Post 2: Text Mining with tm and wordcloud

Ashley Chien

November 20, 2017

1) Introduction

Language has always interested me. Although I am not much of a writer, I really enjoy reading, so I thought it would be fun to do my post on text analysis. This is how I came upon the R packages tm and wordcloud First, let's get a little introduction of what these packages are used for, which is text mining.

What is text mining?

Text mining is just a fancy name for text analysis. Basically, you find information from text through discovering patterns after structuring the data.

Text mining

The first step is to actually read in the text data. Let's see how we can do that in the next section.

First, let's load in the tm and wordcloud packages! Make sure you have installed these packages first (use `install.packages("tm")`)

```
library(tm)
```

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

```
library(wordcloud)
```

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

2) The Data

Reproducibility Information

I'll start off by talking about how to get these results in your own computer. Here are all of the relevant details:

- R version 3.4.2 (2017-09-28) – "Short Summer"
- tm version 0.7.2
- wordcloud version 2.5
- Data is a subset of the rows from this website: https://www.reddit.com/r/datasets/comments/1uyd0t/200000_jeopardy_questions_in_a_json_file/
- the csv file I am using can be downloaded from my github repo (if it weren't private)
- use a random seed of 123 when creating the wordcloud

In this post, we are going to be working with data from past Jeopardy! questions because I am a huge fan of the show Jeopardy! Let's read the file in from my github repo. This data can be downloaded from my repo, if it were public.

Jeopardy

```
# Note that we are setting stringsAsFactors to be FALSE because
# we don't want this variable to be a factor

jeopardy = read.csv("data/jeopardy.csv", stringsAsFactors = FALSE)
```

Let's just take out the vector of the questions to manipulate in this post.

```
# Extract just the question column from the data frame
questions = jeopardy$Question
```

Now we have a vector of 29 Jeopardy! questions. What do we next? We need to start analyzing it!

3) The Source and the Corpus

There are many ways text data may be stored. For example, it may be stored in a vector or it may be stored in a DataFrame. In order to read this data in and start messing with it, there is a general order of steps that we need to take. We will walk through these steps one by one.

3.1. Source

First, we pass our data into a `Source` object. There are several different kinds of `Source` objects. For example, there is a `VectorSource` and a `DataframeSource`. It is pretty self explanatory that you pass a vector into a `VectorSource` and a dataframe into a `DataframeSource`.

Let's do that:

```
# pass in the questions vector
j_src = VectorSource(questions)
```

We're still not ready to analyze our text yet. We need to pass this source into yet another data structure.

3.2 Corpus

The main data structure from the tm library is the `Corpus`. A Corpus is a collection of text documents. There are two main kinds of corpora: the VCorpus and the PCorpus. The VCorpus stands for Volatile Corpus, while the PCorpus is a Permanent Corpus. A Permanent Corpus's documents are physically stored (usually on a database).

We are going to be working with the VCorpus in this post.

```
# pass in the VectorSource
j_corpus = VCorpus(j_src)
```

4) Exploring our Corpus

Now that we have our corpus, let's examine it closely. If we just look at the variable:

```
j_corpus
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 29
```

It doesn't give us much useful information, just the number of documents it has (29). How do we get the documents out?

We need to use double brackets to get information about a certain document out of the corpus. Say we wanted to see information about the first document. We would use `[[1]]`.

```
# Note the double brackets!
j_corpus[[1]]
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 96
```

This only shows us the metadata of the first document. How do we get the content out? We just index for the 1st thing within this!

```
j_corpus[[1]][1]
```

```
## $content
## [1] "For the last 8 years of his life, Galileo was under house arrest for espousing this man's theory"
```

We did it! Do you know the answer to this Jeopardy! question? Hint: it starts with the letter “C”.

5) Cleaning the Data

Text data is more complicated than something like numbers because text data is very non-uniform. It includes many different kinds of characters (maybe ^&@#*!), upper-cased and lower-cased letters, and other oddities. Thus, an important step in data mining is to clean the data. Let’s explore how to do this.

There are a couple of operations we will cover that could be useful on text data.

5.1 tolower

Say we wanted to turn all letters to lower case. We could do this with a for loop, iterating through all of the characters, and changing them one by one if they are upper case to lower case, but that’s too much work! Instead, there is a handy-dandy tolower function. Let’s see it in action:

```
# part of base R
tolower("wEiRd")
```

```
## [1] "weird"
```

Cool, our data seems more uniform now!

5.2 removeNumbers

Sometimes, we just don’t want numbers in our text. We only care about the words! removeNumbers can handle this for us.

```
removeNumbers("I don't want this 3 here")
```

```
## [1] "I don't want this  here"
```

5.3 removePunctuation

Similarly, we can remove punctuation from text easily.

```
removePunctuation("What? Is that true?!")
```

```
## [1] "What Is that true"
```

5.4 removeWords

Let’s say some words are super unnecessary for our analysis. Words like “the”, “and”, and “a” appear in every text, so they are not useful for our analysis. Thus, we should remove them from the text. The way we can do this is with the `removeWords` function. The `removeWords` function takes in a String and a list of words to remove. Let’s see an example:

```
# list of words we want to remove
toRemove <- c("dog", "bird")

# pass in String and toRemove character vector
removeWords("cat dog bird", toRemove)
```

```
## [1] "cat  "
```

5.5 tm_map

We've seen a few String manipulation functions, but we don't want to just apply these functions to one String here or there in text mining. We want to apply these functions to all of our documents in our Corpus! We can do this easily with the `tm_map` function. The `tm_map` function takes in two things: the first argument is the Corpus. The second is the operation that we want to perform (`removePunctuation` for example).

A note on the second thing we pass in: if the function is *not* from the tm package, we need to wrap it in `content_transformer` (`content_transformer(function_name)`).

Out of the list of the four functions we looked at earlier, here are a list of things in the tm package:

- removePunctuation
- removeWords
- removeNumbers

Here a list of things in other packages:

- tolower (tolower is from base R)

Let's use some of the functions we have learned! Remember that the corpus that we're working with is called `j_corpus`

```
# Applying removePunctuation to our j_corpus, saving it in new_corpus
new_corpus <- tm_map(j_corpus, removePunctuation)

# Old corpus
print(j_corpus[[1]][1])
```

```
## $content
## [1] "For the last 8 years of his life, Galileo was under house arrest for espousing this man's theory"
```

```
# New corpus
print(new_corpus[[1]][1])
```

```
## $content
## [1] "For the last 8 years of his life Galileo was under house arrest for espousing this mans theory"
```

Tada! The apostrophe and the comma are both gone! Amazing!

Let's try using `tolower`. We have to wrap it in a `content_transformer` because it is not from the `tm` package.

```
# Note the wrapping of tolower!
new_corpus <- tm_map(j_corpus, content_transformer(tolower))

# Old corpus
print(j_corpus[[2]][1])
```

```
## $content
## [1] "No. 2: 1912 Olympian; football star at Carlisle Indian School; 6 MLB seasons with the Reds, Giants & Brave
s"
```

```
# New corpus
print(new_corpus[[2]][1])
```

```
## $content
## [1] "no. 2: 1912 olympian; football star at carlisle indian school; 6 mlb seasons with the reds, giants & brave
s"
```

Tada! It's lower case now!

From our Jeopardy! corpus, let's convert it to all lowercase, and then remove all common English stopwords (stopwords are words that are used really frequently like “the” and “and”).

```
j_corpus <- tm_map(j_corpus, content_transformer(tolower))
j_corpus <- tm_map(j_corpus, removeWords, stopwords("english"))
```

6) Making a word cloud

Now that we have our data all cleaned up, let's visualize it. We are going to visualize it with a word cloud, which is an image of words, where the size of the word in the image corresponds to how important or how frequent the word appears in the text.

Creating a word cloud is easy with wordcloud!

First, we convert our Corpus into a `TermDocumentMatrix`. A `TermDocumentMatrix` is a matrix where the column names are the words in the documents and the row names are the document names. Within the matrix are values of counts of the frequency of words.

```
tdm <- TermDocumentMatrix(j_corpus)

tdm
```

```
## <<TermDocumentMatrix (terms: 243, documents: 29)>>
## Non-/sparse entries: 257/6790
## Sparsity           : 96%
## Maximal term length: 14
## Weighting          : term frequency (tf)
```

As you might expect, this matrix is pretty sparse (many counts are 0 because many words only appear in one question and none of the other questions).

To create a word cloud, we follow these steps:

1. Change the `TermDocumentMatrix` into an R matrix by using the `as.matrix` function.
2. Gather the counts of these words using the function `rowSums`, and then sort so that we can get counts of the frequencies of the words in decreasing order.
3. Create a data frame with the names and the frequencies.

```
# Convert to an R matrix
m <- as.matrix(tdm)

# Sort by the frequency of the word
freqs <- sort(rowSums(m), decreasing = TRUE)

# One column: the words, second column: frequencies of the words
df <- data.frame(word = names(freqs), freq = freqs)
```

Let's look at a little bit of our data frame using the head function.

```
head(df, 5)
```

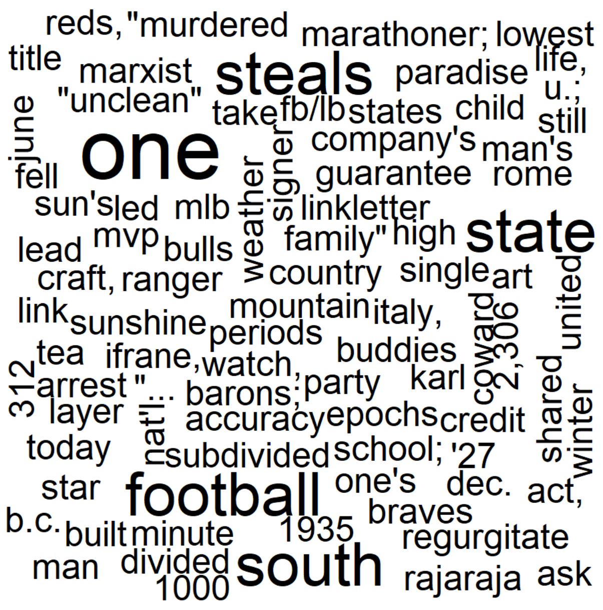
```
##              word freq
## first       first    3
## one           one    3
## began       began    2
## company   company    2
## football football    2
```

Interesting, seems like "first" and "one" are pretty popular (with frequencies of 3).

Let's plot our word cloud now. All we need to specify is the words from our data frame, and the frequency of those words. Let's also throw in that the word has to appear at least once to show up in our word cloud.

To always get the same plot, let's set a random seed of 123.

```
set.seed(123)
suppressWarnings(wordcloud(words = df$word, freq = df$freq, min.freq = 1))
```



Wow, this word cloud doesn't look too great because Jeopardy! questions don't really follow too much of a pattern.. oh well! Making a word cloud is pretty cool.

7) Next Steps and Motivation

After you do all of this data cleaning, the next step is to analyze the data for patterns to extract cool and exciting insights! I'm not going to go into this here – the material is a lot more complicated and confusing, but I'd highly recommend that you check it out for yourself if this interests you.

Take-home message: Lots of cool packages are available in R to tinker with; it just requires some reading of documenation and experimentation to do something fun :)

References:

1. https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf
2. https://www.reddit.com/r/datasets/comments/1uyd0t/200000_jeopardy_questions_in_a_json_file/
3. https://ischool.syr.edu/infospace/wp-content/files/2013/04/text-mining-featured.jpg
4. https://www.datacamp.com/courses/intro-to-text-mining-bag-of-words
5. https://www.rdocumentation.org/packages/tm/versions/0.7-2
6. https://eight2late.wordpress.com/2015/05/27/a-gentle-introduction-to-text-mining-using-r/
7. https://stackoverflow.com/questions/30435054/how-to-show-corpus-text-in-r-tm-package
8. http://www.sthda.com/english/wiki/text-mining-and-word-cloud-fundamentals-in-r-5-simple-steps-you-should-know