IMDB NLP

Jiaqi Sun, Huifei Xu, Hao He, Yaquan Yang

2022-10-16

Package usage

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(reshape2)
library(topicmodels)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(tidytext)
library(Rmpfr)
```

```
## Loading required package: gmp
```

```
##
## Attaching package: 'gmp'
```

```
## The following objects are masked from 'package:base':
##
##     %*%, apply, crossprod, matrix, tcrossprod
## C code of R package 'Rmpfr': GMP using 64 bits per limb
```

```
##
## Attaching package: 'Rmpfr'
```

```
## The following object is masked from 'package:gmp':
##
##     outer
```

```
## The following objects are masked from 'package:stats':
##
##     dbinom, dgamma, dnbinom, dnorm, dpois, dt, pnorm
```

```
## The following objects are masked from 'package:base':
##
##     cbind, pmax, pmin, rbind
```

```r
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:reshape2':
##
##     smiths
```

```r
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```r
library(ggraph)
library(grid)
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following object is masked from 'package:tidyr':
##
##     crossing
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
library(ggraph)
library(stringr)
library(qdap)
```

```
## Loading required package: qdapDictionaries
```

```
## Loading required package: qdapRegex
```

```
##
## Attaching package: 'qdapRegex'
```

```
## The following object is masked from 'package:ggplot2':
##
##     %+%
```

```
## The following object is masked from 'package:dplyr':
##
##     explain
```

```
## Loading required package: qdapTools
```

```
##
## Attaching package: 'qdapTools'
```

```
## The following object is masked from 'package:dplyr':
##
##     id
```

```
##
## Attaching package: 'qdap'
```

```
## The following object is masked from 'package:igraph':
##
##     diversity
```

```
## The following objects are masked from 'package:tm':
##
##     as.DocumentTermMatrix, as.TermDocumentMatrix
```

```
## The following object is masked from 'package:NLP':
##
##     ngrams
```

```
## The following objects are masked from 'package:base':
##
##     Filter, proportions
```

```
library(ggthemes)
library(SnowballC)
library(textstem)
```

```
## Loading required package: koRpus.lang.en
```

```
## Loading required package: koRpus
```

```
## Loading required package: sylly
```

```
## 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:qdap':
##
##     SMOG
```

```
## The following object is masked from 'package:tm':
##
##     readTagged
```

```
##
## Attaching package: 'textstem'
```

```
## The following object is masked from 'package:qdap':
##
##     stem_words
```

In this topic modeling assignment, we use the IMDB Movie Reviews (50K) dataset from Kaggle to analyze the review sentiment and pull out independent topic categories(i.e., movie genre) based on group of keywords.

1. The tidy text format

First, we import data and break down each review into a list of words. Here, we tokenize at word level and treat each review as a separate "document". We manually created a review number for each document since the movie titles for each review are not included in this dataset. Capitalization in reviews is kept for the initial screening.

```
IMDB <- read.csv("IMDB Dataset.csv")

IMDB_df <- tibble(IMDB)
# glimpse(IMDB_df)
# two columns: `sentiment` indicates whether the review is labeled negative or
# positive; `review` includes the text of each review.
head(IMDB_df)
```

```
## # A tibble: 6 × 2
##   review                                                                     senti…¹
##   <chr>                                                                      <chr>
## 1 "One of the other reviewers has mentioned that after watching just 1 … positi…
## 2 "A wonderful little production. <br /><br />The filming technique is … positi…
## 3 "I thought this was a wonderful way to spend time on a too hot summer… positi…
## 4 "Basically there's a family where a little boy (Jake) thinks there's … negati…
## 5 "Petter Mattei's \"Love in the Time of Money\" is a visually stunning… positi…
## 6 "Probably my all-time favorite movie, a story of selflessness, sacrif… positi…
## # … with abbreviated variable name ¹sentiment
```

```
# create review id
IMDB_df %>%
  mutate(review_number = row_number()) ->IMDB_df
```

```
# keep the CAPS for sentiment analysis
tidy_IMDB <- IMDB_df %>%
  unnest_tokens(word, review, to_lower = FALSE)%>% anti_join(stop_words)
```

```
## Joining, by = "word"
```

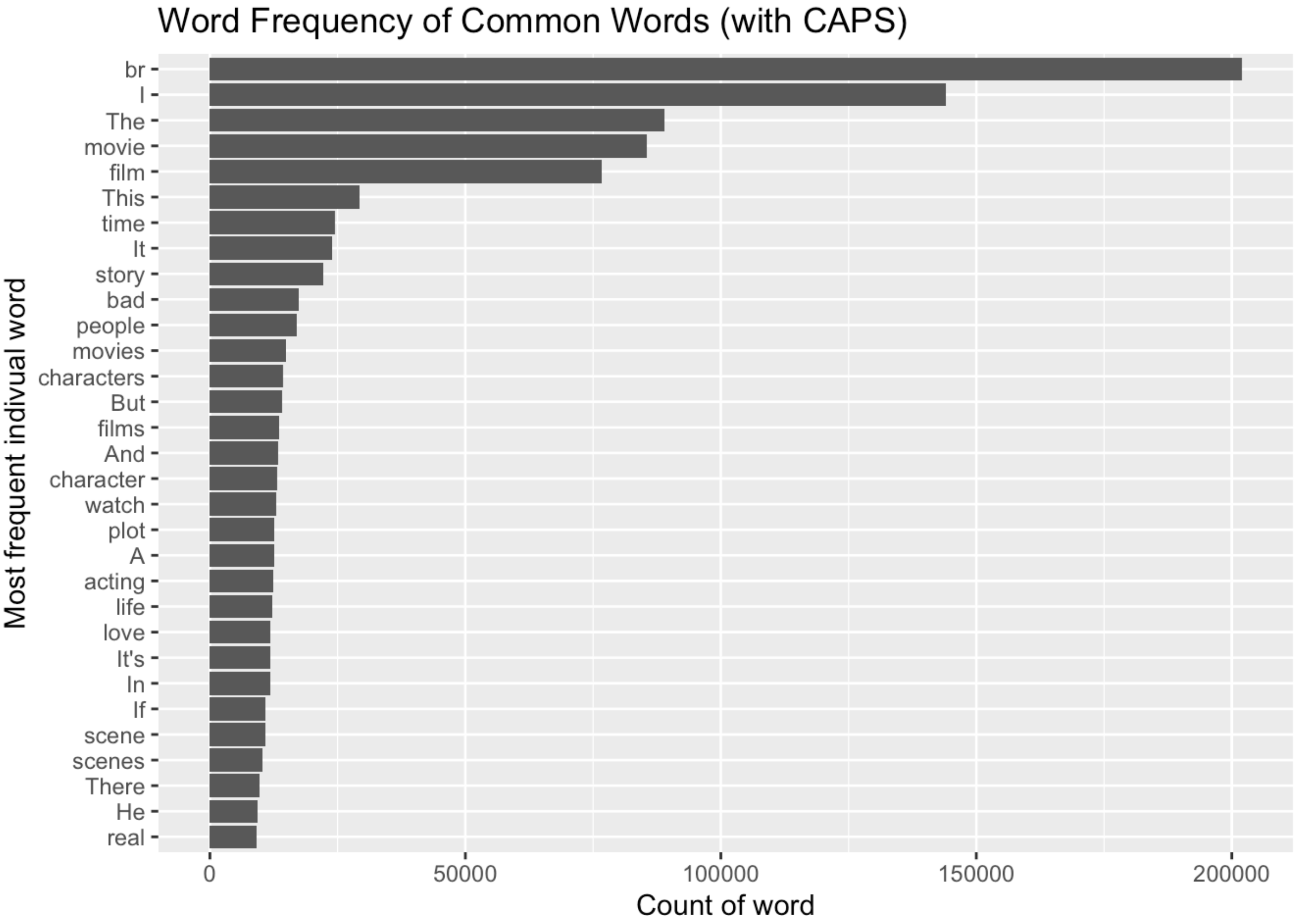
```
tidy_IMDB
```

```
## # A tibble: 5,290,273 × 3
##    sentiment review_number word
##    <chr>             <int> <chr>
##  1 positive              1 One
##  2 positive              1 reviewers
##  3 positive              1 mentioned
##  4 positive              1 watching
##  5 positive              1 1
##  6 positive              1 Oz
##  7 positive              1 episode
##  8 positive              1 hooked
##  9 positive              1 They
## 10 positive              1 happened
## # … with 5,290,263 more rows
```

```
# most common words
tidy_IMDB %>%
count(word, sort = TRUE)
```

```
## # A tibble: 148,150 × 2
##    word       n
##    <chr>  <int>
##  1 br    201948
##  2 I     143943
##  3 The    89011
##  4 movie  85598
##  5 film   76602
##  6 This   29260
##  7 time   24470
##  8 It     23880
##  9 story  22252
## 10 bad    17423
## # … with 148,140 more rows
```

```
#plot frequencies
tidy_IMDB %>%
  count(word, sort = TRUE) %>%
  filter(n > 9000) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col() +
  labs(y = NULL,
       title = "Word Frequency of Common Words (with CAPS)") +
  xlab("Count of word")+
  ylab("Most frequent indivual word")
```



From the frequency plot above, we can see that if we keep CAPS in each review, the most common words would include some words (such as pronouns) that are useless for sentiment analysis and topic identification, so we decide to lowercase all words.

```
tidy_IMDB <- IMDB_df %>%
  unnest_tokens(word, review)%>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
tidy_IMDB
```

```
## # A tibble: 4,601,528 × 3
##    sentiment review_number word
##    <chr>             <int> <chr>
##  1 positive              1 reviewers
##  2 positive              1 mentioned
##  3 positive              1 watching
##  4 positive              1 1
##  5 positive              1 oz
##  6 positive              1 episode
##  7 positive              1 hooked
##  8 positive              1 happened
##  9 positive              1 br
## 10 positive              1 br
## # … with 4,601,518 more rows
```

`Stop_words` is a data frame from tidytext package that contains English stop words from three lexicons. Except for these stop-words, we also customize a list of domain specific stop-words.

See below:

Add “br” to be a customized stop-word as it’s a leftover from html format. Add “film”, “movie”, “em” to customized stop-word as they are appearing in any review and meaningless

```
data(stop_words)
stop_words <- bind_rows(tibble(word = c("br", "film", "movie", "films",
"movies", "characters", "character", "story", "time","people", "watching",
"scene", "scenes", "plot", "watch", "real", "cast", "director", "lot", "pretty",
"10", "actors", "1", "oz", "makes", "2","em"), lexicon = c("custom")),
stop_words)

tidy_IMDB <- tidy_IMDB %>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
tidy_IMDB
```

```
## # A tibble: 3,978,800 × 3
##    sentiment review_number word
##    <chr>             <int> <chr>
##  1 positive              1 reviewers
##  2 positive              1 mentioned
##  3 positive              1 episode
##  4 positive              1 hooked
##  5 positive              1 happened
##  6 positive              1 struck
##  7 positive              1 brutality
##  8 positive              1 unflinching
##  9 positive              1 violence
## 10 positive              1 set
## # … with 3,978,790 more rows
```

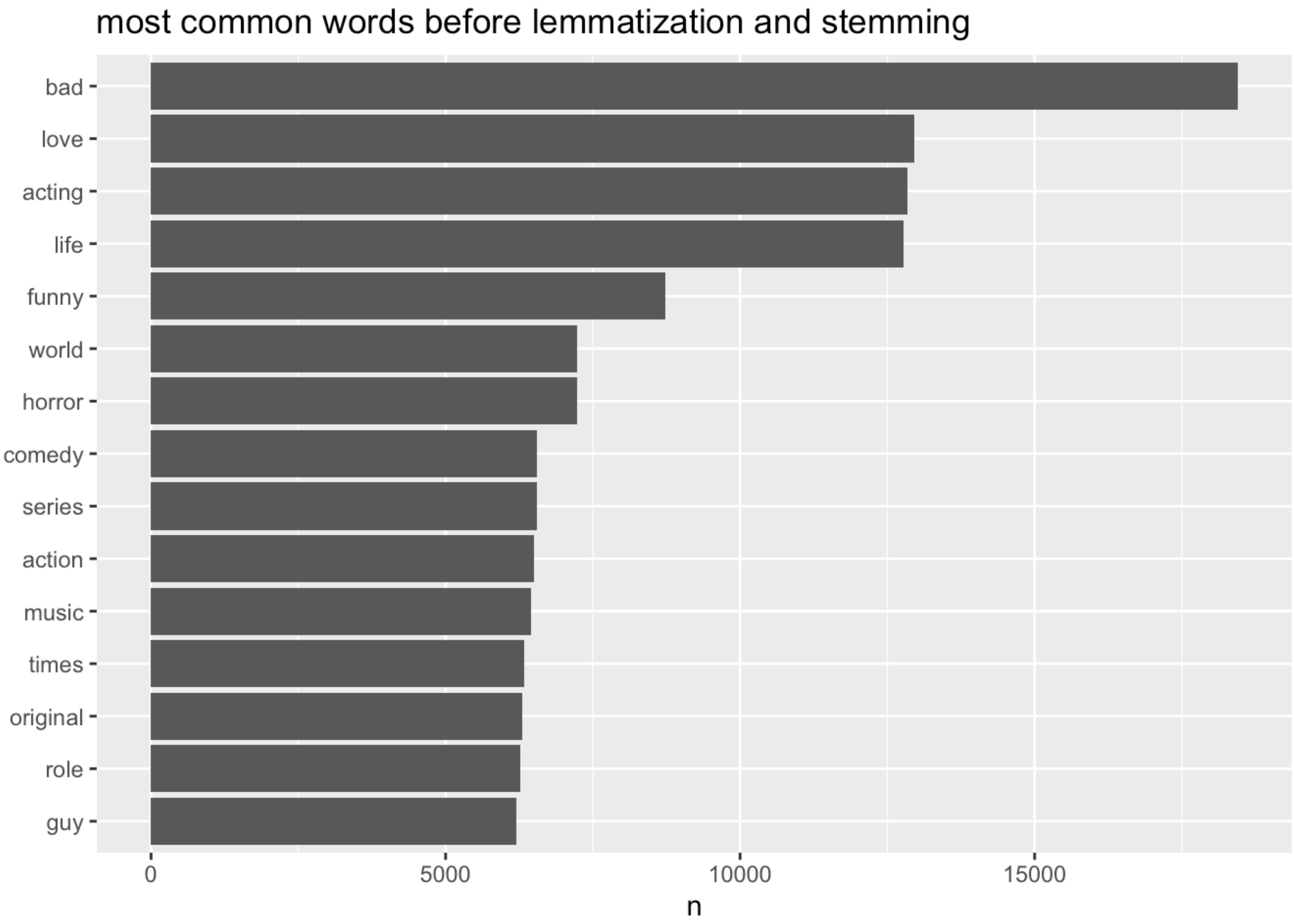
After remove all the stop-words, we try stemming and lemmatization. Stemming using rules to cut words down to their stems, operating on the word by itself. Lemmatization uses knowledge about a language's structure to reduce words down to their lemmas, the canonical or dictionary forms of words, operating on the word in its context. As an important part of NLP pipelines, these methods would help us reduce the feature space of text data and should decrease the sparsity of text data. In this way, we may have a better fitted LDA model in the later process.

```
# stemming and lemmatization
library(textstem)
lemmatization <- tidy_IMDB %>% mutate(lemma = word%>% lemmatize_words())

library(SnowballC)
stemming <- tidy_IMDB %>% mutate(stem = wordStem(word))
```

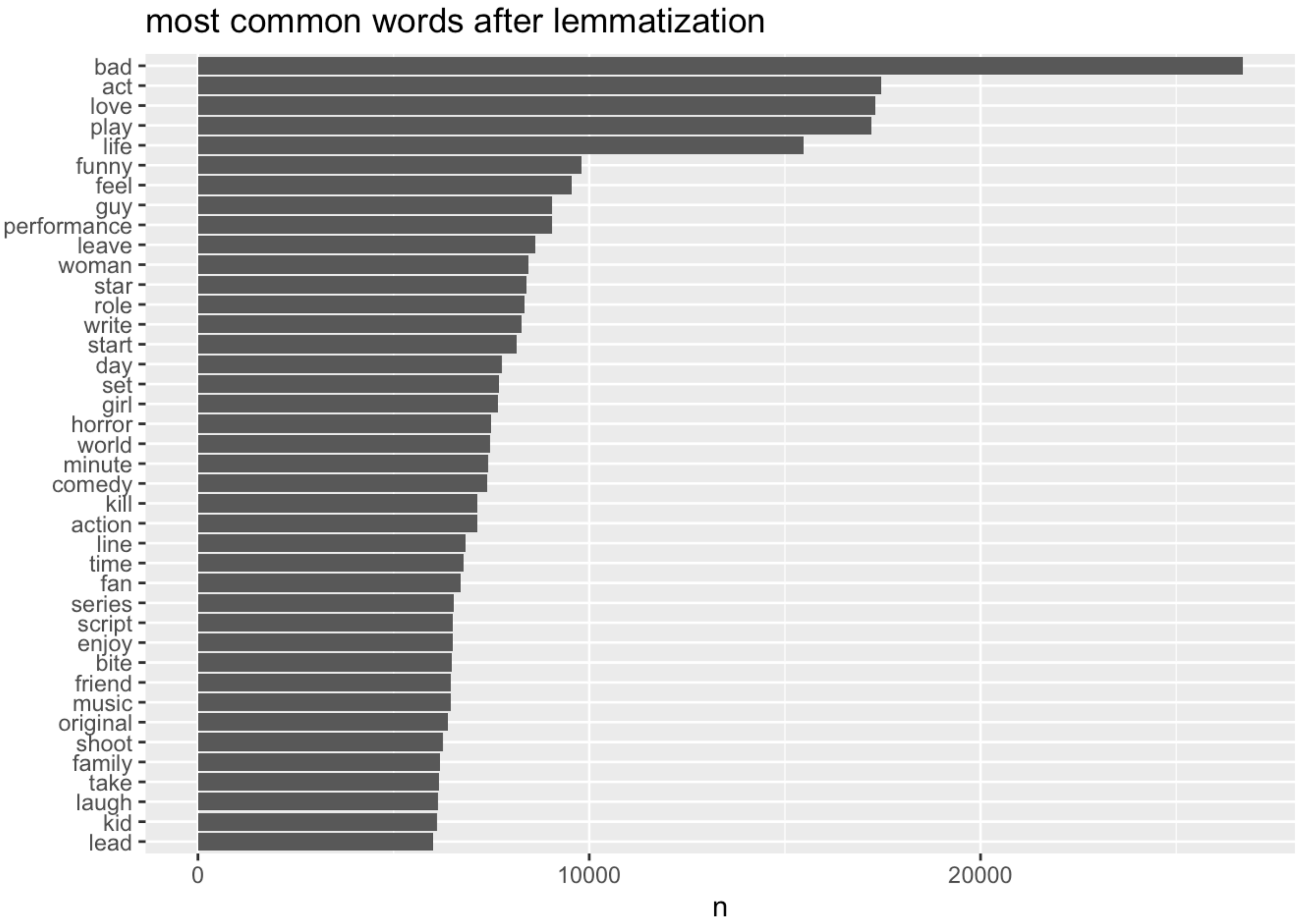
```
tidy_IMDB %>%
  count(sentiment, word, sort = TRUE)
```

```
## # A tibble: 168,241 × 3
##    sentiment word        n
##    <chr>     <chr>   <int>
##  1 negative  bad     14706
##  2 positive  love     8669
##  3 negative  acting   8070
##  4 positive  life     8040
##  5 negative  worst    4884
##  6 positive  acting   4772
##  7 negative  life     4735
##  8 negative  funny    4636
##  9 positive  world    4610
## 10 negative  love     4293
## # … with 168,231 more rows
```

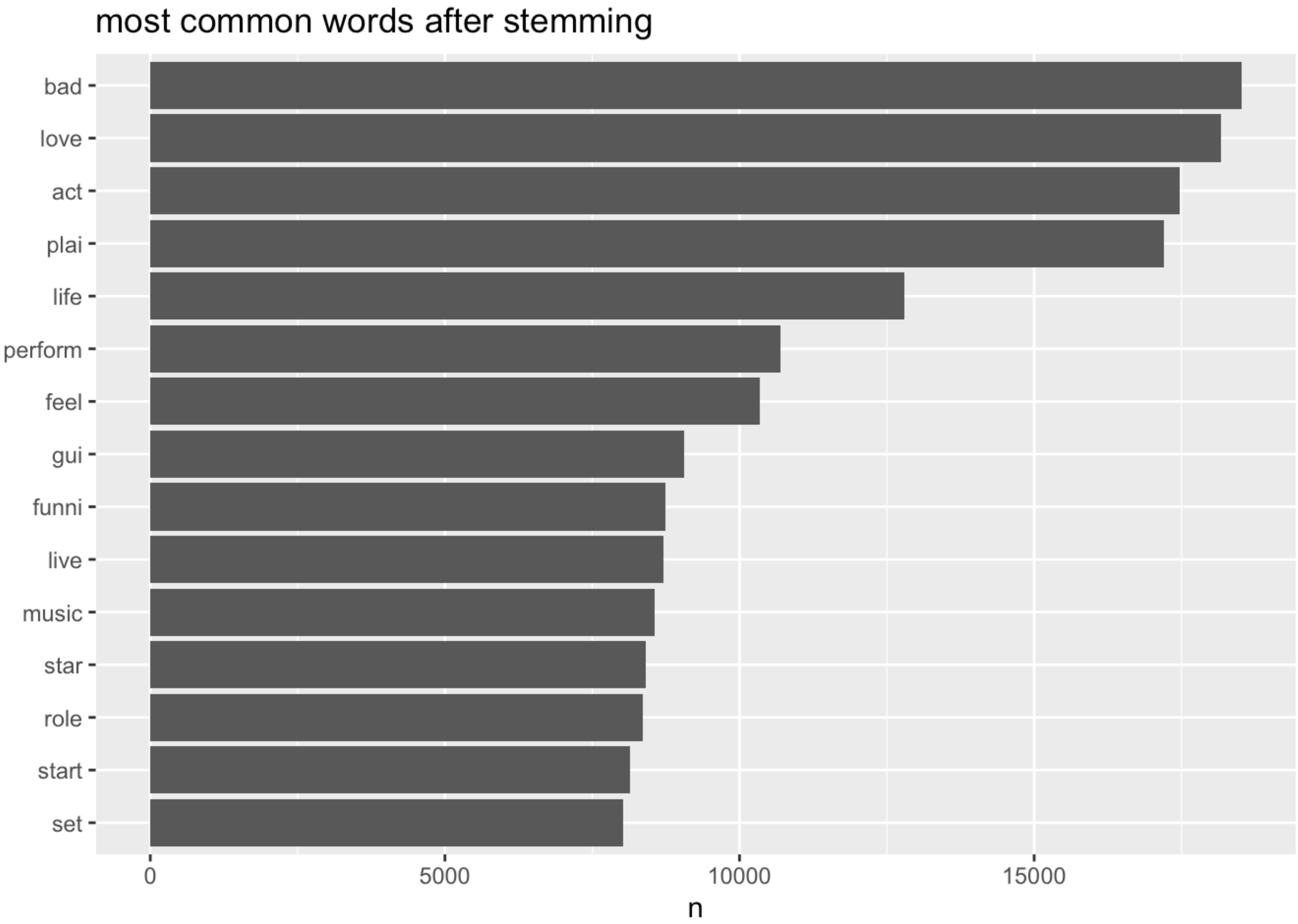
```
#The most common words in IMDB comments
tidy_IMDB %>%
  count(word, sort = TRUE) %>%
  filter(n > 6000) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col() +
  labs(y = NULL,title = "most common words before lemmatization and stemming")
```



```
# The most common words after lemmatization
lemmatization %>%
  count(lemma, sort = TRUE) %>%
  filter(n > 6000) %>%
  mutate(lemma = reorder(lemma, n)) %>%
  ggplot(aes(n, lemma)) +
  geom_col() +
  labs(y = NULL, title = "most common words after lemmatization")
```



```
# The most common words after stemming
stemming %>%
  count(stem, sort = TRUE) %>%
  filter(n > 8000) %>%
  mutate(stem = reorder(stem, n)) %>%
  ggplot(aes(n, stem)) +
  geom_col() +
  labs(y = NULL, title = "most common words after stemming")
```



It’s easy to see that stemming would convert “plays” to “plai”, while in lemmatization it would be “play”.

```
tidy_IMDB %>%
  count(sentiment)
```

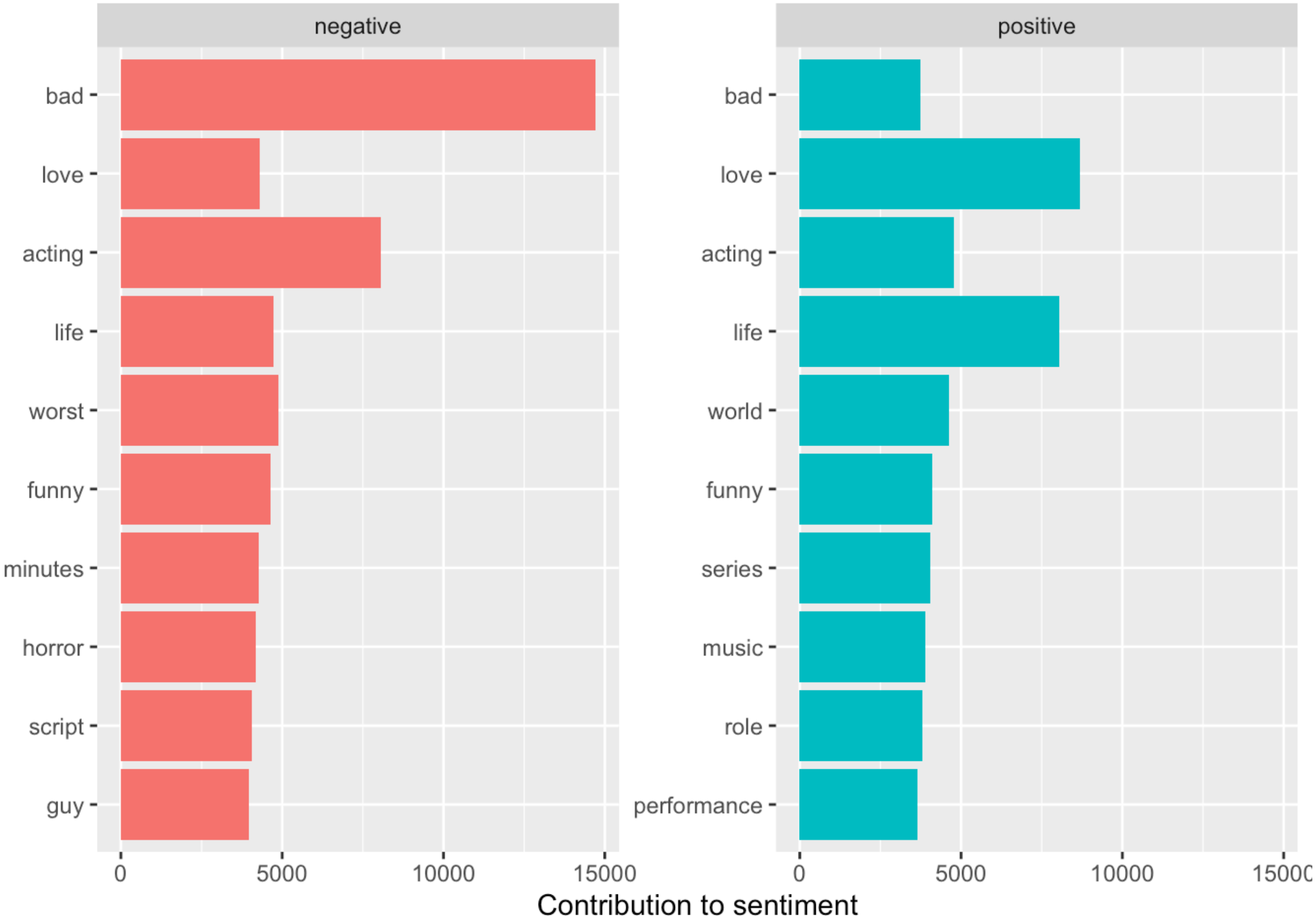
```
## # A tibble: 2 × 2
##   sentiment       n
##   <chr>       <int>
## 1 negative  1928848
## 2 positive  2049952
```

2. Sentiment analysis with tidy data

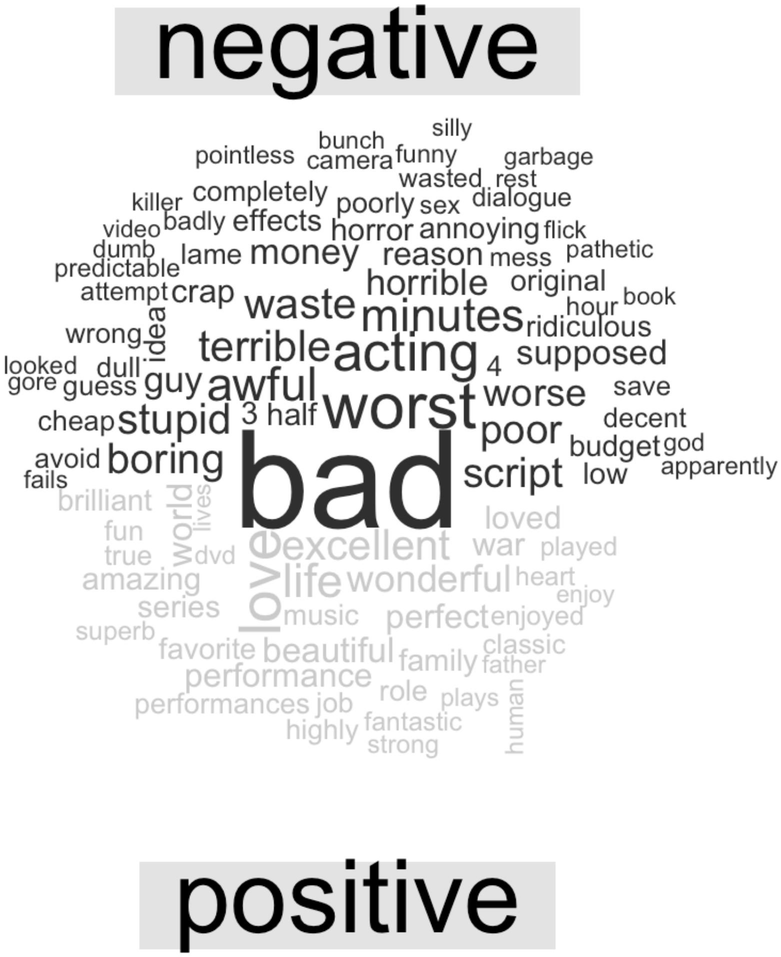
```
bing_word_counts <- tidy_IMDB %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

bing_word_counts
```

```
## # A tibble: 168,241 × 3
##    word   sentiment     n
##    <chr>  <chr>     <int>
##  1 bad    negative  14706
##  2 love   positive   8669
##  3 acting negative   8070
##  4 life   positive   8040
##  5 worst  negative   4884
##  6 acting positive   4772
##  7 life   negative   4735
##  8 funny  negative   4636
##  9 world  positive   4610
## 10 love   negative   4293
## # … with 168,231 more rows
```

```
#Words that contribute to positive and negative sentiment in IMDB
bing_word_counts %>%
  group_by(sentiment) %>%
  slice_max(n, n = 10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(x = "Contribution to sentiment",
       y = NULL)
```



```
# Most common positive and negative words in IMDB

tidy_IMDB %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("gray20", "gray80"),
                   max.words = 100)
```



```
table(IMDB$sentiment)
```

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

The polarity function scans the subjectivity lexicon for positive and negative words, and we can further visualize the following results to compare with our sentiment tags. The subjective dictionary used by the qdap package contains about 6800 words with tags, which would be introduced here to help measuring the density of keywords

```
Sys.setlocale("LC_ALL", "C")
```

```
## [1] "C/C/C/C/C/en_US.UTF-8"
```

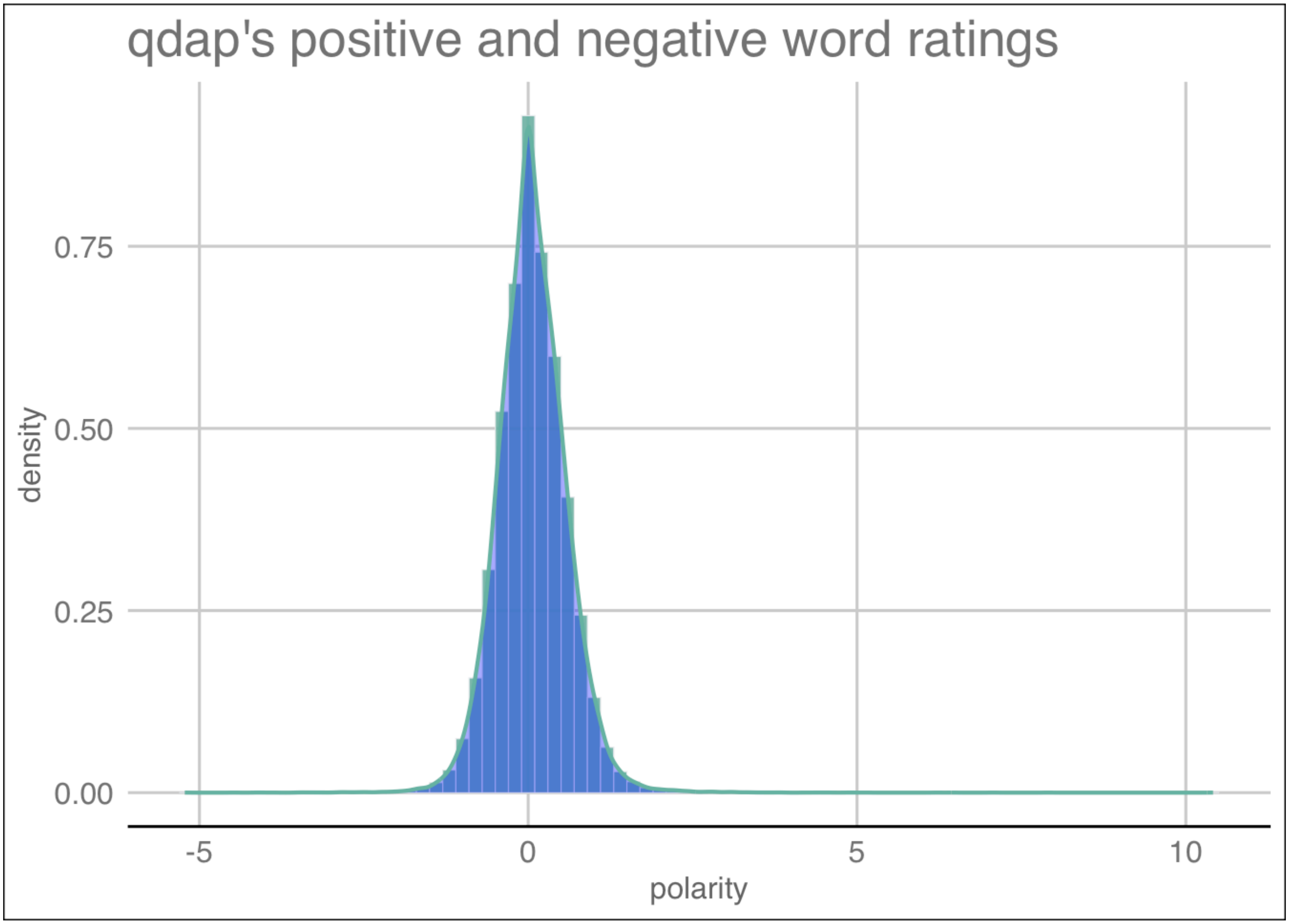
```
imdb.pol<-polarity(as.character(IMDB$review))
```

```
## Warning in polarity(as.character(IMDB$review)):
##   Some rows contain double punctuation.  Suggested use of `sentSplit` function.
```

```
ggplot(imdb.pol$all,aes(x=polarity,y=..density..))+
  theme_gdocs()+
  geom_histogram(binwidth=0.2,
                 fill= "#69b3a2",
                 colour="#e9ecef",
                 alpha=0.9,
                 size=0.2)+
  geom_density(size=0.75,
               fill="blue",
               adjust=1.5,
               alpha=0.4,
               color="#69b3a2")+
  ggtitle("qdap's positive and negative word ratings")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
```



3. Analyzing word and document frequency: tf-idf

```
IMDB_tf_idf <-  tidy_IMDB %>%
  count(review_number, word, sort = TRUE) %>%
  bind_tf_idf(word, review_number, n) %>%
  arrange((tf_idf))

IMDB_tf_idf
```

```
## # A tibble: 3,423,798 x 6
##    review_number word        n       tf   idf  tf_idf
##            <int> <chr>   <int>    <dbl> <dbl>   <dbl>
##  1         31482 world       1 0.000899  2.22 0.00200
##  2         31437 love        1 0.00117   1.73 0.00202
##  3         31241 love        1 0.00118   1.73 0.00204
##  4         40522 acting      1 0.00140   1.54 0.00217
##  5         31482 tv          1 0.000899  2.47 0.00222
##  6         31482 watched     1 0.000899  2.53 0.00227
##  7         31482 series      1 0.000899  2.57 0.00232
##  8         12648 acting      1 0.00152   1.54 0.00235
##  9         31482 play        1 0.000899  2.65 0.00239
## 10         31437 funny       1 0.00117   2.06 0.00242
## # ... with 3,423,788 more rows
```

```
summary(IMDB_tf_idf)
```

```
##  review_number       word                 n                tf           
##  Min.   :    1   Length:3423798     Min.   : 1.000   Min.   :0.0008993  
##  1st Qu.:12464   Class :character   1st Qu.: 1.000   1st Qu.:0.0062500  
##  Median :24964   Mode  :character   Median : 1.000   Median :0.0111111  
##  Mean   :24980                      Mean   : 1.162   Mean   :0.0146037  
##  3rd Qu.:37469                      3rd Qu.: 1.000   3rd Qu.:0.0190476  
##  Max.   :50000                      Max.   :36.000   Max.   :1.0000000  
##       idf             tf_idf        
##  Min.   : 1.446   Min.   :0.001996  
##  1st Qu.: 3.559   1st Qu.:0.029030  
##  Median : 4.711   Median :0.051464  
##  Mean   : 5.044   Mean   :0.071160  
##  3rd Qu.: 6.185   3rd Qu.:0.089869  
##  Max.   :10.820   Max.   :6.542192  
```

The inverse document frequency (and thus tf-idf) is very low (near zero) for words that occur in many of the reviews in a collection; this is how this approach decreases the weight for common words. The inverse document frequency will be a higher number for words that occur in fewer of the documents in the collection.

4. Relationships between words: n-grams and correlations

```
IMDB_bigrams <- IMDB_df %>%
  unnest_tokens(bigram, review, token = "ngrams", n = 2) %>%
  filter(!is.na(bigram))

IMDB_bigrams
```

```
## # A tibble: 11,687,524 x 3
##    sentiment review_number bigram
##    <chr>             <int> <chr>
##  1 positive              1 one of
##  2 positive              1 of the
##  3 positive              1 the other
##  4 positive              1 other reviewers
##  5 positive              1 reviewers has
##  6 positive              1 has mentioned
##  7 positive              1 mentioned that
##  8 positive              1 that after
##  9 positive              1 after watching
## 10 positive              1 watching just
## # ... with 11,687,514 more rows
```

```
IMDB_bigrams %>%
  count(bigram, sort = TRUE)
```

```
## # A tibble: 2,344,943 x 2
##    bigram          n
##    <chr>       <int>
##  1 br br      101039
##  2 of the      77235
##  3 in the      50216
##  4 this movie  31166
##  5 and the     26639
##  6 is a        26080
##  7 the film    25118
##  8 to the      23697
##  9 to be       23483
## 10 the movie   23186
## # ... with 2,344,933 more rows
```

As one might expect, a lot of the most common bigrams are pairs of common (uninteresting) words, such as of the and to be: what we call “stop-words” (see Chapter 1). This is a useful time to use tidyr’s separate(), which splits a column into multiple based on a delimiter. This lets us separate it into two columns, “word1” and “word2”, at which point we can remove cases where either is a stop-word.

```
bigrams_separated <- IMDB_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts
```

```
## # A tibble: 820,351 x 3
##    word1      word2          n
##    <chr>      <chr>      <int>
##  1 special    effects     2240
##  2 low        budget      1812
##  3 sci        fi          1384
##  4 bad        acting       657
##  5 bad        guys         656
##  6 tv         series       587
##  7 bad        guy          573
##  8 production values       562
##  9 highly     recommend    553
## 10 world      war          529
## # ... with 820,341 more rows
```

bigram and visualisation for negative, positive remarks

```
get.bigram.tidy<-function(df)
{
df %>%
  unnest_tokens(bigram, review, token = "ngrams", n = 2) %>%
  filter(!is.na(bigram)) %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word) %>%
  count(word1, word2, sort = TRUE)
}
```

```
viz.bigram.graph <- function(bigrams,layout='fr') {
  set.seed(2022)
  a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

  bigrams %>%
    graph_from_data_frame() %>%
    ggraph(layout = layout) +
    geom_edge_link( show.legend = FALSE, arrow = a) +
    geom_node_point(color = "lightblue", size = 5) +
    geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
    theme_void()+
   coord_equal(clip='off')+
   theme(plot.margin=unit(rep(2,4),'lines'))
}
```

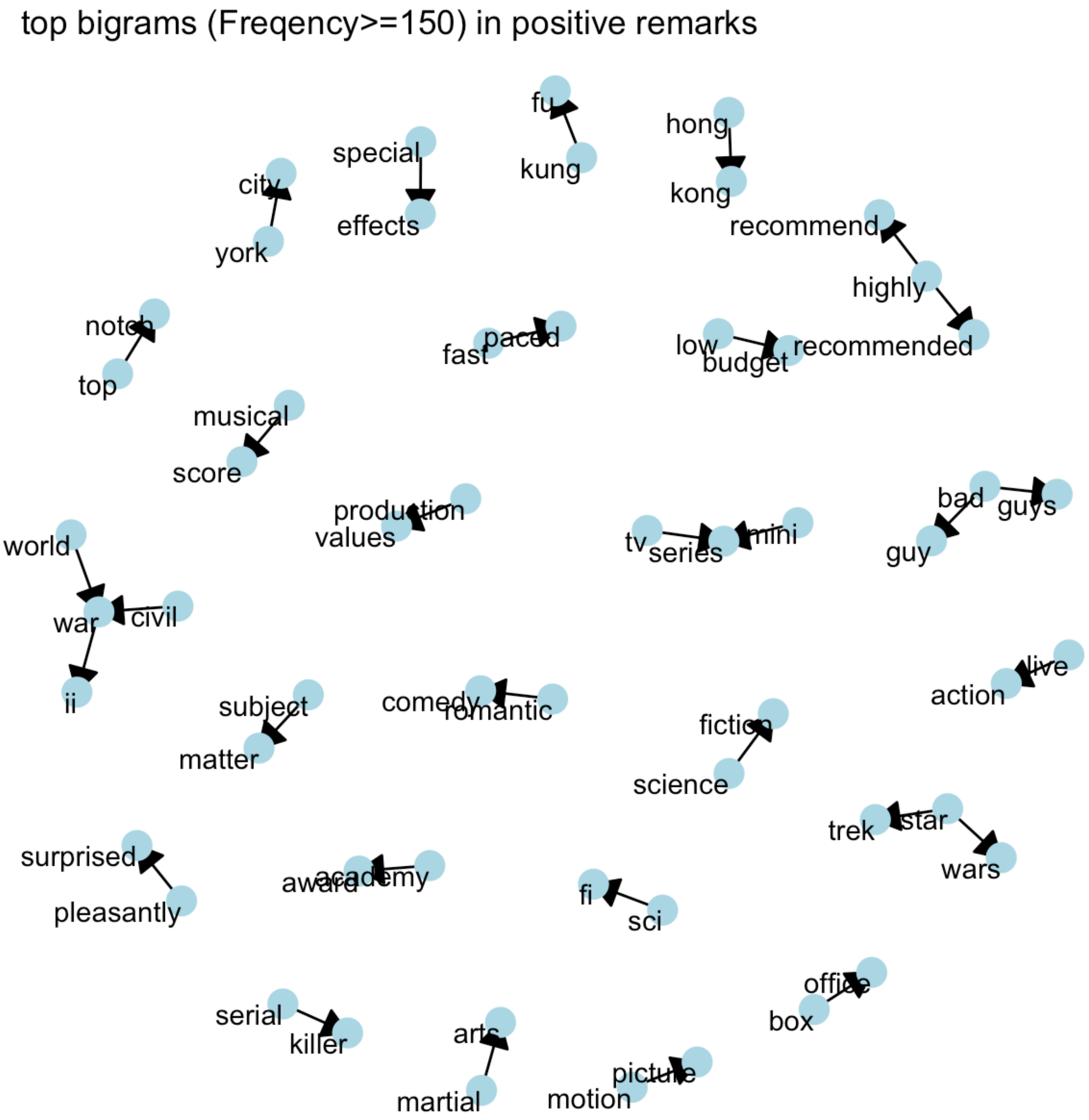
```
pos<-IMDB_df %>% filter(sentiment=='positive')
neg<-IMDB_df %>% filter(sentiment=='negative')

pos.bigram<-get.bigram.tidy(pos)
neg.bigram<-get.bigram.tidy(neg)
```

```
pos.bigram
```

```
## # A tibble: 465,153 x 3
##    word1   word2           n
##    <chr>   <chr>       <int>
##  1 special effects       804
##  2 sci     fi            620
##  3 low     budget        592
##  4 highly  recommend     502
##  5 world   war           380
##  6 highly  recommended   372
##  7 tv      series        337
##  8 top     notch         295
##  9 kung    fu            271
## 10 science fiction       257
## # ... with 465,143 more rows
```

```
ig.pos<-pos.bigram %>%
  dplyr::filter(n > 150,
         !str_detect(word1, "\\d"),
         !str_detect(word2, "\\d")) %>%
   viz.bigram.graph() +
   ggtitle('top bigrams (Freqency>=150) in positive remarks')

ig.pos
```

```
## Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` in the `default_aes` field and elsewhere instead.
```



top bigrams (Freqency>=150) in positive remarks

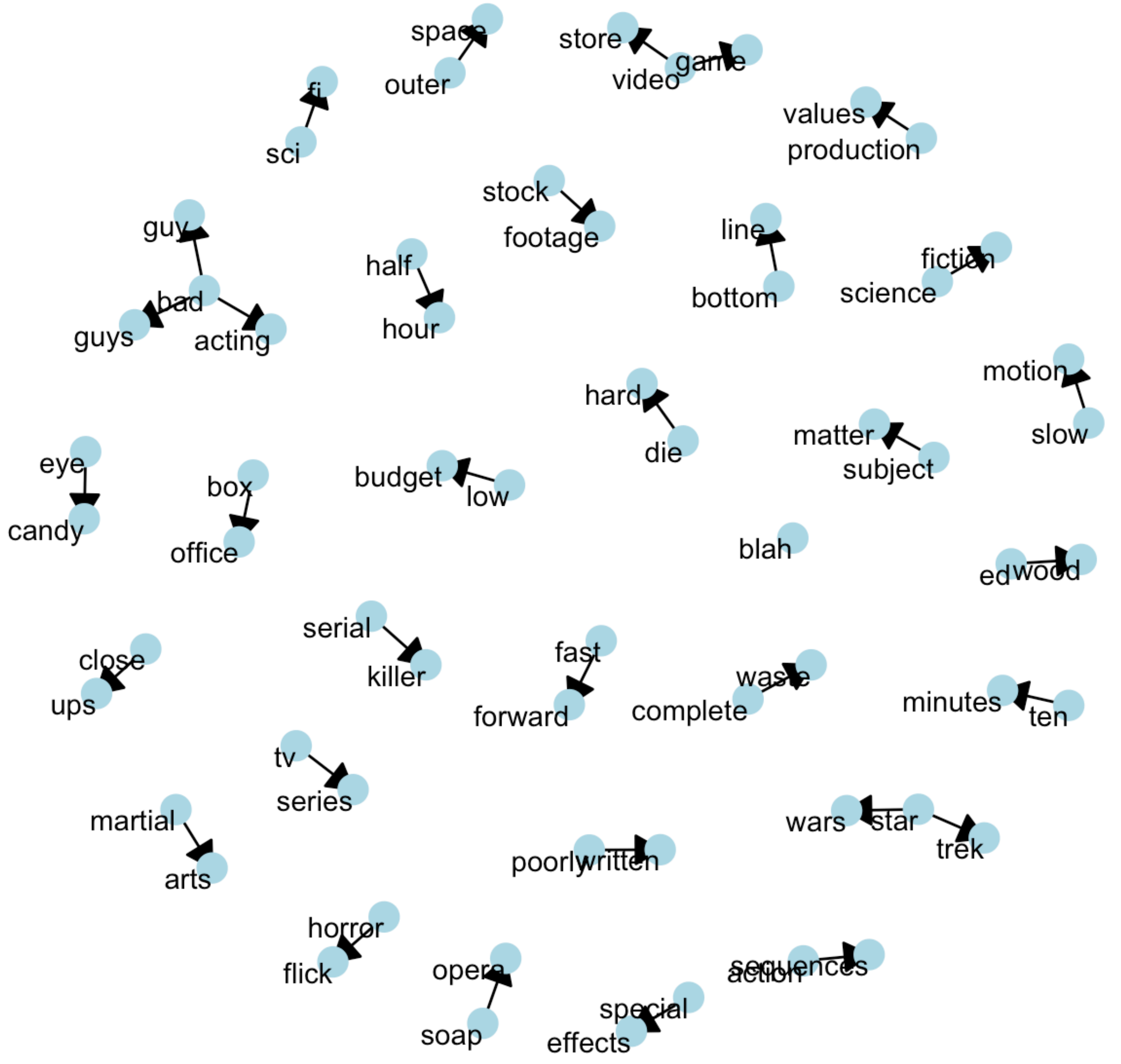
top bigram graph in negative remarks

```
neg.bigram
```

```
## # A tibble: 416,404 x 3
##    word1      word2       n
##    <chr>      <chr>   <int>
##  1 special    effects  1436
##  2 low        budget   1220
##  3 sci        fi        764
##  4 bad        acting    603
##  5 bad        guys      408
##  6 bad        guy       364
##  7 production values    345
##  8 90         minutes   308
##  9 20         minutes   290
## 10 martial    arts      260
## # ... with 416,394 more rows
```

```
ig.neg<-neg.bigram %>%
  filter(n > 150,
         !str_detect(word1, "\\d"),
         !str_detect(word2, "\\d")) %>%
   viz.bigram.graph() +
   ggtitle('top bigrams (Freqency>=150) in negative remarks')

ig.neg
```

top bigrams (Freqency>=150) in negative remarks



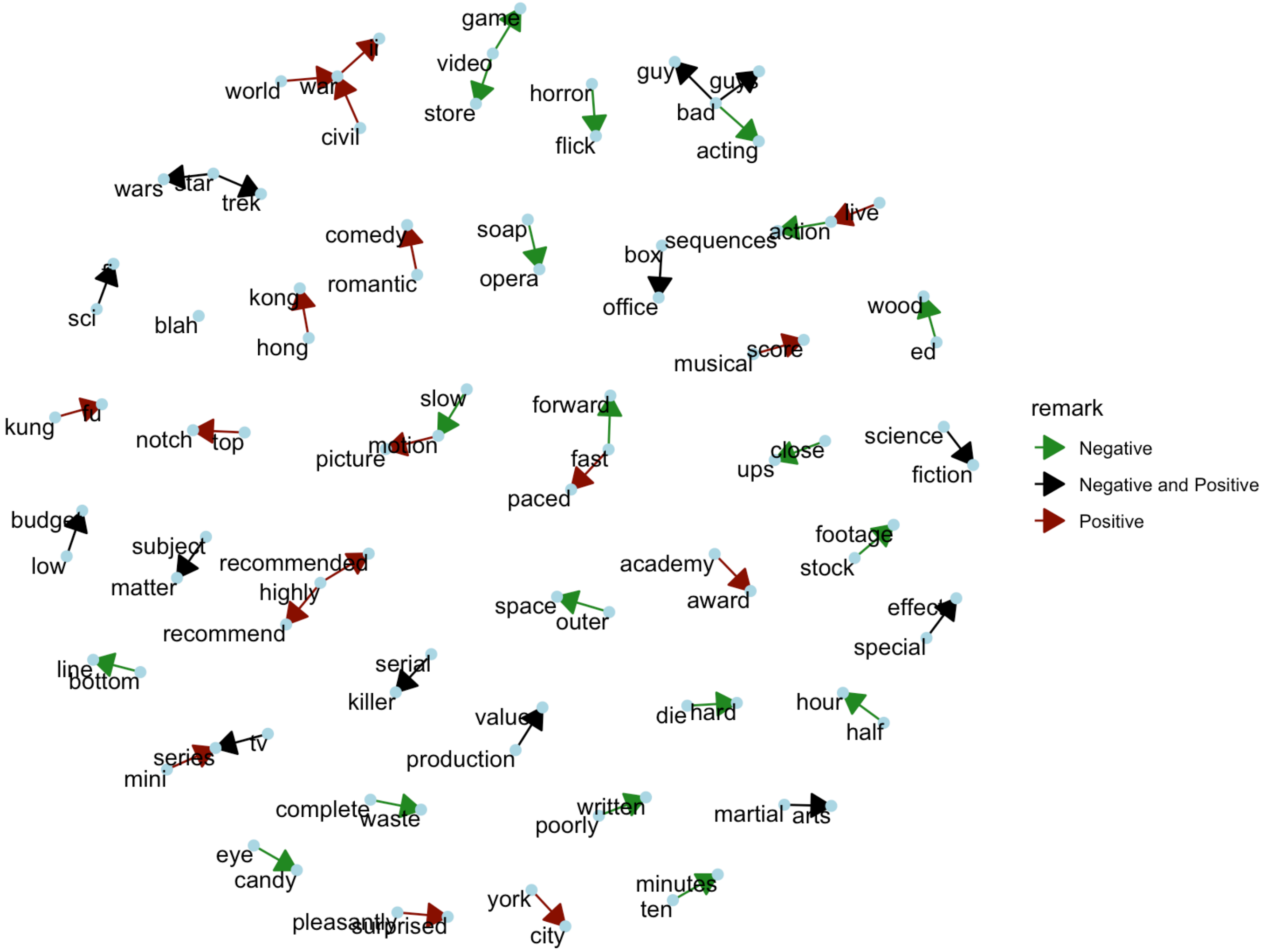
From above two tables and graphs, we found some bigrams present both in negative remarks and positie remarks, for example, bigram of `special` and `effects` occupied the first place in both ranklist!

So we’are about to show top bigrams in positive remarks only and negative remarks only and both in positive and negative remarks as following:

```r
pos.top.bigram<-pos.bigram %>%
  filter(n > 150,
         !str_detect(word1, "\\d"),
         !str_detect(word2, "\\d"))

neg.top.bigram<-neg.bigram %>%
  filter(n > 150,
         !str_detect(word1, "\\d"),
         !str_detect(word2, "\\d"))

common<-inner_join(pos.top.bigram,neg.top.bigram,by=c('word1','word2')) %>%
        mutate(n=n.x+n.y) %>%
        select(word1,word2,n) %>%
        mutate(remark='Negative and Positive')

pos.only<-anti_join(pos.top.bigram,neg.top.bigram,by=c('word1','word2')) %>%
          mutate(remark='Positive')


neg.only<-anti_join(neg.top.bigram,pos.top.bigram,by=c('word1','word2')) %>%
          mutate(remark='Negative')


allbigram<-rbind(common,pos.only,neg.only)

allbigram %>% group_by(remark) %>% slice_head(n=3)
```

```
## # A tibble: 9 x 4
## # Groups:   remark [3]
##   word1   word2           n remark
##   <chr>   <chr>       <int> <chr>
## 1 bad     acting        603 Negative
## 2 fast    forward       247 Negative
## 3 ten     minutes       221 Negative
## 4 special effects      2240 Negative and Positive
## 5 sci     fi           1384 Negative and Positive
## 6 low     budget       1812 Negative and Positive
## 7 highly  recommend     502 Positive
## 8 world   war           380 Positive
## 9 highly  recommended   372 Positive
```

```
set.seed(2022)

a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

allbigram %>%
graph_from_data_frame() %>%
    ggraph(layout = 'fr') +
    geom_edge_link(aes(colour=remark), show.legend = TRUE, arrow = a) +
    geom_node_point(color = "lightblue", size = 2) +
    geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
    theme_void()+
    coord_equal(clip='off')+
    theme(plot.margin=unit(rep(2,4),'lines'))+
    scale_edge_colour_manual(values=c('forestgreen','black','darkred'))
```



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

```
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")

bigrams_united
```

```
## # A tibble: 1,319,960 x 3
##    sentiment review_number bigram
##    <chr>             <int> <chr>
##  1 positive              1 faint hearted
##  2 positive              1 drugs sex
##  3 positive              1 oswald maximum
##  4 positive              1 maximum security
##  5 positive              1 emerald city
##  6 positive              1 experimental section
##  7 positive              1 glass fronts
##  8 positive              1 aryans muslims
##  9 positive              1 muslims gangstas
## 10 positive              1 gangstas latinos
## # ... with 1,319,950 more rows
```

In other analyses, we may be interested in the most common trigrams, which are consecutive sequences of 3 words. We can find this by setting n = 3:

```
IMDB_df %>%
  unnest_tokens(trigram, review, token = "ngrams", n = 3) %>%
  filter(!is.na(trigram)) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word,
         !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)
```

```
## # A tibble: 411,042 x 4
##    word1   word2    word3        n
##    <chr>   <chr>    <chr>    <int>
##  1 world   war      ii         230
##  2 sci     fi       channel    164
##  3 low     budget   horror     134
##  4 bad     acting   bad        103
##  5 mystery science  theater     97
##  6 blah    blah     blah        90
##  7 texas   chainsaw massacre    90
##  8 blair   witch    project     84
##  9 local   video    store       80
## 10 tour    de       force       77
## # ... with 411,032 more rows
```

5. Converting to and from non-tidy formats

Just as some existing text mining packages provide document-term matrices as sample data or output, some algorithms expect such matrices as input. Therefore, tidytext provides cast_verbs for converting from a tidy form to these matrices.

```
IMDB_dtm <- tidy_IMDB %>%
  count(review_number, word) %>%
  cast_dtm(review_number, word, n)

IMDB_dtm
```

```
## <<DocumentTermMatrix (documents: 50000, terms: 118916)>>
## Non-/sparse entries: 3423798/5942376202
## Sparsity           : 100%
## Maximal term length: 72
## Weighting          : term frequency (tf)
```

6. Topic modeling

Latent Dirichlet allocation is one of the most common algorithms for topic modeling. Without diving into the math behind the model, we can understand it as being guided by two principles.

This function returns an object containing the full details of the model fit, such as how words are associated with topics and how topics are associated with documents.

```
# choose number of topics, 6
k = 6

# set a seed so that the output of the model is predictable
IMDB_lda <- LDA(IMDB_dtm, k, control = list(seed = 1234))
IMDB_lda
```

```
## A LDA_VEM topic model with 6 topics.
```

```
#> A LDA_VEM topic model with 6 topics.
```

To start to exploring the lda model, we list the most frequent 30 terms in the topic listed, in rank order.

```
imdb.topics <- topicmodels::topics(IMDB_lda, 1)
## In this case the output returns the top 30 terms.
imdb.terms <- as.data.frame(topicmodels::terms(IMDB_lda, 30),
                            stringsAsFactors = FALSE)
imdb.terms[1:k]
```

```
##         Topic 1      Topic 2   Topic 3     Topic 4  Topic 5     Topic 6
## 1        acting         love       bad         bad     love      acting
## 2          life         feel      life        life      bad      series
## 3         funny        funny    acting      comedy     life        feel
## 4         music      minutes    horror      acting      set    original
## 5  entertaining        found      star       funny     role        role
## 6         times          bad     takes     minutes    world         bit
## 7        series        world  terrible       worst    music         guy
## 8        played       horror    action         guy original      played
## 9        comedy       acting      play      script   script          tv
## 10  performance     original       guy         bit    start       music
## 11         hard        death    father         day    woman         fun
## 12         book         evil  american   hollywood     idea       times
## 13          bit  performance     piece    terrible      dvd       worth
## 14         shot performances     lines       times     true         war
## 15          fan        worst     world        idea     half         bad
## 16       screen        plays       day performance   action        book
## 17        plays         main      dead        read     mind        home
## 18         left         nice  audience         boy   family        read
## 19          job       family      head        shot    worst         low
## 20   completely         girl excellent      reason    times   excellent
## 21        house         hard      rest        nice    found        true
## 22       script        sense     black    audience   horror     version
## 23        actor          guy        tv       sense    funny        girl
## 24         girl      classic       job        book american        main
## 25       family          job    school        fine  version      mother
## 26         john        enjoy     funny         sex     live performance
## 27    beautiful       series     money     effects    wrong      screen
## 28      watched    recommend      past      stupid    stuff    remember
## 29         mind     american  dialogue        hard    death      school
## 30          fun         life     plays        half    actor        idea
```

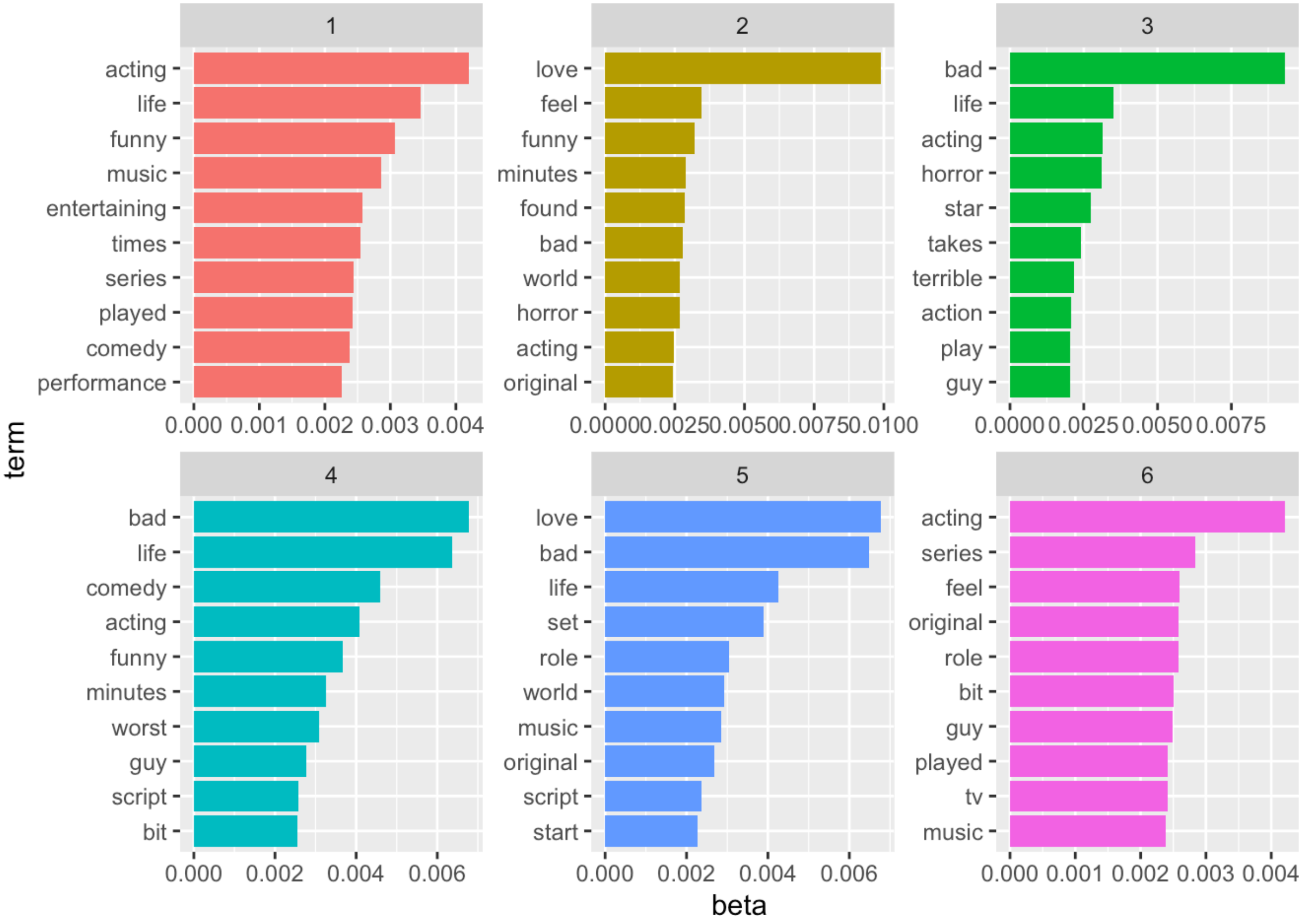
The tidytext package provides this method for extracting the per-topic-per-word probabilities, called (“beta”), from the model.

```
IMDB_topics <- tidy(IMDB_lda, matrix = "beta")
IMDB_topics
```

```
## # A tibble: 713,496 x 3
##    topic term             beta
##    <int> <chr>           <dbl>
##  1     1 accustomed 0.0000102
##  2     2 accustomed 0.0000173
##  3     3 accustomed 0.00000972
##  4     4 accustomed 0.0000175
##  5     5 accustomed 0.0000144
##  6     6 accustomed 0.0000184
##  7     1 agenda     0.0000403
##  8     2 agenda     0.00000599
##  9     3 agenda     0.0000622
## 10     4 agenda     0.0000585
## # ... with 713,486 more rows
```

We could use dplyr’s slice_max() to find the 10 terms that are most common within each topic and create a visualization.

```
top_terms <- IMDB_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()
```

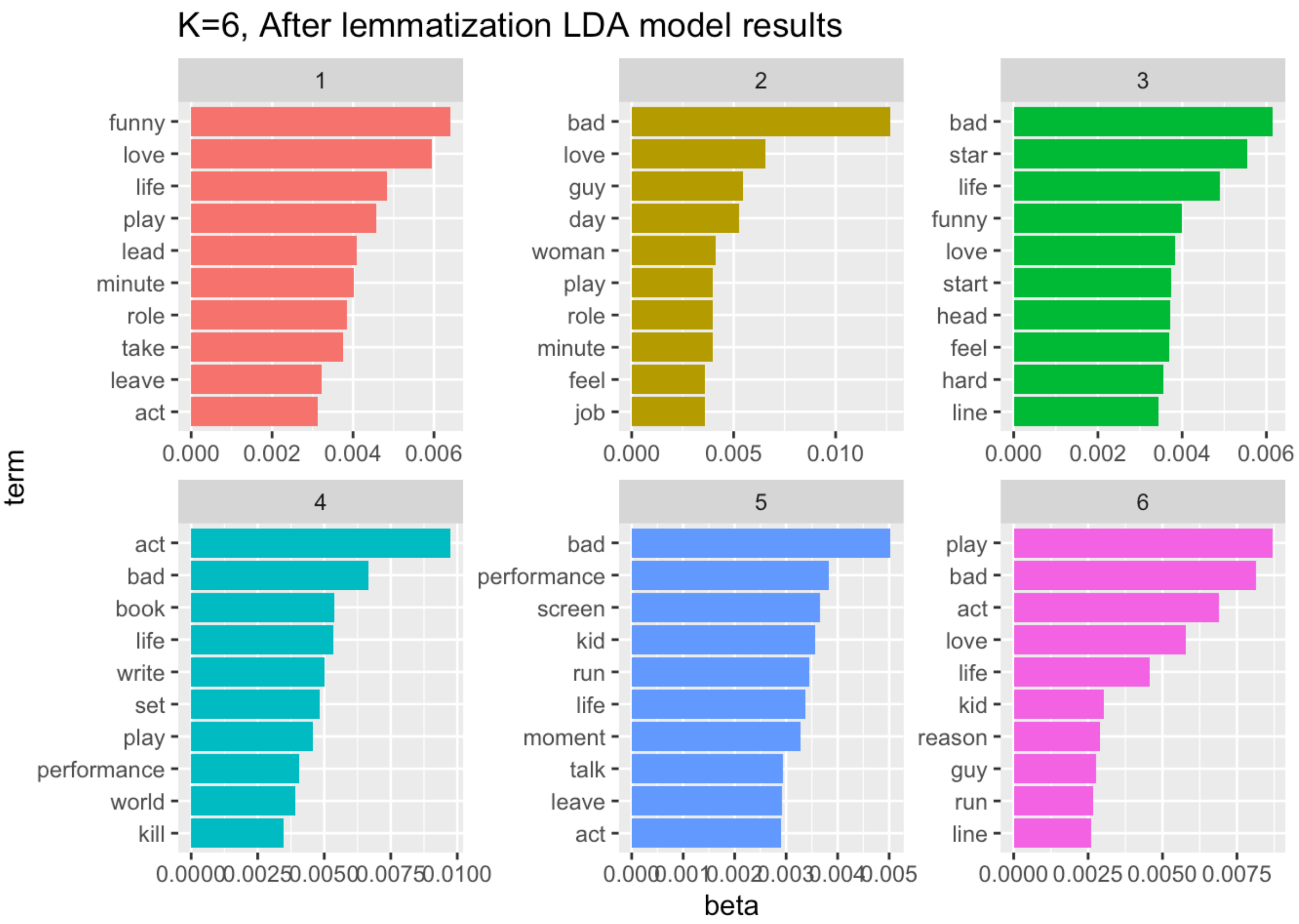


Now we use data after stemming and lemmatization to see if we have an improved results for topic identification. Because stemming would convert “plays” to “plai” which can be confusing for us to interpret the modeling results, we use lemmatized data to fit the model.

```
# LDA model fitting after stemming and lemmatization

lemma_dtm <- lemmatization%>% count(review_number, lemma) %>%
  cast_dtm(review_number, lemma, n)
# lemma_dtm still has sparsity of 100%, but has less terms: 100852

lemma_lda <- LDA(lemma_dtm, k, control = list(seed = 1234))
lemma_terms <- as.data.frame(topicmodels::terms(lemma_lda, 30),
                             stringsAsFactors = FALSE)
lemma_terms[1:k]
```

```
##      Topic 1     Topic 2  Topic 3     Topic 4     Topic 5 Topic 6
## 1     funny         bad      bad         act         bad    play
## 2      love        love     star         bad performance     bad
## 3      life         guy     life        book      screen     act
## 4      play         day    funny        life         kid    love
## 5      lead       woman     love       write         run    life
## 6    minute        play    start         set        life     kid
## 7      role        role     head        play      moment  reason
## 8      take      minute     feel performance        talk     guy
## 9     leave        feel     hard       world       leave     run
## 10      act         job     line        kill         act    line
## 11 original      action     live     version       begin    girl
## 12      fan        find     take         eye        girl   laugh
## 13     feel         boy   action       funny        time   write
## 14     star       child   series         fun         set  comedy
## 15    woman         act    shoot         guy      horror    kill
## 16    start         fan     main       enjoy        hard  horror
## 17 audience        girl      day        time       music    mind
## 18     line       music   horror       leave      friend  couple
## 19   script       leave    short      series       world    talk
## 20      boy         fun    sense        bite      family    fall
## 21    shoot      comedy     play    remember       sound    find
## 22    actor performance  picture        feel        love   enjoy
## 23    wrong      friend       tv        hope  completely  screen
## 24   happen       bring    write       house       laugh    call
## 25      run        time   family       death         low   money
## 26     kill        lead  classic       woman     version    true
## 27     call    original    piece        role      direct    bite
## 28      die      happen      low        wife       death   sense
## 29    black      script     dead        live        tell    idea
## 30    house  experience original      comedy      series    view
```

```
lemma_topics <- tidy(lemma_lda, matrix = "beta")

lemma_top_terms <- lemma_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

lemma_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()+
  labs(title = "K=6, After lemmatization LDA model results")
```



The most common words for each topic now seems more exclusive and words like “played” don’t appear in the plot. However, based on the output, setting topic(K) to 6 cannot really help us distinguish one topic from the other one by eyeballing. So we increase K to 10 given the data size of our dataset and the results improve by having less common words associated with each topic.

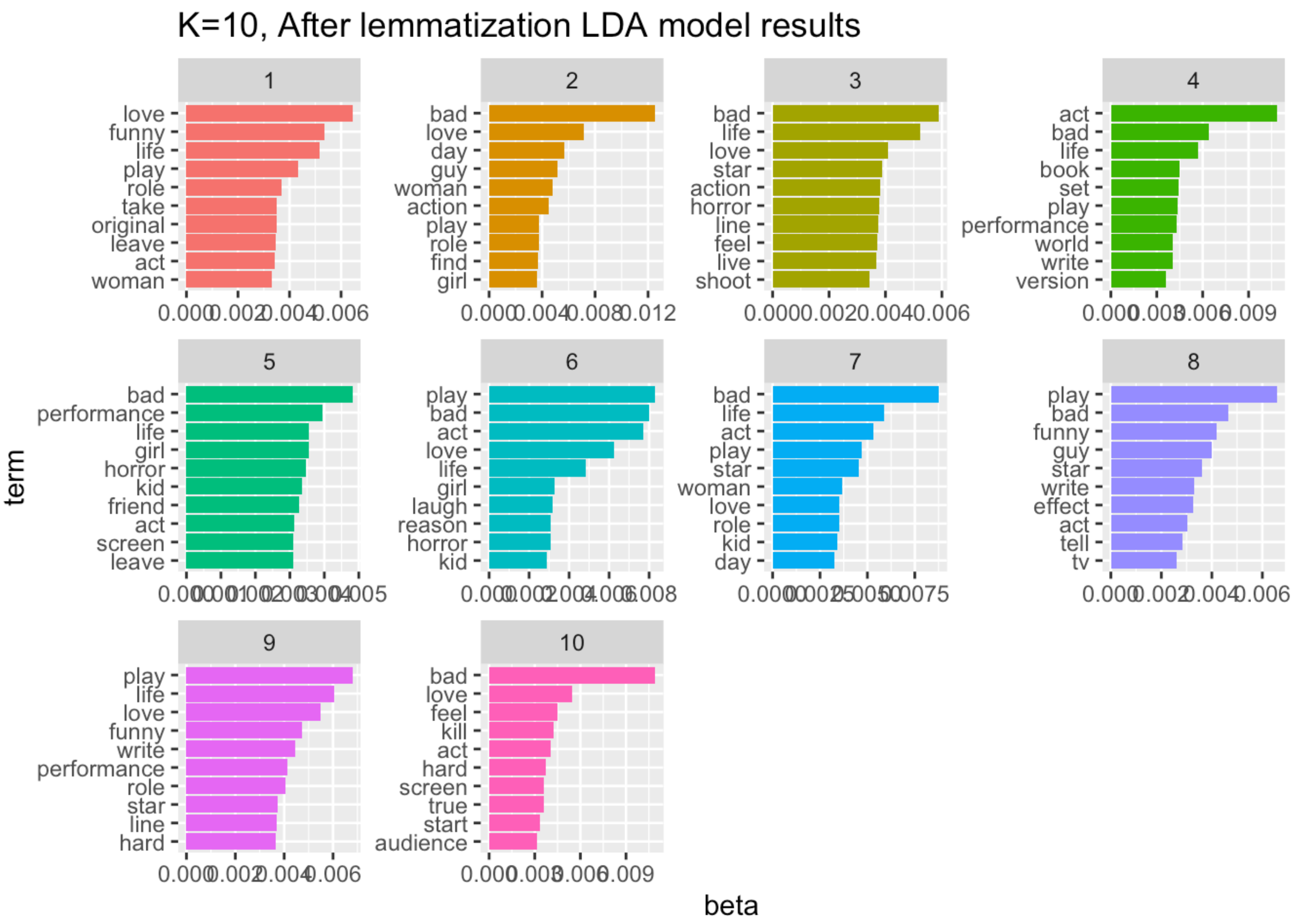
```
# choose nnumber of topics, 10
k = 10
IMDB_lda <- LDA(IMDB_dtm, k, control = list(seed = 1234))

IMDB_topics <- tidy(IMDB_lda, matrix = "beta")
imdb.terms <- as.data.frame(topicmodels::terms(IMDB_lda, 30),
                            stringsAsFactors = FALSE)
imdb.terms[1:k]
```

```
##        Topic 1      Topic 2    Topic 3      Topic 4  Topic 5      Topic 6
## 1       acting         love        bad          bad      bad       acting
## 2        music      minutes     acting       acting     love          bit
## 3        funny        funny     horror         life    world        music
## 4          bit        world     father      minutes    music         role
## 5        times       acting       life       comedy     life       played
## 6       played         feel       star        worst     role       series
## 7         life       horror   american        funny      set           tv
## 8  performance        worst     world         bit original    original
## 9       screen     original     takes      script     true       worth
## 10        book  performance      play         guy      dvd       times
## 11       actor        sense       guy         day   script        true
## 12      series          bad      dead       times    worst         guy
## 13       enjoy       family  terrible       sense   family        feel
## 14      comedy        enjoy     money        read     live         war
## 15        shot         evil     sound performance    times         fun
## 16     watched         main  dialogue       sound    actor        book
## 17      family         girl    action        shot     half      screen
## 18      father        plays       day       worth     mind        read
## 19        hard        actor    person        true    woman     version
## 20        girl        found      head      stupid  version performance
## 21     finally     american    screen   hollywood    sense   excellent
## 22   beautiful         wife   picture        book    money        girl
## 23      script         hard        tv       short    match         low
## 24      friend performances      past      reason   horror        home
## 25       plays       killer excellent          tv    night        main
## 26        left   production  audience    terrible  finally       short
## 27        mind          guy      hope     effects american     picture
## 28         job       series     loved     special    funny      family
## 29     version          fun     lines      entire        5        dead
## 30         fun        death     funny      person    awful         bad
##         Topic 7   Topic 8      Topic 9   Topic 10
## 1           bad    series          bad       love
## 2         funny      life         love       life
## 3          life    acting         life     acting
## 4           set       bad       comedy        bad
## 5        horror      role          guy     action
## 6           day    action        funny     horror
## 7           guy  original        found       feel
## 8           fan      shot          war       book
## 9          role    screen          day     script
## 10       family      feel      classic   original
## 11        house     takes          fan        fun
## 12       budget     woman        plays      times
## 13         love      love        world        boy
## 14         left       set    recommend      found
## 15   completely      john         home    watched
## 16  performance     found entertaining      woman
## 17      effects   classic    beautiful      black
## 18         main    played         hard      death
## 19     terrible     worst          fun       hard
## 20         idea      idea    wonderful       play
## 21        wrong       dvd      effects       john
## 22        start      star        human       left
## 23       series    reason        worst       idea
## 24         girl        tv       acting      sense
## 25         head       guy        times        low
## 26      friends      girl        takes       main
## 27      friend  american           tv      music
## 28        game beautiful         nice        job
## 29        play     start     remember      short
## 30          tv     house       played completely
```

```
# LDA model fitting after stemming and lemmatization

lemma_dtm <- lemmatization%>% count(review_number, lemma) %>%
  cast_dtm(review_number, lemma, n)
# lemma_dtm has sparsity of 100%, but has less terms: 100852

lemma_lda <- LDA(lemma_dtm, k, control = list(seed = 1234))

lemma_topics <- tidy(lemma_lda, matrix = "beta")
lemma_terms <- as.data.frame(topicmodels::terms(lemma_lda, 30),
                             stringsAsFactors = FALSE)
lemma_terms[1:k]
```

```
##     Topic 1     Topic 2  Topic 3     Topic 4     Topic 5 Topic 6 Topic 7
## 1     love         bad      bad         act         bad    play     bad
## 2    funny        love     life         bad performance     bad    life
## 3     life         day     love        life        life     act     act
## 4     play         guy     star        book        girl    love    play
## 5     role       woman   action         set      horror    life    star
## 6     take      action   horror        play         kid    girl   woman
## 7 original        play     line performance      friend   laugh    love
## 8    leave        role     feel       world         act  reason    role
## 9      act        find     live       write      screen  horror     kid
## 10   woman        girl    shoot     version       leave     kid     day
## 11  script        feel     main         fun       laugh    line     job
## 12    line         fun    start        kill         run  comedy   music
## 13  happen      friend    funny        bite     version    mind  effect
## 14    feel         act   series        time       begin     guy    find
## 15  minute      minute     take       leave        time  couple   leave
## 16   shoot      happen      day         eye      family   money    time
## 17     fan       leave       tv       enjoy        love    kill  comedy
## 18   start      comedy     head      series       world    find   enjoy
## 19     die         boy  picture        hope      moment    call    lead
## 20   wrong      horror    short        feel        talk    fall  moment
## 21    lead         fan   family       woman         set     run    half
## 22   black performance     play         guy        move    idea     guy
## 23    call      script original       funny       music    bite     dvd
## 24     boy       music    sense        live        mind   enjoy    fall
## 25  expect         job    black      comedy      series   write     run
## 26   money        time   writer   excellent         low  script   death
## 27   laugh    original      war    remember        hour  friend     set
## 28    kill        move  classic        lose      script    talk    nice
## 29   actor      expect     hear       laugh         war   human  action
## 30    star       bring      low          tv      action  viewer    main
##        Topic 8     Topic 9 Topic 10
## 1         play        play      bad
## 2          bad        life     love
## 3        funny        love     feel
## 4          guy       funny     kill
## 5         star       write      act
## 6        write performance     hard
## 7       effect        role   screen
## 8          act        star     true
## 9         tell        line    start
## 10          tv        hard audience
## 11       leave       start     book
## 12    audience         bad   minute
## 13       night        feel     girl
## 14        take        lead    funny
## 15       world         kid      set
## 16         run      minute    music
## 17         boy         set     call
## 18      moment       begin  episode
## 19      series       world   family
## 20       sound        bite      run
## 21         fan         fan      war
## 22     classic        take american
## 23        kill        head   friend
## 24      minute        live   comedy
## 25      family       black    fight
## 26 performance         guy     lead
## 27       laugh       short   direct
## 28        lead         act    child
## 29        edit         eye    video
## 30     version       actor     view
```

```
lemma_top_terms <- lemma_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

lemma_top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()+
  labs(title = "K=10, After lemmatization LDA model results")
```



Besides estimating each topic as a mixture of words, LDA also models each document as a mixture of topics. We can examine the per-document-per-topic probabilities, called (“gamma”), with the matrix = “gamma” argument to tidy().

```
IMDB_gamma <- tidy(IMDB_lda, matrix = "gamma")
IMDB_gamma
```

```
## # A tibble: 500,000 x 3
##    document topic  gamma
##    <chr>    <int>  <dbl>
##  1 1            1 0.102
##  2 2            1 0.100
##  3 3            1 0.101
##  4 4            1 0.101
##  5 5            1 0.0988
##  6 6            1 0.0999
##  7 7            1 0.0981
##  8 8            1 0.100
##  9 9            1 0.101
## 10 10           1 0.0995
## # ... with 499,990 more rows
```

7. Topic Modeling extension: Determine k number of topics

IN section 6, we assume the number of topics to be 6 to conduct the LDA since not pre-topic tags were provided. However, one major aspect of LDA, is that we need to know the exact k number of optimal topics for the documents. In order to accomplish this task, we are going to use a harmonic mean method to determine k based on Martin Ponweiser's thesis.(see README.md file)

First, we set up the fuction for computing the harmonic mean

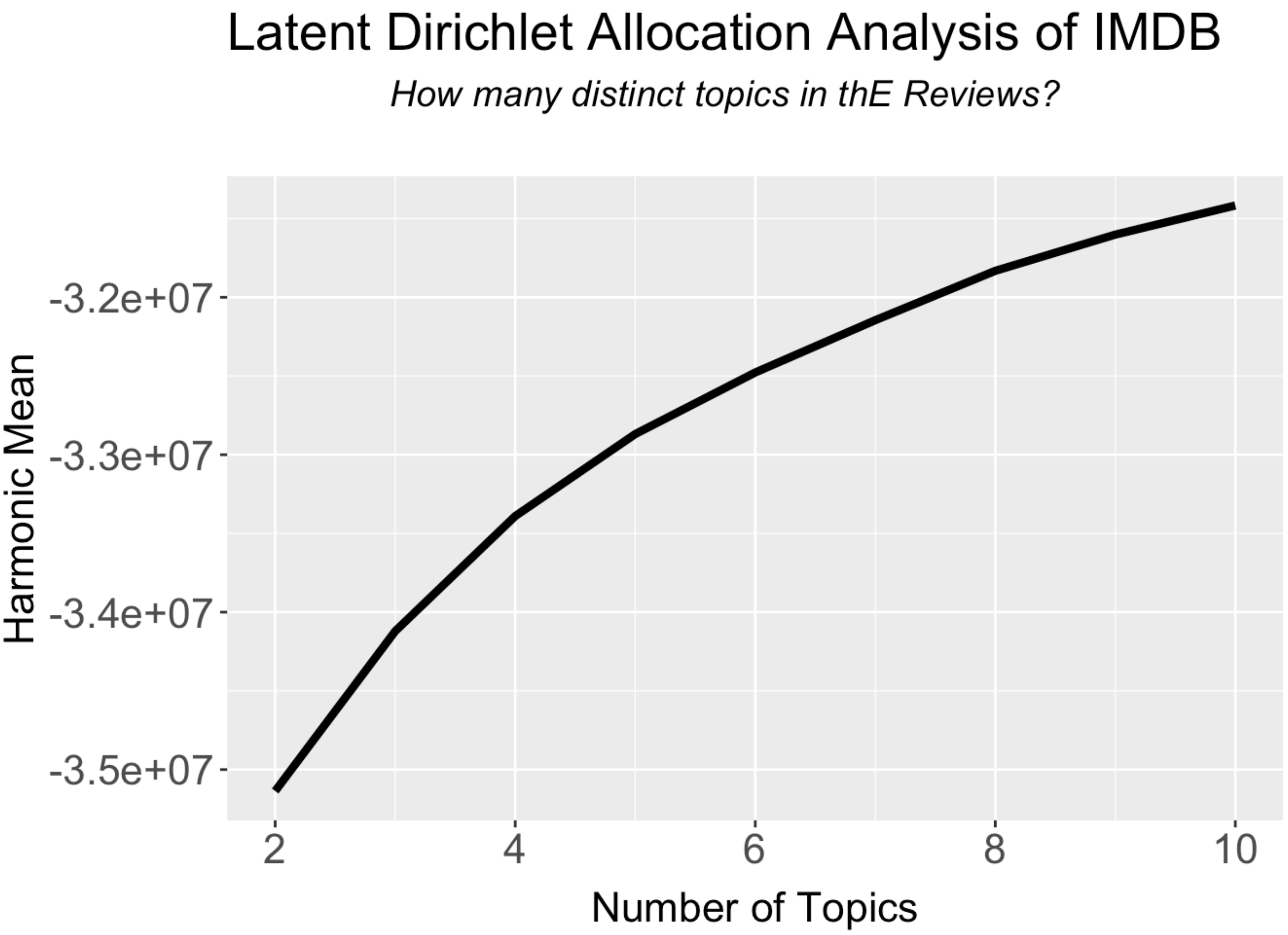
```r
# The harmonic mean function
harmonicMean <- function(logLikelihoods, precision = 2000L) {
  llMed <- median(logLikelihoods)
  as.double(llMed - log(mean(exp(-mpfr(logLikelihoods,
                                       prec = precision) + llMed))))
}
```

In order to find the best value for k, we do this over a sequence of topic models with different vales for k. This will generate numerous topic models with different numbers of topics, creating a vector to hold the k values.

```r
up_k <- 10

# We will use a sequence of numbers from 2 to up_k
seqk <- seq(2, up_k, 1)
burnin <- 1000
iter <- 1000
keep <- 50
fitted_many <- lapply(seqk, function(k) topicmodels::LDA(IMDB_dtm, k = k,
method = "Gibbs", control = list(burnin = burnin, iter = iter, keep = keep)))

# extract logliks from each topic
logLiks_many <- lapply(fitted_many, function(L)  L@logLiks[-c(1:(burnin/keep))])

# compute harmonic means
hm_many <- sapply(logLiks_many, function(h) harmonicMean(h))
```

We could visualize the results of harmonic means by plotting the results

```
ldaplot <- ggplot(data.frame(seqk, hm_many), aes(x=seqk, y=hm_many)) +
  geom_path(lwd=1.5) +
  theme(text = element_text(family= NULL),
        axis.title.y=element_text(vjust=1, size=16),
        axis.title.x=element_text(vjust=-.5, size=16),
        axis.text=element_text(size=16),
        plot.title=element_text(size=20)) +
  xlab('Number of Topics') +
  ylab('Harmonic Mean') +
  ggtitle(expression(atop("Latent Dirichlet Allocation Analysis of IMDB",
                          atop(italic("How many distinct topics in thE Reviews?"),
                               ""))))
ldaplot
```



From the plot above, we can see that the optimal number of topics for our model is 10.

The following code returns the optimal number of topics:

```
seqk[which.max(hm_many)]
```

```
## [1] 10
```