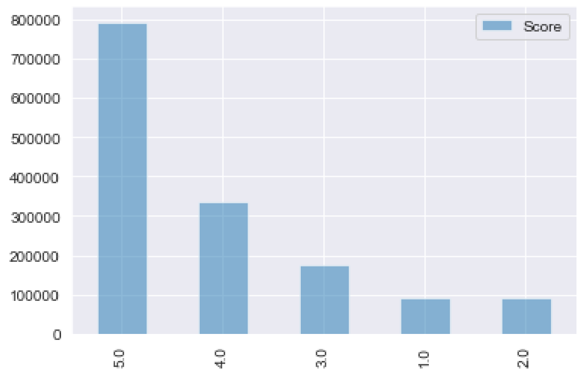
Nick Galis
CS506

Midterm Writeup

The goal of this project is to predict star ratings from Amazon Movie Reviews using features extracted from the reviews dataset. The challenge is to develop a machine learning model that can effectively capture the patterns in the data to predict star ratings as accurately as possible. In this writeup, we will detail the approach taken, including feature engineering, model selection, hyperparameter tuning, and evaluation metrics. Additionally, we will discuss specific techniques employed to improve the model's performance.

The code begins by loading and examining the training and test datasets. The cell in question uses the pandas library to read the testing and training CSV files, which are then stored in the variables trainingSet and testingSet. This is merely to set up the proper data structures that will be used to train and test the model in later stages. It also provides the graph seen at the right of the page, providing the distributions of scores.



This displays that the number of ratings decreases as the star rating goes down. Imbalanced classes can lead to a model that tends to predict the majority class (5-star) more often, potentially sacrificing accuracy on lower ratings (1-4 stars).

After setting up all the data into DataFrames, we then move on to feature engineering. One of the “tricks” I implemented to increase the accuracy of the model is using sentiment analysis on the text of the reviews. Implemented in the code is a function calculate_sentiment is defined to compute the sentiment score of the given text, returning the compound score. The compound score is a single, normalized score that summarizes the sentiment of the text. The sentiment analysis is applied to the 'Text' column in the trainingSet, adding a new column 'sentiment' to store these scores. Reviews often contain subjective opinions expressed through natural language. Sentiment analysis helps in quantifying these opinions as positive, negative, or neutral sentiment scores. For instance, a highly positive review will have a high positive sentiment score, while a very negative review will have a lower or negative sentiment score. Thus, it will be easy to predict the score of a given review if we can predict its sentiment accurately. In addition to sentiment, we also engineer other features including:

- Helpfulness
    - Calculation: Ratio of HelpfulnessNumerator to HelpfulnessDenominator.
    - Purpose: Indicates the percentage of users who found the review helpful.
- Review Length

    - Calculation: Number of words in the review text.
    - Purpose: Longer reviews may have more detailed opinions, which could correlate with what score the user gives.
- Review Year
    - Calculation: Extracts the year from the UNIX timestamp of the review.
    - Purpose: Trends and fads might influence how customers feel and thus what scores they give.

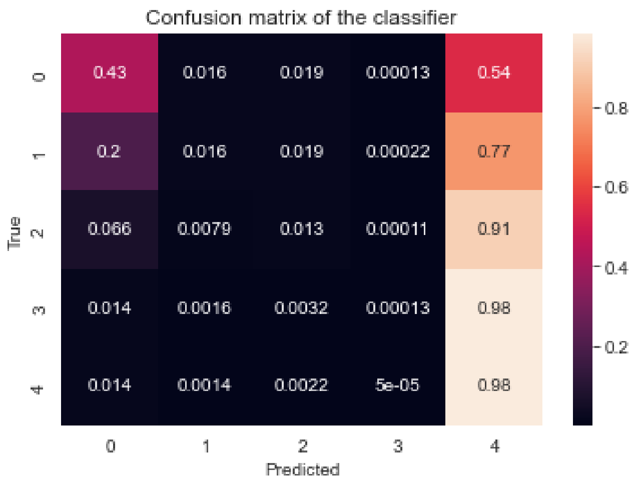
These additional features provide more context and information and improve the model's ability to predict review scores accurately. The code checks if feature-extracted datasets already exist. If they do, it reads them; otherwise, it performs feature engineering and saves the results.

We then do some preliminary processing of the data before defining our model. Merges the trainingSet with the testingSet on the Id column to include the new features in the submission set. We first split the data, ensuring that the model is evaluated on data it hasn't seen during training. This helps in assessing the model's generalization capability to unseen data. Additionally, we remove the target column 'Score' from the training features, so the model is only trained on the other features. The model then saves the resulting DataFrames X_train and X_submission to CSV files for use later in the predictive process. The train_test_split function from scikit-learn is used to split the data such that 25% of the data is used as the testing set, and 75% remains as the training set.

Now, we can begin analyzing the technical details of the model. In the code, I decided to use a Random Forest Classifier on the training set. Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes or mean prediction of the individual trees. After the model is initialized, a parameter grid param_grid is specified for hyperparameter tuning. This includes different values for the number of trees in the forest (n_estimators), the maximum depth of the trees (max_depth), and the minimum number of samples required to split an internal node (min_samples_split). From here, we move onto another “trick” that I implemented to obtain better accuracy with my model: a Stratified K-Fold Cross-Validation. This was used to ensure that each fold maintains the same proportion of classes. This helps the model generalize better by exposing it to different subsets of the data while preserving class distribution. Using SKF ensures that the model evaluation process leads to more reliable insights and better-performing models, especially when dealing with the imbalance of ratings we saw visualized earlier.

Grid Search is a method for hyperparameter tuning that evaluates a model for each combination of hyperparameter values specified in a grid. It aims to find the optimal hyperparameters that yield the best performance on the validation set. First, a grid of hyperparameters is defined with possible values. For each combination of hyperparameters, cross-validation is performed on the training set to evaluate its performance. After each

combination of values is evaluated systematically, the combination yielding the best performance (e.g., highest accuracy) is selected as the optimal set of hyperparameters.
It evaluates all possible combinations, ensuring that the true optimal set of hyperparameters can be found (within the confines of the grid), and due to this exhaustive nature I chose it to tune my hyperparameters.



Let us now analyze the results. The model is generally good at correctly predicting 4- and 5-star reviews, as indicated by the higher values along the diagonal for these classes. This is likely because these classes are more common (as seen in the previous graph), so the model has more data to learn from. For the lower star ratings (especially 1, 2, and 3 stars), the model has a much harder time distinguishing correctly. This might be due to the fact that these classes are underrepresented, making it difficult for the model to accurately predict them. The high concentration of predictions for 5-star ratings is likely a direct result of the imbalanced dataset, as the model seems biased towards predicting the 5-star class. This is evident from the high false positive rate in the 5-star column, where lower ratings are misclassified as 5-star ratings.

The model achieved an overall accuracy of 55%, indicating it correctly predicted the star ratings just over half the time. The confusion matrix shows a strong bias toward predicting 5-star ratings, likely due to the class imbalance in the dataset, where 5-star reviews are far more frequent than other ratings. The model performs reasonably well on 4- and 5-star ratings but struggles significantly with lower ratings (1, 2, and 3 stars), often misclassifying them as higher ratings. This misclassification is most evident for 1-star reviews, which are frequently predicted as 5 stars. The high accuracy for the 5-star class suggests the model has overfit to this majority class, making it less effective at capturing nuances in the lower ratings. Overall, while the model captures general patterns in high ratings, it needs further adjustments—such as class weighting, sampling techniques, or alternative evaluation metrics—to improve performance on the minority classes and provide more balanced predictions across all star ratings.