Learning from Big Data: Assignment 1

Shinyoung Kim 632604

16/10/2024

1. Introduction

This analysis is performed using a dataset containing movie reviews and their associated star rating and helpfulness rating. I utilize NLP techniques, including Word2Vec and regression analysis, to explore the relationship between the helpfulness and extremeness of a review.

Research Question: Do people find reviews that use extreme words more helpful than those using neutral language?

H1: Reviews containing extreme words (either highly positive or negative) are perceived as more helpful because they evoke stronger reactions. In contrast, moderate or neutral language may have a lesser impact on perceived helpfulness.

H2: Extreme negative words have a more significant effect on review helpfulness compared to extreme positive words. Specifically, the helpfulness of a review by word category follows this order: EXT negative > EXT positive > Neutral.

2. Load libraries

```
# Required packages. P_load ensures these will be installed and loaded.
rm(list=ls())
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tm, openNLP, nnet, dplyr, tidyr, ggplot2, reshape2,latex2exp,
magrittr, slam, stringr, topicmodels, word2vec, tokenizers, LDAvis, car,
caret, emmeans)
```

3. Load review data

```
Reviews_Raw <- read.csv("Reviews_tiny.csv", fileEncoding="ISO-8859-1")
Reviews_Raw <- Reviews_Raw %>%
               select(movie_name, review_code,  reviewer,   review_date,
                      num_eval, words_in_lexicon_sentiment_and_review,
                      ratio_helpful, raters, full_text, release_date,
first_week_box_office) %>%
              rename(words_in_lexicon_helpfulness_and_review =
words_in_lexicon_sentiment_and_review) #sentiment analysis is replaced by
helpfulness
```

```r
#additional columns for the further analysis
Reviews_Raw <- Reviews_Raw %>%
  mutate(prob_helpful = NA, prob_EXTP = NA,prob_NEUT = NA,prob_EXTN = NA)
```

4. Data aggregation and formatting

I opt for a review-level approach because it aligns with my research question, which aims to explore the relationship between the specific language used in reviews and the helpfulness of those reviews. A random sampling of 30% of the review data is performed to train the models that analyze the correlation between extreme word usage and review helpfulness.

```r
# Select 30% of data for training
train_data <- Reviews_Raw %>% sample_frac(0.3)

# The remaining 70% is used for testing
test_data <- Reviews_Raw[!rownames(Reviews_Raw) %in% rownames(train_data), ]
```

5. Supervised learning - The Naive Bayes classifier

Naive Bayes Classifier is implemented to estimate two key aspects for each review:

1. Helpfulness Prediction: The likelihood that a review is perceived as helpful or unhelpful by other users.
2. Sentiment Extremeness: The probability that the review contains extremely positive (EXTP), neutral (NEUT), or extremely negative (EXTN) language.

5.0 Load support functions and global parameters

```r
# FUNCTIONS
Compute_posterior_helpfulness = function(prior, corpus_in, dict_words,
p_w_given_c,TOT_DIMENSIONS){
  output <-  capture.output
     (word_matrix <-inspect(
       DocumentTermMatrix(corpus_in, control=list(stemming=FALSE,
                                                  language = "english",

dictionary=as.character(dict_words)))))

  # If no relevant words are found, return the prior
  if (sum(word_matrix) == 0) {posterior<-prior ; words_ <- c("")} else
  {
     # Get words found in the review and their occurrences
    word_matrix_indices <- which(word_matrix>0)
    textual_words_vec   <- colnames(word_matrix)[word_matrix_indices]
```

```
    # Loop around each word found in the review
    WR <- length(word_matrix_indices) ;word_matrix_indices_index=1
    for (word_matrix_indices_index in 1: WR)
    {
      word <-
colnames(word_matrix)[word_matrix_indices[word_matrix_indices_index]]
      p_w_given_c_index <- which(as.character(p_w_given_c$word) == word)

      # Loop around occurrences of each word
      occurrences_current_word=1
      for (occurrences_current_word in 1:
           word_matrix[1,word_matrix_indices[word_matrix_indices_index]] )
      {
        # initialize variables
        posterior     <- c(rep(0, TOT_DIMENSIONS))

        # Compute likelihood for 'helpful' and 'unhelpful'
        vec_likelihood<- as.numeric(c(p_w_given_c$helpful[p_w_given_c_index],

p_w_given_c$unhelpful[p_w_given_c_index]))

        # Update posterior for 'helpful'
        numerat         <-  prior[1] *

as.numeric(p_w_given_c$helpful[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[1]    <- numerat /  denomin

        # Update posterior for 'unhelpful'
        numerat         <-  prior[2] *
as.numeric(p_w_given_c$unhelpful[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[2]    <- numerat /  denomin


        if (sum(posterior)>1.01) { ERROR <- TRUE }
        prior <- posterior  # Update prior for the next word
      }
    }
    words_ <-  colnames(word_matrix)[word_matrix_indices]
  }

  return(list(posterior_=posterior, words_=words_) )
}

Compute_posterior_extremeness <- function(prior,  word_matrix, p_w_given_c ,
BIGRAM, TOT_DIMENSIONS){
```

```
  # If no relevant words are found, return the prior
  if (sum(word_matrix) == 0) {posterior<-prior} else
    {
    # Get words found in the review and their occurrences
    word_matrix_indices <- which(word_matrix>0)
    textual_words_vec   <- colnames(word_matrix)[word_matrix_indices]

    # Loop around each word found in review
    WR <- length(word_matrix_indices) ;word_matrix_indices_index=1

    for (word_matrix_indices_index in 1: WR) {
      word <-
colnames(word_matrix)[word_matrix_indices[word_matrix_indices_index]]
      p_w_given_c_index <- which(as.character(p_w_given_c$word) == word)

      # Loop around occurrences of each word
      occ_current_word <- 1
      for (occ_current_word in
1:word_matrix[1,word_matrix_indices[word_matrix_indices_index]])
      {
        # initialize variables
        posterior     <- c(rep(0, TOT_DIMENSIONS))

        # Compute likelihood for EXT positive, NEUT, and EXT negative
        vec_likelihood <-as.numeric(c(p_w_given_c$EXTP[p_w_given_c_index],
                                      p_w_given_c$NEUT[p_w_given_c_index],
                                      p_w_given_c$EXTN[p_w_given_c_index]) )

        # Update posterior for EXTP
        numerat         <-  prior[1] *
as.numeric(p_w_given_c$EXTP[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[1]    <-  numerat /  denomin

        # Update posterior for NEUT
        numerat         <-  prior[2] *
as.numeric(p_w_given_c$NEUT[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[2]    <-  numerat /  denomin

        # Update posterior for EXTN
        numerat         <-  prior[3] *
as.numeric(p_w_given_c$EXTN[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[3]    <-  numerat /  denomin

        if (sum(posterior)>1.01) {
```

```
          ERROR <- TRUE }
        prior <- posterior } } }

  return (posterior_= posterior)}

# GLOBAL PARAMETERS
PRIOR_HELP   = 1/2
PRIOR_EXTREME = 1/3
TOT_REVIEWS = length(Reviews_Raw[,1])
```

5.1 Creating lexicons

#helpfulness lexicon To create the helpfulness likelihoods, I created the lexicon for helpful/unhelpful reviews. Reviews with helpful ratio higher than 0.5 is classified as WL_helpful and the rest is WL_unhelpful. This classification is made based on the assumption that a helpfulness ratio of over 0.5 (i.e., more than half of the evaluators found the review helpful) is a reasonable threshold for defining a review as "helpful."

```
# Clean reviews and extract lexicons
clean_corpus <- function(corpus) {
  corpus %>%
    tm_map(content_transformer(function(x) iconv(x, from = "latin1", to = 
"UTF-8", sub = ""))) %>%  # Ensure UTF-8 encoding
    tm_map(content_transformer(tolower)) %>%
    tm_map(removePunctuation) %>%
    tm_map(removeNumbers)}

# Function to extract, clean, and create a word list from reviews, excluding 
gibberish
create_word_list <- function(reviews, condition = TRUE) {
  reviews %>% filter(condition) %>% pull(full_text) %>%
    paste(collapse = " ") %>% iconv(from = "latin1", to = "UTF-8", sub = "") 
%>%
    VectorSource() %>% VCorpus() %>% clean_corpus() %>% DocumentTermMatrix() 
%>%
    as.matrix() %>% colnames() %>%.[grepl("^[a-zA-Z]+$", .)]}

# Create word lists for helpful and unhelpful reviews
WL_helpful <- create_word_list(train_data, train_data$ratio_helpful > 0.5)
WL_unhelpful <- create_word_list(train_data, train_data$ratio_helpful <= 0.5)
```

#extremeness lexicon To create the extremeness likelihoods, I created the lexicon for extremely positive(EXTP), neutral(NEUT) and extremely negative(EXTN) reviews, based on the the rating out of 10. The decision to classify reviews based on a 10-point rating scale is made because extreme ratings (near the endpoints of the scale) are likely to reflect stronger opinions and use of extreme language. Reviews with moderate ratings (between 3 and 8) are expected to contain more neutral or balanced language, fitting the research's focus on extremeness.

```
# word list of EXTP, NEUT, EXTN
WL_EXTP <- create_word_list(train_data, train_data$num_eval >= 9)
WL_EXTN <- create_word_list(train_data, train_data$num_eval <= 2 )
WL_NEUT <- create_word_list(train_data, train_data$num_eval >2 &
train_data$num_eval<9)

#word list of training text
WL_reviews <- create_word_list(train_data, !is.na(train_data$full_text))
```

5.2 Likelihoods_helpfulness

When computing likelihoods, the log likelihood method is used as a smoothing method. Logarithmic transformations help avoid issues with underflow when dealing with very small probabilities, which is common in text analysis. Also, when I compared it with the laplace smoothing, log likelihood gave me a higher accuracy.

```
compute_likelihoods_helpfulness<- function(reviews, WL_1, WL_2, smoothing =
1) {
  # Create a Document-Term Matrix for the reviews
  corpus <- VCorpus(VectorSource(reviews)) %>%
    clean_corpus()
  dtm <- DocumentTermMatrix(corpus)

  # Get the total word counts for each review
  total_word_counts <- slam::row_sums(dtm)  # Total word count per review
  vocab_size <- length(colnames(dtm))  # Vocabulary size for smoothing

  # Initialize a dataframe to store log likelihoods
  likelihoods <- data.frame(word = character(),
                            helpful = numeric(),
                            unhelpful = numeric(),
                            stringsAsFactors = FALSE)

  # Loop through each word in the DTM to calculate log likelihoods for WL_1
and WL_2
  for (word in colnames(dtm)) {
    # Get the word index in the Document-Term Matrix
    word_index <- which(colnames(dtm) == word)

    # Count the occurrences of the word across all reviews
    word_counts <- slam::col_sums(dtm[, word_index])

    # Calculate the log likelihood of the word being in WL_1 (helpful)
    if (word %in% WL_1) {
```

```
      help_likelihood <- (sum(word_counts) + smoothing) /
                         (sum(total_word_counts) + vocab_size * smoothing)
      help_likelihood <- log(help_likelihood)
    } else {
      help_likelihood <- log(smoothing / (sum(total_word_counts) + vocab_size
* smoothing))}

    # Calculate the log likelihood of the word being in WL_2 (unhelpful)
    if (word %in% WL_2) {
      unhelp_likelihood <- (sum(word_counts) + smoothing) /
                           (sum(total_word_counts) + vocab_size * smoothing)
      unhelp_likelihood <- log(unhelp_likelihood)
    } else {
      unhelp_likelihood <- log(smoothing / (sum(total_word_counts) +
vocab_size * smoothing))}

    # Append to the likelihoods dataframe
    likelihoods <- rbind(likelihoods,
                         data.frame(word = word,
                                    helpful = help_likelihood,
                                    unhelpful = unhelp_likelihood))}
  return(likelihoods)}

# Compute the log likelihoods for the helpful lexicon using both WL_helpful
and WL_unhelpful
likelihoods_helpfulness<-
compute_likelihoods_helpfulness(train_data$full_text, WL_helpful,
WL_unhelpful, 1)
lexicon_helpfulness <- as.character(likelihoods_helpfulness$word)
```

5.3 Likelihoods_extremeness

```
# Create a function to compute log likelihoods for three word lists (extreme
positive, neutral, extreme negative)
compute_likelihoods_extremeness <- function(reviews, WL_EXTP, WL_NEUT,
WL_EXTN, smoothing = 1) {

  # Create a Document-Term Matrix for the reviews
  corpus <- VCorpus(VectorSource(reviews)) %>%
    clean_corpus()  # Ensure you have the 'clean_corpus()' function defined
  dtm <- DocumentTermMatrix(corpus)

  # Get the total word counts for each review
  total_word_counts <- slam::row_sums(dtm)  # Total word count per review
  vocab_size <- length(colnames(dtm))  # Vocabulary size

  # Initialize a dataframe to store likelihoods
  likelihoods <- data.frame(word = character(),
```

```r
                              EXTP = numeric(),  # Extreme Positive
                              NEUT = numeric(),  # Neutral
                              EXTN = numeric(),  # Extreme Negative
                              stringsAsFactors = FALSE)

  # Loop through each word in the DTM to calculate log likelihoods for
WL_EXTP, WL_NEUT, and WL_EXTN
  for (word in colnames(dtm)) {
    # Get the word index in the Document-Term Matrix
    word_index <- which(colnames(dtm) == word)

    # Count the occurrences of the word across all reviews
    word_counts <- slam::col_sums(dtm[, word_index])

    # Calculate the likelihood of the word being in WL_EXTP
    if (word %in% WL_EXTP) {
      extp_likelihood <- (sum(word_counts) + smoothing) /
                         (sum(total_word_counts) + vocab_size * smoothing)
      extp_likelihood <- log(extp_likelihood)
    } else {
      extp_likelihood <- log(smoothing / (sum(total_word_counts) + vocab_size
* smoothing))}

    # Calculate the log likelihood of the word being in WL_NEUT
    if (word %in% WL_NEUT) {
      neut_likelihood <- (sum(word_counts) + smoothing) /
                         (sum(total_word_counts) + vocab_size * smoothing)
      neut_likelihood <- log(neut_likelihood)
    } else {
      neut_likelihood <- log(smoothing / (sum(total_word_counts) + vocab_size
* smoothing))}

    # Calculate the log likelihood of the word being in WL_EXTN
    if (word %in% WL_EXTN) {
      extn_likelihood <- (sum(word_counts) + smoothing) /
                         (sum(total_word_counts) + vocab_size * smoothing)
      extn_likelihood <- log(extn_likelihood)
    } else {
      extn_likelihood <- log(smoothing / (sum(total_word_counts) + vocab_size
* smoothing))}

    # Append to the likelihoods dataframe
    likelihoods <- rbind(likelihoods,
                         data.frame(word = word,
                                    EXTP = extp_likelihood,
                                    NEUT = neut_likelihood,
                                    EXTN = extn_likelihood))}
```

```
  return(likelihoods)}

# Compute the likelihoods for the extremeness lexicon using WL_EXTP (Extreme
Positive), WL_NEUT (Neutral), WL_EXTN (Extreme Negative)
likelihoods_extremeness <-
compute_likelihoods_extremeness(train_data$full_text, WL_EXTP, WL_NEUT,
WL_EXTN, smoothing = 1)

# Extract the words
lexicon_extremeness <- as.character(likelihoods_extremeness$word)

# Combine likelihoods from helpfulness and extremeness
likelihoods <- data.frame(likelihoods_helpfulness, likelihoods_extremeness
%>%
  select(EXTP, NEUT, EXTN))

# Save the likelihoods to a CSV file
write.csv(likelihoods, file = "Reviews_likelihoods.csv", row.names = FALSE)
```

6.1 Run NBC for helpfulness

```
for (review_index in 1:TOT_REVIEWS) {
  prior_help     <- as.numeric(c(PRIOR_HELP,1-PRIOR_HELP))   # Reset the
prior as each review is looked at separately
  text_review    <- as.character(Reviews_Raw$full_text[review_index])
  text_review <- iconv(text_review, from = "latin1", to = "UTF-8", sub = "")

  # Pre-process the review: remove punctuation and numbers
  corpus_review <- VCorpus(VectorSource(text_review)) %>%
    tm_map(removePunctuation) %>%
    tm_map(removeNumbers)

  # Compute posterior probability for helpfulness
  TOT_DIMENSIONS = 2
  output <-capture.output(help.results <- Compute_posterior_helpfulness(prior
= prior_help,
                                              corpus_in  = corpus_review,
                                              dict_words =
lexicon_helpfulness,
                                              p_w_given_c=
likelihoods_helpfulness,
                                              TOT_DIMENSIONS) )
  # Get words and posteriors
  words_help  <- help.results$words_
  posterior_help <- help.results$posterior_

  # Store the computed posteriors and words
  Reviews_Raw$prob_helpful[review_index] <- posterior_help[1]
```

```
  Reviews_Raw$words_in_lexicon_helpfulness_and_review[review_index] <-
paste(words_help,collapse =" ") }
```

6.2 Run NBC for extremeness

```
for (review_index in 1: TOT_REVIEWS) {
  # If the review is not empty, continue and calculate posterior
  if (Reviews_Raw$full_text[review_index]!=""){

    # Process the text of the non-empty review
    text_review    <- as.character(Reviews_Raw$full_text[review_index])
    text_review <- iconv(text_review, from = "latin1", to = "UTF-8", sub =
"")
    # Pre-process the review to remove numbers and punctuation
    corpus_review <- VCorpus(VectorSource(text_review))
    output <-capture.output(extreme_word_matrix <-
                              inspect(DocumentTermMatrix(corpus_review,
                                         control = list(stemming=FALSE,
                                                        language = "english",

removePunctuation=TRUE,
                                                        removeNumbers=TRUE,

dictionary=as.character(lexicon_extremeness)))))

    # Compute posterior probabilities for extremeness
    TOT_DIMENSIONS <- 3
    posterior <- Compute_posterior_extremeness(prior= matrix(PRIOR_EXTREME,
ncol=TOT_DIMENSIONS),
                                           extreme_word_matrix,

p_w_given_c=likelihoods_extremeness,
                                           TOT_DIMENSIONS)

    # Store the posteriors
    Reviews_Raw$prob_EXTP[review_index]    <- posterior[1]
    Reviews_Raw$prob_NEUT[review_index]       <- posterior[2]
    Reviews_Raw$prob_EXTN[review_index] <- posterior[3]
  }

}
# Saves the updated file
write.csv(Reviews_Raw,file="Reviews_posteriors.csv" , row.names = FALSE )
```

7. Supervised Learning - Inspect the NBC performance

7.0 Load judges scores

The lexicon of words with extremeness scores was considered as a ground truth to evaluate my Naive Bayes Classifier (NBC) implementation. The scores are from the AFINN-111 sentiment lexicon, which rates words on a scale from -5 to 5 based on sentiment strength. The words are classified by their scores within the range of -5 to 5. I define three lexicons;

EXTP: score >=3, NEUT: -3< score <3, EXTN: score <= -3

```
# Load word scores
word_score <- read.csv2('AFINN-111.csv')

# Define lexicons for extreme positive, neutral, and extreme negative
extreme_pos <- word_score %>% filter(score >= 3 & score <= 5)
extreme_neg <- word_score %>% filter(score >= -5 & score <= -3)
neutral <- word_score %>% filter(score >= -2 & score <= 2)
```

7.1 Compute confusion matrix, precision and recall

To evaluate the NBC model, I compare the predicted labels against the actual labels from the ground truth.

```
# Function to count occurrences of words from a lexicon in a review
count_occurrences <- function(review_text, lexicon) {
  sum(sapply(lexicon$word, function(word) str_count(review_text,
fixed(word))))}

# Count occurrences of extreme_pos, extreme_neg, and neutral words in each
review
Reviews_Raw$extreme_pos_count <- sapply(Reviews_Raw$full_text,
count_occurrences, lexicon = extreme_pos)
Reviews_Raw$extreme_neg_count <- sapply(Reviews_Raw$full_text,
count_occurrences, lexicon = extreme_neg)
Reviews_Raw$neutral_count <- sapply(Reviews_Raw$full_text, count_occurrences,
lexicon = neutral)

# Label the reviews based on extreme_pos and extreme_neg counts
Reviews_Raw$actual_label <- ifelse(Reviews_Raw$extreme_pos_count == 0 &
Reviews_Raw$extreme_neg_count == 0, 0, 1)

# Create a predicted label based on the highest posterior probability
(extreme positive, negative, neutral)
Reviews_Raw$predicted_label <- apply(Reviews_Raw[, c("prob_EXTP",
```

```
"prob_EXTN", "prob_NEUT")], 1, function(x) {
  if (which.max(x) == 3) { # If prob_NEUT is the highest (3rd column)
    return(0)
  } else {
    return(1)  # If extreme positive or extreme negative is the highest, mark
as 1 (extreme)
}})

# Create confusion matrix
confusion_matrix <- table(Predicted = Reviews_Raw$predicted_label, Actual =
Reviews_Raw$actual_label)

# Print confusion matrix
print(confusion_matrix)

# Check for empty cells to avoid errors
TP <- confusion_matrix[2, 2]  # True Positive
TN <- confusion_matrix[1, 1]  # True Negative
FP <- confusion_matrix[2, 1]  # False Positive
FN <- confusion_matrix[1, 2]  # False Negative

# Compute performance metrics
precision <- ifelse((TP + FP) == 0, 0, TP / (TP + FP))  # Avoid division by
zero
accuracy <- (TP + TN) / sum(confusion_matrix)

# Print the results
cat("Precision: ", precision, "\n")

## Precision:  0.9969758

cat("Accuracy: ", accuracy, "\n")

## Accuracy:  0.989
```

At the first attempt, the threshold for each class were like the following. EXTP: score >=4, NEUT: -3<= score <=3, EXTN: score <= -4 with the precision of 0.768 and 0.765 accuracy. Then to improve the quality of my model, I expanded the range for EXTP: score >=3, NEUT: -2<= score <=2, EXTN: score <= -3. This way the model captures more words that contribute to stronger sentiment, which likely leads to clearer distinctions between neutral and extreme words.

Previously, with a narrower range for extremeness (EXTP ≥ 4, EXTN≤ -4), certain reviews that could have been classified as extreme may have been labeled neutral due to the stricter thresholds.

8. Unsupervised Learning: Predict box office using LDA

8.0 Load data

Since the previous section did not compute the posterior of topics, I used the given fake likelihood for LDA analysis.

```
# Determining the total number of reviews in our dataset
total_reviews <- nrow(Reviews_Raw)

# Loading fake likelihoods data
likelihoods <- read.csv("example_100_fake_likelihood_content.csv")

# set out lexicon equal to the first column of the likelihoods data and 
inspecting its structure
lexicon_content  <- as.character(likelihoods[ ,1] )
```

8.1 Structure data

```
# Put processed reviews in corpus format
corpus_review <- VCorpus(VectorSource(Reviews_Raw$full_text))

# Creates the document term matrix that will be passed to the LDA function
dtm <- DocumentTermMatrix(corpus_review,
                       control = list(stemming = FALSE,  # Which is also the 
default
                                      language = "english",
                                      removePunctuation = TRUE,
                                      removeNumbers = TRUE,
                                      dictionary = 
as.character(lexicon_content)) )

# LDA parameters
seed <- 20080809
burnin <-  1000
iter <- 1000
```

8.2 Finding the ideal k(number of topics)

To find the ideal number of topics, I evaluated average log likelihood using cross validation(5-folds) and chose the k that has the highest log likelihood. The range of k values are arbitrary chosen from 2 to 14.

```
# Define range of k values to test
k_values <- seq(2, 14, by = 2)

# defind number of folds for cross-validation
n_folds <- 5
```

```r
# Empty list to store average log-likelihoods
avg_log_likelihoods <- numeric(length(k_values))

# Create folds
set.seed(123)
folds <- createFolds(1:nrow(dtm), k = n_folds)

# Loop over each k
for (i in seq_along(k_values)) {

  # Set the current k
  k <- k_values[i]

  # Store log-likelihoods for each fold
  fold_log_likelihoods <- numeric(n_folds)

  # Loop over each fold
  for (fold_idx in seq_along(folds)) {

    # Get train and test indices
    test_idx <- folds[[fold_idx]]
    train_idx <- setdiff(1:nrow(dtm), test_idx)
    # Create train and test sets
    dtm_train <- dtm[train_idx, ]
    dtm_test <- dtm[test_idx, ]

    # Train LDA model on training set
    model_lda <- LDA(dtm_train, k = k, method = "Gibbs", control = list(seed
= seed, burnin = burnin, iter = iter))
    # Compute log-likelihood on the test set
    fold_log_likelihoods[fold_idx] <- logLik(model_lda, newdata = dtm_test)}

  # Compute the average log-likelihood across folds for the current k
  avg_log_likelihoods[i] <- mean(fold_log_likelihoods)}

results <- data.frame(k = k_values, avg_log_likelihood = avg_log_likelihoods)
print(results)
```

8.3 Run the LDA

```r
k <- 14 # generated highest avg log_likelihood in the previous section
#Create an LDA model using GIBBS sampling
model_lda <- LDA(dtm, k, method = "Gibbs", control = list(seed = seed, burnin
= burnin, iter = iter) , mc.cores = 4)
save(model_lda , file = paste("LDA_model_" ,k,".RData" ,sep=""))
```

```r
#posterior probabilities per document by topic
posteriors_lda <- posterior(model_lda)$topics
```

8.4 Box office prediction-LDA

Finally, I used the topic distributions (posterior probabilities) from LDA as features to predict the log-transformed box office using linear regression. The regression model uses the LDA topics as predictors, and the performance of the model gives insights into which latent topics are most influential in predicting the box office.

```r
# Prepare data for box office prediction
log_BO <- log(as.numeric(gsub(",", "", Reviews_Raw$first_week_box_office)))

# Combine log_BO with posterior probabilities
data_reg <- cbind(log_BO, posteriors_lda)
colnames(data_reg) <- c("LogBoxOffice", paste("Topic", 1:k, sep = "_"))

# Fit the linear regression model
box_office_model <- lm(LogBoxOffice ~ ., data = as.data.frame(data_reg))
summary(box_office_model)

predictions_LDA <- predict(box_office_model, newdata =
as.data.frame(data_reg))

# Exponentiate to get back to the original scale of the box office
predicted_box_office_LDA <- exp(predictions_LDA)
```

The regression analysis identifies several topics that significantly impact box office performance. **Topic 9** stands out with the most substantial positive effect (estimate: 29.02), indicating that movies associated with this theme tend to perform exceptionally well financially. Similarly, **Topic 10** (estimate: 13.52) and **Topic 13** (estimate: 11.39) also contribute positively, suggesting that these topics are linked to higher box office revenue. On the other hand, **Topic 5** and **Topic 2** shows a negative influence suggesting that the content related to these topics may not appeal as strongly to viewers.

The model accounts for approximately 27.4% of the variance in box office performance, reflecting a moderate explanatory power. This indicates that while the distribution of topics is a useful predictor, other factors are likely to have a significant influence on a film's box office.

9. Unsupervised Learning: Predict box office using Word embeddings given by Word2Vec

9.1 Training Step

```r
# Obtain the column with the reviews and convert it to lower case
x <-  Reviews_Raw$full_text
x <- tolower(x)
```

```
# number of topics in Word2Vec
total_topics_word2vec <- 14

model <- word2vec(x = x, type = "cbow", dim = total_topics_word2vec, iter =
20)
embedding <- as.matrix(model)
```

9.2 Construct variables from word embeddings

```
# Create an empty matrix to store the posteriors (mean of word vectors)
posteriors_w2v <- matrix(0, nrow = total_reviews, ncol =
total_topics_word2vec)

# Loop over all reviews in the test set
for (k in 1:total_reviews) {
  # Tokenize the review (split into words)
  tokenized_review <- unlist(strsplit(Reviews_Raw$full_text[[k]],"[^a-zA-Z0-
9]+"))  # Tokenize by non-word characters

  # Get the word vectors per review using the trained model
  embedding_review <- predict(model, tokenized_review, type = "embedding")

  # compute mean across all words in the review
   posteriors_w2v[k,] <- apply(embedding_review, 2, mean, na.rm=TRUE)}
```

9.3 Box office prediction-w2v

```
# Order the reviews by their review date
Reviews_Raw <- Reviews_Raw[order(Reviews_Raw$review_date), ]

# Prepare the box office data: log-transform the actual box office values
log_BO <- log(as.numeric(gsub(",", "", Reviews_Raw$first_week_box_office)))

# Combine the log-transformed box office data with the Word2Vec posteriors
data_reg <- cbind(log_BO, posteriors_w2v)
colnames(data_reg) <- c("LogBoxOffice", paste("w2v_",
as.character(1:total_topics_word2vec), sep=""))

# Fit a linear regression model to predict box office from Word2Vec features
w2v_BO_lm <- lm(LogBoxOffice ~ ., data = as.data.frame(data_reg))
  # Use all w2v variables in the regression
summary(w2v_BO_lm)

# Evaluate the model performance
# Make predictions on the test set
predictions_w2v <- predict(w2v_BO_lm, newdata = as.data.frame(data_reg))
# Exponentiate predictions to revert to the original scale of the box office
```

```r
predicted_box_office_w2v <- exp(predictions_w2v)
# Compare predictions with actual box office values
actual_box_office <- exp(log_BO)

# Calculate performance metrics: R-squared, MSE, and MAE
rsq <- summary(w2v_BO_lm)$r.squared
mse <- mean((predicted_box_office_w2v - actual_box_office)^2)
mae <- mean(abs(predicted_box_office_w2v - actual_box_office))

cat("R-squared:", rsq, "\n")

## R-squared: 0.02664392

cat("Mean Squared Error:", mse, "\n")

## Mean Squared Error: 1.408928e+15

cat("Mean Absolute Error:", mae, "\n")

## Mean Absolute Error: 18746483

# Create a new data frame to store both predictions (if you also have LDA
predictions)
box_office_prediction <- Reviews_Raw %>%
  select(movie_name, review_code, first_week_box_office) %>%
  mutate(predicted_box_office_w2v = c(predicted_box_office_w2v, rep(NA,
nrow(Reviews_Raw) - length(predicted_box_office_w2v))))

write.csv(box_office_prediction, file = "box_office_prediction_combined.csv",
row.names = FALSE)
```

*Since the Word2Vec model generates different results everytime I ran the code, I have added a screenshot of the regression in the appendix and you could find the analysis in the PDF file.

w2v_8 and w2v_11 have comparably higher coefficient suggesting that changes in the respective word embedding have a larger impact on the predicted value of the box office. For example, **w2v_8 (1.62)** suggests that for every unit increase in the value of this word embedding feature, the predicted log-transformed box office increases by **1.62** units, holding all other variables constant. However, the current data/model does not provide strong enough evidence to confirm these effects. It might be due to such as insufficient sample size, model misspecification, or high variance.

10. Analysis for my research question - Regression

```r
# Order the reviews data by review date
Reviews_Raw <- Reviews_Raw[order(Reviews_Raw$review_date), ]
helpfulness_values <- Reviews_Raw$ratio_helpful  # or log(transform if
```

```r
needed)

# Create dummy variables for extremeness based on the probability thresholds
you defined earlier
Reviews_Raw <- Reviews_Raw %>%
  mutate(
    extreme_positive = ifelse(prob_EXTP > 0.5, 1, 0),
    neutral = ifelse(prob_NEUT > 0.5, 1, 0),
    extreme_negative = ifelse(prob_EXTN > 0.5, 1, 0))

# Combine helpfulness values with extremeness variables
data_reg <- data.frame(helpfulness = helpfulness_values,
                       extreme_positive = Reviews_Raw$extreme_positive,
                       neutral = Reviews_Raw$neutral,
                       extreme_negative = Reviews_Raw$extreme_negative)

# Fit a linear regression model to predict helpfulness from extremeness
variables
helpfulness_lm <- lm(helpfulness ~ extreme_positive + neutral +
extreme_negative, data = data_reg)
summary(helpfulness_lm)
```

H1: partially confirmed. The results support the first part of H1 regarding **extreme positive** language, which significantly increases helpfulness (coefficient of 0.34792, $p < 0.001$). This suggests that reviews with extreme positive language are indeed perceived as more helpful. However, the effect of **neutral** language was not statistically significant ($p = 0.405$), indicating that it does not significantly contribute to perceived helpfulness. Therefore, H1 is partially confirmed.

H2: rejected. The coefficient for **extreme negative** words is -0.20447, which is significant ($p < 0.001$), indicating that extreme negative language decreases helpfulness. While extreme positive words also significantly increase helpfulness, the effect of extreme negative words (negative coefficient) indicates that they have a stronger adverse impact. The ranking suggested in H2 (EXT negative > EXT positive > Neutral) is somewhat supported by the significant negative impact of extreme negative words compared to the positive impact of extreme positive words. However, the neutral language did not significantly contribute to the model.

While extreme negative words negatively affect helpfulness, the specific ranking (EXT negative > EXT positive) isn't conclusively supported by the results since the effect of extreme positive language is positive and significant.

#Appendix

1.Confusion matrix

```
##            Actual
## Predicted   0   1
##         0   0   8
##         1   3 989
```

2.K average log likelihood

```
##    k avg_log_likelihood
## 1  2          -68275.22
## 2  4          -57501.04
## 3  6          -51828.01
## 4  8          -47753.55
## 5 10          -45372.17
## 6 12          -43634.75
## 7 14          -42136.09
```

3.LDA prediction regression result

```
## 
## Call:
## lm(formula = LogBoxOffice ~ ., data = as.data.frame(data_reg))
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -4.2198 -1.1958 -0.4347  0.7550  4.7338 
## 
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 12.51291    2.36420   5.293 1.49e-07 ***
## Topic_1      2.23629    3.57195   0.626   0.5314    
## Topic_2     -8.35185    3.63305  -2.299   0.0217 *  
## Topic_3      0.09968    3.48319   0.029   0.9772    
## Topic_4      2.84133    3.59634   0.790   0.4297    
## Topic_5    -18.85649    3.21402  -5.867 6.06e-09 ***
## Topic_6     -6.34464    3.47593  -1.825   0.0683 .  
## Topic_7      1.99487    3.82185   0.522   0.6018    
## Topic_8      3.73776    3.56666   1.048   0.2949    
## Topic_9     29.02075    3.18752   9.105  < 2e-16 ***
## Topic_10    13.51628    3.45747   3.909 9.89e-05 ***
## Topic_11    -3.62842    3.10351  -1.169   0.2426    
## Topic_12     0.96325    3.56793   0.270   0.7872    
## Topic_13    11.38790    3.57624   3.184   0.0015 ** 
## Topic_14          NA         NA      NA       NA    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 1.757 on 986 degrees of freedom
## Multiple R-squared:  0.2737, Adjusted R-squared:  0.2642 
## F-statistic: 28.59 on 13 and 986 DF,  p-value: < 2.2e-16
```

4.Word2Vec prediction regression result

```
##
## Call:
## lm(formula = LogBoxOffice ~ ., data = as.data.frame(data_reg))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.5052 -1.5365 -0.5791 -0.0335  4.2145
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  15.6855     2.1823   7.187  1.3e-12 ***
## w2v_1         1.0988     1.8253   0.602    0.547
## w2v_2         0.2549     0.8467   0.301    0.763
## w2v_3        -1.0787     1.2108  -0.891    0.373
## w2v_4         0.9587     0.9800   0.978    0.328
## w2v_5         0.3316     0.9603   0.345    0.730
## w2v_6        -0.5271     1.0404  -0.507    0.613
## w2v_7         0.1558     2.0898   0.075    0.941
## w2v_8         1.6230     1.0220   1.588    0.113
## w2v_9        -0.8540     0.7729  -1.105    0.269
## w2v_10       -0.4314     1.0290  -0.419    0.675
## w2v_11        1.1548     1.7819   0.648    0.517
## w2v_12       -0.6643     0.7110  -0.934    0.350
## w2v_13       -0.3476     1.6143  -0.215    0.830
## w2v_14        0.6493     0.7092   0.916    0.360
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.04 on 985 degrees of freedom
## Multiple R-squared:  0.02219,    Adjusted R-squared:  0.008295
## F-statistic: 1.597 on 14 and 985 DF,  p-value: 0.07381
```

5.Regression model result

```
##
## Call:
## lm(formula = helpfulness ~ extreme_positive + neutral + extreme_negative,
##     data = data_reg)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.43201 -0.43201  0.06799  0.23466  0.64946
##
## Coefficients: (1 not defined because of singularities)
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.43201    0.01167  37.011   <2e-16 ***
## extreme_positive  0.28190    0.12983   2.171   0.0302 *
## neutral                NA         NA      NA       NA
## extreme_negative -0.08147    0.04568  -1.783   0.0748 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3421 on 923 degrees of freedom
##   (74 observations deleted due to missingness)
## Multiple R-squared:  0.008675,   Adjusted R-squared:  0.006527
## F-statistic: 4.039 on 2 and 923 DF,  p-value: 0.01793
```