# Nicholar carter books sentimental analysis project

yashika dutta

22/11/2022

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

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)
```

## Including Plots

You can also embed plots, for example:

```
#plot(pressure)
```

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

---

---

---



---

## I. Introduction and Overview

The aim of this project is to build a sentiment analysis model which will allow us to categorize words based on their sentiments, that is whether they are positive, negative and also the magnitude of it. I implemented it over the data set of Nicholas Carter's books. I delineated it through various visualizations after performing data wrangling. I used a lexical analyzer – 'bing' in this instance of the project. I also represented the sentiment score through a plot and also made a visual report of word clouds.

This project is being completed for Data Science: Capstone (PH125.9x) course in the HarvardX Professional Certificate Data Science Program.

This project is quite different from what is taught in the program. but this is a project which I have always wanted to work on due to having an interest in reading and writing both. I wanted to do this project to help myself in learning different writing styles and how it affects the sentiments of a story. And while working on this project I realized that now with this developed project I can use any data to analyze sentiments and writing styles now.

As I have noticed that sentimental analysis is done usually with Jane Austin pre developed package. But I wanted to try the same study on a different author and a complete different writing style. So I used the methods and observation studied for Jane Austin Novels analysis for My Projects Analysis that is on three novels by Nicholas Carter.

## II. Methods/Analysis

```
a.  Setting The Data

# Initial set up.
## This is the script used to download the Project Gutenberg text files
install.packages("remote",repos = "http://cran.us.r-project.org" )

## package 'remote' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

library(remotes)
install.packages("gutenbergr",repos = "http://cran.us.r-project.org" )

## package 'gutenbergr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

#would have to update few packages version separately in further steps
library(gutenbergr)
install.packages("readr", repos = "http://cran.us.r-project.org")

## package 'readr' successfully unpacked and MD5 sums checked
##
```

```
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

library(readr)
```

After dividing the Nicholas novels into lines, the sentiment can be calculated by tagging the words that have positive or negative sentiment and keeping a cumulative sum over the span of the line.

```
#to run the below make sure to have open network connection

twinmystery <- gutenberg_download(65783)$text
twinmystery <- twinmystery[14:length(twinmystery)]
stolenname <- gutenberg_download(64147)$text
hiddenfoes <- gutenberg_download(62860)$text
hiddenfoes <- iconv(hiddenfoes, "latin1", "UTF-8")

## First, read the plain text files from Project Gutenberg
## Skip lines at the beginning to remove Project Gutenberg header information
## Remove lines at the end to get rid of Project Gutenberg footer information
## A few of these files ended up with NA lines

twinmystery <- read_lines("https://www.gutenberg.org/files/65783/65783-
0.txt", skip = 33)
twinmystery <- twinmystery[1:(length(twinmystery) - 370)]
twinmystery <- twinmystery[!is.na(twinmystery)]

stolenname <- read_lines("https://www.gutenberg.org/files/64147/64147-0.txt",
skip = 30)
stolenname <- stolenname[1:(length(stolenname) - 366)]
stolenname <- stolenname[!is.na(stolenname)]

hiddenfoes <- read_lines("https://www.gutenberg.org/files/62860/62860-0.txt",
skip = 29)
hiddenfoes <- hiddenfoes[1:(length(hiddenfoes) - 367)]
hiddenfoes <- hiddenfoes[!is.na(hiddenfoes)]
```

## Tidy data frame of nicholas carter's 3 completed, published novels

Returns a tidy data frame of nicholas carter's 3 completed, published novels with two columns: , which contains the text of the novels divided into elements of up to about 70 characters each, and , which contains the titles of the novels as a factor in order of publication. @details Users should be aware that there are some differences in usage between the novels as made available by Project Gutenberg. For example, "anything" vs. "any thing", "Mr" vs. "Mr.", and using underscores vs. all caps to indicate italics/emphasis. @return A data frame with two columns:  and  @name carter_books @examples #library(dplyr) #carter_books() %>% group_by(book) %>% # summarise(total_lines = n()) # @export

create a function to named carter_books

```
carter_books <- function(){
  books <- list(
    "The Twin Mystery" = twinmystery,
    "A Stolen Name" = stolenname,
    "The Hidden Foes" = hiddenfoes
  )
  ret <- data.frame(text = unlist(books, use.names = FALSE),
                    stringsAsFactors = FALSE)
  ret$book <- factor(rep(names(books), sapply(books, length)))
  ret$book <- factor(ret$book, levels = unique(ret$book))
  structure(ret, class = c("tbl_df", "tbl", "data.frame"))
}

globalVariables(c("twinmystery", "stolenname", "hiddenfoes",
                  "book"))
```

```
## [1] "twinmystery" "stolenname"  "hiddenfoes"  "book"
```

```
#' The text of nicholas carter's novel "The Twin Mystery"
#'
#' A dataset containing the text of nicholas carter's novel "The
#' Twin Mystery". The UTF-8 plain text was sourced from Project Gutenberg
#' and is divided into elements of up to about 70 characters each.
#' (Some elements are blank.)
#'
#' @source \url{https://www.gutenberg.org/ebooks/65783}
#' @format A character vector with 12262 elements
"twinmystery"
```

```
## [1] "twinmystery"
```

```
#' The text of nicholas carter's novel "A Stolen Name"
#'
#' A dataset containing the text of nicholas carter's novel "A Stolen
#' Name". The UTF-8 plain text was sourced from Project Gutenberg
#' and is divided into elements of up to about 70 characters each.
#' (Some elements are blank.)
#'
#' @source \url{https://www.gutenberg.org/ebooks/64147}
#' @format A character vector with 12447 elements
"stolenname"
```

```
## [1] "stolenname"
```

```
#' The text of nicholas carter's novel "Hidden foes"
#'
#' A dataset containing the text of nicholas carter's novel "Hidden
#' Foes". The UTF-8 plain text was sourced from Project Gutenberg
#' and is divided into elements of up to about 70 characters each.
```

```
#' (Some elements are blank.)
#'
#' @source \url{https://www.gutenberg.org/ebooks/62860}
#' @format A character vector with 14768 elements
"hiddenfoes"

## [1] "hiddenfoes"
```

### III. Results

```
    a. Word Frequencies
```

The below script borrows heavily from the Jane Austin Sentimental Analysis Projects.The idea was taken from the famous Jane Austin sentimental analysis project and used for a different who has complete opposite writing style from Jane. We will be using same approaches used for Jane Austin Books to analyse Nicholas Carter Books.

#We will now start from first scenario to test sentimental analysis

The following packages will be installed.

```
#Note: This script will take a while to run. About 10-12 minutes on a system

install.packages('Rtools', repos = "http://cran.us.r-project.org")
install.packages('tidyverse', repos = "http://cran.us.r-project.org") #
importing, cleaning, visualizing

## package 'cli' successfully unpacked and MD5 sums checked
## package 'tidyverse' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('stringr', repos = "http://cran.us.r-project.org")

## package 'stringr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('tidytext', repos = "http://cran.us.r-project.org") #
working with text

## package 'tidytext' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('tidyr', repos = "http://cran.us.r-project.org")
```

```
## package 'tidyr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('ggplot2', repos = "http://cran.us.r-project.org")

## package 'ggplot2' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('reshape2', repos = "http://cran.us.r-project.org")

## package 'reshape2' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('gridExtra', repos = "http://cran.us.r-project.org") # extra
plot options

## package 'gridExtra' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('grid', repos = "http://cran.us.r-project.org") # extra plot
options
install.packages('keras', repos = "http://cran.us.r-project.org") # deep
learning with keras

## package 'keras' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('RColorBrewer', repos = "http://cran.us.r-project.org")

## package 'RColorBrewer' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('dplyr', repos = "http://cran.us.r-project.org")

## package 'dplyr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages('wordcloud', repos = "http://cran.us.r-project.org") #
visualising text
```

```
## package 'wordcloud' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages("textdata", repos = "http://cran.us.r-project.org")

## package 'textdata' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

#if the mentiond packages under tidyverse are not updated to latest version
#then tidyverse might fail to load
#so please update the said packages ggplot2, dplyr, readr, purrr, tibble,
stringr, forcats.
library(tidyverse) #would have to update rlang and vctrs version if not
already
library(tidyverse)
library(stringr)
library(gridExtra)
library(dplyr)
library(grid) # extra plot options
library(keras)
library(tidytext)
library(tidyr)
library(ggplot2) #would have to update vctrs version if not already
library(reshape2)
library(RColorBrewer)
library(wordcloud)
```

We can understand a particular writing style or language used by a writer from looking at the word frequencies of the book. Word Frequency is simply finding how many times a word is repeated among a group of words.

```
#the total number of words per novel

carter_words <- carter_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word, sort = TRUE)

total_words <- carter_words %>%
  group_by(book) %>%
  summarize(total = sum(n))
```

the number of occurrences of words

```
# “n” is the total count of each word

carter_words <- left_join(carter_words, total_words)
```

```
## # A tibble: 6 × 4
##   book              word      n total
##   <fct>             <chr> <int> <int>
## 1 A Stolen Name     the    3549 67979
## 2 The Twin Mystery the    3162 51797
## 3 The Hidden Foes   the    2956 54547
## 4 A Stolen Name     to     2031 67979
## 5 A Stolen Name     that   1938 67979
## 6 A Stolen Name     of     1687 67979
```

The above table shows that the most commons words in Carter novels are words like “the”, “of”, and other such closed case words.

This can also be shown in a histogram.

This shows a similar trajectory across all three novels in which there is a downward slope to the right. The figure above depicts that these words not only repeat in own novel but are same for other two novels. This is to be expected, but the words found are meaningful in context to with the story.

To find more useful word frequencies. Usully the words that are used most are no significance to the story context So we will look for words that are not used much often by finding tf-idf values

```
#tf = n/total
#idf = ln(1/tf)
#tf-idf = tf * idf

book_tf_idf <- carter_words %>%
  bind_tf_idf(word, book, n)
```

```
## # A tibble: 6 × 7
##   book              word      n total     tf   idf tf_idf
##   <fct>             <chr> <int> <int>  <dbl> <dbl>  <dbl>
## 1 A Stolen Name     the    3549 67979 0.0522     0      0
## 2 The Twin Mystery the    3162 51797 0.0610     0      0
## 3 The Hidden Foes   the    2956 54547 0.0542     0      0
## 4 A Stolen Name     to     2031 67979 0.0299     0      0
## 5 A Stolen Name     that   1938 67979 0.0285     0      0
## 6 A Stolen Name     of     1687 67979 0.0248     0      0
```

Now of we arrange the dataset in descending order, we will find words that are used not much often

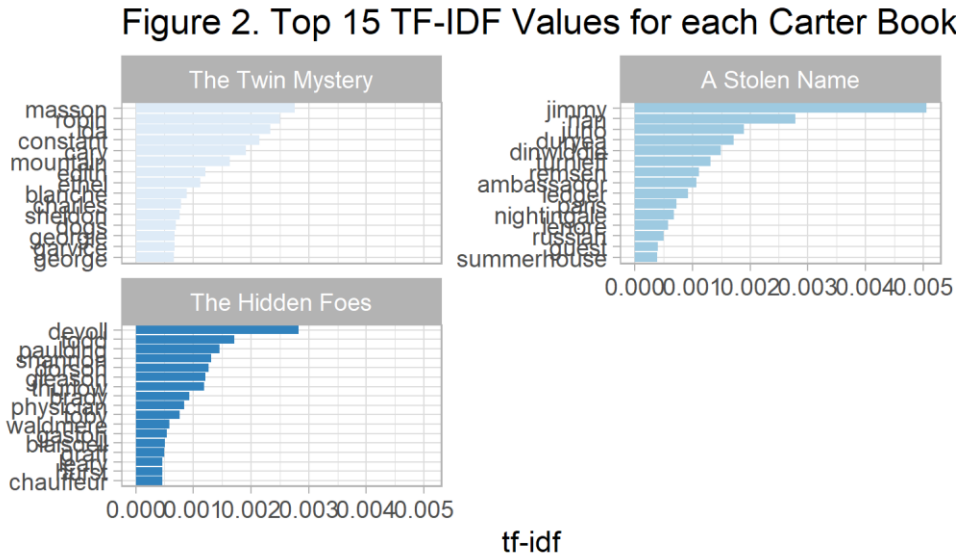
```
head(book_tf_idf %>%
  arrange(desc(tf_idf)))
```

```
## # A tibble: 6 × 7
##   book              word       n total      tf   idf  tf_idf
##   <fct>             <chr>  <int> <int>   <dbl> <dbl>   <dbl>
## 1 A Stolen Name     jimmy    313 67979 0.00460  1.10 0.00506
```

```
## 2 The Hidden Foes  devoll   140 54547 0.00257  1.10 0.00282
## 3 A Stolen Name    nan      172 67979 0.00253  1.10 0.00278
## 4 The Twin Mystery masson   130 51797 0.00251  1.10 0.00276
## 5 The Twin Mystery robin    118 51797 0.00228  1.10 0.00250
## 6 The Twin Mystery ida      110 51797 0.00212  1.10 0.00233
```

Lets plot this observation to get much better insight

```
book_tf_idf %>%
  group_by(book) %>%
  slice_max(tf_idf, n = 15) %>%
  ungroup() %>%
  ggplot(aes(tf_idf, fct_reorder(word, tf_idf), fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_y") +
  labs(x = "tf-idf", y = NULL) +
  scale_fill_brewer() +
  theme_light() +
  ggtitle("Figure 2. Top 15 TF-IDF Values for each Carter Books")
```




We observe that the terms that have the highest tf-idf values are character names and locations.

## III. Results

```
    b. Sentimental Analysis
```

We will first try extracting some sentences to check if the books formatting is valid

Sample of books

```
#extracting random sentence from books
```

```
carter_sentences <- carter_books() %>% group_by(book) %>%
unnest_tokens(sentence, text, token = "sentences") %>% ungroup()
carter_sentences$sentence[30]

## [1] "really, my dear nick, i have a   contempt for the so-called detective
ability."
```

Firstly, we will get 'bing' sentiments. The function get_sentiments() allows us to get specific sentiment lexicons with the appropriate measures for each one.

```
#get sentiments
sentiments

## # A tibble: 6,786 × 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

head(get_sentiments("afinn"))

## # A tibble: 6 × 2
##   word       value
##   <chr>      <dbl>
## 1 abandon       -2
## 2 abandoned     -2
## 3 abandons      -2
## 4 abducted      -2
## 5 abduction     -2
## 6 abductions    -2

head(get_sentiments("bing"))

## # A tibble: 6 × 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate  negative
```

```
head(get_sentiments("nrc"))
```

```
## # A tibble: 6 × 2
##   word      sentiment
##   <chr>     <chr>
## 1 abacus    trust
## 2 abandon   fear
## 3 abandon   negative
## 4 abandon   sadness
## 5 abandoned anger
## 6 abandoned fear
```

chapters numbers and string divided into single element of words

```
#Read chapter numbers and divide the whole string into single elements of
words based on grouping of books

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

head(tidy_data)
```

```
## # A tibble: 6 × 4
##   book              linenumber chapter word
##   <fct>                  <int>   <int> <chr>
## 1 The Twin Mystery           3       0 the
## 2 The Twin Mystery           3       0 twin
## 3 The Twin Mystery           3       0 mystery
## 4 The Twin Mystery           4       0 or
## 5 The Twin Mystery           5       0 a
## 6 The Twin Mystery           5       0 dashing
```

Let's start by checking if sentiments works fine. We will look for most joy words in the book twin mystery

```
#get sentiment ncr - joy
nrc_joy <- get_sentiments("nrc") %>%
  filter(sentiment == "joy")

#let's filter() the data frame with the text from the books for the words
from twin mystery and then count the most joy words in the book
head(tidy_data %>%
  filter(book == "The Twin Mystery") %>%
  inner_join(nrc_joy) %>%
  count(word, sort = TRUE))
```

```
## # A tibble: 6 × 2
##   word        n
##   <chr>   <int>
## 1 found      48
## 2 good       47
## 3 money      40
## 4 love       23
## 5 fortune    21
## 6 finally    20
```

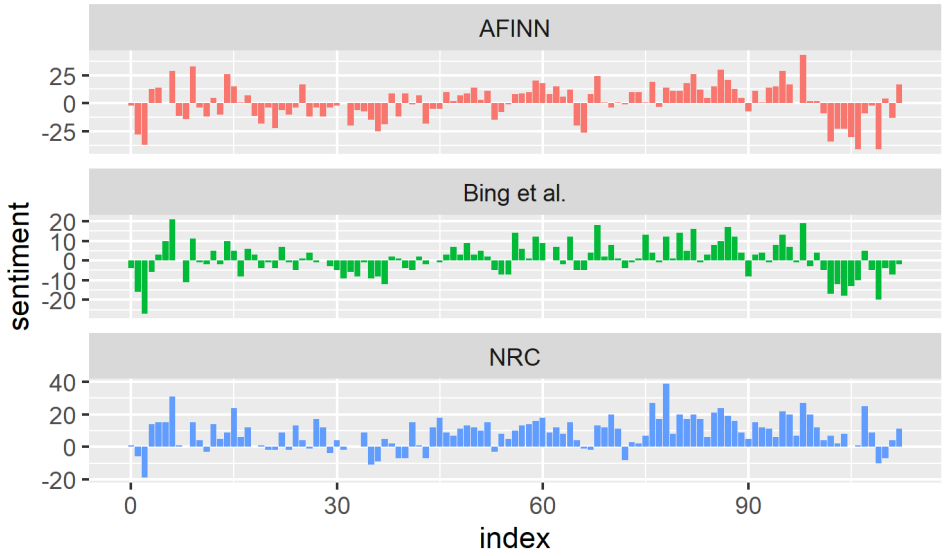
We can also examine how sentiment changes throughout each novel

```
#find a sentiment score for each word using the Bing lexicon and
inner_join().
carter_sentiment <- tidy_data %>%
  inner_join(get_sentiments("bing")) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>%
  mutate(sentiment = positive - negative)

#count up how many positive and negative words there are in defined sections
of each book and plot in a graph
ggplot(carter_sentiment, aes(index, sentiment, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```


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

To understand the better purpose of all three sentiment lexicons, let's compare the three and examine how the sentiment changes across the narrative arc of stolen name

```r
#use filter() to choose only the words from the one novel we are interested
in
astolenname <- tidy_data %>%
  filter(book == "A Stolen Name")

#use inner_join() to calculate the sentiment in different ways.
afinn <- astolenname %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(index = linenumber %/% 80) %>%
  summarise(sentiment = sum(value)) %>%
  mutate(method = "AFINN")

#use integer division (%/%) to define larger sections of text that span
multiple lines, and we can use the same pattern with count(), pivot_wider(),
and mutate() to find the net sentiment in each of these sections of text.
bing_and_nrc <- bind_rows(
  astolenname %>%
    inner_join(get_sentiments("bing")) %>%
    mutate(method = "Bing et al."),
astolenname %>%
    inner_join(get_sentiments("nrc") %>%
                 filter(sentiment %in% c("positive",
                                         "negative"))
    ) %>%
    mutate(method = "NRC")) %>%
  count(method, index = linenumber %/% 80, sentiment) %>%
  pivot_wider(names_from = sentiment,
              values_from = n,
              values_fill = 0) %>%
  mutate(sentiment = positive - negative)

#bind the net sentiment and visualize them
bind_rows(afinn,
          bing_and_nrc) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```




#We see similar dips and peaks in sentiment at about the same places in the novel, but the absolute values are slightly different. The NRC sentiment is high, the Bing et al. sentiment has more variance, the Affin 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.

#From this it look clear that Bing et al. has more systematic sentiments. This analysis will help us in choosing a correct lexicon sentiment.

no of rows in carter book's 3 completed novels

```
#find no of rows in carter book's 3 completed novels
carter_books() %>% group_by(book) %>%
  summarise(total_lines = n())
```

```
## # A tibble: 3 × 2
##   book              total_lines
##   <fct>                   <int>
## 1 The Twin Mystery         7900
## 2 A Stolen Name            9000
## 3 The Hidden Foes          7526
```

positive sentiments and negative sentiments

```
#save positive sentiments and negavtive sentiments

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

negative_senti <- get_sentiments("bing")%>%
  filter(sentiment == "negative")
```

positive sentiments count in twinmystery

```
#count the positive sentiments in the book by words count in book twinmystery

head(tidy_data %>%
  filter(book == "The Twin Mystery") %>%
  semi_join(positive_senti) %>%
  count(word, sort = TRUE))

## Joining with `by = join_by(word)`

## # A tibble: 6 × 2
##   word      n
##   <chr> <int>
## 1 well    104
## 2 great    50
## 3 right    48
## 4 good     47
## 5 like     40
## 6 work     37
```

#We will look same for negative sentiment and then compare the both sentiments usage

negative sentiments count in twinmystery

```
#count the negative sentiments in the book by words count in book twinmystery

head(tidy_data %>%
  filter(book == "The Twin Mystery") %>%
  semi_join(negative_senti) %>%
  count(word, sort = TRUE))

## Joining with `by = join_by(word)`

## # A tibble: 6 × 2
##   word        n
##   <chr>   <int>
## 1 mystery    35
## 2 death      25
## 3 murder     25
## 4 killed     21
## 5 trouble    18
## 6 crime      17
```

seperate positive and negative elements to find the difference for twinmystery

```
#seperate data into positive and negative elements and then difference of
positive and negative elements
bing <- get_sentiments("bing")
twinmystery_sentiment <- tidy_data %>%
  inner_join(bing) %>%
  count(book = "The Twin Mystery" , index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
```

```
  mutate(sentiment = positive - negative)
head(twinmystery_sentiment)

## # A tibble: 6 × 5
##   book              index negative positive sentiment
##   <chr>             <dbl>    <dbl>    <dbl>     <dbl>
## 1 The Twin Mystery     0       56       33       -23
## 2 The Twin Mystery     1       62       27       -35
## 3 The Twin Mystery     2       80       38       -42
## 4 The Twin Mystery     3       43       37        -6
## 5 The Twin Mystery     4       41       50         9
## 6 The Twin Mystery     5       45       42        -3
```

plot the different for twinmystery

```
# To see the data, we plot a bar graph of the difference
# we see starting chapter starts with more on negative sentiments
#otherwise rest chapters are balanced on sentiments
#plot
ggplot(twinmystery_sentiment, aes(index, sentiment, fill = book)) +
  geom_bar(stat = "identity", show.legend = TRUE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```


seperate positive and negative elements to find the difference for stolen name

```
#seperate data into positive and negative elements and then difference of
positive and negative elements
bing <- get_sentiments("bing")
stolenname_sentiment <- tidy_data %>%
  inner_join(bing) %>%
  count(book = "A Stolen Name" , index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
```

```
  mutate(sentiment = positive - negative)
head(stolenname_sentiment)

## # A tibble: 6 × 5
##   book          index negative positive sentiment
##   <chr>         <dbl>    <dbl>    <dbl>     <dbl>
## 1 A Stolen Name     0       56       33       -23
## 2 A Stolen Name     1       62       27       -35
## 3 A Stolen Name     2       80       38       -42
## 4 A Stolen Name     3       43       37        -6
## 5 A Stolen Name     4       41       50         9
## 6 A Stolen Name     5       45       42        -3
```

plot the different for stolenname

```
# To see the data, we plot a bar graph of the difference
# we see starting chapter starts with more on negative sentiments
#otherwise rest chapters are balanced on sentiments
#plot
ggplot(stolenname_sentiment, aes(index, sentiment, fill = book)) +
  geom_bar(stat = "identity", show.legend = TRUE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```


We can see the sentiments are equally divided among both books,which means despite the different in negative and positive sentiments usage for both books. Nicholas carter is able to manage an same balance of sentiments trajectory across all his books. Which is a trace of good writing style.

## III. Results

```
    c. Word Cloud
```

We are trying a new plot style to have more clear idea of the common words usage by carter in his books

```
#wordcloud for max pos and neg words
tidy_data %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "dark green"),
                   max.words = 100)
```


#Now we will try another scenario for same sentimental analysis approach

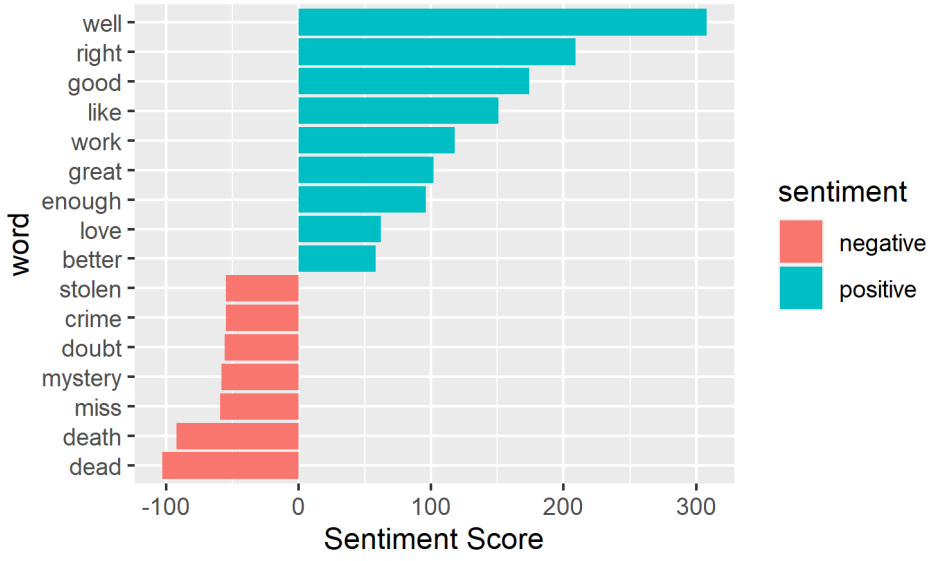
Let's check what are the most common words by emotions used in all three books by carter

```
#most common words
counting_words <- tidy_data %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE)
head(counting_words)

## # A tibble: 6 × 3
##   word  sentiment     n
##   <chr> <chr>     <int>
## 1 well  positive    308
## 2 right positive    209
## 3 good  positive    174
## 4 like  positive    151
```

```
## 5 work  positive    118
## 6 dead  negative    103
```

To further investigate this, we plot the common words to see the which sentiments or words have more weightage

```
#plot1
counting_words %>%
  filter(n > 50) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment))+
  geom_col() +
  coord_flip() +
  labs(y = "Sentiment Score")
```




We will try a different approach to sentimental analysis for the carterbooks. By removing stop words and sorting books chapters wise

```
#for this we would need two packages
#installing packages below
install.packages("widyr", repos = "http://cran.us.r-project.org")

## package 'widyr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

install.packages("viridis", repos = "http://cran.us.r-project.org")

## package 'viridis' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages
```

```
library(viridis)
library(widyr)
```

sort books chapter wise

```
#sort books by chapter wise
sorted_books <- carter_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                                                 ignore_case = TRUE)))) %>%
  ungroup()

#show sorted books
head(sorted_books)
```

```
## # A tibble: 6 × 4
##   text                                          book            linen…¹
chapter
##   <chr>                                         <fct>             <int>
<int>
## 1 ""                                            The Twin Myste…       1
0
## 2 ""                                            The Twin Myste…       2
0
## 3 "                          THE TWIN MYSTERY;" The Twin Myste…       3
0
## 4 "                                 OR,"        The Twin Myste…       4
0
## 5 "                           A Dashing Rescue" The Twin Myste…       5
0
## 6 ""                                            The Twin Myste…       6
0
## # … with abbreviated variable name ¹linenumber
```

```
#tidy the sorted books
tidy_data <- sorted_books %>% unnest_tokens(word,text)
head(tidy_data)
```

```
## # A tibble: 6 × 4
##   book             linenumber chapter word
##   <fct>                 <int>   <int> <chr>
## 1 The Twin Mystery          3       0 the
## 2 The Twin Mystery          3       0 twin
## 3 The Twin Mystery          3       0 mystery
## 4 The Twin Mystery          4       0 or
## 5 The Twin Mystery          5       0 a
## 6 The Twin Mystery          5       0 dashing
```

We will remove stop words the sorted books

```
#remove stop words from the books
data("stop_words")
tidy_data <- tidy_data %>% anti_join(stop_words)

#count the times words repeating in the books
head(tidy_data %>% count(word, sort = TRUE))
```

```
## # A tibble: 6 × 2
##   word          n
##   <chr>     <int>
## 1 nick        856
## 2 carter      824
## 3 patsy       499
## 4 chick       483
## 5 detective   335
## 6 time        321
```

We will get sentiments and plot the sentiments based on sentiment score

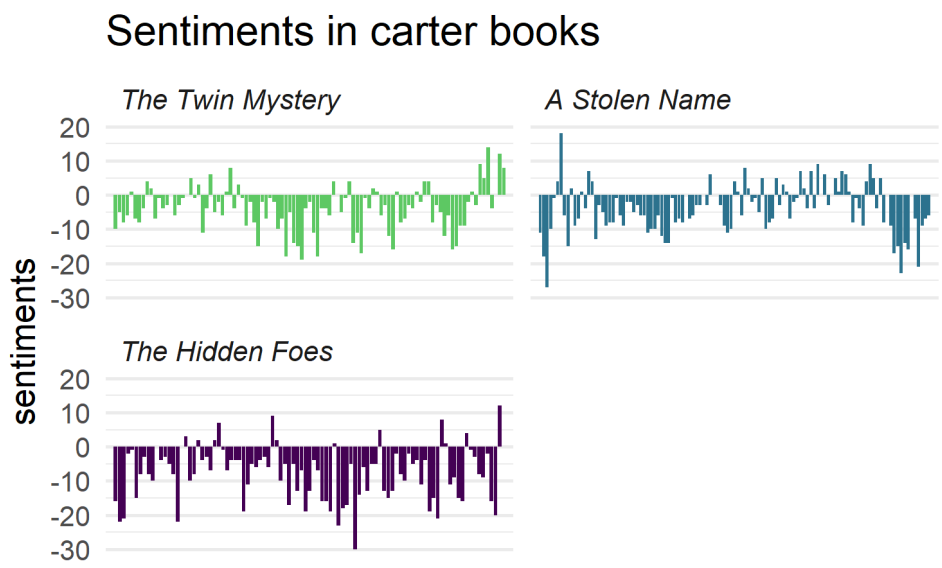
```
#getting sentiments
bing <- get_sentiments("bing")

#setting sentiment score in carter books by chapter index wise
cartersentiment <- tidy_data %>%
  inner_join(bing) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
head(cartersentiment)
```

```
## # A tibble: 6 × 5
##   book             index negative positive sentiment
##   <fct>            <dbl>    <dbl>    <dbl>     <dbl>
## 1 The Twin Mystery     0       13        3       -10
## 2 The Twin Mystery     1       10        5        -5
## 3 The Twin Mystery     2       18       10        -8
## 4 The Twin Mystery     3        9        3        -6
## 5 The Twin Mystery     4        7        8         1
## 6 The Twin Mystery     5       12        5        -7
```

Now we will plot the observations

```
#plotting sentiments on basis of sentiment score
cartersentiment %>% ggplot(aes(index, sentiment, fill = book)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x") +
  labs(title = "Sentiments in carter books", y = "sentiments") +
  theme_minimal(base_size = 13) +
  scale_fill_viridis(end = 0.75, discrete = TRUE, direction = -1)+
  scale_x_discrete(expand = c(0.02,0))+
  theme(strip.text = element_text(hjust = 0))+
```

```
  theme(strip.text = element_text(face = "italic"))+
  theme(axis.title.x =  element_blank())+
  theme(axis.ticks.x = element_blank())+
  theme(axis.text.x = element_blank())
```




#Based on the plot graph trajectory, every book has distinct sentiment score range. Hidden Foes is bend more toward negative score with little positive difference.Twin Mystery and Stolen Name has a equal balance of sentiment words both negative and positive. Negative sentiments also include thrilling or scary emotions or mystery elements. This tells Carter is good at keeping his readers on edge with thrilled mystery books. Giving some positive elements too to keep readers engaged.

The below script borrows heavily from the fantastic book 'Deep Learning with R' by Francois Chollet and J.J. Allaire

## IV. Conclusion

```
    a. Naive Prediction
```

#We will be using some test train data, deep learning approach to try our last approach to sentimental analysis.

Before doing actual prediction on the data set, we will check if the data set is valid for sentiments estimation and can be trusted with the power of model to predict data. By seeing how close train test data are, we will move forward on basis of results.

```
#for this we would need grid base packages
install.packages("grid",repos = "http://cran.us.r-project.org" )
library(grid)

#sort books
sorted_books <- carter_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
```

```
        chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                                                ignore_case = TRUE)))) %>%
  ungroup()
```

Let's have a look at some high level comparisons between the train and test set. We'll use tidytext to split phrases into n-grams and see what we find both with (left) and without (right) stop words.

```
#Prepare test and train data
traindata <- sorted_books %>% filter(book == "The Hidden Foes")
testdata <- sorted_books %>% filter(book == "The Twin Mystery")

# Combine
train = traindata %>% mutate(Split = "train")
test = testdata %>% mutate(Split = "test")
full = data.frame(rbind(train %>% select(-book), test %>% select( -book)))
head(full)
```

```
##                    text linenumber chapter Split
## 1                                 1       0 train
## 2                                 2       0 train
## 3                                 3       0 train
## 4                                 4       0 train
## 5 Transcriber's Notes:            5       0 train
## 6                                 6       0 train
```

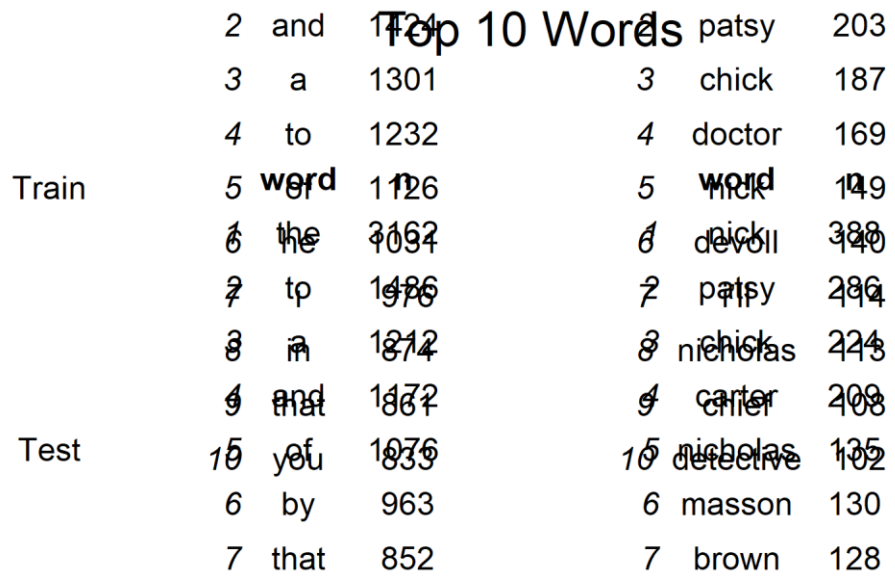
## Top words ——————————————————————————

### Have a look at the most common words (having removed stop words)

```
#top_words_train
top_words_train = full %>%
  filter(Split == "train") %>%
  unnest_tokens(output = word, input = text) %>%
  group_by(word) %>%
  summarise(n = n()) %>%
  arrange(desc(n))

#top_words_test
top_words_test = full %>%
  filter(Split == "test") %>%
  unnest_tokens(output = word, input = text) %>%
  group_by(word) %>%
  summarise(n = n()) %>%
  arrange(desc(n))
```

## Plot the top 10 words for train/test with and without stop words

```
grobs = list(
  tableGrob(head(top_words_train,10), theme = ttheme_minimal()),
  tableGrob(head(top_words_train %>% anti_join(stop_words),10), theme =
ttheme_minimal()),
  tableGrob(head(top_words_test,10), theme = ttheme_minimal()),
  tableGrob(head(top_words_test %>% anti_join(stop_words),10), theme =
ttheme_minimal())
)

lg <- tableGrob(c("", "Train", "Test"), theme= ttheme_minimal())
rg <- arrangeGrob(grobs = grobs, ncol=2,
                  top = textGrob("Top 10 Words",gp=gpar(fontsize=18)))
grid.newpage()
grid.draw(cbind(lg, rg, size = "last"))
```




#Let's make some changes and remove some of the weird "lrb" words. # Adjustments ——————————————————————

## There are a few odd things in the data based on the above that I want to make adjustments for

```
full = full %>% mutate(
  text = gsub(" n't"," not", tolower(text)),
  text = gsub("he 's","he is", tolower(text)),
  text = gsub("she 's","she is", tolower(text)),
  text = gsub("what 's","what is", tolower(text)),
  text = gsub("that 's","that is", tolower(text)),
  text = gsub("there 's","there is", tolower(text)),
  text = gsub("-lrb-"," ", tolower(text)),
  text = gsub("-rrb-"," ", tolower(text)),
```

```
  # Going to remove all instances of "'s" that remain (nearly always
possession)
  # This way we retain the immediate connection between the possession and
possessor in our sequence
  # Otherwise we will end up padding it with zeros and lose some information

  text = gsub(" 's "," ", tolower(text))
)
```

Let's now take a look at some interesting distributions for both the train and test set. Possible things to consider:

How many words per sentence? How many words per phrase? How many phrases per sentence? How many characters per phrase?

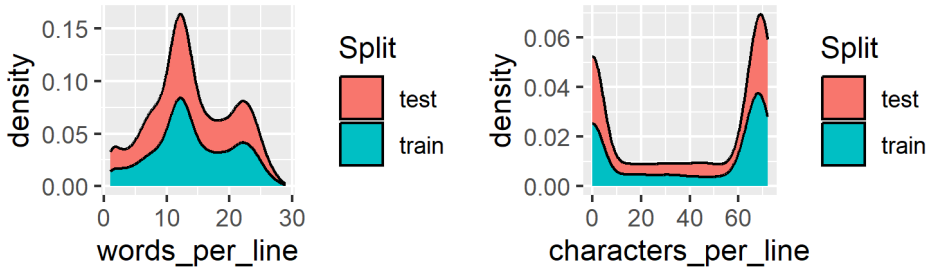
## Visualise ——————————————————————

## Create some summary statistics

```
chapter_summaries = full %>%
  unnest_tokens(output = word, input = text) %>%
  group_by(chapter) %>%
  summarise(
    words_per_chapter = n_distinct(word),
    lines_per_chapter = n_distinct(linenumber)
  )

lines_summaries = full %>%
  unnest_tokens(output = word, input = text) %>%
  group_by(linenumber) %>%
  summarise(
    words_per_line = n_distinct(word)
  )


full_summaries = full %>%
  left_join(chapter_summaries, c("chapter" = "chapter")) %>%
  left_join(lines_summaries, c("linenumber" = "linenumber")) %>%
  mutate(
    characters_per_line = nchar(text)
  )
head(full_summaries)
```

```
##                   text linenumber chapter Split words_per_chapter
## 1                                1       0 train              5344
## 2                                2       0 train              5344
## 3                                3       0 train              5344
## 4                                4       0 train              5344
```

```
## 5 transcriber's notes:          5       0 train              5344
## 6                               6       0 train              5344
##   lines_per_chapter words_per_line characters_per_line
## 1              5550             NA                    0
## 2              5550             NA                    0
## 3              5550              3                    0
## 4              5550              1                    0
## 5              5550              5                   20
## 6              5550             NA                    0

a = ggplot(full_summaries, aes(words_per_line,fill = Split)) +
  geom_density(position = "stack")

b = ggplot(full_summaries, aes(characters_per_line,fill = Split)) +
  geom_density(position = "stack")
grid.arrange(ncol = 2, nrow = 2, a, b)
```




Distributions look sufficiently similar with respect to these features, so I'm happy to say that our training set is representative.

## III. Conclusion

```
    a. Cross Validation with confusion Matrix

# Initial set up.
install.packages("textutils",repos = "http://cran.us.r-project.org" )

## package 'textutils' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

library(textutils)
install.packages("RTextTools",repos = "http://cran.us.r-project.org" )
```

```
## package 'RTextTools' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

library(RTextTools)
install.packages("tm", repos = "http://cran.us.r-project.org")

## package 'tm' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\disha\AppData\Local\Temp\RtmpSUjljv\downloaded_packages

library(tm)
```

We will be cross validating our data through confusion matrix. Do final evaluation on our data set, to see how much this data set model can have power to generalize its estimations of sentiments in books.

```
# Initial set up.

prepdata <- tidy_data %>%
  inner_join(bing, by = "word") %>%
  mutate(sentiment = ifelse(is.na(sentiment), "neutral", sentiment)) %>%
  group_by(book) %>%
  ungroup()

#We will mutate the data by giving 0-1 score to sentiment type
carter_data <- prepdata[,!names(prepdata) %in% c("linenumber", "chapter")]
carter_data <- carter_data %>% mutate(score = ifelse(sentiment ==
"positive",1, 0))%>%
  group_by(book) %>%
  ungroup()
```

We will shuffle the data now so that any default sorting doesn't affect our model

```
# Set seed
set.seed(1985)

#Shuffle the data
carter_data <- carter_data[sample(nrow(carter_data)),]
```

A Preview of the data set

```
## # A tibble: 6 × 4
##   book             word       sentiment score
##   <fct>            <chr>      <chr>     <dbl>
## 1 A Stolen Name    thoughtful positive      1
## 2 The Twin Mystery weak       negative      0
## 3 The Twin Mystery struck     negative      0
## 4 A Stolen Name    cleverly   positive      1
```

```
## 5 The Twin Mystery trick      negative      0
## 6 The Twin Mystery fortune    positive      1
```

Using create_matrix function, we will pre process the data before fitting the model.

```
#data pre-processing.
mat <- create_matrix(carter_data$word, language="english",
                     removeStopwords=TRUE, removeNumbers=TRUE,
                     stemWords=TRUE, weightTfIdf)
```

We will now cross-validate data with subset of data set to check the power that our model will have to generalize its estimations of sentiments in tweets. Create container will generates object for cross-validation

```
#cross-validation.
container <- create_container(mat, carter_data$score,
                              trainSize=1:1000,virgin=FALSE)

#perform a 10-fold cross validation
cvres <- cross_validate(container, nfold=10, algorithm="SVM", seed=1985)

## Fold 1 Out of Sample Accuracy = 0.7215909
## Fold 2 Out of Sample Accuracy = 0.7247706
## Fold 3 Out of Sample Accuracy = 0.7365591
## Fold 4 Out of Sample Accuracy = 0.7109005
## Fold 5 Out of Sample Accuracy = 0.7025641
## Fold 6 Out of Sample Accuracy = 0.7466667
## Fold 7 Out of Sample Accuracy = 0.704918
## Fold 8 Out of Sample Accuracy = 0.7238095
## Fold 9 Out of Sample Accuracy = 0.6753927
## Fold 10 Out of Sample Accuracy = 0.7268293
```

The results are for each of the ten parts as the validation set. As you see above, accuracy is rather moderate, between 0.7 and 0.8. Given that we have only used 1000 rows for the cross-validation, we can expect the final model to be a bit more accurate.

As we have idea of the performance of our model, we can noe fit the data for test set.

```
#we take the first 80% of the data as the training dataset and the last 20%
as the test dataset

trainids <- seq(1, floor(nrow(carter_data)*0.8))
testids <- seq(floor(nrow(carter_data)*0.8)+1, nrow(carter_data))

container <- create_container(mat, carter_data$score,
                              trainSize=trainids,virgin=FALSE)

models <- train_models(container, algorithms="SVM")
```

Now we can see how classifier behaves with two sentiment examples

```
#we take the first 80% of the data as the training dataset and the last 20%
as the test dataset

texts <- c("sad", "happy")
newmatrix <- create_matrix(texts, language="english",
                           removeStopwords=TRUE, removeNumbers=TRUE,
                           stemWords=TRUE, weightTfIdf, originalMatrix = mat)
#The function predict() of the resulting model can now be applied to this
matrix to classify each row according to the model:
predict(models[[1]], newmatrix)

## 1 2
## 0 1
## Levels: 0 1
```

You see in the result that the sad sentiment is classified as 0, while the happy one as 1

The above example is nice, but we need to be more formal with the evaluation. We can use the test data set to do this.

```
#we build a term-document matrix with all the texts in the test data set:
texts <- carter_data$word[testids]
trueclass <- carter_data$score[testids]
testmatrix <- create_matrix(texts, language="english",
                            removeStopwords=TRUE, removeNumbers=TRUE,
                            stemWords=TRUE, weightTfIdf, originalMatrix =
mat)

#we can run the classifier over this resulting matrix
results = predict(models[[1]], testmatrix)
table(trueclass, results)

##          results
## trueclass   0   1
##         0 854   0
##         1 197 335
```

We can measure this better if we calculate accuracy, precision, and recall. We will do final evaluation with confusion matrix .

```
#accuracy sentiment
sum(trueclass==results)/length(results)

## [1] 0.8578644

#precision sentiment
sum(trueclass==results & results==1)/sum(results==1)

## [1] 1

#recall sentiment
sum(trueclass==results & trueclass==1)/sum(trueclass==1)
```

```
## [1] 0.6296992
```

As you see, the precision is high but the recall is low. The model has learned well from the examples in its training data

### IV. Conclusion

```
    b. Final Conclusion
```

A model that produces no false positives has a precision of 1.0. A perfect precision score of 1.0 means that every result retrieved by a search was relevant

Precision-Recall values can be very useful to understand the performance of a specific algorithm and also helps in producing results based on the requirements.

Sentiment analysis provides a way to understand the attitudes and opinions expressed in texts. In this chapter, we explored how to approach sentiment analysis using tidy data principles; when text data is in a tidy data structure, sentiment analysis can be implemented as an inner join. We can use sentiment analysis to understand how a narrative arc changes throughout its course or what words with emotional and opinion content are important for a particular text.

## Citations


Kira Horiuchi. 2021. The MovieLens Datasets: Jane Austen Novels: Word Frequency and Sentiment Analysis.=https://rpubs.com/kmkhoriuchi/849274

TOM AINDOW.2017.Deep Learning with R: Sentiment Analysis.https://www.kaggle.com/code/taindow/deep-learning-with-r-sentiment-analysis

Julia Silge and David Robinson.2022."Text Mining with R: A Tidy Approach".https://www.tidytextmining.com/sentiment.html

Dr. David Garcia.2023.Training supervised text models in R.https://dgarcia-eu.github.io/SocialDataScience/3_Affect/036_SupervisedTextClassification/SupervisedTextClassification.html