Amazon Movie Review Rating Prediction:

Introduction:

The objective of this Midterm was to predict the star rating associated with user reviews from Amazon Movie Reviews using the available features. This task was approached using Support Vector Machines, Model Evaluation, Ensemble methods, and text processing.

I will mainly work with the following features from the dataset:

- Text features, the review text, and the summary.
- Numerical features, such as Helpfulness votes, and the timestamp.
- The target, which is the star rating

Implementation:

Because of the different data features, the implementation used a hybrid approach to combine text and numerical features. To process and use this data, I first had to format it so that it would be easier to use.

To process the text data, first, I combined the reviews and the summaries to capture all the text in one piece, secondly, I did some text cleaning to remove special characters, covert everything to lowercase, and normalized whitespaces. Then, I used HashingVectorizer to vectorize the text. This choice was made for several reasons, memory efficiency, the ability to handle large vocabularies without having to store the mapping, support for n-grams to capture phrase patterns, and built-in stop word removal.

```
vectorizer = HashingVectorizer(
    n_features=1200,
    stop_words='english',
    ngram_range=(1, 2)
)
```

To process the numerical data, I obtained the Helpfulness ratio (HelpfulnessNumerator / HelpfulnessDenominator) to capture the reliability of a given review, and we also normalized the timestamps ((Time - min_time) / (max_time - min_time)) to capture temporal patterns. The timestamp was also normalized to be compatible with the text features.

The chosen model was the Stochastic Gradient Descent (SGD) Classifier, some of the deciding factors of this choice were:

- Memory efficiency: This model processes one sample at a time, meaning that I don't have to load the entire data all at once, which would use a great amount of RAM because of the large training set.
- Speed: Faster than SVD, updates the model incrementally, can handle new data without retaining from scratch.

- Good for text data with many features

```
model = SGDClassifier(
    loss='modified_huber',
    penalty='l2',
    alpha=1e-4,
    max_iter=100,
    tol=1e-3,
    random_state=42,
    n_jobs=-1
)
```

The choice of ‘modified_huber’ brings tolerance to outliers as well as probability estimates, which is helpful with a large amount of 5-star reviews. The ‘l2’ penalty with 1e-4 alpha prevents overfitting with training data. Random state is used to shuffle the data and set n_jobs to -1 allowing to use of all CPUs.

To train, I implemented a batch training approach to use less RAM, I set each batch to process 10000 reviews at a time and then cleaned the memory to prevent its accumulation so that there is overall better performance, memory leaks, fewer system slowdowns, and better sharing of systems resources.

My validation strategy was to keep 20% of the data for validation and use a stratified split to maintain rating distributions and catch overfitting more easily.

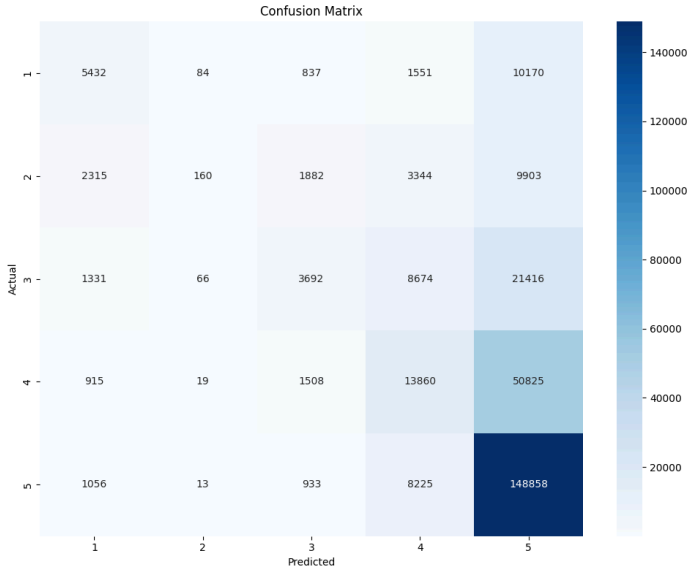
Special Tricks/Optimization:

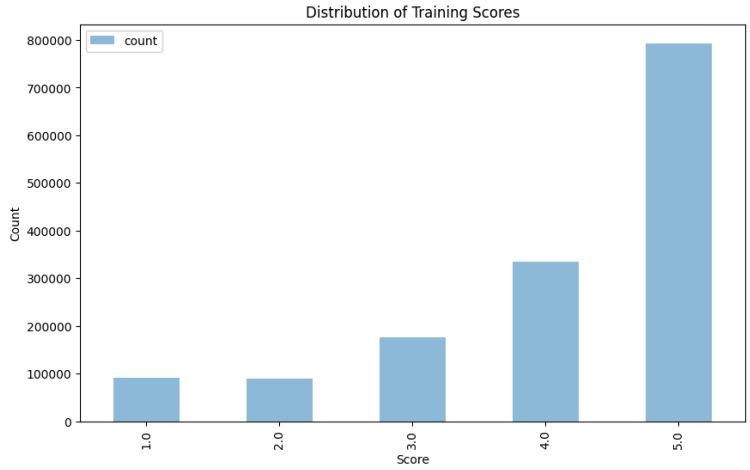
To make text cleaning more efficient I reduced regex operation to the minimum required, early return for non-string inputs, and overall kept the text cleaning to only the essential. Additionally, as mentioned before, the summary and the reviews were combined into a single text to double-weight the important phrases that might appear in both, handle missing values without losing data, and because it’s more efficient to process them together than separately.

I chose batches of 10000 reviews as it was small enough to not consume too much RAM, but big enough for efficient processing. Also, it’s a good number for SGD learning and works well with CPU cache size. I then also used stacking, both vertical and horizontal, which reduced significantly the memory usage, and also increased the processing speed.

Another change that notably improved the performance was setting n_jobs to -1, as remarked before, this setting allows the use of all CPU cores, which was a key setting for faster training, and better CPU utilization.

Results and Observations:





As observed in the distribution of training scores, there is a clear bias towards 5.0 stars. We can observe on the confusion matrix that the best performance was predicting the 5.0 stars, the precision is quite high but there was some overprediction. This can be linked to the number of 4.0 stars, which were sometimes labeled as 5.0 stars instead. 3.0-star reviews are the toughest to predict as they are right in the middle, usually confused with 4.0 or 2-star reviews. However, the poorest performance was predicting the 2.0-star reviews, which makes sense as is difficult to capture a moderate negative sentiment.

There are several areas where improvements can be made to make it more accurate, process text in a more sophisticated way, add sentiment analysis, combine multiple different models to catch different aspects and improve the predictions, and balance the data, as in the data used there were prominent 5.0-star reviews compared to the ratings.

Documentation:
https://www.geeksforgeeks.org/stochastic-gradient-descent-classifier/
https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.SGDClassifier.html
https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html