angel Canela, IESE Business School

October 31, 2017

Contents

Introduction

In a broad sense, **sentiment analysis** aims to determine the attitude of a speaker or a writer, in general, or with respect to a specific topic. In the complex variants of sentiment analysis, this attitude may be judgement, affective state or intended emotional communication. There are many methods to estimate sentiment, some of them involving very complex algorithms.

Sentiment can be measured in various ways. Business applications are usually restricted to **polarity** (positive/negative), based on counting positive and negative terms. Examples of positive and negative terms are:

- **Positive terms**: love, best, cool, great, good, amazing.
- **Negative terms**: hate, worst, sucks, awful, nightmare.

The sentiment analyst works on a collection of documents, counting the occurrences of the terms of both lists in every document. From these counts, different polarity measures can be calculated. In the example of this lecture, I explore how a company can measure customer sentiment using **Twitter data**. The 140-character limit imposed to tweets makes them manageable, and no normalization for the document length is needed. Moreover, although tweets may look as poor writing (they are), the length limit forces the writer to use a simple style which facilitates sentiment analysis.

The Twitter API

Most social networks, including Twitter, have an **application programming interface** (API). The API allows us to access the network database in a programmatic way (see `dev.twitter.com/overview/api` for an overview of the Twitter API). As in many other networks, the Twitter API requires the user to be authenticated. The **authentication** is carried out by means of a standard testing procedure called **OAuth** (see `oauth.net`).

In practice, the procedure is simple: an authorization server issues a set of passwords, called **tokens**, to the API client, so he/she can use these tokens to access the resources hosted by a resource server. In the case of Twitter, you have to make two steps:

- Register in Twitter, if you are not already registered. In this step, you create an account under a Twitter username.
- Create an application in the Twitter Applications website (`apps.twitter.com`). As a result, you get a set of four passwords: the **Consumer Key**, the **Consumer Secret**, the **Access Token** and the

Access Token Secret. These four passwords are needed to authenticate you before starting a tweet search.

Strictly speaking, the Twitter API is not an API, but a set of API's. More specifically, there is the **REST API**, on which we perform downloads one-at-a-time, and the **Streaming API**, which involves a continuous connection and is used to monitor or process tweets in real-time. In this lecture, I use the **Twitter Search API**, which is an element of the REST API which allows queries by content.

Capturing Twitter data

The Twitter data consist in the text of the tweets plus **metadata**. Depending on the extraction process, the metadata can have more or less attributes. There are many ways to download tweets, among them:

- Directly from the API website, using the browser.
- Using Java, as explained in Kumar *et al.* (2013).
- Using Python, as explained in Russell (2013).
- Using `R`, with the package `twitteR`.

In the example of this lecture, I use `twitteR`. The first step is the authentication, performed in `twitteR` with the function `setup_twitter_oauth`. This function has four arguments, which are the four passwords mentioned above. The passwords used here are associated to a Twitter identity created specially for this course, so you can use them freely.

```r
library(twitteR)
setup_twitter_oauth("9PQLhrb4ZUvte5Cb2VZRa7wce",
  "Xxkae5otxok1j30dwPNLgScJKzvLqSKJcvU77D13azYCDmoO1D",
  "2795394087-drbIfGPKwFZTfiNIKYXUqUrUvg1ypC2A7DfdXal",
  "FfN5Up4WtArKoLlitGUC2ukXKElmyA7paMpYphVxGQhlP")
```

```
## [1] "Using direct authentication"
```

If there is no error message, I have been authenticated, so I can start downloading tweets. As shown below, the function `searchTwitter` is quite easy to use. We specify a **search string**, such as '@Delta', which is send to the Twitter API, and taken according to Twitter rules, not to `R` rules. For instance, Twitter search is not case sensitive, so '@Delta' is the same as '@delta'. also, sentences are taken as a collection of words, so 'good bad' does not return tweets containing this sentence, but those containing *both* words 'good' and 'bad'.

I cannot download again the data used in the example, since the Twitter Search API only offers free access to recent tweets (one-week), but I describe here the data collection process, using `searchString="@Delta"`, and the actual date, which in `R` is `Sys.Date()`. Previously, I set the range of dates as one week before today. Note that `Sys.Date()` comes in a date format, so I can subtract 7 days. I transform the dates into strings with the function `as.character`, since this is what `searchTwitter` requires.

I set `n=20` to get the last 20 tweets for the period specified, meaning the last ones of the day before the date specified in the parameter `until`, which in this case means the yesterday's last 20 tweets. The default is `n=100`. In the example, I set `n=10^5`, which exceeds the numbers of tweets available in all cases. Mind that, when you asks for thousands of tweets, the download takes time.

```r
date1 <- as.character(Sys.Date() - 7)
date2 <- as.character(Sys.Date())
twlist = searchTwitter(searchString="@Delta", n=20, since=date1, until=date2)
```

This creates a list of `status` objects. I transform the list into a data frame with the function `twListToDF`. This is recommended, because it allows us to skip dealing with `status` objects.

```
twdf <- twListToDF(twlist)
```

Now, `twdf` is an ordinary data frame, with 100 rows and 16 columns, so we can work on it as usual.

```
dim(twdf)
```

```
## [1] 20 16
```

I check the contents of this data frame. I do not use `str` here, since it can give me trouble, due to the occurrence of special characters in the tweets, which is probable.

```
names(twdf)
```

```
##  [1] "text"          "favorited"     "favoriteCount" "replyToSN"
##  [5] "created"       "truncated"     "replyToSID"    "id"
##  [9] "replyToUID"    "statusSource"  "screenName"    "retweetCount"
## [13] "isRetweet"     "retweeted"     "longitude"     "latitude"
```

In the example of this lecture, I will only use two columns of this data frame: `text`, which is the text of the tweet, and `isRetweet`, which is a logical vector which takes the value `TRUE` for **retweets**.

NOTE. If you use the same passwords repeatedly, your download rate can be limited by the Twitter API, which creates a temporary problem. In that case, you get a message saying "Rate limited . . . ". So, if you plan to get practice capturing Twitter data, it is better that you create your own application.

The lexicons

Many attempts have been done to develop **lexicons** for various purposes, in particular the lists of positive and negative words that are used in sentiment analysis. Those used in the example have been prepared by M Hu and B Liu (see Hu & Liu, 2004). They come in two text files, `positive-words.txt` and `negative-words.txt`. The positive list contains 2,006 terms, starting with 'a+' and 'abound'. The negative list contains 4,783 terms, starting with '2-faced' and '2-faces'. I have downloaded these files from `www.cs.uic.edu~liub/FBS/sentiment-analysis`. Other choices are discussed in references 1, 2, 5 and 7.

Example: Consumer attitude towards airlines

The data

Airlines rank very low among the different industries in the American Customer Satisfaction Index (ACSI). But there is no uniformity within the airline industry. At the beginning of 2015, among the six principal companies in the United States, JetBlue and Southwest were on top, followed by Delta. In this example, inspired in a talk of J Breen at the Boston Predictive Analytics MeetUp in 2011, I present an approach to measuring customer attitude towards airlines, based on Twitter data.

I describe with detail the analysis for Delta, which was replicated for the other cases. In total, it involved six companies: American Airlines (AA) Delta Airlines (DL), JetBlue Airways (B6), Southwest Airlines (WN), United Airlines (UA) and US Airways (US).

As search strings, I used the Twitter names of the six companies: @AmericanAir, @Delta, @JetBlue, @SouthwestAir, @united and @USAirways. Omitting the at sign (@) could create trouble here, since some companies, like JetBlue, are easily identified, but 'Delta' could refer to other things (such as the delta of the Nile). The Twitter names allow an equal treatment of the six companies. I limited the search to the week starting January 5 (Monday) and ending January 11 (Sunday), obtaining 22,344 tweets mentioning @AmericanAir, 8,075 for @Delta, 6,125 for @JetBlue, 8,189 for @SouthwestAir, 16,487 for @united and 6,069 for @USAirways.

The data are stored in `RData` files, which can be imported with the function `load`. Alternatively, by double-clicking the icon of the `RData` file. This would open the `R` console (or RStudio, depending on the configuration of our computer) and load the objects contained in the file.

```
load("Delta.RData")
```

This `RData` file contains a single object, a data frame called `twdf`. This can be checked with the function `ls`, which lists all the objects that the user has created in the current session.

```
ls()
```

```
## [1] "twdf"
```

I take a look at the first column of this data frame.

```
twdf$text[1:5]
```

```
## [1] "Tonight's @delta flight from  got a toe-tapper / armrest rapper better than a farter #businesst
## [2] "#fixitjesus apparently @delta can't"
## [3] "I'm talking about you @delta"
## [4] "@KLM @Delta ah thanks for the tip! We gaan er naar kijken. ^^ het moet idd een mooie trip worde
## [5] "@Delta I love your sky magazine articles- especially this month's feature on @AmericanExpress c
```

Preprocessing

I present next some transformations performed before counting positive and negative words. Some of these transformations are specific of this case. Some are not needed here, since the objective of the analysis is to assess the polarity of the messages, but they are typical of text mining, and help to have a clean picture of the Twitter data. In most cases, I add extra white space to prevent terms merging.

I start by removing all **control characters**, such as '\n', which indicates a line break. This can occur in Twitter data, since users are allowed to break lines within the tweet (e.g. tweet 345). Note that, if a positive term, such as 'good', appears in a tweet after a line break, in the corresponding row of our data set we would find '\ngood', which would not be counted as positive. So, we need to take care of this. Also, as usual in this type of analysis, I remove the apostrophes.

```
library(stringr)
twdf$text <- str_replace_all(twdf$text, "[[:cntrl:]]", " ")
twdf$text <- str_replace_all(twdf$text, "'", "")
```

Emoticons, such as ':)', reflect positive or negative sentiment. Here, I transform ':)' and ':-)' into 'smiley', and ':(' and ':((' into 'weeping'. I use regular expressions to shorten the code. The question mark in the first line means *zero or more times* and the plus sign in the second line *one or more times*. The square brackets are needed, because the parenthesis can play a special role in a regular expression.

```
twdf$text <- str_replace_all(twdf$text, ":-?[)]", " smiley ")
twdf$text <- str_replace_all(twdf$text, ":[(]+", " weeping ")
```

Twitter text can be contaminated with **HTML conventions**, such as '&amp;' which is a **character entity** which stands for the ampersand symbol. Other character entities are '&lt;' and '&gt;', which mean 'less than' (<) and 'greater than' (>), respectively. The second one appears frequently in these tweets, sometimes before an URL, sometimes to indicate a route, as in 'ATL -> LAX'. I remove all this in one shot, again with a regular expression.

```
twdf$text <- str_replace_all(twdf$text, "&[a-z]+;", " ")
```

I remove all the URL's, such as `http://t.co/U2SXuBOaEU` in tweet 5. This regular expression would not work for any possible URL, but it is enough for those found here, which are not the real URL's written in the tweets, but fictitious ones provided by Twitter to keep things short.

```
twdf$text <- str_replace_all(twdf$text, "http://[/a-zA-Z0-9.]+", " ")
```

I remove all the Twitter usernames, such as '@Delta' in tweet 1.

```
twdf$text <- str_replace_all(twdf$text, "@([A-Za-z]+[A-Za-z0-9]+)", " ")
```

It is typical to get trouble when processing Twitter data, due to the inclusion in the tweets of special characters and emojis. These characters would not affect my polarity calculations, but some of them can stop the conversion to lower case, which we cannot skip. Special characters may look differently in the `R` console in different computers. For instance, in my Windows computer, I find in tweet 84 something presented as `\u0083`. At home, the same thing appears in my Macintosh as `\x83`. You can find it as `<U+0083>` somewhere else. It is a symbol called *no break here*, which can be depicted as a calligraphic *f*. So, I remove everything which is not letters or white space.

```
twdf$text <- str_replace_all(twdf$text, "[^a-zA-Z ]", " ")
```

This transformation also removed punctuation, the dollar ($) and the **hashtag** (#) symbols, and the numbers that survived to the previous transformations. I can perform now the conversion to lower case.

```
twdf$text <- str_to_lower(twdf$text)
```

Finally, I strip white space. In the first line, I replace any sequence of consecutive white spaces by a single space. In the second line, I use the function `str_trim` to trim the white space at the extremes of every element of the vector `twdf$text`.

```
twdf$text <- str_replace_all(twdf$text, " +", " ")
twdf$text <- str_trim(twdf$text)
```

I take a new look at the first five tweets, to check the effect of these preprocessing operations.

```
twdf$text[1:5]
```

```
## [1] "tonights flight from got a toe tapper armrest rapper better than a farter businesstravel"
## [2] "fixitjesus apparently cant"
## [3] "im talking about you"
## [4] "ah thanks for the tip we gaan er naar kijken het moet idd een mooie trip worden"
## [5] "i love your sky magazine articles especially this months feature on ceo kenneth chenault"
```

Polarity measurement

I import the positive and negative lists with the function `readLines`. This creates two character vectors of lengths 2,006 and 4,783, respectively.

```
pos.terms <- readLines("positive-words.txt")
neg.terms <- readLines("negative-words.txt")
```

I add some terms to these lists, to make them specific for the airlines industry. Breen suggests adding 'upgrade' to the positive terms, and I also added 'smiley', explained above. To the list of negative words, I add some suggested by Breen, some on my own, based on a quick examination of some tweets (you can probably do better).

```
pos.terms <- c(pos.terms, "smiley", "upgrade")
neg.terms <- c(neg.terms, "cancel", "cancelled", "fix", "weeping", "fucked",
  "mechanical", "mistreat", "mistreated", "piss", "pissed", "shitty",
  "stranded", "wait", "waiting", "wtf")
```

Next, I define two functions that count the matches with the positive and negative lexicons, respectively.

```
posCount <- function(x) sum(unlist(str_split(x, " ")) %in% pos.terms)
negCount <- function(x) sum(unlist(str_split(x, " ")) %in% neg.terms)
```

These functions canot be applied directly to `twdf$text`, because `unlist` would transform the list resulting from `str_split` into a single vector, so we would get the total number of positive and negative terms in the whole collection of tweets. To apply separately these functions to the each document of the Delta collection, I use a `for` loop. Note that, since I have to start the loop with a predefined object, I define previously the new vectors as `NULL`.

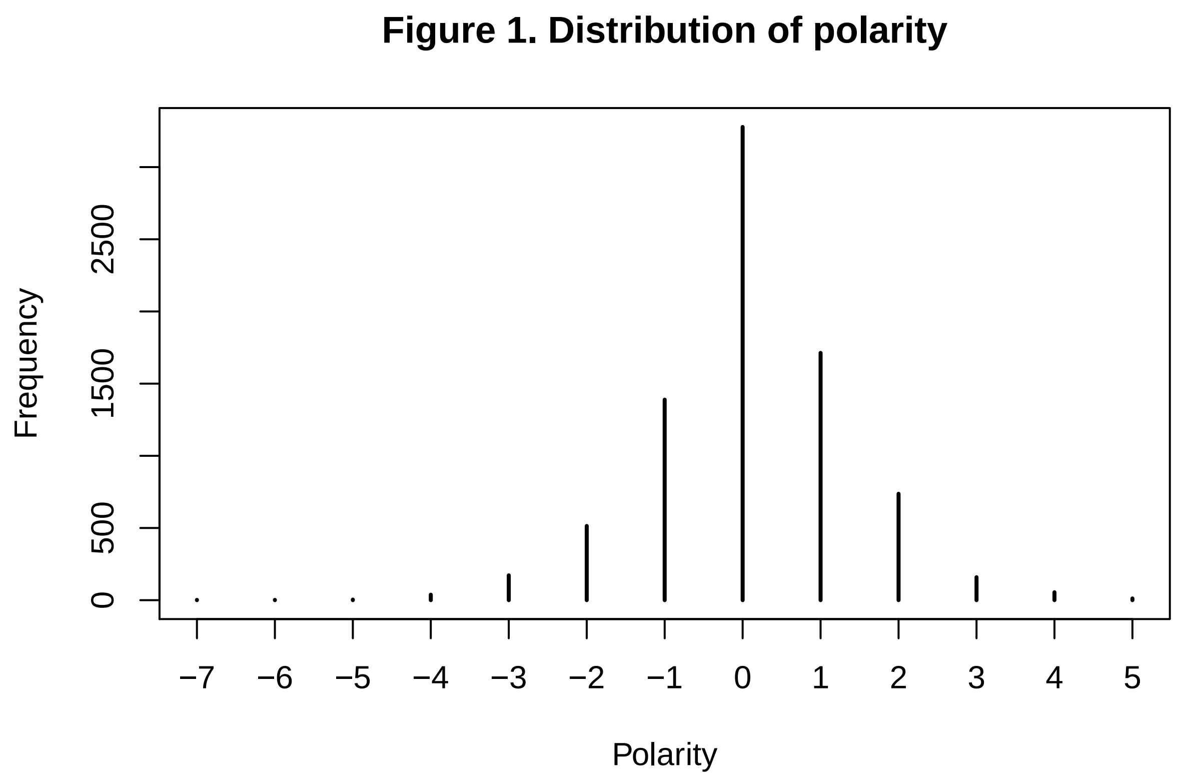
```
positive <- NULL
for(i in 1:8075) positive[i] <- posCount(twdf$text[i])
negative <- NULL
for(i in 1:8075) negative[i] <- negCount(twdf$text[i])
```

This leaves us with two numeric vectors of length 8,075. The polarity score is obtained as the difference.

```
polarity <- positive - negative
```

The distribution of the polarity scores can be seen in Figure 1.

```
plot(table(polarity), main="Figure 1. Distribution of polarity",
  xlab="Polarity", ylab="Frequency", type="h")
```



Now, I calculate a polarity measure for each company as the ratio of the number of positive tweets by the number of negative tweets. This gives me 1.263 for Delta.

```
sum(polarity>0)/sum(polarity<0)
```

```
## [1] 1.261669
```

Discarding retweets

As you probably guess, tweets starting with 'RT' are retweets, that is, redundant information. So, some analysts ignore them, restricting the analysis to the non-retweets (you may disagree). In our data set, this is easy to manage with the field `isRetweet`.

```r
sum(twdf$isRetweet)/nrow(twdf)
```

```
## [1] 0.2557276
```

For our six companies, the percentage of retweet ranges between 20 and 40%. To get this, I count the terms where `isRetweet` is false. The admiration mark (!) means NOT, so `!isRetweet` is true for the non-retweets. So, I can easily recalculate the polarity measure as follows.

```r
sum(polarity>0 & !twdf$isRetweet)/sum(polarity<0 & !twdf$isRetweet)
```

```
## [1] 1.017719
```

I have collected in Table 1 the results of this analysis, for the six companies.

Table 1: Results for the six airlines

	All tweets	Non-retweets	Prop. retweet	ACSI
American	1.08	0.68	0.63	66
Delta	1.26	1.02	0.74	71
JetBlue	1.87	1.53	0.70	79
Southwest	1.93	1.74	0.81	78
United	0.49	0.54	0.75	60
US Airways	0.52	0.57	0.63	66

Alternative polarity measure

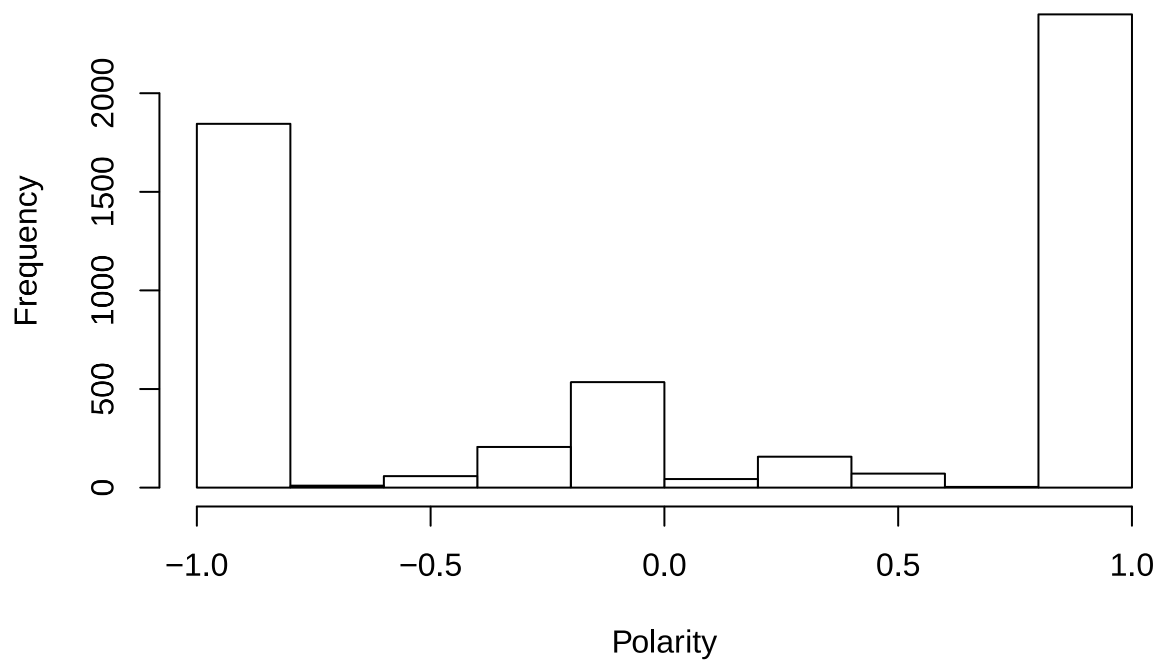
The analysis of the previous section followed (loosely) Breen's approach. I present now an alternative analysis. For every tweet, I define a polarity score as the following ratio.

```r
polarity <- (positive - negative)/(positive + negative)
```

By definition, this measure is already normalized, taking values from -1 to 1. Note that, when a tweet does not contain positive nor negative words, this is 0/0, so `R` produces a `NaN` value for those tweets. The distribution of this polarity measure in the Delta collection is seen in Figure 2, obtained as follows.

```r
hist(polarity, main="Figure 2. Distribution of polarity (alternative)",
  xlab="Polarity")
```

Figure 2. Distribution of polarity (alternative)



Here, most of the tweets fall on the extremes, which is convenient for my analysis. Table 2 shows the average polarity for the six companies. The proportion of neutral tweets (not containing positive nor negative words) is quite uniform across companies. Sorting the six companies by the average polarity is consistent with the ACSI ranking. The calculations were performed as follows. The analysis has been replicated for the subcollection of tweets obtained dropping the retweets.

```
mean(polarity, na.rm=T)
```

```
## [1] 0.1036782
```

```
mean(polarity[!twdf$isRetweet], na.rm=T)
```

```
## [1] 0.01079973
```

Table 2: Alternative analysis

	All tweets	Non-retweets
American	0.05	-0.15
Delta	0.10	0.01
JetBlue	0.27	0.19
Southwest	0.28	0.22
United	-0.28	-0.24
US Airways	-0.26	-0.22

References

1. S Baccianella, A Esuli & F Sebastiani (2010), SentiWordNet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining, *Proceedings of the Seventh Conference on International Language Resources and Evaluation*, 2200-2204.

2. `http://www.wjh.harvard.edu/$\sim$inquirer`.

3. M Hu & B Liu (2004), Mining and Summarizing Customer Reviews, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, Seattle (Washington, USA).

4. S Kumar (2013), *Twitter Data Analytics*, Springer.

5. FA Nielsen (2011), A new ANEW: Evaluation of a word list for sentiment analysis in microblogs, arXiv:1103.2903v1.

6. MA Russell (2013), *Mining the Social Web*, O'Reilly.

7. T Wilson, J Wiebe & P Hoffmann (2005), Recognizing contextual polarity in phrase-level sentiment analysis, *Proceedings of HLT-EMNLP-2005*.