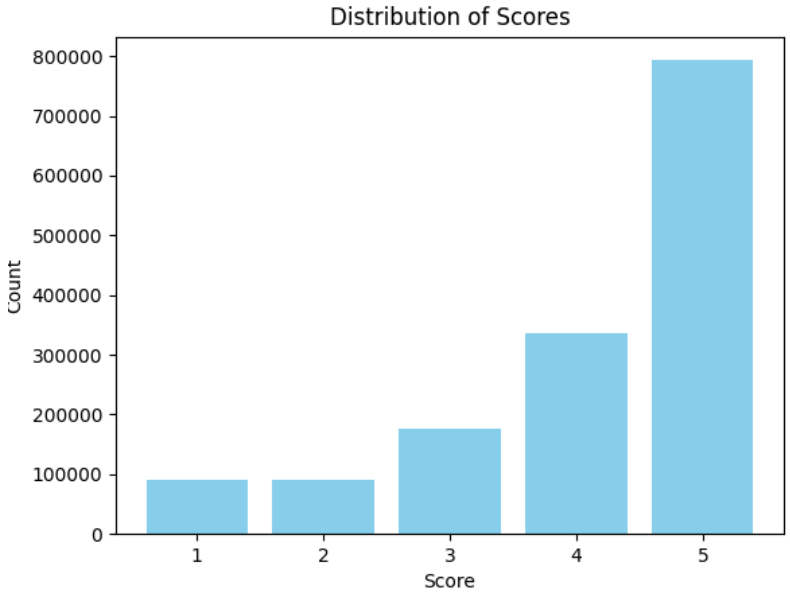
Contents

1 Model

I trained a random forest classifier as this was the advice the TA gave me during office hours. The random forest creates a decision tree to classify the inputs into buckets.

1.1 Optimization

Below is the distribution of scores. I tried to use weights inversly proportional to the frequency of each score to perform better in class-balanced performance metrics; however, the highest accuracy achieved was through no weights in my random forest model.



1.2 Assumptions

I assumed that the description and helpfulness metrics were the only valuable metrics. I could have incorporated more parameters, i.e. the movie name; however, I wanted to train the model as fast as possible and decrease my state space since training took very long on my localhost and on google collab.

I translated each description review of a movie into a ratio of positive words divided by total words (negative and positive) using nltk SentimentAnalyzer().

For the helpfulness ratio I calculated the Helpfulness numerator divided by the Helfulness denominator. I used these two inputs to classify a movie into one of 5 buckets (1 to 5 stars).

1.3 Hyperparameter search

I decided to use a ratio of 0.9 for my $\text{train}_{\text{size}}$ in my $\text{train}_{\text{test}}$ split because I wanted as much training data as I could get. Also, for some reason one time when I ran the model it got 66% on the test splut after changing this ratio, but before submitting to kaggle I tried to add gridsearch to further optimize and could not reach that score again.

I tried using used 10 iterations in my grid search because my time was constrained and the model would often run out of memory and crash. As for the hyperparameter gridsearch I used the following grid:

```
param_grid = { # for CV search
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'class_weight': [None, 'balanced']
}
```

I used balanced after the class weight ratio of frequency failed to give better results, and for the other hyperparameters I referenced this medium article on hyperparameters in RF to create a subset of his parameters for a small grid.

1.4 Problems encountered

Model training took forever and I didn't have enough cores or memory sometimes to train a model with more than 2 parallel jobs. For example, I would

train for 15 minutes then the model would crash before finishing. E.g.:

```
/usr/lib/python3/dist-packages/joblib/externals/loky/process_executor.py:752:
↪  UserWarning: A worker stopped while some jobs were given to the
↪  executor. This can be caused by a too short worker timeout or by
↪  a memory leak.
  warnings.warn(
```

To address this issue I cached the creation of input data (i.e. the helpfulness ratio and sentiment ratio). Also, I cached the best model found through RandomSearchCV.

I tried to do a larger grid search with 300 fits but ran into memory crashes. I used the same grid search in the medium article to search a large space of hyperparameter combinations, but did not have proper hardware to execute this well.

1.5 Matching submission

The last submission I made (0.53745 public score) is the one that the jupyter notebook matches. The best one I got was 0.55036 but I didn't checkpoint this in saving my jupyter notebook and was trying to get better before the deadline, so I didn't save this one. I do have the model code cached locally which I can share but it's 1.0GB large.