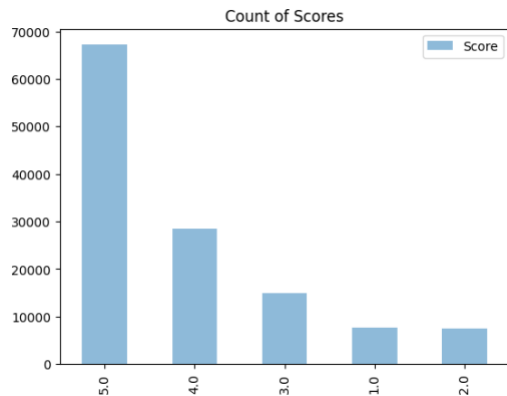


506 Mid-term Competition

Zhengxu Wang

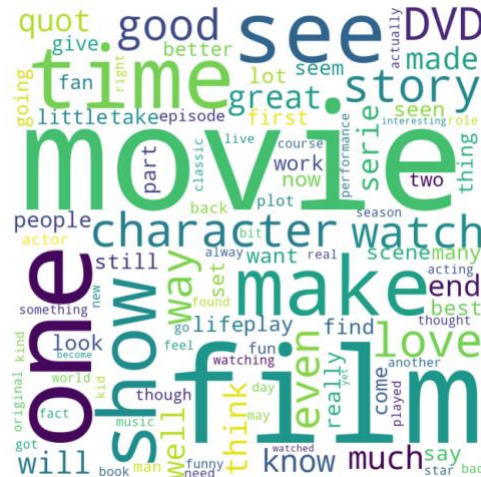
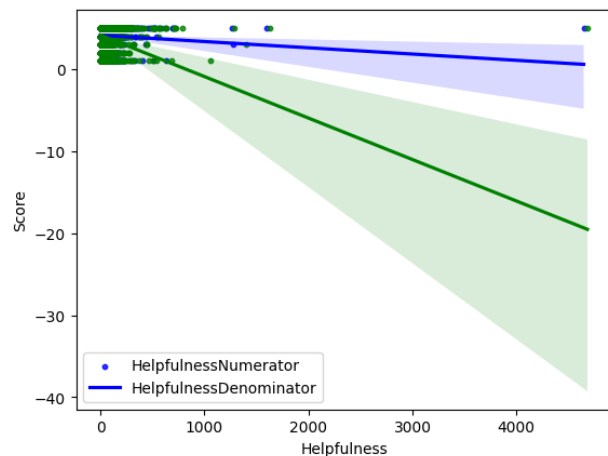
Kaggle: wangzhengxu

Data Analysis:



The Score classes are very unbalanced, so I modified my model using unbalanced weights in the training process to give different losses. But after training and testing, I found it is better without unbalanced weights. That's might because the submission's test set and validation set both from the original dataset, so they subordinate to the same distribution. So, it is not a bad idea to train a biased model.

```
weights = {1: 2, 2: 1, 3: 1, 4: 1, 5: 0.5}
sample_weights = np.array(list(map(weights.get, Y_train)))
```



The graph shows two scatter plots, one for HelpfulnessNumerator and one for HelpfulnessDenominator, with Score on the y-axis. The blue points represent the relationship between HelpfulnessNumerator and Score, while the green points represent the relationship between HelpfulnessDenominator and Score. Each point in the scatter plot represents a review in the training dataset.

From the plot, we can see that there is a slight positive correlation between Score and both HelpfulnessNumerator and HelpfulnessDenominator, but the relationship is not very strong. The regression lines are relatively flat, indicating that there is not a large change in Score for each unit increase in Helpfulness. So I decide to keep these features. And I also analyzed about UserId and review Ids, they are very irrelevant to Scores, so I dropped them.

The second graph is The wordcloud is a visual representation of the most frequently occurring words in the Text and Summary columns of the training data.

The size of each word in the wordcloud corresponds to its frequency in the text, so larger words appear more frequently in the dataset. The color of each word is random and doesn't have any meaning.

From the wordcloud, we can see that some of the most common words in the Text and Summary columns include a lot of meaningless words, so we need to filter these words.

Data Preprocessing & Text Vectorize:

```
# Define the column transformer to apply the preprocessing steps to the appropriate columns
preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), ['HelpfulnessNumerator', 'HelpfulnessDenominator', 'Helpfulness', 'ReviewLength']),
    ('text', TfidfVectorizer(stop_words='english', max_features=5000, ngram_range=(1,2)), 'Text'),
    ('summary', TfidfVectorizer(stop_words='english', max_features=5000, ngram_range=(1,2)), 'Summary'),
])
```

The preprocessor is a ColumnTransformer that applies various transformations to the input data to preprocess it for modeling.

There are three transformers defined in the preprocessor. The first transformer (num) applies the StandardScaler to standardize the numerical features HelpfulnessNumerator, HelpfulnessDenominator, Helpfulness, and ReviewLength.

The second and third transformers (text and summary) apply the TfidfVectorizer to convert the text features Text and Summary into numerical features. The TfidfVectorizer converts the raw text into a matrix of TF-IDF (term frequency-inverse document frequency) features, which represent how important each word is to the document relative to the entire corpus of documents. The stop_words='english' argument removes common English words like "the", "a", and "an" from the input, and the max_features=5000 argument limits the number of features to the top 5000 most frequent words. The ngram_range=(1,2) argument includes both unigrams and bigrams as features, meaning that single words and pairs of adjacent words are both included in the feature set.

By applying these transformations, the preprocessor converts the raw input data into a preprocessed feature matrix that can be used as input to a machine learning model.

Model:

```
# Define the Ridge model
ridge_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', Ridge(alpha=7.5))
])

# Define the Linear Regression model
linreg_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', LinearRegression())
])

# Define the stacking regressor with the Ridge and Linear Regression models as base estimators
stacked_model = StackingRegressor(estimators=[('ridge', ridge_model), ('linreg', linreg_model)])

# Train the model
stacked_model.fit(X_train, Y_train)
```

I used a StackingRegressor model which is an ensemble learning technique that combines multiple base regressors, in this case, a Ridge and a Linear Regression model, to improve the overall performance. I also tried a lot of other models and combinations, this is the best one.



The preprocessor is used to preprocess the data, which includes scaling the numerical features and applying TF-IDF vectorization to the text and summary columns. The Ridge model and Linear Regression model are then applied to the preprocessed data in separate pipelines.

Finally, the stacked model is created by stacking the two pipelines, where the predictions of the base models are used as features for a final regression model. The stacked model is trained on the preprocessed data, X_train and Y_train, and evaluated on the test data, X_test and Y_test.

Output:

I analyzed the output data, I found there are some data greater than 5 or lower than 1. So, I make my predictions keep in range [1,5].

RMSE on testing set = 0.6479185334534749

7	Wang ZhengXu		0.79296	19	6h
 Your Best Entry! Your submission scored 0.79304, which is not an improvement of your previous score. Keep trying!					