Yingjie Ouyang
Predicting Amazon Movie Star Ratings
Kaggle username: yingjieo

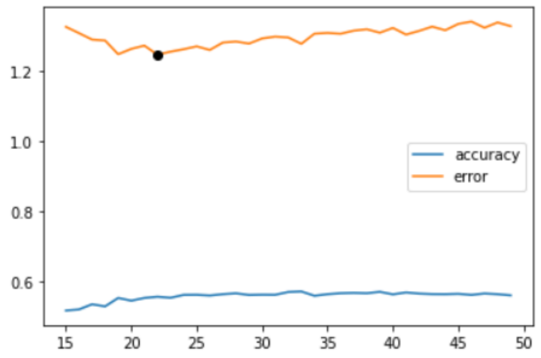
The goal of this project was to create a classification model to predict Amazon movie star ratings. The dataset contained information such as the ProductID, review of the product, the UserID of the user who reviewed the product. My aim was to predict the Score column of the data, which contained values from 1 to 5, representing the star rating the reviewer gave the product. The dataset had very little easily-interpretable information, and it was unclear what correlation they may have had with the outcome variable from initial data visualization. Thus, data preprocessing and feature engineering were key to building a strong model.

Feature Engineering

The feature I focused on was the reviews, which contained the users' impressions of the movie. Commonly-recurring words or phrases across different reviews could indicate similar scores, and key terms such as "amazing" or "worst" could suggest a higher or lower rating. Before doing any processing, I minimized potential noise in the text, such as removing capitalization and punctuation. I also performed lemmatization on the text, which considers a word's base form and combines similar-meaning words. To extract information from the reviews, I utilized Term Frequency-Inverse Document Frequency, a measure of word frequency in a document which also takes into account its overall importance in the document. To implement this, I used the TFIDFVectorizer library in sklearn. I generated a list of stopwords for the vectorizer to ignore in order to further reduce noise. In addition, I also tested out the vectorizer's performance on different n-grams. For example, a unigram could detect the word "good", whereas a bigram could detect "no good"–two very polarizing phrases that could affect performance. I also added sentiment scores from the VADER library to my features, which I thought could aid the model in detecting the differences between a positive, neutral, and negative review.

Model Building

After data preprocessing, I sampled about 20,000 entries in the training data and split the sample into train and test sets. I fit various classification models on the train set and evaluated their performance on the test set. I also conducted parameter tuning for each of these models by fitting models with different parameters and graphing their accuracies and RMSEs. I chose models with parameters that yielded the lowest test RMSE.

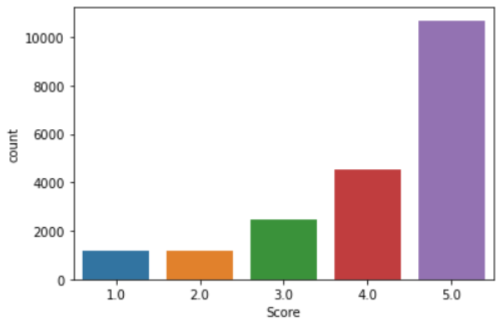


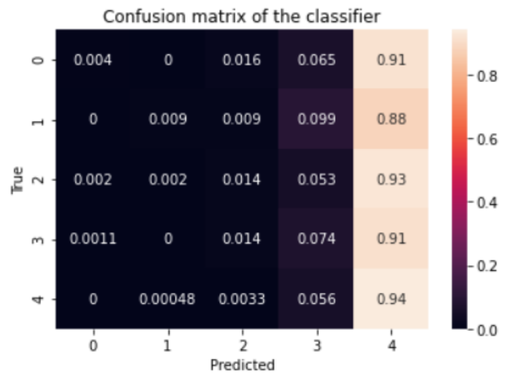
Tuning of max_depth parameter of RandomForestClassifier

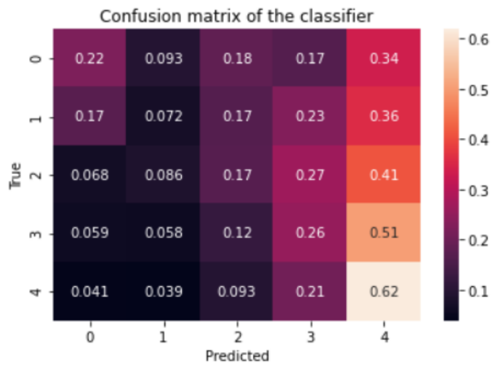
K-Nearest Neighbors: The initial model provided with n_neighbors = 20 had an average RMSE of about 1.4. The optimal number of neighbors varied depending on features.
Decision Tree/Random Forest: Performed better than KNN, but was more complicated to tune as it had many parameters and was also prone to overfitting. Random Forest consistently performed better than Decision Trees, however took much longer to fit to the data.
Support Vector Machine: I utilized the SVC library from sklearn. The model performed better than KNN but worse than Random Forest and took a long time to run.
Naive Bayes: There are three implementations of Naive Bayes in sklearn that I tested: GaussianNB, ComplementNB, and MultinomialNB. Multinomial performed consistently poorer than Gaussian and Complement. Naive Bayes is sensitive to features, so tuning my parameters for TFIDFVectorizer drastically changed the models' performances.
Logistic Regression: LogisticRegression performed the best in classifying reviews overall, while its accuracy tended to be similar to Naive Bayes. Adjusting class_weight to "balanced" and mutli_class to "ovr" improved its performance.

Performance Evaluation

All of the models suffered from similar issues. Namely, they predicted 5-star and 1-star reviews well, while doing poorly on 2, 3, and 4 star reviews. A preliminary look at the distribution of scores in the data, even when sampling, revealed there are a greater number of 5-star reviews compared to other star reviews. The countplot on the right shows the number of each score in the sample of 20,000 reviews. Notice that 5-star reviews make up over half of the distribution. Thus, the models would often predict 5-stars for the majority of the reviews.



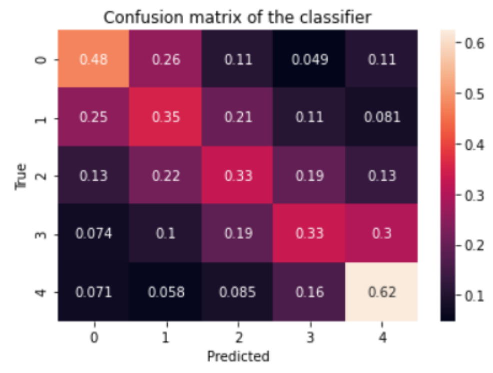




Above are the confusion matrices for KNN with n_neighbors = 20 and DecisionTreeClassifier initialized with the default parameters, respectively. For KNN, the accuracy was 0.51 and RMSE = 1.46. However, as shown by the confusion matrix, the model predicted 5-stars for the majority of the data. The Decision Tree detected 1, 3, and 4-star reviews better than KNN, but had a lower overall accuracy of 0.42 and higher RMSE of 1.47. The confusion matrix is also skewed heavily towards 5-star reviews.

While the accuracy consistently stayed at around 0.5, the metric is misleading as it was misclassifying lower-star ratings. With the disparity between number of 5-star reviews compared to lower-star reviews, both TF-IDF and the models have more data associated with 5-star ratings, there is less information on the other star reviews for the model to train on. This imbalance of data leads to misleading accuracy, as the model could rate every review as 5-stars and still obtain a decent score. I added a measure of the F1-score in my model evaluation, which yielded a better understanding of the model's performance. For example, the accuracy of Logistic Regression (right) is around 0.50, while the F1-score is slightly higher at 0.52. Just from looking at the confusion matrix, it is clear that Logistic Regression performs classification better than KNN and Decision Tree, notably on 2, 3, and 4-star reviews.

```
Accuracy on testing set =  0.49783333333333335
RMSE on testing set =  1.396364326862203
F1 Score on testing set =  0.5209440572488047
```



True \ Predicted	0	1	2	3	4
0	0.48	0.26	0.11	0.049	0.11
1	0.25	0.35	0.21	0.11	0.081
2	0.13	0.22	0.33	0.19	0.13
3	0.074	0.1	0.19	0.33	0.3
4	0.071	0.058	0.085	0.16	0.62

There are two primary ways to address the issue of data imbalance, both of which I attempted. First, I picked equal samples of each star rating and trained my model on the resulting dataset. As a result, the performance for 1-2 star ratings improved, but overall performance decreased since the model was predicting 5-star reviews wrong more often. In addition, there is a risk of overfitting as the sample is not representative of the entire dataset. The second option was to implement bagging and boosting, which improves performance where the models are predicting poorly. I implemented BaggingClassifier and both AdaBoostClassifier and GardientBoostingClassifier, and discovered that they rarely increased the performance by a lot. In the case of boosting, they often performed worse.

I also implemented regression models to compare their performance to the classification models, and found out that they often performed better than classification. My intuition is that since the classification models predict a label of 1 to 5, the majority of the error comes from when it struggles with predicting 2 to 4-star reviews. Since regression predicts a continuous value, the model does better on ambiguous cases. For example, if a user rates a product 2 stars, a classification model could have trouble differentiating between 2 and 3-star reviews, and output a label of 3. Alternatively, a regression model could predict 2.5, which overall lowers its error rate. Surprisingly, the best-performing regression models were Ridge and RandomForestRegressor, both which consistently yielded an RMSE of under 1.0, depending on how well I selected my features.

Future steps would be to look into principal component analysis and singular value decomposition, which could help with feature selection and overall speeding up the running time it takes to train and test models. I would have also liked to implement k-fold cross-validation in order to evaluate model performance. As my models were generalizing well to the test data, it was not a top priority, but it could likely help in improving overall performance. In addition, I focused mostly on the text review feature to extract information, but perhaps it would have also been useful to look at ProductID, UserID, and Time, to see if any more relationships could be discovered.