EBC4223 - Assignment 2

Quang Phong & Robert Agatić

2/22/2022

INTRODUCTION

In this assignment we will learn how to transform the review text into topics and word embedding underlying the text. We can use these extracted topics and embeddings for subsequent analyses.

```
# Load necessary libraries
library(dplyr) # used for data manipulation
library(tm) # used for text mining
library(ldatuning) # used for identifying the number of topics in a text corpus
library(topicmodels) # provides basic infrastructure for fitting topic models based on data structures
library(tidytext) # applies the principles of the tidyverse to analyzing text
library(wordcloud2) # a fast visualization tool for creating wordcloud
library(wordcloud) # visual representation of text data
library(reshape2) # used for data manipulation
library(data.table) # used for fast aggregation of multiple columns
library(word2vec) # used for getting word vectors and document vecotrs based on word2vec model
```

Question 1: Load data

```
# Load the dataset from the assignment 1
df_Reviews <- readRDS("df_Reviews.RDS")
df_Movies <- readRDS("df_Movies.RDS")
```

DATA PREPROCESSING AND ANALYSIS

Question 2: Create corpus

For latent dirichlet allocation (LDA), we need to view our set (corpus) of reviews as 2469 separate documents. We can do so by combining and applying the VectorSource() with the Corpus() functions to create a corpus of text. A corpus is a list of a document.

```
# Create a corpus of text for the reviews
corpus_Reviews <- Corpus(VectorSource(df_Reviews$review_text2))

# Get the dimensions of the corpus
str(corpus_Reviews)
```

```
## Classes 'SimpleCorpus', 'Corpus'  hidden list of 3
```

```
##  $ content: chr [1:2469] "aveng endgam expans sequel aveng infin war also whole last year marvel cin
##  $ meta   :List of 1
##   ..$ language: chr "en"
##   ..- attr(*, "class")= chr "CorpusMeta"
##  $ dmeta  :'data.frame': 2469 obs. of  0 variables
```

Question 3: Clean corpus

Now we clean the corpus.

```
# Transform the corpus to a word frequency matrix
dtm <- DocumentTermMatrix(corpus_Reviews)

# Inspect detailed information of the corpus
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 2469, terms: 23931)>>
## Non-/sparse entries: 259345/58826294
## Sparsity           : 100%
## Maximal term length: 37
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   charact film get good just like movi one stori time
##   1045      11   23   3    6    4   22    1  13     5    1
##   1874      12   23   4    8   12    8    4  12     2    5
##   1978      10   13   1    3    8    4   12   5     3    6
##   2197       5   23   3    1    5    8    2  15     2    7
##   290        1   22   6    1    1    6   18   7     1    3
##   475       10   18   2    6    3    9    8  11     2    4
##   65         7    8  10    2    7   16    9   7     1    2
##   674       15   17   5    3    8    6    1  18     2    7
##   78         5   20   4    6    6   10    8  11     8    5
##   9          2    0   1    3    0    0    2   5     1   11
```

```
# Removing words that do not occur in at least 10% of the documents
dtm2 <- removeSparseTerms(dtm, 0.9)

# Inspect detailed information of the corpus
inspect(dtm2)
```

```
## <<DocumentTermMatrix (documents: 2469, terms: 154)>>
## Non-/sparse entries: 68612/311614
## Sparsity           : 82%
## Maximal term length: 9
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   charact film get good just like movi one stori time
##   1045      11   23   3    6    4   22    1  13     5    1
##   1145       6   16   6    5    8    6    3  16     7    1
##   1874      12   23   4    8   12    8    4  12     2    5
##   1978      10   13   1    3    8    4   12   5     3    6
```

```
##   2197       5   23   3    1    5    8    2  15     2    7
##   357       15   15   7    4    5    5    4   9     3    2
##   400        7   13   3    6    9   14    7  19     4    5
##   475       10   18   2    6    3    9    8  11     2    4
##   674       15   17   5    3    8    6    1  18     2    7
##   78         5   20   4    6    6   10    8  11     8    5
```

From 29321 terms in the former matrix, we have reduced the number to only 154 terms in the latter matrix.

Now we need to remove documents in which none of these 154 terms appear. This is necessary before doing LDA.

```
# Create a matrix out of the new DocumentTermMatrix
matrix_dtm <- as.matrix(dtm2)

# Compute the rowSums() for each document
rowSums <- rowSums(matrix_dtm)

# See which document(s) should be removed
which(rowSums == 0)
```

```
## 1031 2255
## 1031 2255
```

```
# Remove them from dtm2
dtm2 <- dtm2[rowSums > 0, ]

# Inspect detailed information of the corpus
inspect(dtm2)
```

```
## <<DocumentTermMatrix (documents: 2467, terms: 154)>>
## Non-/sparse entries: 68612/311306
## Sparsity           : 82%
## Maximal term length: 9
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   charact film get good just like movi one stori time
##   1045      11   23   3    6    4   22    1  13     5    1
##   1145       6   16   6    5    8    6    3  16     7    1
##   1874      12   23   4    8   12    8    4  12     2    5
##   1978      10   13   1    3    8    4   12   5     3    6
##   2197       5   23   3    1    5    8    2  15     2    7
##   357       15   15   7    4    5    5    4   9     3    2
##   400        7   13   3    6    9   14    7  19     4    5
##   475       10   18   2    6    3    9    8  11     2    4
##   674       15   17   5    3    8    6    1  18     2    7
##   78         5   20   4    6    6   10    8  11     8    5
```

Two rows, 1031 and 2255, have been removed from the dtm2, resulting in the final number of 2467 rows.

Question 4: Determine the number of topics

Before we can estimate a topic model, we first need to determine the number of topics. This can be using the "ldatuning".

```
result <- FindTopicsNumber(
  dtm2,
  topics = seq(from = 2, to = 25, by = 1),
  metrics = c("Deveaud2014"),
  method = "Gibbs",
  control = list(seed = 12),
  verbose = TRUE
  )
```

Question 5: Estimate 12 latent topics

The package topicmodels is used to estimate the 12 latent topics, Apply the LDA function to the document term matrix with the number of topics as determined previously, and the same model specifications as in 4.

```
lda <- LDA(dtm, k = 12,
           method= "Gibbs",
           control = list(seed = 1010, burnin = 4000, iter=2000)
           )
```

Question 6: view topics

Let's see the first 10 words of each topic. Following is the first method.

```
terms(lda , 10)
```

```
##       Topic 1   Topic 2 Topic 3 Topic 4     Topic 5  Topic 6  Topic 7
##  [1,] "perform" "best"  "like"  "movi"      "love"   "film"   "famili"
##  [2,] "man"     "two"   "get"   "watch"     "dog"    "play"   "horror"
##  [3,] "music"   "one"   "thing" "good"      "best"   "comedi" "find"
##  [4,] "john"    "film"  "know"  "just"      "film"   "stori"  "hous"
##  [5,] "life"    "war"   "one"   "enjoy"     "famili" "make"   "escap"
##  [6,] "stori"   "boy"   "dont"  "well"      "day"    "life"   "death"
##  [7,] "song"    "high"  "even"  "like"      "way"    "want"   "get"
##  [8,] "cat"     "scene" "peopl" "realli"    "time"   "women"  "tri"
##  [9,] "time"    "girl"  "that"  "entertain" "work"   "role"   "will"
## [10,] "jewel"   "world" "just"  "quit"      "home"   "real"   "tell"
##       Topic 8     Topic 9   Topic 10   Topic 11 Topic 12
##  [1,] "power"     "film"    "action"   "anim"   "one"
##  [2,] "marvel"    "charact" "two"      "kid"    "film"
##  [3,] "comic"     "stori"   "john"     "origin" "also"
##  [4,] "action"    "much"    "black"    "stori"  "realli"
##  [5,] "franchis"  "plot"    "new"      "new"    "like"
##  [6,] "univers"   "though"  "cop"      "voic"   "even"
##  [7,] "part"      "littl"   "kill"     "king"   "see"
##  [8,] "superhero" "make"    "agent"    "back"   "charact"
##  [9,] "battl"     "fun"     "offic"    "will"   "say"
## [10,] "termin"    "great"   "director" "like"   "much"
```

Another way to do this is using tidytext package.

```
lda_topics <- tidy(lda, matrix = "beta")

lda_top_10_terms <- lda_topics %>%
```

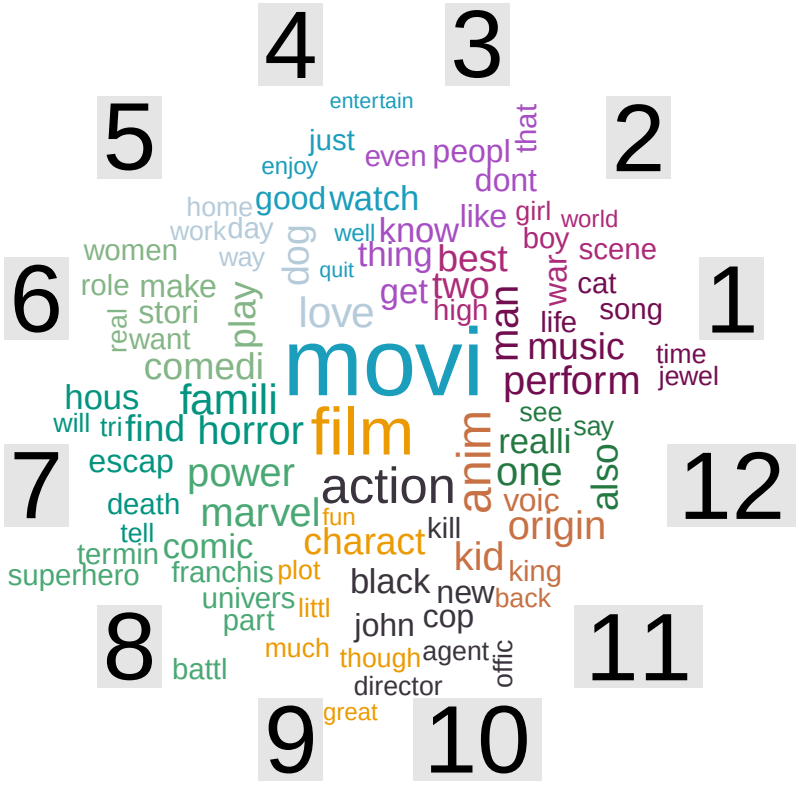
```
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

#Have a look of the first 20 rows
lda_top_10_terms[1:20, ]
```

```
## # A tibble: 20 x 3
##    topic term       beta
##    <int> <chr>     <dbl>
##  1     1 perform 0.0111
##  2     1 man     0.0106
##  3     1 music   0.00965
##  4     1 john    0.00801
##  5     1 life    0.00782
##  6     1 stori   0.00729
##  7     1 song    0.00673
##  8     1 cat     0.00647
##  9     1 time    0.00606
## 10     1 jewel   0.00561
## 11     2 best    0.0101
## 12     2 two     0.00995
## 13     2 one     0.00902
## 14     2 film    0.00871
## 15     2 war     0.00774
## 16     2 boy     0.00678
## 17     2 high    0.00670
## 18     2 scene   0.00596
## 19     2 girl    0.00531
## 20     2 world   0.00469
```

We can go a little bit further by visualizing comparison clouds of the 10 top words in each topic using wordcloud, wordcloud2, and reshape 2 package.

```
lda_top_10_terms %>%
  acast(term ~ topic, value.var = "beta", fill = 0) %>%
  comparison.cloud(colors = c("#720d50", "#ac2b78", "#a64ec2",
                              "#1b9dbc", "#b5cbda", "#84b586",
                              "#03947f", "#4aa874", "#ec9a02",
                              "#3b343e", "#c77144", "#247942" ),
                   max.words = 120, # 120 because this shows top 10 of 12 topics.
                   scale = c(3,.4)) # adjust scale so all 120 words have space to be plotted
```



Based on this observation, we can label a few topics. For example, topic 9 is about film elements such as character, plot, story. Topic 8 is about superhero-related world. Topic 7 is about horror movies. Topic 11 is about kid. Topic 1 is about music with elements such as song, performer, and music. Topic 4 is about overall tags such as good, enjoy, well, like, entertain. Topic 5 is about the audience's life: family, day, time, work, home, live, life. Topic 6 is about humor with words like comedy, funny, humor. However, it is still challenging to label all 12 topics differently.

Question 7: Topic probability

Using the topics() function, we will extract the topic with the highest probability per review from the model. Then we will compare how these topics relate to the review-level polarity scores computed previously.

```
# Add the topic with the highest probability to each document in reviews data
df_Reviews$topic <- topics(lda, 1)

# Get an quick look at the first 5 rows
df_Reviews[1:5,c(1,8,9)]
```

```
##   movie_id  polarity topic
## 1        1 1.4309933     8
## 2        1 0.2647585     8
## 3        1 0.3570575    12
## 4        1 0.3494816     8
## 5        1 1.0629821     9
```

```
# Compute the average polarity scores of reviews
df_Reviews %>%
```

```
  group_by(topic) %>%
  summarize(avg_polarity = mean(polarity))
```

```
## # A tibble: 12 x 2
##    topic avg_polarity
##    <int>        <dbl>
##  1     1       0.189
##  2     2       0.198
##  3     3       0.0622
##  4     4       0.528
##  5     5       0.361
##  6     6       0.216
##  7     7      -0.228
##  8     8       0.141
##  9     9       0.422
## 10    10      -0.0850
## 11    11       0.207
## 12    12       0.484
```

We can see that reviews that fall in topic 4 shows highest polarity (0.53), followed by topic 12 (0.48) and topic 9 (0.42). Reviews belonging to topic 7 and 10 displays negative polarity. Topic 7 includes words such as horror, escape, death, etc. while topic 10 contains words such as action, black, cop, kill, etc.

Question 8: Topic counts and regression

Now, we examine how the topics relate to revenue.

```
# Count how often a topic is mentioned in relation to a movie
array_count <- array(0, c(100,12))

# Create data frame to count number of topics in reviews about every movie
df_count <- df_Reviews %>%
  group_by(movie_id, topic) %>%
  count()

# Get a quick look at the data frame
head(df_count)
```

```
## # A tibble: 6 x 3
## # Groups:   movie_id, topic [6]
##   movie_id topic     n
##      <int> <int> <int>
## 1        1     3     1
## 2        1     4     4
## 3        1     6     1
## 4        1     8    13
## 5        1     9     3
## 6        1    12     3
```

```
# Run loop over the df_count table to add count numbers to the array_count
for (i in 1:nrow(df_count)) {
  a <- as.numeric(df_count[i, 1])
  b <- as.numeric(df_count[i, 2])
```

```
  array_count[a, b] <- as.numeric(df_count[i, 3])
}

head(array_count)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## [1,]    0    0    1    4    0    1    0   13    3     0     0     3
## [2,]    0    0    4    3    0    1    0    0    0     0    15     2
## [3,]    0    1    4    2    0    0    0    1    3     0    11     3
## [4,]    0    0    1    3    0    0    1    0    4     0    14     2
## [5,]    0    0    0    1    1    0    1    0   20     0     0     2
## [6,]    0    0    3    3    0    0    0   13    2     0     0     4
```

```
# Add the newly created array to the df_Movies before running regression
df_Movies2 <- data.table(df_Movies)
df_Movies2[, `:=` (topic1 = array_count[,1],
                   topic2 = array_count[,2],
                   topic3 = array_count[,3],
                   topic4 = array_count[,4],
                   topic5 = array_count[,5],
                   topic6 = array_count[,6],
                   topic7 = array_count[,7],
                   topic8 = array_count[,8],
                   topic9 = array_count[,9],
                   topic10 = array_count[,10],
                   topic11 = array_count[,11],
                   topic12 = array_count[,12])]

head(df_Movies2)
```

```
##    Rank                                        Release     Gross Max Th   Open
## 1:    1                              Avengers: Endgame 858373000   4662 Apr 26
## 2:    2                                  The Lion King 543638043   4802 Jul 19
## 3:    3 Star Wars: Episode IX - The Rise of Skywalker 515202542   4406 Dec 20
## 4:    4                                     Frozen II 477373578   4440 Nov 22
## 5:    5                                   Toy Story 4 434038008   4575 Jun 21
## 6:    6                                 Captain Marvel 426829839   4310  Mar 8
##     Close                          Distributor movie_id averageReviewLength
## 1: Sep 12 Walt Disney Studios Motion Pictures        1              2254.16
## 2:  Dec 5 Walt Disney Studios Motion Pictures        2              1732.28
## 3:      - Walt Disney Studios Motion Pictures        3              2198.16
## 4:      - Walt Disney Studios Motion Pictures        4              1809.56
## 5:  Dec 5 Walt Disney Studios Motion Pictures        5              1638.60
## 6:  Jul 4 Walt Disney Studios Motion Pictures        6              1912.32
##    reviewPolarity stdPolarity  lnGross topic1 topic2 topic3 topic4 topic5
## 1:      0.4683305   0.4943186 20.57055      0      0      1      4      0
## 2:      0.2527416   0.5542439 20.11379      0      0      4      3      0
## 3:      0.1288735   0.4316331 20.06007      0      1      4      2      0
## 4:      0.3518811   0.7113025 19.98381      0      0      1      3      0
## 5:      0.4612540   0.4884192 19.88864      0      0      0      1      1
## 6:      0.6994693   0.4925952 19.87190      0      0      3      3      0
##    topic6 topic7 topic8 topic9 topic10 topic11 topic12
## 1:      1      0     13      3       0       0       3
```

```
## 2:      1      0      0      0       0      15       2
## 3:      0      0      1      3       0      11       3
## 4:      0      1      0      4       0      14       2
## 5:      0      1      0     20       0       0       2
## 6:      0      0     13      2       0       0       4
```

```
# Run regression with revenue as dependent variable
model1 <- lm(Gross ~ topic1+topic2+topic3+topic4+topic5+topic6
             +topic7+topic8+topic9+topic10+topic11+topic12, data = df_Movies2)

summary(model1)
```

```
##
## Call:
## lm(formula = Gross ~ topic1 + topic2 + topic3 + topic4 + topic5 +
##     topic6 + topic7 + topic8 + topic9 + topic10 + topic11 + topic12,
##     data = df_Movies2)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -197372298  -52764872  -15971731   30217926  621822812
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  59305209  123157638   0.482   0.6313
## topic1       -1873508    5411033  -0.346   0.7300
## topic2       -1581725    5485132  -0.288   0.7738
## topic3         876721    6489816   0.135   0.8929
## topic4       -4962041    7518383  -0.660   0.5110
## topic5        -594385    5540453  -0.107   0.9148
## topic6        -638735    5423714  -0.118   0.9065
## topic7       -1123616    5511197  -0.204   0.8389
## topic8        9307335    5940581   1.567   0.1208
## topic9        3470169    6317429   0.549   0.5842
## topic10      -1478678    5531230  -0.267   0.7898
## topic11       8027362    5477587   1.465   0.1464
## topic12      21816434   10507298   2.076   0.0408 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 122300000 on 87 degrees of freedom
## Multiple R-squared:  0.2916, Adjusted R-squared:  0.1939
## F-statistic: 2.985 on 12 and 87 DF,  p-value: 0.001568
```

```
# Run regression with ln of revenue as dependent variable
model2 <- lm(lnGross ~ topic1+topic2+topic3+topic4+topic5+topic6
             +topic7+topic8+topic9+topic10+topic11+topic12, data = df_Movies2)

summary(model2)
```

```
##
## Call:
## lm(formula = lnGross ~ topic1 + topic2 + topic3 + topic4 + topic5 +
```

```
##     topic6 + topic7 + topic8 + topic9 + topic10 + topic11 + topic12,
##     data = df_Movies2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.30469 -0.51930 -0.09746  0.38772  1.78559
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.4943047  0.7596586  23.029  < 2e-16 ***
## topic1      -0.0129086  0.0333762  -0.387  0.69988
## topic2      -0.0004671  0.0338333  -0.014  0.98902
## topic3       0.0134300  0.0400304   0.335  0.73806
## topic4      -0.0105577  0.0463747  -0.228  0.82044
## topic5       0.0041729  0.0341745   0.122  0.90310
## topic6      -0.0037277  0.0334544  -0.111  0.91154
## topic7      -0.0012087  0.0339941  -0.036  0.97172
## topic8       0.0546240  0.0366426   1.491  0.13965
## topic9       0.0261509  0.0389670   0.671  0.50393
## topic10     -0.0112536  0.0341176  -0.330  0.74231
## topic11      0.0647954  0.0337867   1.918  0.05842 .
## topic12      0.1782046  0.0648109   2.750  0.00726 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7543 on 87 degrees of freedom
## Multiple R-squared:  0.353,  Adjusted R-squared:  0.2638
## F-statistic: 3.956 on 12 and 87 DF,  p-value: 7.311e-05
```

```
# Run regression with revenue as dependent variable and topic12 as the only feature
model3 <- lm(Gross ~ topic12, data = df_Movies2)

summary(model3)
```

```
##
## Call:
## lm(formula = Gross ~ topic12, data = df_Movies2)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -146140005  -64923961  -23572262   17164934  702379297
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 52199256   19197070   2.719 0.007743 **
## topic12     34598149    9069763   3.815 0.000239 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 127700000 on 98 degrees of freedom
## Multiple R-squared:  0.1293, Adjusted R-squared:  0.1204
## F-statistic: 14.55 on 1 and 98 DF,  p-value: 0.0002386
```

```r
# Run regression with log of revenue as dependent variable and topic12 as the only feature
model4 <- lm(lnGross ~ topic12, data = df_Movies2)

summary(model4)
```

```
##
## Call:
## lm(formula = lnGross ~ topic12, data = df_Movies2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.48195 -0.61377 -0.09351  0.57753  2.18643
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  17.6256     0.1213 145.326  < 2e-16 ***
## topic12       0.2528     0.0573   4.412 2.63e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.807 on 98 degrees of freedom
## Multiple R-squared:  0.1657, Adjusted R-squared:  0.1572
## F-statistic: 19.47 on 1 and 98 DF,  p-value: 2.627e-05
```

Based on the two models, topic 12 has a statistically significant positive association with the revenue although the size of coefficient is small.

Topic models try to extract latent topics from text using a generative model, which yields interpretable topic vectors. An alternative approach is that of word embeddings: Vectors that give the correlation between words, and allow to predict relations between words accurately. However, the vectors themselves are not interpretable, as they are numeric representations of the underlying text space. Thus, they mainly serve as input to predictive models. In R, we can use the library word2vec to estimate word embedding.

Question 9: Estimate word embedding by skip-gram model

We use the function word2vec() to estimate a skip-gram model on the review text.

```r
set.seed(0) # to make sure the same ouput will be obtained during knitting
sgm <- word2vec(df_Reviews$review_text)
embd <- as.matrix(sgm)

vocabulary_dt <- summary(sgm, type = "vocabulary")
```

Question 10: Word embeddings

To get a feeling for what word embeddings can do, we use the predict() function to estimate the words closed related to an input word. For example, try to predict which 5 words relate to e.g., 'war' and 'family, using type = 'nearest' and top_n = 5. Show some example with other words.

```r
predict(sgm, c("prison", "cop", "food", "kill", "wizard"), type = "nearest", top_n = 5)
```

```
## $prison
##    term1     term2 similarity rank
## 1 prison    limits  0.9629675    1
```

```
## 2 prison Gabriella  0.9628506    2
## 3 prison Literally  0.9622449    3
## 4 prison  warnings  0.9619958    4
## 5 prison Afterward  0.9619620    5
##
## $cop
##   term1      term2 similarity rank
## 1   cop    officer  0.9018584    1
## 2   cop mysterious  0.8993610    2
## 3   cop  assistant  0.8981683    3
## 4   cop       Dell  0.8916267    4
## 5   cop    Natalie  0.8908245    5
##
## $food
##   term1      term2 similarity rank
## 1  food  seemingly  0.9296557    1
## 2  food conditions  0.9291791    2
## 3  food       pain  0.9239454    3
## 4  food      dealt  0.9213982    4
## 5  food      legal  0.9204324    5
##
## $kill
##   term1    term2 similarity rank
## 1  kill     save  0.9512246    1
## 2  kill  protect  0.9435052    2
## 3  kill convince  0.9370518    3
## 4  kill  destroy  0.9315931    4
## 5  kill   escape  0.9314401    5
##
## $wizard
##    term1    term2 similarity rank
## 1 wizard   Hunter  0.9603999    1
## 2 wizard    elder  0.9592554    2
## 3 wizard overdose  0.9571517    3
## 4 wizard Katarina  0.9550224    4
## 5 wizard   Sultan  0.9549493    5
```

As we can see, the word embeddings have predicted words related to chosen words fairly accurately and reasonably. For example, "save", "protect", "convince", "escape", "destroy" are predicted to be related to "kill" and "officer", "mysterious", "assistant" are predicted to be related to "cop", which make sense.

Question 11: Combined embeddings

Taking this one step further, we can also create new embeddings by combining word embeddings, and subsequently generate predictions using these combined embeddings.

Store the result e.g., in a object pred. Now, we can use mathematical operations on the words we used to generate the embedding to generate a new embedding. For example, we could do pred["war",] + pred["family"], and store this embedding. Then, again using the predict function as in 10), but now with newdata the previously created new embedding, we could predict which words relate closest to terms including both 'war' and 'family'. Of course, we can also use subtraction and more words. Show some examples of different word combinations.

```
# First, use the predict() function as before, but now with type = 'embedding'.
pred <- predict(sgm, c("prison", "cop", "wizard", "kill"), type = "embedding")
head(pred)
```

```
##               [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## prison -0.32700586 -0.7404578 -0.5793890 -0.4890277  0.2834349 -0.379755
## cop     0.04939550 -0.6996601  0.4061074 -0.1405660 -1.6220175  1.343278
## wizard -0.23454700 -0.4341388 -0.6406150 -0.2265225  0.9242856  1.559523
## kill    0.01285268 -0.4726520 -1.2834169  0.5931116 -0.6861994  1.542368
##               [,7]       [,8]       [,9]      [,10]      [,11]       [,12]
## prison -0.3938553 -0.3520001 -0.2687165 -0.8108478 -2.1163363 -0.84192944
## cop    -0.1682402  0.7655241  0.3293126  0.7734701 -0.7490776 -1.29230714
## wizard -0.2202749 -0.2380556 -0.9644495 -0.4413480 -1.5092102 -0.73869264
## kill    0.5346541  1.1778374 -2.0938406 -0.6533548 -0.6358493 -0.07153145
##              [,13]       [,14]     [,15]      [,16]       [,17]       [,18]
## prison -1.9271637 -0.03605062 0.8093144  1.0476068 -0.35774076 -0.61812359
## cop    -0.6096261  0.87081009 1.1224734  0.7927240 -0.12512821  0.03333307
## wizard -1.7647300  0.07755910 1.4651372  1.1922184 -0.04160722  0.01439018
## kill   -0.4891476 -1.23803282 0.1084457 -0.4573556 -1.60912466  0.05616074
##              [,19]      [,20]        [,21]     [,22]      [,23]      [,24]
## prison -0.2671431 0.1603457 -0.087587520 0.9499843 -0.1497635  0.8348202
## cop    -0.6497918 0.8547211  0.002918076 1.6620344 -0.6395086  1.5786691
## wizard -0.4973513 0.7372524 -0.565449774 0.7390441 -0.3986529  1.5628698
## kill    0.2890097 0.6077628  0.100696541 0.7977399 -1.6338738 -0.6001760
##              [,25]       [,26]      [,27]      [,28]      [,29]     [,30]
## prison -1.8644089 -0.08264136  0.1125075 -1.0184455 -0.6810499 -1.981750
## cop    -1.0712856  0.19774994  0.6520666  0.1022232 -1.3492994 -2.926932
## wizard -1.7473783 -0.55956930  0.6550471 -0.9395607 -1.3465780 -2.083821
## kill   -0.4295177 -0.60751301 -0.8255349 -0.4060130 -1.8831412 -2.150137
##            [,31]      [,32]      [,33]      [,34]      [,35]     [,36]
## prison 3.204007 -0.3782987  0.5827514 -0.5492674  0.0474924 0.6527794
## cop    2.020836 -0.2624800 -0.7672018 -1.4374901  0.5716833 0.6792378
## wizard 3.247858  0.3337622  0.9997637 -1.3411865 -0.8323830 0.4668242
## kill   1.246869 -1.1442956  0.2988437 -1.7264196 -1.2192378 1.6804618
##              [,37]       [,38]       [,39]         [,40]       [,41]      [,42]
## prison  0.1280377  0.05332812 -0.84987295  1.1896333694 -0.05599492 -1.5607486
## cop    -1.6135182 -1.19498301  0.29722509 -0.7209005952  0.95087016  0.4485277
## wizard -0.1443366  0.30207479 -0.03567475 -0.0004378215 -0.01038814 -0.9728017
## kill   -1.4353257 -0.81998008  0.36775884  0.1832571626  0.38731492 -1.5767722
##              [,43]      [,44]      [,45]       [,46]      [,47]      [,48]
## prison -0.82334262 0.7718560 -2.7142498  0.50510335 -0.5297005  0.2458502
## cop     0.07692646 1.2043062 -1.5917248 -0.60034567 -0.2011669 -0.6585570
## wizard  0.06308567 0.7745911 -1.4024450  0.48448783 -1.1930766 -0.1198625
## kill    0.53175354 1.0410252  0.1597108 -0.09221104 -0.2295131  1.4322808
##              [,49]     [,50]
## prison  0.6386841 0.0396005
## cop    -0.7201813 0.8892425
## wizard -0.4133697 0.5738710
## kill   -0.1799371 0.6567686
```

```
emb1 <- pred["cop", ] + pred["wizard", ] +  pred["prison", ] + pred["kill",]

predict(sgm, emb1, type = "nearest", top_n = 10)
```

```
##          term similarity rank
```

```
## 1       happy  0.9997353    1
## 2   absurdity  0.9997265    2
## 3        Down  0.9987038    3
## 4  situations  0.9986644    4
## 5      Action  0.9985183    5
## 6       esque  0.9984964    6
## 7      hoping  0.9984335    7
## 8        down  0.9978453    8
## 9    enjoying  0.9975860    9
## 10         PG  0.9970816   10
```

```
emb2 <- - pred["cop", ] - pred["wizard", ] +  pred["prison", ] + pred["kill",]

predict(sgm, emb2, type = "nearest", top_n = 10)
```

```
##          term similarity rank
## 1  appreciate  0.8594252    1
## 2    overlook  0.8589099    2
## 3       alone  0.8559509    3
## 4    consider  0.8406356    4
## 5       leave  0.8226222    5
## 6     without  0.8197309    6
## 7      please  0.8190933    7
## 8       fully  0.8183423    8
## 9         buy  0.8089637    9
## 10    goodbye  0.8069286   10
```

Question 12: Document-level predictions

Finally, we can also use our Word2vec model to generate document-level predictions. This can be done using the doc2vec() function, with as inputs the model from 9) and the cleaned review text. This will generate word vectors for each review in the dataset. Using the polarity score from assignment 1 as dependent variable and the word vectors generated as independent variables, use linear regression to estimate which words vector relate to polarity. Shortly discuss your findings with respect to predictive ability. Note: as said, word embedding vectors are not interpretable, so there's no need to try and interpret the vectors as we did with topics in topic modelling.

```
embd_document <- as.data.frame(doc2vec(sgm, df_Reviews$review_text, type="embedding"))
embd_document$polarity <- df_Reviews$polarity
revenue_polarity <- lm(polarity~.,data=embd_document)
summary(revenue_polarity)
```

```
##
## Call:
## lm(formula = polarity ~ ., data = embd_document)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.97984 -0.30486 -0.00335  0.30384  1.90937
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.80889    0.18354  -4.407 1.09e-05 ***
```

```
## V1           0.08922    0.11836   0.754 0.451060
## V2          -0.10933    0.08782  -1.245 0.213246
## V3           0.31159    0.07316   4.259 2.13e-05 ***
## V4          -0.20621    0.08647  -2.385 0.017163 *
## V5          -0.08707    0.09459  -0.920 0.357433
## V6           0.29357    0.09377   3.131 0.001765 **
## V7           0.27233    0.11046   2.465 0.013756 *
## V8          -0.50753    0.08090  -6.274 4.16e-10 ***
## V9           0.13611    0.08214   1.657 0.097640 .
## V10          0.27861    0.11849   2.351 0.018789 *
## V11         -0.06552    0.05634  -1.163 0.244941
## V12          0.57001    0.11510   4.952 7.85e-07 ***
## V13          0.05871    0.09718   0.604 0.545833
## V14         -0.12344    0.11817  -1.045 0.296295
## V15          0.03091    0.10127   0.305 0.760261
## V16          0.25404    0.07617   3.335 0.000865 ***
## V17          0.36061    0.09132   3.949 8.08e-05 ***
## V18         -0.07486    0.11716  -0.639 0.522920
## V19         -0.01576    0.08723  -0.181 0.856663
## V20          0.02085    0.08773   0.238 0.812165
## V21         -0.53379    0.11154  -4.786 1.81e-06 ***
## V22          0.20140    0.07393   2.724 0.006494 **
## V23          0.09432    0.07526   1.253 0.210217
## V24         -0.27993    0.09762  -2.868 0.004172 **
## V25          0.13731    0.07269   1.889 0.059023 .
## V26         -0.30047    0.08824  -3.405 0.000672 ***
## V27         -0.48165    0.09408  -5.120 3.30e-07 ***
## V28          0.11694    0.06694   1.747 0.080766 .
## V29         -0.17024    0.07491  -2.273 0.023138 *
## V30          0.25429    0.08697   2.924 0.003491 **
## V31          0.24358    0.10357   2.352 0.018764 *
## V32         -0.02623    0.08420  -0.311 0.755448
## V33         -0.26944    0.09937  -2.712 0.006744 **
## V34         -0.51540    0.09911  -5.200 2.16e-07 ***
## V35          0.04367    0.09042   0.483 0.629158
## V36          0.01722    0.09441   0.182 0.855317
## V37          0.56181    0.10171   5.524 3.67e-08 ***
## V38         -0.26530    0.10084  -2.631 0.008568 **
## V39          0.04320    0.11957   0.361 0.717931
## V40         -0.03396    0.06388  -0.532 0.595054
## V41         -0.33003    0.10972  -3.008 0.002657 **
## V42         -0.20783    0.08796  -2.363 0.018214 *
## V43         -0.46604    0.08260  -5.642 1.88e-08 ***
## V44         -0.14936    0.08327  -1.794 0.073004 .
## V45          0.12906    0.08886   1.452 0.146516
## V46          0.08303    0.07980   1.041 0.298209
## V47          0.54743    0.09363   5.847 5.69e-09 ***
## V48          0.16717    0.09017   1.854 0.063878 .
## V49         -0.36211    0.08530  -4.245 2.27e-05 ***
## V50         -0.15050    0.09998  -1.505 0.132406
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4808 on 2418 degrees of freedom
```

```
## Multiple R-squared:  0.3332, Adjusted R-squared:  0.3195
## F-statistic: 24.17 on 50 and 2418 DF,  p-value: < 2.2e-16
```

As seen, the model can explain approximately 32% the changes in polarity scores. Word vectors 3, 12, 16, 17, 37 and 47 have statistically significant positive associations with polarity scores. Among which, the biggest coefficient belongs to vector 12 (0.57). Other vectors having marginally significant positive correlations with polarity scores are 6, 7, 10, 22, 30, 31. Meanwhile, word vectors 8, 21, 26, 27, 34, 43, and 49 have statistically significant negative associations with on polarity scores. Among which, the biggest coefficient belongs to vector 21 (-0.53379). Other vectors having marginally significant negative correlations with polarity scores are 4, 24, 29, 33, 38, 41, and 42.

THE END.