Exploring Sentiment Analysis: A Case Study of IMDB Movie Reviews

Devin Fonseca, Rabail Adwani, and Keanan Milton

2023-03-27

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0.9000     v purrr   1.0.1
## v tibble  3.1.8          v dplyr   1.1.0
## v tidyr   1.3.0          v stringr 1.5.0
## v readr   2.1.3          v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidytext)
library(textstem)
```

```
## Warning: package 'textstem' was built under R version 4.2.3

## Loading required package: koRpus.lang.en

## Warning: package 'koRpus.lang.en' was built under R version 4.2.3

## Loading required package: koRpus

## Warning: package 'koRpus' was built under R version 4.2.3

## Loading required package: sylly

## Warning: package 'sylly' was built under R version 4.2.3

## For information on available language packages for 'koRpus', run
##
##   available.koRpus.lang()
##
## and see ?install.koRpus.lang()
##
##
## Attaching package: 'koRpus'
##
## The following object is masked from 'package:readr':
##
##     tokenize
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.2.3

##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.3

## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(e1071)
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.2.3
```

```r
library(tm)
```

```
## Loading required package: NLP
##
## Attaching package: 'NLP'
##
## The following object is masked from 'package:ggplot2':
##
##     annotate
##
##
## Attaching package: 'tm'
##
## The following object is masked from 'package:koRpus':
##
##     readTagged
```

```
reviews_df <- read_csv("F:/MSDS/Statistical Computing/Project/Data/IMDBDataset.csv")
```

```
## Rows: 50000 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): review, sentiment
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
table(reviews_df$sentiment)
```

```
##
## negative positive
##    25000    25000
```

```
set.seed(129)
data <- reviews_df



# Convert reviews into tidy text format
tidy_reviews <- data %>%
  unnest_tokens(word, review)

# Remove stop words
data_stop_words <- stop_words
tidy_reviews_clean <- tidy_reviews %>%
  anti_join(data_stop_words, by = "word")

# Stem words after removing stop words
tidy_reviews_clean$word <- sapply(tidy_reviews_clean$word, function(x) {
  paste0(stem_words(words(x)), collapse = " ")
})
```

Here we load the necessary libraries and data and begin the preprocessing stage.Text data requires careful preprocessing to ensure that our model can make the most of it. We remove punctuation and stop words because they're common and do not provide valuable information for sentiment analysis. Stemming is performed to reduce the dimensionality of our data and to group together different forms of the same word, which helps in capturing the sentiment effectively.

Next we will calculate the most frequent words for both positive and negative reviews.

```
# Calculate term frequencies for each word in positive and negative reviews
word_counts <- tidy_reviews_clean %>%
  group_by(sentiment, word) %>%
  summarise(term_frequency = n(), .groups = "drop") %>%
  arrange(sentiment, desc(term_frequency))

# Identify the most frequent associated words for positive and negative movie reviews
top_words <- word_counts %>%
  group_by(sentiment) %>%
  top_n(10, term_frequency)

top_words
```

```
## # A tibble: 20 x 3
## # Groups:   sentiment [2]
##    sentiment word    term_frequency
##    <chr>     <chr>            <int>
##  1 negative  br              103997
##  2 negative  movi             57872
##  3 negative  film             44044
##  4 negative  time             15305
##  5 negative  watch            14960
##  6 negative  bad              14770
##  7 negative  charact          13993
##  8 negative  scene            11357
##  9 negative  stori            11042
## 10 negative  act              10565
## 11 positive  br               97954
## 12 positive  film             49709
## 13 positive  movi             44345
## 14 positive  time             16559
## 15 positive  stori            14111
## 16 positive  charact          13746
## 17 positive  watch            12879
## 18 positive  love             12440
## 19 positive  scene            10019
## 20 positive  plai              9947
```

The top words in this dataset aren't surprising, they are mostly movie related words. One thing we weren't expecting is the word "br". The next section we will try to identify what this word is and if it is worth removing.

```
#Trying to figure out the word "br"

#Rerun this without removing stop words and without stemming
word_counts2 <- tidy_reviews %>%
  group_by(sentiment, word) %>%
  summarise(term_frequency = n(), .groups = "drop") %>%
```

```r
  arrange(sentiment, desc(term_frequency))

# Identify the most frequent words/features for positive and negative movie reviews
top_words2 <- word_counts2 %>%
  group_by(sentiment) %>%
  top_n(10, term_frequency)

top_words2
```

```
## # A tibble: 20 x 3
## # Groups:   sentiment [2]
##    sentiment word  term_frequency
##    <chr>     <chr>          <int>
##  1 negative  the           326261
##  2 negative  a             158271
##  3 negative  and           147703
##  4 negative  of            137287
##  5 negative  to            136784
##  6 negative  br            103997
##  7 negative  is             99230
##  8 negative  in             87486
##  9 negative  this           81209
## 10 negative  i              81156
## 11 positive  the           340628
## 12 positive  and           176555
## 13 positive  a             163896
## 14 positive  of            152093
## 15 positive  to            131287
## 16 positive  is            111810
## 17 positive  in             99176
## 18 positive  br             97954
## 19 positive  it             77890
## 20 positive  i              72440
```

"br" is present despite not removing stop words or stems. We can conclude that "br" is a word found in the dataset that isn't an artifact of stemming or removing stop words.

Next is finding out the context of how "br" fits in the text by identifying which sentences it appears in.

```r
# Load library
library(stringr)

# Define a function to extract sentences containing the target word
extract_sentences <- function(text, target_word) {
  sentences <- str_split(text, boundary("sentence"))[[1]]
  target_sentences <- sentences[str_detect(sentences, regex(paste0("\\b", target_word, "\\b"), ignore_ca
  return(target_sentences)
}

# Find sentences containing the word "br"
target_word <- "br"
sentences_with_target_word <- data %>%
  mutate(sentences = map(review, extract_sentences, target_word = target_word)) %>%
```

```r
  select(sentiment, review, sentences) %>%
  unnest(sentences)

# View sentences containing the word "br"
sentences_with_target_word
```

```
## # A tibble: 85,169 x 3
##    sentiment review                                                       sente~1
##    <chr>     <chr>                                                        <chr>
##  1 positive  "One of the other reviewers has mentioned that after watch~ "They ~
##  2 positive  "One of the other reviewers has mentioned that after watch~ "Its i~
##  3 positive  "One of the other reviewers has mentioned that after watch~ "Aryan~
##  4 positive  "A wonderful little production. <br /><br />The filming te~ "A won~
##  5 positive  "A wonderful little production. <br /><br />The filming te~ "A mas~
##  6 positive  "I thought this was a wonderful way to spend time on a too~ "While~
##  7 positive  "I thought this was a wonderful way to spend time on a too~ "While~
##  8 negative  "Basically there's a family where a little boy (Jake) thin~ "Basic~
##  9 negative  "Basically there's a family where a little boy (Jake) thin~ "I exp~
## 10 positive  "Petter Mattei's \"Love in the Time of Money\" is a visual~ "This ~
## # ... with 85,159 more rows, and abbreviated variable name 1: sentences
```

We found that "br" represents which is a line break in the review. Now that we know what this is we can drop it.

```r
# Remove stop words
data_stop_words <- stop_words
tidy_reviews_clean <- tidy_reviews %>%
  anti_join(data_stop_words, by = "word")


# Filter out the word "br"
tidy_reviews_clean <- tidy_reviews_clean %>%
  filter(word != "br")

# Stem words after removing stop words
tidy_reviews_clean$word <- sapply(tidy_reviews_clean$word, function(x) {
  paste0(stem_words(words(x)), collapse = " ")
})

head(tidy_reviews_clean)
```

```
## # A tibble: 6 x 2
##   sentiment word
##   <chr>     <chr>
## 1 positive  review
## 2 positive  mention
## 3 positive  watch
## 4 positive  1
## 5 positive  oz
## 6 positive  episod
```

In the head of the clean data above, there is a number that has been considered as a word. Lets remove numbers and any special characters in the dataset.

```r
# Removing any numbers or special characters
tidy_reviews_clean <- tidy_reviews_clean %>%
  filter(!grepl("[^[:alpha:]]", word))

head(tidy_reviews_clean)
```

```
## # A tibble: 6 x 2
##   sentiment word
##   <chr>     <chr>
## 1 positive  review
## 2 positive  mention
## 3 positive  watch
## 4 positive  oz
## 5 positive  episod
## 6 positive  hook
```

```r
write.csv(tidy_reviews_clean, "imdb_clean.csv", row.names = FALSE)
```

We will revisit the most frequent words now that the data has been properly cleaned.

```r
# Calculate term frequencies for each word in positive and negative reviews
word_counts <- tidy_reviews_clean %>%
  group_by(sentiment, word) %>%
  summarise(term_frequency = n(), .groups = "drop") %>%
  arrange(sentiment, desc(term_frequency))

# Identify the most strongly associated words/features for positive and negative movie reviews
top_words <- word_counts %>%
  group_by(sentiment) %>%
  top_n(10, term_frequency)

top_words
```

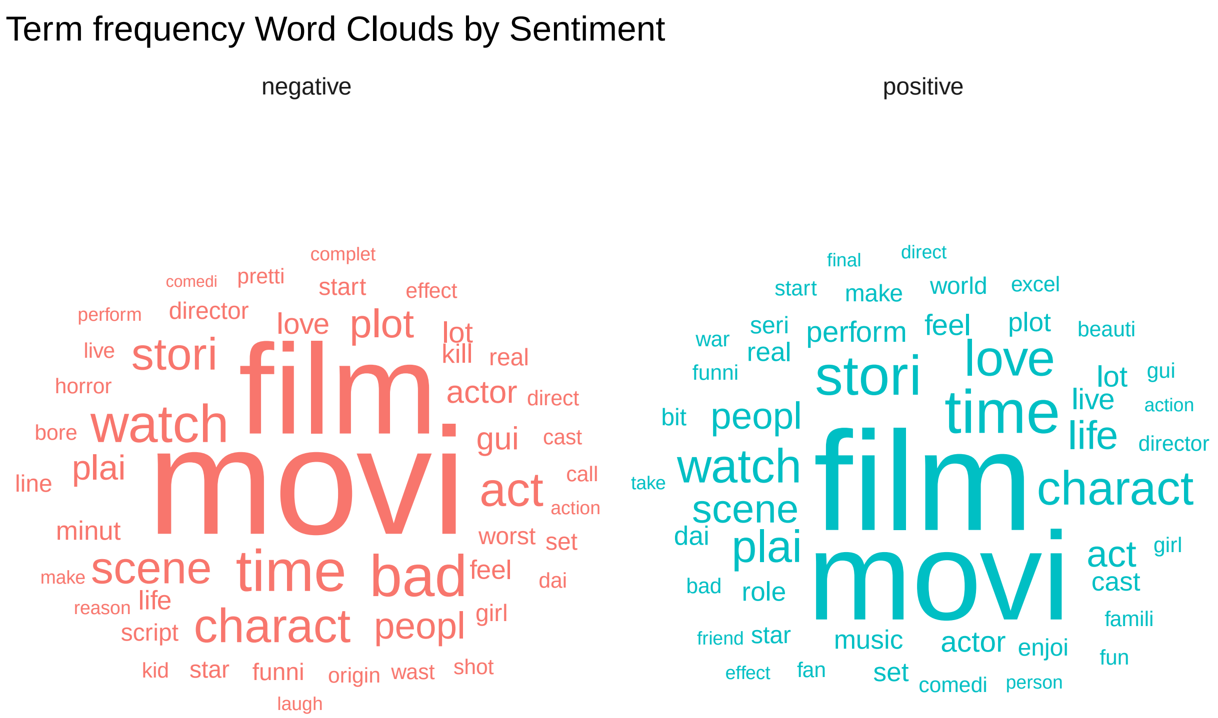
```
## # A tibble: 20 x 3
## # Groups:   sentiment [2]
##    sentiment word    term_frequency
##    <chr>     <chr>            <int>
##  1 negative  movi             57872
##  2 negative  film             44044
##  3 negative  time             15305
##  4 negative  watch            14960
##  5 negative  bad              14770
##  6 negative  charact          13993
##  7 negative  scene            11357
##  8 negative  stori            11042
##  9 negative  act              10565
## 10 negative  peopl             9389
## 11 positive  film             49709
## 12 positive  movi             44345
## 13 positive  time             16559
## 14 positive  stori            14111
## 15 positive  charact          13746
```

```
## 16 positive  watch            12879
## 17 positive  love             12440
## 18 positive  scene            10019
## 19 positive  plai              9947
## 20 positive  peopl             8657
```

```
# Load ggwordcloud library
library(ggwordcloud)

# Create separate word clouds for positive and negative reviews
word_cloud <- word_counts %>%
  top_n(100, term_frequency) %>%
  ggplot(aes(label = word, size = term_frequency, color = sentiment)) +
  geom_text_wordcloud_area() +
  scale_size_area(max_size = 20) +
  theme_minimal() +
  facet_wrap(~sentiment) +
  labs(title = "Term frequency Word Clouds by Sentiment",
       x = NULL, y = NULL)

word_cloud
```

```
## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=12, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
```

```
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels
```



In this word cloud, the size of the words represents their frequency in the dataset. Larger words appear more often in the reviews, while smaller words are less frequent. The words are also colored based on their sentiment—words in the positive sentiment group are blue, while words in the negative sentiment group are red. By looking at this word cloud, you can quickly identify the most frequently used words in positive and negative movie reviews, in this case it appears the most frequently occurring terms are film and movie and they appear in both categories. To see which words are more important in distinguishing between positive and negative movie reviews we will calculate the term frequency-inverse document frequency.

TF-IDF

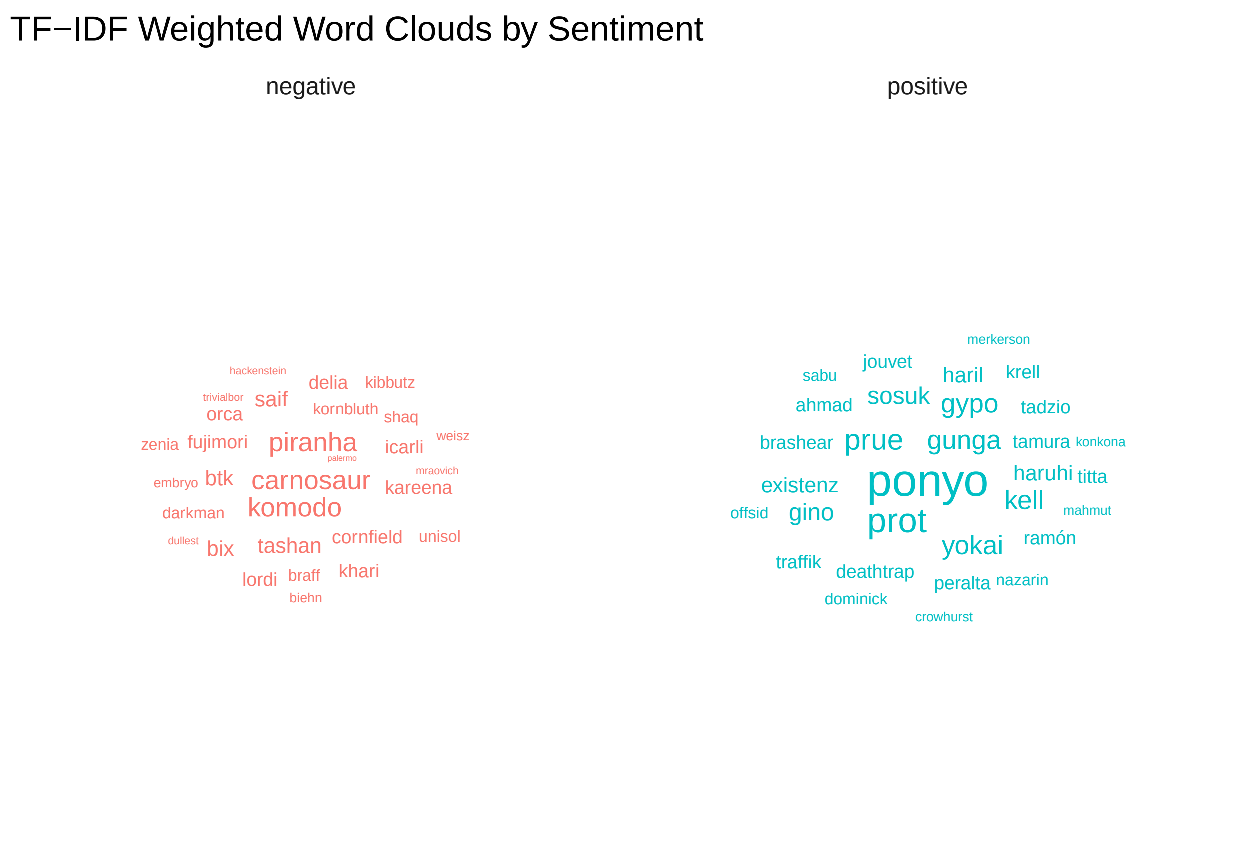
```
tf_idf_reviews <- tidy_reviews_clean %>%
  count(sentiment, word) %>%
  bind_tf_idf(word, sentiment, n) %>%
  arrange(desc(tf_idf))

head(tf_idf_reviews)
```

```
## # A tibble: 6 x 6
##   sentiment word          n        tf   idf    tf_idf
##   <chr>     <chr>     <int>     <dbl> <dbl>     <dbl>
## 1 positive  ponyo       135 0.0000621 0.693 0.0000430
## 2 positive  prot         81 0.0000373 0.693 0.0000258
## 3 negative  carnosaur    67 0.0000323 0.693 0.0000224
## 4 negative  komodo       64 0.0000308 0.693 0.0000214
## 5 negative  piranha      61 0.0000294 0.693 0.0000204
## 6 positive  gunga        61 0.0000281 0.693 0.0000194
```

```r
idf_by_sentiment_word_cloud <- tf_idf_reviews %>%
  group_by(sentiment) %>%
  slice_max(tf_idf, n = 30) %>%
  ggplot(aes(label = word, size = tf_idf, color = sentiment)) +
  geom_text_wordcloud_area() +
  theme_minimal() +
  facet_wrap(~sentiment) +
  labs(title = "TF-IDF Weighted Word Clouds by Sentiment",
       x = NULL, y = NULL)

idf_by_sentiment_word_cloud
```

```
## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=12' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=12' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=12' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=8' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=12' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=8' are unlikely values in pixels
```

```
## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=12' are unlikely values in pixels

## Warning in png(filename = tmp_file, width = gw_pix, height = gh_pix, res =
## dev_dpi, : 'width=16, height=16' are unlikely values in pixels
```



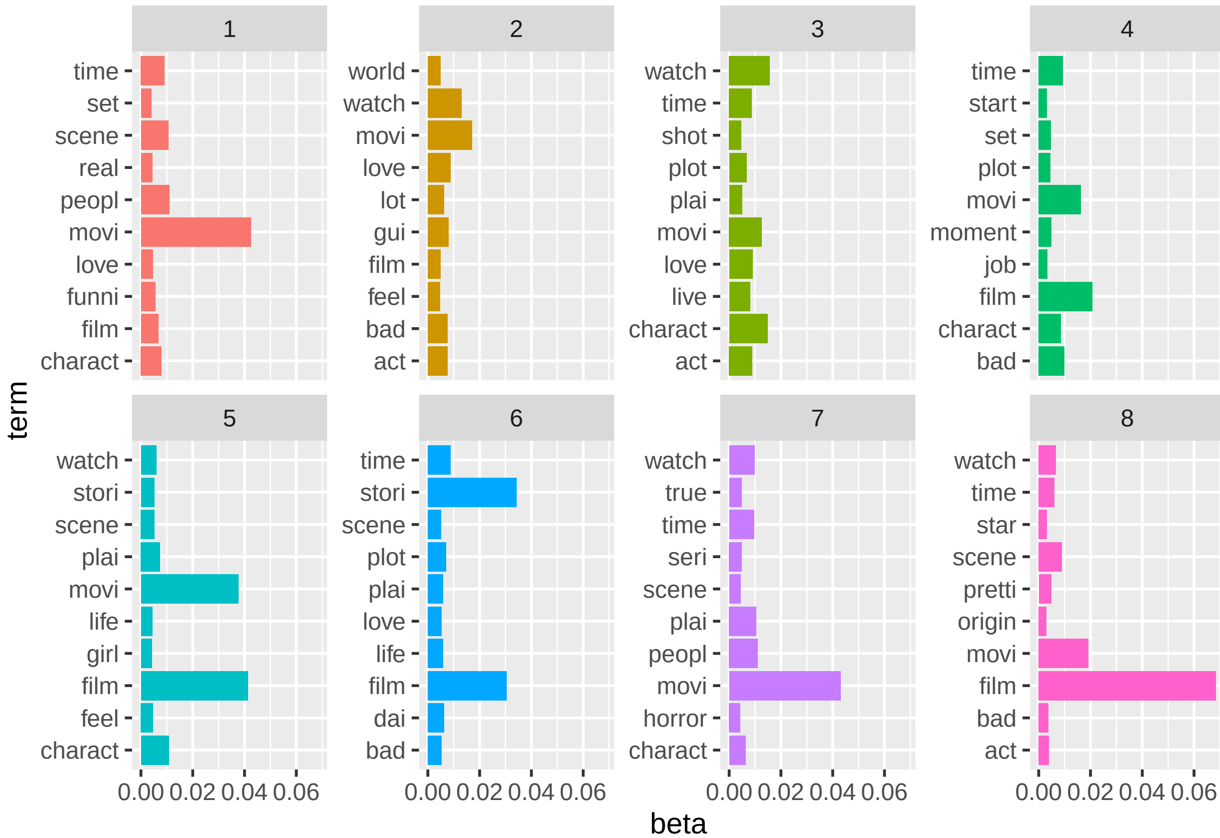
These words could be considered strong indicators of sentiment in movie reviews, as they show a significant difference in usage between positive and negative reviews. The problem we found with the TF-IDF words is they appear to be movie titles or segments of movie titles. It makes sense that the word Ponyo would be a good predictor of positive sentiment as it was an acclaimed movie but these words aren't helpful for predicting sentiment for other movies.

Next we will explore Latent Dirichlet Allocation to see if topics can easily be distinguished in movie reviews.

Latent Dirichlet Allocation

```
library(topicmodels)


# Add review_id to the tidy_reviews_clean dataframe
```

```r
tidy_reviews_clean <- tidy_reviews_clean %>% mutate(review_id = factor(row_number()))

# Count the words for each review
review_word_counts <- tidy_reviews_clean %>%
  group_by(review_id, word) %>%
  summarise(n = n(), .groups = "drop")

# Create the Document-Term Matrix using review_word_counts
document_term_matrix <- review_word_counts %>%
  cast_dtm(document = review_id,
           term = word,
           value = n)

# Run LDA analysis on the Document-Term Matrix
set.seed(29)
reviews_lda <- LDA(document_term_matrix,
                   k = 8,
                   control = list(seed = 29))

# Visualize the topics found
review_topics <- tidy(reviews_lda, matrix = "beta")

top_terms <- review_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  mutate(topic = factor(topic))

top_terms %>%
  ggplot(aes(beta, term, fill = topic)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free_y", ncol = 4)
```



The 8 topics seem to share a lot of the same words and they are all generic terms about movies. There is very little variation in words in each topic. This can be a sign that the model is not able to distinguish between distinct topics or themes within the reviews, this makes sense intuitively as movie reviews have a narrow range of topics. Given the little variation between the 8 topics, we decided pursuing LDA wouldn't be a productive use of time.

The next section focuses on modeling to predict sentiment. We focus and compare two models, Random Forest and XGBoost.

Modeling

```
# Load the necessary libraries
library(quanteda)
```

```
## Warning: package 'quanteda' was built under R version 4.2.3
```

```
## Package version: 3.3.0
## Unicode version: 13.0
## ICU version: 69.1
```

```
## Parallel computing: 8 of 8 threads used.
```

```
## See https://quanteda.io for tutorials and examples.
```

```
##
## Attaching package: 'quanteda'

## The following object is masked from 'package:tm':
##
##     stopwords

## The following objects are masked from 'package:NLP':
##
##     meta, meta<-

## The following objects are masked from 'package:koRpus':
##
##     tokens, types
```

```
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 4.2.3

##
## Attaching package: 'ranger'

## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.0.0 --

## v broom        1.0.3     v rsample      1.1.1
## v dials        1.1.0     v tune         1.0.1
## v infer        1.0.4     v workflows    1.1.3
## v modeldata    1.1.0     v workflowsets 1.0.0
## v parsnip      1.0.4     v yardstick    1.1.0
## v recipes      1.0.5

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x NLP::annotate()          masks ggplot2::annotate()
## x randomForest::combine()  masks dplyr::combine()
## x scales::discard()        masks purrr::discard()
## x dplyr::filter()          masks stats::filter()
## x recipes::fixed()         masks stringr::fixed()
## x yardstick::get_weights() masks keras::get_weights()
## x dplyr::lag()             masks stats::lag()
## x caret::lift()            masks purrr::lift()
## x randomForest::margin()   masks ggplot2::margin()
## x rsample::permutations()  masks e1071::permutations()
## x yardstick::precision()   masks caret::precision()
## x yardstick::recall()      masks caret::recall()
## x yardstick::sensitivity() masks caret::sensitivity()
```

```
## x xgboost::slice()         masks dplyr::slice()
## x yardstick::spec()        masks readr::spec()
## x yardstick::specificity() masks caret::specificity()
## x recipes::step()          masks stats::step()
## x tune::tune()             masks parsnip::tune(), e1071::tune()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```r
# function to clean the data

vectorizer <- function(data){

  # Add unique identifiers for each review
  data <- data %>% mutate(id = row_number())

  # Convert reviews into tidy text format with identifiers
  tidy_reviews <- data %>%
    unnest_tokens(word, review) %>%
    select(id, word)

  # Remove stop words
  tidy_reviews <- tidy_reviews %>%
    anti_join(data_stop_words, by = "word")

  # Remove the word "br"
  tidy_reviews <- tidy_reviews %>% filter(word != "br")

  # Remove numbers and special characters
  tidy_reviews<- tidy_reviews %>%
    filter(!grepl("[^[:alpha:]]", word))

  # Create Document-feature matrices for training and testing data
  tidy_reviews_dfm <- tidy_reviews %>%
    count(id, word) %>%
    cast_dfm(id, word, n)

  # Create training and testing datasets with document-feature matrix and sentiment labels
  tidy_reviews_dfm <- data.frame(sentiment = data$sentiment, as.matrix(tidy_reviews_dfm))

  tidy_reviews_dfm$sentiment <- as.factor(tidy_reviews_dfm$sentiment)

  return(tidy_reviews_dfm)
}

set.seed(123)
imdb_cleaned <- data %>%
  group_by(sentiment) %>%
  sample_n(200) %>%
  ungroup() %>%
  vectorizer()
```

Due to the limitations of R, we are only using 200 movie reviews as part of the modeling exercise. We scrapped the idea of using parsnip package as it doesn't handle large datasets well and can utilize more memory.

Train-test subsets

```r
# Create training and testing split
set.seed(42)
data_split <- initial_split(imdb_cleaned, prop = 0.8, strat=sentiment)
train_data <- training(data_split)
test_data <- testing(data_split)
```

Random Forest

```r
# Fit the model on the training data subset of
imdb_fit_rf <- ranger::ranger(
  sentiment ~ .,
  data = train_data,
  num.trees = 500,
  probability = TRUE,
  importance = "impurity"
)

predictions_probs <- predict(imdb_fit_rf, data = test_data, type = "response")$predictions

predicted_labels <- as.factor(ifelse(predictions_probs[, "positive"] > 0.5, "positive", "negative"))

conf_matrix <- confusionMatrix(predicted_labels, test_data$sentiment)
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       28        8
##   positive       12       32
##
##                Accuracy : 0.75
##                  95% CI : (0.6406, 0.8401)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 4.29e-06
##
##                   Kappa : 0.5
##
##  Mcnemar's Test P-Value : 0.5023
##
##             Sensitivity : 0.7000
##             Specificity : 0.8000
##          Pos Pred Value : 0.7778
##          Neg Pred Value : 0.7273
##              Prevalence : 0.5000
##          Detection Rate : 0.3500
##    Detection Prevalence : 0.4500
```

```
##        Balanced Accuracy : 0.7500
##
##         'Positive' Class : negative
##
```

Using a sample of only 200 movie reviews out of 50,000, we obtained an accuracy rate of 75% with random forest. We believe these results are pretty good for such a small sample size.

In summary, this model has an accuracy of 75% and a kappa of 0.4, indicating a moderate level of agreement between the model predictions and the actual values. It has a fairly balanced sensitivity and specificity, indicating that it performs similarly on both positive and negative classes. However, there's room for improvement, as perfect performance would have these metrics at 1.

Variable Importance

```r
# Extract variable importance scores
var_imp <- imdb_fit_rf$variable.importance

# Print the top 10 most important variables/words
head(var_imp, 10)
```

```
##       acting       actors         anti anticipation          bad        badly
##  0.901695159  0.160751865  0.008239713  0.000000000  1.693133467  0.085395345
##       boring         call         care        cared
##  1.225121809  0.080989080  0.356080748  0.006261058
```

```r
filtered_var_imp <- var_imp[!grepl("^X\\d+", names(var_imp))]

# Print the top 10 most important variables/words without the "X" variables
head(filtered_var_imp, 10)
```

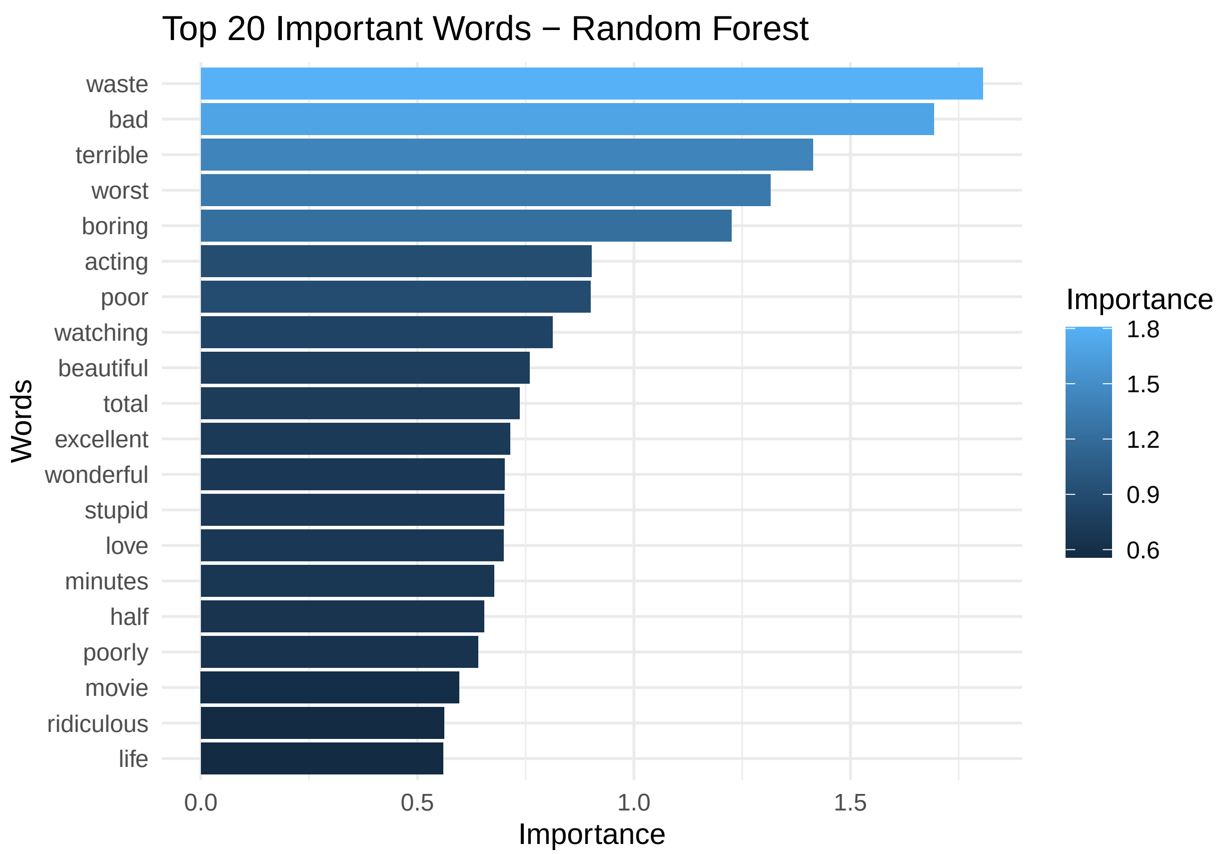
```
##       acting       actors         anti anticipation          bad        badly
##  0.901695159  0.160751865  0.008239713  0.000000000  1.693133467  0.085395345
##       boring         call         care        cared
##  1.225121809  0.080989080  0.356080748  0.006261058
```

```r
# Sort the filtered variable importance in decreasing order
sorted_filtered_var_imp <- sort(filtered_var_imp, decreasing = TRUE)

# Print the top 10 most important variables/words without the "X" variables
head(sorted_filtered_var_imp, 10)
```

```
##     waste       bad  terrible     worst    boring    acting      poor  watching
## 1.8061128 1.6931335 1.4129210 1.3156912 1.2251218 0.9016952 0.9002208 0.8118295
## beautiful     total
## 0.7592939 0.7360453
```

These top 10 words from var_imp represent the words that the model identified as the most important features in predicting sentiment (positive or negative) in the training data. These words are ranked by their importance score, which indicates how much they contribute to the model's accuracy in predicting sentiment.

```r
# Creating variable importance plot for Random Forest
var_imp_df <- data.frame(Word = names(sorted_filtered_var_imp), Importance = sorted_filtered_var_imp)
top_20 <- head(var_imp_df, 20)
ggplot(top_20, aes(x = reorder(Word, Importance), y = Importance, fill = Importance)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Top 20 Important Words - Random Forest", x = "Words", y = "Importance")
```



Out of the various techniques used above (term frequency, TF-IDF, LDA), the words found with the variable importance scores appear to be the words that you would expect to predict sentiment the best. For example terms like "bad", "waste", and "terrible" match our intuition on being good predictors for a negative review.

```r
library(pROC)
```

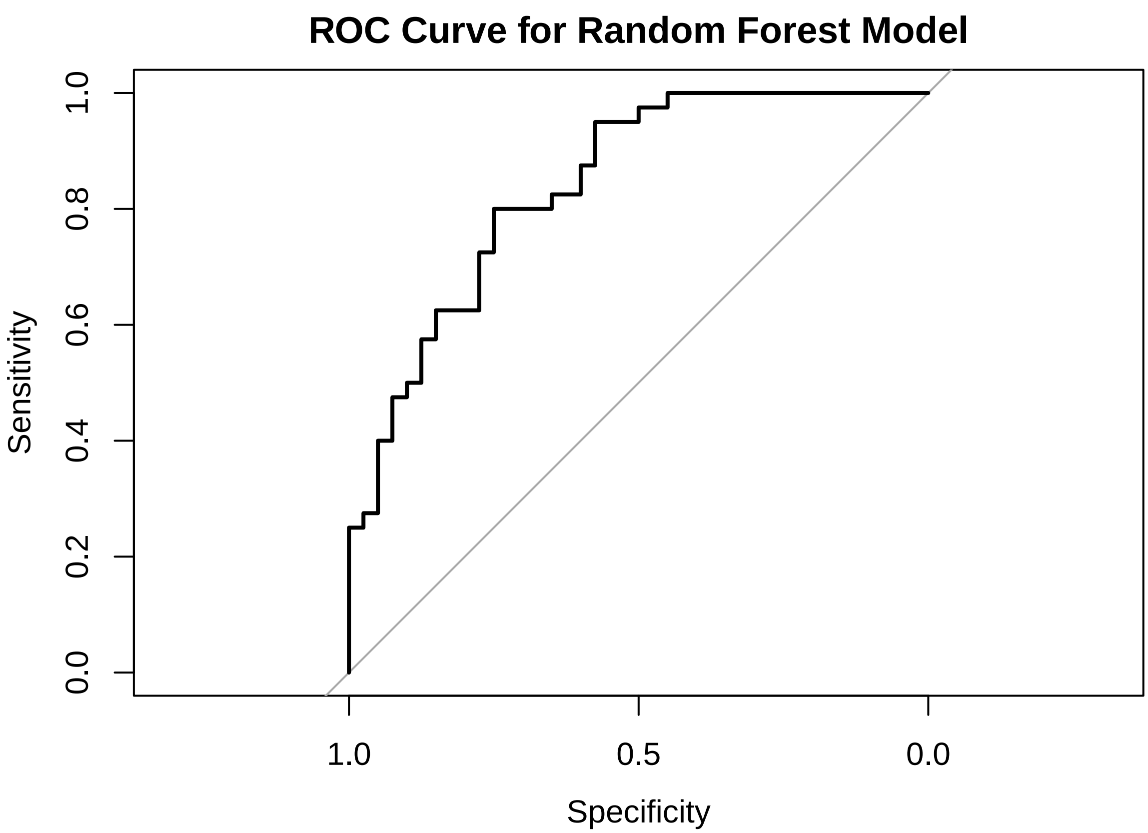
```
## Warning: package 'pROC' was built under R version 4.2.3

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
roc_obj <- roc(test_data$sentiment, predictions_probs[, "positive"])
```

```
## Setting levels: control = negative, case = positive
```

```
## Setting direction: controls < cases
```

```r
plot(roc_obj, main="ROC Curve for Random Forest Model")
```



This curve is about to be expected for the results that we interpreted above. The top-left corner of the ROC space corresponds to a false positive rate of 0 and a true positive rate of 1, which is where a perfect classifier's ROC curve would reach. Our curve for this model is midway between the diagonal and the top-left corner, it suggests that the model's performance is somewhere between moderate and good. It's better than random guessing but it is not excellent and has room for improvement.

Next we will run a xgboost model and compare the results to the random forest model.

xgboost

```r
library(xgboost)

set.seed(876)

# Convert the target to 0 or 1
```

```r
train_data_gbm <- train_data %>% mutate(sentiment = ifelse(sentiment == "positive", 1, 0))
test_data_gbm <- test_data %>% mutate(sentiment = ifelse(sentiment == "positive", 1, 0))

# Transform train and test to predictors only matrix
matrix_predictors.train <- as.matrix(train_data_gbm)[,-1]
matrix_predictors.test <- as.matrix(test_data_gbm)[,-1]

# Set up features and label in a Dmatrix form for xgboost

## Train
pred.train.gbm <- data.matrix(matrix_predictors.train)
imdb.train.gbm <- as.numeric(as.character(train_data_gbm$sentiment))
dtrain <- xgb.DMatrix(data = pred.train.gbm, label=imdb.train.gbm)

## Test
pred.test.gbm <- data.matrix(matrix_predictors.test)
imdb.test.gbm <- as.numeric(as.character(test_data_gbm$sentiment))
dtest <- xgb.DMatrix(data = pred.test.gbm, label=imdb.test.gbm)

# define watchlist
watchlist <- list(train=dtrain, test=dtest)

# define param
param <- list(objective = "binary:logistic", eval_metric = "auc")

# fit XGBoost model and display training and testing data at each round
model.xgb <- xgb.train(param, dtrain, nrounds = 50, watchlist)
```

```
## [1]  train-auc:0.812754  test-auc:0.735938
## [2]  train-auc:0.888965  test-auc:0.766563
## [3]  train-auc:0.901016  test-auc:0.744375
## [4]  train-auc:0.934004  test-auc:0.761563
## [5]  train-auc:0.954766  test-auc:0.749687
## [6]  train-auc:0.959121  test-auc:0.744062
## [7]  train-auc:0.964258  test-auc:0.746563
## [8]  train-auc:0.973652  test-auc:0.743750
## [9]  train-auc:0.982207  test-auc:0.733125
## [10] train-auc:0.985684  test-auc:0.720625
## [11] train-auc:0.989883  test-auc:0.732500
## [12] train-auc:0.991680  test-auc:0.753125
## [13] train-auc:0.992441  test-auc:0.750000
## [14] train-auc:0.993262  test-auc:0.746250
## [15] train-auc:0.994512  test-auc:0.736875
## [16] train-auc:0.995898  test-auc:0.739062
## [17] train-auc:0.996016  test-auc:0.732812
## [18] train-auc:0.996250  test-auc:0.734062
## [19] train-auc:0.996660  test-auc:0.733437
## [20] train-auc:0.997344  test-auc:0.735313
## [21] train-auc:0.997754  test-auc:0.722500
## [22] train-auc:0.997832  test-auc:0.712500
## [23] train-auc:0.997676  test-auc:0.714375
## [24] train-auc:0.998047  test-auc:0.714375
## [25] train-auc:0.998398  test-auc:0.728750
```

```
## [26] train-auc:0.998633  test-auc:0.717500
## [27] train-auc:0.998789  test-auc:0.716250
## [28] train-auc:0.998984  test-auc:0.724375
## [29] train-auc:0.999180  test-auc:0.719375
## [30] train-auc:0.999297  test-auc:0.720625
## [31] train-auc:0.999648  test-auc:0.717500
## [32] train-auc:0.999687  test-auc:0.714375
## [33] train-auc:0.999492  test-auc:0.711250
## [34] train-auc:0.999492  test-auc:0.702500
## [35] train-auc:0.999570  test-auc:0.704375
## [36] train-auc:0.999687  test-auc:0.704375
## [37] train-auc:0.999844  test-auc:0.700625
## [38] train-auc:0.999805  test-auc:0.700000
## [39] train-auc:0.999766  test-auc:0.697500
## [40] train-auc:0.999844  test-auc:0.700000
## [41] train-auc:0.999883  test-auc:0.698750
## [42] train-auc:0.999883  test-auc:0.696875
## [43] train-auc:0.999883  test-auc:0.695625
## [44] train-auc:0.999883  test-auc:0.695625
## [45] train-auc:0.999883  test-auc:0.699375
## [46] train-auc:0.999961  test-auc:0.700000
## [47] train-auc:0.999961  test-auc:0.701250
## [48] train-auc:0.999961  test-auc:0.699375
## [49] train-auc:0.999961  test-auc:0.702500
## [50] train-auc:0.999961  test-auc:0.703750
```

```r
# make predictions on the test set
pred.prob = predict(model.xgb, pred.test.gbm)
prediction <- as.numeric(pred.prob > 0.5)

# confusion matrix
conf_matrix <- confusionMatrix(factor(prediction), factor(imdb.test.gbm))
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23 15
##          1 17 25
##
##                Accuracy : 0.6
##                  95% CI : (0.4844, 0.708)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.04646
##
##                   Kappa : 0.2
##
##  Mcnemar's Test P-Value : 0.85968
##
##             Sensitivity : 0.5750
##             Specificity : 0.6250
##          Pos Pred Value : 0.6053
##          Neg Pred Value : 0.5952
```

```
##              Prevalence : 0.5000
##          Detection Rate : 0.2875
##    Detection Prevalence : 0.4750
##       Balanced Accuracy : 0.6000
##
##        'Positive' Class : 0
##
```

Using the same sample of 200 movie reviews, the XGBoost model obtained an accuracy rate of 60%. Although this is a significant decrease from the 75% accuracy rate achieved by the Random Forest model, the performance is still better than a model that makes random predictions, which would have an accuracy of 50%.

The XGBoost model has a kappa of 0.2, indicating a slight agreement between the model predictions and the actual values. This is lower than the kappa of 0.4 obtained by the Random Forest model, suggesting that the XGBoost model's predictions are less consistent with the actual values.

Sensitivity and specificity of the XGBoost model are 0.575 and 0.625, respectively. These values indicate that the model has a slightly higher performance on the positive class compared to the negative class. However, these values are lower than those of the Random Forest model, which achieved similar performance on both classes.

In summary, the XGBoost model's performance is moderate but lower than the Random Forest model. With an accuracy of 60% and a kappa of 0.2, the XGBoost model offers room for improvement. Its sensitivity and specificity are not as balanced as those of the Random Forest model, suggesting that it may be less reliable for predicting negative review.

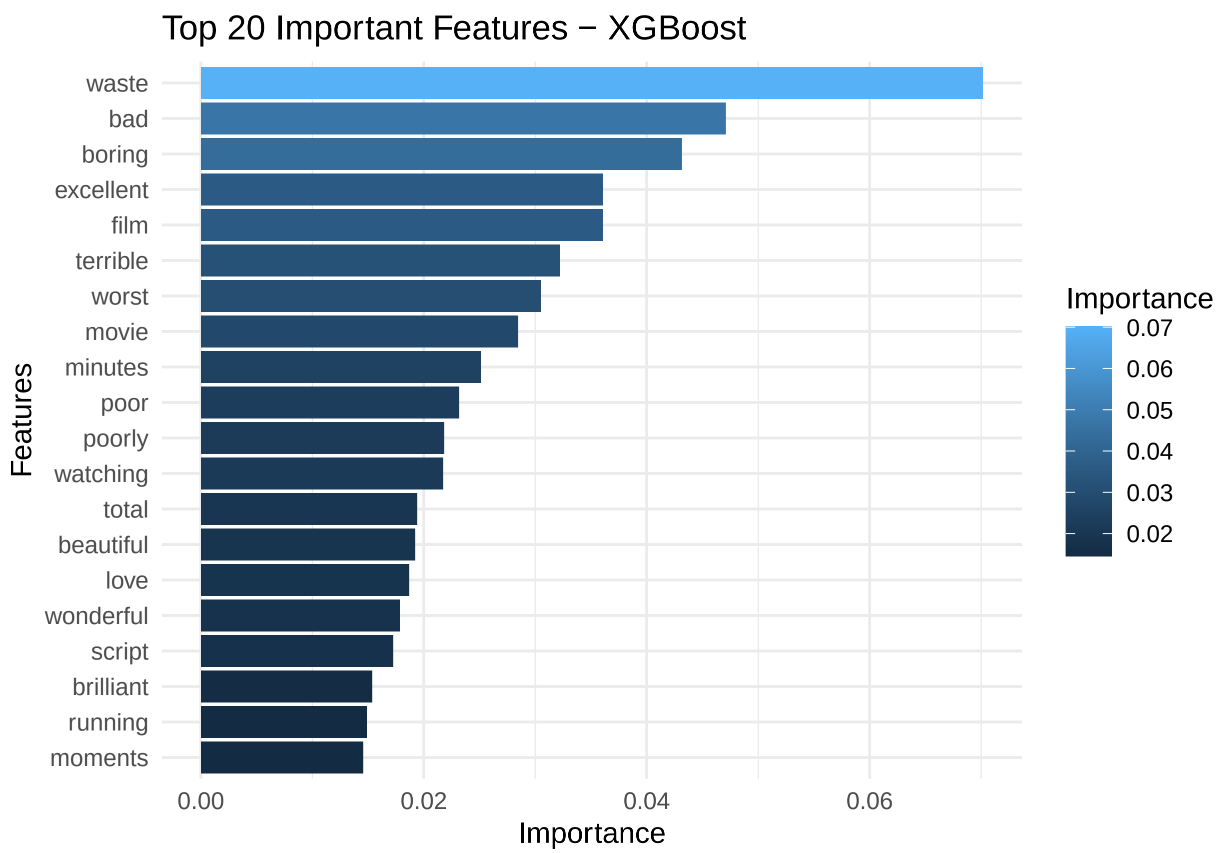
Next we will examine the Variable Importance Plot

```
# Extract variable importance
importance_matrix <- xgb.importance(feature_names = colnames(pred.train.gbm), model = model.xgb)

# Print the top 10 most important features
head(importance_matrix, 10)
```

```
##       Feature       Gain      Cover  Frequency
##  1:     waste 0.07016359 0.03783180 0.01937046
##  2:       bad 0.04707177 0.01347253 0.00968523
##  3:    boring 0.04312693 0.03285804 0.02179177
##  4: excellent 0.03602810 0.03494103 0.02179177
##  5:      film 0.03600733 0.02611963 0.06053269
##  6:  terrible 0.03218384 0.02537529 0.01452785
##  7:     worst 0.03047955 0.02984874 0.01937046
##  8:     movie 0.02845829 0.01103797 0.06053269
##  9:   minutes 0.02508613 0.01961962 0.01452785
## 10:      poor 0.02316336 0.01296174 0.00968523
```

```
# Create a data frame for the plot
var_imp_df <- data.frame(Feature = importance_matrix$Feature, Importance = importance_matrix$Gain)

# Filter the top 20 important features
top_20 <- head(var_imp_df, 20)

# Create a variable importance plot
ggplot(top_20, aes(x = reorder(Feature, Importance), y = Importance, fill = Importance)) +
```

```
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Top 20 Important Features - XGBoost", x = "Features", y = "Importance")
```



There is a lot of overlap with the words found in the Random Forest equivalent of this plot. That being said the Random Forest version probably gives a more accurate assessment of which words are better predictors of sentiment.

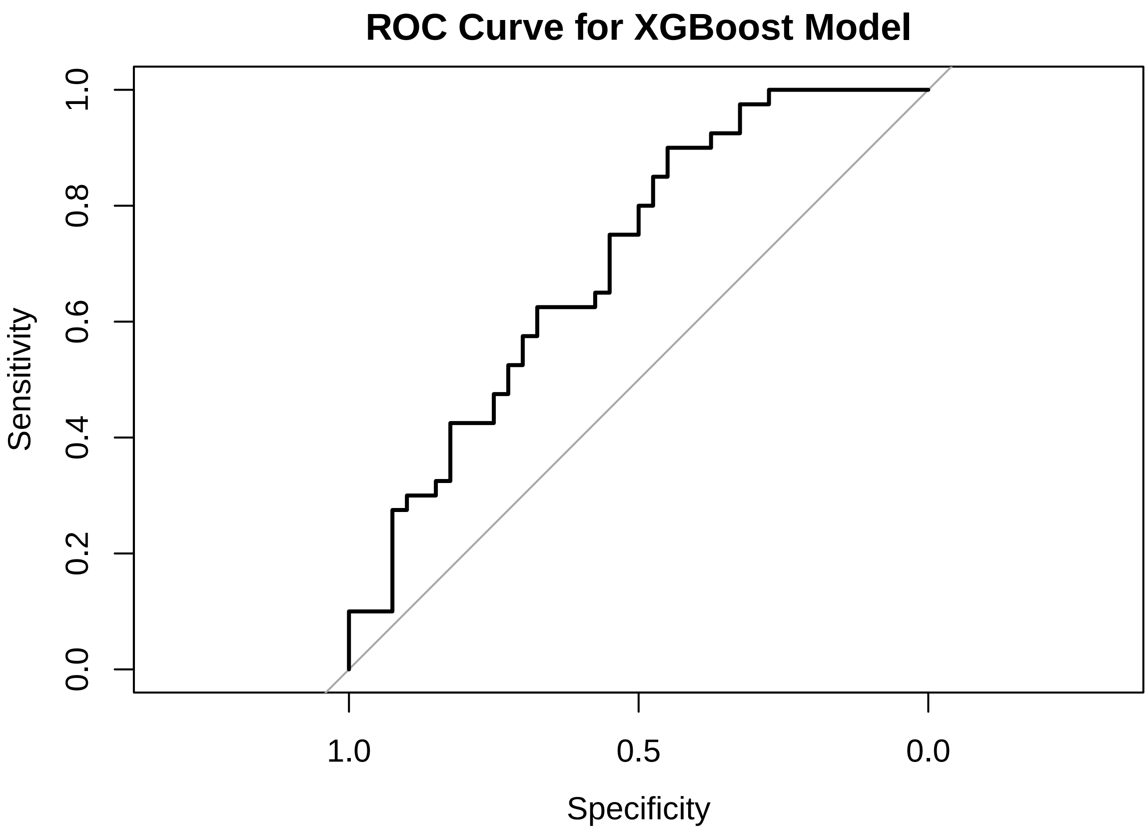
The final step we will take is plotting and interpreting a ROC Curve for the xgboost model.

```
# Calculate ROC
roc_obj_xgb <- roc(imdb.test.gbm, pred.prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
# Plot ROC curve
plot(roc_obj_xgb, main="ROC Curve for XGBoost Model")
```



The ROC curve for the XGBoost model is closer to the diagonal compared to the Random Forest model, this indicates that the XGBoost model's performance is less effective. The diagonal line represents a classifier that predicts outcomes no better than random chance. Thus, the closer the curve is to this diagonal, the less effective the model is at distinguishing between the positive and negative classes.

While the XGBoost model still performs better than random guessing (as the curve is above the diagonal), its performance is not as good as the Random Forest model in this case. This is consistent with the lower accuracy we observed from the confusion matrix for the XGBoost model.

Conclusion

In this project, we implemented a comprehensive approach to sentiment analysis on a set of IMDB movie reviews. The process started with data preprocessing, including tokenization, stop word removal, and cleaning of the text data. We then conducted an exploratory data analysis using techniques such as term frequency and TF-IDF to identify the most common and significant words in the reviews.

We further utilized Latent Dirichlet Allocation (LDA) to try to discover hidden topics within the reviews, providing a deeper understanding of the underlying themes that might be associated with the sentiment of the reviews.

Following this, we shifted to the modeling phase, where we employed Random Forest and XGBoost algorithms for sentiment prediction. The Random Forest model demonstrated superior performance with an accuracy of 75%, compared to the XGBoost model's accuracy of 60%.

The top contributing words for each model were also examined, offering insights into the primary drivers for sentiment prediction. ROC curves were used to visualize the performance of the models, confirming the findings from the accuracy measurements.

In summary, this project illustrates the power of NLP and machine learning techniques in sentiment analysis tasks. Although the models performed reasonably well, there is still potential for further improvement. Future work could explore larger sample sizes, the use of other models, parameter tuning, and feature engineering (length of review, the number of punctuation marks used, or the number of capital letters used) to enhance the predictive accuracy.