CS506 Midterm: Predicting Amazon Movie Ratings

Jaden Hsiao

Overview

The goal of this project is to predict Amazon movie ratings based on customer reviews. I have to build a machine learning model to analyze the text of each review and predict the score someone might give. To help the model understand and work with words, I used a method called TF-IDF (Term Frequency-Inverse Document Frequency) to turn text into numbers, and then used logistic regression as the model. I also tried some specific techniques to improve how well it worked, focusing on patterns I noticed in the data.

1. Preparing the Data

The dataset had two main text columns for each review: a “Summary” (a short description) and “Text” (a longer review). I did two main things to prepare the data:

- **Combining Text Fields**: Since both “Summary” and “Text” were important for understanding each review, I combined them into a new field called “Combined_Text”. This allowed the model to look at all the review content in one place and get a fuller picture of what the reviewer thought.
- **Handling Missing Data**: Some reviews were missing either the “Summary” or “Text”. To keep things simple, I filled these missing values with an empty string before combining the columns. For the target variable, “Score”, I removed rows that were missing scores so the model would only learn from complete data.

Data Pre-Process

1. **Combining Summary and Text**: I assumed that combining these would give the model more useful information and improve accuracy.
2. **Using Only Complete Scores**: By removing rows with missing scores, I assumed that missing data wouldn’t add helpful information for predictions.

2. Feature Extraction with TF-IDF

To prepare the text for the model, I used TF-IDF. This basically helps turn words into numbers based on how important they are. TF-IDF looks at each word's frequency in a review and compares it to its frequency across all reviews. I used TF-IDF with these parameters:

- **Max Features**: I limited the number of words (or "features") the model could focus on to 1,000 common words. This helps the model focus on important words without getting overwhelmed by rare words that might not be useful for predicting scores.
- **Removing Stop Words**: Words like "the," "and," and "of" don't carry much meaning for predicting scores, so I removed them. This cleaned up the text data and made it easier for the model to focus on important words.

Observations

During testing, I noticed that keeping a moderate number of features (1,000 words) performed better than including too many rare words, which didn’t contribute much to the score predictions. Limiting words like this helped the model focus on the most impactful terms in the reviews.

3. Building and Tuning the Model

I built a pipeline to combine the TF-IDF vectorizer and a logistic regression model. A pipeline is a way to streamline each step, so the model could go from raw text to predictions in one process.

Logistic Regression

Logistic regression is a straightforward and powerful model that works well with text data. It’s good at finding patterns in words and is easy to adjust for better accuracy. It fits the project because we want to predict scores based on text, and logistic regression can handle this type of task without being overly complex. (Saving a lot of run time for me)

Parameter Tuning with Grid Search

To find the best version of the model, I used GridSearchCV to test different settings. Grid Search also just made it easier to find the best parameters without manual trial and error. I tried adjusting:

1. **TF-IDF Features**: I tested 500, 1,000, and 1,500 maximum features to see what gave the best results.
2. **Regularization (C)**: This setting controls how flexible the model is. I tried different values to balance accuracy and overfitting.

Observations

Grid search showed that using 1,000 max features and a regularization value of C=10 gave the best results. Higher C values made the model overfit (too specific to the training data), while lower values underfit (too general).

4. Model Evaluation and Results

The best model was tested on a separate set of data to see how well it worked. Here's what I found:

- **Accuracy**: The model achieved an accuracy score of around 0.63, meaning it learned patterns in the reviews pretty well.
- **Precision and Recall**: The model performed best on scores with more examples in the dataset. Scores with fewer examples had slightly lower precision, so there's room to improve there.

Observations and Next Steps

One interesting thing I noticed was that short reviews were sometimes harder for the model to classify. This might be improved by using different feature extraction methods or a more complex model that can handle these shorter texts.

Final Predictions and Submission

After fine-tuning, I used the best version of the model to make predictions on the test set and saved the results in a submission file to upload into Kaggle.

Other Methods Tested

Before I decided to use Logistic Regression and Grid Search for my model, I tested out other machine learning models to see how they performed. I played around with Random Forests and XGBoost to see if they perform well on the data. After testing the two methods on basic parameters, the accuracy was not as good as logistic regression. Therefore, I decided to fine tune logistic regression and use it as my main machine learning model.