CS 506 Midterm:

Algorithm Overview

For this assignment, I implemented the movie review score of Amazon movie reviews based on text analysis and metadata. The main part of the algorithm uses Logistic Regression as the primary classifier, text representation, alongside several preprocessing steps and feature engineering techniques. Logistic regression is a classification algorithm, it's kind of like drawing "lines" (boundaries) to separate different review scores. Using Logistic Regression can help draw multiple boundaries between the 5 scores(1-5).

Implementations

The algorithm comprises three main stages:

1. Text Processing

- Used TF-IDF (Term Frequency-Inverse Document Frequency)
    - A way to convert text into numbers that computers can understand
    - Words that appear frequently in a specific review but are rare across all reviews get higher scores
    - Ex: If "spectacular" appears several times in one review but rarely in others, it gets a high score.
- Configured parameters:
    - max_features=2000: Limited vocabulary to top 2000 terms to prevent overfitting
    - stop_words='english': Used to remove common English words
    - ngram_range=(1, 1): Used individual words only for simplicity and better efficiency
    - This way captures the importance of words while accounting for how often they show up across all reviews

2. Features

Created several custom features to capture different aspects of the reviews:

- Text-based metrics:
    - ReviewLength: Word count of review
    - SummaryLength: Word count of review summary
    - ReviewCharLength: Character count of main review

    - SummaryCharLength: Character count of review summary
    - Long reviews tend to be more thoughtful and seem to have more effort put into them. While short reviews are less detailed or more emotional, indicating a low score(There can be detailed bad reviews).
- Engagement metrics:
    - HelpfulnessRatio: Calculated as HelpfulnessNumerator / (HelpfulnessDenominator + 1)
    - Added +1 to denominator to deal with zero cases

3. Data Cleaning

- Filled missing text fields with empty strings
- Set missing helpfulness scores to 0
- Dropped rows with missing critical fields (ProductId, UserId)
- Making sure the data stays consistent.
- This ensures the model can handle incomplete data without crashing.

4. Libraries used

- sklearn (scikit-learn):
    - LogisticRegression: The model used for prediction
    - accuracy_score: Measures how well the model performs, since we are limited to 5 entries a day I wanted to make sure I knew the approximate score before submitting to kaggle.
    - TfidfVectorizer: Converts review text into numbers
    - StandardScaler: Makes all our numerical features similar in scale
        - Prevents features with large numbers from dominating (Makes features comparable)
        - Making it memory efficient
    - Pipeline: Organizes our data processing steps
        - Keeps everything organized and prevents mistakes
- scipy.sparse:
    - This library is used for efficient matrix operations
    - This is perfect for text data where most values are zero
    - Better efficiency since it uses less memory and makes faster computations.

My thought process

When implementing the algorithm, there were a lot of challenges that I had to face. First, I knew that the dataset given to us was a very large dataset, which could quickly consume a lot of memory if not handled properly. I first thought it was important to clean the data as the dataset was fairly large. I wasn't sure if it was cleaned before it was given to us. I also assumed that the text reviews would be important when predicting the score. I decided to use TF-IDF since I thought it would be perfect for a case like this. I wasn’t too sure on how to handle this all

efficiently and through stackoverflow and ChatGPT, it led me to use sparse matrix operations with scipy's hstack, which helped with the efficiency by making a storage system that only kept track of important values. I maintained this sparsity throughout the pipeline using .tocsr() because every bit of memory optimization helps, especially when it can take ages to run something like this. I made sure to use different types of features rather than just sticking with TF-IDF as I thought it would give me a better result. I thought the other numerical features like review length, and the helpfulness ratios could really help when it came to the accuracy of the algorithm, as more detailed reviews tend to be a higher rating while shorter reviews are more emotional and can be less detailed indicating a lower rating, although there are cases where it's the opposite, this is why a variety of methods can be helpful.

A lot of the libraries I used I have learned from my other data science classes (such as DS 210) but it has also been a while since I used some of them so I did come across a lot of errors and ChatGPT did help me with better understanding these libraries as well as help with errors I had along the way.