Sentiment Analysis on Steam Game Reviews

Team Members:
Xinyu Lian (lian7)
Yu Zhang (yz93)
Yuankai Wang (yuankai4)
Yutao Zhou (yutaoz3)

1 Overview of the Code

The dataset being used in the project is the steam game reviews from Kaggle [https://www.kaggle.com/datasets/smeeeow/steam-game-reviews]. There are totally 187 files in the dataset, each file records the users' review for a single game. In each file, there are 22 features and each row contains the information of a single review from a user that purchased and played the game.

Our code does a sentiment analysis on steam game reviews. We are able to estimate the attitude of a game review and also whether the review can be helpful or funny. Our code can be mainly divided into three parts: dataset preprocessing, text preprocessing, and model building.

2 Software Implementation

2.1 Dataset preprocessing

Before conducting the analysis on the steam game review data, data preprocessing is important. The preprocessing can be divided into 6 parts. The code for each process is provided.

1. Remove empty files
   There are 6 empty files that are removed from the dataset.

```
def is_file_empty(file_path):
  with open(file_path, 'r') as file:
      for line in file:
          if line.strip():  # Check if the line is not empty
              return False
  return True
```

2. Remove unuseful and empty columns
   Among the 187 data files, some files have 23 columns while some have 21 columns. The extra 2 columns are ‘timestamp_dev_responded’ and ‘developer_response’. These two columns are mostly empty and unrelated to the sentiment analysis topic, thus they are removed from the files.

```
# Check if the columns exist in the DataFrame
if 'timestamp_dev_responded' in df.columns:
   # Drop the 'timestamp_dev_responded' column
   df.drop('timestamp_dev_responded', axis=1, inplace=True)

if 'developer_response' in df.columns:
   # Drop the 'developer_response' column
   df.drop('developer_response', axis=1, inplace=True)
```

3. Remove rows with missing rows
   After the analysis of completeness of the files, we find that the missing data only makes up a very small portion of the total data in a single file. As a result, we choose to remove the rows that contain missing value.

```
# Remove rows with empty cells and drop them from the DataFrame
df.dropna(inplace=True)
```

4. Remove non-english reviews
   The project focused on analyzing the sentiment for reviews in English, so the data containing reviews written in other languages are removed. This is conducted by filtering out the rows whose ‘language’ feature is not ‘english’.

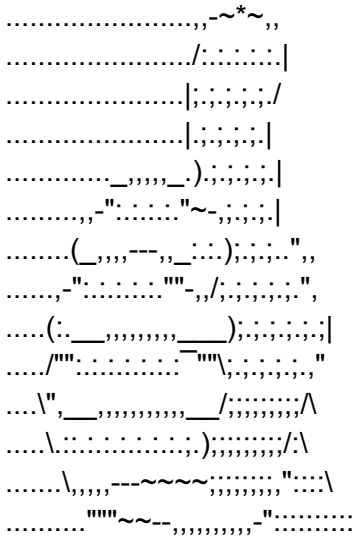
```
# Filter out rows where 'language' is not 'english' or 'English'
df = df[(df['language'].str.lower() == 'english')]
```

5. Remove reviews containing non-English word
   After completing step 4, we still found many reviews not written in English but with ‘language’ equaling ‘english’, which means the feature ‘language’ is not reliable. Thus we use `'nltk.corps.words'` as an English dictionary. If the review contains any non-English word, we will remove it.

```
import nltk
from nltk.corpus import words as nltk_words

nltk.download('words')
nltk_words = set(nltk_words.words())

def contains_non_english_words(text):
  # Check if any word in the text is not found in the English
dictionary
```

```
  words = text.split()
  return not any(word.lower() in nltk_words for word in words)

# Remove rows with non-English words in 'review'
df = df[~df['review'].apply(contains_non_english_words)]
```

6. Remove no letter reviews
   In the game reviews, it is common to see that the reviews contain no letter at all. For example, users will use non-letter symbols to represent an image as a review. Below is an example of using symbols to represent a thumbs up:



   Some users will just put a score like ‘x/10’ score, an emoticon like ‘:)’ or an emoji like ‘👍’ as a simple grading in the review section. Although these representations also convey the sentiment in a way, the topic of the project is sentiment analysis on the texts in natural languages. As a result, the reviews containing no letter at all are removed.

```
def contains_no_letters(text):
  # Check if the text contains no letters
  return not any(char.isalpha() for char in text)
  # Remove rows with no letters in 'review'
  df = df[~df['review'].apply(contains_no_letters)]
```

After the preprocessing on the dataset, all the missing values and files are eliminated and the remaining review texts are the ones containing only English words. Then we need to further process the review texts.

2.2 Review texts preprocessing

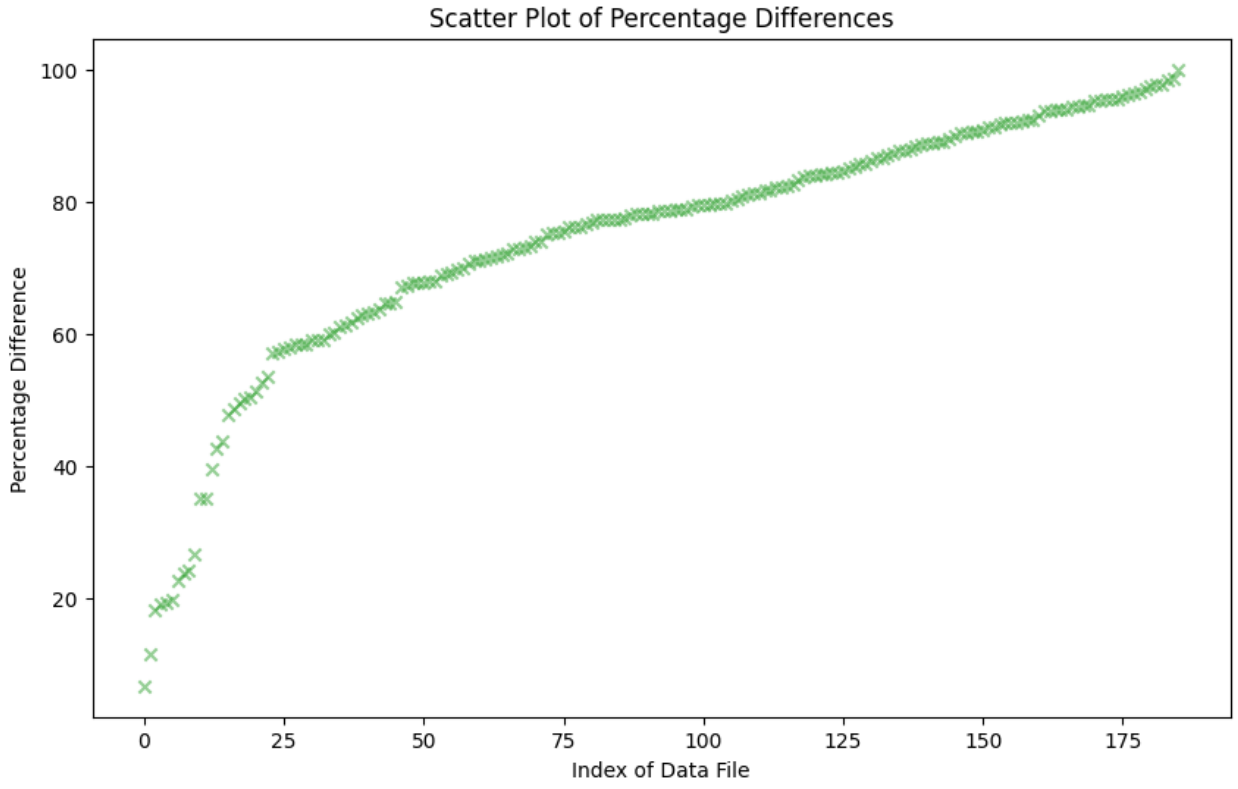
After preprocessing the dataset, we need to further conduct the preprocessing on the review text before further steps. The text preprocessing contains the following steps:

1) Convert the text to lowercase
2) Remove punctuation
3) Remove numbers
4) Remove stopwords
5) Stem the words

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

# Download stopwords from NLTK
nltk.download('stopwords')

# Function to preprocess the text
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in
stop_words])
    # Stem the words
    stemmer = SnowballStemmer('english')
    text = ' '.join([stemmer.stem(word) for word in text.split()])
    return text

# Apply the preprocessing function to the review texts
steam_filtered['review'] = steam_filtered['review'].apply(preprocess_text)
```

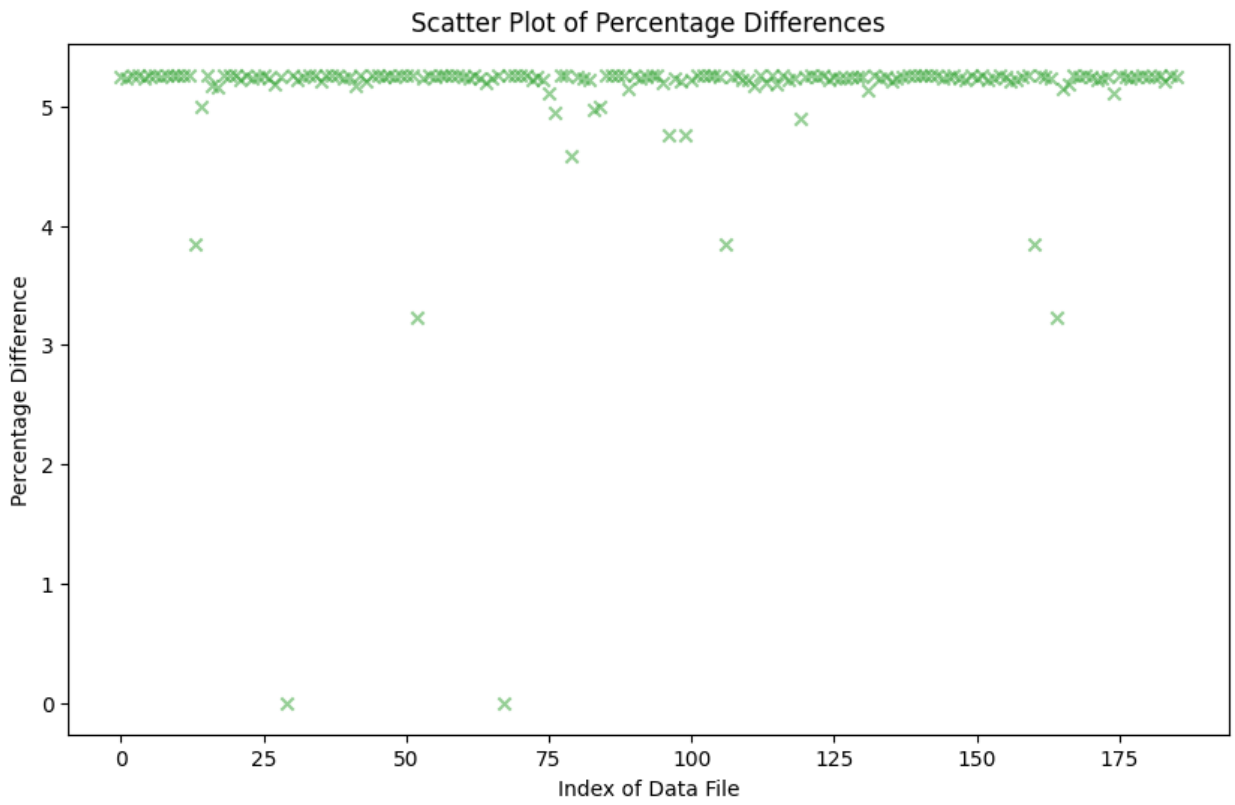
2.3 Unbalanced dataset

Our examination of the dataset showed that some files are very unbalanced, which could negatively affect the performance of our model. We used the percentage difference between two categories to better understand this imbalance. The formula for calculating the percentage difference is $|Value1 - Value2| / (Value1 + Value2) * 100$. Before considering any resampling actions, we noticed that the difference in some files is quite large, nearly reaching 100%. This suggests the need for balancing measures in the dataset.



To resolve this problem, we did the resampling work by manually truncating the category with a larger quantity to ensure that the difference in the number of items between it and the category with fewer items does not exceed 10%.

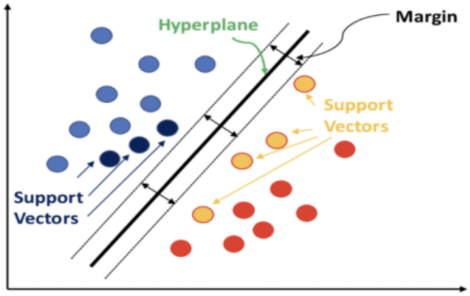
```
# Calculate the target size for the larger category
max_allowed_difference = 0.1  # 10%
smaller_size = min(len(positive_reviews), len(negative_reviews))
larger_size_target = int(smaller_size / (1 - max_allowed_difference))
```



2.4 Model Building

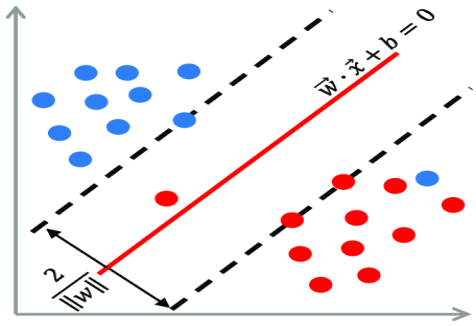
In this part, we have explored multiple algorithms.The following would be the explanation of each algorithm we used.

- **Support Vector Machine (SVM):** SVM finds the optimal hyperplane in a high-dimensional space that best separates different classes in a dataset. This hyperplane acts as a decision boundary, categorizing new data points into one of the classes based on which side of the hyperplane they fall on. It's great for categorizing text data in our case, where the goal is to divide positive and negative sentiments.

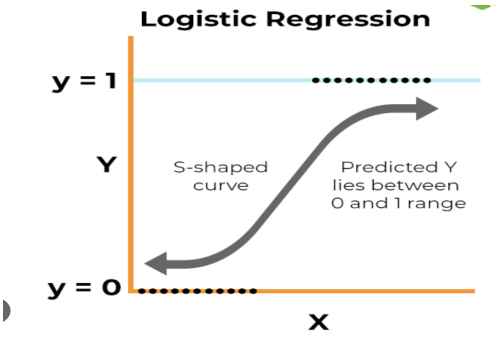


- **Linear Support Vector Classification (LinearSVC):** LinearSVC is a variant of SVM. Unlike traditional SVM that works in high-dimensional spaces, LinearSVC primarily

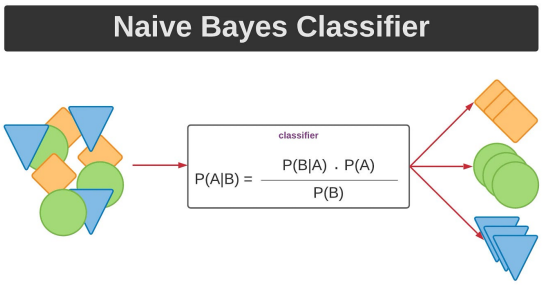
operates by constructing a linear hyperplane. This hyperplane effectively separates different classes within the dataset. By focusing on linear separations, LinearSVC often provides both efficient and accurate results, especially in sentiment classification tasks where the objective is to categorize data into distinct sentiment groups based on their textual content.



- **Logistic Regression:** Logistic Regression is a widely-used statistical method for predicting binary outcomes.This method applies a logistic function to model the probability distribution of the outcomes, making it particularly effective in sentiment analysis.



- **Naive Bayes:** Naive Bayes is a simple probability classifier based on Bayes' theorem, assuming independence among features. Its simplicity and computational efficiency make it highly effective for processing large text datasets. It performs well with bag-of-words features, which simplifies text data into a format where the frequency of each word is used to predict sentiment, making it a robust choice for distinguishing between positive and negative sentiments in text.



- **Decision Trees:** Decision Tree has a tree structure where an internal node represents a feature, and each leaf represents a decision outcome. This structure enables the model to mimic human decision-making processes, making it highly interpretable and easy to visualize. Its ability to handle non-linear relationships between features makes it particularly effective for sentiment analysis, where the context and subtleties of language play a crucial role.

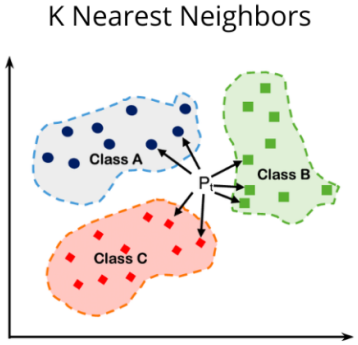
- **Random Forest Tree:** Random Forest takes advantage of multiple decision trees to improve predictive accuracy and control over-fitting. It's effective in sentiment analysis for handling non-linear data and providing insights into the importance of different features.

- **Adaptive Boosting (AdaBoost):** AdaBoost is a boosting algorithm that adjusts the weights of weak classifiers to turn them into strong ones. This process effectively builds a strong composite model from a sequence of weaker ones. Each subsequent model in the series focuses more on the data points that previous models classified incorrectly, leading to improved overall accuracy. This feature of AdaBoost makes it particularly effective for sentiment analysis, as it can adaptively refine its approach to more accurately categorize complex or nuanced sentiments in text data.

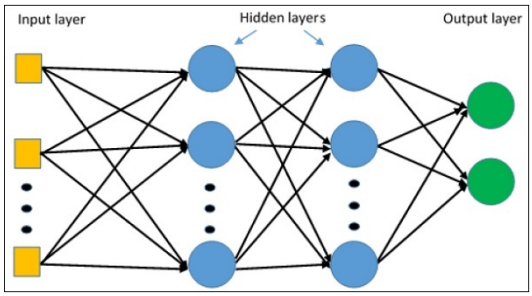
- **Gradient Boosting:** Gradient Boosting is a machine learning technique for regression and classification problems. Its application in sentiment analysis helps in achieving higher accuracy through building sequential models that correct the predecessors' errors.

- **XGBoost:** XGBoost is an optimized distributed gradient boosting library, designed to be highly efficient and flexible. It can handle a variety of data types with efficient speed and performance.

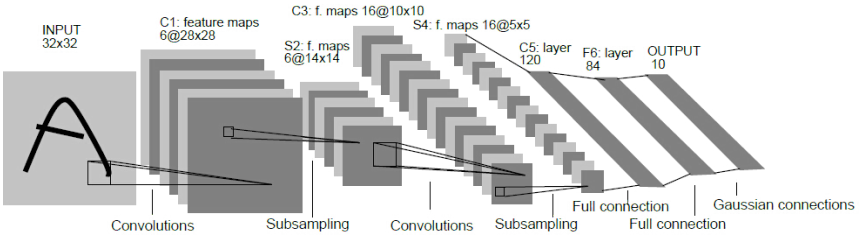
- **K-Nearest Neighbors (KNN):** KNN is an instance-based learning algorithm that stores all available cases and predicts the classification of new instances primarily based on the similarity measures with stored cases. This method works effectively in sentiment analysis by identifying the k-nearest neighbors (where 'k' is a predefined number) to a new data point and classifying it based on the majority sentiment of these neighbors.



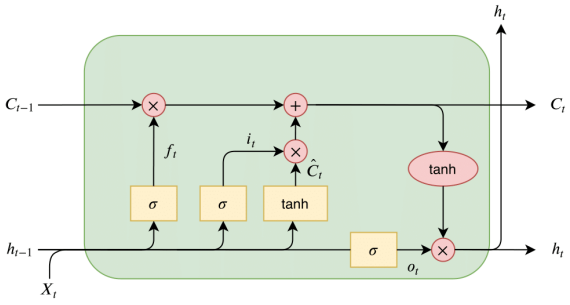
- **Multilayer Perceptron (MLP):** MLP is a type of artificial neural network, consisting of at least three layers of nodes, typically including an input layer, one or more hidden layers, and an output layer. Its deep learning capabilities, derived from the intricate network of interconnected nodes across layers, allow for the nuanced understanding and processing of language features, essential for accurate sentiment analysis.



- **Convolutional Neural Network (CNN):** CNNs are a type of deep neural networks, renowned for their effectiveness in processing data with a grid-like structure, such as images. However, they can also be highly effective for text analysis. The layers apply filters that detect patterns and semantic representations within text data, which can be effective in capturing local features and semantic representations from text data.



- **Long Short-Term Memory (LSTM):** LSTM is a type of recurrent neural network (RNN) that is capable of learning order dependence in sequence prediction problems. This ability to process and remember long sequences of data, including words and their contextual relationships, is crucial in understanding the nuances of language.



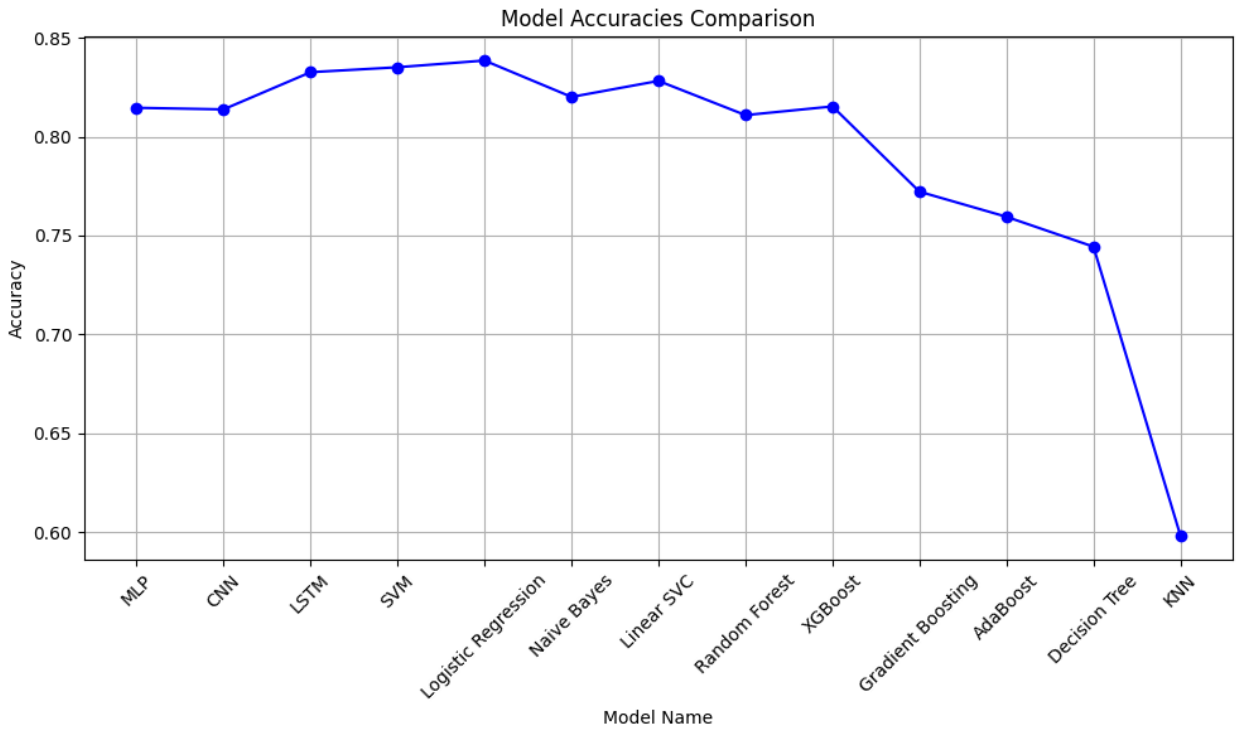
2.5 Results

We built the models used the algorithm mentioned above. Among the models we tried, we tested them using the testset with 11515 pieces of text.

The result looks good and reasonable. Logistic Regression has the top performance with an accuracy of 0.8385, closely followed by SVM and LSTM models, which showcased accuracies of 0.8351 and 0.8326 respectively. Meanwhile, models like MLP, CNN, and XGBoost also performed well, with accuracies above 0.81, suggesting their potential in varied applications.

However, models such as Gradient Boosting, AdaBoost, and Decision Tree lagged behind, with accuracies ranging from 0.7444 to 0.7722, possibly due to overfitting or other limitations. The least effective model in this analysis was KNN, with an accuracy of only 0.5983, indicating it might require extensive parameter tuning. Overall, this evaluation provides insightful guidance for selecting appropriate models based on accuracy and the specific requirements of the task at hand.

Model Name	**Accuracy**
MLP	0.8146
CNN	0.8138
LSTM	0.8326
SVM	0.8351
Logistic Regression	0.8385
Naive Bayes	0.8201
Linear SVC	0.8282
Random Forest	0.8109
XGBoost	0.8153
Gradient Boosting	0.7722
AdaBoost	0.7595
Decision Tree	0.7444
KNN	0.5983



2.6 Model for Helpful and Funny

The dataset also includes the information about the number of up votes and the number of funny votes for each review. So we also train to model to see if a review would be helpful or funny to other people.

Because most of the reviews have few number of votes for the two cases, we divide them into two categories. If the vote number is larger or equal to 5, we would say it is helpful or funny. We use the MLP model, which has a good performance in the previous sentiment model, to predict this value. We get the accuracy of 88.49% for usefulness and 95.27% for funniness.

3 Code Setup

We use Colab to write our code, so they can be run directly online.

First, download data from Kaggle (https://www.kaggle.com/datasets/smeeeow/steam-game-reviews) and store it at some place on Google Drive. Also add the ".ipynb" files to Google Drive.

Second, update the path in the Colab. In "data_processing.ipynb", update the selected path below to point to the dataset and also update the save_path. Then run it.

```
from google.colab import drive
import pandas as pd
import os

drive.mount('/content/drive',force_remount=True)
folder_path = './drive/MyDrive/Colab Notebooks/CS410/data/game_rvw_csvs/'
save_path = './drive/MyDrive/Colab Notebooks/CS410/data/processed_data/'
```

```
Mounted at /content/drive
```

In "data_combination.ipynb", update the selected path below to match the save_path in the previous file and update the output_file to the place that you want to store the "merged.csv" file. Then run it.

```
from google.colab import drive
import pandas as pd
import os

drive.mount('/content/drive',force_remount=True)
folder_path = './drive/MyDrive/Colab Notebooks/CS410/data/processed_data/'
output_file = './drive/MyDrive/Colab Notebooks/CS410/data/merged.csv'
```

```
Mounted at /content/drive
```

In "Steam_review.ipynb", replace the selected path with the path to the "merged.csv" file updated in the last step. We also uploaded the "merged.csv" to Github repository, so you can use that one directly. Finally, run the file.

```
import pandas as pd

# Load the dataset
file_path = './drive/MyDrive/Colab Notebooks/CS410/data/merged.csv'
```

We set up examples to make predictions using one of the trained models as the image below and you can use other models in a similar way.

```
[ ]  # Preprocess the new sentence using the defined preprocess_text function
     text = """
     The game sucks
     """
     new_sentence_to_predict = preprocess_text(text)

     # Transform the preprocessed sentence to TF-IDF vector using the fitted tfidf_vectorizer
     new_sentence_tfidf = tfidf_vectorizer.transform([new_sentence_to_predict])

     # Predict the sentiment using the trained MLP model
     new_sentence_pred = mlp_model.predict(new_sentence_tfidf)

     # Output the prediction (1 for positive, 0 for negative)
     predicted_sentiment = "Positive" if new_sentence_pred[0] == 1 else "Negative"
     predicted_sentiment
```

4 Distribution of Contributions

Data Preprocessing: Yu Zhang, Yutao Zhou
Model Selection, Training and Evaluation: Xinyu Lian, Yu Zhang, Yutao Zhou
Helpful and funny models: Yuankai Wang