Sentiment Analysis of Harry Potter Book Series using R

Divyesh Patel
U & P U. PatelDepartment of Computer Engineering, Charutar University of Science & Technology
divyeshpatel.ce@charusat.ac.in

Abstract - We examined the sentiment analysis of the series of book chapter. Basically we have been working for the text mining to find out some fruitful results from it. We took the series of books available on internet harry potter. Various techniques have been applied on that text to find the sentiment analysis.

Keywords— Sentiment Analysis, Harry Potter Book Series, AFINN, BING, NRC

I. INTRODUCTION

Information Extraction from text allows more specialized automation of information structures to be carried out on a non-structural basis or semi-structures.

First we need to clean up our text and carried out some basic word frequency analysis, the next step is to recognize the view or sentiment in the text. This is considered sentiment analysis and this paper will walk you through a simple approach to perform sentiment analysis on text mining to get sentiment of series of books i.e Harry Potter. During the sentiment analysis we have converted the world used in the book chapter in full length from its short length. For ex tl;dr is converted as too long; didn’t read. It is nothing but the English slang used in communication, as its too long so it is represented in short form. There are too many similar words used in the book series of harry potter mainly during the conversation between two parties. The paper also covers the implementation of tidy text which means it find out the sentiment analysis of the frequency of repeated words. The paper will mainly focus on the implementation of

- Replication requirements: What we need to reproduce the analysis during the implementation
- Sentiment data sets: The primary data sets leveraged to count sentiment
- Basic sentiment analysis: Performing basic sentiment analysis
- Comparing sentiments: Comparing how sentiments differ across the sentiment libraries
- Common sentiment words: Discovery the positive and negative words which are common in the text
- Sentiment analysis with larger units: Analyzing sentiment across larger text units rather than individual words

A. Replication Requirements
This paper leverages the data provided in the harry potter book. I constructed this package to supply the first seven novels in the Harry Potter series to illustrate text mining and analysis capabilities. You can load the harry potter package in R from github with the following code which contains all the series of harry potter book texts:

```r
if (packageVersion("devtools") < 1.4) {
  install.packages("devtools")
}
devtools::install_github("bradleyboehmke/harrypotter")
```

In above code lines the if condition will first check whether the devtools package exists in the R or not, if it does not exist then it required at least the R 1.4 to install it. So, the above programming in R only checks for the devtools package and install it. There are so many alternatives to install the devtools package but we have directly installed through commands only. The devtools command is again used to fetched the texts from the github. So under bradleyboehmke there are so many books but we have fetched all the texts of harry potter series.

Next step is to import the following packages for data plotting. The tidyverse is used for data manipulation and plotting. Package stringr is used for text cleaning and forming regular expressions while tidytext package is used for providing additional text mining functions moreover harrypotter itself is a package which contains all the books of harry potter series and it provides the first seven novels of harry potter.

```r
library(tidyverse)
library(stringr)
library(tidytext)
library(harrypotter)
```

The seven novels on which we are implementing our algorithm for sentiment analysis are as below:


Each text is in a character path with each element indicating a single chapter in R. For example, the following illustrates the raw text of the first two chapters of the philosophers_stone the first novel of harry potter.

```r
philosophers_stone[1:2]
```
If you want the first three chapter you can use [1:3] while you want to do analysis of chapter 4 to 6 then you can use [4:6]. Same way you can select as many as chapter with different novels at a time in R but the above code will give you output as below:

B. Sentiment Data Sets

There are a range of dictionaries that exist for estimating the opinion or emotion in text. The tidytext package contains three sentiment lexicons in the sentiments dataset.

```
> sentiments
# A tibble: 27,314 x 4
  word  abacus abscus abscess
  <chr>  <int>  <int>  <int>
1 ababus     3     9     0
2 abscus    165    158    12
3 abscess   1009  1196  1059
```

So, when you pass the code as sentiment word it shows you the which lexicon is used for each word. Moreover, it shows all the words in R from ascending order with its sentiment type. Various sentiment type is available in Sentiment library as trust, fear, anger, love, sadness, positive, negative. There are three type of lexicon as AIFNN, bing and NRC. All three of these lexicons are based on unigrams (or single words). These lexicons include
many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth. The nrc lexicon classifies words in a binary style (“yes”/“no”) into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. The bing lexicon classifies words in a binary style into positive and negative categories. The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. All of this information is presented in the sentiments dataset, and tidytext provides a function get_sentiments() to get specific sentiment lexicons without the columns that are not used in that lexicon. We can see individual lexicon in R as below:

```r
get_sentiments("afinn")
get_sentiments("bing")
get_sentiments("nrc")
```
C. Basic Sentiment Analysis

To perform sentiment analysis we need to have our data in a tidy format which can be made implemented by tidy library in R. The following converts all seven Harry Potter novels into a tibble that has each word by chapter by book.

```r
> 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
> 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
> 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 abdution -2
6 abductions -2
7 abhor -3
8 abhorred -3
9 abhorrent -3
10 abhors -3
# ... with 2,466 more rows
```
Now let’s use the NRC sentiment data set to assess the different sentiments that are represented across the Harry Potter series. We can see that there is a stronger negative presence than positive.
This gives a good overall sense, but what if we want to understand how the sentiment changes over the course of each novel? To do this we perform the following:

1. create an index that breaks up each book by 500 words; this is the approximate number of words on every two pages so this will allow us to assess changes in sentiment even within chapters
2. join the bing lexicon with inner_join to assess the positive vs. negative sentiment of each word
3. count up how many positive and negative words there are for every two pages
4. spread our data and…
5. calculate a net sentiment (positive - negative)
6. plot our data
7. series %>%
   + right_join(get_sentiments("nrc")) %>%
   + filter(!is.na(sentiment)) %>%
   + count(sentiment, sort = TRUE)
   
   # A tibble: 10 x 2
   
<table>
<thead>
<tr>
<th>sentiment</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56579</td>
</tr>
<tr>
<td>2</td>
<td>38324</td>
</tr>
<tr>
<td>3</td>
<td>35866</td>
</tr>
<tr>
<td>4</td>
<td>32750</td>
</tr>
<tr>
<td>5</td>
<td>23485</td>
</tr>
<tr>
<td>6</td>
<td>21544</td>
</tr>
<tr>
<td>7</td>
<td>21123</td>
</tr>
<tr>
<td>8</td>
<td>14298</td>
</tr>
<tr>
<td>9</td>
<td>13381</td>
</tr>
<tr>
<td>10</td>
<td>12991</td>
</tr>
</tbody>
</table>

> series %>%
+ right_join(get_sentiments("nrc")) %>%
+ filter(!is.na(sentiment)) %>%
+ count(sentiment, sort = TRUE)

Joining, by = "word"

This gives a good overall sense, but what if we want to understand how the sentiment changes over the course of each novel? To do this we perform the following:

1. create an index that breaks up each book by 500 words; this is the approximate number of words on every two pages so this will allow us to assess changes in sentiment even within chapters
2. join the bing lexicon with inner_join to assess the positive vs. negative sentiment of each word
3. count up how many positive and negative words there are for every two pages
4. spread our data and…
5. calculate a net sentiment (positive - negative)
6. plot our data
7. series %>%
   + group_by(book) %>%
   + mutate(word_count = 1:n(),
     index = word_count / 500 + 1)
   + inner_join(get_sentiments("bing")) %>%
   + count(book, index = index, sentiment) %>%
   + mutate(sentiment = positive - negative,
     book = factor(book, levels = titles))
   + ggplot(aes(index, sentiment, fill = book)) +
Now we can see how the plot of each novel changes toward more positive or negative sentiment over the trajectory of the story.

D. Comparing Sentiments

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

```
bind_rows(afinn, bing_and_nrc) %>%
  ungroup() %>%
  mutate(book = factor(book, levels = titles)) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_bar(alpha = 0.8, star = "identity", show.legend = FALSE) +
  facet_grid(book ~ method)
```
The three different lexicons for calculating sentiment give results that are different in an absolute sense but have fairly similar relative trajectories through the novels. We see similar dips and peaks in sentiment at about the same places in the novel, but the absolute values are significantly different. In some instances, it appears the AFINN lexicon finds more positive sentiments than the NRC lexicon. This output also allows us to compare
across novels. First, you get a good sense of differences in book lengths - Order of the Phoenix is much longer than Philosopher’s Stone. Second, you can compare how books differ in their sentiment (both direction and magnitude) across a series.

E. Common Sentiment Words

One advantage of having the data frame with both sentiment and word is that we can analyze word counts that contribute to each sentiment.

We can view this visually to assess the top \(n\) words for each sentiment:

```r
bing_word_counts %>%
group_by(sentiment) %>%
top_n(10) %>%
ggplot(aes(reorder(word, n), n, fill = sentiment)) +
geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
facet_wrap(~sentiment, scales = "free_y") +
labs(y = "Contribution to sentiment", x = NULL) +
coord_flip()
```

## Selecting by \(n\)
F. Sentiment Analysis with Larger Units

Lots of useful work can be done by tokenizing at the word level, but sometimes it is useful or necessary to look at different units of text. For example, some sentiment analysis algorithms look beyond only unigrams (i.e. single words) to try to understand the sentiment of a sentence as a whole. These algorithms try to understand that

I am not having a good day.

is a sad sentence, not a happy one, because of negation. The Stanford CoreNLP tools and the sentimentr R package (currently available on Github but not CRAN) are examples of such sentiment analysis algorithms. For these, we may want to tokenize text into sentences. I’ll illustrate using the philosophers_stone data set.
The argument `token = "sentences"` attempts to break up text by punctuation. Note how “mr.” and “mrs.” was placed on their own line. For most text this will have little impact but it is important to be aware of. You can also unnest text by “ngrams”, “lines”, “paragraphs”, and even using “regex”. Check out `?unnest_tokens` for more details.

Let's go ahead and break up the `philosophers_stone` text by chapter and sentence.

```r
tibble(text = philosophers_stone) %>%
  unnest_tokens(sentence, text, token = "sentences")
```

This will allow us to assess the net sentiment by chapter and by sentence. First, we need to track the sentence numbers and then I create an index that tracks the progress through each chapter. I then unnest the sentences by words. This gives us a tibble that has individual words by sentence within each chapter. Now, as before, I join the AFINN lexicon and compute the net sentiment score for each chapter. We can see that the most positive sentences are half way through chapter 9, towards the end of chapter 17, early in chapter 4, etc.

```r
ps_sentences <- tibble(chapter = 1:length(philosophers_stone),
                        text = philosophers_stone) %>%
  unnest_tokens(sentence, text, token = "sentences")

book_sent <- ps_sentences %>%
  group_by(chapter) %>%
  mutate(sentence_num = 1:n(),
         index = round(sentence_num / n(), 2)) %>%
  unnest_tokens(word, sentence) %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(chapter, index) %>%
  summarise(sentiment = sum(score, na.rm = TRUE)) %>%
  arrange(desc(sentiment))
```

`book_sent`
We can visualize this with a heatmap that shows the most positive and negative sentiments as we progress through each chapter:

```r
ggplot(book_sent, aes(index, factor(chapter, levels = sort(unique(chapter), decreasing = TRUE)), fill = sentiment)) +
  geom_tile(color = "white") +
  scale_fill_gradient2() +
  scale_x_continuous(labels = scales::percent, expand = c(0, 0)) +
  scale_y_discrete(expand = c(0, 0)) +
  labs(x = "Chapter Progression", y = "Chapter") +
  ggtitle("Sentiment of Harry Potter and the Philosopher's Stone",
           subtitle = "Summary of the net sentiment score as you progress through each chapter") +
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        legend.position = "top")
```

![Heatmap of sentiment scores](image)

---

**Sentiment of Harry Potter and the Philosopher's Stone**

Summary of the net sentiment score as you progress through each chapter.
G. Word Cloud

Many of the words in the top ten most frequently appearing words are stop-words such as “the”, “and”, “to”, etc., so let’s discard those for now. Below, you can see a word cloud showing the most frequently occurring non-stop words in the series. The cloud contains the top 100 most frequently occurring words, and the larger a word appears in the cloud, the more frequently that word occurred in the text. It comes as no surprise to Harry Potter readers that most of the largest words in the cloud are names like “Harry”, “Ron” and “Hermione”.

```
series$book <- factor(series$book, levels = rev(titles))
series
anti_join(stop_words)
count(word)
with(wordcloud(word, n, max.words = 100))
```

Similarly to the word cloud we created above, we can use the ‘bing’ lexicon to make a comparison cloud. This cloud displays the 50 most frequently occurring words in the series that were categorized by ‘bing’, and color-codes them based on negative or positive sentiment. You’ll notice that words like “Harry”, “Hermione” and “Ron” don’t appear in this cloud, because character names are not classified as positive or negative in ‘bing’.
Just looking at the top ten words in the list, we can see that most of the words that are preceded by “not” in the series, have negative sentiment. This means, we may be over estimating the negative sentiment present in the text. Of course, there are many other negation words such as “never”, “no”, etc. One could explore all of these possible negation words to get a better idea of how negation is affecting the sentiment analysis.

We can also create a graph that connects our most frequently occurring words with each other. Looking at the graph below, we can see a couple of larger clusters that give some context to what the series might be about. For example, there is a cluster with the word “professor” in the center, with several other words connected to it such as “McGonagall” and “Lupin”.

```
series %>%
inner_join(get_sentiments("bing")) %>%
count(word, sentiment, sort = TRUE) %>%
acast(word ~ sentiment, value.var = "n", fill = 0) %>%
comparison.cloud(colors = c("#F8766D", "#00BFC4"),
max.words = 50)
```
```
bigram_graph <- bigram_counts %>% filter(n > 70) %>% graph_from_data_frame()
```

```
set.seed(2017)
ggraph(bigram_graph, layout = "fr") + geom_edge_link() + geom_node_point() + geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```
II. CONCLUSIONS

This paper concludes that on basic text analytics with the Harry Potter series. This is by no means a comprehensive analysis, but it should have demonstrated some of the basic facets of text mining with the tidy verse in R. We have compared two algorithms and find out the best one among them. We have generated the data frame with sentiment derived from the NRC and visualize which word in the NRC sentiment dictionary appear most frequently. Moreover we have also visualize the positive/negative words for each book over the time using SFINN dictionary and the content of each chapter in each book using the NRC dictionary.

REFERENCES


