The project began with loading and inspecting the train.csv dataset to understand its structure and content. Upon exploration, I observed that the dataset comprised user reviews for movies on Amazon, including textual reviews and associated ratings. Recognizing that the sentiment expressed in these reviews would likely have a significant impact on the star ratings, I considered incorporating sentiment analysis into the feature set.

To capture the emotional tone of the reviews, I introduced two popular sentiment analysis tools I used before: TextBlob and VADER (Valence Aware Dictionary and Sentiment Reasoner). TextBlob is a general-purpose library that provides simple APIs for common natural language processing tasks, including sentiment analysis. VADER, on the other hand, is specifically designed for sentiment analysis of social media texts and short reviews, making it potentially more suitable for this dataset.

After implementing both tools, I compared their performance in capturing the sentiments of the reviews. It became apparent that VADER provided more accurate and nuanced sentiment scores for the movie reviews than TextBlob. TextBlob tended to give less precise sentiment polarity, possibly due to its broader focus and lack of optimization for shorter, informal texts. Consequently, I decided to proceed with VADER for sentiment analysis and removed TextBlob from the pipeline to streamline the process and reduce computational overhead.

Feature Engineering

Building upon the initial features, I engaged in extensive feature engineering to enrich the dataset with meaningful attributes that could enhance the model's predictive capabilities. The key features engineered included:

1. **Text Cleaning**: Implemented a text cleaning function to preprocess the review texts and summaries. This function converted text to lowercase, removed non-alphabetic characters, and extra whitespace. Cleaning the text was essential to ensure consistency and accuracy in subsequent text analysis steps.
2. **Text Length Metrics**: Calculated the lengths of the cleaned review text and summary in characters. The hypothesis was that longer reviews might contain more detailed opinions, potentially influencing the star rating.
3. **Word Counts**: Computed the number of words in the cleaned review text and summary. This provided another measure of content richness and verbosity, which could correlate with the user's satisfaction level or dissatisfaction.
4. **Sentiment Scores**: Used VADER to obtain sentiment polarity scores for both the cleaned review text and summary. These scores ranged from -1 (most negative) to 1 (most positive), offering a quantitative measure of the emotional tone of the reviews.

While examining the data, I noticed patterns suggesting that longer reviews and those with strong positive or negative sentiment scores tended to correlate with higher or lower star ratings, respectively. Including these features aimed to help the model capture the underlying sentiments and behaviors reflected in the reviews, thus improving its ability to predict the ratings accurately.

Random Forest

With the enriched feature set, I initially employed a Random Forest classifier due to its robustness and ability to handle a mix of numerical and categorical variables. I conducted hyperparameter tuning using Grid Search to optimize the model. However, several challenges emerged:

1. **Computational Limitations**: The Random Forest model, combined with Grid Search, resulted in exceedingly long training times, exceeding six hours per run. The lack of GPU acceleration support for Random Forest in scikit-learn contributed to this issue, making it impractical to iterate and refine the model efficiently.
2. **Overfitting**: Despite achieving a high accuracy of 96% on the split validation set, the model performed poorly on the unseen test data, with an accuracy of around 55% on the Kaggle test set. This discrepancy indicated that the model was overfitting the training data, failing to generalize to new data.
3. **Limited Improvement from Feature Adjustment**: Attempts to adjust features and reduce overfitting yielded marginal improvements. The model continued to struggle with generalization, suggesting that a different approach was necessary.

Recognizing these challenges, I sought advice from ChatGPT, which led me to explore alternative algorithms better suited for the task at hand.

XGBoost

Based on the insights gained, I decided to transition to XGBoost, an optimized gradient-boosting framework known for its speed and performance, particularly with large datasets and high-dimensional feature spaces. XGBoost supports GPU acceleration and includes built-in regularization parameters, making it a promising candidate to address both the computational and overfitting issues encountered with the Random Forest model.

Implementing the XGBoost Model

To implement the XGBoost model, several key steps were taken:

1. **Data Preparation**: Converted the feature set into a format suitable for XGBoost. This involved creating DMatrix objects, which are optimized data structures that XGBoost uses for efficient computation.
2. **Label Encoding**: Encoded the target variable, "Score," using LabelEncoder to convert the categorical star ratings into numerical labels required for multi-class classification.
3. **Parameter Configuration**: Carefully configured the XGBoost parameters to balance model complexity and prevent overfitting. Key parameters included:
    - **Objective Function**: Set to "multi" for multi-class classification.
    - **Number of Classes**: Specified based on the unique star ratings in the dataset.
    - **Max Depth**: Limited to 8 to prevent overfitting by restricting the depth of the trees.
    - **Learning Rate**: Set to 0.1 to control the contribution of each tree.
    - **Subsampling**: Set subsample and colsample_bytree to 0.8 to introduce randomness and reduce overfitting.
    - **Tree Method**: Used "hist" for faster computation.
    - **Device**: Specified "cuda" to utilize GPU acceleration.
    - **Evaluation Metric**: Employed "mlogloss" to evaluate model performance during training.

4. **Training and Validation**: Trained the XGBoost model on the training data and validated its performance on a split validation set. The use of GPU acceleration significantly reduced training times, allowing for more iterations and fine-tuning.

Handling High-Dimensional Sparse Data

The inclusion of TF-IDF vectorized text features resulted in high-dimensional sparse matrices. During implementation, I discovered that the TF-IDF vectorization process was computationally intensive and time-consuming due to the large vocabulary size and dataset volume. To efficiently handle this data:

- **Sparse Matrix Operations**: Converted numerical features into sparse matrices and used hstack from scipy.sparse to combine them with the TF-IDF matrices. This approach conserved memory and improved computation speed.
- **Memory Management**: Proactively managed memory usage by deleting unnecessary variables and invoking garbage collection, preventing memory exhaustion during training.

Observations and Patterns Noticed

1. **Sentiment as a Strong Predictor**: The sentiment scores obtained from VADER proved to be significant predictors of the star ratings. Positive sentiments correlated with higher ratings, while negative sentiments correlated with lower ratings.
2. **Limitations of TextBlob**: TextBlob was less effective in capturing the sentiments expressed in the reviews, leading to its removal from the feature set.
3. **Importance of Textual Features**: Incorporating textual data through TF-IDF vectorization and sentiment analysis enhanced the model's ability to understand the content and context of the reviews.
4. **Overfitting in Random Forest**: The initial overfitting issue highlighted the need for algorithms with built-in regularization and better handling of high-dimensional data.
5. **Efficiency of XGBoost**: The ability to leverage GPU acceleration and optimized algorithms made XGBoost more efficient, allowing for quicker iterations and model refinement.

Patterns Noticed and Utilized

- **Correlation Between Sentiment and Ratings**: Observed that sentiment scores were strong indicators of the star ratings, which influenced the decision to focus on sentiment analysis.
- **Overfitting with Complex Models**: Recognized the tendency of Random Forest to overfit on high-dimensional data without proper regularization, leading to the switch to XGBoost.
- **Efficiency Gains with Appropriate Tools**: Noted significant improvements in training times and model performance when using tools and algorithms suited to the data's characteristics

Special Techniques Employed

- **Feature Engineering**: Extracted meaningful features from the text data, including text length, word counts, and sentiment scores.
- **Efficient Data Handling**: Utilized sparse matrices and memory management techniques to handle high-dimensional data without resource exhaustion.
- **GPU Acceleration**: Leveraged CUDA support in XGBoost to significantly reduce training times and enable more extensive hyperparameter tuning.
- **Small Dataset:** uses small datasets to test the output since the full dataset is too big

Citations

Pandas: McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 51-56.

NumPy: Harris, C.R., Millman, K.J., van der Walt, S.J., et al. (2020). Array programming with NumPy. Nature, 585, 357–362.

Scikit-learn: Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

NLTK: Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media Inc.

SciPy: Virtanen, P., Gommers, R., Oliphant, T.E., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17, 261–272.

Regular Expressions (re module): Part of Python's Standard Library.

Garbage Collection (gc module): Part of Python's Standard Library.

Loria, S. (2018). TextBlob: Simplified Text Processing. TextBlob Documentation. Retrieved from https://textblob.readthedocs.io/en/dev/

Hutto, C.J., & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Proceedings of the International AAAI Conference on Web and Social Media, 216-225.

Virtanen, P., Gommers, R., Oliphant, T.E., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17, 261–272.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

OpenAI. (2024). *ChatGPT* (GPT-4). https://chat.openai.com/