Predicting Movie Review Scores with Machine Learning Models

Edaad Azman

U38459100

Boston University

CS506 Data Science Tools and Applications

Prof. Lance Galletti

October 28, 2024

Introduction

The primary objective of this project was to predict movie review scores accurately using machine learning techniques. Using a dataset containing both textual reviews and associated metadata, I aimed to extract meaningful features and build robust models to achieve high accuracy. Throughout the project, I faced various challenges, from handling large datasets to optimizing model performance, all while applying concepts learned in our CS506 class.

Feature Engineering

Understanding the Data

At first, I looked at the dataset to understand the structure and identify the most important features. The dataset included several columns, such as HelpfulnessNumerator, HelpfulnessDenominator, Time, Summary, Text, ProductId, UserId, and Score. The first thing I thought of doing was to transform categorical data into numerical formats so its suitable for modeling. Then I added several new features to improve the predictive capabilities of the model. I first calculated the helpfulness ration using the numerator and denominator to measure the helpfulness of a review. I also added temporal features by extracting Review_Year, Review_Month, and Review_Day so I can capture those patterns in reviews. I then realized that the verbosity of the reviews could also help with improving the accuracy so I computed character length of words in Summary and Text. And then I just also calculated the number of words in both of those which helped with assessing the detail level provided in each review. Finally, to understand the popularity or activity associated with each product and user, I applied frequency encoding to these categorical features, creating ProductFreq and UserFreq.

Sampling Strategy

Since the given dataset was very large, processing the entire dataset while running it multiple times through different iterations was taking a very long time. To solve this, I made the decision to sample 10% of the training data. This sampling not only reduced processing times but also maintained a good representation of the distribution of the target variable.

Data Preprocessing

Text Cleaning and Lemmatization

To prepare the Combined_Text for vectorization I removed non-alphabetic characters and converted all text to lowercase to make sure the data was uniform. I used NLTK's predefined stop words list to eliminate common words that do not contribute meaningful information to the analysis. Applying WordNet Lemmatizer, I reduced words to their base forms to help in capturing the essence of the text without redundancy.

Sentiment Analysis

Sentiment analysis added useful features (Polarity and Subjectivity) derived from the Combined_Text field of each review, which were then used as input features for the XGBoost

model. The addition of sentiment data as features allowed the model to incorporate not only numeric data but also information about text sentiment. This enhances the model, leading to more accurate predictions on the test set.

Feature Vectorization and Encoding

TF-IDF Vectorization

I used TF-IDF Vectorization to transform textual data into numerical format with Max Features set to 10,000 to capture a broad range of significant terms while managing computational load. N-Gram Range was configured to (1,2) to include both unigrams and bigrams, capturing both individual words and common word pairs.

Combining Features

Post vectorization, I merged the TF-IDF matrix with the numerical features using hstack to form a comprehensive feature set (X_train_combined for training and X_test_combined for testing). This integration made sure that both textual and numerical information was leveraged by the models.

Model Building and Evaluation

Models

I experimented with multiple models to identify the best-performing one:

K-Nearest Neighbors: The starter code employed the KNN algorithm as the primary classification model but it wasn't ideal because of its computational inefficiency and sensitivity to irrelevant features.

Logistic Regression: This served as the baseline model. It is a statistical method for binary and multi-class classification problems. Initial accuracy on the validation set was 54.8%.

Multinomial Naive Bayes: Trained next, achieving an accuracy of 45.2%. This model underperformed compared to Logistic Regression, possibly due to its assumption of feature independence, which doesn't hold true for text data.

Random Forest Classifier: Despite its robustness, the model only achieved 40.0% accuracy, indicating potential overfitting or inefficiency in handling high-dimensional data.

Hyperparameter Tuning

Given that Logistic Regression had the performance, I focused on optimizing its parameters using GridSearchCV. The hyperparameters tuned included Regularization Strength and Solver. Post-tuning, Logistic Regression's accuracy improved to ~61%, demonstrating the efficiency of hyperparameter tuning.

XGBoost

Seeking further performance gains, I transitioned to XGBoost, a powerful gradient boosting framework known for its efficiency and handling of complex data patterns. The model achieved

an accuracy of ~62%. This slight improvement justified the switch, highlighting XGBoost's ability to capture intricate relationships within the data.

Cross-Validation
I was worried that because I used a small sample size to train my model, it could be vulnerable to overfitting and that could be very problematic. Therefore, to ensure the generalizability of the model, I used cross-validation before generating the submission file. This basically assesses the model's performance across different subsets of the data to avoid the possibility of overfitting. This result ensures that the model's performance is robust and reinforces the results of the

Challenges Faced

One of the significant challenges was the extensive processing time required for feature engineering and model training, especially given the dataset's size. To address this I opted to use a 10% sample of the training data, significantly reducing computation times while maintaining data representativeness. I also implemented progress bars using tqdm to monitor long-running processes, enhancing transparency and allowing for better time management. Another challenge I faced was determining which features to retain which were critical to prevent overfitting. To address that I focused on features that significantly impacted model performance, such as the Helpfulness ratio and temporal features, while discarding less impactful ones. I also reduced the number of TF-IDF features to manage computational efficiency.

Conclusion

Through proper feature engineering, strategic sampling, and rigorous model evaluation, I successfully developed a machine learning pipeline capable of predicting movie scores with an accuracy of ~62% using XGBoost. Focusing on key features like the helpfulness ratio and temporal components, combined with effective handling of class imbalance and hyperparameter tuning, significantly enhanced the model's performance. This project not only reinforced the concepts learned in our class but also provided practical insights into real-world data challenges and the iterative nature of machine learning model development. Future work could explore more advanced models, such as deep learning approaches, neural networks, or incorporate additional features to further boost accuracy.

References

Banerjee, Prashant. "XGBoost + k-fold CV + Feature Importance." *Kaggle*, 2020, https://www.kaggle.com/code/prashant111/xgboost-k-fold-cv-feature-importance. Accessed 28 October 2024.

Ganesan, Kavita. "How to Use Tfidftransformer & Tfidfvectorizer - A Short Tutorial." *Kavita Ganesan*, https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/. Accessed 28 October 2024.

Guide, Step. "TextBlob | Making Natural Language Processing easy with TextBlob." *Analytics Vidhya*, 14 October 2024, https://www.analyticsvidhya.com/blog/2021/10/making-natural-language-processing-easy-with-textblob/. Accessed 28 October 2024.

Jain, Sandeep. "Removing stop words with NLTK in Python." *GeeksforGeeks*, 3 January 2024, https://www.geeksforgeeks.org/removing-stop-words-nltk-python/. Accessed 28 October 2024.

"LogisticRegression — scikit-learn 1.5.2 documentation." *Scikit-learn*, https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed 28 October 2024.

Medium, Anurag, and kevin turcios. "TF-IDF Vectorizer Explained. The Term Frequency-Inverse Document… | by Anurag." *Medium*, 30 April 2024, https://medium.com/@gusainanurag58/tf-idf-vectorizer-explained-373b3f07d23b. Accessed 28 October 2024.

"Sample usage for wordnet." *NLTK*, https://www.nltk.org/howto/wordnet.html. Accessed 28 October 2024.

"Tutorial: Quickstart — TextBlob 0.18.0.post0 documentation." *TextBlob*, https://textblob.readthedocs.io/en/dev/quickstart.html. Accessed 28 October 2024.

"What is Hyperparameter Tuning? - Hyperparameter Tuning Methods Explained." *AWS*, https://aws.amazon.com/what-is/hyperparameter-tuning/. Accessed 28 October 2024.

"What Is Logistic Regression?" *IBM*, https://www.ibm.com/topics/logistic-regression. Accessed 28 October 2024.

"XGBoost Documentation¶." *XGBoost Documentation — xgboost 2.1.1 documentation*, https://xgboost.readthedocs.io/en/stable/. Accessed 28 October 2024.

"XGBoost – What Is It and Why Does It Matter?" *NVIDIA*, https://www.nvidia.com/en-us/glossary/xgboost/. Accessed 28 October 2024.