Movie Score Prediction

Xulun Huang

Introduction

This project focuses on building a supervised machine learning model to predict movie scores from 1 to 5 based on different user reviews. The training dataset consists of about 1.7 million reviews, and the test dataset contains about 212,000 reviews with only IDs and an empty Score column.

I chose Logistic Regression as the classifier due to its efficiency, simplicity, and interpretability, particularly for large-scale multiclass problems. The reviews are analyzed using TF-IDF vectorization for textual features, combined with additional numerical features such as helpfulness metrics and text lengths. I implemented a streamlined data preprocessing pipeline to ensure efficient and consistent handling of both training and test datasets.

Part 1. Feature Extraction

Text-Based Feature Extraction Using TF-IDF

Each review consists of a summary and detailed text, which are concatenated along with the ProductId into a single feature called Text_information. I find that ProductId helped the model learn movie-specific patterns, boosting performance. For text processing, TF-IDF (Term Frequency-Inverse Document Frequency) vectorization is applied to transform the text data into numerical features.

During implementation, I have chosen the following parameters:

```
TfidfVectorizer(

            max_features=20000,
            stop_words='english',
            ngram_range=(1,2),
            min_df=5
        )
```

Max feature is set to 20000 to capture enough features to train the model, stop_words here removes useless and repetitive words that don’t help with the prediction, ngram_range=(1,2) means that the features are able to contain at most two words, which would be helpful to capture more meanings that take two words like ‘highly recommended’. Originally, I also used nltk to get rid of those irrelevant words and tokens as part of the preprocessing, only to find that it was time consuming and didn’t do much to improve the final results, therefore the part was removed.

This process basically produces a matrix of weighted word features, enabling the model to focus on terms that are meaningful for sentiment prediction, such as "excellent" or "disappointing".

Numerical Feature Engineering

At first, I only trained on all the text information without using the numerical features, because I don't see a clear link between these numbers with the movie scores. Later on, I added the following numerical features and implemented an add_all_features function and the accuracy was further enhanced to my surprise.

1. Helpfulness: By dividing HelpfulnessNumerator with HelpfulnessDenominator, I obtained the Helpness Metrics. Reviews without any ratings are assigned a helpfulness score of 0 to avoid division errors.
2. Year of Review:
   Timestamps from the Time column are converted to year values, capturing temporal trends in review behavior.
3. LenText and LenSummary:
   Length of text and summary is also included, they measure the character count of the detailed review and summary, reflecting the level of detail provided by the reviewer.

Because Helpfulness here is a fraction number from 0 to 1 but year and text length numbers are way larger, therefore all numerical features are scaled using StandardScaler to ensure uniform contribution during model training.

Part 2. Model Implementation, Training and Testing

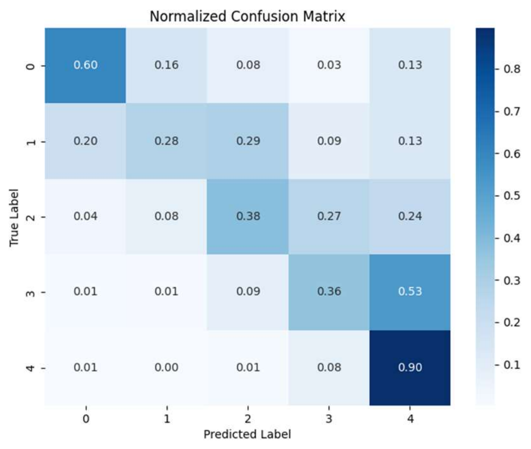
I used scikit-learn's Pipeline and ColumnTransformer to streamline the preprocessing and modeling steps. This approach allowed us to avoid manual feature stacking and ensured consistent handling of both training and test data.

Pipeline Components:

1. TfidfVectorizer for the allText feature.
2. StandardScaler for numerical features: Helpfulness, Year, Text Length, and Summary Length.
3. LogisticRegression as the classifier with max_iter=1000 to ensure convergence.

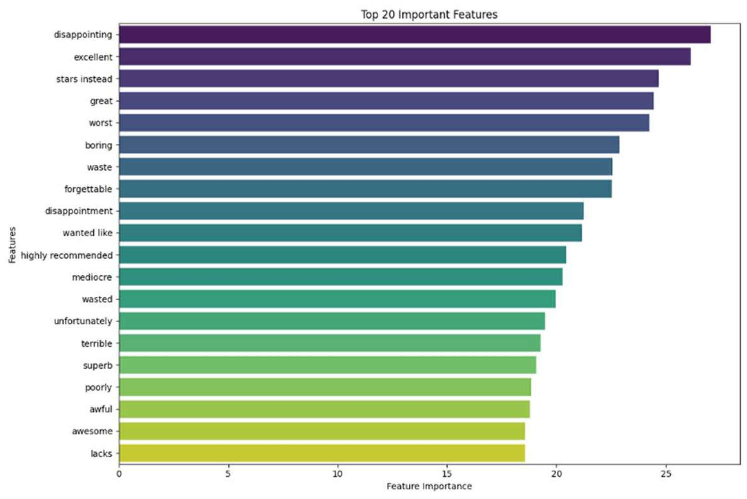
Originally, I tried to use SVM as the classifier, however it seems to take too long to converge and does not give the best result. Different solvers are also tried for the logistical regression, such as saga, but the result is not as good as using the default solver lbfgs, which has a relatively fast convergence and delivers solid results. I also removed the class_weight="balanced" parameter to allow the model to rely more on meaningful patterns rather than artificially adjusted weights, further enhancing performance.

The results are summarized by two graphs, first is the confusion matrix:



As you can see the prediction for score=5 is quite good. However, it occasionally confuses adjacent scores, such as predicting 4 instead of 5 or 3 instead of 4. This is expected since the distinction between adjacent scores can be subtle and subjective.

Second is the Top 20 TF-IDF Features:



The top 20 TF-IDF features were extracted based on the absolute values of their coefficients in the Logistic Regression model. These features provide insights into the terms that most strongly influenced the sentiment predictions.

Positive Indicators: "excellent", "great", "highly recommended", "awesome"

Negative Indicators: "disappointing", "waste", "poorly", "terrible"

These insights validate the model's ability to capture meaningful patterns in the reviews, with terms like "excellent" indicating higher scores and "disappointing" indicating lower ones.