Text mining

Miguel-Angel Canela, IESE Business School

November 8, 2017

Contents

Introduction 1
Preprocessing text data 1
Extra packages 2
Regular expressions 2
Special characters 2
Example: Yelp reviews 3
References 12

Introduction

Broadly speaking, **text mining** is data mining applied to a collection of documents. The text mining process can start when we already have the collection, or building it may be part of the process. I assume in this lecture that the documents have already been captured and stored in electronic form, as plain text.

Most of the text data on which data scientists work are captured from Internet. Specially relevant, for business, are the data captured from social networks. So, **web mining** has emerged as a new, promising subfield, similar to text mining but taking advantage of the extra information available in Web documents. These documents come in **XML**, **HTML** or **JSON** format, and include structural **markup**, which distinguishes parts of the document. The markup can be used in the mining process. Although it sometimes involves arbitrary and unpredictable choices by individual web designers, this disadvantage is offset by the amount of data available.

To make text data useful, unstructured text data is converted into structured data for further processing. This conversion is typically performed by counting terms. The resulting table is called the **term-document matrix**. The rows correspond to the documents in our collection and the columns to a set of selected terms. The entries can be 1/0 (binary frequencies), raw frequencies, or some normalized version of the frequency with which that term occurs in that document.

Preprocessing text data

Previous to building the term-document matrix, some transformations are performed. They are called, generically, **preprocessing**. Tools for preprocessing text are available in many programming languages, including `R`.

Typical preprocessing steps are:

- Converting to lower case, which simplifies the preprocessing task and helps to avoid confusion when counting the occurrences of terms in the documents.
- Removing numbers and punctuation.
- Removing postal and email addresses, telephone numbers and URL's, since they only affect one particular document (there are exceptions to this).
- Removing **stopwords**, that is, articles and other words that convey little or no information. Lists of stopwords are available in Internet for practically any language you may be interested in. For

the English language, a popular choice is the list of the SMART information retrieval system (`http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a11-smart-stop-list/english.stop`), which I use in the example of this lecture. Note that, although removing stopwords is regarded as a preprocessing routine, stopwords can also be discarded after counting terms.

- Replacing words that are synonyms, and plural and other variants of words, by a single term. In the example of this lecture, this is based on a list of synonyms created ad hoc, but there are more sophisticated methods.

In addition to these typical steps, it is important, in some examples, to take into account that users of systems such as SMS or e-mail have shortenings for communication. The resulting genre of communication is called **texting**. For instance, the word 'text' may appear as 'txt', or 'your' as 'ur'.

Extra packages

`stringr` is a popular package for text manipulation, which I use in the example of this lecture. Almost all the `stringr` functions are available in plain `R`, but their names in `stringr` are intuitively clear, and the syntax is straightforward. A clear introduction to `stringr` can be found in the `www.stringr.tidyverse.org`. For a more general presentation of string functions, I would recommend Chapter 14 of Wickham & Grolemund (2016).

The package `tm` contains functions that render operations like removing stopwords or grouping synomyms very easy, and also has a function for building term-document matrices. But this comes at the price of working with complex objects, so I do not use it in this lecture. Nevertheless, `tm` is quite popular and other packages, such as `tm.plugin.webmining`, for capturing news, use `tm` objects. So, if you plan to work on this, it may be worth learning a bit about these objects.

Regular expressions

Some of the transformations performed in the preprocessing stage are dramatically simplified by using **regular expressions**. A regular expression is a pattern which describes a set of strings. For instance, `[0-9]` stands for any digit, and `[A-Z]` for any capital letter. The square brackets indicate *any* of the characters enclosed. This is called a **character class**.

Regular expressions get more powerful when complemented with **quantifiers**. For instance, followed by a plus sign (+), a character class indicates a sequence of any length. So, `[0-9]+` indicates any sequence of digits, therefore any number.

Regular expressions are a whole chapter of programming, with entire books, such as Friedl (2007), devoted to them. Beginners may also try the `regexr` website at `www.regexr.com`, which makes fun of learning regular expressions. To learn how to use regular expressions within `stringr` functions, you can look at `www.stringr.tidyverse.org/articles/regular-expressions.html`

Special characters

Text imported from PDF or HTML documents, or from devices like mobile phones, may contain special symbols like the n-dash (–), the left/right quotation marks (", ', etc), or the three-dot character (...), which is better to control, to avoid confusion. Since the example of this lecture is simple enough, I do not develop this point here, but mind that, if you capture text data on your own, you will probably find some of this in your data. Even if the documents are expected to be in English, they can be contaminated by other languages: Han characters, German umlaut, Spanish ñ, etc.

Another source of trouble is that these special symbols can be encoded by different computers or different text editors in different ways. The typical **encodings** in the Western world are UTF-8 and Latin-1. `R` can

deal with these two, but may have trouble with other encodings. I do not discuss encodings in this course. If you are interested, you may take a look at Korpela (2006).

Example: Yelp reviews

Introduction

Yelp is a well known platform where customers post their reviews on restaurants throughout the world. The reviewers share their opinions on various aspects, like table service, the food itself, or the drinks, rating the restaurant (1-5 scale).

Besides comparing ratings across places and brands, it is interesting to use the reviews in a serious, systematic way. After capturing the information displayed in the Yelp website, the reviews can be translated into numeric data, so that they could be correlated with the ratings. If we could manage this, we would be able to identify what the satisfied and the unsatisfied customers talk about, which would allow us to learn about customer's feelings and expectations.

By means of **web scraping** techniques, many data companies are building databases from data published on Internet. Once the code for the systematic extraction of the data has been written, it can be periodically applied for monitoring sales, sentiment, trends, etc. A static web page is an HTML file which contains all the information sought. Dynamic web pages contain also code snippets, usually in **JavaScript**. The structure of these HTML documents is always the same, and the difficulty to scrape this information and store it in a table depends on the dynamic part of the page. In the case of Yelp, the task is relatively easy.

The data

The data for this example are associated to restaurants: (a) with the 'Tacos' tag, (b) in the 'Moderate Price' segment, and (c) in the New York area. The list of restaurants can be found in

`www.yelp.com/search?find_loc=New+York+NY&start=0&cflt=tacos&attrs=RestaurantsPriceRange2.2`.

The actual data set, coming in the file `tacos.csv`, covers the five leading restaurants (at the time of the data collection): Arriba Arriba Sunnyside (172 reviews), Donna (107), Dylan Murphy's (87), La Esquina (47) and Villalobos (222). Each restaurant has a series of webpages, displaying all the reviews and ratings. The data set has five columns, corresponding to the restaurant name, the rating, the date, the text of the review and the author of the review. We import the data in the usal way, with `read.csv`.

```
tacos <- read.csv("tacos.csv", stringsAsFactors=F)
str(tacos)
```

```
## 'data.frame':    635 obs. of  5 variables:
##  $ restaurant: chr  "Arriba Arriba Sunnyside" "Arriba Arriba Sunnyside" "Arriba Arriba Sunnyside" "A
##  $ rating    : int  4 2 5 4 5 1 4 5 5 5 ...
##  $ date      : chr  "2017-03-13" "2017-02-06" "2017-03-07" "2016-04-30" ...
##  $ review    : chr  "Great happy hour spacial , food and service is great  Did I mention $6 frozen m
##  $ author    : chr  "Jason M." "Michael T." "Mike P." "Nick M." ...
```

Now, `tacos` is a data frame with 635 rows and 5 columns. According to popular wisdom, customers are more willing to write about the services provided by a particular company when they are unhappy with that company than when they are happy. But we find both satisfaction and unsatisfaction here, with average ratings between 3 and 4 for the five restaurants.

```
sort(tapply(tacos$rating, tacos$restaurant, mean), decreasing=T)
```

```
##                   Donna          Dylan Murphy's              La Esquina
##                3.934579                3.793103                3.744681
##              Villalobos Arriba Arriba Sunnyside
##                3.387387                3.029070
```

Anyway, it is true that the unsatisfied reviewers write longer than the satisfied ones. To manage the length of the reviews, I use the function `str_length`, which calculates the number of characters of a string (it is not exactly so if there are special characters, but this does not matter here). The following tables show that, on the average, unsatisfied reviewers write longer.

```
library(stringr)
sort(tapply(str_length(tacos$review), tacos$restaurant, mean))
```

```
##          Dylan Murphy's              La Esquina                   Donna
##                377.5172                448.0213                537.1869
## Arriba Arriba Sunnyside              Villalobos
##                580.7209                697.7973
```

```
tapply(str_length(tacos$review), tacos$rating, mean)
```

```
##        1        2        3        4        5
## 717.5904 750.3373 629.2286 583.4565 393.9556
```

In this presentation, I use the first 3 reviews to track the transformations performed in the subsequent steps of the processing stage.

```
tacos$review[1:3]
```

```
## [1] "Great happy hour spacial , food and service is great  Did I mention $6 frozen margaritas"
## [2] "I'll give the food a solid 4 stars. It is very good. My issue is with the shady business practi
## [3] "Good food and great drinks. The Diablo wings live up to the name, and the Mole dishes are excel
```

Preprocessing

To get a clear job, I create a separate vector of reviews (not needed). The conversion to lower case is performed with the function `str_to_lower`.

```
review <- tacos$review
review <- str_to_lower(review)
```

Substitutions can be performed with the function `str_replace_all`, which has three arguments with no default: `string` (the string in which the substitution takes place, `pattern` (what we wish to be replaced) and `replacement` (what replaces the pattern). If the arguments are specified in this precise order, they do not need to be named. First, I drop the apostrophe,

```
review <- str_replace_all(string=review, pattern="'", replacement="")
```

The next step is to remove the punctuation: period (.), comma (,), colon (:), semicolon (;), question mark (?), etc. I have already taken care of the apostrophe, which was just dropped (we wish to transform 'are'nt' into 'arent', not into two separate words). I replace the rest of the punctuation marks by white space, to prevent mergers. I use a regular expression that covers all the punctuation marks and allows me to get rid also of the numbers and the dollar symbol ($). The 'hat' symbol within the brackets means negation, so the regular expression `[^a-z ]` stands for *anything which is not a letter or white space*. I add extra white space to prevent terms merging.

```
review <- str_replace_all(review, "[^a-z ]", " ")
```

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 `review`.

```
review <- str_replace_all(review, " +", " ")
review <- str_trim(review)
```

I list again the first 3 elements of `review`.

```
review[1:3]
```

```
## [1] "great happy hour spacial food and service is great did i mention frozen margaritas"
## [2] "ill give the food a solid stars it is very good my issue is with the shady business practices i
## [3] "good food and great drinks the diablo wings live up to the name and the mole dishes are excelle
```

Bag of words data

I transform each document into a **bag of words**. This is easily done with the function `str_split`, whose syntax is straightforward: the argument `pattern` specifies the substring used for the split. Since I have already removed the extra white space and the punctuation, I can use a single white space as the split pattern.

```
term_list <- str_split(review, pattern=" ")
```

The split is a trivial operation, but there is a technical point here. Splitting the first document gives us a bag of 14 words, which in `R` is managed as a character vector of length 14. But the second document has 136 words, the third document 31 words, etc. So, `term_list` cannot be a matrix nor any rectangular object. R creates here a list (a list is like a vector, but its elements can have any structure). Here, `term_list` is a list of length 635, whose elements are character vectors of different lengths.

```
term_list[1:3]
```

```
## [[1]]
##  [1] "great"      "happy"      "hour"       "spacial"    "food"
##  [6] "and"        "service"    "is"         "great"      "did"
## [11] "i"          "mention"    "frozen"     "margaritas"
##
## [[2]]
##   [1] "ill"          "give"         "the"          "food"
##   [5] "a"            "solid"        "stars"        "it"
##   [9] "is"           "very"         "good"         "my"
##  [13] "issue"        "is"           "with"         "the"
##  [17] "shady"        "business"     "practices"    "i"
##  [21] "ordered"      "enchiladas"   "the"          "menu"
##  [25] "stated"       "of"           "them"         "when"
##  [29] "i"            "received"     "my"           "order"
##  [33] "there"        "were"         "i"            "triple"
##  [37] "checked"      "the"          "item"         "i"
##  [41] "ordered"      "and"          "i"            "was"
##  [45] "correct"      "the"          "menu"         "item"
##  [49] "clearly"      "stated"       "as"           "i"
##  [53] "am"           "not"          "one"          "to"
##  [57] "quickly"      "write"        "a"            "bad"
##  [61] "review"       "i"            "contacted"    "the"
##  [65] "restaurant"   "via"          "the"          "yelp"
##  [69] "app"          "no"           "response"     "i"
##  [73] "went"         "to"           "attempt"      "to"
##  [77] "order"        "the"          "same"         "item"
##  [81] "and"          "a"            "banner"       "popped"
##  [85] "up"           "saying"       "the"          "description"
##  [89] "of"           "the"          "item"         "had"
##  [93] "changed"      "since"        "i"            "last"
##  [97] "ordered"      "it"           "the"          "new"
## [101] "description"  "not"          "stating"      "a"
## [105] "quantity"     "so"           "management"   "saw"
```

```
## [109] "my"           "inquiry"      "ignored"      "me"
## [113] "and"          "changed"      "the"          "menu"
## [117] "item"         "without"      "offering"     "any"
## [121] "compensation" "that"         "is"           "some"
## [125] "shady"        "business"     "you"          "just"
## [129] "lost"         "a"            "customer"     "over"
## [133] "one"          "enchilada"    "adios"        "muchachos"
##
## [[3]]
##  [1] "good"      "food"      "and"       "great"     "drinks"
##  [6] "the"       "diablo"    "wings"     "live"      "up"
## [11] "to"        "the"       "name"      "and"       "the"
## [16] "mole"      "dishes"    "are"       "excellent" "taco"
## [21] "tuesdays"  "are"       "great"     "way"       "to"
## [26] "try"       "different" "styles"    "tacos"     "for"
## [31] "cheap"
```

Removing stopwords

My next step is to shorten the vectors of this list by removing the stopwords. First, I import the stopwords list. In the file `stopwords.txt`, each term takes one line, meaning that they are separated by a return character. To import such files, it is faster (and simpler) to use the function `readLines`.

```
stopwords <- readLines("stopwords.txt")
str(stopwords)
```

```
##  chr [1:571] "a" "a's" "able" "about" "above" "according" ...
```

Now, `stopwords` is a character vector of length 571. Even if this is a well known and respectable collection, I am not entirely happy with it. After a quick inspection, I enlarge the stopwords catalogue with some of my own. I include in the list also the names of the restaurants.

```
stopwords <- c(stopwords, "arent", "aint", "cant", "couldnt", "didnt",
  "doesnt", "dont", "havent", "hasnt", "im", "ive", "thats", "theyre",
  "wasnt", "whos", "youre", "arriba", "donna", "dylan", "esquina",
  "villalobos", "montclair", "sunnyside")
```

Next, I define a function that removes any term of the stopwords collection from a character vector.

```
stopRemove <- function(x) x[!(x %in% stopwords)]
```

To understand how this function works, let me use a very simple example,

```
x <- c("Messi", "is", "a", "good", "player")
```

Then,

```
x %in% stopwords
```

```
## [1] FALSE  TRUE  TRUE FALSE FALSE
```

is a logical vector. Now, the exclamation mark (`!`) turns `TRUE` into `FALSE` and conversely. So, the following selection retains those terms which are not stopwords.

```
x[!(x %in% stopwords)]
```

```
## [1] "Messi"  "good"   "player"
```

I apply this selection to every element of `term_list`. Since `term_list` is a list, I cannot apply `stopRemove` directly, so I use a `for` loop. Note that I use double brackets to select a single element of the list.

```
for(i in 1:635) term_list[[i]] <- stopRemove(term_list[[i]])
term_list[1:3]
```

```
## [[1]]
##  [1] "great"      "happy"      "hour"       "spacial"    "food"
##  [6] "service"    "great"      "mention"    "frozen"     "margaritas"
##
## [[2]]
##  [1] "ill"          "give"         "food"         "solid"
##  [5] "stars"        "good"         "issue"        "shady"
##  [9] "business"     "practices"    "ordered"      "enchiladas"
## [13] "menu"         "stated"       "received"     "order"
## [17] "triple"       "checked"      "item"         "ordered"
## [21] "correct"      "menu"         "item"         "stated"
## [25] "quickly"      "write"        "bad"          "review"
## [29] "contacted"    "restaurant"   "yelp"         "app"
## [33] "response"     "attempt"      "order"        "item"
## [37] "banner"       "popped"       "description"  "item"
## [41] "changed"      "ordered"      "description"  "stating"
## [45] "quantity"     "management"   "inquiry"      "changed"
## [49] "menu"         "item"         "offering"     "compensation"
## [53] "shady"        "business"     "lost"         "customer"
## [57] "enchilada"    "adios"        "muchachos"
##
## [[3]]
##  [1] "good"      "food"      "great"     "drinks"    "diablo"
##  [6] "wings"     "live"      "mole"      "dishes"    "excellent"
## [11] "taco"      "tuesdays"  "great"     "styles"    "tacos"
## [16] "cheap"
```

Most frequent terms

Now, I build a table of most frequent terms, sorted by frequency. First, I transform `term_list` into a vector with the function `unlist`. Then, I count the number of times every term occurs with `table`. Finally, I sort this with the function `sort`. This produces a very long vector, with 7,228 entries, most of them irrelevant. I pick, for exploratory purposes, the top-100 terms.

```
freq_term <- sort(table(unlist(term_list)), decreasing=T)
length(freq_term)
```

```
## [1] 4944
```

```
freq_term[1:100]
```

```
##
##         food         good        place        tacos        great
##          447          404          393          369          319
##      service         taco          bar         back       drinks
##          226          218          196          191          174
##         time      mexican         nice         menu        night
##          170          152          152          146          138
##        order   restaurant      ordered        chips    delicious
##          134          134          119          117          115
##    guacamole         love       people       pretty        drink
##          114          113          107          105           97
```

```
##        staff      chicken        table         make        small
##           97           96           92           86           81
##         made        salsa        happy     friendly        music
##           80           78           77           76           75
##          bit      friends        fresh   margaritas         guac
##           74           74           73           72           71
##     waitress      amazing         fish         hour        times
##           69           68           67           66           65
##       dinner         give        asked          bad     cocktail
##           63           61           60           60           60
##       friend         spot         meal       shrimp         feel
##           57           57           56           56           52
##         side        taste         cool        lunch   atmosphere
##           52           52           51           51           50
##          eat       flavor         open      awesome          fun
##           50           50           50           49           49
##         wait          big   experience      special    cocktails
##           49           48           48           48           46
##          day    margarita neighborhood      minutes        super
##           46           46           46           45           45
##      thought         told       wanted    bartender         pork
##           45           45           45           44           44
##    recommend        thing        beans        decor         find
##           44           44           43           43           43
##         left        tasty         area         corn           id
##           43           43           42           41           41
##         long        space        worth    attentive   bartenders
##           41           41           41           40           40
##       prices         rice       server       coming          lot
##           40           40           40           39           39
```

Since I only have to deal with a low number of terms, I create a list of equivalent terms manually.

```
top_terms <- list(food="food", good="good", place="place",
  taco=c("taco", "tacos"), great="great", service="service", bar="bar",
  drink=c("drink", "drinks", "drinking"),
  time=c("time", "times", "timely"), mexican="mexican", nice="nice",
  menu="menu", night="night", order=c("order", "ordered", "ordering"),
  restaurant="restaurant", chips="chips", delicious="delicious",
  guac=c("guac", "guacamole"), love=c("love", "lovely", "loving"),
  people="people", staff="staff", chicken="chicken",
  table=c("table", "tables"), make=c("make", "makes", "made"),
  small="small", salsa="salsa",
  friend=c("friend", "friends", "friendly"), music="music",
  fresh="fresh", margarita=c("margarita", "margaritas"),
  waiter=c("waiter", "waitress"), fish="fish", hour=c("hour", "hours"),
  dinner="dinner", give=c("give","gives", "gave", "giving", "given"),
  ask=c("ask", "asks", "asking", "asked"), bad="bad",
  cocktail=c("cocktail", "coktail", "cocktails"))
```

Term-document matrix

I build a binary term-document matrix, whose columns are dummies for the occurrence of these top terms in the documents of the corpus. First, I create a matrix of the adequate dimensions,

```
TD <- matrix(nrow=635, ncol=38)
```

Then, I replace every term of the matrix by the corresponding 1/0 value. This can be done in many ways, but using a `for` **loop** is probably the simplest way.

```
for(i in 1:635) for(j in 1:38) TD[i,j] <- max(top_terms[[j]] %in% term_list[[i]])
colnames(TD) <- names(top_terms)
```

To understand how this works, pick `i=3` and `j=8`. Then

```
top_terms[[8]] %in% term_list[[3]]
```

```
## [1] FALSE  TRUE FALSE
```

creates a logical vector indicating which terms of `top_terms[[8]]` (`drink`, `drinks` and `drinking`) are in `term_list[[3]]`. When we apply the function `max`, the `TRUE/FALSE` values are coerced to `1/0`, and the maximum is `1` when there is at least one `TRUE`.

```
max(top_terms[[8]] %in% term_list[[3]])
```

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

We can take a look at this matrix with

```
TD[1:10, 1:12]
```

```
##       food good place taco great service bar drink time mexican nice menu
##  [1,]    1    0     0    0     1       1   0     0    0       0    0    0
##  [2,]    1    1     0    0     0       0   0     0    0       0    0    1
##  [3,]    1    1     0    1     1       0   0     1    0       0    0    0
##  [4,]    1    1     0    0     0       1   0     0    0       0    0    1
##  [5,]    1    1     1    0     0       0   0     1    0       1    0    0
##  [6,]    0    0     0    0     0       0   0     1    1       0    0    0
##  [7,]    1    1     1    0     1       0   0     0    0       1    1    0
##  [8,]    1    0     1    1     0       0   0     0    1       1    1    0
##  [9,]    1    1     1    0     1       0   0     1    1       0    0    1
## [10,]    1    0     0    0     0       0   0     1    0       0    0    1
```

```
df <- data.frame(TD, rating=tacos$rating)
```

First analysis

We take a high rating as an indication of a reviewer being satisfied. So, a positive correlation between `rating` and one of the columns of the term document matrix `TD` suggests that the satisfied reviewers include the corresponding term in their reviews more often than the unsatisfied ones. A negative correlation will suggest the opposite. So, I create here here a vector (`term_cor`) whose terms are these correlations. More specifically, the element `i` of this vector will be the correlation of `rating` and the column `TD[, i]`. I do this with a loop, creating first a vector of length zero, since `R` cannot start the loop in the void.

```
term_cor <- NULL
for(i in 1:38) term_cor[i] <- cor(tacos$rating, TD[, i])
```

To make it readable, I put names to the elements of this vector, sorting them. Also, since `R` reports too many decimals, I round to 3 decimals.

```
names(term_cor) <- colnames(TD)
round(sort(term_cor), 3)
```

```
##        ask     waiter        bad      order      table       give
```

```
##     -0.278     -0.258     -0.220     -0.218     -0.202     -0.182
## restaurant       hour       food      chips     people    service
##     -0.156     -0.135     -0.129     -0.114     -0.109     -0.106
##      place      salsa       time     dinner  margarita      small
##     -0.089     -0.061     -0.060     -0.057     -0.053     -0.031
##    chicken       menu    mexican       fish       make      drink
##     -0.027     -0.021     -0.001      0.000      0.012      0.016
##       good       guac      staff      night     friend       nice
##      0.020      0.022      0.027      0.038      0.046      0.053
##       taco      fresh   cocktail        bar      music  delicious
##      0.073      0.079      0.127      0.133      0.145      0.209
##       love      great
##      0.213      0.289
```

Some of the strongest correlations are obvious, like those of 'bad', 'great' and 'love'. But some others uncover interesting facts, like those of 'ask' and 'music'. On the bottom side, the most relevant trait is the accumulation of terms related to the service. On the top side, terms not directly related to the food itself (cocktails, bar and music). So the tacos are not the main issue.

Detailed analysis

The correlation analysis performed above was interesting, but it has a shortcoming. What if the customers are happy with the fish in this place but not elsewhere? This fact will not probably show up unless we perform the analysis separately for each restaurant.

To select the reviews of Arriba Arriba Sunnyside, I create a logical vector

```
arriba <- tacos$restaurant == "Arriba Arriba Sunnyside"
```

I get so a list correlations for this restaurant, which is the lowest rated. We get the music on top, but not the cocktails. Also, 'mexican' and 'margarita' go up. Since the margarita is itself a cocktail, this may be suggesting something, but I'm not in the situation to say more from here.

```
term_cor_arriba <- NULL
for(i in 1:38) term_cor_arriba[i] <- cor(tacos$rating[arriba], TD[arriba, i])
names(term_cor_arriba) <- colnames(TD)
round(sort(term_cor_arriba), 3)
```

```
##        ask       give      order        bad      table     waiter
##     -0.220     -0.211     -0.188     -0.171     -0.168     -0.152
##       hour   cocktail restaurant       time     people    service
##     -0.098     -0.095     -0.049     -0.048     -0.032     -0.015
##     dinner      chips       fish       food      drink      salsa
##     -0.006      0.004      0.016      0.022      0.024      0.029
##      place      small      staff       menu      night        bar
##      0.030      0.032      0.038      0.039      0.056      0.058
##       taco       guac    chicken       make  margarita      fresh
##      0.059      0.062      0.065      0.086      0.090      0.101
##       nice     friend    mexican  delicious       good       love
##      0.112      0.135      0.178      0.214      0.255      0.267
##      music      great
##      0.283      0.356
```

I turn to Donna next. Tacos and cocktails are strong here. This is interesting, since Donna is on top of the ranking. The music is no longer relevant.

```
donna <- tacos$restaurant == "Donna"
term_cor_donna <- NULL
for(i in 1:38) term_cor_donna[i] <- cor(tacos$rating[donna], TD[donna, i])
names(term_cor_donna) <- colnames(TD)
round(sort(term_cor_donna), 3)
```

```
##        ask     waiter        bad      place    service       food
##     -0.340     -0.312     -0.276     -0.192     -0.158     -0.139
##     people     dinner      table     friend      order       good
##     -0.134     -0.120     -0.117     -0.110     -0.106     -0.096
##       give       hour    chicken       fish       menu       make
##     -0.078     -0.072     -0.041     -0.041     -0.027     -0.010
##       nice       time      staff      music restaurant  margarita
##     -0.007     -0.004      0.000      0.005      0.006      0.006
##      salsa      small      night        bar      drink       guac
##      0.010      0.018      0.034      0.042      0.050      0.055
##      chips      fresh    mexican  delicious      great   cocktail
##      0.060      0.060      0.092      0.126      0.185      0.188
##       taco       love
##      0.253      0.263
```

Now comes Dylan Murphy's, whose correlation vector is shorter (33 terms). The R output also comes here with warnings about zero standard deviation, not printed in this document. You may remember that the formula of the correlation has the standard deviations in the denominator, so correlation does not make sense if one of the variables has zero standard deviation, which happens when it is constant. This is the case for some of the term document matrix columns if we restrict the data to this restaurant. Since these columns are dummies for the occurrence of one word, this happens when this particular word does not appear in any of the reviews. This is more likely to happen when the number of reviews is low. The five missing words for Dylan Murphy's are 'mexican, 'restaurant', 'guac', 'salsa' and 'margarita'.

This can be explained. When the data were collected, Dylan Murphy's was tagged with 'Tacos' and 'Irish Pub'. It is, indeed, an Irish Pub serving non-Irish food, including tacos. It is nowadays tagged as 'Irish Pub', 'Breakfast & Brunch' and 'Burgers'. It was probably a mistake to include it in this study.

```
dylan <- tacos$restaurant == "Dylan Murphy's"
term_cor_dylan <- NULL
for(i in 1:38) term_cor_dylan[i] <- cor(tacos$rating[dylan], TD[dylan, i])
names(term_cor_dylan) <- colnames(TD)
round(sort(term_cor_dylan), 3)
```

```
##       ask       bad    waiter  cocktail     drink   service     place
##    -0.345    -0.292    -0.268    -0.223    -0.173    -0.161    -0.131
##    friend    people    dinner     table      menu     order      hour
##    -0.113    -0.095    -0.090    -0.063    -0.033    -0.026     0.008
##     fresh      fish     chips      food   chicken     night     small
##     0.017     0.017     0.024     0.029     0.045     0.054     0.074
## delicious      give     music      time      make      taco      nice
##     0.076     0.076     0.093     0.114     0.117     0.122     0.160
##     staff      good      love       bar     great
##     0.184     0.233     0.242     0.266     0.300
```

The cocktails and drinks fall to the bottom, but the bar is on top (interesting). Also up is staff, which is, overall, a neutral issue. Some terms (2) are also are missing in La Esquina, which has margaritas and drinks on opposite sides, and fish as the most relevant food issue.

```
esquina <- tacos$restaurant == "La Esquina"
term_cor_esquina <- NULL
for(i in 1:38) term_cor_esquina[i] <- cor(tacos$rating[esquina], TD[esquina, i])
names(term_cor_esquina) <- colnames(TD)
round(sort(term_cor_esquina), 3)
```

```
##        ask       hour      order restaurant       time  margarita
##     -0.477     -0.416     -0.212     -0.194     -0.147     -0.116
##       love      table       food       good      salsa     friend
##     -0.082     -0.066     -0.052     -0.044     -0.018     -0.018
##       guac    mexican      small     people      staff        bar
##      0.002      0.013      0.017      0.033      0.033      0.047
##       give        bad    service      place      chips     dinner
##      0.047      0.047      0.052      0.055      0.059      0.059
##    chicken       make       taco       menu      music       fish
##      0.068      0.077      0.112      0.135      0.140      0.142
##      fresh       nice      drink      night      great  delicious
##      0.198      0.201      0.202      0.202      0.212      0.345
```

Finally, in Villalobos, with one term missing, guacamole and chips are on opposite sides, and drinks do not seem to matter. What is common in these analyses? The occurrence of the words related to the service, such as ask, table or order. This seems to be a common source of dissatisfaction in the tacos segment. So, if we want to start a business there, we have identified an opportunity.

```
lobos <- tacos$restaurant == "Villalobos"
term_cor_lobos <- NULL
for(i in 1:38) term_cor_lobos[i] <- cor(tacos$rating[lobos], TD[lobos, i])
names(term_cor_lobos) <- colnames(TD)
round(sort(term_cor_lobos), 3)
```

```
##      table     waiter        ask      order        bad     people
##     -0.281     -0.281     -0.253     -0.238     -0.221     -0.202
##      chips       hour       give restaurant       good      place
##     -0.196     -0.195     -0.193     -0.156     -0.137     -0.136
##       time       food      small    service      salsa       make
##     -0.123     -0.114     -0.113     -0.092     -0.080     -0.075
##    chicken     dinner      staff       menu      night       nice
##     -0.063     -0.052     -0.039     -0.035     -0.030     -0.023
##    mexican       fish      drink       taco        bar  margarita
##     -0.017     -0.001      0.013      0.027      0.042      0.082
##      music       guac     friend      fresh       love      great
##      0.097      0.100      0.109      0.123      0.207      0.277
##  delicious
##      0.327
```

References

1. JEF Friedl (2007), *Mastering Regular Expressions*, O'Reilly.
2. K Jarmul & R Lawson (2017), *Python Web Scraping*, Packt.
3. JK Korpela (2006), *Unicode Explained*, O'Reilly.
4. S Munzert, C Ruba, P Meissner & D Nyhuis (2015), *Automated Data Collection with R*, Wiley.
5. H Wickham & G Grolemund (2016), *R for Data Science*, O'Reilly.