Sentiment Analysis

Prateek

6/6/2020

SENTIMENT ANALYSIS

This is a project to understand the basics of R language. I referred to the various websites particulary Datacamp and Data-Flair for getting the most of the codes. From my end I read and understood the working and functioning each of the packages and R functions used in this project.

By end of this project I won't claim I learnt everything about R language, however the self paced learning taught me the art of exploring, understanding and finding my way out when working on R. Although the entire code on Sentiment Analysis is available on many websites, I found most of them failed to give a thorough explanation of:

why a particular function is used?,

Role of a particular step in the end result, and

The various attributes available with a particular function.

In this article we will try to understand the usage of each step of code w.r.t. to the above questions that comes to any first time R language learner.

Let's begin

First step is to load these 4 packages to the global environment.

```
library(tidytext)
library(janeaustenr)
library(stringr)
library(dplyr)
```

`tidytext` is an important part of this project. The package and its associated functions lets us handle text data. Remember that in tidytext package the table of tidy data is stored in a token format and is in a format of **one token per row**. Token usually means a one single word but it can also be a sentence, paragraph or even one complete chapter.

In our project, we aim to analyse each word in the Jane Austen book and rate those words on a sentiment scale. Hence, we would be tokenizing the tidydata to one word/one token in our case.

`janeaustenr` loads a package containing each word published in 6 books written by Jane Austen. We will be doing sentiment analysis of words in one of the above 6 books.

`stringr` loads a package which lets us use the pipe operator`%>%` in next step and other string functions such as `str_detect`.

`dplyr` is a package that lets us use the `group_by` function.

Tidying Text Data

In next step, we will create a tibble in which the words in `austen_books()` are grouped by order of book, provided with a chapter number against each word depending upon the chapter it appears in, and are

tokenized to one word level. In simple terms, we are *tidying* the text data to form `tidy_data` tibble.

```
tidy_data<- austen_books()%>%group_by(book)%>% mutate(linenumber=row_number(),
            chapter=cumsum(str_detect(text,regex("^chapter [\\divxcl]",
            ignore_case = T))))%>%ungroup()%>%unnest_tokens(word,text)
```

The detailed explanation for the code is:

Pipe Operator(`%>%`) - The pipe operator takes the output of one statement and makes it the input of next statement. This is somewhat similar to chaining. e.g. f(g(h(x))) can be piped as `x%>%h()%>%g()%>%f()`. Pipe operator hence allows the chaining, without needing intermediate variables to store the value. Also, it eliminates the use of lots of parenthesis making it *readable* in a code chunk and presents the chained code in a *logically sequenced* format.

`group_by()`- The function groups the austen_books() data frame by order of the books present in data. This is not a visual change i.e. group_by function won't make a readable change to the structure of data.However it changes how th data acts with other dplyr verbs (functions).

`mutate()` - This is an important tool in dplyr package. It adds a new variable to data frame. We will understand the usage with a simple example.

Take the data frame airquality.

```
airquality
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

With mutate function we can add a new variable *temperature/wind* to the data frame. Infact we can add any new variable to an existing dataframe using this function.

```
mutate(airquality, "temp/wind" = Temp/Wind)
```

```
##   Ozone Solar.R Wind Temp Month Day temp/wind
## 1    41     190  7.4   67     5   1  9.054054
## 2    36     118  8.0   72     5   2  9.000000
## 3    12     149 12.6   74     5   3  5.873016
## 4    18     313 11.5   62     5   4  5.391304
## 5    NA      NA 14.3   56     5   5  3.916084
## 6    28      NA 14.9   66     5   6  4.429530
```

We can see the mutate function added a new variable `temp/wind` to the existing data frame. This makes the mutate function very useful in data mining and processing.

In our project of sentiment analysis, after grouping the austen_book dataframe in order of books we are adding two new variables (`linenumber` and `chapter`). The line number is the row number corresponding to each line of text. This is important because later we will be tokenizing each word of the data frame and to keep a track of which row it belonged to in the actual data frame, we are assigning the row number from original data frame as the line numbers.

Also we need to add another variable that would tell us the chapter number each word belongs to. For this, as told, we are using mutate function but with an added set of codes. `chapter=cumsum(str_detect(text,regex("^chapter [\\divxcl]", ignore_case=T)))`

`\\divxcl` - \d includes all digits and vxcl will add any roman numerals, if the chapter numbers are marked in roman format, to the count.

`str_detect(text,regex())` - is a regular expression. Regular expressions are concise,flexible tool for describing patterns in strings.

We will learn this with an example:

```
bananas <- c("banana", "Banana", "BANANA")
str_detect(bananas,regex("banana"))
```

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

As you can see, the str_detect and regex tool helps us to extract a part of text from a token word and this is useful in identifying the "CHAPTER X" format present in the text variable of the dataframe. The `ignore_case` attribute of str_detect function lets us ignore any case where "chapter" is in small or caps.

Once we identify a Chapter number in the dataframe we need to add any subsequent words appearing in the dataframe to that chapter number till we encounter the next chapter number.

`cumsum()`- We have identified the Chapter number we need to make a cumulative sum of the words that appear in one chapter. For this the `cumsum()` function is used. By the end of this chunk of code we ungroup and unnest the tokens to bring the dataframe to its original format. The new dataframe `tidy_data` is now in a tidy format with each row having one word, the corresponding line number, and the corresponding chapter it appears in.

This is how our new dataset looks like after *tidying*:

```
tidy_data
```

```
## # A tibble: 725,055 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Sense & Sensibility          1       0 sense
##  2 Sense & Sensibility          1       0 and
##  3 Sense & Sensibility          1       0 sensibility
##  4 Sense & Sensibility          3       0 by
##  5 Sense & Sensibility          3       0 jane
##  6 Sense & Sensibility          3       0 austen
##  7 Sense & Sensibility          5       0 1811
##  8 Sense & Sensibility         10       1 chapter
##  9 Sense & Sensibility         10       1 1
## 10 Sense & Sensibility         13       1 the
## # ... with 725,045 more rows
```

We are now ready to use sentiment analysis tools on our tidy data.

Sentiment Lexicons

To start with Sentiment analysis, we should understand the usage and meaning of different sentiment lexicons.

Sentiment Lexicons are different ways to measure text sentiments. There are 3 Sentiment Lexicons: `AFINN` `bing`, and `loughran`

All the 3 models are based on unigrams. Unigrams are sequence of 1 word only. If you recall, we had tokenized the words from JaneAusten's book to one word level. It was for this reason. The AFINN lexicon measures each word sentiment on a scale of -5 to +5. -5 being the most negative sentiment and +5 being the most positive.

The bing lexicon model uses a binary type format of negative and positive. So a word is rated either negative or positive. Negative words depict negative sentiment and positive words depict positive sentiment.

Loughran lexicon is created for use with financial documents. This lexicon labels words with six possible sentiments important in financial contexts: “negative”, “positive”, “litigious”, “uncertainty”, “constraining”, or “superfluous”.

In our project, we are using bing lexicon model. This is how the `bing` lexicon model looks like.

```
get_sentiments("bing")
```

```
## # A tibble: 6,786 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faces     negative
##  2 abnormal    negative
##  3 abolish     negative
##  4 abominable  negative
##  5 abominably  negative
##  6 abominate   negative
##  7 abomination negative
##  8 abort       negative
##  9 aborted     negative
## 10 aborts      negative
## # ... with 6,776 more rows
```

Finding Positive Words in the Book “Sense & Sensibility”

In this step, our aim is to filter out all the positive words that appear in the book Sense and Sensibility. This is an easy task, given that we have already prepared a tidy_data format.

To proceed with, first we filter out all the positive words from the bing lexicon and assign the filtered positive words to a dataframe `positive_senti`.

```
positive_senti<- get_sentiments("bing")%>%filter(sentiment=="positive")
```

So we now have a tibble of all positive words from bing lexicon. Let’s see how this looks like. Compare this with the bing lexicon we earlier called upon by the code `get_sentiments("bing")`.

```
## # A tibble: 2,005 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 abound      positive
##  2 abounds     positive
##  3 abundance   positive
##  4 abundant    positive
##  5 accessable  positive
##  6 accessible  positive
##  7 acclaim     positive
##  8 acclaimed   positive
##  9 acclamation positive
## 10 accolade    positive
## # ... with 1,995 more rows
```

Next we will use `positive_senti` to filter out the positive words from the book Sense & Sensibility. For this we will be using the `tidy_data` tibble which we earlier created.

```
positivetidy_data<- tidy_data%>% filter(book=="Sense & Sensibility")%>%
  semi_join(positive_senti, by = "word")%>%count(word,sort = T)
```

In the above chunk of codes, we first filter the rows from tidy_data tibble which have Sense & Sensibility in the book variable. Next, we use semi_join function to find those rows of Sense & Sensibility which have their words match with the positive words of bing lexicon(positive_senti)

semi_join - semi_join returns rows of first table where it can find a match in the second table. In our case, we are using the table of tidy_data (filtered with the book Sense & Sensibility) and finding a match of all the words in this filtered tidy_data that appears in the positive_senti table.

count - this function counts the number of times each positive words from Sense & Sensibility appears in the bing lexicon. The sort=TRUE attribute sorts the result showing the largest number at the top.

So after this operation, we can now see the number of times positive words appear in the book Sense & Sensibility. Mind it, that there may be words that are not in the bing lexicon. Those words are not rated by the bing lexicon and won't appear in our result.

```
## # A tibble: 593 x 2
##    word          n
##    <chr>     <int>
##  1 well        240
##  2 good        177
##  3 great       149
##  4 enough      103
##  5 happy       100
##  6 like         83
##  7 affection    79
##  8 better       78
##  9 love         77
## 10 pleasure     67
## # ... with 583 more rows
```

Interestingly, we can now find that the positive word "well" appears 240 times in the book Sense & Sensibiltiy. On a similar note, we can other positive words and the frequency of their appearance. Remember, this data only has positive words and not the negative words.

In the next step, we will segregate our data into separate columns of positive and negative sentiments. We will then calculate the difference between positive and negative sentiment.However, now we will divide our dataframe into equal parts of 80 words each and find the total number of positive and negative words in those 80 words and then we will continue to do so in next batch of 80 words and so on. Now the question arises, why in a batch of 80 words specifically? Wait for the next step for the answer when we will be plotting ggplot for this data. We will discuss that in detail.

So for now first load the tidyr package. tidyr package lets us use the spread function. We filter all the words from the book Sense & Sensibility from tidy_data. Next we use inner_join function to filter those words which appear in bing sentiment lexicon and also put up the sentiment (positive or negative) against the filtered words.

```
library(tidyr)
bing <- get_sentiments("bing")
Senseandsensibility_sentiment<- tidy_data%>% filter(book=="Sense & Sensibility")%>%        inner_join(bi
                    index = linenumber %/% 80, sentiment) %>%
spread(sentiment, n, fill = 0) %>%mutate(sentiment = positive - negative)
```

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

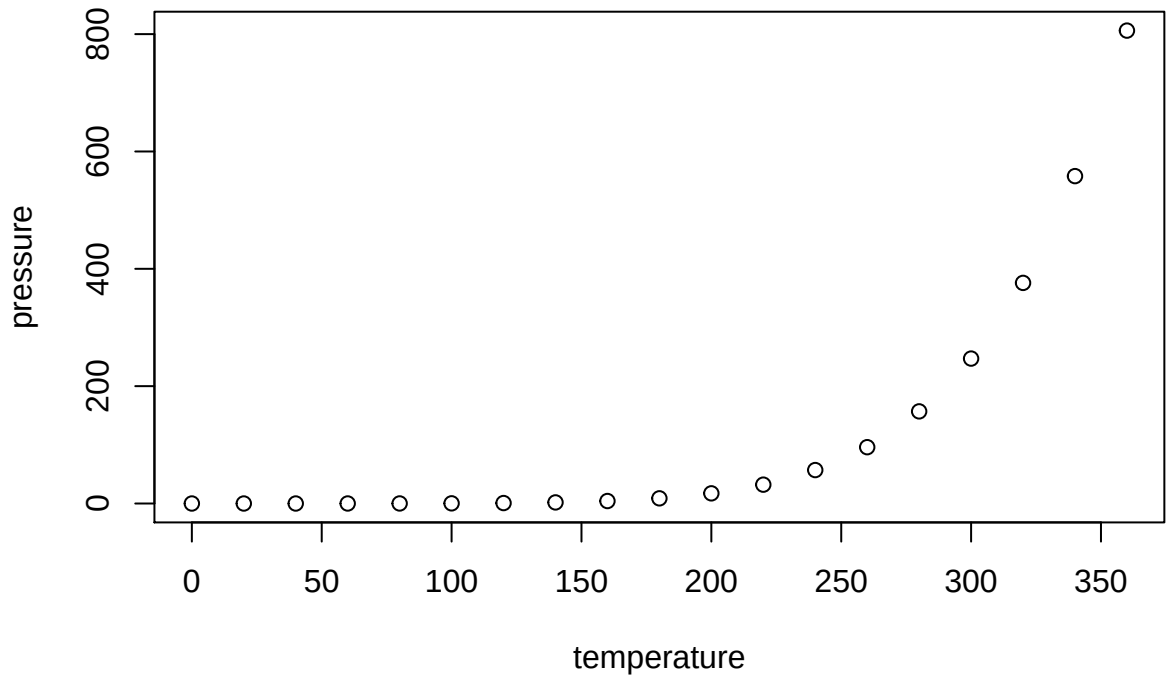
```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
```

```
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.