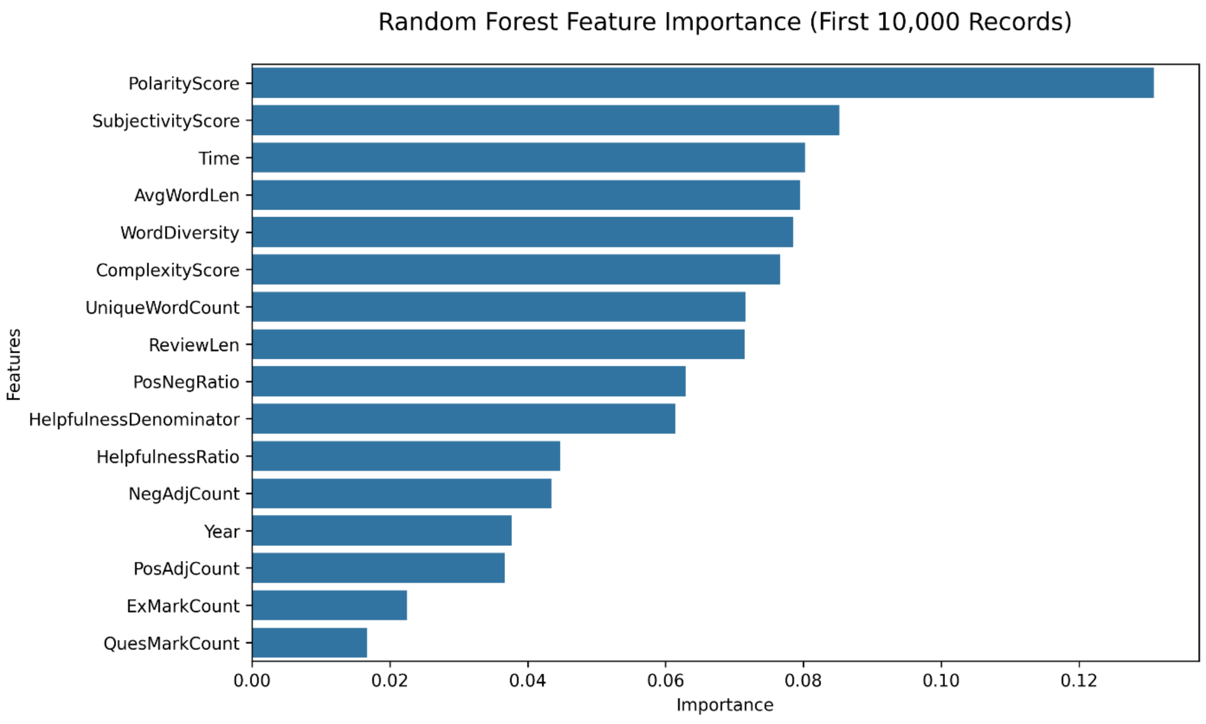
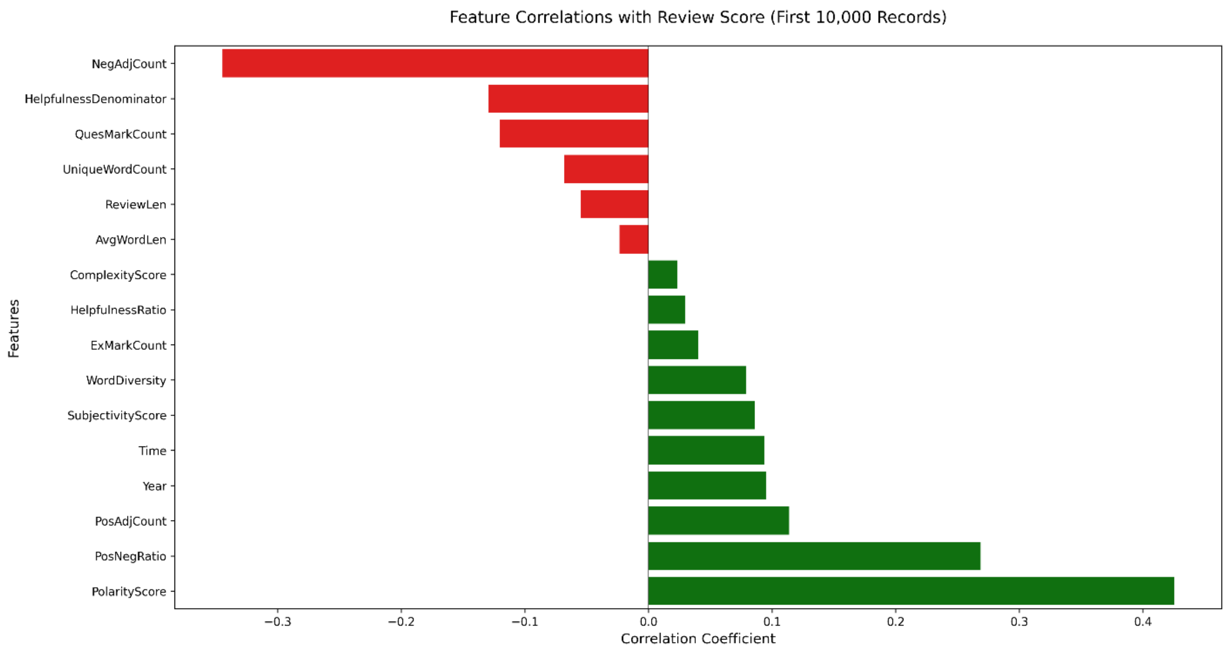
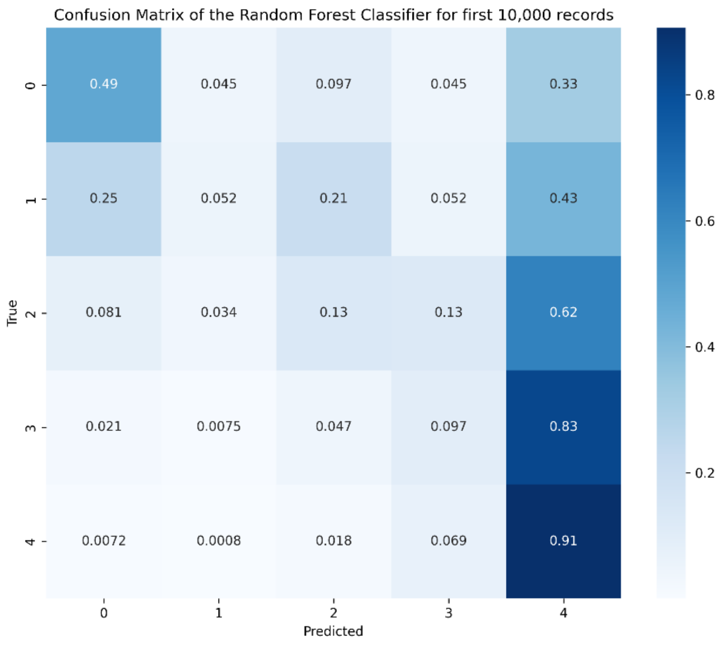
CS506 Midterm Fall 2024 Report

OVERVIEW

In building a model for predicting movie review scores, my priority was to select insightful features that would enhance the model's accuracy. Each feature's correlation with the review score was calculated to understand its influence. Features with a strong positive or negative correlation indicated intense sentiments in reviews. Very positive (5 stars) or very negative (1 star) ratings helped the model more clearly distinguish between high and low scores. I used an iterative, trial-and-error approach, incrementally adding features and evaluating their impact on the model's performance, removing those with low correlation to the test scores. Through this process, I removed features with minimal correlation (< 0.10), as they primarily added noise. However, low-correlation features were not always discarded; some contributed meaningfully when combined with other attributes to form new high-correlation features, thus improving the model's performance.







Strong performance for high ratings (4-5 stars): 91% accuracy for 4-star and 83% for 3-star reviews - Moderate performance for neutral ratings: 62% accuracy for 2-star reviews. Lower accuracy for extreme negative ratings

IMPORTANT FEATURE

- **Polarity & Subjectivity** – After researching various sentiment analysis libraries, I chose Python's TextBlob library for evaluating the subjectivity and polarity of sentences. I applied these methods to the movie reviews column to assess how strongly reviewers felt (high positive polarity = 1, high negative polarity = -1) and the level of subjectivity (factual = 0, opinionated = 1). Both attributes showed strong correlations with review scores (SubjectivityScore = 0.087, PolarityScore = 0.406), making them valuable predictors for the model. My assumption was that a high PolarityScore (indicating positive sentiment) would align with higher scores, as positive language reflects satisfaction or endorsement, while negative polarity suggests dissatisfaction. By quantifying sentiment and opinion strength, these features helped the model capture nuanced aspects of review tone and reviewer bias, enhancing its accuracy in predicting movie scores.

- **PosAdjCount & NegAdjCount** – From the training data, I manually compiled two separate lists (NEG_WORDS & POS_WORDS) of 150 commonly used positive and negative adjectives in train.csv movie reviews. Using these lists, I counted occurrences of the adjectives in the 'Text' (Reviews) column to calculate PosAdjCount and NegAdjCount, measuring the frequency of positive or negative language in each review. Both features showed significant correlations with review scores (PosAdjCount: 10.82, NegAdjCount: -37.91), indicating their predictive value for the model. Reviews with higher positive adjective counts tended to have high ratings (4-5 stars), while those with more negative adjectives were generally rated lower (1-2 stars).

- **Special Character Count** – I captured the frequency of exclamation marks and question marks in each review as ***ExMarkCount*** and ***QuesMarkCount*** to reflect the reviewer's tone. Enthusiastic or highly positive reviewers often use exclamation marks, which tended to correlate positively with scores (6.84). In contrast, question marks indicated confusion or doubt, aligning with a negative correlation (-15.79). These counts helped my model pick up on subtle cues in tone, enhancing its understanding of reviewer sentiment**.**

- **Text Complexity and Diversity Features -** To capture the diversity and sophistication of language used in each review, I introduced several features: **AvgWordLen**, **UniqueWordCount**, **WordDiversity**, and **ComplexityScore**. I calculated AvgWordLen as the measure of average word length to indicate shifts toward more complex vocabulary, though it showed a low correlation with movie review scores (-0.11). UniqueWordCount measured the count of unique words, while WordDiversity normalized this measurement by dividing it by ReviewLen, which provided insight into vocabulary variety regardless of review length. Higher WordDiversity (correlation: 11.71) indicated a more nuanced or thoughtful review, shared by reviewers who had a positive experience, while lower WordDiversity indicated simpler, potentially less enthusiastic reviews. For ComplexityScore, I combined AvgWordLen and **UniqueWordCount**, and then normalized the product by ReviewLen to assess the overall language complexity in each review (correlation: 9.28). This feature helped the model capture richness of language, indicating an engaged/expressive review style associated with polarized scores. Together, these features contributed subtle but meaningful information on each review's depth, sophistication, and engagement level, enhancing the model's ability to interpret complex sentiments.

- **Year –** For the Year feature, I converted each review's timestamp to the year format to capture any temporal trends in scoring. Since review standards or audience expectations can change over time, this feature had a positive correlation of 9.94 with review scores, potentially reflecting evolving reviewer sentiment. Including Year feature helped the model adjust for these trends, providing context on how reviews and scores may have varied across different periods.

- **PosNegRatio –** To assess the balance of positive and negative sentiment, I created PosNegRatio, defined as the count of positive adjectives divided by the count of negative adjectives plus one to avoid division by zero. This feature had a strong positive correlation with scores (27.33), as reviews with a higher PosNegRatio likely contained more favorable language, while a lower ratio implied negativity. By emphasizing this sentiment balance, PosNegRatio helped the model capture the overall tone of reviews, making it a valuable predictor for scores.

REPLAYABILITY

To ensure that my model's results were consistent and replicable for grading, I used the random library in Python to set a predetermined seed. By defining a function called setup_env, I established a controlled environment where each run produces the same results.

CHALLENGES

A key challenge I faced when developing this model was trying to minimize its runtime. While we were graded solely on our **accuracy**, **efficiency** was essential for enabling **multiple iterations** and **improvements**. As I added more features, my model's execution time significantly increased, with the add_features_to() function alone taking several hours to process the training dataset. To streamline feature selection, I limited my model to run on the first 2,000 records to quickly assess which features had the highest impact. As mentioned earlier, I created a correlation function to measure the relationship between each feature and the target score, allowing me to focus on features with the highest predictive power. By Sunday night, I had implemented over 16 features, carefully balancing feature selection to avoid introducing noise or unnecessary calculations. This approach was crucial in overcoming the runtime challenge, as I could identify and retain only the most effective features, maximizing predictive accuracy while reducing computational overhead.

SPECIAL TRICKS AND ASSUMPTIONS

In constructing features, I applied several assumptions to try to capture detail aspects of movie reviews. For example, I assumed that words with positive sentiment generally correlate with higher scores, as favorable language often reflects satisfaction or endorsement of a movie, while negative sentiment words indicate dissatisfaction, leading to lower scores. Similarly, I assumed that reviews with complex vocabulary or higher word diversity might indicate a more engaged or thoughtful reviewer, resulting in a polarized score (either high or low). These assumptions guided the creation of features like **PolarityScore**, **PosNegRatio**, **WordDiversity**, and **ComplexityScore**, each aiming to capture different aspects of sentiment, tone, and reviewer engagement. By building features around these assumptions, I aimed to enhance the model's ability to understand and predict movie review scores effectively.

CHOOSING THE MODEL & PARAMETERS

I initially experimented with a K-Nearest Neighbors (KNeighborsClassifier) model to predict movie review scores. However, I soon encountered significant challenges with this approach due to the size of the dataset—1.7 million reviews. The model took an average of 5–6 hours to complete each run, making it too slow for effective testing and iteration. As a result, I explored alternative classifiers and ultimately chose RandomForestClassifier. This model not only managed large datasets more efficiently but also offered the added benefit of built-in feature importance evaluation.

RandomForestClassifier offered several key advantages over K-Nearest Neighbors. First, it was significantly faster, completing runs in about 2 hours on the same dataset. Additionally, RandomForestClassifier evaluated feature importance, which was essential given the 16 features I developed, each with varying predictive value (e.g., sentiment scores, word counts, text complexity scores). This capability allowed the model to focus more on high-impact features, enhancing predictive accuracy. To improve the model's performance, I focused on fine-tuning the RandomForestClassifier's hyperparameters. By experimenting with smaller subset of 30,000 records, I found that using 700 trees struck a balance between accuracy and runtime. Increasing the number of trees beyond 700 didn't result in much improvement in accuracy and only increased the overall runtime. I also set the max_depth to 20 to avoid overfitting, as deeper trees tended to capture noise in the training data.