Natural Language Processing on California's Disneyland Reviews

Kayla Asmus, Amanda Huang, Sophie Shi, Xiaofeng Cai

Table of contents

134 Final Project: Using Natural Language Processing on California's Disneyland Reviews to Use Predictive Modeling

Introduction

Analyzing customer reviews helps companies like Disneyland improve their services. **Natural Language Processing (NLP)** is a common way and powerful tool to conduct such an analysis. It can extract meaningful patterns from unstructured textual data, such as customer reviews. Our group is interested in using NLP to see how reviewers feel about California's Disneyland.

A data set imported from *Kaggle* (https://www.kaggle.com/code/ahmedterry/disneyland-reviews-nlp-sentiment-analysis). Important to note: we will be working in **English** (Bender rule!).

The data set we will be working with has $42,000$ TripAdvisor reviews of three branches of Disneyland parks: California, Hong Kong, and Paris. For the sake of computational power and personal interest we will only be working with the **California** Disneyland reviews.

Our data set has a total of 6 columns or variables:

- `Review_ID`: unique identifier for each review
- `Rating`: 1-5 star rating of the park by each reviewer
- `Year_Month`: the date of visit of each reviewer
- `Reviewer_Location`: home country of each reviewer
- `Review_Text`: the text review
- `Branch`: which Disneyland park location the reviewer visited

After processing the reviews using NLP, we then use our data to create a few predictive models, including: **Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Logistic Regression**. We are interested in predicting both the *sentiment* of Disneyland reviews, as well as predicting the *reviewers' rating (1-5 stars, from unsatisfactory to satisfactory)* of the park.

Packages, Data Cleaning, EDA

The first and most unforgettable step of any data science project: loading the necessary packages necessary for the project's purpose.

```
library(dplyr)
library(tidytext)
library(tidyverse)
library(textdata)
library(wordcloud)
library(reshape2)
library(janitor)
library(stopwords)
library(stringr)
library(word2vec)
library(tm)
library(ggraph)
```

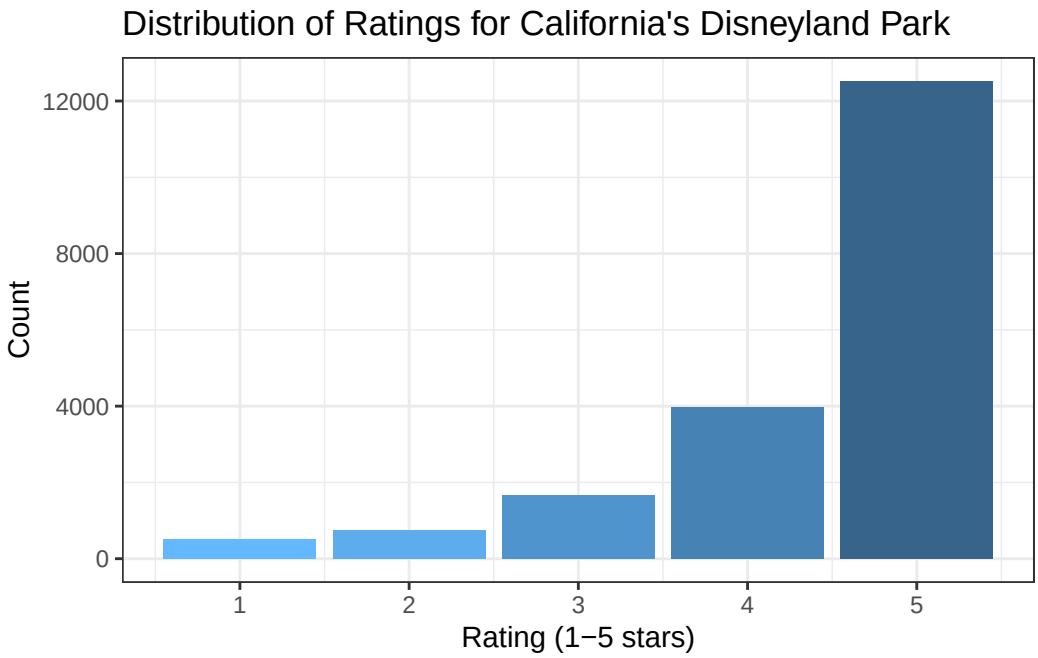
```
library(igraph)
library(tidymodels)
library(kableExtra)
```

Next, we import the data set from the .csv file and perform any necessary (or preferred) data cleaning procedures. Here, we will use `clean_names()`, filter for only California's Disneyland reviews, and perform some simple Exploratory Data Analysis (EDA) on our outcome variable, `rating`.

```
disney_reviews <- read.csv("./data/DisneylandReviews.csv") %>% clean_names()

# filter for only California's Disneyland reviews
disney_cali <- disney_reviews %>%
  filter(branch == "Disneyland_California")

# distribution of ratings
table(disney_cali$rating)
```

```
  1     2     3     4     5
499   747  1661  3981 12518
```

```
disney_cali %>% ggplot(aes(x=rating)) +
  geom_bar(fill = c("steelblue1", "steelblue2", "steelblue3", "steelblue", "steelblue4")) +
  labs(title = "Distribution of Ratings for California's Disneyland Park",
       x = "Rating (1-5 stars)",
       y = "Count") +
  theme_bw()
```



Here we can see that the majority of reviewers of California's Disneyland park had a great experience, rating the park 5 stars. In order to perform **sentiment analysis**, we will be categorizing a new variable, `sentiment`, into two categories based on `rating`:

- *positive*: a rating of 5 stars
- *negative*: a rating of 4 stars or lower

We chose to categorize the new variable this way for a few reasons: (1) this method results in the least amount of class imbalance for `sentiment`, and (2) logically, if a reviewer rates below 5 stars, there existed *some* negative aspect to their trip that caused them to rate the park less than a perfect rating. We create the `sentiment` variable below, and select only the columns we need to perform NLP.

```
disney_cali$id <- seq.int(nrow(disney_cali))
disney_cali <- disney_cali %>%
  mutate(sentiment = case_when(rating == 1 ~ "negative",
                               rating == 2 ~ "negative",
                               rating == 3 ~ "negative",
                               rating == 4 ~ "negative",
                               rating == 5 ~ "positive")) %>%
  select(review_text, rating, sentiment, id)
```

Natural Language Processing (NLP)

Cleaning and Tokenizing

Now, we begin our **Natural Language Processing** portion of our project. First, we will clean up the reviews; remove punctuation, convert all letters to lowercase, remove digits and symbols, and replace any odd or unusual characters.

```
remove <- c('\n',
            '[[:punct:]]',
            '[[:digit:]]',
            '[[:symbol:]]') %>%
  paste(collapse = '|')
disney_cali$review_text <- disney_cali$review_text %>%
  str_remove_all('\'') %>%
  str_replace_all(remove, ' ') %>%
  str_replace_all("([a-z])([A-Z])", "\\1 \\2") %>%
  tolower() %>%
  str_replace_all("\\s+", " ") %>%
  str_replace_all("[^\x01-\x7F]", "")

disney_cali$review_text[100:102] # verification
```

```
[1] "we visited disneyland in october november we bought our day passes online and due to an
[2] "this was our first visit to disneyland and it was fantastic we have been to the orlando
[3] "disneyland is a place i like in general but this visit was a bit disappointing the cast
```

From these few reviews, we can see that our cleaning process worked. Next, we **tokenize** the reviews into words and remove any stop words simultaneously. This step is essential because tokenization allows us to split the text into meaningful units, such as words. We will use the *SMART* pre-made list of stop words. *SMART*, which stands for System for the Mechanical Analysis and Retrieval of Text, is an Information Retrieval System, an information retrieval system developed at Cornell University in the 1960s.

```
data("stop_words")
disney_cleaned <- disney_cali %>%
  unnest_tokens(word, review_text) %>%
  anti_join(stop_words)
disney_cleaned$word[200:250] # verification
```

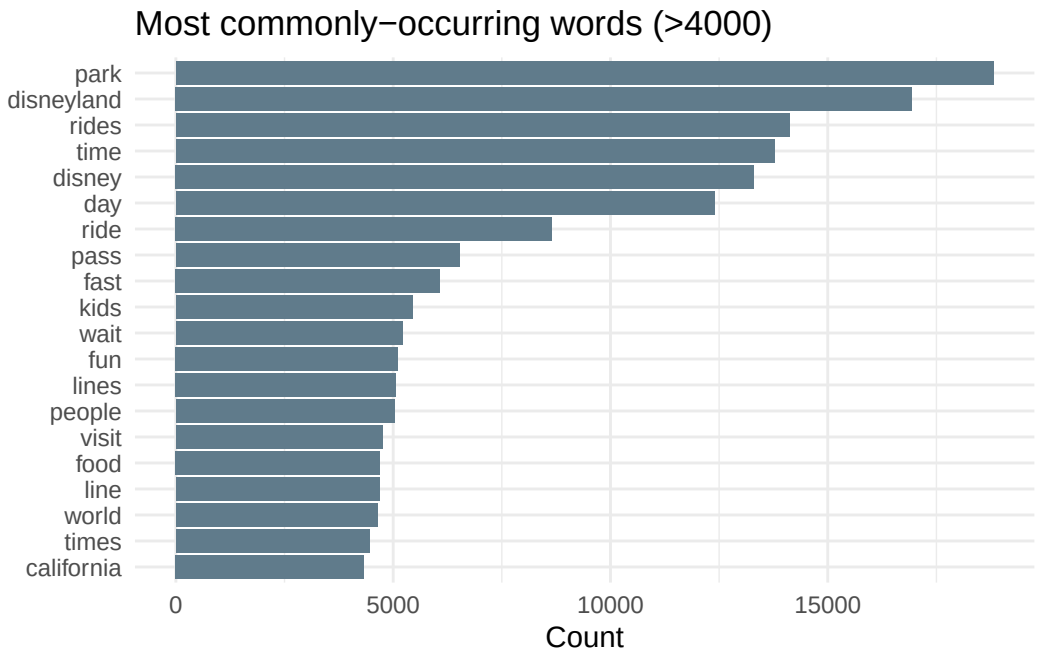
```
 [1] "plan"       "complete"   "attraction" "add"        "disneyland"
```

```
 [6] "bucket"     "list"       "youve"      "park"       "maintained"
[11] "parking"    "cheap"      "tickets"    "cost"       "adult"
[16] "peak"       "periods"    "food"       "cheap"      "family"
[21] "seasons"    "pass"       "easily"     "spend"      "hundreds"
[26] "dollars"    "admission"  "tickets"    "frequently" "disney"
[31] "credit"     "card"       "free"       "park"       "perks"
[36] "park"       "promotions" "downtown"   "disney"     "fun"
[41] "visit"      "disneyland" "earliest"   "memories"   "love"
[46] "visiting"   "park"       "won"        "friendlier" "helpful"
[51] "staff"
```

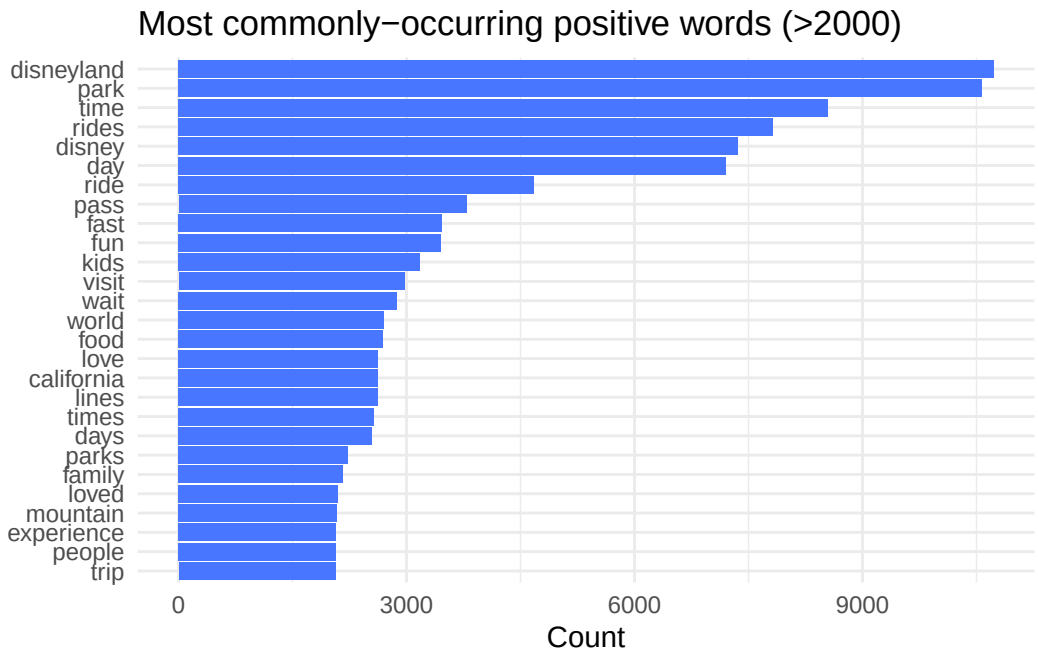
Most Commonly-Occurring Words

It looks like we are good to continue with our natural language processing. We will now perform some additional EDA, and create bar charts of our most commonly-occurring words overall, then the most commonly-occurring words according to sentiment.

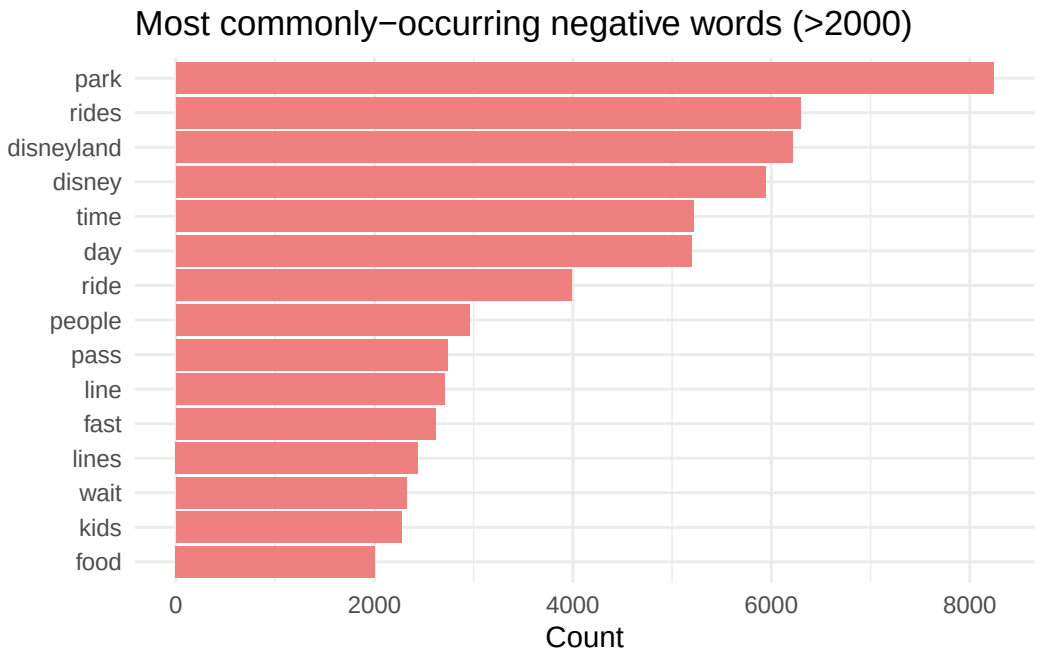
```
# most commonly-occurring words overall (>4000)
disney_cleaned %>%
  count(word, sort = TRUE) %>%
  mutate(word = reorder(word, n)) %>%
  filter(n > 4000) %>%
  ggplot(aes(n, word)) +
  geom_col(fill = "lightskyblue4") +
  labs(title = "Most commonly-occurring words (>4000)",
       x = "Count",
       y = NULL) +
  theme_minimal()
```




```
# most commonly-occurring positive words (>2000)
disney_cleaned %>%
  filter(sentiment == "positive") %>%
  count(word, sort=TRUE) %>%
  mutate(word = reorder(word, n)) %>%
  filter(n > 2000) %>%
  ggplot(aes(n, word)) +
  geom_col(fill = "royalblue1") +
  labs(title = "Most commonly-occurring positive words (>2000)",
       x = "Count",
       y = NULL) +
  theme_minimal()
```




```
# most commonly-occurring negative words (>2000)
disney_cleaned %>%
  filter(sentiment == "negative") %>%
  count(word, sort=TRUE) %>%
  mutate(word = reorder(word, n)) %>%
  filter(n > 2000) %>%
  ggplot(aes(n, word)) +
  geom_col(fill = "lightcoral") +
  labs(title = "Most commonly-occurring negative words (>2000)",
       x = "Count",
       y = NULL) +
  theme_minimal()
```



Some of the most commonly-occurring words overall are park, disneyland, rides, time, disney, day, and ride. Considering that there are more positive reviews (5 stars) than negative reviews (4 stars or less), it makes sense that there are far more positive words than negative words. The most commonly-occuring *positive* words include: disneyland, park, time, rides, disney, day, ride, and pass. The most commonly-occurring *negative* words include: park, rides, disneyland, disney, time, day, and ride.

We can see that the most commonly-occurring words for all three bar charts are pretty similar. This makes sense in the context that these words, like disneyland, park, rides, day, are all going to be common words in any review, negative or positive. So, it is helpful to look at the words with lower counts in the negative and positive charts, as it provides better insight into the more personal sentiments from reviewers.

In the *positive* bar chart, common words that reflect reviewer's personal experience at the park include: pass, fast, fun, family, loved, mountain, food, etc. On the *negative* chart, we see the common words of people, pass, line, wait, kids, food, etc.

From these words, we can use our own pre-existing knowledge of theme parks that *positive* reviews most likely talked about things like fast passes, spending time with family, having fun, rides with "mountain" in the name, and food sold at the park. The *negative* reviews hint at complaints or issues reviewers might have had with the park, like long wait times, long lines, a lot of people, expensive fast passes, kids, and food. However, it is not safe to assume that these are common things that people are saying in their reviews. So, we can analyze the most commonly-occurring **bigrams** to provide more context on these most commonly-occurring words according to sentiment.

Word Cloud

Before generating bigrams, we will also create a word cloud according to sentiment.

```
## cloud
disney_cleaned %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "blue"), scale=c(3, 0.5), max.words = 100)
```

The larger the words, the more common they are. The most commonly occurring words in *positive* reviews, such as fast and fun, indicate satisfaction with time-saving options, and the most commonly occurring words in *negative* reviews, such as crowded and expensive, indicate common complaints about wait times and cost, which all match with our analysis above.

TF-IDF values

One way to measure the importance of a word is its *term frequency (tf)*, which indicates how frequently a word occurs in a document. However, some words that occur many times may not be important, and certain stop words may be more important in some documents than others. Another approach is *inverse document frequency (idf)*, which decreases the weight of commonly used words and increases the weight of words that are less frequent across a

collection of documents. By combining two approaches, we can calculate a term's **tf-idf** (the product of tf and idf), which adjusts the frequency of a word by how rare it is used.

```
#tf_idf
tf_idf <- disney_cleaned %>%
  count(id, word) %>%
  bind_tf_idf(term = word,
              document = id,
              n = n)

#largest tf-idf values
tf_idf %>%
  arrange(desc(tf_idf)) %>%
  head(n = 30) %>%
  kbl() %>%
  scroll_box(width = "400px", height = "500px") %>%
  add_header_above(c("Top 30 Most Common" = 6))
```

Top 30 Most Common					
id	word	n	tf	idf	tf_idf
18168	whatelse	1	0.5000000	9.873338	4.936669
17268	wonderfull	1	0.5000000	7.475442	3.737721
4370	commercialize	1	0.3333333	9.873338	3.291113
11558	fanatastic	1	0.3333333	9.873338	3.291113
16996	selected	1	0.5000000	6.346977	3.173488
16892	cleanest	1	0.5000000	6.317990	3.158995
16937	adding	1	0.5000000	6.159765	3.079883
17846	definatley	1	0.3333333	9.180190	3.060064
17998	nuff	1	0.3333333	8.774725	2.924908
16871	grandparents	1	0.5000000	5.698950	2.849475
17591	therapy	1	0.3333333	8.487043	2.829014
14426	outgrow	1	0.3333333	8.263900	2.754633
16892	organised	1	0.5000000	5.248365	2.624182
2291	hollowen	2	0.2857143	9.180190	2.622911
16996	travelled	1	0.5000000	5.209899	2.604949
16823	exhaust	1	0.3333333	7.793896	2.597965
16982	elbow	2	0.3333333	7.793896	2.597965
18080	relevant	1	0.3333333	7.793896	2.597965
1747	kr	1	0.2500000	9.873338	2.468334
13881	olf	1	0.2500000	9.873338	2.468334
16625	blan	1	0.2500000	9.873338	2.468334
16625	ticketsbeing	1	0.2500000	9.873338	2.468334
16904	majical	1	0.2500000	9.873338	2.468334
17513	materialistic	1	0.2500000	9.873338	2.468334
17833	spares	1	0.2500000	9.873338	2.468334
17753	tons	1	0.5000000	4.882905	2.441453
18080	hard	2	0.6666667	3.456605	2.304403
11253	vocation	1	0.2500000	9.180190	2.295048
17827	especailly	1	0.2500000	9.180190	2.295048
17555	commercial	1	0.3333333	6.654462	2.218154

The *tf* value is the number of times a word appears in an essay divided by the total number of words in that essay. Most *idf* values tend to be relatively large. This indicates that these words are less common. As a result, they tend to have a larger *tf_idf* value.

Bigrams

We finally come to the part about **bigrams**, which means grouping tokens into pairs of consecutive words. We will also create bigram graphs for *positive* and *negative* reviews to visualize

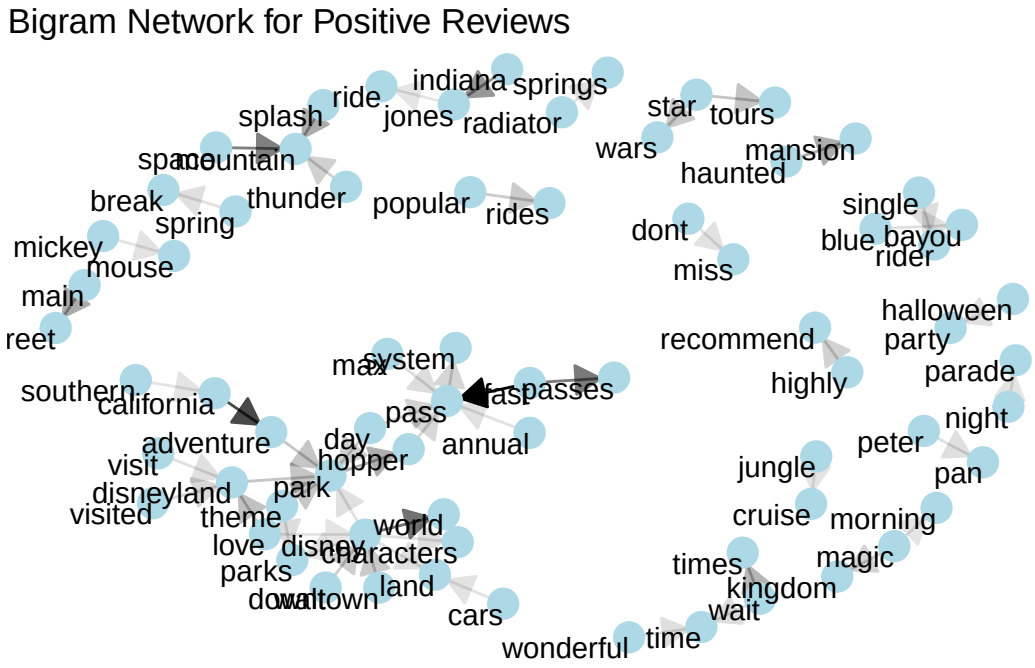
the relationships between pairs of words.

```
bigrams <- disney_cali %>%
  unnest_tokens(bigram, review_text, token = "ngrams", n = 2) %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word) %>%
  filter(!is.na(word1) | !is.na(word2))

unite_bigrams <- bigrams %>%
  unite(bigram, word1, word2, sep = " ")

unite_bigrams %>%
  count(bigram, sort = TRUE) %>%
  head(n = 30) #%>%
```

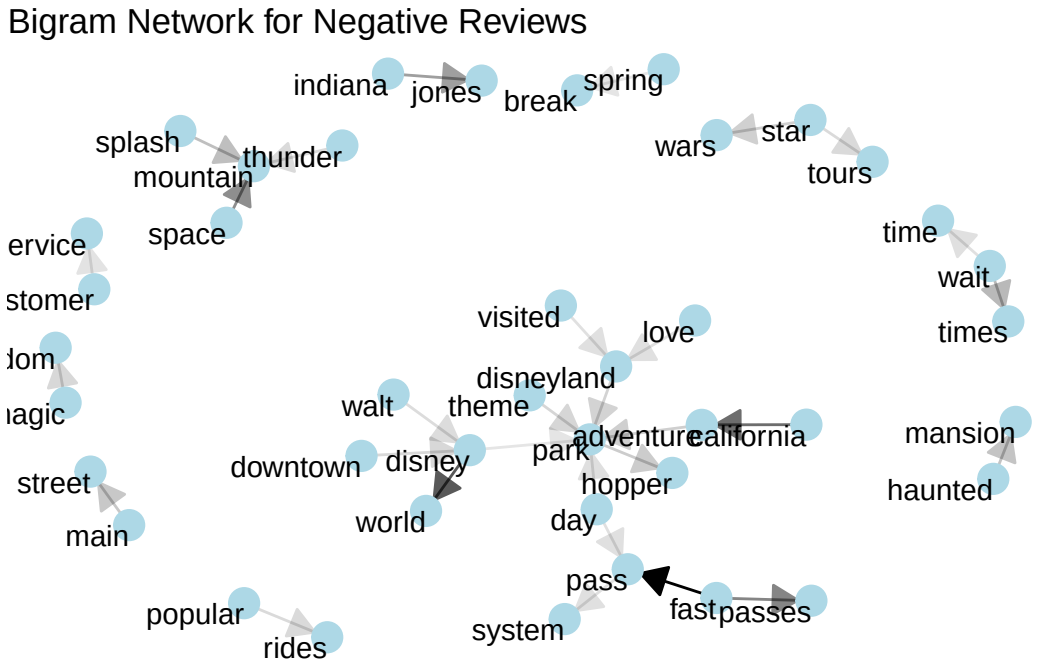
```
                bigram    n
1            fast pass 3593
2  california adventure 2329
3          disney world 2111
4           fast passes 1795
5        space mountain 1686
6         indiana jones 1387
7       splash mountain 1016
8       haunted mansion  984
9           main street  971
10           wait times  971
11          park hopper  916
12       adventure park  716
13      disneyland park  686
14            star wars  681
15           star tours  658
16      love disneyland  653
17           theme park  649
18      downtown disney  611
19        popular rides  587
20          walt disney  567
21     thunder mountain  556
22        magic kingdom  535
23             day park  533
24             day pass  490
25          pass system  467
```

```
26   visited disneyland  452
27            wait time  415
28    highly recommend  412
29         hopper pass  380
30           peter pan  361
```

```
  #kbl() %>%
  #scroll_box(width = "400px", height = "500px")

bigram_graph_positive <- bigrams %>%
  filter(sentiment == "positive") %>%
  count(word1, word2) %>%
  filter(n > 150) %>%
  graph_from_data_frame()

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

ggraph(bigram_graph_positive, layout = "kk") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  ggtitle("Bigram Network for Positive Reviews") +
  theme_void()
```




```
bigram_graph_neg <- bigrams %>%
  filter(sentiment == "negative") %>%
  count(word1, word2) %>%
  filter(n > 150) %>%
  graph_from_data_frame()

ggraph(bigram_graph_neg, layout = "kk") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  ggtitle("Bigram Network for Negative Reviews") +
  theme_void()
```



Unsurprisingly, both *positive* and *negative* reviews often mention terms like fast pass, fast passes, disney world, and california adventure, which are the main topics of reviews. In addition, there are more bigram terms in *positive* reviews than *negative* reviews, suggesting the reviewers are more expressive with reviewing their satisfaction of California's Disneyland park.

Model-fitting

Finally, we will conduct the model fitting, in which we will use the tokenized words and their corresponding Term Frequency-Inverse Document Frequency (TF-IDF) values to fit a few different models. We began by fitting **Binary Support Vector Machine, K-Nearest Neighbors, and Logistic Regression** to predict the sentiment of a given review as either positive or negative.

Since TF-IDF values can prioritize frequent words in a given review but are less common across the entire data set, it helps models focus on contextually significant words. We also split the data set into 70% training and 30% testing and use stratified sampling to ensure a balanced representation of positive and negative sentiments.

```
#set up for binary classification
set.seed(123)
disney_final <- disney_cleaned %>%
```

```
  group_by(sentiment) %>%
  mutate(sentiment=factor(sentiment)) %>%
  count(id, word) %>%
  bind_tf_idf(term=word,document=id,n=n)

disney_split <- initial_split(disney_final, strata=sentiment, prop=0.7)
disney_train <- training(disney_split)
disney_test <- testing(disney_split)

disney_recipe <- recipe(sentiment ~ tf+idf+tf_idf, data=disney_train) %>%
  step_zv(all_predictors())
prep(disney_recipe) %>% bake(disney_train) %>% head()
```

```
# A tibble: 6 x 4
      tf   idf tf_idf sentiment
   <dbl> <dbl>  <dbl> <fct>
1 0.0385  3.89 0.150  negative
2 0.0385  5.15 0.198  negative
3 0.0385  2.16 0.0832 negative
4 0.0385  4.27 0.164  negative
5 0.0385  7.31 0.281  negative
6 0.0385  4.60 0.177  negative
```

```
#binary svm
model <- svm_linear() %>%
  set_engine("LiblineaR") %>%
  set_mode("classification")

wflow <- workflow() %>%
  add_model(model) %>%
  add_recipe(disney_recipe)

fit <- fit(wflow, disney_train)
metrics <- metric_set(accuracy)
disney_results <- predict(fit, new_data=disney_test %>% select(-sentiment))
disney_results <- bind_cols(disney_results, disney_test %>% select(sentiment))
accuracy_bsvm <- metrics(disney_results, truth=sentiment, estimate=.pred_class)
```

```
#knn model
knn_mod <- nearest_neighbor(neighbors = 5) %>%
```

```
  set_mode("classification") %>%
  set_engine("kknn")

knn_wkflow <- workflow() %>%
  add_model(knn_mod) %>%
  add_recipe(disney_recipe)

knn_fit <- fit(knn_wkflow, disney_train)
metrics <- metric_set(accuracy)
disney_results <- predict(knn_fit, new_data=disney_test %>% select(-sentiment))
disney_results <- bind_cols(disney_results, disney_test %>% select(sentiment))
accuracy_knn <- metrics(disney_results, truth=sentiment, estimate=.pred_class)
```

```
#logistic model
log_reg <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

log_wkflow <- workflow() %>%
  add_model(log_reg) %>%
  add_recipe(disney_recipe)

log_fit <- fit(log_wkflow, disney_train)
metrics <- metric_set(accuracy)
disney_results <- predict(log_fit, new_data=disney_test %>% select(-sentiment))
disney_results <- bind_cols(disney_results, disney_test %>% select(sentiment))
accuracy_log <- metrics(disney_results, truth=sentiment, estimate=.pred_class)
```

Results 1

```
results <- tibble(
  model = c("Binary SVM", "KNN", "Logistic Regression"),
  .metric = "accuracy",
  .estimate = c(accuracy_bsvm$.estimate, accuracy_knn$.estimate, accuracy_log$.estimate))

results %>% arrange(desc(.estimate))
```

```
# A tibble: 3 x 3
  model               .metric  .estimate
  <chr>               <chr>        <dbl>
```

```
1 Binary SVM          accuracy     0.574
2 Logistic Regression accuracy     0.574
3 KNN                 accuracy     0.489
```

Among these three classification models, the **binary Support Vector Machine (SVM)** shows the highest accuracy, 57.41%. This result makes sense since SVM is able to handle the complexity of textual data, making it the optimal model for natural language processing. Our Logistic Regression model also has a similar predictive power, with an accuracy of 57.36%, which is very close to the accuracy of Binary SVM. However, its performance lagged slightly behind SVM due to the *linearity assumption*, which may not fully capture the nuanced relationships in textual data. Finally, our KNN model has a slightly lower accuracy of 48.86%, compared with the other two models, which suggests its ability to predict higher dimensional textual data is limited.

Overall, SVM is the most effective model for this task, while Logistic Regression is also a good model for predicting sentiment.

Next, we apply a **Multi-class Support Vector Machine** to predict customer ratings on term frequency-inverse document frequency (TF-IDF) values of textual reviews. Again, we split the data set into 70% training and 30% testing and use stratified sampling.

```
#multiclass SVM
disney_final <- disney_cleaned %>%
  group_by(rating) %>%
  mutate(rating=factor(rating)) %>%
  count(id, word) %>%
  bind_tf_idf(term=word,document=id,n=n)

disney_split <- initial_split(disney_final, strata=rating, prop=0.7)
disney_train <- training(disney_split)
disney_test <- testing(disney_split)

disney_recipe <- recipe(rating ~ tf+idf+tf_idf, data=disney_train) %>%
  step_zv(all_predictors())
prep(disney_recipe) %>% bake(disney_train) %>% head()
```

```
# A tibble: 6 x 4
      tf   idf tf_idf rating
   <dbl> <dbl>  <dbl> <fct>
1 0.0145  3.92 0.0568 1
2 0.0290  3.20 0.0928 1
3 0.0145  6.35 0.0920 1
4 0.0145  5.15 0.0747 1
```

```
5 0.0145  5.02 0.0728 1
6 0.0145  8.49 0.123  1
```

```
model <- svm_linear() %>%
  set_engine("LiblineaR") %>%
  set_mode("classification")
wflow <- workflow() %>%
  add_model(model) %>%
  add_recipe(disney_recipe)

fit <- fit(wflow, disney_train)
metrics <- metric_set(accuracy)
disney_results <- predict(fit, new_data=disney_test %>% select(-rating))
disney_results <- bind_cols(disney_results, disney_test %>% select(rating))
accuracy_msvm <- metrics(disney_results, truth=rating, estimate=.pred_class)
```

Results 2

```
results_2 <- tibble(
  model = c("Multi-class SVM"),
  .metric = "accuracy",
  .estimate = c(accuracy_msvm$.estimate))
results_2
```

```
# A tibble: 1 x 3
  model           .metric  .estimate
  <chr>           <chr>        <dbl>
1 Multi-class SVM accuracy     0.575
```

The **Multi-class Linear Support Vector Machine** has an accuracy of 57.49%, which indicates a fair ability to differentiate among the rating categories. These results reaffirm SVM's strength in processing high-dimensional, complex data structures.

As a conclusion to our project, our findings can provide insights that can guide California's Disneyland in developing marketing strategies, improving customer satisfaction, and enabling the park to be able to address consumer needs more effectively. Future work could incorporate bigram models to better capture contextual phrases in reviews, as well as including the other two locations of Disneyland parks (Hong Kong and Paris) in order to have a more diverse and detailed set of data to perform our natural language processing procedures and predictive model fitting on.