Predicting Amazon Movie Review Ratings

By Bryan Teoh

1. Introduction

This project focused on developing a model to predict Amazon movie review ratings on a 1-5 scale. The main challenge involved processing a large dataset of reviews to extract meaningful features for accurate prediction. The project required balancing multiple factors: handling text data effectively, managing uneven rating distributions, and developing a reliable prediction model. This report details the methodical approach taken, from initial data analysis through feature engineering to final model implementation.

2. Data Exploration and Strategic Insights

The initial analysis revealed an uneven distribution of ratings, with significantly more 4 and 5-star reviews than lower ratings. I investigated the data further to see if there were any relationships between the existing features and the score:

1. **Text Patterns in Ratings:**
    - One-star reviews: Common words included ['bad', 'boring', "don't", 'was', 'one', 'waste', 'worst', 'no', 'film', 'time']
    - Three-star reviews: Mixed terminology like ['good', 'great', 'was', 'you', 'bad', 'just', 'ok', 'better', "it's", 'too']
    - Five-star reviews: Consistent positive words like ['great', 'best', 'love', 'good', 'one', 'classic', 'my', 'film', 'excellent', 'you']
    - Two and four star reviews shared words with the other star reviews so this feature was mostly helpful for one and five star reviews
2. **Review Length Patterns:**
    - Extreme ratings (1 and 5 stars) typically had longer reviews
    - Three-star reviews tended to be shorter
    - Summary length varied less than full review length
3. **Time-Based Analysis:** After converting timestamps to years and months, the data showed:
    - Consistent rating distributions across different time periods
    - No significant seasonal patterns
    - Minimal time influence on ratings

3. Feature Engineering

After analyzing the data patterns, I added four different categories of features:

1. **Basic Features**:
    a. Helpfulness Score: [given]
    b. Text Length Metrics:
        i. text_length: Character count of full review
        ii. summary_length: Character count of review summary
        iii. word_count: Word count of full review
        iv. summary_word_count: Word count of summary
2. **Important Terms**:

    a. score_{score}_word_matches: Generating scores for occurrences of representative words of the given rating:
        i. Counted occurrences of rating-specific words in both text and summary features
        ii. Summary matches were weighted double due to their concentrated nature
    b. score_{score}_word_ratio: a measure of normalized ratio of the words
3. **Sentiment**:
    a. Created sentiment features for both text and summary
        i. pos_words, neg_words, pos_ratio, neg_ratio

4. The Model

1. **K-Neighbors Classifier:**
    a. I kept with the originally suggested model because it was suitable for large datasets with numerical features, it was simple to implement and modify, was consistent across runs, and was easy to understand and explain
    b. Performance: 0.40944319968976883
        i. k set as 3 for first run
2. **Improvement 1: Adding the basic features**
    a. Added the text patterns features: text and summary length and word count
    b. Performance: 0.48049744705603553
        i. with k set as 3; basic features added as context
3. **Improvement 2: Parameter testing**
    a. Tried different numbers of neighbors for a higher accuracy and toggled the settings to have the model run faster with the “ball_tree” algorithm. I also started evaluating the parameters by looking into the mean squared error metric. The mean squared error was usually inverse to the accuracies so it helped me choose a reasonable amount of neighbors.
    b. Performance: 0.5184953788482668
        i. with k set as 21; with the basic features; ball tree algorithm (made no changes to accuracy, but made the model generate much faster); mean squared error: around 2.2
4. **Improvement 3: Adding sentiment**
    a. Couldn’t get it to work, the program kept crashing, my computer couldn’t handle the load to completion
5. **Improvement 4: Ensemble model, stratified to the weights of different ratings**
    a. Randomly sampled a number of test samples to vote on the most likely rating, and weighing the proportion of each rating there were to avoid most models predicting 5 all the time.
    b. Performance: 0.5333256134605856
        i. 27 neighbors; 8 estimator models voting; Weighted voting based on weights of rating occurrences; mean squared error: 1.9

5. Results Analysis

Performance metrics showed steady improvement:

1. **Accuracy Progress:**
    - Initial model: 40.94%
    - Added features: 48.05%
    - Optimized parameters: 51.85%

    - Final bagged model: 53.33%
2. **Model Evaluation:**
    - Confusion matrices showed better rating distribution
    - Mean squared error helped prevent overfitting
    - Final model achieved better balance between accuracy and prediction distribution
3. **Workflow Improvements:**
    - Converting some objects to numpy matrices for faster calculation
    - Parallelizing some operations
    - Obscuring the complexity of some of the messy work into organized functions

6. Future Improvements

Based on project experience and panel discussions, several improvements could be made:

1. **Alternative Models:**
    - Test logistic regression
    - Implement Naive Bayes for text
    - Try TF-IDF vectorization
2. **Feature Improvements:**
    - Implement word embeddings
    - Create user-product matrices
    - Refine feature selection
    - Change the term counts and sentiments to become a score rather than counts
3. **Process Enhancements:**
    - More systematic feature testing
    - Better experiment documentation
    - Broader model exploration

7. Conclusion

Given what I know now, I would structure future projects differently: saving data objects in pickles as checkpoints, creating separate files for experiments instead of modifying the main code, and spending more time exploring feature patterns before committing to a model. While the accuracy improved from 40.94% to 53.33%, the more valuable takeaway was learning how to approach the development process step by step.

After the panel discussion, I realized there were many alternatives I could have explored, such as logistic regression, Naive Bayes, and TF-IDF vectorization. Working with this dataset showed me the importance of careful feature selection and the need to understand model strengths—KNN might not have been the most effective choice with an increasing number of features.

Despite not achieving the highest possible accuracy, I gained practical experience in handling text data, managing imbalanced datasets, and implementing ensemble methods. The project's open-ended nature pushed me to tackle new challenges and develop a more methodical approach for future machine learning problems.

References

1. Implementation Tools:
    - Pandas, scikit-learn, nltk, numpy, seaborn/matplotlib