# Amazon Movie Review Classification

Adrish Dey

October 28, 2024

## 1 Introduction

In this project, we focus on classifying Amazon movie reviews into one of five ranking categories, from 1 to 5, based on the sentiment and content of the reviews. While the original dataset consists of a substantial 3 million entries, we utilize a representative subset of 100,000 reviews for efficient training and analysis. To achieve high-performance classification, we employ parallelized training techniques alongside Optuna, a powerful hyperparameter optimization framework, to identify the optimal model architecture and hyperparameter configuration. For the final model predictions, we incorporate a majority voting ensemble approach, combining predictions from multiple optimized models to improve robustness and accuracy in classifying review rankings accurately. This approach balances computational efficiency with high predictive accuracy, yielding a streamlined and effective solution for sentiment-based classification.

## 2 Exploratory Data Analysis

In our initial data exploration, we observe a significant class imbalance among the review scores, with certain rankings being overrepresented. This imbalance poses a challenge for training, as models might be biased toward the majority classes, leading to poor performance on minority classes. To address this, we apply SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset by generating synthetic samples for the underrepresented classes.

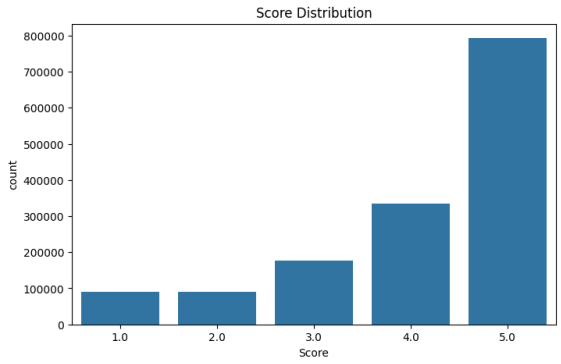
For efficient handling of our large dataset, we leverage GPU-accelerated KMeans clustering from the ‘cuML‘ library as part of the SMOTE process. This allows us to perform oversampling at scale, ensuring a more balanced distribution of review scores in the training set. After applying SMOTE, we visualize the distribution of scores, which now reflects a more balanced dataset, providing a solid foundation for robust model training and improved predictive performance.

## 3 Methods

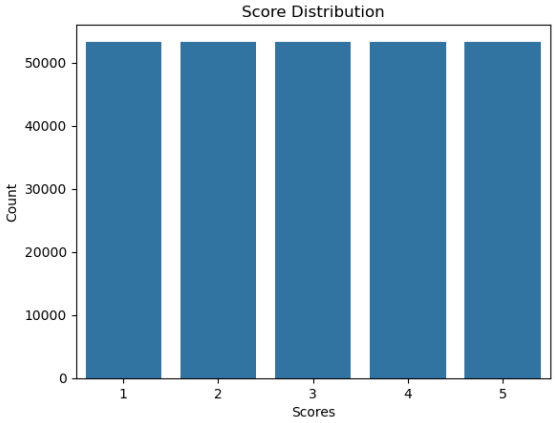
In this study, we focus on utilizing the textual review data as the primary feature for classification. The review text is preprocessed using a two-step vectorization process: first, by applying `CountVectorizer()` to generate a sparse matrix of word counts, and then transforming these counts into Term Frequency (TF) values using the Term Frequency Transformer. This preprocessing pipeline, as recommended in the scikit-learn tutorials, enables the generation of effective feature vectors that capture word frequency while accounting for document length.

To identify the optimal model architecture and hyperparameters, we perform hyperparameter tuning using Optuna in conjunction with Ray for parallelized search. This setup efficiently explores a wide range of parameter configurations and scales well with large datasets. The classifiers explored include Support Vector Classification (SVC) implemented with `SGDClassifier` and hinge loss, $k$-Nearest Neighbors (KNN), Logistic Regression implemented with `SGDClassifier` and log loss, Multinomial Naive Bayes (MultinomialNB) and XGBoost.

For model training, we sample 100,000 representative entries from the full dataset to create a balanced and manageable training subset. To further optimize memory usage and training speed, we employ mini-batch training and the `partial_fit` method for models compatible with incremental learning, specifically `SGDClassifier` and the Naive Bayes classifiers. This approach minimizes memory requirements and improves computational efficiency, enabling us to effectively train models even on large datasets.



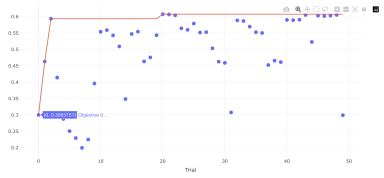

(a) Class frequency distribution before SMOTE, showing a significant imbalance among the score classes with certain rankings being overrepresented. This imbalance can lead to biased model predictions that favor majority classes over minority ones.



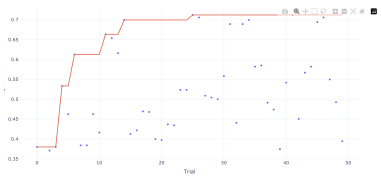

(b) Class frequency distribution after SMOTE, demonstrating a more balanced dataset. SMOTE oversampling has increased the representation of minority classes, which should help improve model performance across all score categories.

Figure 1: Class frequency distribution of Amazon movie review scores before and after applying SMOTE. The SMOTE technique has effectively increased the representation of underrepresented classes, leading to a more balanced dataset that enables more robust training and accurate classification across all review scores.

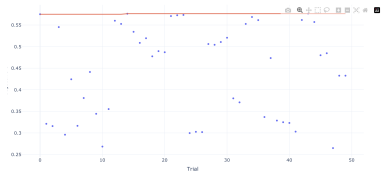
Each classifier is trained and tuned individually, with final model being weighted majority voting, with weights chosen to be the (normalized) accuracy of these estimators over the validation set.



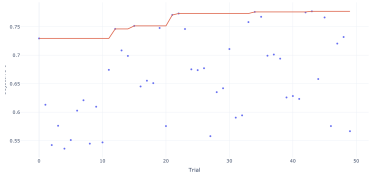
(a) SVC



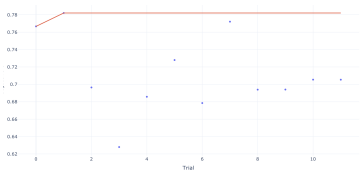
(b) KNN



(c) Logistic Regression



(d) Multinomial NB



(e) XGBoost

Figure 2: Bayesian Hyper Parameter Optimization on different models using Optuna

# Weighted Majority Voting

The ensemble of models is performed by taking a majority vote of the results from individual classifiers. This process is achieved through the following steps:

1. **Conversion to One-Hot Vectors**: Each model's predicted class labels are converted to one-hot vectors. A one-hot vector represents the class as a binary vector where the index of the correct class is marked with a 1, and all other indices are marked with 0.

2. **Weighted Majority Vote**: The accuracy scores of each model are utilized as weights for the majority vote. First, these accuracy scores are normalized using the softmax function. The softmax function is defined as:

Table 1: Accuracy of Classifiers on 20% Validation Set

| Classifier | Accuracy |
|---|---|
| SVM Classifier | 0.61 |
| KNN | 0.71 |
| Logistic Regression | 0.58 |
| Multinomial Naive Bayes | 0.78 |
| XGBoost | 0.782 |

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

where $x_i$ represents the accuracy score of the $i$-th model, and the summation is over all models.

3. **Weighted One-Hot Labels**: The one-hot labels are then weighted with these softmax probabilities. The weighted one-hot vectors are combined using a linear combination, yielding a resultant vector for each class.

4. **Final Label Prediction**: The final predicted label is obtained by taking the argmax of the combined one-hot vectors. This yields the class with the highest weighted score, which is deemed the final prediction.

In summary, the ensemble method effectively aggregates predictions from multiple models, enhancing the overall accuracy and robustness of the classification results. The final accuracy of the ensembled model on the validation set is 83%, demonstrating the improved performance achieved through this approach.

## Parallelization and Code Optimization

In this approach, we leverage several advanced tools and techniques to optimize our computations.

To speed up DataFrame processing, we utilize the CUDF library instead of the traditional Pandas. CUDF is designed for GPU-accelerated operations, allowing for significantly faster manipulation of large datasets. This is particularly beneficial when working with large volumes of data, as it minimizes processing time.

For machine learning tasks, we employ the `cuml` library's `KNeighbors` and TF-IDF implementations, which are optimized for GPU execution. This allows us to perform nearest neighbor searches and text vectorization much more rapidly than CPU-bound alternatives.

We also implement batched Stochastic Gradient Descent (SGD) for classifiers such as Support Vector Machines (SVM) and Logistic Regression for batched data fitting. We do a similar batched fitting for Multinomial Naive Bayes. By processing data in batches, we reduce the memory footprint of our models, enabling us to work with larger amount of data without exceeding available memory.

We use Optuna, a flexible and efficient library for hyperparameter tuning. To further enhance our optimization process, we integrate Ray, which facilitates parallel execution of hyperparameter optimization across multiple processes and GPUs. This parallelization allows us to explore hyperparameter configurations more efficiently, completing each model's optimization in 50 trials.

In addition to these optimizations, we address class imbalance using the Synthetic Minority Over-sampling Technique (SMOTE) from the `imblearn` library, accelerated with CUML's implementation of `KNeighbors`. SMOTE generates synthetic samples for the minority class, thereby improving the balance between classes and enhancing model performance.

In summary, our approach combines GPU acceleration, batched processing, parallel optimization, and techniques for handling class imbalance to create a highly efficient and effective machine learning workflow.

# 4 Final Outcome

Final accuracy obtained on Kaggle: 56.18%