Ads project1

Cindy Xu UNI:cx2199

2/3/2018

# Section 1: Check and install needed packages. Load the libraries and functions.

packages.used<-c("ggplot2","dplyr","tibble","tidyr","stringr", "tidytext","topicmodels","wordcloud",  
 "ggridges",'igraph','ggraph','sentimentr',"devtools",  
 'exploratory',"ldatuning",'CTM','purrr','stm','corpus'  
 ,'tm','quanteda')  
# check packages that need to be installed.  
packages.needed <- setdiff(packages.used, intersect(installed.packages()[,1], packages.used))  
  
# install additional packages  
if(length(packages.needed) > 0) {  
 install.packages(packages.needed, dependencies = TRUE, repos = 'http://cran.us.r-project.org')  
}  
  
#devtools::install\_github("exploratory-io/exploratory\_func")  
  
library(ggplot2)  
library(dplyr)  
library(tibble)  
library(tidyr)  
library(stringr)  
library(tidytext)  
library(topicmodels)  
library(RColorBrewer)  
library(wordcloud)  
library(ggridges)  
library(igraph)  
library(ggraph)  
library(sentimentr)  
library(syuzhet)  
library(broom)  
library(urltools)  
library(exploratory)  
library(ldatuning)  
library(purrr)  
library(CTM)  
library(stm)  
library(corpus)  
library(tm)  
library(quanteda)  
source("../libs/multiplot.R")

# Section 2: Read in the data

The following code assumes that the dataset spooky.csv lives in a data folder (and that we are inside a doc folder).

## Step 1: Using spooky

spooky<-read.csv('../data/spooky.csv',as.is=T)

### An overview of the data structure and content

Let’s first remind ourselves of the structure of the data.

dim<-dim(spooky)  
dim

## [1] 19579 3

head(spooky)

## id  
## 1 id26305  
## 2 id17569  
## 3 id11008  
## 4 id27763  
## 5 id12958  
## 6 id22965  
## text  
## 1 This process, however, afforded me no means of ascertaining the dimensions of my dungeon; as I might make its circuit, and return to the point whence I set out, without being aware of the fact; so perfectly uniform seemed the wall.  
## 2 It never once occurred to me that the fumbling might be a mere mistake.  
## 3 In his left hand was a gold snuff box, from which, as he capered down the hill, cutting all manner of fantastic steps, he took snuff incessantly with an air of the greatest possible self satisfaction.  
## 4 How lovely is spring As we looked from Windsor Terrace on the sixteen fertile counties spread beneath, speckled by happy cottages and wealthier towns, all looked as in former years, heart cheering and fair.  
## 5 Finding nothing else, not even gold, the Superintendent abandoned his attempts; but a perplexed look occasionally steals over his countenance as he sits thinking at his desk.  
## 6 A youth passed in solitude, my best years spent under your gentle and feminine fosterage, has so refined the groundwork of my character that I cannot overcome an intense distaste to the usual brutality exercised on board ship: I have never believed it to be necessary, and when I heard of a mariner equally noted for his kindliness of heart and the respect and obedience paid to him by his crew, I felt myself peculiarly fortunate in being able to secure his services.  
## author  
## 1 EAP  
## 2 HPL  
## 3 EAP  
## 4 MWS  
## 5 HPL  
## 6 MWS

summary(spooky)

## id text author   
## Length:19579 Length:19579 Length:19579   
## Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character

sum(is.na(spooky))

## [1] 0

spooky$author<-as.factor(spooky$author)  
unique(spooky$author)

## [1] EAP HPL MWS  
## Levels: EAP HPL MWS

When we look into spooky data set, it is a 19579 rows and 3 columns dataset. Each row correspoding a unique id number, an excerpt of texts, and author name. Addtionally, there are no missing values. There are three authors, Like HPL is Lovecraft, MWS is Shelly, and EAP is Poe.

## 

## Step 2: Data Processing

## 1: Punctuation – typical sentence structure. Clauses they have. Number of commas or semicolons.

str\_count(spooky,',')

## [1] 19578 57798 19578

str\_count(spooky,';' )

## [1] 0 5159 0

Poe used commas 19578 times, Lovecraft used commas 57798 times, Shelly used commas 19578 times. Poe used semicolons 0 times, Lovecraft used semicolons 5159 times, Shelly used semicolons 0 times.

### 

### 2: He/she

str\_count(spooky,'He')

## [1] 0 1251 0

str\_count(spooky,'he')

## [1] 0 61490 0

str\_count(spooky,'She')

## [1] 0 320 0

str\_count(spooky,'she')

## [1] 0 2028 0

Poe and Shelly did not use he/she, Lovecraft used he more than she.

## Step 3: Data Cleaning

### 1: Drop all punctuation and transform all words into lower case.

spooky\_wrd<-unnest\_tokens(spooky,word,text)  
head(spooky\_wrd)

## id author word  
## 1 id26305 EAP this  
## 1.1 id26305 EAP process  
## 1.2 id26305 EAP however  
## 1.3 id26305 EAP afforded  
## 1.4 id26305 EAP me  
## 1.5 id26305 EAP no

### 

### 2: Bi-grams, n-grams

If we wanna get relationships between words, we use n-grams. So far we’ve considered words as individual units, and considered their relationships to sentiments or to documents. However, many interesting text analyses are based on the relationships between words, whether examining which words tend to follow others immediately. we’ll explore some of the methods tidytext offers for calculating and visualizing relationships between words in your text dataset. This includes the token = “ngrams” argument, which tokenizes by pairs of adjacent words rather than by individual ones. We’ll also introduce two new packages: ggraph, which extends ggplot2 to construct network plots, and widyr, which calculates pairwise correlations and distances within a tidy data frame. Together these expand our toolbox for exploring text within the tidy data framework.

#### Tokenizing by n-gram

We’ve been using the unnest\_tokens function to tokenize by word, or sometimes by sentence, which is useful for the kinds of sentiment and frequency analyses we’ve been doing so far. But we can also use the function to tokenize into consecutive sequences of words, called n-grams. By seeing how often word X is followed by word Y, we can then build a model of the relationships between them. We do this by adding the token = “ngrams” option to unnest\_tokens(), and setting n to the number of words we wish to capture in each n-gram. When we set n to 2, we are examining pairs of two consecutive words, often called “bigrams”

# Make a table with one word per row and remove `stop words` (i.e. the common words).  
bigrams<-unnest\_tokens(spooky,bigram, text, token = "ngrams", n = 2)  
head(bigrams)

## id author bigram  
## 1 id00001 MWS idris was  
## 2 id00001 MWS was well  
## 3 id00001 MWS well content  
## 4 id00001 MWS content with  
## 5 id00001 MWS with this  
## 6 id00001 MWS this resolve

bigrams\_HPL<-unnest\_tokens(spooky[spooky$author=='HPL',],bigram, text, token = "ngrams", n = 2)  
head(bigrams\_HPL)

## id author bigram  
## 1 id00002 HPL i was  
## 2 id00002 HPL was faint  
## 3 id00002 HPL faint even  
## 4 id00002 HPL even fainter  
## 5 id00002 HPL fainter than  
## 6 id00002 HPL than the

bigrams\_MWS<-unnest\_tokens(spooky[spooky$author=='MWS',],bigram, text, token = "ngrams", n = 2)  
head(bigrams\_MWS)

## id author bigram  
## 1 id00001 MWS idris was  
## 2 id00001 MWS was well  
## 3 id00001 MWS well content  
## 4 id00001 MWS content with  
## 5 id00001 MWS with this  
## 6 id00001 MWS this resolve

bigrams\_EAP<-unnest\_tokens(spooky[spooky$author=='EAP',],bigram, text, token = "ngrams", n = 2)  
head(bigrams\_EAP)

## id author bigram  
## 1 id00003 EAP above all  
## 2 id00003 EAP all i  
## 3 id00003 EAP i burn  
## 4 id00003 EAP burn to  
## 5 id00003 EAP to know  
## 6 id00003 EAP know the

This data structure is still a variation of the tidy text format. It is structured as one-token-per-row (with extra metadata, such as author, still preserved), but each token now represents a bigram.

##### (1): Counting and filtering n-grams

Our usual tidy tools apply equally well to n-gram analysis. We can examine the most common bigrams using dplyr’s count():

bigrams\_count<-count(bigrams,bigram,sort=T)  
head(bigrams\_count)

## # A tibble: 6 x 2  
## bigram n  
## <chr> <int>  
## 1 of the 5581  
## 2 in the 2743  
## 3 to the 1847  
## 4 and the 1343  
## 5 it was 1037  
## 6 from the 1036

bigrams\_EAP\_count<-count(bigrams\_EAP,bigram,sort=T)  
head(bigrams\_EAP\_count)

## # A tibble: 6 x 2  
## bigram n  
## <chr> <int>  
## 1 of the 2877  
## 2 in the 1237  
## 3 to the 823  
## 4 of a 530  
## 5 to be 431  
## 6 and the 428

bigrams\_MWS\_count<-count(bigrams\_MWS,bigram,sort=T)  
head(bigrams\_MWS\_count)

## # A tibble: 6 x 2  
## bigram n  
## <chr> <int>  
## 1 of the 1217  
## 2 in the 605  
## 3 to the 534  
## 4 and the 412  
## 5 of my 359  
## 6 on the 356

bigrams\_HPL\_count<-count(bigrams\_HPL,bigram,sort=T)  
head(bigrams\_HPL\_count)

## # A tibble: 6 x 2  
## bigram n  
## <chr> <int>  
## 1 of the 1487  
## 2 in the 901  
## 3 and the 503  
## 4 to the 490  
## 5 on the 428  
## 6 from the 350

As one might expect, a lot of the most common bigrams are pairs of common (uninteresting) words, such as of the and in the: what we call “stop-words” . This is a useful time to use tidyr’s separate(), which splits a column into multiple based on a delimiter. This lets us separate it into two columns, “word1” and “word2”, at which point we can remove cases where either is a stop-word.

bigrams\_separated<-separate(bigrams,bigram,c("word1", "word2"),sep = " ")  
bigrams\_filtered<-bigrams\_separated %>%  
 filter(!word1 %in% stop\_words$word) %>%  
 filter(!word2 %in% stop\_words$word)  
  
# new bigram counts:  
bigram\_counts<-bigrams\_filtered %>%   
 count(word1,word2,sort=T)  
head(bigram\_counts)

## # A tibble: 6 x 3  
## word1 word2 n  
## <chr> <chr> <int>  
## 1 lord raymond 27  
## 2 fellow creatures 22  
## 3 ha ha 22  
## 4 main compartment 21  
## 5 madame lalande 20  
## 6 chess player 18

bigrams\_HPL\_separated<-separate(bigrams\_HPL,bigram,c("word1", "word2"),sep = " ")  
bigrams\_HPL\_filtered<-bigrams\_HPL\_separated %>%  
 filter(!word1 %in% stop\_words$word) %>%  
 filter(!word2 %in% stop\_words$word)  
  
# new bigram counts:  
bigram\_HPL\_counts<-bigrams\_HPL\_filtered %>%   
 count(word1,word2,sort=T)  
head(bigram\_HPL\_counts)

## # A tibble: 6 x 3  
## word1 word2 n  
## <chr> <chr> <int>  
## 1 heh heh 17  
## 2 shunned house 16  
## 3 tempest mountain 14  
## 4 brown jenkin 13  
## 5 herbert west 13  
## 6 yog sothoth 12

bigrams\_MWS\_separated<-separate(bigrams\_MWS,bigram,c("word1", "word2"),sep = " ")  
bigrams\_MWS\_filtered<-bigrams\_MWS\_separated %>%  
 filter(!word1 %in% stop\_words$word) %>%  
 filter(!word2 %in% stop\_words$word)  
  
# new bigram counts:  
bigram\_MWS\_counts<-bigrams\_MWS\_filtered %>%   
 count(word1,word2,sort=T)  
head(bigram\_MWS\_counts)

## # A tibble: 6 x 3  
## word1 word2 n  
## <chr> <chr> <int>  
## 1 lord raymond 27  
## 2 fellow creatures 22  
## 3 native country 14  
## 4 natural philosophy 10  
## 5 poor girl 10  
## 6 human race 9

bigrams\_EAP\_separated<-separate(bigrams\_EAP,bigram,c("word1", "word2"),sep = " ")  
bigrams\_EAP\_filtered<-bigrams\_EAP\_separated %>%  
 filter(!word1 %in% stop\_words$word) %>%  
 filter(!word2 %in% stop\_words$word)  
  
# new bigram counts:  
bigram\_EAP\_counts<-bigrams\_EAP\_filtered %>%   
 count(word1,word2,sort=T)  
head(bigram\_EAP\_counts)

## # A tibble: 6 x 3  
## word1 word2 n  
## <chr> <chr> <int>  
## 1 ha ha 22  
## 2 main compartment 21  
## 3 madame lalande 20  
## 4 chess player 18  
## 5 left arm 13  
## 6 tea pot 13

We can see that these phrases are the most common pairs in spooky data set.

In other analyses, we may want to work with the recombined words. tidyr’s unite() function is the inverse of separate(), and lets us recombine the columns into one. Thus, “separate/filter/count/unite” let us find the most common bigrams not containing stop-words.

bigrams\_united<-bigrams\_filtered %>%  
 unite(bigram, word1, word2, sep = " ")  
head(bigrams\_united)

## id author bigram  
## 1 id00002 HPL hateful modernity  
## 2 id00002 HPL accursed city  
## 3 id00003 EAP dark valley  
## 4 id00004 EAP unusual clearness  
## 5 id00004 EAP necessarily lost  
## 6 id00004 EAP lost sight

##### (2): Analyzing bigrams

A bigram can also be treated as a term in a document in the same way that we treated individual words. For example, we can look at the tf-idf of bigrams across spooky dataset.

TF stands for term frequency or how often a word appears in a text and it is what is studied above in the word cloud. IDF stands for inverse document frequncy, and it is a way to pay more attention to words that are rare within the entire set of text data that is more sophisticated than simply removing stop words. Multiplying these two values together calculates a term’s tf-idf, which is the frequency of a term adjusted for how rarely it is used. We’ll use tf-idf as a heuristic index to indicate how frequently a certain author uses a word relative to the frequency that all the authors use the word. Therefore we will find words that are characteristic for a specific author, a good thing to have if we are interested in solving the author identification problem.

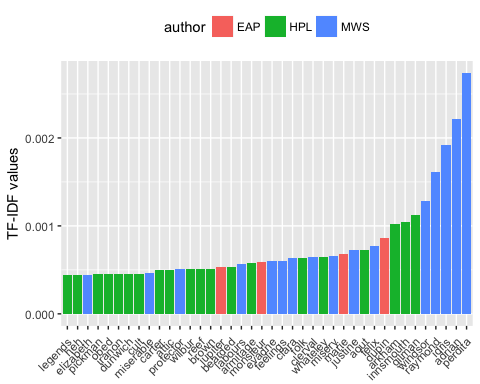
#get rid of stop words  
spooky\_wrd <- anti\_join(spooky\_wrd, stop\_words, by = "word")  
frequency<-count(spooky\_wrd,author,word)  
tf\_idf<-bind\_tf\_idf(frequency,word,author,n)  
head(tf\_idf)

## # A tibble: 6 x 6  
## author word n tf idf tf\_idf  
## <chr> <chr> <int> <dbl> <dbl> <dbl>  
## 1 EAP à 9 0.000124 1.10 0.000136   
## 2 EAP a.m 3 0.0000412 0.405 0.0000167  
## 3 EAP aaem 1 0.0000137 1.10 0.0000151  
## 4 EAP ab 1 0.0000137 1.10 0.0000151  
## 5 EAP aback 2 0.0000275 1.10 0.0000302  
## 6 EAP abandon 7 0.0000961 0 0

tail(tf\_idf)

## # A tibble: 6 x 6  
## author word n tf idf tf\_idf  
## <chr> <chr> <int> <dbl> <dbl> <dbl>  
## 1 MWS youth's 1 0.0000160 0.405 0.00000649  
## 2 MWS youthful 10 0.000160 0 0   
## 3 MWS youths 2 0.0000320 0.405 0.0000130   
## 4 MWS zaimi 2 0.0000320 1.10 0.0000352   
## 5 MWS zeal 7 0.000112 0 0   
## 6 MWS zest 3 0.0000480 0 0

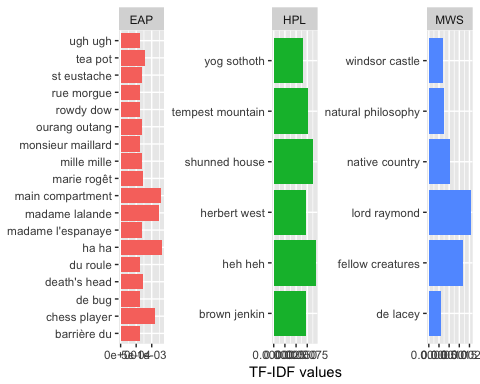
tf\_idf<-arrange(tf\_idf,desc(tf\_idf))  
tf\_idf<-mutate(tf\_idf, word = factor(word,levels= rev(unique(word))))  
  
# Grab the top fourty tf\_idf scores in all the words   
tf\_idf\_40<- top\_n(tf\_idf,40,tf\_idf)  
  
ggplot(tf\_idf\_40) +  
 geom\_col(aes(word,tf\_idf,fill = author)) +  
 labs(x = NULL, y = "TF-IDF values") +  
 theme(legend.position ="top",axis.text.x= element\_text(angle=45,hjust=1,vjust=0.9))



Note that in the above, many of the words recognized by their tf-idf scores are names. This makes sense – if we see text referencing Raymond, Idris, or Perdita, we know almost for sure that MWS is the author. But some non-names stand out. EAP often uses “monsieur” and “jupiter” while HPL uses the words “bearded” and “attic” more frequently than the others. We can also look at the most characteristic terms per author.

Then we can look at the tf-idf of bigrams across spooky datasts.

bigram\_tf\_idf<-bigrams\_united %>%  
 count(author,bigram) %>%  
 bind\_tf\_idf(bigram,author,n) %>%  
 arrange(desc(tf\_idf))  
bigram\_tf\_idf\_30<-head(bigram\_tf\_idf,30)  
ggplot(bigram\_tf\_idf\_30) +  
 geom\_col(aes(bigram,tf\_idf, fill = author)) +  
 labs(x = NULL, y = "bigram\_tf\_idf") +  
 theme(legend.position = "none") +  
 facet\_wrap(~ author,ncol =3,scales="free")+  
 coord\_flip() +  
 labs(y = "TF-IDF values")



There are advantages and disadvantages to examining the tf-idf of bigrams rather than individual words. Pairs of consecutive words might capture structure that isn’t present when one is just counting single words, and may provide context that makes tokens more understandable. However, the per-bigram counts are also sparser: a typical two-word pair is rarer than either of its component words.

# Section 3: Sentiment Analysis

## Step1: Word level

### 1: Using bigrams to provide context in sentiment analysis

Our sentiment analysis approach in simply counted the appearance of positive or negative words, according to a reference lexicon. One of the problems with this approach is that a word’s context can matter nearly as much as its presence. For example, the words “happy” and “like” will be counted as positive, even in a sentence like “I’m not happy and I don’t like it!”

Now that we have the data organized into bigrams, it’s easy to tell how often words are preceded by a word like “not”:

bigrams\_separated %>%  
 filter(word1 == "not") %>%  
 count(word1, word2, sort = TRUE)

## # A tibble: 946 x 3  
## word1 word2 n  
## <chr> <chr> <int>  
## 1 not to 139  
## 2 not be 131  
## 3 not the 103  
## 4 not a 88  
## 5 not have 72  
## 6 not only 66  
## 7 not in 57  
## 8 not so 57  
## 9 not even 44  
## 10 not been 37  
## # ... with 936 more rows

By performing sentiment analysis on the bigram data, we can examine how often sentiment-associated words are preceded by “not” or other negating words. We could use this to ignore or even reverse their contribution to the sentiment score.

Let’s use the AFINN lexicon for sentiment analysis, which you may recall gives a numeric sentiment score for each word, with positive or negative numbers indicating the direction of the sentiment.

AFINN<-get\_sentiments("afinn")

We can then examine the most frequent words that were preceded by “not” and were associated with a sentiment.

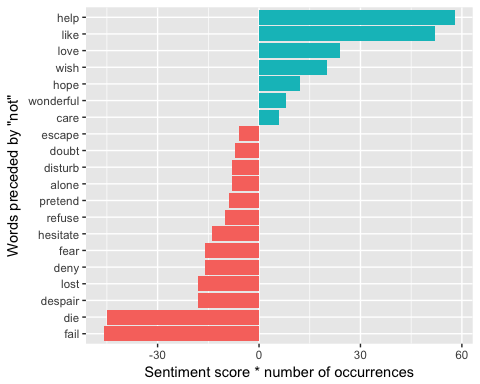
not\_words<-bigrams\_separated %>%  
 filter(word1 == "not") %>%  
 inner\_join(AFINN, by = c(word2 = "word")) %>%  
 count(word2, score, sort = TRUE) %>%  
 ungroup()  
not\_words

## # A tibble: 158 x 3  
## word2 score n  
## <chr> <int> <int>  
## 1 help 2 29  
## 2 like 2 26  
## 3 fail -2 23  
## 4 wish 1 20  
## 5 die -3 15  
## 6 pretend -1 9  
## 7 deny -2 8  
## 8 fear -2 8  
## 9 love 3 8  
## 10 doubt -1 7  
## # ... with 148 more rows

For example, the most common sentiment-associated word to follow “not” was “help”, which would normally have a (positive) score of 2.

It’s worth asking which words contributed the most in the “wrong” direction. To compute that, we can multiply their score by the number of times they appear (so that a word with a score of +3 occurring 10 times has as much impact as a word with a sentiment score of +1 occurring 30 times). We visualize the result with a bar plot.

not\_words %>%  
 mutate(contribution = n \* score) %>%  
 arrange(desc(abs(contribution))) %>%  
 head(20) %>%  
 mutate(word2 = reorder(word2, contribution)) %>%  
 ggplot(aes(word2, n \* score, fill = n \* score > 0)) +  
 geom\_col(show.legend = FALSE) +  
 xlab("Words preceded by \"not\"") +  
 ylab("Sentiment score \* number of occurrences") +  
 coord\_flip()



The 20 words preceded by ‘not’ that had the greatest contribution to sentiment scores, in either a positive or negative direction. The bigrams “not help” and “not like” were overwhelmingly the largest causes of misidentification, making the text seem much more positive than it is. But we can see phrases like “not fail” and “not die” sometimes suggest text is more negative than it is.

“Not” isn’t the only term that provides some context for the following word. We could pick four common words (or more) that negate the subsequent term, and use the same joining and counting approach to examine all of them at once.

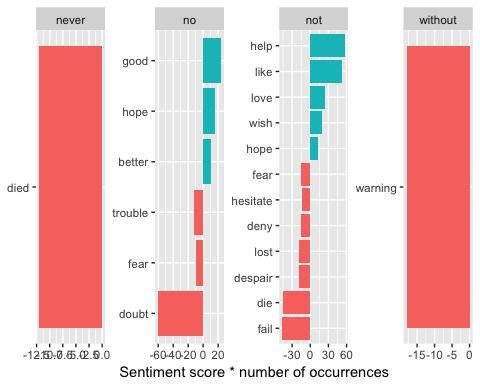
negation\_words <- c("not", "no", "never", "without")  
  
negated\_words<-bigrams\_separated %>%  
 filter(word1 %in% negation\_words) %>%  
 inner\_join(AFINN, by = c(word2 = "word")) %>%  
 count(word1, word2, score, sort = TRUE)   
head(negated\_words)

## # A tibble: 6 x 4  
## word1 word2 score n  
## <chr> <chr> <int> <int>  
## 1 no doubt -1 60  
## 2 not help 2 29  
## 3 not like 2 26  
## 4 not fail -2 23  
## 5 not wish 1 20  
## 6 not die -3 15

negated\_words$word1<-as.factor(negated\_words$word1)  
unique(negated\_words$word1)

## [1] no not never without  
## Levels: never no not without

negated\_words %>%  
 mutate(contribution = n \* score) %>%  
 arrange(desc(abs(contribution))) %>%  
 head(20) %>%  
 mutate(word2 = reorder(word2, contribution)) %>%  
 ggplot(aes(word2, n \* score, fill = n \* score > 0)) +  
 geom\_col(show.legend = FALSE) +  
 labs(x = NULL, y = "Sentiment score \* number of occurrences")+  
 facet\_wrap(~ word1,ncol =4,scales="free")+  
 coord\_flip()



“not doubt” and “not help” are the two most common examples, we can also see pairings such as “no hope” and “never forget.” We could combine this to reverse the AFINN scores of each word that follows a negation.

### 2: Compare Afinn, Bing with NRC

# Keep words that have been classified within the NRC lexicon.  
get\_sentiments('afinn')

## # A tibble: 2,476 x 2  
## word score  
## <chr> <int>  
## 1 abandon -2  
## 2 abandoned -2  
## 3 abandons -2  
## 4 abducted -2  
## 5 abduction -2  
## 6 abductions -2  
## 7 abhor -3  
## 8 abhorred -3  
## 9 abhorrent -3  
## 10 abhors -3  
## # ... with 2,466 more rows

sentiments\_afinn <- inner\_join(spooky\_wrd, get\_sentiments('afinn'), by = "word")  
head(sentiments\_afinn)

## id author word score  
## 1 id26305 EAP no -1  
## 2 id26305 EAP perfectly 3  
## 3 id17569 HPL mistake -2  
## 4 id11008 EAP cutting -1  
## 5 id11008 EAP fantastic 4  
## 6 id11008 EAP greatest 3

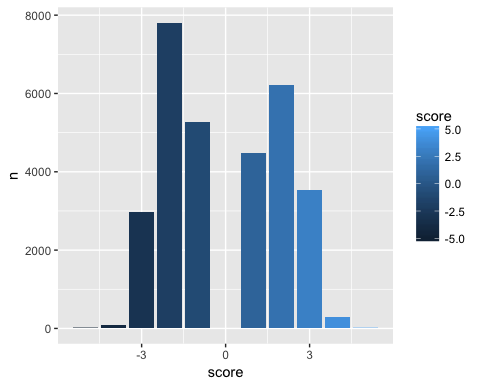
count(sentiments\_afinn, score)

## # A tibble: 10 x 2  
## score n  
## <int> <int>  
## 1 -5 12  
## 2 -4 98  
## 3 -3 2961  
## 4 -2 7810  
## 5 -1 5280  
## 6 1 4479  
## 7 2 6220  
## 8 3 3529  
## 9 4 291  
## 10 5 9

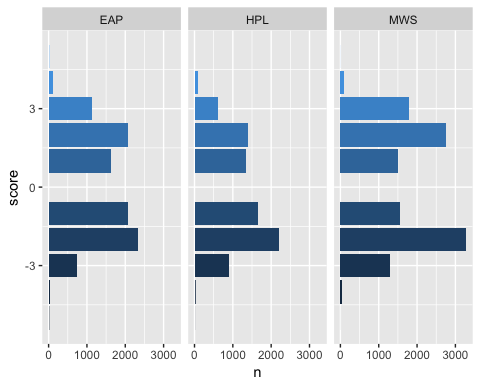
count(sentiments\_afinn, author, score)

## # A tibble: 29 x 3  
## author score n  
## <chr> <int> <int>  
## 1 EAP -5 9  
## 2 EAP -4 28  
## 3 EAP -3 751  
## 4 EAP -2 2321  
## 5 EAP -1 2072  
## 6 EAP 1 1639  
## 7 EAP 2 2071  
## 8 EAP 3 1121  
## 9 EAP 4 113  
## 10 EAP 5 7  
## # ... with 19 more rows

ggplot(count(sentiments\_afinn, score)) +   
 geom\_col(aes(score, n, fill = score))



ggplot(count(sentiments\_afinn, author, score)) +   
 geom\_col(aes(score, n, fill = score)) +   
 facet\_wrap(~ author) +  
 coord\_flip() +  
 theme(legend.position = "none")



get\_sentiments('bing')

## # A tibble: 6,788 x 2  
## word sentiment  
## <chr> <chr>   
## 1 2-faced negative   
## 2 2-faces negative   
## 3 a+ positive   
## 4 abnormal negative   
## 5 abolish negative   
## 6 abominable negative   
## 7 abominably negative   
## 8 abominate negative   
## 9 abomination negative   
## 10 abort negative   
## # ... with 6,778 more rows

sentiments\_bing<- inner\_join(spooky\_wrd, get\_sentiments('bing'), by = "word")  
head(sentiments\_bing)

## id author word sentiment  
## 1 id26305 EAP dungeon negative  
## 2 id26305 EAP perfectly positive  
## 3 id17569 HPL mistake negative  
## 4 id11008 EAP gold positive  
## 5 id11008 EAP fantastic positive  
## 6 id11008 EAP incessantly negative

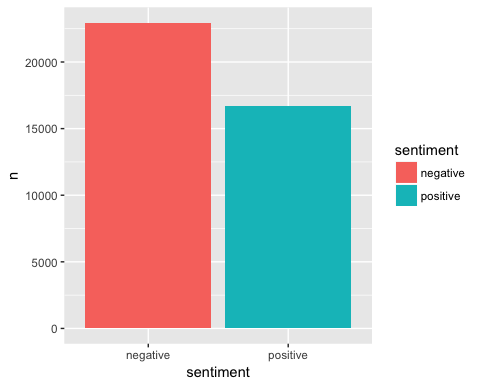
count(sentiments\_bing,sentiment)

## # A tibble: 2 x 2  
## sentiment n  
## <chr> <int>  
## 1 negative 22958  
## 2 positive 16674

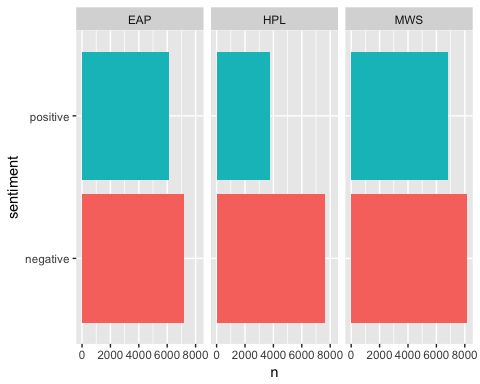
count(sentiments\_bing,author,sentiment)

## # A tibble: 6 x 3  
## author sentiment n  
## <chr> <chr> <int>  
## 1 EAP negative 7203  
## 2 EAP positive 6144  
## 3 HPL negative 7605  
## 4 HPL positive 3731  
## 5 MWS negative 8150  
## 6 MWS positive 6799

ggplot(count(sentiments\_bing,sentiment)) +   
 geom\_col(aes(sentiment, n, fill = sentiment))



ggplot(count(sentiments\_bing, author, sentiment)) +   
 geom\_col(aes(sentiment, n, fill = sentiment)) +   
 facet\_wrap(~ author) +  
 coord\_flip() +  
 theme(legend.position = "none")



get\_sentiments('nrc')

## # A tibble: 13,901 x 2  
## word sentiment  
## <chr> <chr>   
## 1 abacus trust   
## 2 abandon fear   
## 3 abandon negative   
## 4 abandon sadness   
## 5 abandoned anger   
## 6 abandoned fear   
## 7 abandoned negative   
## 8 abandoned sadness   
## 9 abandonment anger   
## 10 abandonment fear   
## # ... with 13,891 more rows

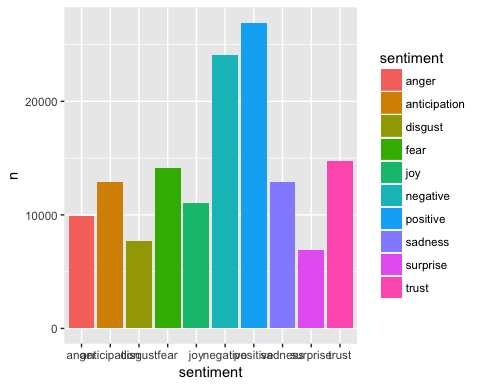
sentiments <- inner\_join(spooky\_wrd, get\_sentiments('nrc'), by = "word")  
  
count(sentiments, sentiment)

## # A tibble: 10 x 2  
## sentiment n  
## <chr> <int>  
## 1 anger 9869  
## 2 anticipation 12912  
## 3 disgust 7731  
## 4 fear 14096  
## 5 joy 11077  
## 6 negative 24084  
## 7 positive 26934  
## 8 sadness 12896  
## 9 surprise 6903  
## 10 trust 14777

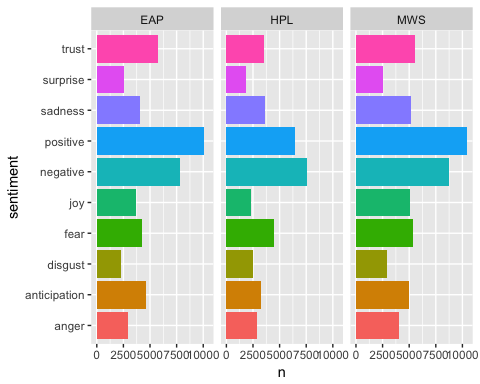
count(sentiments, author, sentiment)

## # A tibble: 30 x 3  
## author sentiment n  
## <chr> <chr> <int>  
## 1 EAP anger 2962  
## 2 EAP anticipation 4656  
## 3 EAP disgust 2273  
## 4 EAP fear 4287  
## 5 EAP joy 3652  
## 6 EAP negative 7833  
## 7 EAP positive 10083  
## 8 EAP sadness 4045  
## 9 EAP surprise 2538  
## 10 EAP trust 5739  
## # ... with 20 more rows

ggplot(count(sentiments, sentiment)) +   
 geom\_col(aes(sentiment, n, fill = sentiment))



ggplot(count(sentiments, author, sentiment)) +   
 geom\_col(aes(sentiment, n, fill = sentiment)) +   
 facet\_wrap(~ author) +  
 coord\_flip() +  
 theme(legend.position = "none")



Based on afinn, we see the whole text contains more negative words, like score=-5. And Shelly uses more extreme words than others.

Based on bing, we learn three authors are all negative and Shelly has more negative emotions than other two.

Based on nrc, we get the number of words for different emotions for whole text. And then, we can get information for different authors. They pay attention on different emotions.

We then use spread() so that we have negative and positive sentiment in separate columns, and lastly calculate a net sentiment (positive - negative). And Now we can plot these sentiment scores across the plot trajectory of each author. Notice that we are plotting against the index on the x-axis that keeps track of text.

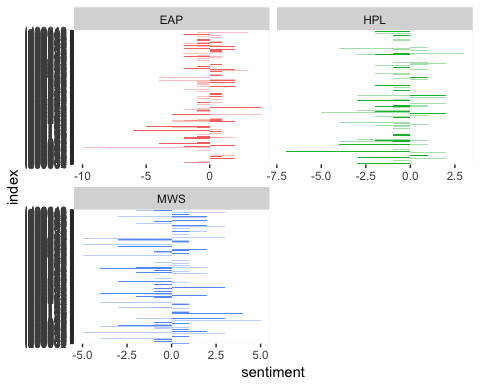
head(spooky\_wrd)

## id author word  
## 1 id26305 EAP this  
## 1.1 id26305 EAP process  
## 1.2 id26305 EAP however  
## 1.3 id26305 EAP afforded  
## 1.4 id26305 EAP me  
## 1.5 id26305 EAP no

specialsentiment<-spooky\_wrd%>%inner\_join(get\_sentiments("bing")) %>%  
 count(index=id, author, sentiment)%>%  
 spread(sentiment, n, fill = 0) %>%  
 mutate(sentiment = positive - negative)

## Joining, by = "word"

specialsentiment\_300<-head(specialsentiment,300)  
  
ggplot(specialsentiment\_300, aes(index , sentiment, fill = author)) +  
 geom\_col(show.legend = FALSE) +  
 facet\_wrap(~author, ncol = 2, scales = "free\_x")+  
 coord\_flip()



We can see how the plot of each author changes toward more positive or negative sentiment over the trajectory of the story.

### 3: Comparing the three sentiment dictionaries

With several options for sentiment lexicons, you might want some more information on which one is appropriate for your purposes. Let’s use all three sentiment lexicons and examine how the sentiment changes across the author.

afinn\_method<-spooky\_wrd%>%   
 inner\_join(get\_sentiments("afinn")) %>%   
 group\_by(index =id) %>%   
 summarise(sentiment = sum(score)) %>%   
 mutate(method = "AFINN")

## Joining, by = "word"

afinn\_300<-head(afinn\_method,300)  
bing\_method<-spooky\_wrd%>%   
 inner\_join(get\_sentiments("bing")) %>%  
 mutate(method = "Bing et al.") %>%  
 count(method, index =id, sentiment) %>%  
 spread(sentiment, n, fill = 0) %>%  
 mutate(sentiment = positive - negative)

## Joining, by = "word"

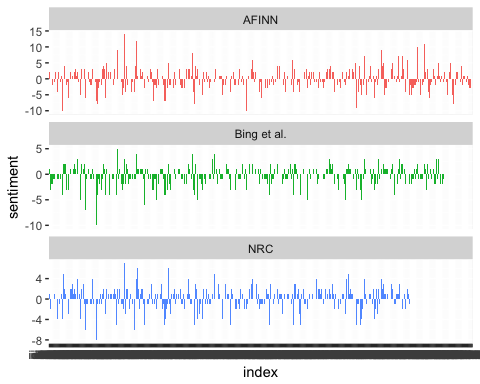
bing\_300<-head(bing\_method,300)  
nrc\_method<-spooky\_wrd%>%  
 inner\_join(get\_sentiments("nrc") %>%  
 filter(sentiment %in% c("positive","negative"))%>%  
 mutate(method = "NRC")) %>%  
 count(method, index =id, sentiment) %>%  
 spread(sentiment, n, fill = 0) %>%  
 mutate(sentiment = positive - negative)

## Joining, by = "word"

nrc\_300<-head(nrc\_method,300)

We now have an estimate of the net sentiment (positive - negative) in each chunk of the text for each sentiment lexicon. Let’s bind them together and visualize them.

bind\_rows(afinn\_300,   
 bing\_300, nrc\_300) %>%  
 ggplot(aes(index, sentiment, fill = method)) +  
 geom\_col(show.legend = FALSE) +  
 facet\_wrap(~method, ncol = 1, scales = "free\_y")



The three different lexicons for calculating sentiment give results that are different in an absolute sense but have similar relative trajectories through the author. We see similar dips and peaks in sentiment at about the same places in the author, but the absolute values are significantly different. The AFINN lexicon gives the largest absolute values, with high positive values. The lexicon from Bing et al. has lower absolute values and seems to label larger blocks of contiguous positive or negative text. The NRC results are shifted higher relative to the other two, labeling the text more positively, but detects similar relative changes in the text. Sentiment appears to find longer stretches of similar text, but all three agree roughly on the overall trends in the sentiment through a narrative arc.

## Step2: Do sentiment analysis at sentense level

spooky<-read.csv('../data/spooky.csv',as.is=T)  
spooky.sentense<-spooky%>%  
 mutate(sentiment = get\_sentiment(text))

## Warning in split\_warn(text.var, "sentiment\_by", ...): Each time  
## `sentiment\_by` is run it has to do sentence boundary disambiguation when  
## a raw `character` vector is passed to `text.var`. This may be costly of  
## time and memory. It is highly recommended that the user first runs the raw  
## `character` vector through the `get\_sentences` function.

count(spooky.sentense, sentiment)

## # A tibble: 8,704 x 2  
## sentiment n  
## <dbl> <int>  
## 1 -2.42 1  
## 2 -2.15 1  
## 3 -1.92 1  
## 4 -1.66 1  
## 5 -1.65 1  
## 6 -1.59 1  
## 7 -1.56 1  
## 8 -1.53 1  
## 9 -1.48 1  
## 10 -1.48 1  
## # ... with 8,694 more rows

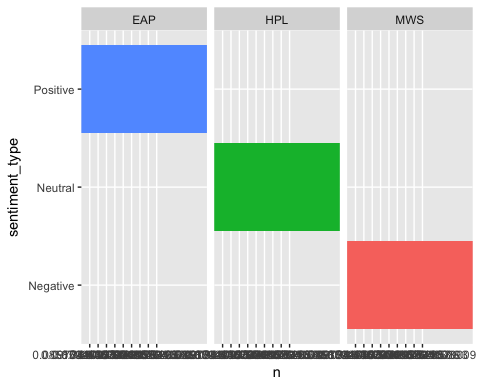
count(spooky.sentense, author, sentiment)

## # A tibble: 11,370 x 3  
## author sentiment n  
## <chr> <dbl> <int>  
## 1 EAP -2.42 1  
## 2 EAP -2.15 1  
## 3 EAP -1.66 1  
## 4 EAP -1.56 1  
## 5 EAP -1.45 1  
## 6 EAP -1.42 1  
## 7 EAP -1.40 1  
## 8 EAP -1.35 1  
## 9 EAP -1.33 1  
## 10 EAP -1.32 1  
## # ... with 11,360 more rows

spooky.sentense.data<-spooky.sentense %>%  
 mutate(sentiment\_type = if\_else(sentiment >0, "Positive", if\_else(sentiment <0, "Negative", "Neutral")))%>%  
 select(sentiment, sentiment\_type,text,author)  
order.spooky.sentense<-spooky.sentense.data[order(spooky.sentense.data$sentiment),]  
positive.rate<-sum(spooky.sentense.data$sentiment\_type=='Positive')/nrow(spooky.sentense.data)  
positive.rate

## [1] 0.4305634

count.whole.table<-count(spooky.sentense.data%>%group\_by(author))  
interger.EAP<-as.integer(count.whole.table[count.whole.table$author=='EAP',]$n)  
interger.HPL<-as.integer(count.whole.table[count.whole.table$author=='HPL',]$n)  
interger.MWS<-as.integer(count.whole.table[count.whole.table$author=='MWS',]$n)  
count.table<-count(spooky.sentense.data%>%group\_by(sentiment\_type, author))   
frequency.EAP<-count.table[count.table$author=='EAP',]$n/  
 as.integer(count.whole.table[count.whole.table$author=='EAP',]$n)  
frequency.HPL<-count.table[count.table$author=='HPL',]$n/  
 as.integer(count.whole.table[count.whole.table$author=='HPL',]$n)  
frequency.MWS<-count.table[count.table$author=='MWS',]$n/  
 as.integer(count.whole.table[count.whole.table$author=='MWS',]$n)  
n<-c(frequency.MWS,frequency.HPL,frequency.EAP)  
author<-c('MWS','MWS','MWS','HPL','HPL','HPL','EAP','EAP','EAP')  
sentiment\_type<-c('Negative','Negative','Negative','Neutral','Neutral','Neutral',  
 'Positive','Positive','Positive')  
frequency.table<-as.data.frame(cbind(sentiment\_type,author,n))  
ggplot(frequency.table)+geom\_col(aes(sentiment\_type, n, fill = sentiment\_type)) +   
 facet\_wrap(~ author) +  
 coord\_flip() +  
 theme(legend.position = "none")

 Proportion of sentences are ‘postive’ is 43%, and for each author based on sensitive level, Poe, Lovecraft, Shelly are positive, neutral, and negative compare to each other.