General Idea

I tested several mainly used models in machine learning as the classifier. For the features used in text processing, I use the ones suggested in this site **link**. I tested some of them. And the numerical data in the dataset also contains much information, I set some numerical features, too. Since the limitation of the computational resources, I apply a truncated SVD dimension reduction before test. After finding the best combination of features and classifiers in the sample with fewer dimensions, I finally test its performance in full dataset and get the final result.

Features Tested

1. Text Features

- **TF-IDF**: I used the TF-IDF (Term Frequency-Inverse Document Frequency) method from *sklearn.feature_extraction.text.TfidfVectorizer* to extract features from the text of the reviews. TF-IDF helps identify important words by comparing their frequency in one review with their overall presence in other reviews. I limited the maximum number of features to 5000 to make the model more efficient.
- **N-grams**: I used *sklearn.feature_extraction.text.CountVectorizer* to generate 2-grams (pairs of words) and capture short sequences of words. This helps the model consider phrases instead of individual words, which can provide better context. I limited the maximum number of features to 3000 for N-grams.

2. Numerical Features

- **Helpfulness Ratio**: I calculated the helpfulness ratio by dividing the number of users who found a review helpful (HelpfulnessNumerator) by the total number of users who rated its helpfulness (HelpfulnessDenominator). This was done using basic Python calculations. I added 1 to the denominator to avoid division by zero. This feature shows how useful a review was to other customers.
- **Review Year**: I extracted the year from the timestamp of each review using pandas.to_datetime. Older reviews may have different patterns compared to newer ones, so this feature helps the model understand the time factor.
- **Other Features**: I also extracted product and user review counts by grouping the data using pandas.groupby on the ProductId and UserId. These features represent how active a user is or how popular a product is.

Models Tested

1. Naive Bayes

I used *sklearn.naive_bayes.MultinomialNB* to build the Naive Bayes model. This model is efficient for text classification tasks and works well with sparse features like TF-IDF. After training the model, I evaluated it using accuracy, precision, recall, F1 score, and confusion matrix from *sklearn.metrics*.

2. Logistic Regression

For the Logistic Regression model, I used *sklearn.linear_model.LogisticRegression* with a maximum iteration limit of 1000 to ensure convergence. Logistic Regression handles sparse text features like TF-IDF well and can be used for classification.

3. Random Forest

I used *sklearn.ensemble.RandomForestClassifier* to build a Random Forest model. Random Forest can handle both numerical and text features efficiently. It is also useful for ranking feature importance. I used 100 trees (n_estimators=100) to balance between performance and efficiency.

Key Steps in the Code

1. **Data Preprocessing**: I first processed the data to fill missing values and extract useful features. For text features, I used TF-IDF and N-grams. For numerical features, I calculated helpfulness ratio and extracted the review year.
2. **Feature Combination**: I combined all selected features (TF-IDF, N-grams, sentiment, and numerical) using *scipy.sparse.hstack*. This allowed me to test different combinations of features easily.
3. **Dimension Reduction:** To handle the high dimensionality of the combined feature matrix, I applied Truncated SVD for dimensionality reduction. I reduced the feature space to 30 dimensions. I used the package *sklearn.decomposition.TruncatedSVD* to implement this.
4. **Model Training**: For each model, I split the data into training and testing sets. I used *model.fit* to train the model and *model.predict* to generate predictions on the test set.
5. **Evaluation**: After training each model, I used *sklearn.metrics* to calculate accuracy, precision, recall, F1 score, and confusion matrix to evaluate the model's

performance. This gave me insights into how well the model predicted the star ratings.

Result

I found that use all features I selected will have a better result compared to use the combination of some of them.

For the models I applied, my plan was to first use the data after truncated SVD to test which model is the best, then train a model using the full dataset. The result turns out that the random forest is the best. But the training of random forest takes too long, so I train full dataset on linear regression instead. So the final best result comes from linear regression.