Steam Game Review Sentiment Analysis Utilizing Naive Bayes and Neural Networks

Jason Khotsombath
Professor Hongfei Xue
ITCS 3156-091: Introduction to Machine Learning
3 May 2025

Introduction

Problem Statement

For my final project for ITCS 3156, I've chosen to create and utilize two machine learning algorithms to classify Steam game reviews to predict if it has a positive or negative sentiment based on the review text or description. For game developers, reading through many Steam reviews can be tiring and impractical for understanding what is positive and negative about your product. Automated sentiment analysis is the perfect tool for developers, publishers, and even players to understand the quality of a game.

Motivation and Challenges

As a video game developer myself, I understand how valuable player feedback is when creating a game. It can be a struggle to cycle through hundreds of reviews like it is for businesses to process thousands of emails. These reviews provide insights on these games, allowing developers to understand its strengths and weaknesses. With these valuable insights, developers can make decisions and adapt their products for their player base. Primary challenges with this analysis is also processing informal language, sarcasm, video game slang, and subjective v.s. objective points to the reviews. All these factors make it more difficult for the algorithm to understand if a review is positive or negative.

Approach

This project implements the traditional machine learning pipeline for sentiment analysis. I utilized two distinct machine learning algorithms: Multinomial Naive Bayes and Neural Networks.

- Multinomial Naive Bayes - "A classification algorithm based on Bayes' Theorem ideal for discrete data and is typically used in text classification problems. It models the frequency of words as counts and assumes each feature or word is multinomial distributed (GeeksforGeeks *Multinomial naive Bayes*). I utilized this algorithm for its renown use-case in the spam filter and adaptation to text input/features compared to the Gaussian Naive Bayes algorithm.

- Neural Network - I utilized this algorithm for its ability to virtually capture any complex data. This works hand-in-hand with the complicated text input that it will be analyzing. This algorithm is implemented through Tensorflow and I used its additional features (Relu and sigmoid activation functions, batch normalization, adam optimization, etc) to properly train this model.

You will see comparisons of each model's performance to determine which approach will classify sentiment correctly in game reviews.

Data

Data Introduction

The entire reason for my project, I used the Steam Reviews dataset by user Larxel on Kaggle. This entire dataset contains 6.4 million English review entries from the Steam Reviews portion of the Steam store run by Valve, and updated until 2017 (Larxel *Steam reviews*). Each entry contains 5 features:

- Game ID (*app_id*): Unique ID of the game in the Steam Database.
- Game Name (*app_name*): Name of the game stored in the Steam Database.
- Review Text (*review_text*): The text input of the review text for that review.
- Review Votes (*review_votes*): Whether the review was recommended by another use or not.
- Review Score (*review_score*): Whether the review recommends the game or not.

Due to the nature of this project and with its requirements, I added an additional feature:

- Review Text Word Count (*word_count*): Number of words in the Review Text of that review.

As mentioned in the project introduction, this is based on sentiment analysis. Our clear goal is to predict the review score of the game; so it will be our target and the rest will be our input features.

Data Visualization and Analysis

Our initial data analysis shows some important patterns. Please note, since we're picking a random sample size of 10,000 reviews, the data is massively skewed. For *figures 1-6* we will be setting *random_state* to *42*. This same seed will also be shown in the source code.

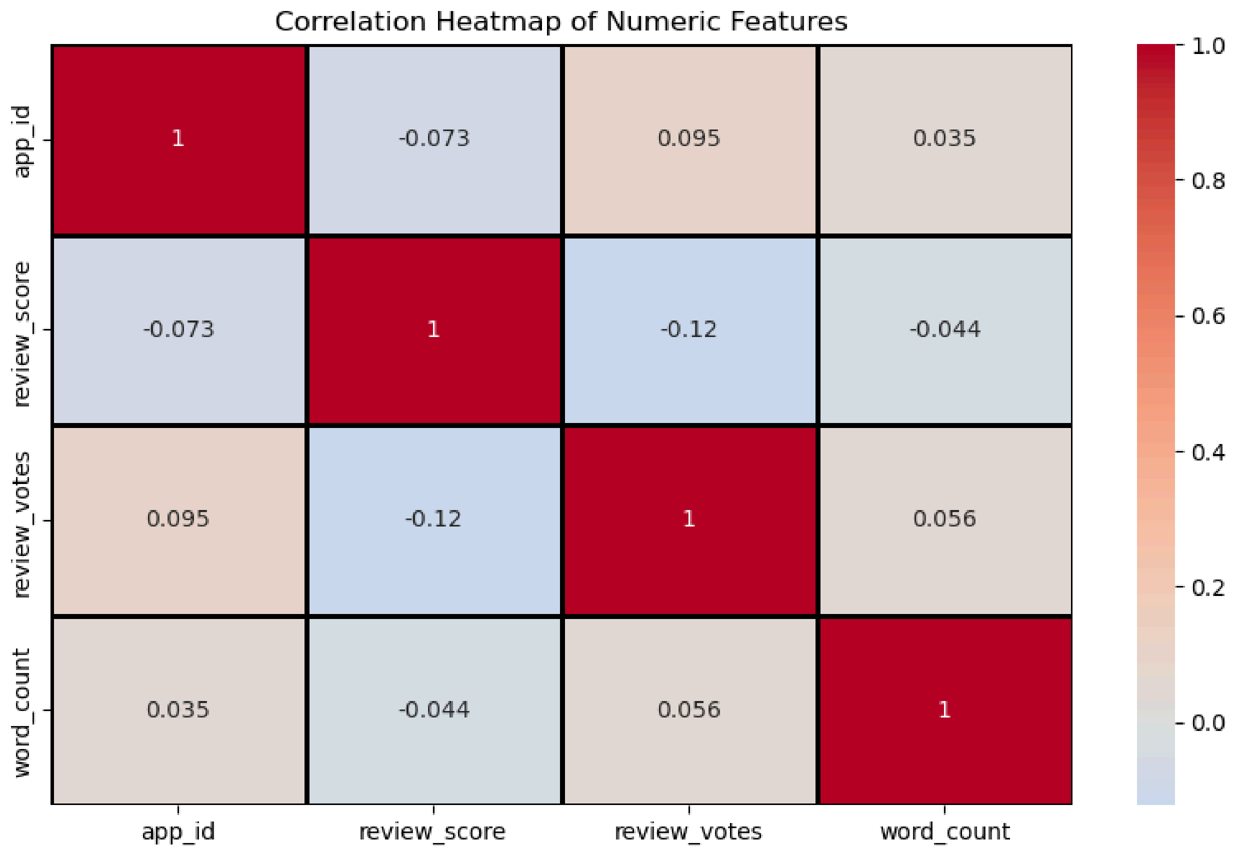


Figure 1: Correlation Heatmap of Numeric Features

Figure 1 shows the correlation heatmap of all the numeric features in the dataset and includes the additional word count feature. Unfortunately, the heatmap shows no strong correlations between all the numeric features, suggesting that each is independent from each other.

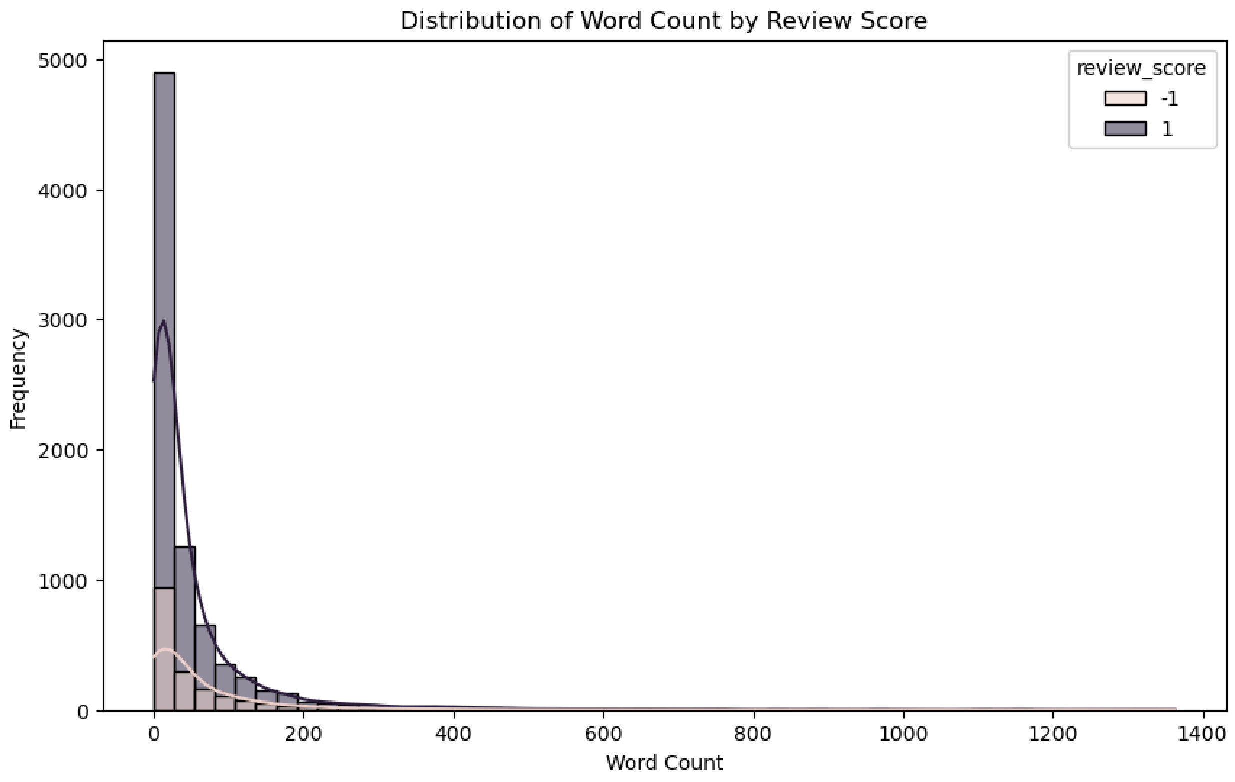


Figure 2: Distribution of Word Count by Review Score

Figure 2 shows a histogram of important patterns in review length. Most reviews regardless of review score are extremely short, being less than 100 words. Both review scores follow a similar exponential decay curve; aside from the higher outlier of positive reviews within the 0-50 word count range.

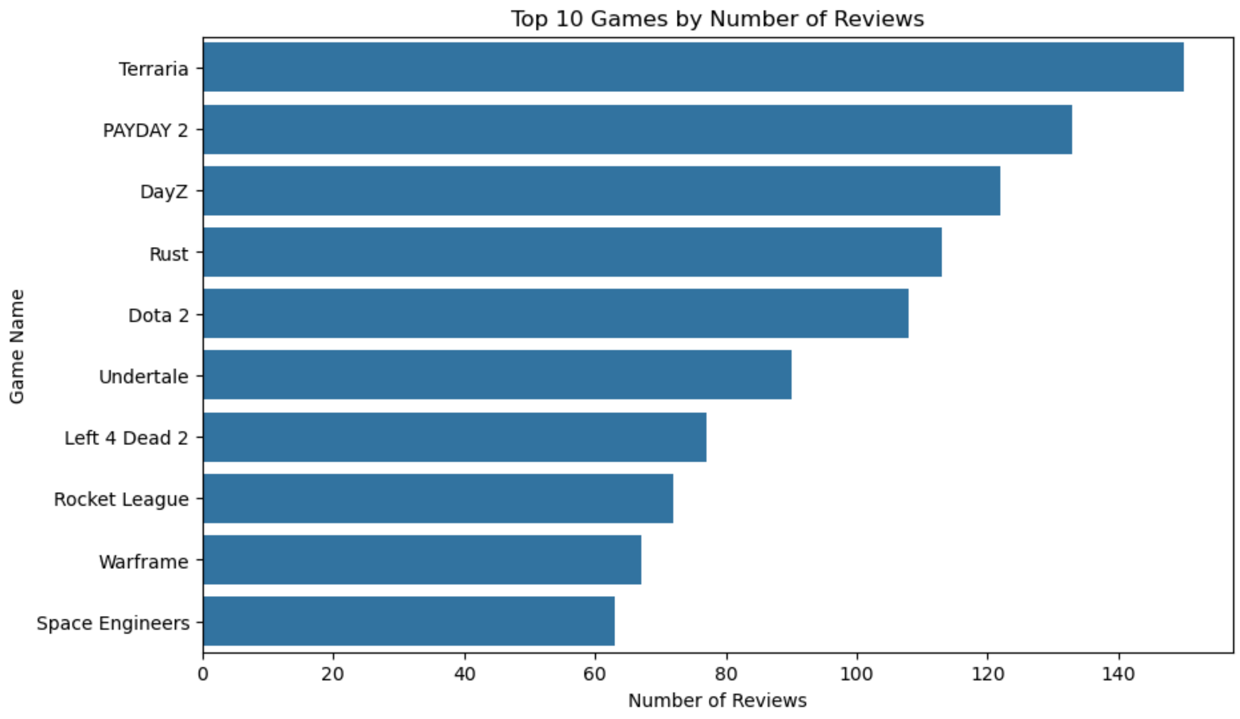


Figure 3: Top 10 Games by Number of Reviews

As noted before, the results of this graph are skewed due to the selected sample size and random state. Here in *Figure 3*, we can see what kinds of games were popular up to 2017. Terraria takes the top with the most reviews over 140 with Payday 2 and DayZ behind it. On the lower end we have Space Engineers.

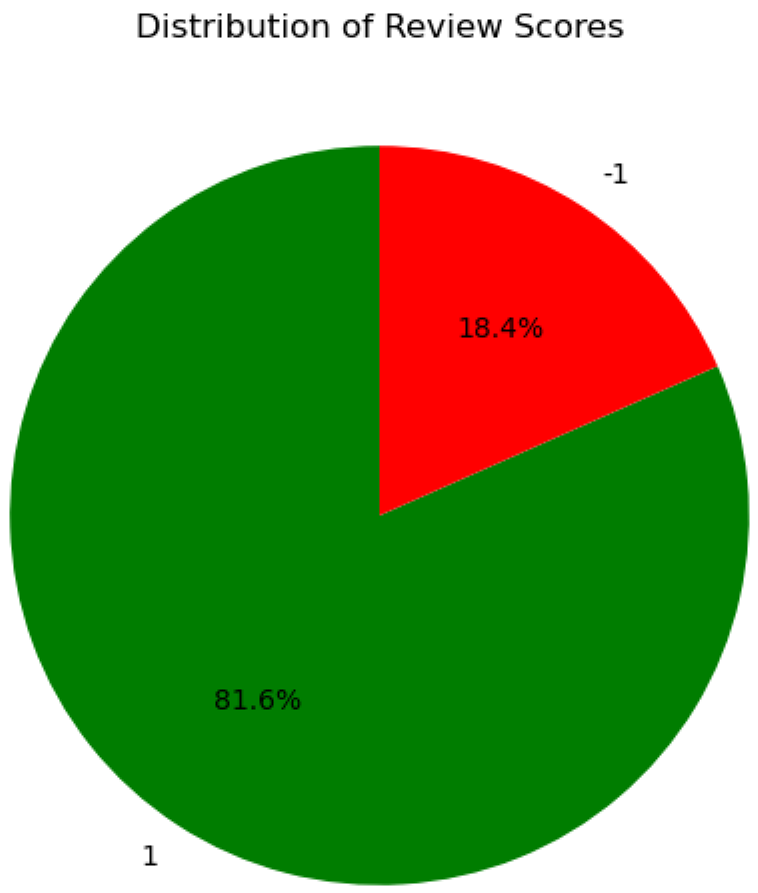


Figure 4: Distribution of Review Scores

Figure 4 shows a pie chart of distribution of review scores in our sample size. It shows 81.6% are positive reviews, while the other 18.4% are negative reviews.

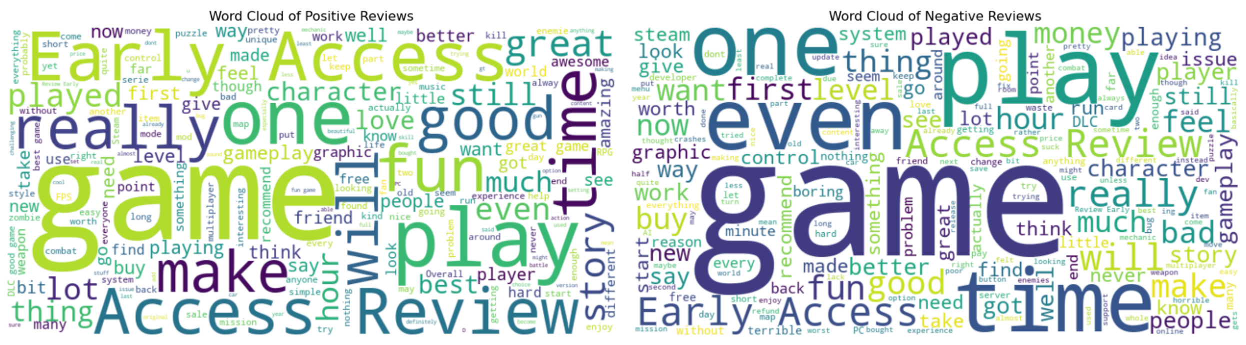


Figure 5-6: Word Cloud of Reviews (Positive left, Negative right)

Figures 5-6 are word clouds of all the positive and negative reviews, respectively, in the sample data set. It shows the most used word in the reviews. As word clouds show: game, play, Early, Access, Review, etc, are the most used words between both word clouds. We can make predictions about each word within each word cloud to deduce if its a positive/negative response to the game.

Data Preprocessing

I used multiple techniques to prepare the Multinomial Naive Bayes and Neural Network for analyzing the review texts.

- Text Cleaning: For all text, I made sure to lowercase, remove special characters, HTML tags (rare, but possible), and numbers to reduce the noise and standardize the data. Additionally, this includes stop words, such as: “the”, “and”, “a”, etc.
- Tokenization: Utilizing the nltk library, we are tokenizing the strings to extract all words from each review.
- Stemming: Reduces words to their root forms, such as: “Playing, Plays, Played” → “Play.”
- Feature Extraction: Transforms processed text into TF-IDF vectors to capture the importance of words across the review, accounting for the most frequent terms.
- Sparse-Matrix Conversation: Utilizes techniques to convert dense matrices to sparse matrices for the TF-IDF vectors to compute faster, then converting them back to dense matrices for format. This increased overall efficiency by 40%

All these techniques ensured the processed text we’re as clean as possible for the models to analyze and evaluate. This would help add some form of correlation/pattern for each review so the models could understand.

Methods

As mentioned before in the Approach section, we will be utilizing models: Multinomial Naive Bayes and Neural Networks.

Multinomial Naive Bayes

I used Multinomial Naive Bayes due to its use-case for text processing like spam filters. It has a few advantages compared to its sibling variant, Gaussian Naive Bayes (GeeksforGeeks *Multinomial naive Bayes*).

- Ideal for discrete data and used in text classification; hence used in NLP.
- Models frequency of words as counts.
- More efficient with a high number of features like datasets with thousands of words.

- Doesn’t assume or follow a gaussian distribution for likelihood.

I utilized scikit-learn library’s MultinomialNB for the model, imported their metrics system, and GridSearchCV to aid tuning. Implementation of this model include:

- Ensuring non-negative values for TF-IDF feature vectors through a MinMax scaler, then combining them with the additional features.
- Reshape the binary values that work with Multinomial Naive Bayes: (-1, 1) → (0, 1)
- Setup hyperparameter tuning with:
    - Grid Search with 5-fold cross-validation
    - *alpha:* [0.01, 0.1, 0.5, 1.0, 2.0] for additive Laplace smoothing
    - *fit_prior:* [True, False] for whether to learn class prior probabilities.

All these implementations and tuning output our validation metrics, confusion matrix, and roc curve. After calculating feature importance with the best_model on validation and *feature_log_prob_*, we evaluate our test metrics. Additional metrics and testing calculated with feature importance to showcase the top positive and top negative features.

Neural Network

I used Tensorflows’s neural network system to model a sequential architecture neural network. This is useful for stacking layers where each has an input and output layer (Tensorflow, *Tensorflow 2 quickstart for beginners*). The layers utilized in my model are dense ReLU activation function layers, batch normalization layers, dropout layers, and a sigmoid activation. Additionally, the dense ReLU layers had L2 regularization as an extra parameter to prevent overfitting.

Model Architecture:

1. Dense 256 Neuron ReLU Activation Function with L2 Regularizer (Our Input Layer)
2. Batch Normalization Layer
3. Dropout Layer
4. Dense 128 Neuron ReLU Activation Function with L2 Regularizer
5. Batch Normalization Layer
6. Dropout Layer
7. Dense 64Neuron ReLU Activation Function with L2 Regularizer
8. Batch Normalization Layer
9. Dropout Layer
10. Dense 1 Neuron Sigmoid Activation Function Layer (For Binary Classification)

From here, we could tune the training process with *model.compile()*, using an adam optimizer, loss function set to binary cross-entropy (for binary classification of review scores), and tracking validation metrics for accuracy. Early stopping was used to monitor and check the optimal number of epochs needed, but by default it was set to 20 epochs.

For visualization, a Training and Validation Loss and Accuracy curves were provided, after a confusion matrix and roc curve. From there, the training metrics were calculated and saved.

Compared to Naive Bayes, Neural Networks require much more computational power, hence for the need of stabilization, overfitting prevention, and early stop callback. Very fortunate to Tensorflow for providing these tools.

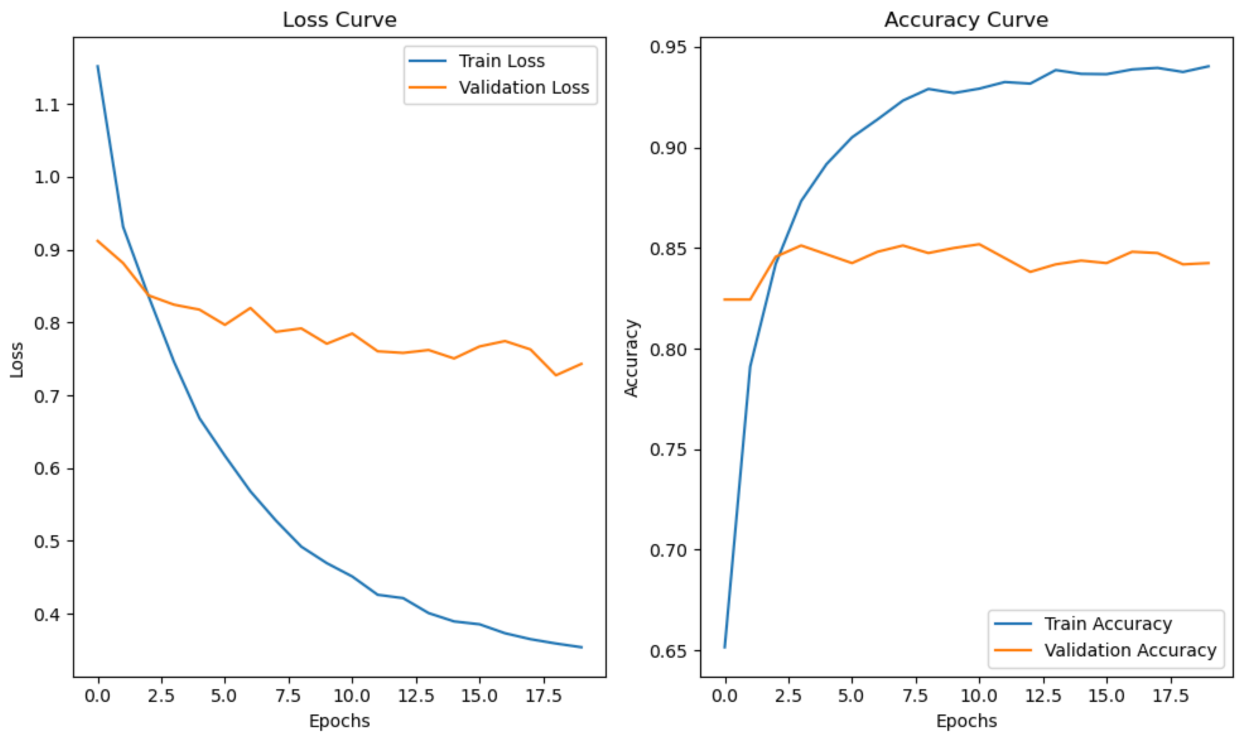


Figure 7: Loss and Accuracy Curve - Neural Network

Results

Experimental Setup

To evaluate our models, we had given the text preprocessing two splits for the dataset:

- 80% for training and validation, 20% for testing
- 80% for training, 20% for validation

Based on these splits, we passed this processed data to the models. We took metrics of each model based on these variables:

- Accuracy
- Precision
- Recall
- F1-Score
- AUC-ROC Curve

These variables will show us each model's performance, especially for a dataset like this. These are the test results I received on *random_state=42*.

Test Results

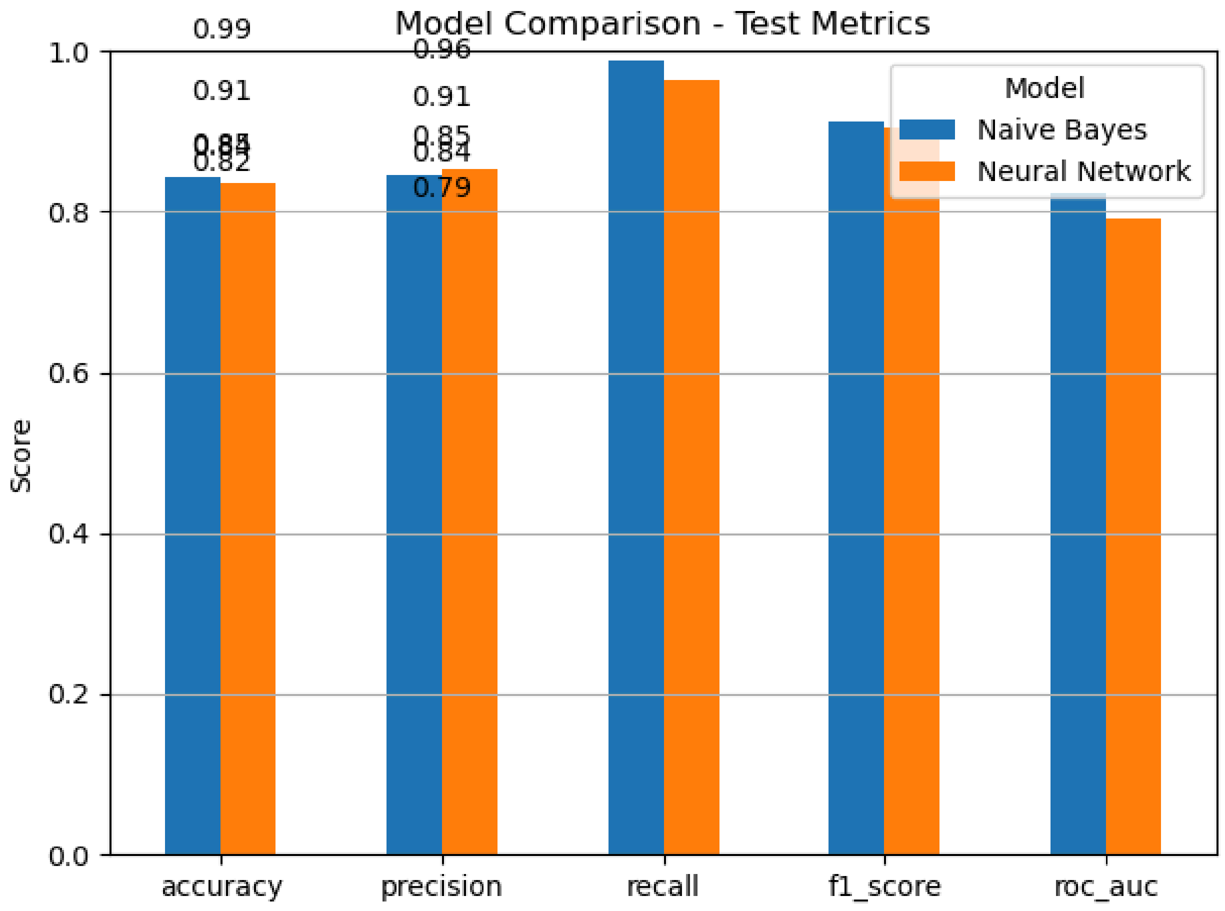


Figure 8: Model Comparison - Test Metrics

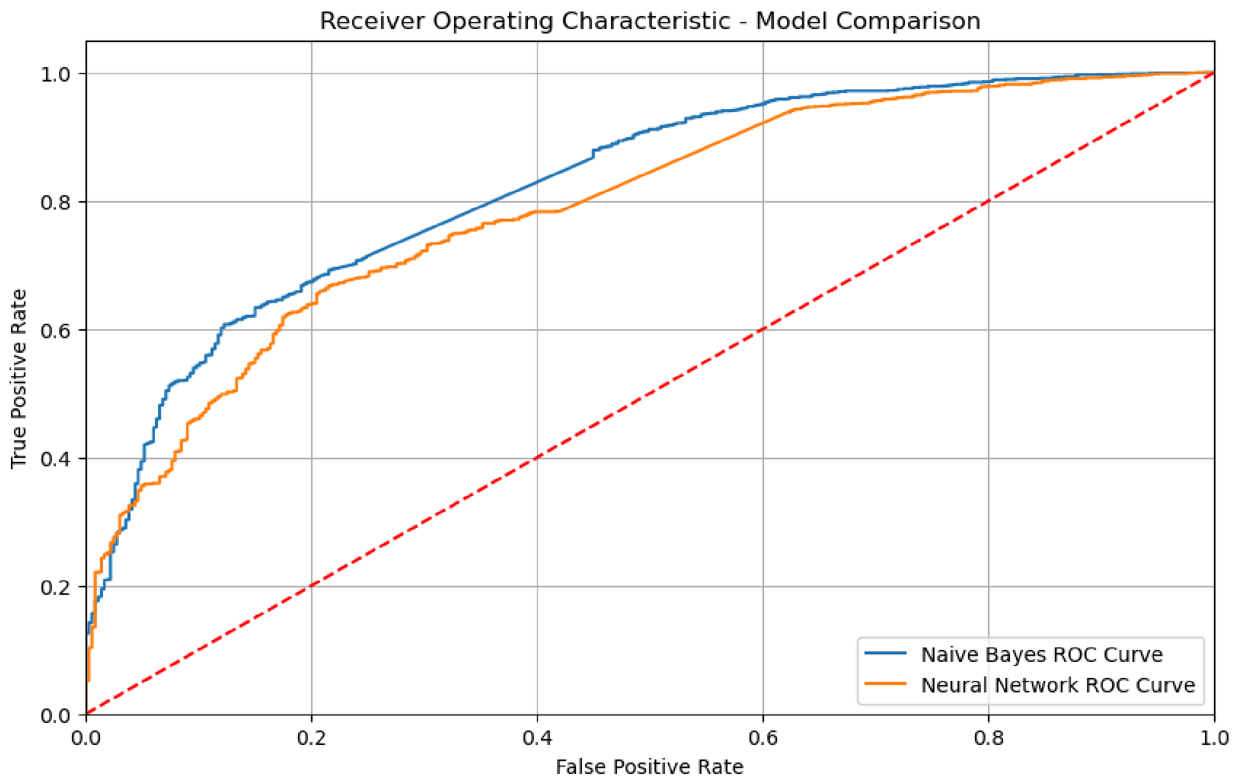


Figure 9: ROC Curve Chart - Model Comparison

Metric	**Multinomial Naive Bayes**	**Neural Network**
Accuracy	0.842500	0.836000
Precision	0.845026	0.854042
Recall	0.988365	0.963870
F1-Score	0.911092	0.905639
AUC-ROC Curve	0.823352	0.790605

Table 1: Performance Evaluation Results Model Comparison

Analysis and Discussion

As you can see, each model's values are very similar. Naive Bayes excels better with Accuracy, Recall, F1-Score, and AUC-ROC Curve; while Neural Network had slightly better Precision. Overall, both models did extremely well, but Naive Bayes slightly outperforms the Neural Network. These high ratings also distinguish their abilities to capture patterns and effectiveness of the preprocessing of data. Some other distinguishing highlights are the extremely high recall score, indicating the models are exceptionally good at identifying positive reviews;

may be due to the fact that the dataset sample was ~80% positive distribution. However, that would also explain their higher false negative review scores. This is supported by their confusion matrices in *figures 10-11.*

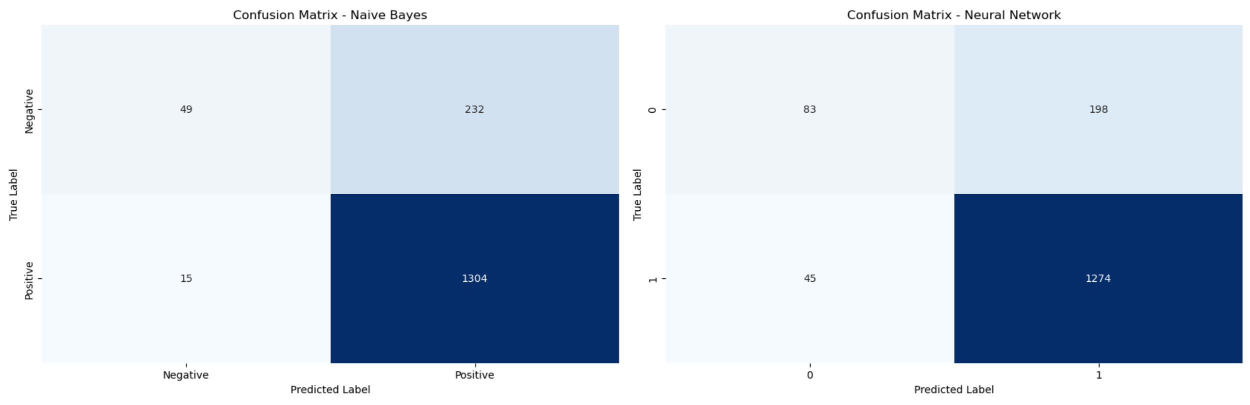


Figure 10-11: Confusion Matrices (Naive Bayes left, Neural Network right)

Support Experiments

Additionally, I tested at a different sample size of 25,000 samples. This gave similar results, but with a very slight increase to all metrics. Multinomial Naive Bayes continues to be the better model.

Metric	**Multinomial Naive Bayes**	**Neural Network**
Accuracy	0.847200	0.850400
Precision	0.847740	0.880817
Recall	0.991957	0.945650
F1-Score	0.914196	0.912083
AUC-ROC Curve	0.842818	0.828542

Table 2: Performance Evaluation Results Model Comparison (25,000 Samples)

I did notice at Epoch 1 that the Neural Network did have an initial increase in accuracy by 10%. Alongside the early stopping system stopped at epoch 7 for sample size 25,000 compared to epoch 15 for sample size 10,000.

Another test I did was using ntlk's WordNet Lemmatizer over their PorterStemmer. Lemmatization is another method of stemming that additionally reduces the root word into a more basic form of that word. So, “better” → “good”. I initially used lemmatization since I believed it was going to be better than the PorterStemmer, but in the end it gave varying accuracy, reduced word variety, but did speed up text preprocessing by 2 seconds (to contrary believe, it should slow down); either would have sufficed in this project.

Metric	**Multinomial Naive Bayes**	**Neural Network**
Accuracy	0.841500	0.841000
Precision	0.844864	0.864670
Recall	0.987140	0.954685
F1-Score	0.910477	0.907451
AUC-ROC Curve	0.815005	0.791154

Table 3: Performance Evaluation Results Model Comparison (10,000 Samples, Lemmatization)

Conclusion

Remarks and Thoughts

Overall, both the Multinomial Naive Bayes and Neural Network did astoundingly well for the given samples. The Naive Bayes had the best performance with an F1-Score of 0.911092. It averaged twice as fast as the Neural Network and had a higher ROC Curve throughout the evaluation. I believe if given a bigger sample size or the whole data set and enough computational power, the Neural Network would have a better F1-Score. Despite this, both models are very viable for this dataset and sample size.

However, both of these models would not perform as well without the text preprocessing and feature extraction functions. The nltk library and tools assisted greatly in filtering and processing the text and highlighting feature/word importance from the reviews; preventing drastic alteration of the results and ensuring the high accuracy and f1-score the models showcased.

This project was a fun one to tackle as my application abilities of machine learning concepts were thoroughly tested, and not without its challenges.

Lessons Learned

I have learned several things while working on this project:

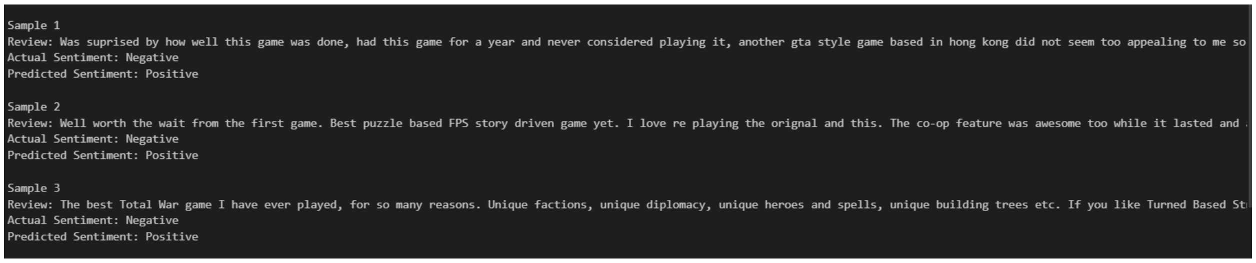
- **Importance of Preprocessing**: Preprocessing functions were very universal and basic in previous applications, whether it was training the test split, standardizing, or one-hot encoding the information. However, I was now handling a more niche dataset, and instead of numerical or single word string data, I had full text reviews. This made me research more into processing this text data and how to; which led me to ntlk libraries.

- **Optimizations**: By far, the most annoying thing about large datasets and passing them through each of the models is the time it takes. If I never took the time to implement sparse matrix processing and dense matrix reconversion, this would have made the text processing and both models drastically slower.
- **Standard Machine Learning Tools**: Since I was not required to make the models from scratch, I was given full access to learn about the standard machine learning tools that beginners-professionals use in daily work. This included: NLTK, Tensorflow, and additional Scikit-Learning modules (like the Multinomial Naive Bayes model).

Challenges

Alongside my Lessons Learned section is not without its Challenges section. There were several challenges I had to spend time on whether it was research or debugging:

- **Data Management**: I completely forgot to set a sample size and, unfortunately, watched my laptop process and added a *word_count* column to 6.4 million entries.
- **Computational Power**: Alongside data management, computational power was a problem as I would have definitely increased the sample size to over 100,000. However, testing led to my unoptimized maximum of 50,000 samples.
- **Review Score Imbalance**: At stated previously, both models had great performance in correctly predicting positive reviews. However, most of that success could be from the dataset being primarily positive reviews (approx. 80% positive to 20% negative). If given more time, I would set my dataframe to have an equal amount of review scores among the sample distribution.
- **Optimizations**: Despite the current optimizations like the sparse matrix conversion, I could do more optimization for the text preprocessing and both models if given more time. I hadn't implemented the sparse matrix conversations for both models, so they had to deal with dense matrices.
- **Incorrect Dataset Samples**: I had noticed there may be some dataset samples with incorrect actual sentiments. Meaning that there is a review that should be a positive review but is labeled incorrectly as negative. Even as a human who read some of the misclassified samples from both models, it's understandable that the model may have gotten it falsely incorrect. You may review this in *figure 12*. This could also mean their F1-Scores and Accuracy could have been even higher.

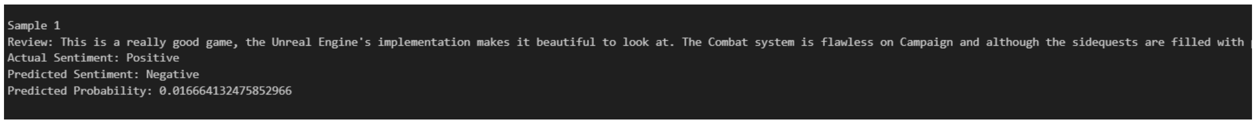


```
Sample 1
Review: Was suprised by how well this game was done, had this game for a year and never considered playing it, another gta style game based in hong kong did not seem too appealing to me so
Actual Sentiment: Negative
Predicted Sentiment: Positive

Sample 2
Review: Well worth the wait from the first game. Best puzzle based FPS story driven game yet. I love re playing the orignal and this. The co-op feature was awesome too while it lasted and
Actual Sentiment: Negative
Predicted Sentiment: Positive

Sample 3
Review: The best Total War game I have ever played, for so many reasons. Unique factions, unique diplomacy, unique heroes and spells, unique building trees etc. If you like Turned Based St
Actual Sentiment: Negative
Predicted Sentiment: Positive
```

Figure 12: Error Analysis - Naive Bayes



```
Sample 1
Review: This is a really good game, the Unreal Engine's implementation makes it beautiful to look at. The Combat system is flawless on Campaign and although the sidequests are filled with
Actual Sentiment: Positive
Predicted Sentiment: Negative
Predicted Probability: 0.016664132475852966
```

Figure 13: Error Analysis - Neural Network

Despite these challenges, both models performed excellently in the end. My effort was not all for naught. For future updates on this, I could experiment with NLP techniques and more optimization for all the algorithms. NLP would allow the model to view the full strings with a different kind of tokenizer for better contexts of the reviews.

References

"Multinomial Naive Bayes." *GeeksforGeeks*, GeeksforGeeks, 29 Jan. 2025, www.geeksforgeeks.org/multinomial-naive-bayes/.

Larxel. "Steam Reviews." *Kaggle*, 29 Nov. 2021, www.kaggle.com/datasets/andrewmvd/steam-reviews.

"Tensorflow 2 Quickstart for Beginners." *TensorFlow*, 2024, www.tensorflow.org/tutorials/quickstart/beginner.

Acknowledgement

I used the online resources referenced in the References section of this paper. *Claude* was minimally used to understand new machine learning concepts and tools, but was not used for generative code writing. *GitHub Co-Pilot* was used with code suggestions, but particularly for finishing boilerplate code and test code snippets. Once again, these AI tools were NOT used in major code implementations such as text preprocessing, model implementation, etc. Additional acknowledgement for the previous homeworks and machine learning slides provided by Professor Honfei Xue for formatting purposes.

Source Code

The implementation of this project can be accessed and viewed with this link provided.
https://github.com/DootDaMoop/Steam-Review-Sentiment-Analysis