Sentimental Analysis and Prediction on Amazon Reviews

DA5030

Nithya Sarabudla

2024-04-15

Contents

Introduction

The growth of e-commerce has led to an abundance of user-generated content in the form of reviews, providing valuable insights into product perceptions and consumer sentiments. In this project, I analyze a dataset from the "Office Products" category, sourced from the Amazon Reviews dataset collected in 2023 by the McAuley Lab. This dataset comprises a rich collection of user reviews and item metadata, making this my dataset of choice to perform my analysis.

Dataset Overview

The "Office Products" category dataset consists of 7.6 million users and 710.4 thousand items, with a total of 12.8 million ratings. The dataset includes 574.7 million review tokens and 682.8 million metadata tokens, providing a substantial amount of textual data for analysis. Each user review includes information such as the rating of the product, the title and text body of the review, images posted by users, product IDs, reviewer IDs, timestamps, and helpful votes.

Purpose of Analysis

The primary goal of this analysis is to perform sentiment analysis on the user reviews in the "Office Products" category. By extracting insights from the textual data, I aim to understand the sentiments expressed by customers towards office products and identify key factors that influence their ratings and reviews. This analysis can provide valuable insights to businesses and manufacturers to improve product offerings and customer satisfaction.

Dataset Source

The dataset is sourced from the Amazon Reviews dataset collected in 2023 by the McAuley Lab. It includes user reviews, item metadata, and links between users and items, providing a comprehensive view of customer interactions on Amazon. The dataset is publicly available and can be accessed from the following link: Amazon Reviews Dataset.

In the following sections, I will discuss the preprocessing steps, the construction of machine learning classifiers, the development of an ensemble model, and the creation of a function for estimating product ratings based on text analysis.

1. Preprocessing

```r
# required packages
packages <- c("jsonlite", "dplyr", "tidytext", "ggplot2", "wordcloud", "tm", "caret", "e1071", "randomF

# Checking if packages are already installed
missing_packages <- setdiff(packages, rownames(installed.packages()))

# Installing the missing packages
if (length(missing_packages) > 0) {
  install.packages(missing_packages)
}

# Importing the required packages
library(jsonlite) # To read the JSONL data
library(dplyr) # To handle data frames
library(tidytext) # To get stop_words
library(ggplot2) # To plot graphs
library(wordcloud) # To generate word-clouds
library(tm) # To create corpus/DTM
library(caret) # To make data partitions
library(e1071) # To build Naive Bayes/SVM model
library(randomForest) # To build Random Forest model
```

1.1 Load Data

In the preprocessing stage, due to the dataset’s large size, I opted to read it row by row in Python as it was too large for R to handle. I filtered the top 1000 rows from each rating category, resulting in a more manageable dataset of 5000 rows. I then saved this new dataset as a separate JSONL file. I uploaded this new dataset file to my GitHub to make it publicly available.

```
# This is the python script I used to preprocess the data.
################################################################################
import json

file = "Office_Products.jsonl"
limit_for_each_rating = 1000
filtered = []
counter = {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0}
limit_reached = set()
with open(file, ’r’) as fp:
    for line in fp:
        parsed_line = json.loads(line.strip())
        current_rating = int(parsed_line["rating"])
        if counter[current_rating] < limit_for_each_rating:
            filtered.append(parsed_line)
            counter[current_rating] += 1
            if sum(counter.values()) >= 5 * limit_for_each_rating:
                break

print("Done reading!")

new_file_prefix = "New_Office_Products_"
with open(f"{new_file_prefix}{5 * limit_for_each_rating}.jsonl", "w") as f:
    for item in filtered:
        json.dump(item, f)
        f.write("\n")

print(f"Done, Created {new_file_prefix}{5 * limit_for_each_rating}.jsonl")
################################################################################
```

```
# Defining the data file name
url <- "https://raw.githubusercontent.com/sarabudla/Amazon_Office_Products_Reviews/main/New_Office_Produ
```

```
# Loading the dataset
data <- tibble(stream_in(url(url)))
```

```
##  Found 500 records... Found 1000 records... Found 1500 records... Found 2000 records... Found 2500 re
```

```
# Printing the dimensions of data
cat("Original shape of the dataset:", dim(data))
```

```
## Original shape of the dataset: 5000 10
```

In the above chunk, the dataset is loaded into a variable named `data`.

1.2 Data Cleaning and Tokenization

```
# filtering only required columns
data <- data[, c("rating", "text")]
# Dropping rows with missing values
data <- data[complete.cases(data), ]
# Filtering rows with review text more than 50 characters
data <- data[nchar(data$text) > 50, ]
# Printing the dimensions of data
cat("Shape of the dataset after filtering:", dim(data))
```

```
## Shape of the dataset after filtering: 4333 2
```

In the above chunk, I filtered the rows that doesn't contain any missing values (note: the rows contains just ratings and text, I'm checking for missing ratings or missing reviews). While dealing with these kind of datasets, we cannot impute any missing values for text as it is completely manual text. Additionally, I'm filtering the rows with review-text length of more than 50 characters as we cannot gain much from short sentences.

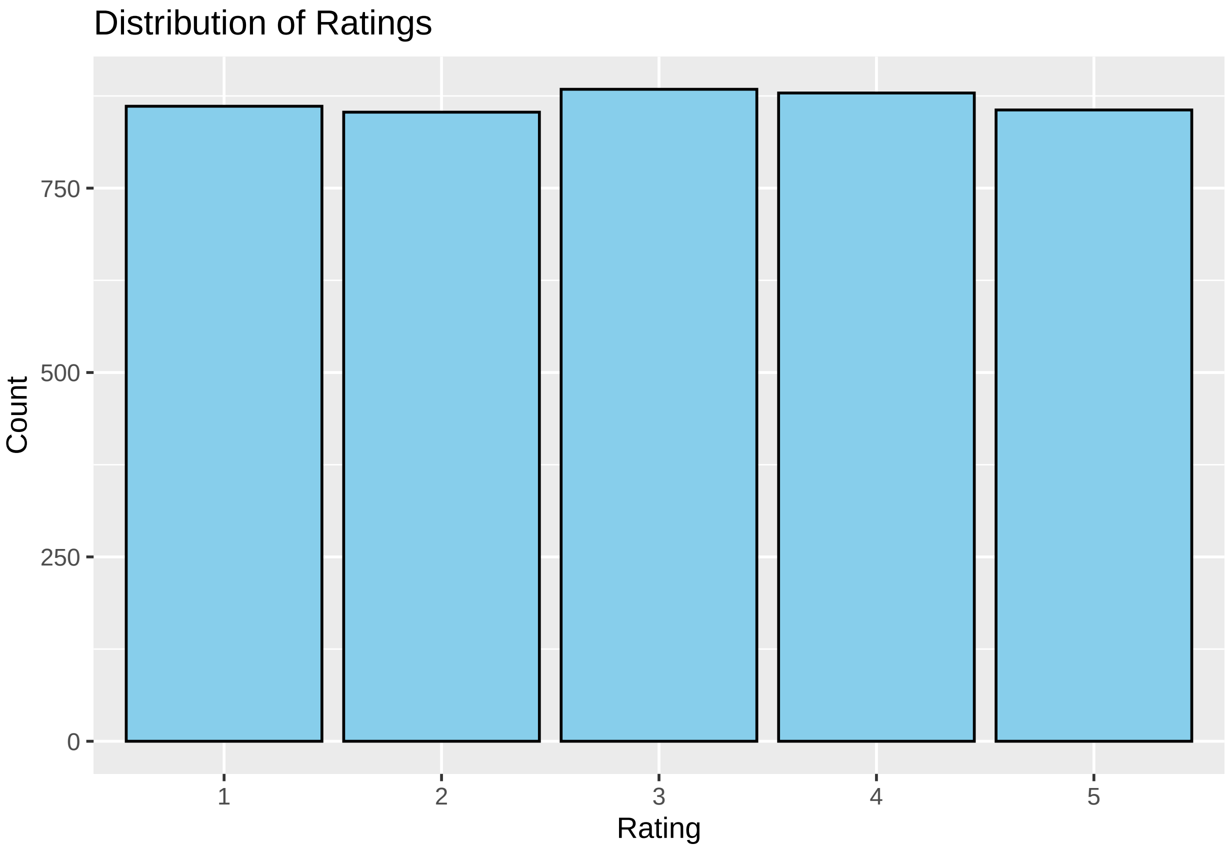
```
# Unnesting the tokens and subtracting the stop-words
tokens <- data %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)

cat("Found", nrow(tokens), "tokens!")
```

```
## Found 112646 tokens!
```

In the above chunk, I'm filtering to consider only ["rating", "text"] columns, then dropping the rows with missing values and filtering the reviews with more than 50 characters. Next, I converted these reviews into tokens.

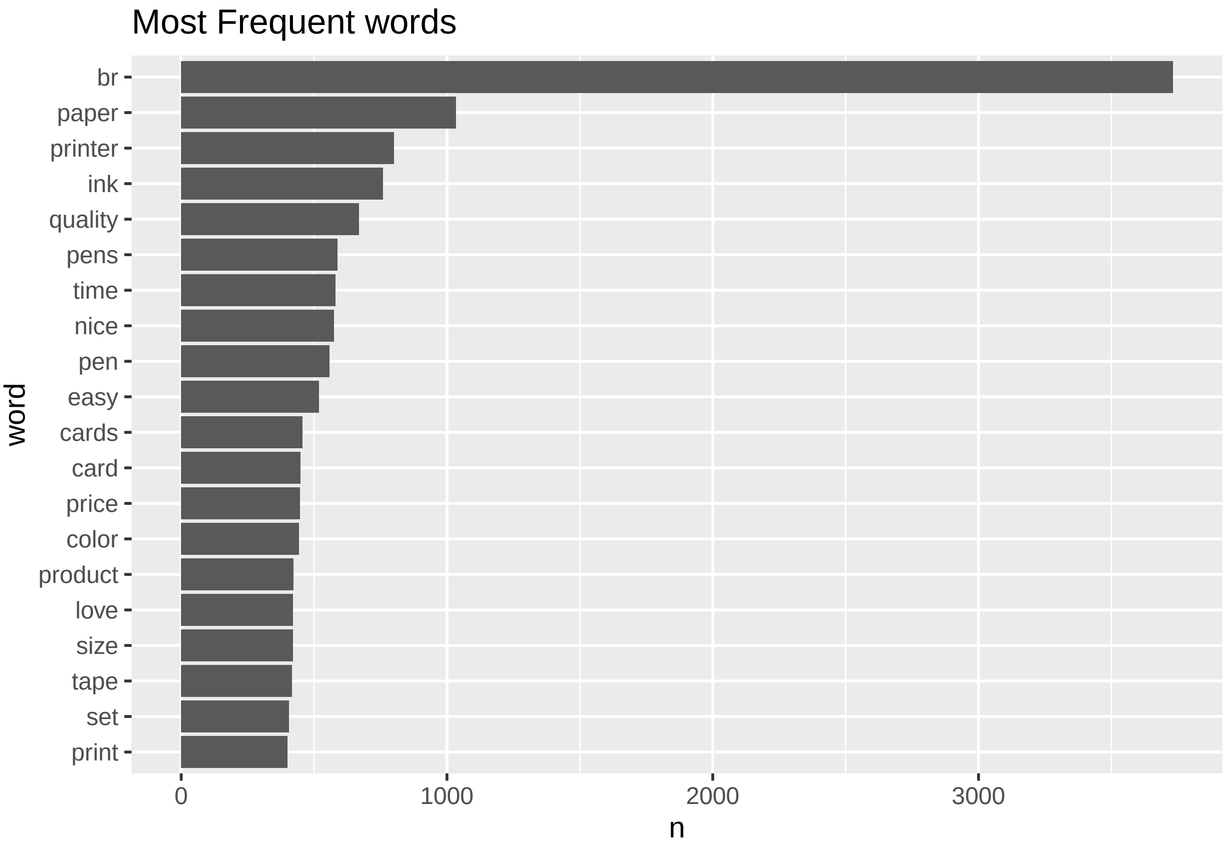
1.3 Exploratory Data Analysis

```r
# Converting 'rating' to a factor
data$rating <- factor(data$rating, levels = sort(unique(data$rating)))

# Creating a ggplot bar chart
ggplot(data, aes(x = rating)) +
  geom_bar(fill = "skyblue", color = "black") +
  labs(title = "Distribution of Ratings", x = "Rating", y = "Count")
```



The plot describes the proportions of each rating present in the dataset. All the ratings are almost uniformly distributed.

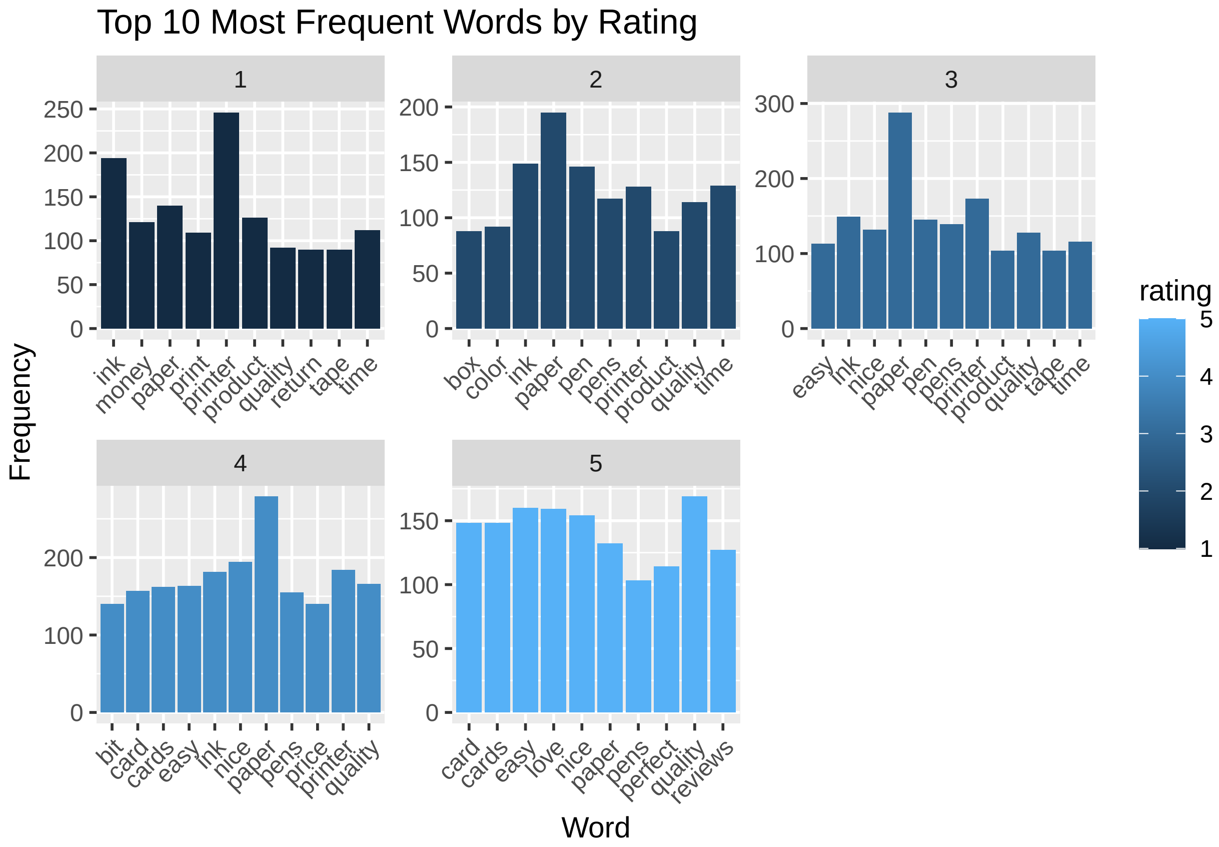
```r
# Counting the tokens, and plot the top 20 tokens sorted by their frequency.
tokens %>%
  count(word, sort = TRUE) %>%
  head(20) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col() +
  ggtitle("Most Frequent words")
```



The plot shows the most frequent tokens in the dataset. The most repeated token is “br”, this is due to the HTML encoding of the reviews, where `br` is to indicate a break line character. In the next step, I will remove that word from further consideration.

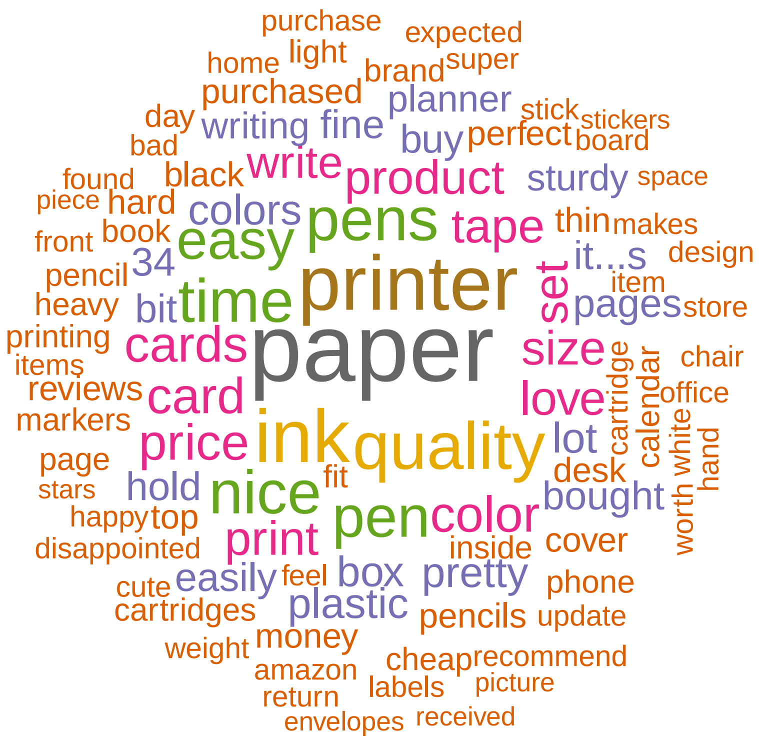
```r
# Filtering the unnecessary tokens such as "br" or any other numbers.
tokens <- tokens %>%
  filter(!word %in% c("br", as.character(seq(0:9))))
```

```r
# Grouping the tokens by their corresponding review's rating, and taking top 10 most frequent tokens fr
top_words <- tokens %>%
    group_by(rating, word) %>%
    summarise(freq = n(), .groups = "drop_last") %>%
    top_n(10, freq) %>%
    arrange(rating, desc(freq))

# Plotting charts showing the frequency
ggplot(top_words, aes(x = word, y = freq, fill = rating)) +
  geom_bar(stat = "identity") +
  facet_wrap(~rating, scales = "free") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Top 10 Most Frequent Words by Rating", x = "Word", y = "Frequency")
```



The above charts show the most frequent tokens in each rating group. It is clear that 1-star ratings are mainly due to the quality, ink, based on other reviews, and most of them are returned. On the other hand, for the 5-star ratings, users love the products, and are satisfied with the quality. Additionally, most of the words are related to printers, indicating it as the top product of this category. However, most of the words are common in all rating groups, this can easily confuse the models. To tackle this, I will group the ratings into 2 groups (positive and negative), 1, 2, and 3 star ratings fall in negative group and 4 and 5 stars fall in positive group. This makes it much simpler to classify.

```
word_freq <- table(tokens$word)
word_freq_df <- data.frame(word = names(word_freq), freq = as.numeric(word_freq))
wordcloud(words = word_freq_df$word, freq = word_freq_df$freq, max.words = 100, scale = c(3, 0.5), rand
```



The above plot is the wordcloud showing the frequencies of each word.

2. Preprocess for ML Classifiers

Before building machine learning classifiers for sentiment analysis, it is crucial to preprocess the textual data to extract meaningful features and prepare it for modeling. This section covers the steps involved in preprocessing the data, including creating a corpus, cleaning the data, creating a document-term matrix (DTM), and splitting the data into training and validation sets. These preprocessing steps are essential for ensuring the quality of the input data and improving the performance of the classifiers.

2.1 Create Corpus

The first step is to group the ratings and convert review text into a format that can be processed. This involves creating a corpus, which is a collection of reviews in a specific format that can be used for text analysis.

```
# grouping ratings into 2 categories
data <- data %>%
  mutate(rating_binary = recode(rating, `1` = 0, `2` = 0, `3` = 0, `4` = 1, `5` = 1))

# Converting the review text field to a vector source and then converting it to vector corpus
corpus <- VCorpus(VectorSource(data$text))

# A function to clean corpus.
```

```
clean_corpus <- function(corpus){
  corpus <- tm_map(corpus, stripWhitespace)
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, removeNumbers)
  corpus <- tm_map(corpus, removeWords, c(stopwords("en"), "br"))
  return(corpus)
}

cleaned_corpus <- clean_corpus(corpus)
```

In the above chunk, the data is converted to a vector source, and then it is converted to a vector corpus. After that, the corpus is cleaned going through multiple stages (stripping white space, removing punctuation marks, converting to lower case, removing numbers, removing stop words).

2.2 Create Document-Term Matrix

The Document-Term Matrix (DTM) is computed from the corpus and represents the frequency of terms (words) in each document. The process involves tokenizing the text (splitting it into words), counting the frequency of each word in each document, and constructing a matrix where rows represent documents and columns represent terms.

```
# Building the Document Term Matrix (DTM) on cleaned corpus
dtm <- DocumentTermMatrix(cleaned_corpus)
# Converting the DTM object to a matrix
dtm_matrix <- as.matrix(dtm)
# Printing the dimensions of dtm_matrix
dim(dtm_matrix)
```

```
## [1]  4333 12583
```

In the above chunk, the cleaned corpus is used to build a DTM object. It is then converted to a matrix like structure for further processing. It has 826 rows with 4407 columns. Each row indicate a review and each column indicates a token, and the value at a location indicates the frequency of that token. However, that is too many number of columns to represent each review, most of the columns would be sparse because of rare words, it is a good idea to remove these sparse columns, and thus reduce dimensionality.

```
# Dropping the sparse terms with a threshold of 0.98
dtm_matrix_non_sparse <- removeSparseTerms(dtm, 0.98)
dtm_df <- as.data.frame(as.matrix(dtm_matrix_non_sparse))
dim(dtm_df)
```

```
## [1] 4333  309
```

In the above chunk, I'm filtering the columns based on the sparsity. If a column is at least 98% sparse, then it is removed. This makes the new matrix to have just 298 columns, which is much better than 4407 dimensions.

2.3 Split Data

Splitting the data involves dividing it into two parts: one for training the machine learning model (usually around 80%) and the other for testing its performance (usually around 20%). This ensures the model is trained on one set of data and evaluated on another to assess various metrics like accuracy. I can also use cross-validation instead of this, but I'm going with traditional two sets approach as I made sure both the sets are stratified and contains similar class proportions.

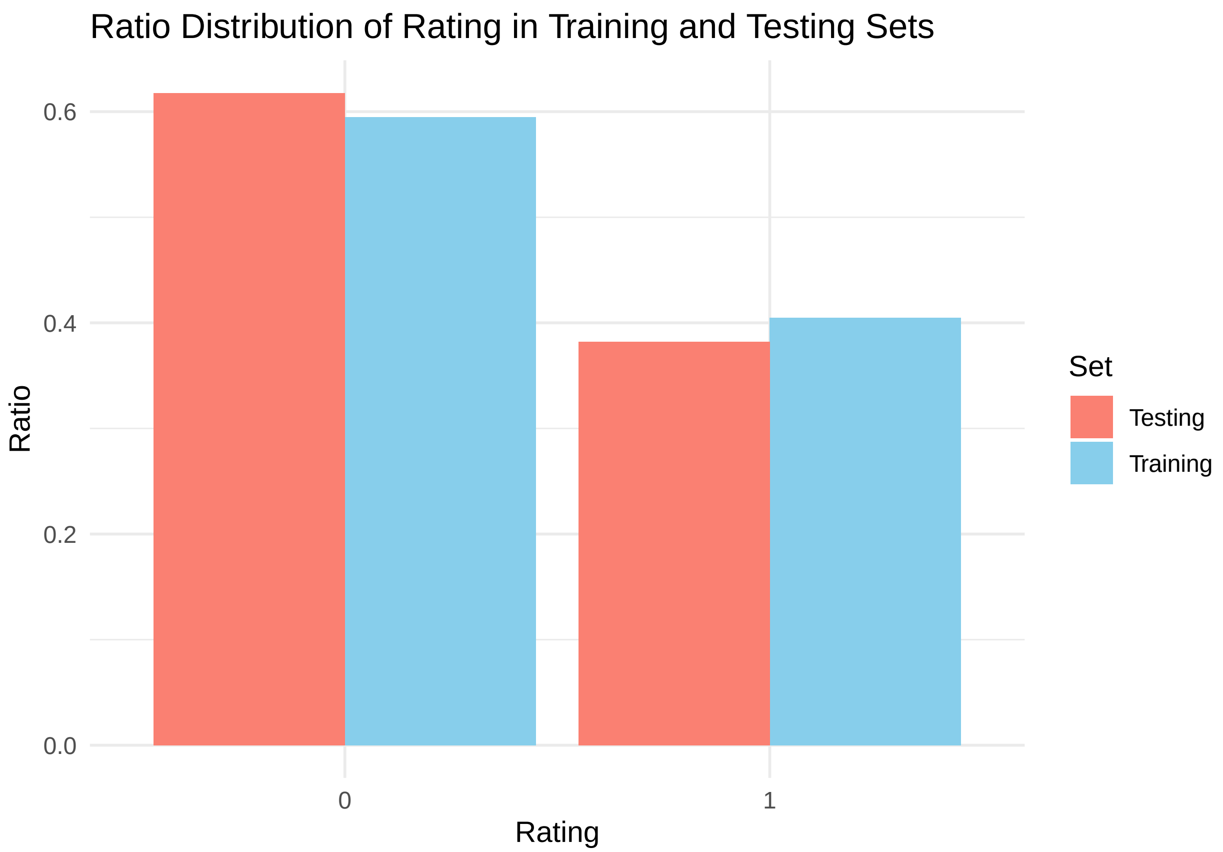
```r
# Adding the label (rating) to the dtm_df to keep everything in the same dataframe
dtm_df$rating <- as.factor(data$rating_binary)
colnames(dtm_df) <- make.names(colnames(dtm_df))

set.seed(123)

train_ratio = 0.8
indices <- createDataPartition(data$rating_binary, p = train_ratio, list = FALSE)

train_data <- dtm_df[indices, ]
test_data <- dtm_df[-indices, ]

cat("Train data dimensions:", dim(train_data), "\n")
```

```
## Train data dimensions: 3467 310
```

```r
cat("Test data dimensions:", dim(test_data), "\n")
```

```
## Test data dimensions: 866 310
```

```r
# Calculating the ratio distribution of the rating in both sets
train_rating_ratio <- prop.table(table(train_data$rating))
test_rating_ratio <- prop.table(table(test_data$rating))

# Combining the ratios and setting the labels
rating_ratio_data <- data.frame(
  Set = c(rep("Training", length(train_rating_ratio)), rep("Testing", length(test_rating_ratio))),
  Rating = factor(rep(names(train_rating_ratio), 2), levels = names(train_rating_ratio)),
  Ratio = c(train_rating_ratio, test_rating_ratio)
)

# Plotting the distribution using the ggplot bar chart
ggplot(rating_ratio_data, aes(x = Rating, y = Ratio, fill = Set)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Ratio Distribution of Rating in Training and Testing Sets", x = "Rating", y = "Ratio") +
  scale_fill_manual(values = c("Training" = "skyblue", "Testing" = "salmon")) +
  theme_minimal()
```



From the above charts, we can see that the distribution is identical. This stratification is necessary here because of the class imbalances.

3. Build Classifiers

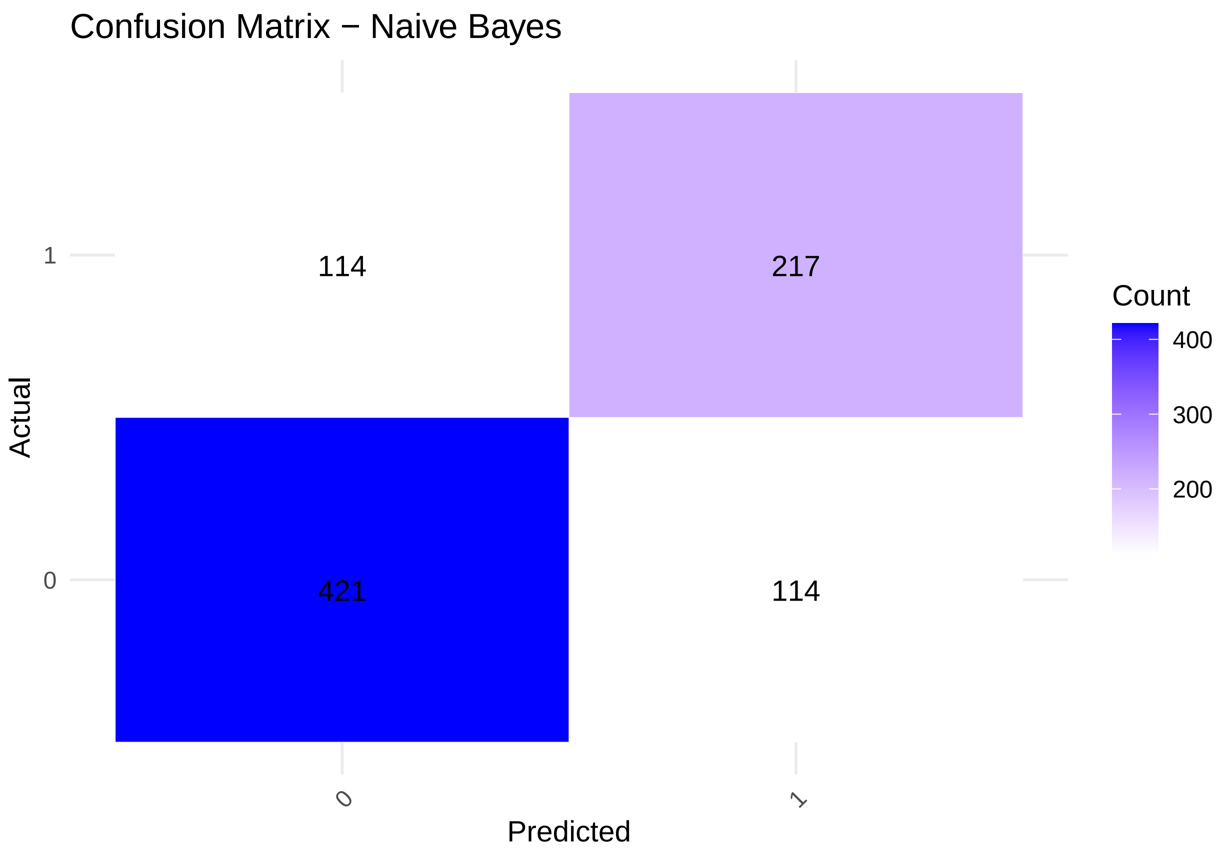
In this section, I will explore the process of building and training machine learning classifiers for sentiment analysis. I will start by selecting suitable models for the task, followed by training these models on the preprocessed data from the last section. Finally, I will evaluate the performance of each classifier to determine its performance in predicting sentiment.

```
# Defining few helper functions

# A function to plot confusion matrix like a heat map
plot_confusion_matrix <- function(cm, name = "") {
  # Convert confusion matrix to data frame
  cm_df <- as.data.frame(cm)
  colnames(cm_df) <- c("Predicted", "Actual", "Count")

  # Creating plot
  ggplot(data = cm_df, aes(x = Predicted, y = Actual, fill = Count)) +
    geom_tile(color = "white") +
    scale_fill_gradient(low = "white", high = "blue") +
    geom_text(aes(label = Count), vjust = 1) +
    labs(title = paste("Confusion Matrix -", name), x = "Predicted", y = "Actual") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
}

# A function to compute metrics such as Accuracy, Precision, Recall, F1-Score
# based on confusion matrix
compute_metrics <- function(cm) {
  # Computing overall accuracy
  total <- sum(cm)
  correct <- sum(diag(cm))
  accuracy <- correct / total

  # Computing precision, recall, and F1 score for each class
  metrics <- data.frame(Class = rownames(cm))
  metrics$Precision <- diag(cm) / rowSums(cm)
  metrics$Recall <- diag(cm) / colSums(cm)
  metrics$F1_Score <- 2 * (metrics$Precision * metrics$Recall) / (metrics$Precision + metrics$Recall)
  metrics$Accuracy <- rep(accuracy, nrow(metrics))
  return(metrics)
}

# Separating test set's rating column for easy processing when performing predictions.
test_labels <- test_data$rating
```

3.1 Naive Bayes Classifier

Naive Bayes is a probabilistic classifier based on applying Bayes' theorem with a "naive" assumption of feature independence. Despite its simplicity, Naive Bayes is effective in text classification and other tasks. It's particularly useful when working with a large number of features and limited training data.

```
nb_model <- naiveBayes(rating ~ ., data=train_data)
nb_predictions <- predict(nb_model, newdata=test_data)

nb_confusion_matrix <- table(test_labels, nb_predictions)
plot_confusion_matrix(nb_confusion_matrix, "Naive Bayes")
```



```
compute_metrics(nb_confusion_matrix)
```

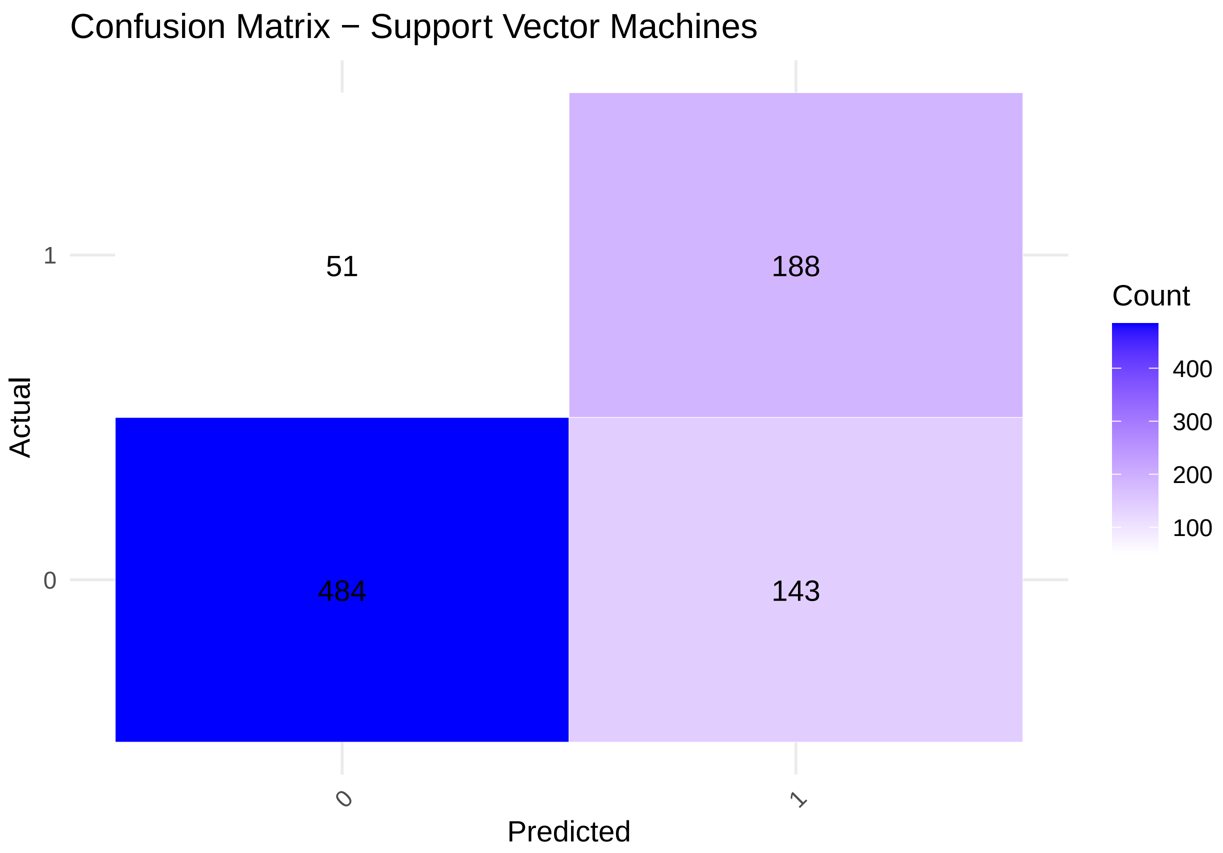
```
##   Class Precision Recall F1_Score Accuracy
## 1     0     0.787  0.787    0.787    0.737
## 2     1     0.656  0.656    0.656    0.737
```

Naive Bayes achieved an accuracy of 73.7%, with a precision of 78.7% and recall of 78.7% for negative reviews (rating 0), and a precision of 65.6% and recall of 65.6% for positive reviews (rating 1). It correctly predicted 114 negative reviews and 114 positive reviews, but it misclassified 217 positive reviews as negative and 421 negative reviews as positive.

3.2 SVM Classifier

SVM is a powerful supervised learning algorithm used for classification and regression tasks. It works by finding the hyperplane that best separates different classes in the feature space while maximizing the margin between the classes. SVMs are effective in high-dimensional spaces and are versatile due to the use of different kernel functions for non-linear decision boundaries. This can be very helpful in this project, because of several tokens and each token represent a dimension.

```
svm_model <- svm(rating ~ ., data=train_data)
svm_predictions <- predict(svm_model, newdata=test_data)

svm_confusion_matrix <- table(test_labels, svm_predictions)
plot_confusion_matrix(svm_confusion_matrix, "Support Vector Machines")
```



```r
compute_metrics(svm_confusion_matrix)
```

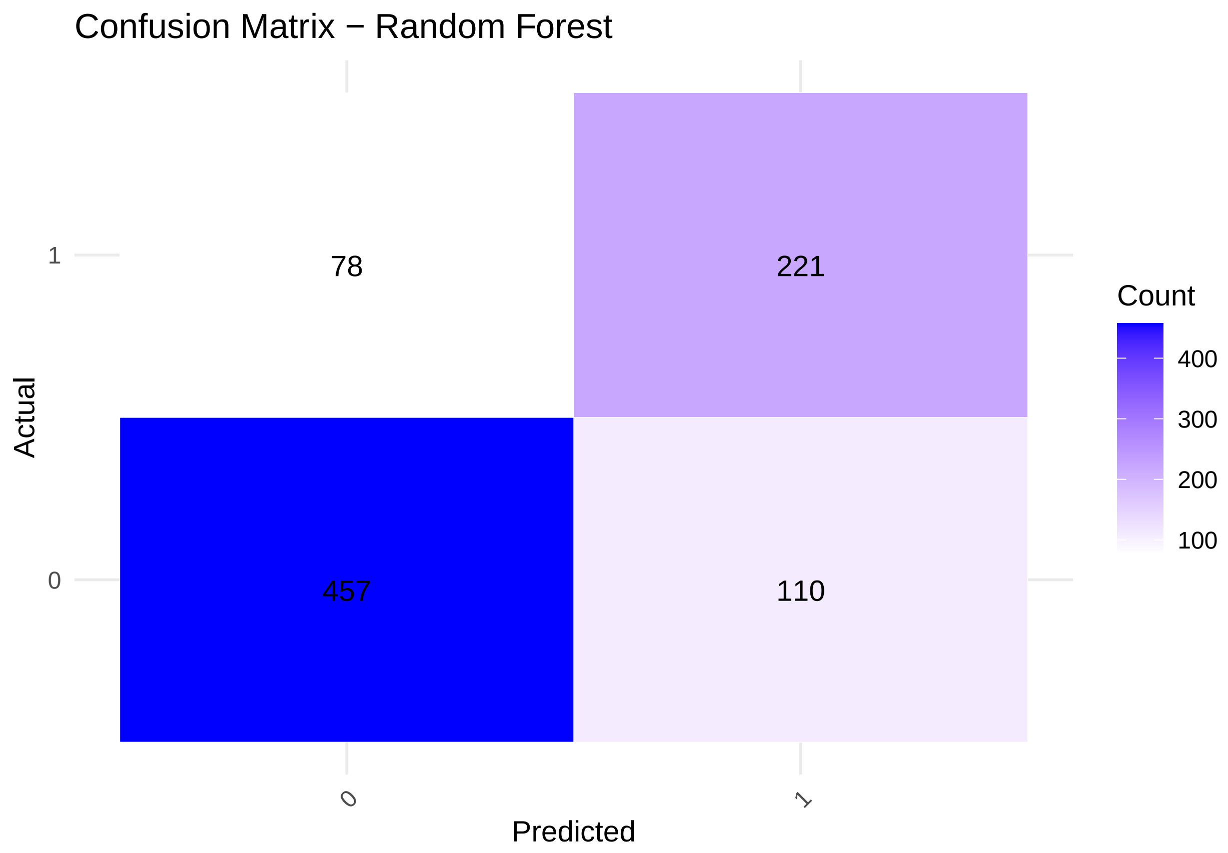
```
##   Class Precision Recall F1_Score Accuracy
## 1     0     0.905  0.772    0.833    0.776
## 2     1     0.568  0.787    0.660    0.776
```

The SVM model had an accuracy of 77.6%. It showed higher precision for negative reviews (90.5%) compared to positive reviews (56.8%), indicating its strength in correctly identifying negative sentiment. However, it misclassified 188 negative reviews as positive and 51 positive reviews as negative.

3.3 Random Forest Classifier

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Random Forests are known for their high accuracy, scalability, and ability to handle large datasets with high dimensionality. They also provide a way to measure the relative importance of each feature in the classification.

```r
rf_model <- randomForest(rating ~ ., data=train_data, ntree=500)
rf_predictions <- predict(rf_model, newdata=test_data)

rf_confusion_matrix <- table(test_labels, rf_predictions)
plot_confusion_matrix(rf_confusion_matrix, "Random Forest")
```

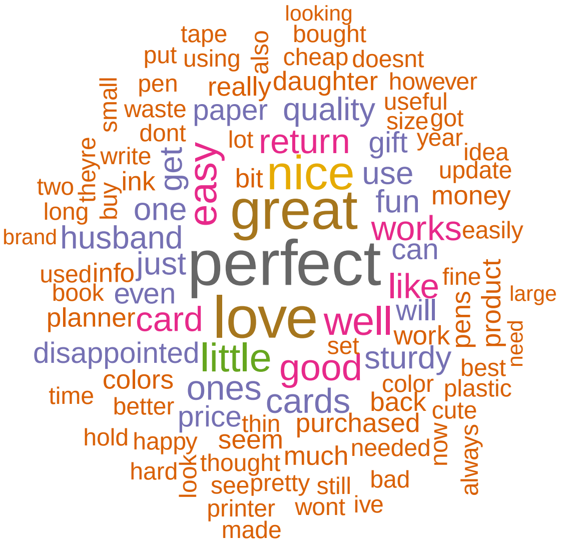


```
compute_metrics(rf_confusion_matrix)
```

```
##   Class Precision Recall F1_Score Accuracy
## 1     0     0.854  0.806    0.829    0.783
## 2     1     0.668  0.739    0.702    0.783
```

Random Forest also achieved an accuracy of 78.3%, with a precision of 85.4% and recall of 80.6% for negative reviews, and a precision of 66.8% and recall of 73.9% for positive reviews. It correctly predicted 110 positive reviews and 457 negative reviews, but it misclassified 221 positive reviews as negative and 78 negative reviews as positive.

```
importance <- rf_model$importance
rf_importance_df <- data.frame(term = rownames(importance), importance = importance[, "MeanDecreaseGini
wordcloud(words = rf_importance_df$term, freq = rf_importance_df$importance, min.freq = 1, max.words =
```



The above word cloud represents the feature importances, that indicates the most influencing words to take decisions. Words like great, perfect, nice, good, etc have more impact than the rest in deciding the sentiment of a review.

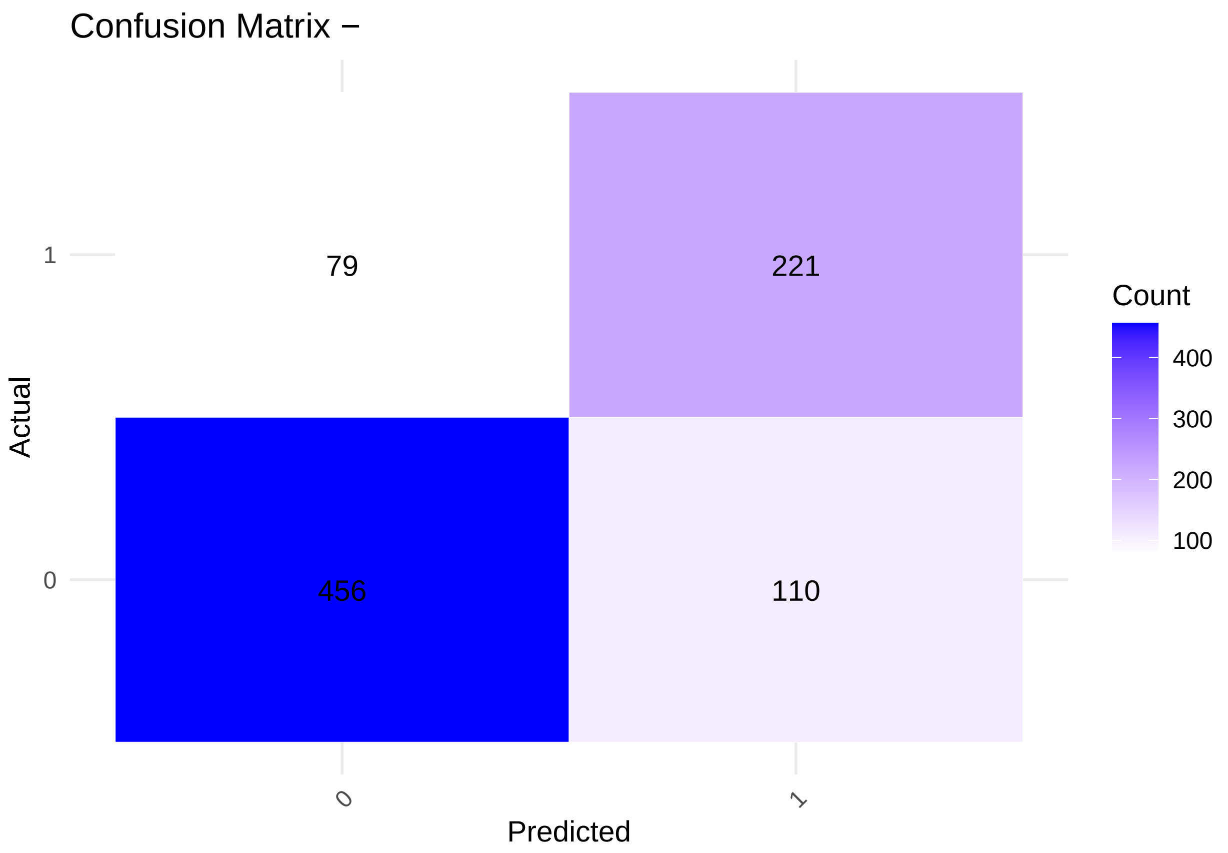
4. Build an Ensemble

In this section, I aim to combine the strengths of three above classifiers (Naive Bayes, Support Vector Machine (SVM), and Random Forest) into a single ensemble learner. Ensembles are powerful machine learning techniques that utilize the diversity of multiple models to improve overall predictive performance. By combining the predictions of individual classifiers, we can usually achieve better results than any single model alone. This section will demonstrate the process of creating an ensemble learner from these classifiers and evaluate its performance against the individual models.

4.1 Ensemble Model

```
# Creating a new data frame with the predictions of each model
ensemble_train_data <- data.frame(
  nb = predict(nb_model, newdata = train_data),
  svm = predict(svm_model, newdata = train_data),
  rf = predict(rf_model, newdata = train_data,),
  rating = train_data$rating
)
```

```r
# Training a Random Forest model on top of the predictions
ensemble_rf_model <- randomForest(rating ~ ., data = ensemble_train_data)

# Creating a new data frame with the predictions of each model on the test data
ensemble_test_data <- data.frame(
  nb = predict(nb_model, newdata = test_data),
  svm = predict(svm_model, newdata = test_data),
  rf = predict(rf_model, newdata = test_data)
)

# Making predictions using the ensemble Random Forest model
ensemble_predictions <- predict(ensemble_rf_model, newdata = ensemble_test_data)

# Computing confusion matrix
ensemble_confusion_matrix <- table(test_labels, ensemble_predictions)
plot_confusion_matrix(ensemble_confusion_matrix)
```



```r
compute_metrics(ensemble_confusion_matrix)
```

```
##   Class Precision Recall F1_Score Accuracy
## 1     0     0.852  0.806    0.828    0.782
## 2     1     0.668  0.737    0.700    0.782
```

The ensemble learner, which combines the predictions of Naive Bayes, SVM, and Random Forest, did not significantly outperform the individual models. It achieved an accuracy of 78.3%, with similar precision and recall values to the individual models. However, it did not provide a substantial improvement in classification accuracy compared to using the individual models alone.

4.2 Performance Comparison

When comparing the performance of the individual models to the ensemble learner, it is evident that the ensemble model does not offer a significant improvement in accuracy or other metrics. The ensemble learner achieved an accuracy of 78.3%, which is comparable to the accuracies of the individual models (Naive Bayes: 73.7%, SVM: 77.6%, Random Forest: 78.3%). Similarly, the precision, recall, and F1 scores of the ensemble model are similar to those of the individual models, indicating that combining the predictions of multiple models did not lead to a significant enhancement in performance.

I feel that one possible reason for this lack of improvement could be the similarity in the misclassifications made by the individual models. For example, both Naive Bayes and Random Forest frequently misclassified positive reviews as negative, which might have influenced the ensemble model's predictions. Additionally, the ensemble model's performance is limited by the performance of its component models. If the individual models have similar biases or weaknesses, the ensemble model may not be able to overcome them effectively. Also, I guess the model's performance can be improved using boosting as each learn tries to address the mistakes made by its previous learner.k

4.3 Testing

```r
# A function to perform predictions on a user's custom sentence
ensemble_predict <- function(sentence){

  cleaned_sentence <- clean_corpus(VCorpus(VectorSource(sentence)))

  # Converting the sentence to a document-term matrix
  sentence_dtm <- DocumentTermMatrix(
    cleaned_sentence,
    control = list(dictionary = Terms(dtm_matrix_non_sparse))
  )

  # Converting the DTM to a matrix
  sentence_matrix <- as.matrix(sentence_dtm)
  sentence_df <- as.data.frame(sentence_matrix)
  colnames(sentence_df) <- make.names(colnames(sentence_df))

  # Creating a data frame with the predictions of each model
  predictions <- data.frame(
    nb = predict(nb_model, newdata = sentence_df),
    svm = predict(svm_model, newdata = sentence_df),
    rf = predict(rf_model, newdata = sentence_df)
  )

  # Making predictions using the ensemble Random Forest model
  ensemble_prediction <- predict(ensemble_rf_model, newdata = predictions)

  # Printing the predictions of each model
  cat("Naive Bayes Prediction:", as.integer(predictions$nb) - 1, "\n")
  cat("SVM Prediction:", as.integer(predictions$svm) - 1, "\n")
  cat("Random Forest Prediction:", as.integer(predictions$rf) - 1, "\n")
  cat("Ensemble Prediction:", as.integer(ensemble_prediction) - 1, "\n")
}
```

```
test_sentence = "This is a great printer, It was so easy to install. The paper quality is also so good.
# test_sentence = "The quality is so bad, I want to return this item."
ensemble_predict(test_sentence)
```

```
## Naive Bayes Prediction: 1
## SVM Prediction: 1
## Random Forest Prediction: 1
## Ensemble Prediction: 1
```

Using this function, I can pass it any custom review, and it predicts the sentiment. Based on the precision/recall scores users/companies can choose the result of any model. The output shows the result from all the 4 models, where 0 indicates a review with negative sentiment and 1 indicates a positive sentiment. With this, the models can be integrated into apps, tools, or websites where users can pass it a sentence to check the tone of it.

Challenges

I encountered several challenges during the project. Initially, the dataset I selected is too huge to be loaded in R. So, I had to preprocess it using Python by reading it line by line. Then the dataset was highly imbalanced, with some rating classes having significantly fewer samples than others. To address this, I again used Python to preprocess the data and ensure each rating class had an equal number of samples.

However, even after balancing the dataset, I faced a new challenge with the model confusing between 1 and 2 star ratings, as well as 4 and 5 star ratings. To overcome this issue, I decided to merge the 1, 2, and 3 star ratings into a single group (0, negative sentiment), and the 4 and 5 star ratings into another group (1, positive sentiment). This approach helped improve the model's performance and reduce confusion between similar rating classes.

Conclusion

In this project, I conducted a sentiment analysis on Amazon's "Office_Products" category reviews using machine learning techniques in R. I followed the CRISP-DM process, starting with data preprocessing, where I cleaned and tokenized the review text. I then built and trained three different classifiers (Naive Bayes, SVM, Random Forest) to predict the sentiment of the reviews.

The Naive Bayes model achieved an accuracy of 73.7%, with a precision of 78.7% for negative sentiment (rating 0) and 65.6% for positive sentiment (rating 1). The SVM model achieved an accuracy of 77.6%, with a precision of 90.5% for negative sentiment and 56.8% for positive sentiment. The Random Forest model achieved an accuracy of 78.3%, with a precision of 85.4% for negative sentiment and 66.8% for positive sentiment.

After evaluating the individual classifiers, I created an ensemble learner by combining the predictions of the three models. The ensemble model achieved an accuracy of 78.3%, with a precision of 85.4% for negative sentiment and 66.8% for positive sentiment. This ensemble model showed an equal performance compared to the individual classifiers.

I also explored techniques to address imbalanced data and experimented with different strategies to improve the accuracy of the models. By mapping the ratings to two classes (positive and negative), I achieved a higher accuracy on the testing data.

Overall, this project demonstrates the application of machine learning in sentiment analysis, showcasing the importance of preprocessing, model selection, and ensemble learning. The developed models can be useful for businesses to analyze customer sentiment and make informed decisions based on customer feedback.

References

@article{hou2024bridging, title={Bridging Language and Items for Retrieval and Recommendation}, author={Hou, Yupeng and Li, Jiacheng and He, Zhankui and Yan, An and Chen, Xiusi and McAuley, Julian}, journal={arXiv preprint arXiv:2403.03952}, year={2024} }