

To tackle the objective of creating a prediction model for star ratings (scores from 1 to 5) associated with user reviews from Amazon Movie Reviews, I started by examining the provided data, which consisted of two CSV files: train.csv and test.csv. The data contained over 1.6 million rows of reviews and metadata regarding those reviews. The key challenges in creating the model were handling the large dataset efficiently, extracting meaningful features, and iteratively improving the model's predictive performance within a short period of time.

The first step was loading all of the data and preprocessing and cleaning it to ensure usability. I created two datasets, train_data and test_data, which contained the respective CSVs. The train_data ensured that the Score column was always present, as the original train.csv may not have included entries with Scores. I then handled missing values by populating them with empty string values. Although the Summary column contained more concise and specific information, I decided to combine all the string data into one FullText column, as this simplified the process and reduced the complexity of the model. The next step was to clean the data extensively since it was mainly text-based. To achieve this, I used the NLTK library, which offers an extensive suite of tools for cleaning and preprocessing. I removed non-alphabetic characters and converted everything to lowercase as a start. I then tokenized the words and removed stop words using NLTK's stopwords. After that, I lemmatized the words, which essentially distills words to their base forms, making the data even simpler.

Parallel Processing: During the initial steps of cleaning and preprocessing the data, I encountered significant slowness issues. I quickly realized that I needed to parallelize these tasks as much as possible to enhance speed. I decided to use Python's multiprocessing library along with joblib, as they are extensively used in modeling programs. I split up my data cleaning and preprocessing into chunks equal to the number of CPU cores the machine had and noticed a significant improvement in speed. To further accelerate the process, I spun up a 196-core AWS notebook machine, which made a drastic difference. I decided to utilize the 196-CPU machine and employ

a model that would allow extensive parallelism to reduce my iteration and feedback cycle.

Feature Engineering: The next part was to transform the cleaned data into usable features. I decided to use TF-IDF vectorization, as it is well-established for tasks like these and relatively computationally efficient. By using TF-IDF and penalizing common words while highlighting rare but potentially significant words, it was able to focus on words that can distinguish between different classes (in this case, different review ratings). TF-IDF also helps prevent overfitting by balancing term frequency with inverse document frequency. A notable part of this process was the use of additional parallel processing during the transformation stage to speed things up. I utilized `scipy.sparse` because TF-IDF generated sparse vectors, which I chunked and processed in parallel, resulting in speed improvements.

Additional Features: Using the other columns from our `train.csv`, I added several helpful features to use in our model. One of the main additional features was the ratio of `HelpfulnessNumerator` to `HelpfulnessDenominator`. However, there was an edge case of dividing by zero, so this needed to be properly handled. I also wanted to incorporate the timestamp feature rather than ignoring it, so I decided to implement temporal features. I converted the Unix timestamp into daytime and created the features `ReviewYear`, `ReviewMonth`, and `ReviewDayOfWeek`. This allowed us to highlight any patterns of reviews throughout the years or seasons and could even highlight the impact of cultural events or holidays. The next feature was review length, as there might be an association between the length of reviews and their ratings. Although a note about this is that it was post-data processing to maintain consistency. The following feature was the frequency of the number of times a user or product appears in the dataset. This allowed us to capture popularity and repeat reviewers and extract patterns from that. Highly reviewed products might receive more balanced ratings due to diverse user experiences, while niche products might have ratings that reflect a smaller, more homogeneous user base. Frequency encoding enables the model to consider product popularity and its potential influence on ratings. I also added a straightforward sentiment analysis feature using `TextBlob`, as it is a simple library. I used a sentiment polarity score from -1 to 1 (negative to positive sentiment) and applied that to each review. For


all of these new features, I was able to batch and parallelize them, resulting in even more speed improvements. Finally, I applied Min-Max Scaling to the numerical features to normalize them between 0 and 1. This scaling ensures that all features contribute equally to the model and prevents features with larger scales from dominating the learning process, which is helpful for models sensitive to feature scales.

Because of the high number of features from our TF-IDF matrix, I applied Truncated Singular Value Decomposition (TruncatedSVD) to reduce the feature space to 200 components. This decreases computational complexity significantly, making the model runnable in a reasonable amount of time. It also helps reduce overfitting. Lowering the dimensionality ensured that the most informative aspects of the text data were retained, enhancing the model's ability to generalize without being bogged down by redundant or irrelevant features.

Model Training: I used LightGBM, a gradient boosting framework, for its efficiency and scalability with large datasets. LightGBM handles sparse datasets well and supports multiclass specification, which aligns perfectly with the task of predicting ratings from 1 to 5. Additionally, its parallel training capabilities allowed me to maximize CPU core usage. LightGBM is also a highly optimized and efficient model, enabling even faster training. An obvious issue was the dataset's size, making exhaustive grid searches computationally impractical. To enhance the performance of the LightGBM model, I used hyperparameter tuning with RandomizedSearchCV. This significantly reduced computational time without sacrificing much in terms of finding optimal or near-optimal solutions. RandomizedSearchCV also allowed for parallelization, improving performance. However, training was still too computationally taxing. Because I was in a time crunch due to having surgery and only having a day and a half to work on this, I decided to sample only 10% of the data and use that subset. This proved to be extremely impactful as it reduced training time to only 20 minutes with the full 196 cpu cores. Unfortunately I was still unable to submit to the leaderboard on time, missing the mark by 6 minutes which is somewhat cruel. Following the data set sub-set choice, I used a simpler parameter grid with only key hyperparameters that significantly impact LightGBM's performance, such as `num_leaves`, `learning_rate`, and `n_estimators`. For these hyperparameters, I selected a smaller range of values to reduce the search space

and computational load. I also decreased the number of cross-validation folds to reduce processing time while still obtaining reliable performance estimates and decreased the number of iterations. I enabled parallel processing once again to take full advantage of all resources and speed up the training process. Even with all of this optimization and a reduced dataset, the final trained model resulted in a score of 0.63197, proving that most of the steps taken were impactful. T

This project underscored the significance of efficient data handling, thoughtful feature engineering, and strategic model selection in building robust machine learning models. The integration of parallel processing and advanced feature techniques played a pivotal role in managing the dataset's scale and enhancing model performance

Submission and Description		Private Score ⓘ	Public Score ⓘ
<div> submission(2).csv</div> <div>Complete (after deadline) · 7h ago</div>		0.64210	0.63917