First Tests

I used google colab to run my tests. First, I took a smaller sample (50,000) and extracted some key features. I graphed these features against score to visualize their distribution and if they would help my model learn.

- Text length

2,3, and 4 star reviews were longer. These scores have more nuance, so it makes sense they would be longer.

- Sentiment analysis of Text + Summary

This was a very strong feature able to determine whether the text and summary was overall positive or negative

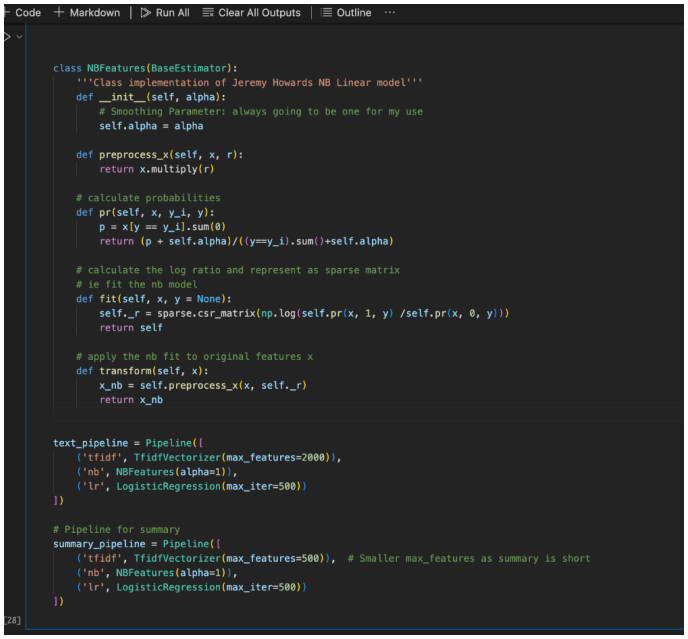
- Subjectivity analysis of Text + Summary

A very subjective review has less emotion in it, meaning the reviewer did not feel as strongly about the product. This also indicates a more reasonable review (2-4).

I created and graphed a few more like the number of user reviews and product reviews, but they had little feature importance. This could be seen from their graphs not showing any differences between groups, and feature importance tests from a random trees network.

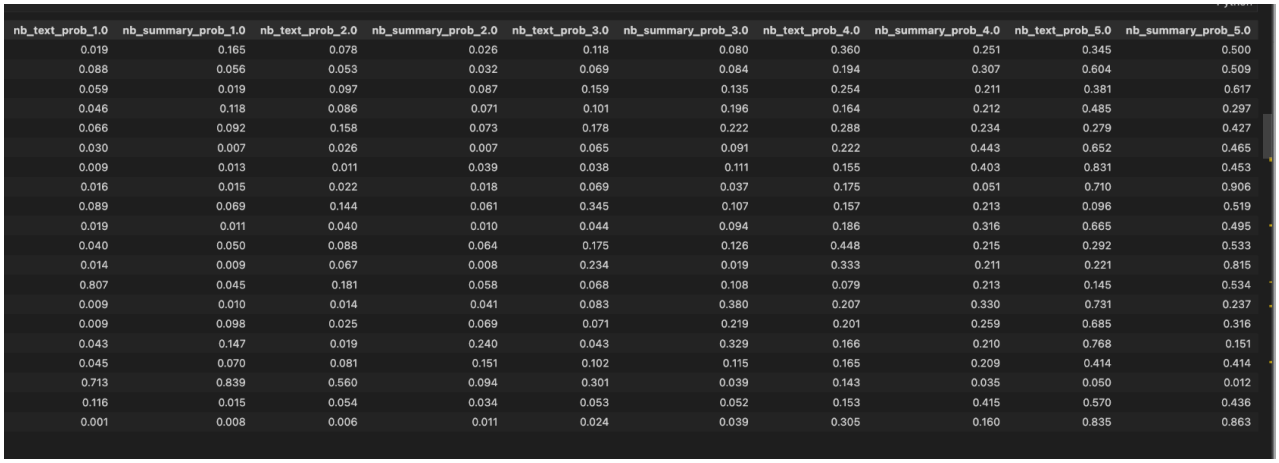
Naive Bayes

From searching on google, I found a smart usage for Naive Bayes. The Bayes theorem deals with “givens” where it determines the probability that a review will be x-stars given the words in their text and summary. By training a Naives Bayes class on the data, it amassed a dictionary that could help it determine between different scores.



```
Code  Markdown | Run All  Clear All Outputs | Outline ···

class NBFeatures(BaseEstimator):
    '''Class implementation of Jeremy Howards NB Linear model'''
    def __init__(self, alpha):
        # Smoothing Parameter: always going to be one for my use
        self.alpha = alpha

    def preprocess_x(self, x, r):
        return x.multiply(r)

    # calculate probabilities
    def pr(self, x, y_i, y):
        p = x[y == y_i].sum(0)
        return (p + self.alpha)/((y==y_i).sum()+self.alpha)

    # calculate the log ratio and represent as sparse matrix
    # ie fit the nb model
    def fit(self, x, y = None):
        self._r = sparse.csr_matrix(np.log(self.pr(x, 1, y) /self.pr(x, 0, y)))
        return self

    # apply the nb fit to original features x
    def transform(self, x):
        x_nb = self.preprocess_x(x, self._r)
        return x_nb


text_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=2000)),
    ('nb', NBFeatures(alpha=1)),
    ('lr', LogisticRegression(max_iter=500))
])

# Pipeline for summary
summary_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=500)),  # Smaller max_features as summary is short
    ('nb', NBFeatures(alpha=1)),
    ('lr', LogisticRegression(max_iter=500))
])
```

My pipeline began with a TfidfVectorizer that turned the text into a matrix that emphasized more important words in the dataset. For each score, the pipeline would refit based on a binary class (if score = x then 1, else 0). This would give a probability for each score value that could be used as more features.



nb_text_prob_1.0	nb_summary_prob_1.0	nb_text_prob_2.0	nb_summary_prob_2.0	nb_text_prob_3.0	nb_summary_prob_3.0	nb_text_prob_4.0	nb_summary_prob_4.0	nb_text_prob_5.0	nb_summary_prob_5.0
0.019	0.165	0.078	0.026	0.118	0.080	0.360	0.251	0.345	0.500
0.088	0.056	0.053	0.032	0.069	0.084	0.194	0.307	0.604	0.509
0.059	0.019	0.097	0.087	0.159	0.135	0.254	0.211	0.381	0.617
0.046	0.118	0.086	0.071	0.101	0.196	0.164	0.212	0.485	0.297
0.066	0.092	0.158	0.073	0.178	0.222	0.288	0.234	0.279	0.427
0.030	0.007	0.026	0.007	0.065	0.091	0.222	0.443	0.652	0.465
0.009	0.013	0.011	0.039	0.038	0.111	0.155	0.403	0.831	0.453
0.016	0.015	0.022	0.018	0.069	0.037	0.175	0.051	0.710	0.906
0.089	0.069	0.144	0.061	0.345	0.107	0.157	0.213	0.096	0.519
0.019	0.011	0.040	0.010	0.044	0.094	0.186	0.316	0.665	0.495
0.040	0.050	0.088	0.064	0.175	0.126	0.448	0.215	0.292	0.533
0.014	0.009	0.067	0.008	0.234	0.019	0.333	0.211	0.221	0.815
0.807	0.045	0.181	0.058	0.068	0.108	0.079	0.213	0.145	0.534
0.009	0.010	0.014	0.041	0.083	0.380	0.207	0.330	0.731	0.237
0.009	0.098	0.025	0.069	0.071	0.219	0.201	0.259	0.685	0.316
0.043	0.147	0.019	0.240	0.043	0.329	0.166	0.210	0.768	0.151
0.045	0.070	0.081	0.151	0.102	0.115	0.165	0.209	0.414	0.414
0.713	0.839	0.560	0.094	0.301	0.039	0.143	0.035	0.050	0.012
0.116	0.015	0.054	0.034	0.053	0.052	0.153	0.415	0.570	0.436
0.001	0.008	0.006	0.011	0.024	0.039	0.305	0.160	0.835	0.863

Training

With these features, I trained a random forest classifier. I also tried a KNN at first but it was much weaker, so I stuck with a random forest. I then increased my dataset to 240,000 training entries and 60,000 test entries. This got me an accuracy of .64.

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred = rf_model.predict(X_test)

# Calculate accuracy and print a classification report
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.64625
Classification Report:
              precision    recall  f1-score   support

         1.0       0.57      0.57      0.57      3684
         2.0       0.40      0.25      0.30      3622
         3.0       0.46      0.37      0.41      7113
         4.0       0.48      0.36      0.41     13541
         5.0       0.74      0.88      0.81     32040

    accuracy                           0.65     60000
   macro avg       0.53      0.48      0.50     60000
weighted avg       0.62      0.65      0.62     60000
```

Next, I created all my features on the test submission and then put it on kaggle. If I had more time, I would mess around with more of the specific parameters and maybe give it more data. I would also try to make it better at guessing 3 and 4 star reviews. One way would be to get rid of the “nb_summary_prob_5.0”. My model overwhelmingly guessed 5 stars, which resulted in many 4 star reviews being classified as 5 stars.