Text Analytics Notes - Sentiment Analysis in R

Promothesh Chatterjee*

Sentiment Analysis in R

Some of the R packages that are popularly used in sentiment analysis are • Syuzhet • Rsentiment • SentimentR • SentimentAnalysis • Tidytext • Sentometrics • Quanteda • CleanNLP

All of these packages follow the bag-of-words model, using only the words ignoring order, the role of syntax and grammar. These tokenized words are compared against the dictionary words which have been already been tagged as positive or negative (or even as a list of emotions) and are often given a score that reflects the degree of intensity (of positiveness or negativeness). Overall sentiment can be measured with a numeric score that uses number of matches as well as the intensity measures of the sentiment associated with those words.

I will mostly use the packages Quanteda, SentimentR and CleanNLP. I will not use Tidytext package (which is probably the most popular R package for sentiment analysis online, you checkout the nuances in this datacamp course here: https://learn.datacamp.com/courses/sentiment-analysis-in-r-the-tidy-way)

```
# Load up the libraries and dataset
library(tidyverse)
library(sentimentr)
library(caret)
library(quanteda)
library(broom)

# Load up the .CSV data and explore in RStudio.
hotel_raw <- read_csv("C:/Users/u0474728/Dropbox/Utah Department Stuff/Teaching/Text Analysis/Summer 20
set.seed(1234)# we set seed to replicate our results.
hotel_raw<-hotel_raw[sample(nrow(hotel_raw), 5000), ]# take a small sample
corp_hotel <- corpus(hotel_raw, text_field = "Description")# Create corpus
sample<-corpus_sample(corp_hotel, size = 20)# you can sample a corpus too!
```

Using a Dictionary for Sentiment Analysis

Let's first see how we can create a dictionary. Using Quanteda, creating a dictionary is very easy and then we can uncover the sentiments of three sentences very easily. Once we have dictionary, it is very easy to test for number of positive and negative sentiment words in each document by creating a DFM and specifying a dictionary.

```
library(quanteda)
test.lexicon <- dictionary(list(positive.terms = c("happy", "joy", "light"),
                                negative.terms = c("sad", "angry", "darkness")))

testtext<-c("I am happy and confident that the paper will be accepted",
            "Of course, no one can be 100% sure but I am hopeful",
            "In case, it is rejected, I will be sad and angry, we will submit it to another journal")
```

```r
dfm_sentiment1 <- testtext %>% tokens() %>% dfm() %>% dfm_lookup(test.lexicon)
dfm_sentiment1
```

```
## Document-feature matrix of: 3 documents, 2 features (66.67% sparse) and 0 docvars.
##        features
## docs    positive.terms negative.terms
##   text1              1              0
##   text2              0              0
##   text3              0              2
```

This was a trivial example but we can do the same for a larger dataset too. Let's use the hotel reviews dataset for illustration. While tidytext package has Bing dictionary built in it (the name is not due to Microsoft's Bing but is named after Bing Liu the creator of the dictionary), we can easily build it using quanteda's dictionary function. I have downloaded two dictionary files (positive-words.txt and negative-words.txt) from Bing Liu's website (free for academic purposes), we just need to read them in and create the dictionary.

```r
positive_words_bing <- scan("C:/Users/u0474728/Dropbox/Utah Department Stuff/Teaching/Text Analysis/Sum
negative_words_bing <- scan("C:/Users/u0474728/Dropbox/Utah Department Stuff/Teaching/Text Analysis/Sum
sentiment_bing <- dictionary(list(positive = positive_words_bing, negative = negative_words_bing))
```

While reading the two files,we skip the first 35 lines as they contain meta information about the lexicon which is not needed at this point. Quiet=T prevents the output of a status message. Having read in the two files, now we use dictionary function to create the lexicon and test for sentiment.

```r
dfm_sentiment <- corp_hotel %>% tokens() %>% dfm %>% dfm_lookup(sentiment_bing)
dfm_sentiment
```

```
## Document-feature matrix of: 5,000 documents, 2 features (13.31% sparse) and 4 docvars.
##        features
## docs    positive negative
##   text1       12        0
##   text2       10        3
##   text3        1        1
##   text4        7        3
##   text5       10        1
##   text6        6        0
## [ reached max_ndoc ... 4,994 more documents ]
```

```r
dfm_sentiment_df<-convert(dfm_sentiment, to ='data.frame')
dfm_sentiment_df$net<-(dfm_sentiment_df$positive)-(dfm_sentiment_df$negative)
summary(dfm_sentiment_df)# document level summary
```

```
##     doc_id             positive         negative           net
```

```
##  Length:5000        Min.   : 0.000   Min.   :  0.000   Min.   :-56.000
##  Class :character   1st Qu.: 5.000   1st Qu.:  0.000   1st Qu.:  3.000
##  Mode  :character   Median : 8.000   Median :  2.000   Median :  6.000
##                     Mean   : 9.233   Mean   :  3.054   Mean   :  6.179
##                     3rd Qu.:12.000   3rd Qu.:  4.000   3rd Qu.:  9.000
##                     Max.   :77.000   Max.   :102.000   Max.   : 51.000
```

Most R packages tend to come with 3 or 4 dictionaries but you can get a lot more dictionaries (including the one we just created) from the library quanteda.dictionaries (you just need to install it from Github).

```r
install.packages("remotes")
remotes::install_github("kbenoit/quanteda.dictionaries")
```

Let's take the example of dictionary MFD (for details,see https://rdrr.io/github/kbenoit/quanteda.dictionaries/man/data_dictionary_MFD.html) and use liwcalike() function from the quanteda.dictionaries package to analyze text corpora using existing or custom dictionaries.

```r
library("quanteda.dictionaries")
output_mfd <- quanteda.dictionaries::liwcalike(corp_hotel,
                        dictionary = data_dictionary_MFD)
head(output_mfd)
```

```
##   docname Segment      WPS  WC Sixltr  Dic care.virtue care.vice
## 1   text1       1 10.91667 131   6.11 0.00        0.00         0
## 2   text2       2 29.22222 263  14.83 1.52        0.38         0
## 3   text3       3  8.40000  42  11.90 2.38        0.00         0
## 4   text4       4 46.25000 185   7.03 1.08        0.54         0
## 5   text5       5 23.75000 190  11.05 1.05        1.05         0
## 6   text6       6 17.50000  35   5.71 2.86        0.00         0
##   fairness.virtue fairness.vice loyalty.virtue loyalty.vice authority.virtue
## 1               0             0           0.00            0             0.00
## 2               0             0           0.38            0             0.38
## 3               0             0           2.38            0             0.00
## 4               0             0           0.00            0             0.00
## 5               0             0           0.00            0             0.00
## 6               0             0           0.00            0             0.00
##   authority.vice sanctity.virtue sanctity.vice AllPunc Period Comma Colon SemiC
## 1              0            0.00             0   17.56   9.16  5.34     0     0
## 2              0            0.38             0   13.69   3.04  2.28     0     0
## 3              0            0.00             0   11.90  11.90  0.00     0     0
## 4              0            0.54             0    6.49   2.16  2.70     0     0
## 5              0            0.00             0   50.53   4.21  4.74     0     0
## 6              0            2.86             0   14.29   5.71  8.57     0     0
```

```
##   QMark Exclam Dash Quote Apostro Parenth OtherP
## 1     0   0.00 0.00  3.05    3.05    0.00  17.56
## 2     0   0.38 2.66  1.14    1.14    1.90   7.22
## 3     0   0.00 0.00  0.00    0.00    0.00  11.90
## 4     0   1.08 0.00  0.54    0.54    0.00   6.49
## 5     0   0.00 5.79 33.68    0.00    1.05  42.63
## 6     0   0.00 0.00  0.00    0.00    0.00  14.29
```

There is a new package in the quanteda suite of packages called quanteda.sentiment. You can explore it here: https://rdrr.io/github/quanteda/quanteda.sentiment/f/vignettes/sentiment_analysis.Rmd

We can also plot the sentiments using GGplot2. Its easier to plot with proportions than actually frequency, so we will first use quanteda to convert the frequencies into proportions and then plot it.

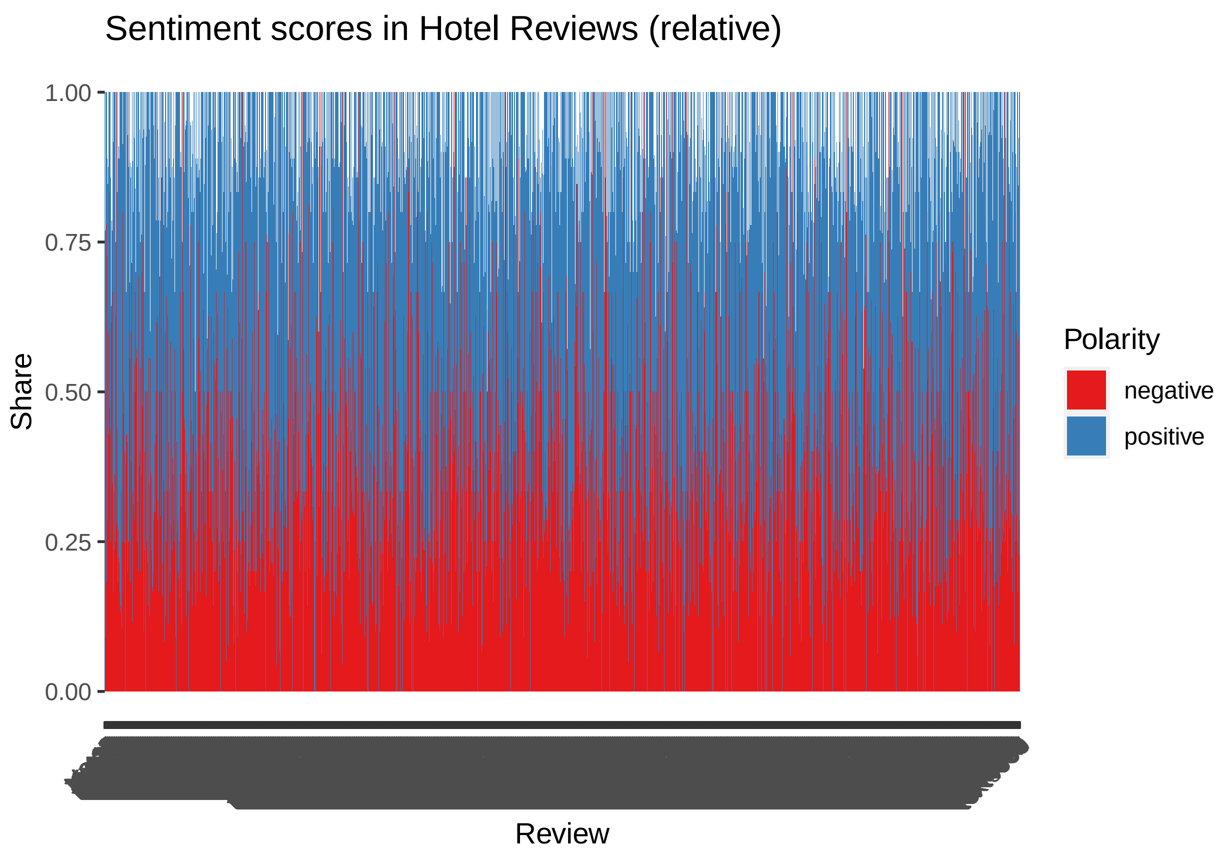
```r
# Proportions instead of numbers

dfm_sentiment_prop <- dfm_weight(dfm_sentiment, scheme = "prop")
dfm_sentiment_prop
```

```
## Document-feature matrix of: 5,000 documents, 2 features (13.31% sparse) and 4 docvars.
##        features
## docs     positive   negative
##   text1 1.0000000 0
##   text2 0.7692308 0.23076923
##   text3 0.5000000 0.50000000
##   text4 0.7000000 0.30000000
##   text5 0.9090909 0.09090909
##   text6 1.0000000 0
## [ reached max_ndoc ... 4,994 more documents ]
```

```r
## Plotting the sentiments
library(tidyverse)
sentiment <- convert(dfm_sentiment_prop, "data.frame") %>%
    gather(positive, negative, key = "Polarity", value = "Share") %>%
    mutate(document = as_factor(doc_id)) %>%
    rename(Review = document)

ggplot(sentiment, aes(Review, Share, fill = Polarity, group = Polarity)) +
    geom_bar(stat='identity', position = position_dodge(), size = 1) +
    scale_fill_brewer(palette = "Set1") +
    theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
    ggtitle("Sentiment scores in Hotel Reviews (relative)")
```



Because there are so many reviews it is difficult to figure out, but majority of the reviews (documents) have both a negative and a positive sentiment, though overall the net sentiment in the corpus seems to be positive. Since we have the sentiment analysis stored both as DFM and dataframe, we can use different tools for visualization and perform the analytical techniques we have seen so far.

Sentence Level Sentiment Analysis

We use the R package sentimentr for sentence level sentiment analysis. Note that in the previous section, we have computed sentiment at the level of documents (or reviews in our case). Using sentimentr we can calculate text polarity sentiment at the sentence level and can also aggregate by rows or grouping variable(s). Most importantly, sentimentr allows us to address negations and other valence shifters. What are valence shifters? Let's look at some examples from sentimentr documentation.

Valence shifters modify the underlying valence of a sentence. For instance, a negator flips the sign of a polarized word (e.g., "I do not like it."). An amplifier (intensifier) increases the impact of a polarized word (e.g., "I really like it."). A de-amplifier (downtoner) reduces the impact of a polarized word (e.g., "I hardly like it."). An adversative conjunction overrules the previous clause containing a polarized word (e.g., "I like it but it's not worth it.").

As per analysis done by Tyler Rinker, across a variety of texts, negators appear approximately 20% of the time a polarized word appears in a sentence. In addition, adversative conjunctions appear with polarized words approximately 10% of the time. Obviously not accounting for these valence shifters will have significant impact on the text model and analysis. Let's run some basic sentiment level analysis.

First thing we need to do is to split the text data into sentences using the get_sentences function and then perform sentiment analysis.

```
mytext<-"I am happy and confident that the paper will be accepted.
Of course, no one can be 100% sure but I am hopeful. In case, it is rejected,
I will be sad and angry, but we will submit it to another journal."

mytext <- get_sentences(mytext)
sentiment(mytext)
```

```
##    element_id sentence_id word_count  sentiment
## 1:          1           1         11  0.7085517
## 2:          1           2         11 -0.3278936
## 3:          1           3         19 -0.5161854
```

We can test for the impact of negator in the sentiment analysis by simply modifying it a little and running the code again.

```
mytext2<-"I am happy and confident that the paper will be accepted.
Of course, no one can be 100% sure but I am hopeful. In case, it is rejected,
I will not be sad and angry, but we will submit it to another journal." # with a negator added
mytext2 <- get_sentences(mytext2)
sentiment(mytext2)
```

```
##    element_id sentence_id word_count  sentiment
## 1:          1           1         11  0.7085517
## 2:          1           2         11 -0.3278936
## 3:          1           3         20  0.0559017
```

To aggregate by grouping variables use sentiment_by using the by argument.

```
out <- with(
    hotel_raw,
    sentiment_by(
        get_sentences(Description), # Reviews are stored in variable Description
        list(User_ID,Device_Used) # grouping variables
    ))
head(out)
```

```
##    User_ID Device_Used word_count        sd ave_sentiment
## 1: id10367      Mobile        267 0.3308392     0.2702683
## 2: id10378      Tablet         70 0.2794307     0.1656761
```

```
## 3: id10382      Mobile         67 0.2070561      0.1956476
## 4: id10385      Tablet         86 0.2871477      0.4573742
## 5: id10398      Tablet         33 0.2512122      0.2497064
## 6: id10407      Tablet        260 0.2840566      0.2015775
```

Sentimentr also allows text highlighting of sentiment line by line with positive/negative sentences highlighted. The highlight function uses a sentiment_by output to produces a highlighted HTML file (positive = green; negative = pink).

```r
library(magrittr)
library(dplyr)
set.seed(234)

hotel_raw %>%
    filter(User_ID %in% sample(unique(User_ID), 4)) %>%
    # %in% operator in R, is used to identify if an element belongs to a vector.
    mutate(review = get_sentences(Description)) %$%
    # The "exposition" pipe operator from magrittr package, %$% exposes the names
    # within the left-hand side object to the right-hand side expression. For
    # instance,
    # iris %>%
    # subset(Sepal.Length > mean(Sepal.Length)) %$%
    # cor(Sepal.Length, Sepal.Width)
    sentiment_by(review, User_ID) %>%
    highlight()
```

Feature Level Sentiment Analysis

For feature level Sentiment Analysis, we first need to run POS tagging on the dataset. As we saw, POS tagging is nothing but assigning various parts-of-speech tags; pretty similar to tagging the text for specific sentiments. Let's use library CleanNLP for this purpose (the output also includes lemmas). We can also use library spacyr in conjunction with quanteda but installing spacyr and accessing it can be very problematic sometimes, so we use the UDPipe backend for CleanNLP.

```r
library(tidyverse)
library(cleanNLP)

cnlp_init_udpipe() # Loading namespace: udpipe which serves backend to cleanNLP

hotel_raw1<-hotel_raw[sample(nrow(hotel_raw), 100), ]# take a small sample as POS
                                                      #tagging is very resource intensive

postag<-hotel_raw1 %>%
cnlp_annotate(text= "Description") # Outcome is list of tokens and documents
```

```
## Processed document 10 of 100
## Processed document 20 of 100
## Processed document 30 of 100
## Processed document 40 of 100
## Processed document 50 of 100
## Processed document 60 of 100
## Processed document 70 of 100
## Processed document 80 of 100
## Processed document 90 of 100
## Processed document 100 of 100
```

```r
head(postag$token,n=10)
```

```
## # A tibble: 10 x 11
##    doc_id   sid tid   token      token_with_ws lemma upos  xpos  feats tid_source
##     <int> <int> <chr> <chr>      <chr>         <chr> <chr> <chr> <chr> <chr>
##  1      1     1 1     The        "The "        the   DET   DT    Defi~ 2
##  2      1     1 2     Alexander  "Alexander "  Alex~ PROPN NNP   Numb~ 0
##  3      1     2 1     Inn        "Inn "        Inn   PROPN NNP   Numb~ 3
##  4      1     2 2     is         "is "         be    AUX   VBZ   Mood~ 3
##  5      1     2 3     perfect    "perfect "    perf~ ADJ   JJ    Degr~ 0
##  6      1     2 4     for        "for "        for   SCONJ IN    <NA>  5
##  7      1     2 5     visiting   "visiting "   visit VERB  VBG   Verb~ 3
##  8      1     2 6     because    "because "    beca~ SCONJ IN    <NA>  17
##  9      1     2 7     all        "all "        all   DET   DT    <NA>  8
## 10      1     2 8     attracti~  "attractions~ attr~ NOUN  NNS   Numb~ 17
## # i 1 more variable: relation <chr>
```

The columns upos and xpos refer to the POS taggers– universal POS (UPOS), treebank-specific POS (XPOS) tags.Let’s understand the Penn Treebank POS terminology.

Tag	Description	Example	Tag	Description	Example	Tag	Description	Examp
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, t*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, w*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, wh*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	$
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	#
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

We can find out the most frequently used adjectives in the dataset by filtering on the part of speech field, group it by lemma, count the rows in each group. We can then sort the output and selecting the top 10 nouns to get a high level summary of the topics of interest within this corpus.

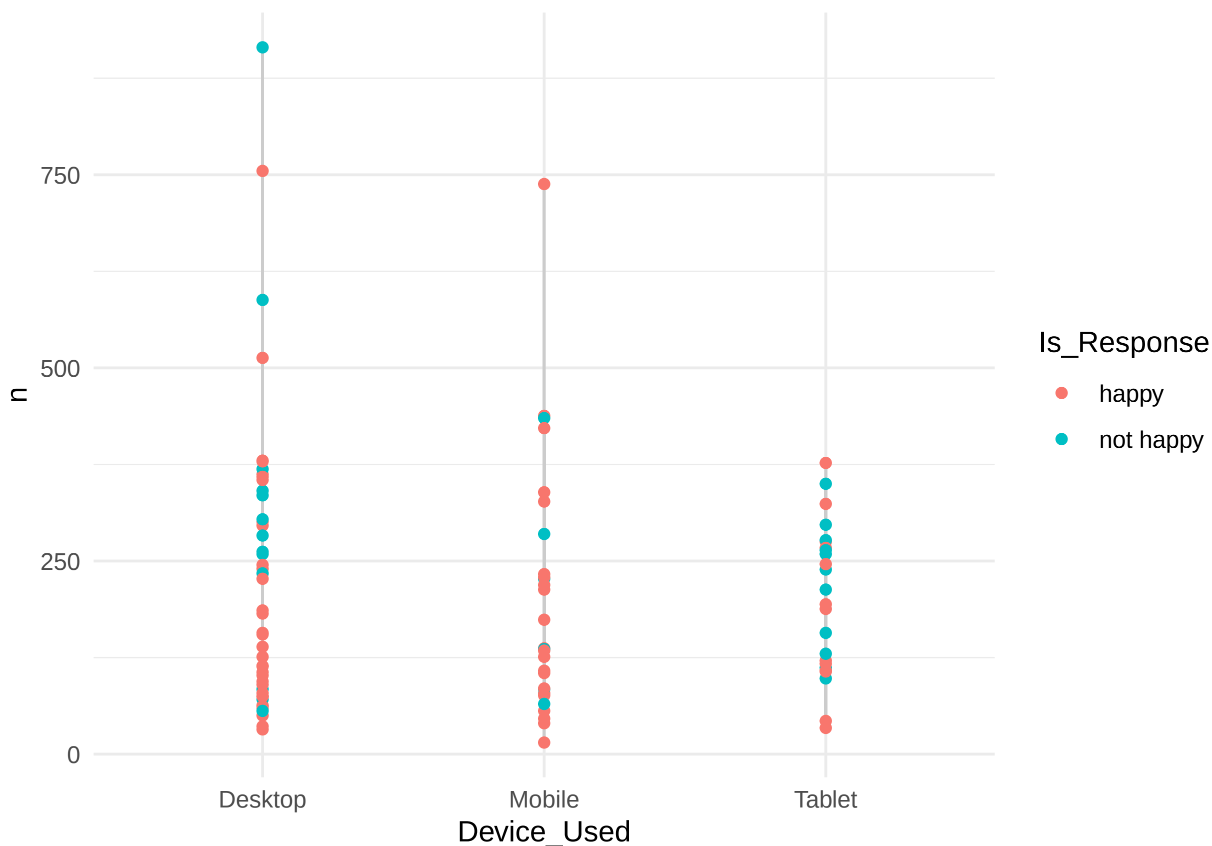
```
postag$token %>%
  filter(xpos == "JJ") %>% # you can play around with different POS
  group_by(lemma) %>%
  summarize(count = n()) %>%
  top_n(n = 10, count) %>%
  arrange(desc(count))
```

```
## # A tibble: 10 x 2
##    lemma       count
##    <chr>       <int>
##  1 great          84
##  2 clean          41
##  3 nice           39
##  4 good           37
##  5 comfortable    32
##  6 friendly       31
##  7 other          30
##  8 close          29
##  9 old            27
## 10 small          27
```

As a possible usage of POS tagger, imagine that you are Coca Cola and you run the above analysis for POS= proper noun. It is probably not a great sign if you end up seeing lots of competitor brands and especially, if the adjectives are negative.

We can join the two outputs postag[["token"]] and postag[["document"]] by using left_join from dplyr; the ggplot2 graph looks nice if time is on the x-axis (we didn't have time as a variable, so I put device used to write review on the x-axis just to illustrate the code)

```
postag$token %>%
  group_by(doc_id) %>%
  summarize(n = n()) %>%
  left_join(postag$document, by="doc_id") %>%
  ggplot(aes(Device_Used, n)) +
    geom_line(color = grey(0.8)) +
    geom_point(aes(color = Is_Response)) +
    geom_smooth(method="loess", formula = y ~ x) +
    theme_minimal()
```



Naive Bayes Classifier for Text

Now that we have looked at using dictionaries for sentiment analysis, let's approach this as a classification task. While we discuss the topic in greater detail in later section, for now as the name suggests, the task is to classify texts based on their inherent characteristics. We will take the example of Binarized Multinomial Naïve Bayes. In Binarized Multinomial Naïve Bayes, the algorithm clips the unique word counts in each document at 1. The basic procedure is to remove all duplicate words from document and then perform Naive Bayes calculation. We will use the library quanteda for this. To fit a "binary multinomial" model, we first convert the dfm to a binary matrix using quanteda function dfm_weight(x, scheme = "boolean").

```
require(quanteda)
require(quanteda.textmodels)
require(caret)

summary(corp_hotel, 5)# let's check the summary of our original corpus
```

```
## Corpus consisting of 5000 documents, showing 5 documents:
##
##   Text Types Tokens Sentences User_ID  Browser_Used Device_Used Is_Response
##  text1    80    131        12 id25566       Firefox     Desktop   not happy
##  text2   148    259         9 id44027          Edge      Mobile       happy
##  text3    33     42         5 id46041       Mozilla      Tablet   not happy
##  text4   119    185         4 id27812       Mozilla      Mobile       happy
##  text5    78    188         8 id25545 Google Chrome     Desktop       happy
```

Let's do the cross-validation stuff (we go into details later but essentially create a training dataset for training the algorithm and a test dataset for testing the efficacy of the algorithm) using quanteda package (we mostly package caret in later exercises).

```
# generate 3500 numbers without replacement
set.seed(300)
id_train <- sample(1:5000, 3500, replace = FALSE)

# create docvar with ID
corp_hotel$id_numeric <- 1:ndoc(corp_hotel)

# get training set
dfmat_training <- corpus_subset(corp_hotel, id_numeric %in% id_train) %>%
tokens(remove_punct = TRUE) %>%  # Tokenize while removing punctuation
  tokens_remove(stopwords("english"), padding = FALSE) %>%  # Remove stopwords
  tokens_wordstem(language = "english") %>%  # Apply stemming
  dfm()

#Since we will run  the binary multinomial NB model, let's convert the dfm to a binary matrix before tr

dfmat_training<-dfm_weight(dfmat_training, scheme = "boolean")

# get test set (documents not in id_train)
```

```r
dfmat_test <- corpus_subset(corp_hotel, !id_numeric %in% id_train) %>%
tokens(remove_punct = TRUE) %>%  # Tokenize while removing punctuation
  tokens_remove(stopwords("english"), padding = FALSE) %>%  # Remove stopwords
  tokens_wordstem(language = "english") %>%  # Apply stemming
  dfm()

dfmat_test<-dfm_weight(dfmat_test, scheme = "boolean")
```

I ran the model both ways, with scheme =boolean and without. As you will see, the prediction accuracy increased substantially when we use the scheme boolean.

Let's train the NB model. Naive Bayes model will learn classification based on the existing responses

```r
tmod_nb <- textmodel_nb(dfmat_training, dfmat_training$Is_Response)
summary(tmod_nb)
```

```
##
## Call:
## textmodel_nb.dfm(x = dfmat_training, y = dfmat_training$Is_Response)
##
## Class Priors:
## (showing first 2 elements)
##     happy not happy
##       0.5       0.5
##
## Estimated Feature Scores:
##                hotel     rate    either    romant      one    boston      agre
## happy     0.011855 0.001584 0.0004054 1.684e-04 0.004534 0.0004989 0.0001434
## not happy 0.009385 0.001403 0.0008195 8.981e-05 0.004850 0.0002806 0.0002582
##                 delux     suit   bathtub   shower      half     glass      wall
## happy     0.0001372 0.001771 0.0001621 0.001297 0.0002931 0.0003991 0.0005426
## not happy 0.0001572 0.001078 0.0002133 0.001706 0.0004939 0.0004154 0.0013583
##                 meant     much    water     went      onto    floor        tv
## happy     0.0001372 0.002027 0.001123 0.001397 0.0001185 0.002457 0.001229
## not happy 0.0001796 0.002088 0.001370 0.001617 0.0002245 0.002627 0.001010
##                 work    proper    comput     room      love      lamp     side
## happy     0.001796 0.0001434 0.0003118 0.01231 0.0026255 9.355e-05 0.001029
## not happy 0.002470 0.0004490 0.0003705 0.01032 0.0008644 1.908e-04 0.001100
##                  bed   servic
## happy     0.004659 0.004503
## not happy 0.003963 0.003547
```

We need to make training set and the test set features identical.

```
dfmat_matched <- dfm_match(dfmat_test, features = featnames(dfmat_training))
```

Let's test how the classification worked:

```
actual_class <- dfmat_matched$Is_Response
predicted_class <- predict(tmod_nb, newdata = dfmat_matched)
tab_class <- table(actual_class, predicted_class)
tab_class
```

```
##              predicted_class
## actual_class happy not happy
##    happy       978        43
##    not happy   136       343
```

```
#confusionMatrix(tab_class, mode = "everything")
```

This is fairly decent result for happy category, a little worse for unhappy category. We will go into much greater details in a later section.

Resources

1) Bing Liu Tutorial: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon
2) https://www.cs.uic.edu/~liub/FBS/NLP-handbook-sentiment-analysis.pdf
3) http://sentiment.christopherpotts.net/index.html
4) Dependency Parsing: https://web.stanford.edu/~jurafsky/slp3/15.pdf

References

Scherer, Klaus R. 1984. Emotion as a Multicomponent Process: A model and some cross-cultural data. In P. Shaver, ed., Review of Personality and Social Psych 5: 37-63.

Ekman, Paul. 1985. Telling Lies: Clues to Deceit in the Marketplace, Politics, and Marriage. New York: Norton.

Sanjiv Das and Mike Chen. 2001. Yahoo! for Amazon: extracting market sentiment from stock message boards. In Proceedings of the 8th Asia Pacific Finance Association Annual Conference.

Pang, Bo; Lillian Lee; and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In Proceedings of the Conference on Empirical Methods in Natural Language Processing. ACL.

Plutchik, Robert. 2002. Emotions and Life. Washington, D.C.: American Psychological Association.

V. Metsis, I. Androutsopoulos, G. Paliouras. 2006. Spam Filtering with Naive Bayes – Which Naive Bayes? CEAS 2006 -Third Conference on Email and AntiSpam.