# My process in building a model to predict Product review Stars

In this project, the goal is to predict the star rating of Amazon reviews based on a dataset of approximately 1.6 million reviews, aiming for the highest possible prediction accuracy. Given the large dataset size, I initially faced computational limitations. Even processing a small fraction (e.g., 200,000 samples) would exceed my computer's memory capacity. To manage this, I switched to Google Colab Pro, leveraging its high-RAM environment.

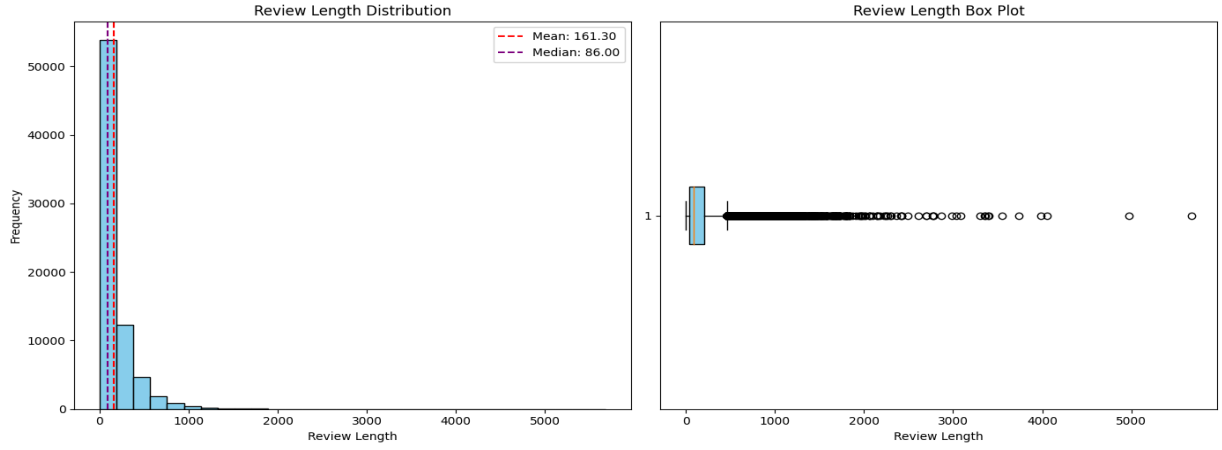
My final model utilizes a stacking approach combining multiple models, including Random Forest, LightGBM, XGBM, and ____. I also employed TF-IDF for text preprocessing and GridSearch for hyperparameter tuning.

## My Approach to Preprocessing

The original dataset contained the following features:

- **ProductId** - unique identifier for the product
- **UserId** - unique identifier for the user
- **HelpfulnessNumerator** - count of users who found the review helpful
- **HelpfulnessDenominator** - count of users who rated the review's helpfulness
- **Score** - star rating between 1 and 5
- **Time** - timestamp of the review
- **Summary** - brief summary of the review
- **Text** - main content of the review
- **Id** - unique identifier for each review

To make the Text and Summary features useful for machine learning, I transformed them into a matrix of term frequency-inverse document frequency (TF-IDF), capturing word importance through weighted values. Initially, I set TF-IDF to capture 1- to 3-word phrases, but after testing, I found that extending it to 1 to 5 words improved accuracy. I also made a function to count the number of words in a text to create a new feature called review length. I plotted the review lengths to see what the average length was for a review.

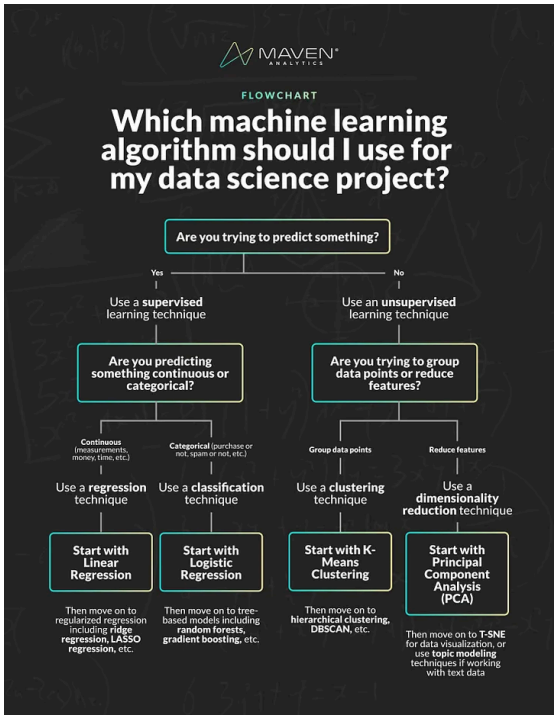


Based on this I set the max features to capture for the text was 100 and 20 for the summary. This way I can capture most of the important words and phrases without including the random words like (and , this , the). I found that instead of leaving the time feature to be a large number which might give it a higher magnitude in the model than it should be,I broke it up into year month and day hour to capture more detailed features that the original timestamp might have missed or overcompensated for.

I tried various feature engineering methods, including applying mathematical transformations to key features and seasonal insights from the Month feature, but they yielded minimal improvements. I also tried to use ProductId and UserId to get the average star on each only on the training dataset and tried to match the ids from unseen data to the ids that I have already calculated in the training data. This is to ensure no data leakage and I found that because I only used a portion of the whole dataset I don't have as many unique ids as I would have liked and wasn't able to match the majority of the ids. If I put more time into this part of the process I might have been able to create a feature that helped predict stars in the middle like 2 ,3 ,4 as the average tends to shift to the middle and to see if a particular user grades harder.

## Model Selection Process

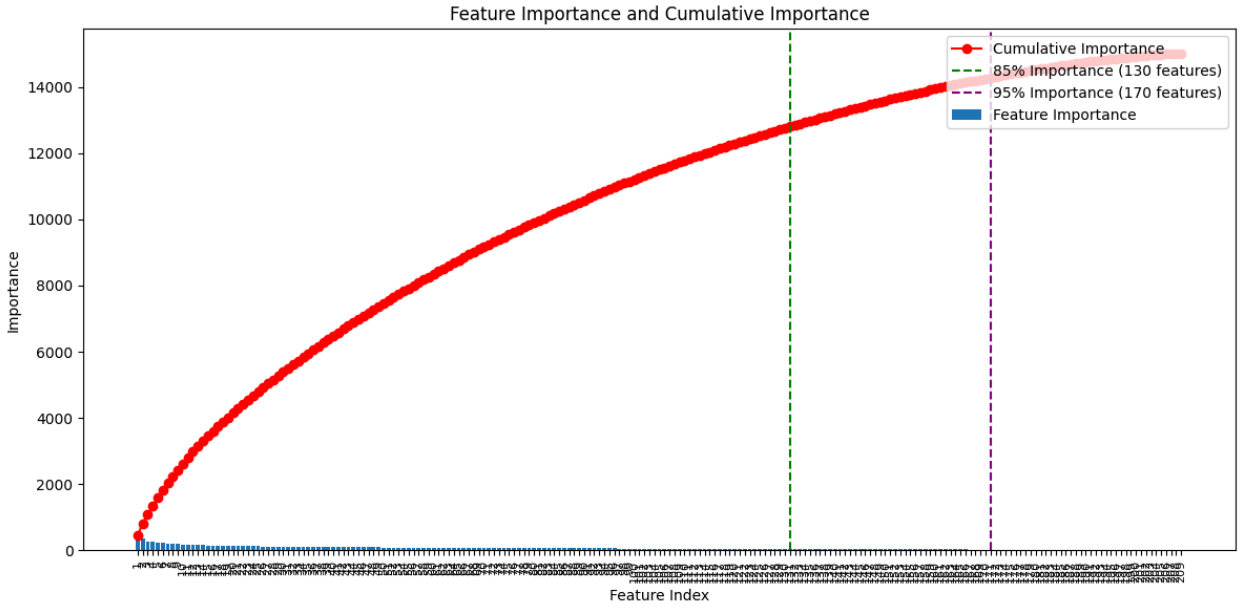
When it came to the model I did some research as to which one I should use. I found this website to be helpful with this following image.




Based on the diagram on the website I started with RandomForest. I used gridsearch and K_folds to find the optimal parameter. I started off just adjusting the basic parameters of randomforest like n_estimators, max_depth, min_samples_split, min_samples_leaf. As I was filtering through the best parameters I found that as the n_estimators increased and max_depth increased the accuracy was also increasing but it was

starting to overfit. As I wasn't able to reach the accuracy I wanted about 60 percent and the model was already overfitting and was taking a long time to process so I decided to test on another model as well.

This led me to LightGBM. It is also a decision tree model like Random Forest but it uses Histogram-Based Decision Tree Learning which makes it a much faster version of XGboost. LightGBM splits the leaf with the highest loss reduction, leading to more efficient trees. This proved to be much better since we are working with a decent amount of data and it cut down time.
This model while hypertuning the parameters produced a higher result about 58 percent accuracy. I found that a higher n_estimator and lower learning rate gave the model the ability to learn more complex features and increase accuracy but can start to overfit if the n_estimators is too high. I also found that the lower the max_depth keeping the trees smaller in depth reduces the overfitting. I also played around with other parameters to try to reduce the overfitting while increasing the accuracy like,lambda_l1, lambda_l2. Another feature I found about Lightgbm is that they have their own built-in method to determine the importance of each feature. I plotted a CDF of the number for features and IMportance.



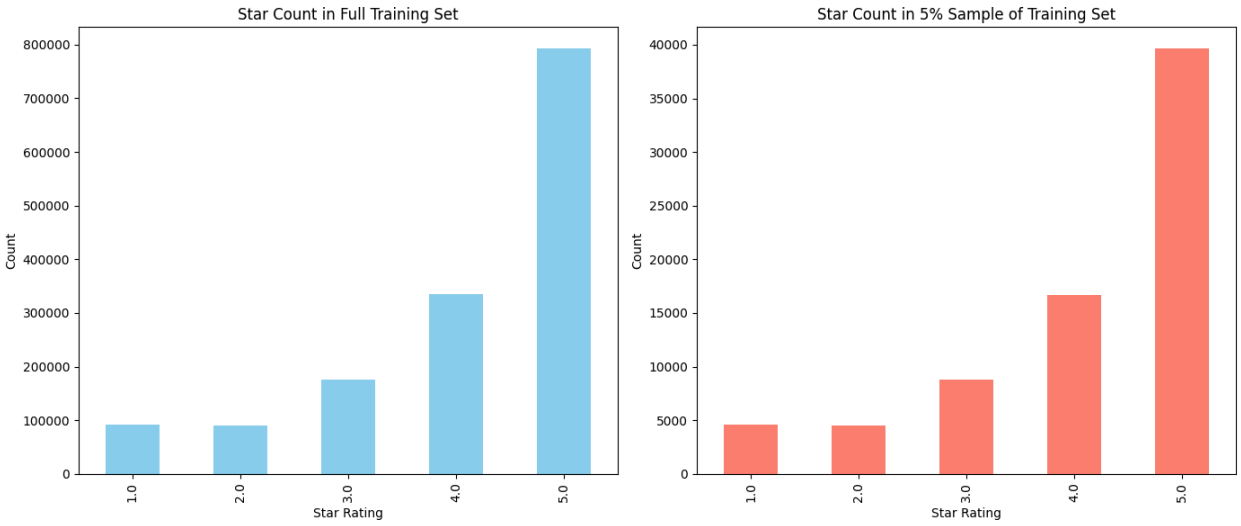

I decided on only keeping 85 percent of the importance to reduce noise and complexity for the model. This is because as the line reaches the top it gets flatter meaning the added up cumulative importance is decreasing. The less features the less complexity and noise the dataset has.

With individual models like Random Forest and LightGBM peaking at about 58% accuracy on unseen data,I decided to try stacking or blending models together to utilize each of their strengths. I ended up using Random Forest, LightGbm, XGB as the base models and logistic regression as the final estimator. Stacking gets the predictions of the base models and uses the final estimator model(meta-learner) to add weights to the predictions optimally. Effectively getting the strengths of each model's predictions. While using the stacking I was only able to receive a 70 percent accuracy on the training data and 60 percent on the unseen data. It still yielded my best results. Although the results may be underwhelming I think it is a good model for the task at hand. I did not have enough time or resources to fully optimize each base model so the accuracy does not represent the optimal solution to this model.

I didn't standardize the data as a decision tree and boosting models do well with raw data because they don't use distance and when I did implement it I got little to no change.

## Class Imbalance Challenges and Experimentation

Things I also tried and noticed but ended up not using in my final model. I noticed the huge imbalance of the data. I tried to use Smoteenn (which uses smote and ENN) to synthetically make new movie reviews for the under-represented and ENN removes outliers and noise from the majority class to balance the data. I found while it balanced the data allowing the model to learn better for movie reviews with 1 star it but the model began to predict the underrepresented class too often and the majority class too little. I was overfitting to the balanced dataset creating bias and performing worse on the new unseen unbalanced test data. I tried to only increase the minority class by a little but I found that matching the ratio of stars in the training dataset to the full dataset gave the best output. Since I chose about 0.05 percent randomly from the whole dataset as my training dataset it captured the ratio of the whole dataset.




I also tried to use class weights and thresholds to help combat against the class imbalance while not changing the distribution of the training set. I ended up not using it as both methods I couldn't produce a result that was much better than not using them. This could be due to the lack of testing.

Another thing I tried was to synthetically only increase the star rating 2, 3,4 as I realized that my stacking model was decently good a t predicting 1s and 5s – plot. THis also did not yield a higher accuracy as the Smoteenn creating synthetic examples from a high number of complex features may create examples that do not represent the class it is trying to create.

## Summary

Overall, this model's architecture—a stacked ensemble of Random Forest, LightGBM, and XGBoost—achieved a respectable 60% accuracy on unseen data. Developing new features based on observed patterns to enhance predictive accuracy was a rewarding aspect of this project. However, the most valuable learning experience came from delving into the intricacies of each model and exploring effective strategies to manage overfitting and address class imbalance. This process not only improved my

technical skills but deepened my understanding of how model choices, feature engineering, and tuning decisions directly impact performance and reliability.

**Sources:**

https://www.geeksforgeeks.org/lightgbm-light-gradient-boosting-machine/

https://medium.com/learning-data/which-machine-learning-algorithm-should-i-use-for-my-analysis-962aeff11102

https://www.geeksforgeeks.org/xgboost/

https://pranav-c.medium.com/smote-vs-smote-enn-which-is-more-effective-for-churn-prediction-in-imbalanced-banking-data-b289414366a0#:~:text=SMOTE%20with%20the%20Edited%20Nearest%20Neighbors%20(ENN)&text=ENN%20is%20a%20data%20filtering,the%20presence%20of%20noisy%20samples.