Midterm Report

This report outlines the development of a machine learning pipeline used to predict review scores from product data. I explored multiple machine learning models, including K-Nearest Neighbors (KNN), Logistic Regression, and Random Forest Classifiers. Throughout the process, I applied several feature engineering techniques and optimizations to improve the model's performance, aiming for better accuracy and computational efficiency. The final solution also incorporates data scaling, dimensionality reduction with PCA, and hyperparameter tuning through cross-validation. This writeup will walk through the core logic behind the model selection, key challenges faced, and the improvements implemented to enhance both accuracy and runtime.

Problem Understanding and Initial Observations:

The problem centers around predicting a numerical score (1–5) for product reviews based on user feedback. A challenge in this task lies in the sparse and noisy nature of the text data, combined with metadata such as user behavior and product identifiers. During our exploratory data analysis, several patterns emerged:

1. **Product reviews tend to have varying lengths**: Some users leave short reviews, while others provide lengthy ones. Longer reviews might contain more sentiment-related clues, suggesting that **Text Length** could be a useful feature.
2. **Helpfulness ratios** (from the HelpfulnessNumerator and HelpfulnessDenominator) indicated the social perception of a review's value, which might correlate with the score.
3. **Temporal factors** such as the year of the review might reveal trends in product performance or reviewer behavior over time.
4. **User and Product Frequencies**: Repeated reviews from the same user or product might reflect biases or specific usage trends.

These patterns informed the core features used in our model. However, efficiently using the textual data was critical, especially for sentiment analysis, as raw text features could quickly introduce dimensionality issues. With these observations in mind, we designed a feature extraction strategy along with model selection and optimization steps.

Feature Engineering and Data Preprocessing:

To effectively utilize the raw data, I introduced new features that captured both sentiment-related and metadata-driven patterns. Below are the key features engineered to improve model performance:

- **Helpfulness Ratio**: Helps quantify how valuable other users found a review. Missing values (when the denominator is 0) were replaced with 0.
- **Text Length and Word Count**: Captured the amount of information provided in the review. Longer reviews generally carry more sentiment, making this a valuable predictor.
- **Sentiment Polarity**: Used the TextBlob library to extract the sentiment polarity of each review. This feature quantifies the positivity or negativity of the review and proved useful for predicting scores.

- **Product and User Frequencies**: High-frequency users and products might show behavioral patterns, which could influence scores. Frequency-encoded ProductId and UserId to reduce dimensionality and capture these patterns.
- **Year of Review**: Extracting the year from the timestamp helped capture any temporal trends that might influence reviews over time.

After engineering these features, I split the data into training and test sets using a **stratified split** to ensure an even distribution of review scores. Also scaled the features using **StandardScaler** to improve model convergence during training.

4. Model Selection and Optimization
I initially tested three models: **K-Nearest Neighbors (KNN)**, **Logistic Regression**, and **Random Forest Classifier**. Each model has unique strengths:

- **KNN**: A simple, non-parametric model that can capture local relationships between data points. However, it suffers from computational inefficiency on large datasets.

- **Logistic Regression**: A linear model suitable for classification tasks. It works well with standardized data and is efficient to train but might struggle with non-linear relationships.

- **Random Forest**: An ensemble model that builds multiple decision trees and aggregates their predictions. It handles non-linearity well and can capture complex patterns in the data.

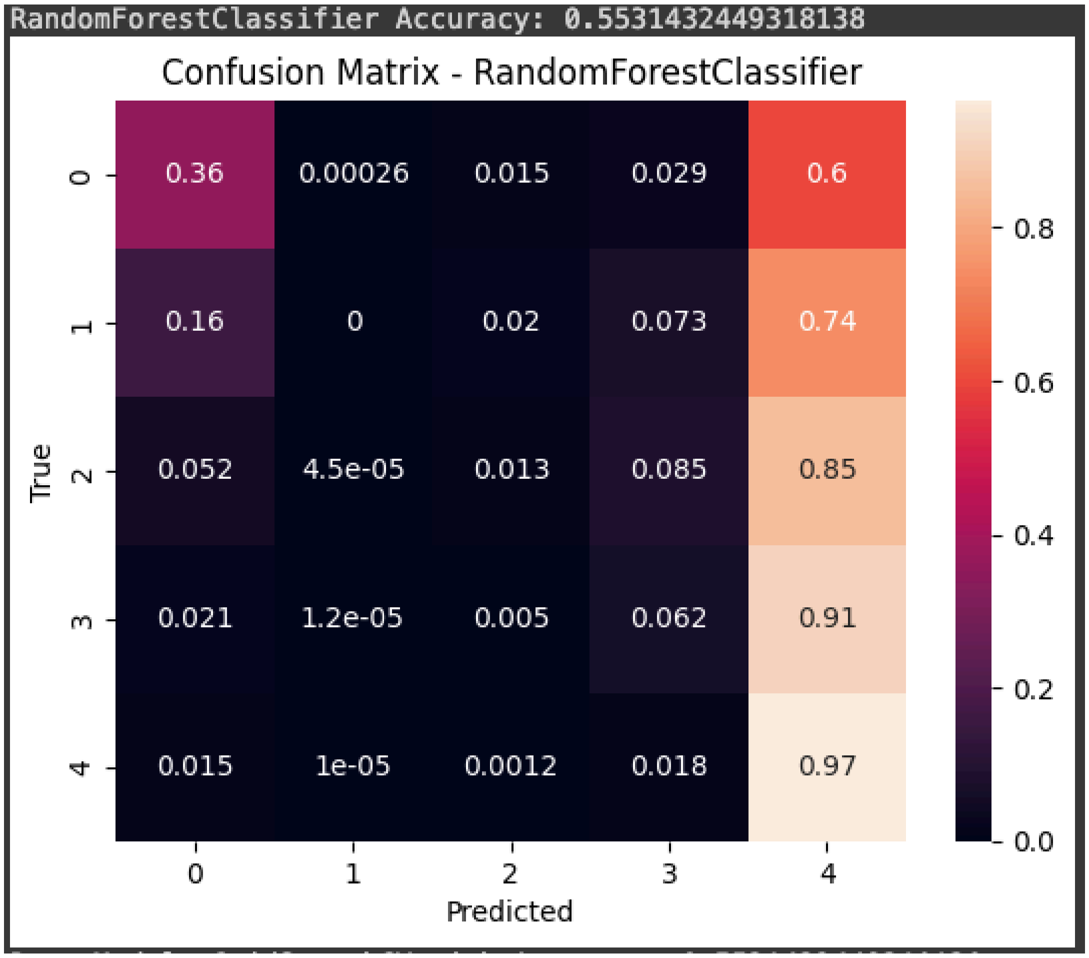
4.1 Dimensionality Reduction with PCA
Given the number of engineered features, I implemented **Principal Component Analysis (PCA)** to reduce dimensionality. PCA helps eliminate redundant information, speeds up training, and prevents overfitting. I optimized PCA to retain **95% of the variance**, ensuring that it didn't lose too much information while reducing the feature space.

4.2 Hyperparameter Tuning
I used **GridSearchCV** with **Stratified K-Folds Cross-Validation** to find the best hyperparameters for each model. Cross-validation ensured that the models generalized well to unseen data. For instance, I tuned the number of neighbors and weights for KNN, the regularization parameter C for Logistic Regression, and the number of estimators and depth for Random Forest.

4.3 Parallelization with joblib
To improve training speed, I employed **parallel processing** using the joblib library. This allowed us to run the grid searches for each model in parallel, significantly reducing training time.

5. Results and Patterns Noticed

After running our models with the tuned hyperparameters, the **Random Forest Classifier** emerged as the best-performing model, with an accuracy of 0.55314 on the test set.



```
RandomForestClassifier Accuracy: 0.5531432449318138
```

	0	1	2	3	4
0	0.36	0.00026	0.015	0.029	0.6
1	0.16	0	0.02	0.073	0.74
2	0.052	4.5e-05	0.013	0.085	0.85
3	0.021	1.2e-05	0.005	0.062	0.91
4	0.015	1e-05	0.0012	0.018	0.97

KNN, while conceptually simple, performed poorly on larger datasets due to its computational inefficiency. Logistic Regression, though fast and interpretable, could not fully capture the non-linear relationships in the data. The **Random Forest Classifier** balanced both accuracy and speed, benefiting from its ability to model complex patterns in the features.