I. Introduction

This project aims to predict **star ratings** (1 to 5 stars) associated with user reviews from **Amazon Movie Reviews**. The task involves building a machine learning model capable of analyzing both **textual content** and **numerical metadata** to make accurate predictions. Our primary challenge was handling **class imbalance**, **high-dimensional data**, and **noisy text reviews**.

During this project, I found that some techniques beyond those covered in the course were necessary to achieve competitive performance. While I learned **Support Vector Machines (SVMs)**, **Singular Value Decomposition (SVD)**, and **classification techniques** in the course, applying them directly wasn't sufficient for the complexity of this task. Thus, I independently explored additional methods, including:

- **TF-IDF Vectorization**: To convert text into numerical features.
- **Polynomial Feature Expansion**: To capture non-linear interactions between numerical features.
- **Hyperparameter Tuning (GridSearchCV)**: To optimize model performance effectively.
- **Stemming with NLTK**: To reduce text dimensionality by converting words to their base forms.

These techniques were chosen to address the specific challenges of text-heavy, high-dimensional data, such as noisy reviews and **class imbalance**. Learning these methods involved independent research through **documentation, online tutorials,** and **self-study of libraries such as Scikit-learn and NLTK**.

Ultimately, I chose **LinearSVC** as the primary classifier due to its efficiency with high-dimensional sparse data and ease of interpretability. This report details the **preprocessing steps, feature engineering**, and **model tuning techniques** that contributed to our final solution.

II. Data Understanding and Preprocessing

The dataset consists of two files:

- **train.csv**: Contains over 1.6 million reviews with metadata and associated scores (1-5 stars).
- **test.csv**: Contains review IDs for which I need to predict the star ratings.

Data Fields Overview

- **Id**: Unique identifier for each review.
- **Summary**: A short summary of the review.
- **Text**: The full text of the review.
- **Score**: The star rating (target variable) ranging from 1 to 5.
  Note: This field is missing in the test set, and I need to predict these values.

Key Challenges Identified

1. **Class Imbalance**:
   The distribution of star ratings is heavily skewed towards 5-star reviews, which poses a risk of the model being biased towards the majority class.
2. **High-Dimensional Text Data**:
   Reviews contain unstructured text that needs to be vectorized, which results in high-dimensional feature space, making it harder to generalize without overfitting.
3. **Missing Values**:
   Some reviews are missing either the Summary or Text field.
   These need to be filled appropriately to avoid loss of information.

Preprocessing Steps

1. **Combining Summary and Text Fields**:
   I merged these two fields into **Combined_Text** to capture all available information.

```
train['Combined_Text'] = train['Summary'].fillna('') + ' ' + train['Text'].fillna('')
test['Combined_Text'] = test['Summary'].fillna('') + ' ' + test['Text'].fillna('')
```

2. **Handling Missing Scores in Training Data**:
   Rows with missing Score values were removed, ensuring the model trains on complete data.

```
train = train.dropna(subset=['Score'])
train['Score'] = train['Score'].astype(np.int8)
```

3. **Sampling for Efficiency**:
   To reduce processing time, I randomly sampled **30,000 reviews** for training.

```
train = train.sample(30000, random_state=42)
```

4. **Stemming**:
   I applied **stemming** using NLTK's SnowballStemmer to reduce words to their root form, minimizing vocabulary size.

```
stemmer = SnowballStemmer("english")
train['Combined_Text'] = train['Combined_Text'].apply(
    lambda x: ' '.join(stemmer.stem(word) for word in x.split())
)
```

III. Feature Engineering

- **Numerical Features Extracted:**
  - **Review_Length: Total character count.**
  - **Word_Count: Total number of words.**
  - **Exclamation_Count: Number of exclamation marks.**
  - **Uppercase_Word_Count: Number of words in uppercase.**
  - **Exclamation_Word_Ratio: Ratio of exclamation marks to word count.**

```
# Feature Engineering
def feature_engineering(df):
    df['Review_Length'] = df['Combined_Text'].str.len()
    df['Word_Count'] = df['Combined_Text'].str.split().str.len()
    df['Exclamation_Count'] = df['Combined_Text'].str.count('!')
    df['Uppercase_Word_Count'] = df['Combined_Text'].apply(lambda x: sum(1 for word in x.split() if word.isupper()))
    df['Exclamation_Word_Ratio'] = df['Exclamation_Count'] / (df['Word_Count'] + 1e-6)
    return df
```

- **Polynomial Feature Expansion**:
  - I expanded numerical features using **degree-3 polynomial features** to capture non-linear relationships.

```
poly = PolynomialFeatures(degree=3, interaction_only=True, include_bias=False)
X_train_poly = poly.fit_transform(X_train_num)
```

- **TF-IDF Vectorization:**
  I used TF-IDF with unigrams and bigrams to represent the text.

```
vectorizer = TfidfVectorizer(
    max_features=5000, ngram_range=(1, 2), stop_words='english',
    min_df=2, max_df=0.85, sublinear_tf=True
)
```

IV. Model Selection and Tuning

- **LinearSVC Model:**
  LinearSVC is suitable for high-dimensional data. I assigned class weights to handle class imbalance.

```
class_weights = {1: 4, 2: 3, 3: 2, 4: 1.5, 5: 1}
model = LinearSVC(class_weight=class_weights, random_state=42, max_iter=3000)
```

- **Hyperparameter Tuning:**

  I used GridSearchCV to tune the C parameter, evaluating on 5-fold Stratified Cross-Validation.

```
param_grid = {'C': [0.01, 0.1, 1, 10]}
grid_search = GridSearchCV(model, param_grid, cv=StratifiedKFold(5), scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_split, y_train_split)
```

V. Model Evaluation

- **Validation Accuracy**:
  - Achieved a **validation accuracy of ~60.48%** on the split validation set.
- **Confusion Matrix**:
  - Analyzed the confusion matrix to ensure the model is not heavily biased towards a single class.
- **Cross-Validation Score**:
  - Cross-validation accuracy: **~60.40%**, showing stability across multiple splits.

VI. Conclusion and Future Work

The final model effectively combines TF-IDF vectorization and numerical feature engineering, achieving a competitive validation accuracy. Using LinearSVC allowed us to handle high-dimensional text efficiently, while GridSearchCV fine-tuned the hyperparameters.

Key Insights:

- Exclamation marks and uppercase words correlate with emotional content, influencing star ratings.
- Class imbalance was addressed through weighted class adjustments, but predicting minority classes remains challenging.

Future Improvements:

- Explore other algorithms, such as RandomForest or XGBoost.
- Experiment with sentiment analysis features to improve predictions.
- Use ensemble models to combine multiple algorithms for better performance.

X. Citations and Learning Sources

1. **Scikit-learn Documentation(https://scikit-learn.org):**
   a) TF-IDF Vectorization, GridSearchCV, LinearSVC.
2. **NLTK Documentation(https://www.nltk.org/):**
   a) SnowballStemmer for stemming.
3. **Online Tutorials and Research Articles:**
   a) Polynomial feature engineering and text classification strategies.