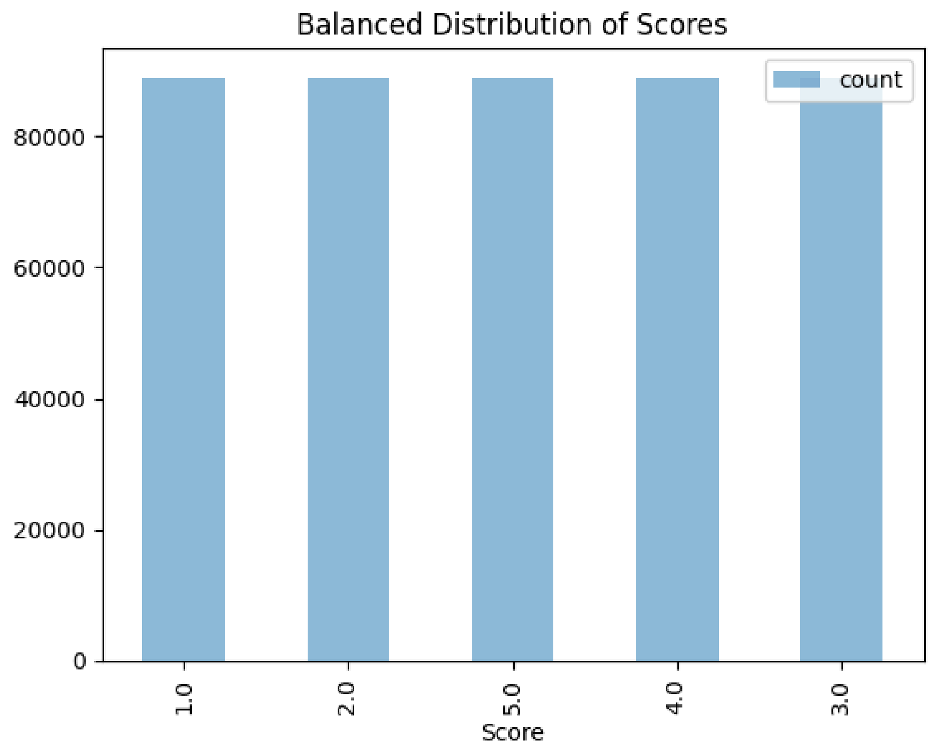
Rebecca Geisberg
CS506 Midterm
Professor Lance Galletti
Monday, October 28th 2024

Exploratory data analysis:
In this assignment we were given a dataset of movie reviews, with several features, and with their respective star rating. After seeing that the original distribution of scores was skewed, with far more 5 star reviews than 1 star reviews, I decided to balance out the dataset using undersampling. With this method, I obtained 89,000 reviews for each star respectively. I believed that by balancing out the dataset, not only would I save runtime, but also my model would be better at predicting the minority classes. Additionally, I was hoping that this would prevent my model from overfitting to the majority class of 5 star reviews. Additionally, I learned from stress testing that predicting that every review is 5 stars gives you an accuracy of .53.

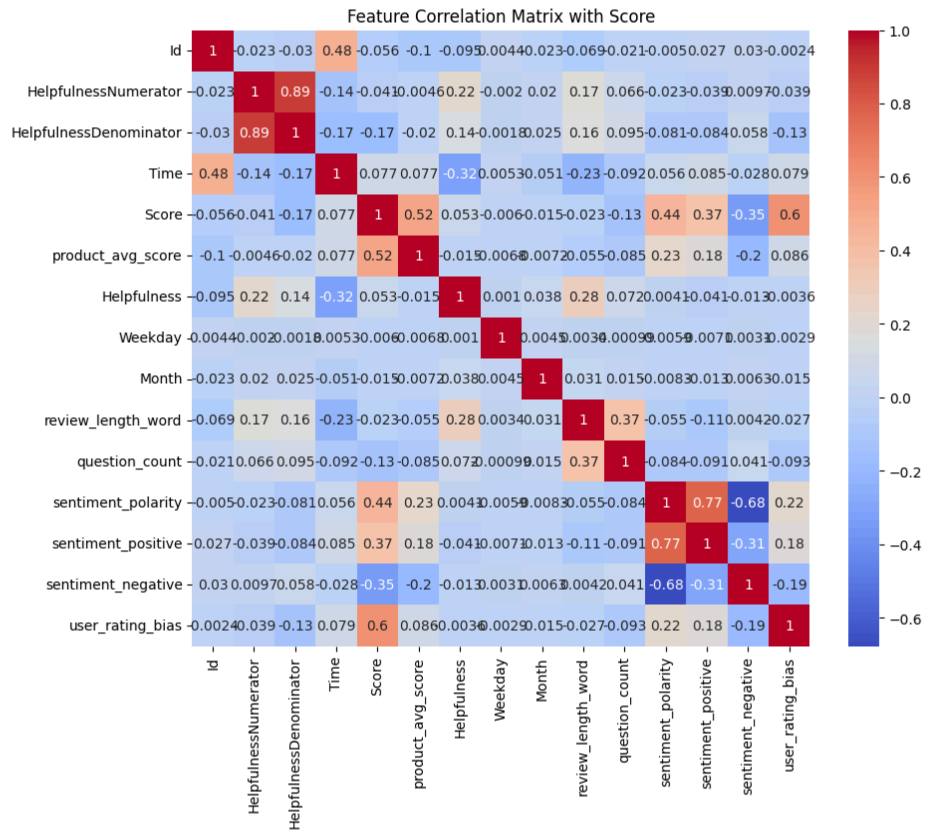


Feature selection and evaluation:

When thinking about the data, I thought that the most impactful features would be:
- Sentiment analysis on both the text and the summary
- The average product rating score, in case we saw that productId again in the test
- Average sentence length, in case more neutral reviews had more to say that was both positive and negative

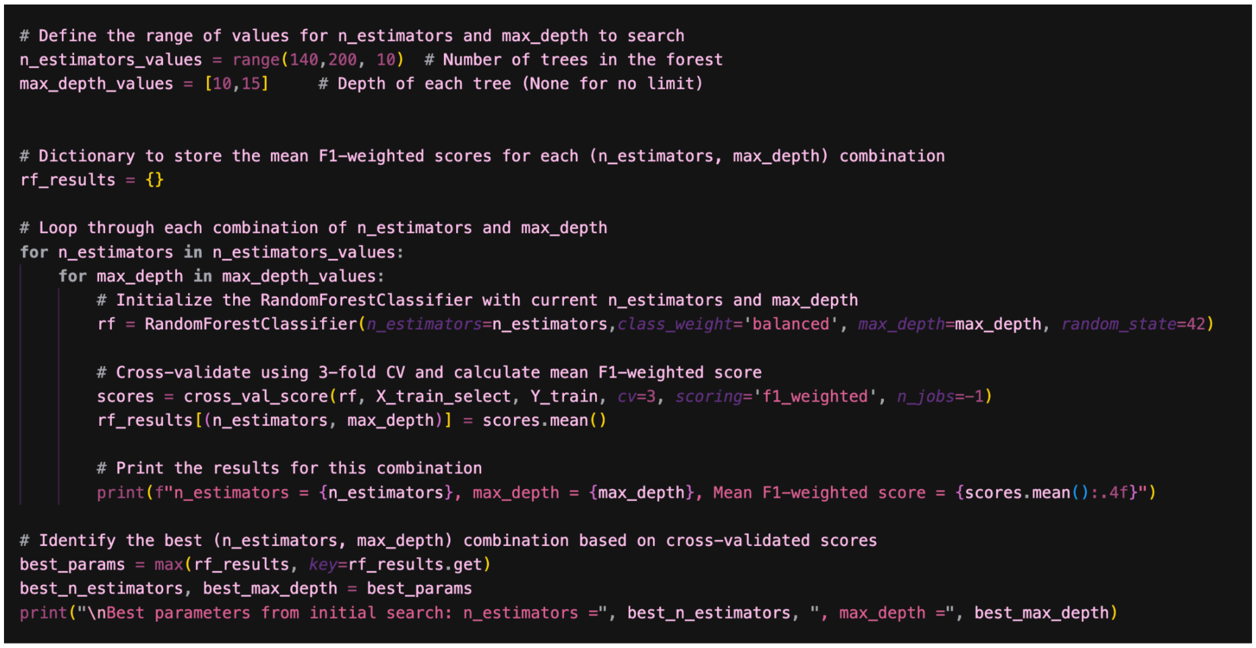
- At one point, I had a sentiment difference feature, which calculated if a review text had varying sentiment, which I thought would be good at identifying the middle star ratings
- I added certain word counters like “average” , “mediocre” and others to see if they would have a high correlation with the overall score, but they did not, so I ended up using a question mark counter/indicator

I made a correlation matrix, in order to help me visualize these features and their impact on the score. From the matrix, we can see that sentiment scores and user rating bias have high correlation with the overall score. Additionally, we can see that the average product score, HelpfulnessDenominator also has a high correlation. The reason I only kept the question mark counter/identifier is because we can see in the matrix that it has decent correlation with score. Based on this matrix, I decided to choose the following features to train my model: ‘HelpfulnessDenominator', 'HelpfulnessNumerator', 'review_length_word', 'question_count', 'sentiment_polarity', 'product_avg_score', 'sentiment_positive', 'sentiment_negative', 'user_rating_bias'. Not only this, but I used TF-IDF in order to try and further analyze the text, and highlight specific words that could be helpful within specific contexts. While not all of these features are highly correlated with score, I thought that retaining some of the less significant features would help with tiebreakers/reviews with 2,3,and 4 stars.



Parameter tuning/ Model selection:

For my model, I chose to use a RandomForest. I chose this model because the data/features we're working with are a mix of numerical and text. Not only this, but RandomForest is a more robust model that is less sensitive to noise and outliers, which is beneficial for our purposes. If there was a text review that was really outlandish for its score/rating, the RandomForest model would handle it well without skewing other predictions as a result. Additionally, RandomForest is good at handling nonlinear relationships, which is extremely useful for 2,3 and 4 star reviews, as the relationships present in those reviews can be more complex. Due to the fact that the middle star reviews were the hardest to predict, I believed that this model would be a good choice. For parameter tuning, I ran a loop that tried different numbers of trees with different depths. To begin, I had the number of trees from 60 → 140, but my optimal case was on the higher end of that so I adjusted as needed. Additionally, originally I was testing depth of 10,15,20 and None for maximum depth, but the highest scores always came from a depth of 15 so I adjusted the loop to save runtime.



```python
# Define the range of values for n_estimators and max_depth to search
n_estimators_values = range(140,200, 10)  # Number of trees in the forest
max_depth_values = [10,15]     # Depth of each tree (None for no limit)


# Dictionary to store the mean F1-weighted scores for each (n_estimators, max_depth) combination
rf_results = {}

# Loop through each combination of n_estimators and max_depth
for n_estimators in n_estimators_values:
    for max_depth in max_depth_values:
        # Initialize the RandomForestClassifier with current n_estimators and max_depth
        rf = RandomForestClassifier(n_estimators=n_estimators,class_weight='balanced', max_depth=max_depth, random_state=42)

        # Cross-validate using 3-fold CV and calculate mean F1-weighted score
        scores = cross_val_score(rf, X_train_select, Y_train, cv=3, scoring='f1_weighted', n_jobs=-1)
        rf_results[(n_estimators, max_depth)] = scores.mean()

        # Print the results for this combination
        print(f"n_estimators = {n_estimators}, max_depth = {max_depth}, Mean F1-weighted score = {scores.mean():.4f}")

# Identify the best (n_estimators, max_depth) combination based on cross-validated scores
best_params = max(rf_results, key=rf_results.get)
best_n_estimators, best_max_depth = best_params
print("\nBest parameters from initial search: n_estimators =", best_n_estimators, ", max_depth =", best_max_depth)
```

How did the model end up performing:

My model did not end up performing well at all!! Just from stress-testing the data, I know that always choosing a five star rating would give you an accuracy of about .53, and yet my model was only approximately .48 percent accurate. Ultimately, I understand that this most likely boils down to not choosing the right features for my dataset. I honestly thought that the features I chose were going to perform better than they did.