UCS2604 - Principles of Machine Learning
MINI PROJECT(with cloud deployment)

TOPIC: AMAZON REVIEW RATING PREDICTION USING MACHINE LEARNING

Harine Cellam NS - 3122225001034
Harinishree K - 3122225001035
Hemalatha R - 3122225001039

A. ABSTRACT

E-commerce platforms rely heavily on customer reviews and ratings to enhance user experience and drive sales. However, challenges such as accurate rating calculations and the ranking of product reviews persist. Incorrect rating computations can affect customer satisfaction, product visibility, and business performance. Additionally, misleading or unverified reviews may result in financial and customer losses.

This dataset focuses on Amazon product reviews and ratings, including metadata such as review text, rating scores, and helpfulness votes. It aims to facilitate the development of robust models for ranking reviews accurately and ensuring fair product evaluations. By analyzing this data, e-commerce platforms can enhance customer trust, improve product recommendations, and optimize sales strategies.

B.INTRODUCTION

Customer reviews and ratings play a vital role in e-commerce, influencing buyer decisions and product visibility. However, ratings may not always accurately reflect product quality due to biased or misleading reviews.

This project focuses on predicting product ratings based on customer feedback by preprocessing review data and training various machine learning models. The goal is to compare model performance and accuracy to identify the most effective approach for rating prediction.

The dataset consists of Amazon product reviews, including review texts, ratings, helpfulness scores, and metadata. Different models are evaluated to determine their effectiveness in predicting ratings based on review data.

i) PROBLEM STATEMENT: Amazon Review Rating Prediction Using ML

With the rapid growth of e-commerce platforms, customer reviews play a crucial role in shaping consumer decisions and business strategies. Amazon, one of the largest online marketplaces, allows users to provide feedback in the form of text reviews and numerical ratings (1 to 5 stars). However, due to the sheer volume of reviews, manually analyzing and summarizing customer sentiment becomes impractical.

This project focuses on developing a Machine Learning-based model for predicting Amazon review ratings based on textual feedback and other relevant features. The primary objective is to automate the rating prediction process, enabling businesses to gain insights into customer satisfaction levels efficiently.

C.LITERATURE SURVEY:

1.Sentiment Analysis for E-commerce Product Reviews: Current Trends and Future Directions

Numerous goods and services are now offered through online platforms due to the recent growth of online transactions like e-commerce. Users have trouble locating the product that best suits them from the numerous products available in online shopping. Many studies in deep learning-based recommender systems (RSs) have focused on the intricate relationships between the attributes of users and items. Deep learning techniques have used consumer or item-related traits to improve the quality of personalized recommender systems in many areas, such as tourism, news, and e-commerce. Various companies, primarily e-commerce, utilize sentiment analysis to enhance product quality and effectively navigate today's business environment. Customer feedback regarding a product is gathered through sentiment analysis, which uses contextual data to split it into separate polarities. The explosive rise of the e-commerce industry has resulted in a large body of literature on e-commerce from different perspectives. Researchers have made an effort to categorize the recommended future possibilities for e-commerce study as the field has grown. There are several challenges in e-commerce, such as fake reviews, frequency of user reviews, advertisement click fraud, and code-mixing. In this review, we introduce an overview of the preliminary design for e-commerce. Second, the concept of deep learning, e-commerce, and sentiment analysis are discussed. Third, we represent different versions of the commercial dataset. Finally, we explain various difficulties facing RS and future research directions.[1][2]

2.Predicting product review helpfulness using machine learning and specialized classification models

In this paper we focus on automatically classifying product reviews as either helpful or unhelpful using machine learning techniques, namely, SVM classifiers. Using LIBSVM and a set of Amazon product reviews from 25 product categories, we train models for each category to determine if a review will be helpful or unhelpful. Previous work has focused on training one classifier for all reviews in the data set, but we hypothesize that a distinct model for each of the 25 product types available in the review dataset will improve the accuracy of classification.! Furthermore, we develop a framework to inform authors on the fly if their review is predicted to be of great use (helpful) to other readers, with the assumption that authors are more likely to rethink their review post and amend it to be of maximum utility to other readers when given some feedback on whether or not it will be found helpful or unhelpful.! Using past research as a baseline, we find that specialized SVM classifiers outperform higher level models of review helpfulness prediction.[3][4]

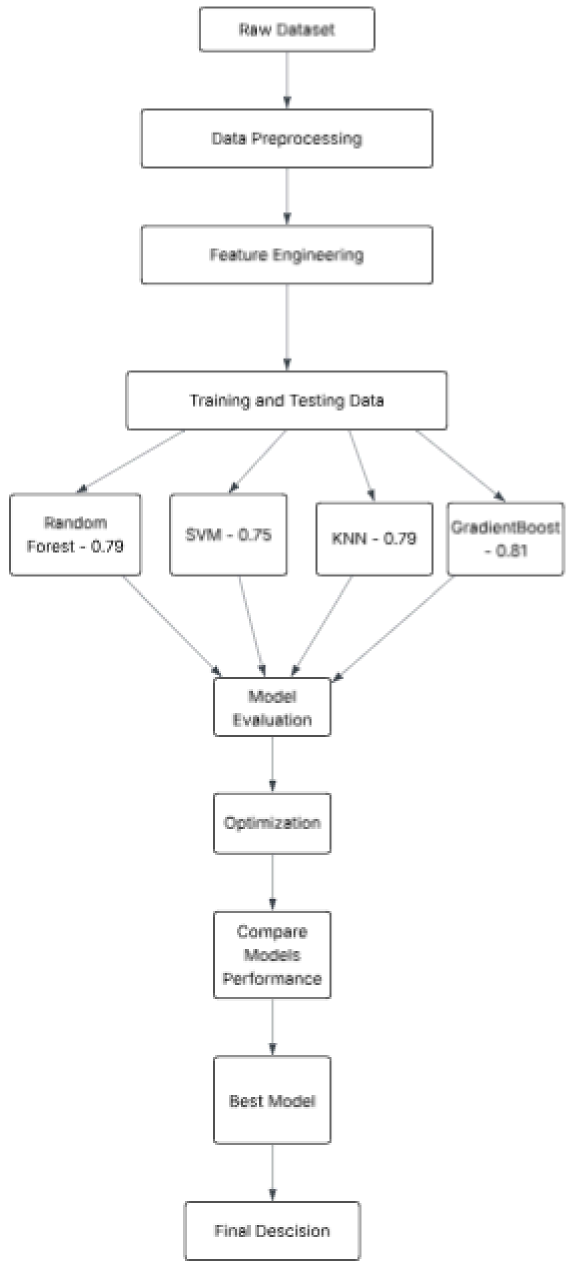
3.Amazon review classification and sentiment analysis

Reviews on Amazon are not only related to the product but also the service given to the customers. If users get clear bifurcation about product reviews and service reviews it will be easier for them to take the decision, in this paper we propose a system that performs the classification of customer reviews followed by finding sentiment of the reviews. A rule based extraction of product feature sentiment is also done. Also we provide a visualization for our result summarization.[5][6]

Relevance to Our Work (Amazon Review Analysis & Rating Prediction)

Our project aims to predict numerical ratings from Amazon reviews, differing from the helpfulness classification focus

D. Architecture Diagram



–

Figure 1

This figure illustrates the workflow architecture for building and evaluating multiple machine learning models for classification. The process follows a structured pipeline, divided into the following key stages:

1. Data Preprocessing

- The pipeline begins with the Raw Dataset, which undergoes Data Preprocessing to clean and prepare the data for feature extraction.

2. Feature Engineering

- After preprocessing, the dataset undergoes Feature Extraction and Feature Engineering to enhance the data representation, making it suitable for model training.

3. Splitting Data

- The processed data is divided into Training & Testing Data, ensuring that the models are evaluated on unseen samples.

4. Model Training

- Four machine learning models are trained in parallel:
  Model 1: Random Forest Classifier
  Model 2: Support Vector Machine (SVM)
  Model 3: Gradient Boosting
  Model 4: K-Nearest Neighbors (KNN)

5. Model Evaluation

- Each model is evaluated based on performance metrics such as accuracy, precision, recall, and F1-score.

6. Model Selection & Final Decision

- Based on the evaluation results, the best model is selected for deployment or further optimization.

This structured approach ensures that multiple models are considered, compared, and rigorously evaluated before making the final decision on the optimal classifier for the given dataset.

E.Implementation and Experiments

I) Development Environment

Programming Language: Python (v3.x)
Libraries & Frameworks: Pandas, NumPy,seaborn,
Scikit-learn,SKlearn.Preprocessing,SKlearn.Featureextraction,Matplotlib.
Integrated Development Environment (IDE): Google Colab notebook

II) Dataset Information:

Dataset: https://www.kaggle.com/datasets/tarkkaanko/amazon
Number of records: 4915
Columns: 12
Names of Features(Columns):

- reviewerName: Name of the reviewer
- overall: Overall rating given by the reviewer
- reviewText: The actual review text
- reviewTime: The date when the review was posted
- day_diff: The difference in days from the review date to the current date
- helpful_yes: Number of "helpful" votes
- helpful_no: Number of "not helpful" votes
- total_vote: Total votes received for the review
- score_pos_neg_diff: Difference between positive and negative votes
- score_average_rating: Average rating score
- wilson_lower_bound: Wilson score for ranking reviews based on helpfulness

III) Implementation:

Mount the Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Load the Dataset

```
import pandas as pd
import numpy as np
# Load dataset
df = pd.read_csv("/content/drive/MyDrive/amazon_reviews.csv")
df.head()
```

```
df.info()
```

	Unnamed: 0	reviewerName	overall	reviewText	reviewTime	day_diff	helpful_yes	helpful_no	total_vote	score_pos_neg_diff	score_average_rating	wilson_lower_bound
0	0	NaN	4.0	No issues.	2014-07-23	138	0	0	0	0	0.0	0.0
1	1	0mie	5.0	Purchased this for my device, it worked as adv...	2013-10-25	409	0	0	0	0	0.0	0.0
2	2	1K3	4.0	it works as expected. I should have sprung for...	2012-12-23	715	0	0	0	0	0.0	0.0
3	3	1m2	5.0	This think has worked out great.Had a diff. br...	2013-11-21	382	0	0	0	0	0.0	0.0
4	4	2&1/2Men	5.0	Bought it with Retail Packaging, arrived legit...	2013-07-13	513	0	0	0	0	0.0	0.0

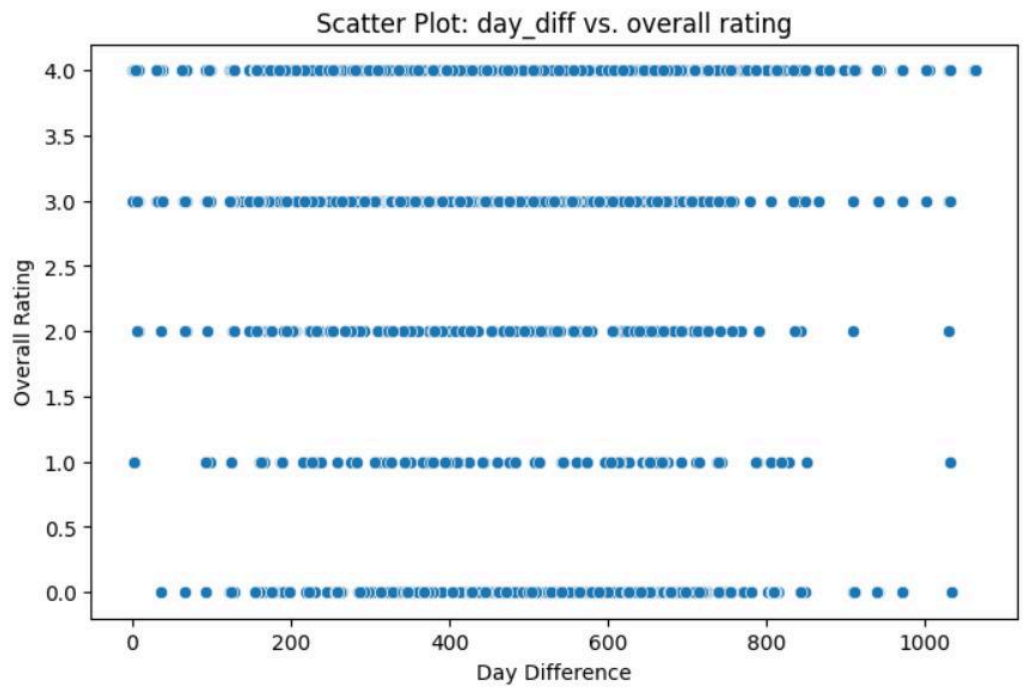
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4915 entries, 0 to 4914
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            4915 non-null   int64
 1   reviewerName          4914 non-null   object
 2   overall               4915 non-null   float64
 3   reviewText            4914 non-null   object
 4   reviewTime            4915 non-null   object
 5   day_diff              4915 non-null   int64
 6   helpful_yes           4915 non-null   int64
 7   helpful_no            4915 non-null   int64
 8   total_vote            4915 non-null   int64
 9   score_pos_neg_diff    4915 non-null   int64
 10  score_average_rating  4915 non-null   float64
 11  wilson_lower_bound    4915 non-null   float64
dtypes: float64(3), int64(6), object(3)
memory usage: 460.9+ KB
```

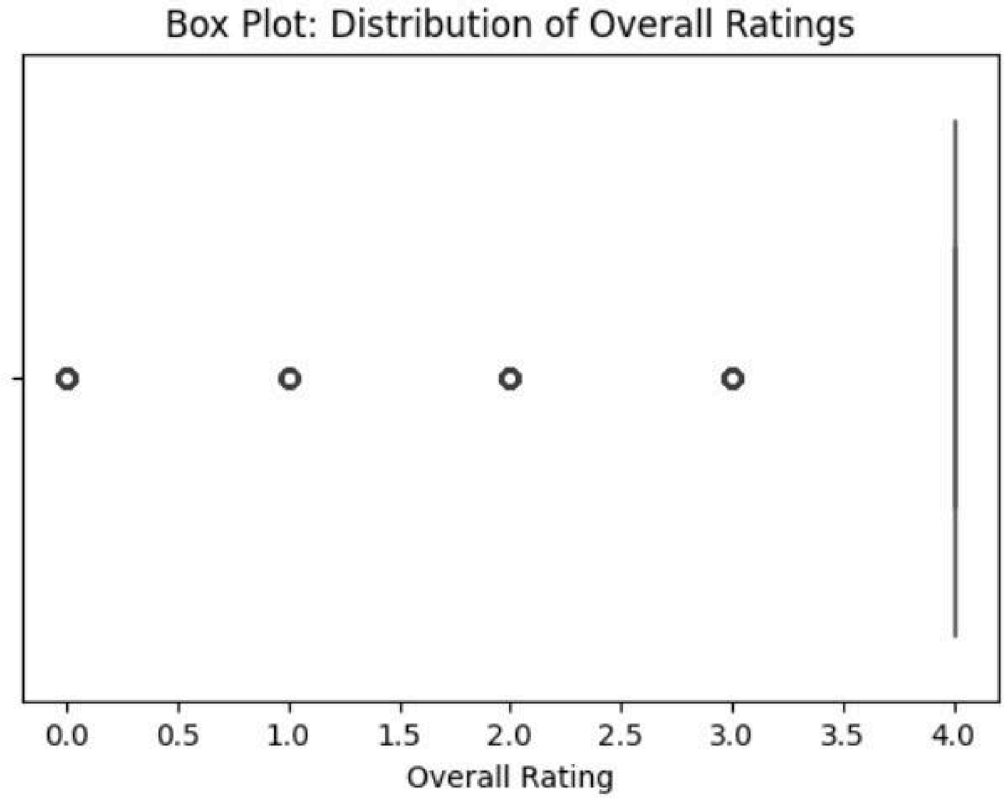
Exploratory Data Analytics

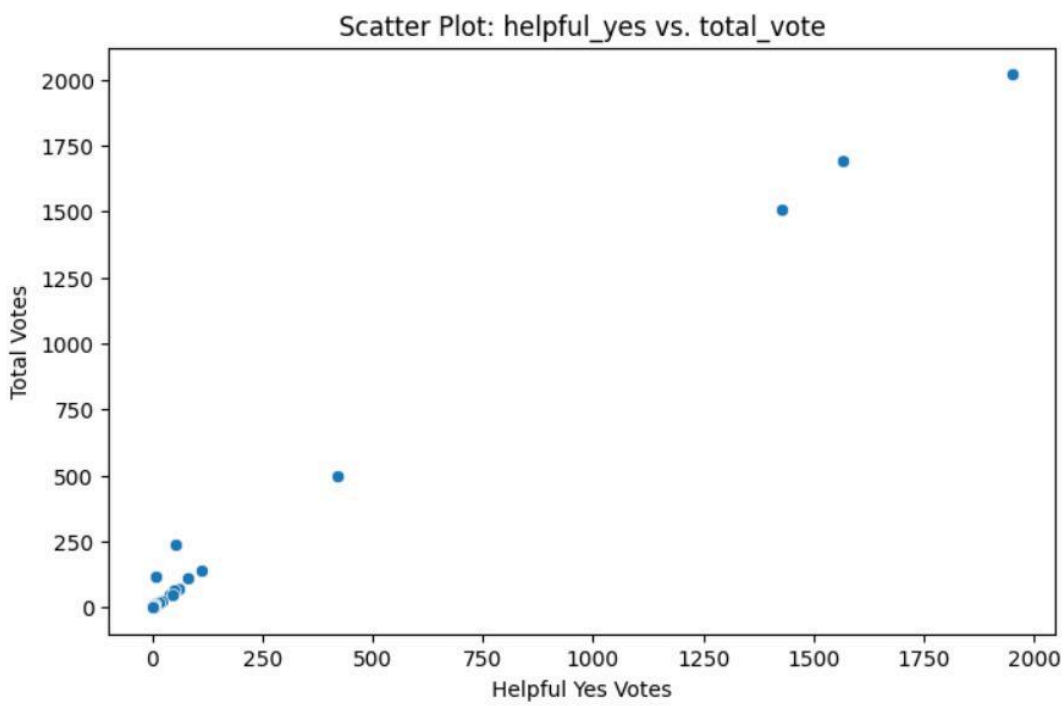
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Scatter Plot: day_diff vs. overall rating
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df['day_diff'], y=df['overall'])
plt.title('Scatter Plot: day_diff vs. overall rating')
plt.xlabel('Day Difference')
plt.ylabel('Overall Rating')
plt.show()
```

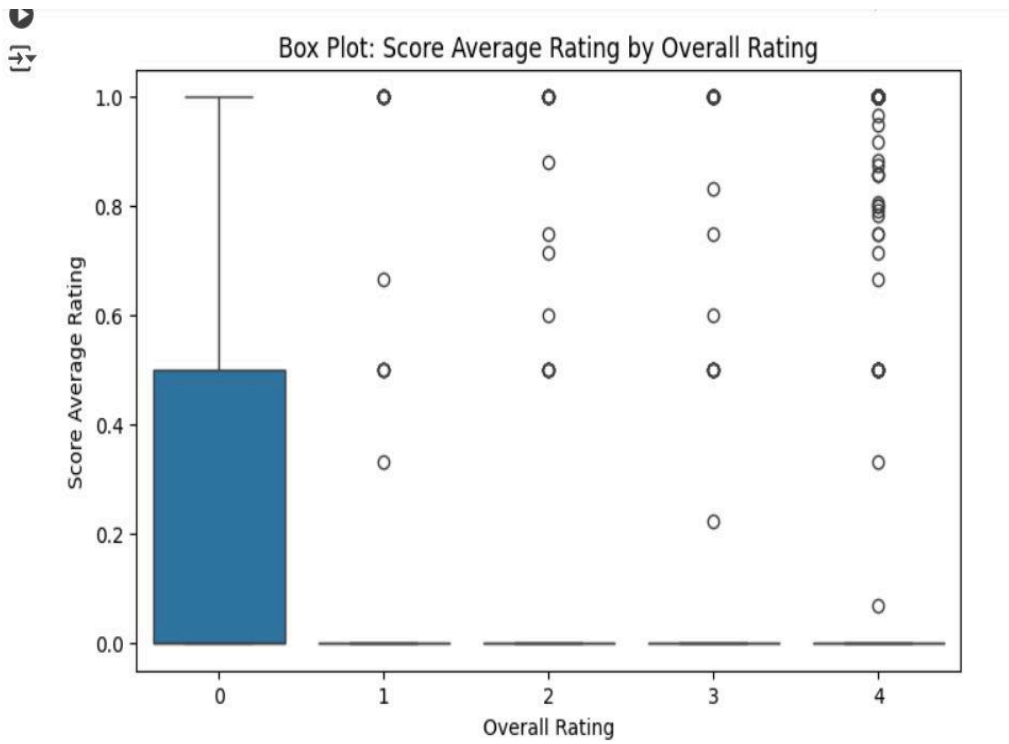
```
# Box Plot: Distribution of overall ratings
plt.figure(figsize=(6, 4))
sns.boxplot(x=df['overall'])
plt.title('Box Plot: Distribution of Overall Ratings')
plt.xlabel('Overall Rating')
plt.show()

# Scatter Plot: helpful_yes vs. total_vote
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df['helpful_yes'], y=df['total_vote'])
plt.title('Scatter Plot: helpful_yes vs. total_vote')
plt.xlabel('Helpful Yes Votes')
plt.ylabel('Total Votes')
plt.show()

# Box Plot: score_average_rating by overall rating
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['overall'], y=df['score_average_rating'])
plt.title('Box Plot: Score Average Rating by Overall Rating')
plt.xlabel('Overall Rating')
plt.ylabel('Score Average Rating')
plt.show()

# Histogram: day_diff distribution
plt.figure(figsize=(8, 5))
sns.histplot(df['day_diff'], bins=30, kde=True)
plt.title('Histogram: Distribution of Day Difference')
plt.xlabel('Day Difference')
plt.ylabel('Count')
plt.show()

# Correlation Matrix with Numeric Data Only
plt.figure(figsize=(10, 8))
corr_matrix = df.select_dtypes(include=['number']).corr()
```

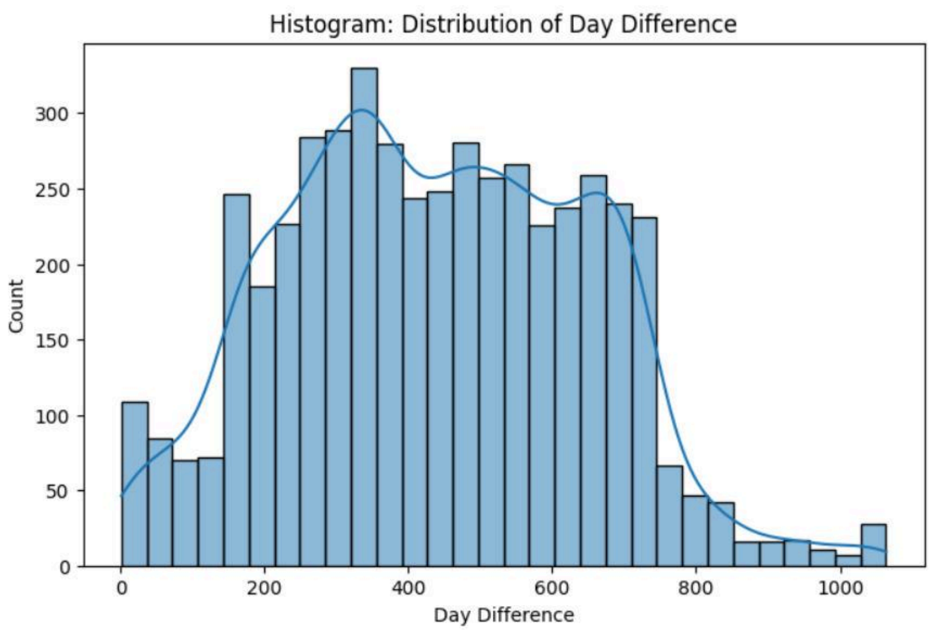
```python
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm',
linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

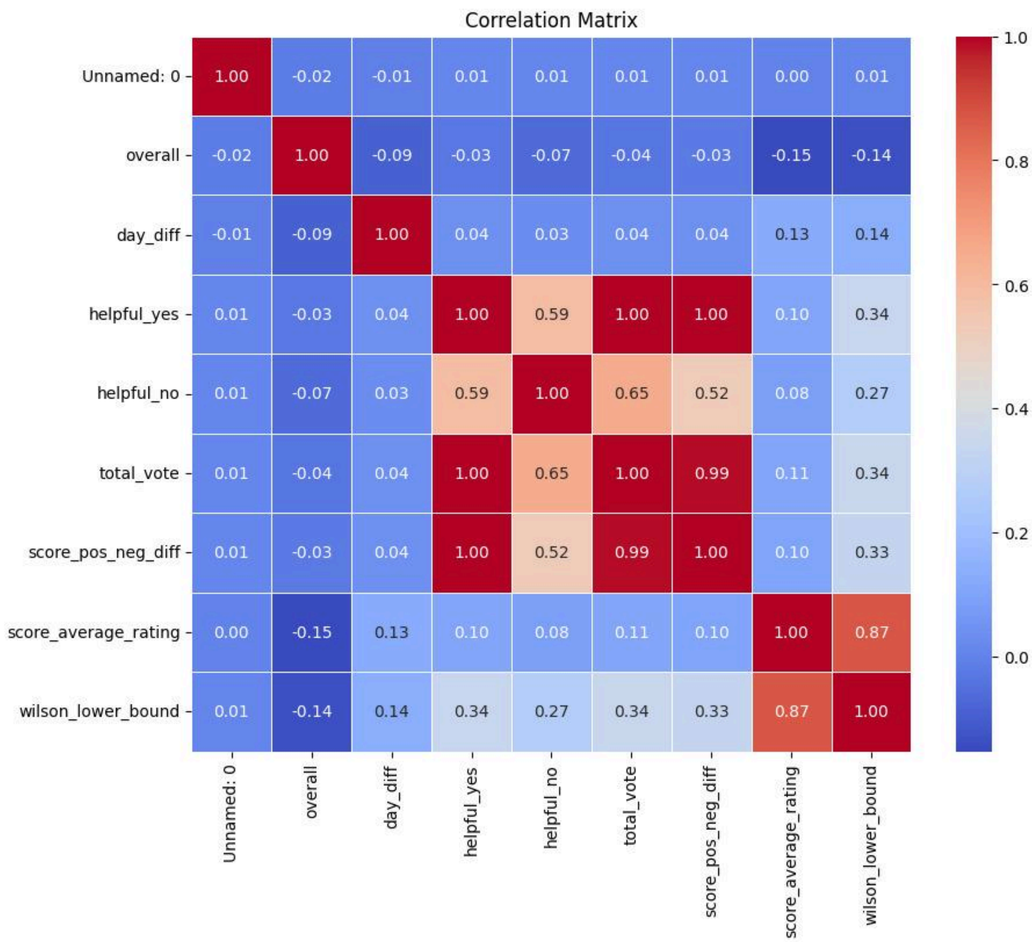



Box Plot: Distribution of Overall Ratings
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Overall Rating


Scatter Plot: helpful_yes vs. total_vote
Total Votes
2000
1750
1500
1250
1000
750
500
250
0
Helpful Yes Votes


Box Plot: Score Average Rating by Overall Rating
Score Average Rating
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
Overall Rating


Histogram: Distribution of Day Difference
Count
300
250
200
150
100
50
0
0
200
400
600
800
1000
Day Difference



Pre-Processing

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler, LabelEncoder

#Drop unnecessary columns
df.drop(columns=['reviewerName', 'reviewTime'], inplace=True)

#Handle missing values
df.fillna("", inplace=True)

#Convert text data (reviewText) to numerical features using
TF-IDF
```

```python
tfidf = TfidfVectorizer(max_features=500)
X_text = tfidf.fit_transform(df['reviewText']).toarray()

#Encode categorical values
label_encoder = LabelEncoder()
df['overall'] = label_encoder.fit_transform(df['overall'])

# Select numerical features
X_numeric = df[['day_diff', 'helpful_yes', 'total_vote',
'score_pos_neg_diff', 'score_average_rating',
'wilson_lower_bound']].values

# Scale numerical features
scaler = StandardScaler()
X_numeric_scaled = scaler.fit_transform(X_numeric)

# Combine text & numerical features
X = np.hstack((X_text, X_numeric_scaled))

# Target variable
y = df['overall']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print("After Preprocessing:")
df.info()
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
After Preprocessing:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4915 entries, 0 to 4914
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            4915 non-null   int64
 1   overall               4915 non-null   int64
 2   reviewText            4915 non-null   object
 3   day_diff              4915 non-null   int64
 4   helpful_yes           4915 non-null   int64
 5   helpful_no            4915 non-null   int64
 6   total_vote            4915 non-null   int64
 7   score_pos_neg_diff    4915 non-null   int64
 8   score_average_rating  4915 non-null   float64
 9   wilson_lower_bound    4915 non-null   float64
dtypes: float64(2), int64(7), object(1)
memory usage: 384.1+ KB
X_train shape: (3932, 506)
X_test shape: (983, 506)
y_train shape: (3932,)
y_test shape: (983,)
```

Model Training

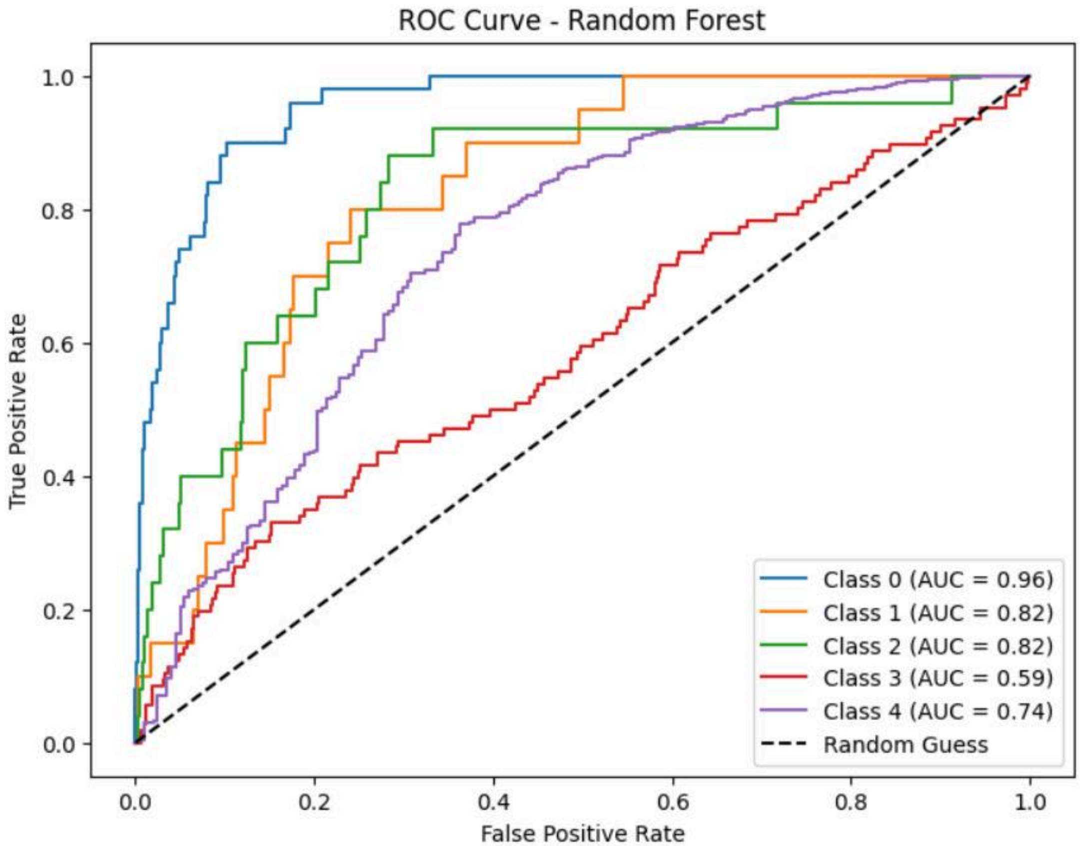
1.Random Forest Classifier

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score
from imblearn.over_sampling import SMOTE

# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)
```

```
# Train Random Forest model with optimized hyperparameters
rf_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    min_samples_split=5,
    min_samples_leaf=3,
    class_weight='balanced',
    random_state=42
)
rf_model.fit(X_train_balanced, y_train_balanced)

# Predict
y_pred_rf = rf_model.predict(X_test)

# Evaluate with zero_division set
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf,
average='macro', zero_division=1)
recall_rf = recall_score(y_test, y_pred_rf, average='macro',
zero_division=1)

print(f"Random Forest - Accuracy: {accuracy_rf:.2f}, Precision:
{precision_rf:.2f}, Recall: {recall_rf:.2f}")

# ROC Curve
y_test_binarized = label_binarize(y_test,
classes=np.unique(y_train_balanced))
y_score = rf_model.predict_proba(X_test)

plt.figure(figsize=(8, 6))
for i in range(y_test_binarized.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_score[:,
i])
    roc_auc = auc(fpr, tpr)
```

```
    plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend()
plt.show()
from sklearn.metrics import classification_report

#classification report
report_rf = classification_report(y_test, y_pred_rf,
zero_division=1)
print("Classification Report - Random Forest:\n", report_rf)
```

Random Forest - Accuracy: 0.79, Precision: 0.53, Recall: 0.32




```
Classification Report - Random Forest:
              precision    recall  f1-score   support

           0       0.54      0.52      0.53        50
           1       1.00      0.00      0.00        20
           2       0.00      0.00      0.00        25
           3       0.27      0.16      0.20       106
           4       0.84      0.94      0.89       782

    accuracy                           0.79       983
   macro avg       0.53      0.32      0.32       983
weighted avg       0.75      0.79      0.76       983
```

Figure 1.1

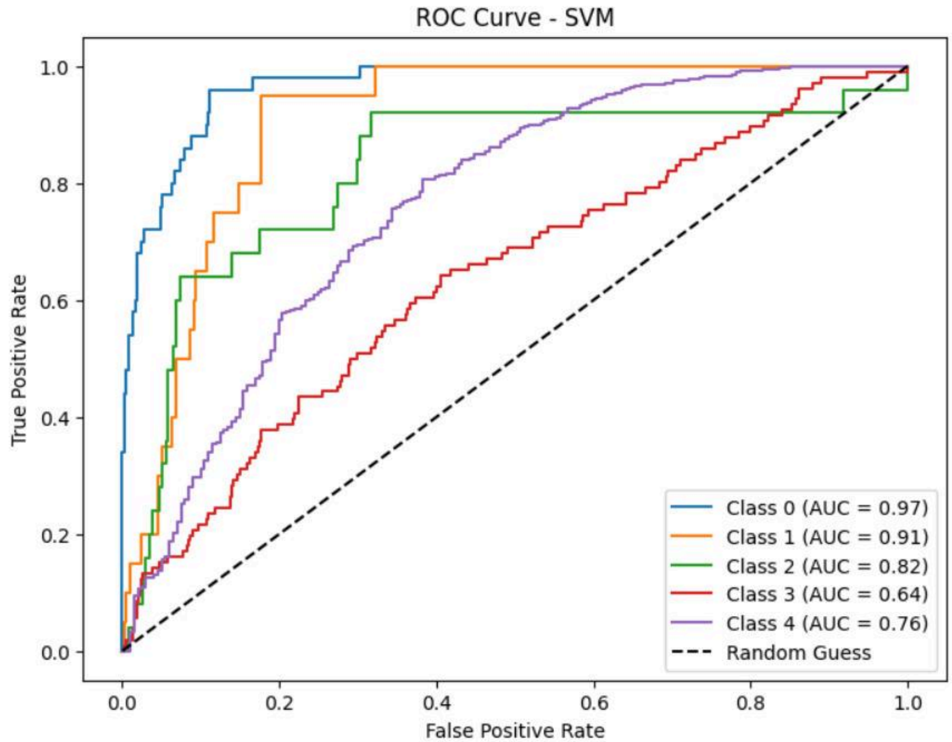
This figure 1.1 shows the performance of the Random Forest model in terms of ROC curves and classification metrics. The model achieves a 79% accuracy, but it struggles with certain classes, particularly Class 1 and Class 2, which have zero recall, meaning they are not correctly classified. Class 3 also has low recall and precision, suggesting that further improvements, such as data balancing or hyperparameter tuning, may be needed to enhance performance.

2.Support Vector Machine

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
recall_score
from imblearn.over_sampling import SMOTE

# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)

# Train SVM model with class balancing
svm_model = SVC(kernel='rbf', C=1.5, class_weight='balanced',
random_state=42, probability=True)
svm_model.fit(X_train_balanced, y_train_balanced)

# Predict
y_pred_svm = svm_model.predict(X_test)

# Evaluate with zero_division set
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm,
average='macro', zero_division=1)
```

```
recall_svm = recall_score(y_test, y_pred_svm, average='macro',
zero_division=1)

print(f"SVM - Accuracy: {accuracy_svm:.2f}, Precision:
{precision_svm:.2f}, Recall: {recall_svm:.2f}")

# ROC Curve
y_test_binarized = label_binarize(y_test,
classes=np.unique(y_train_balanced))
y_score = svm_model.predict_proba(X_test)
from sklearn.metrics import classification_report

# Generate classification report
report_svm = classification_report(y_test, y_pred_svm,
zero_division=1)
print("Classification Report - SVM:\n", report_svm)
plt.figure(figsize=(8, 6))
for i in range(y_test_binarized.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_score[:,
i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - SVM')
plt.legend()
plt.show()
```

SVM - Accuracy: 0.75, Precision: 0.36, Recall: 0.38



SVM - Accuracy: 0.75, Precision: 0.36, Recall: 0.38

Classification Report - SVM:

	precision	recall	f1-score	support
0	0.52	0.66	0.58	50
1	0.10	0.05	0.07	20
2	0.06	0.08	0.07	25
3	0.22	0.23	0.22	106
4	0.88	0.87	0.88	782
accuracy			0.75	983
macro avg	0.36	0.38	0.36	983
weighted avg	0.76	0.75	0.75	983

Figure 1.2

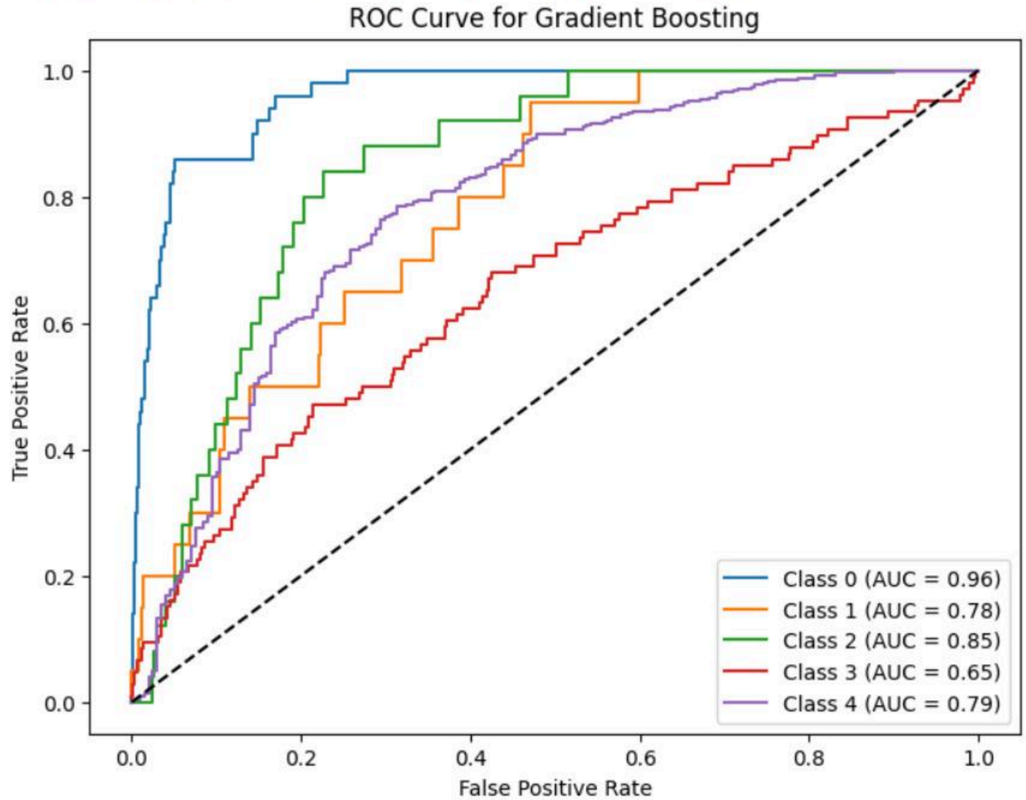
This figure 1.2 shows the performance of the SVM (Support Vector Machine) model in terms of ROC curves and classification metrics. The model achieves a 75% accuracy, but it struggles with certain classes, particularly Class 1 and Class 2, which have very low recall (0.05 and 0.08, respectively), indicating poor classification of these categories. Class 3 also has low recall and precision (0.23 and 0.22, respectively), suggesting further performance challenges.

3.Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize

# Train Gradient Boosting model
gb_model = GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, random_state=42)
gb_model.fit(X_train, y_train)

# Predict
y_pred_gb = gb_model.predict(X_test)
y_proba_gb = gb_model.predict_proba(X_test)  # Get probability
scores

# Evaluate
accuracy_gb = accuracy_score(y_test, y_pred_gb)
precision_gb = precision_score(y_test, y_pred_gb,
average='macro')
recall_gb = recall_score(y_test, y_pred_gb, average='macro')

print(f"Gradient Boosting - Accuracy: {accuracy_gb:.2f},
Precision: {precision_gb:.2f}, Recall: {recall_gb:.2f}")

# ROC Curve
```

```
plt.figure(figsize=(8, 6))

# If binary classification
if len(set(y_test)) == 2:
    fpr, tpr, _ = roc_curve(y_test, y_proba_gb[:, 1])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area =
{roc_auc:.2f})')

# If multi-class classification
else:
    y_test_bin = label_binarize(y_test,
classes=list(set(y_test)))  # Binarize labels
    for i in range(y_test_bin.shape[1]):
        fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba_gb[:,
i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'Class {i} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Gradient Boosting')
plt.legend()
plt.show()
# Generate classification report
report_gb = classification_report(y_test, y_pred_gb,
zero_division=1)
print("Classification Report - Gradient Boosting:\n", report_gb)
```

Gradient Boosting - Accuracy: 0.81, Precision: 0.46, Recall: 0.31



Classification Report - Gradient Boosting:

	precision	recall	f1-score	support
0	0.74	0.46	0.57	50
1	0.33	0.05	0.09	20
2	0.00	0.00	0.00	25
3	0.38	0.05	0.08	106
4	0.83	0.98	0.90	782
accuracy			0.81	983
macro avg	0.46	0.31	0.33	983
weighted avg	0.75	0.81	0.76	983

Figure 1.3

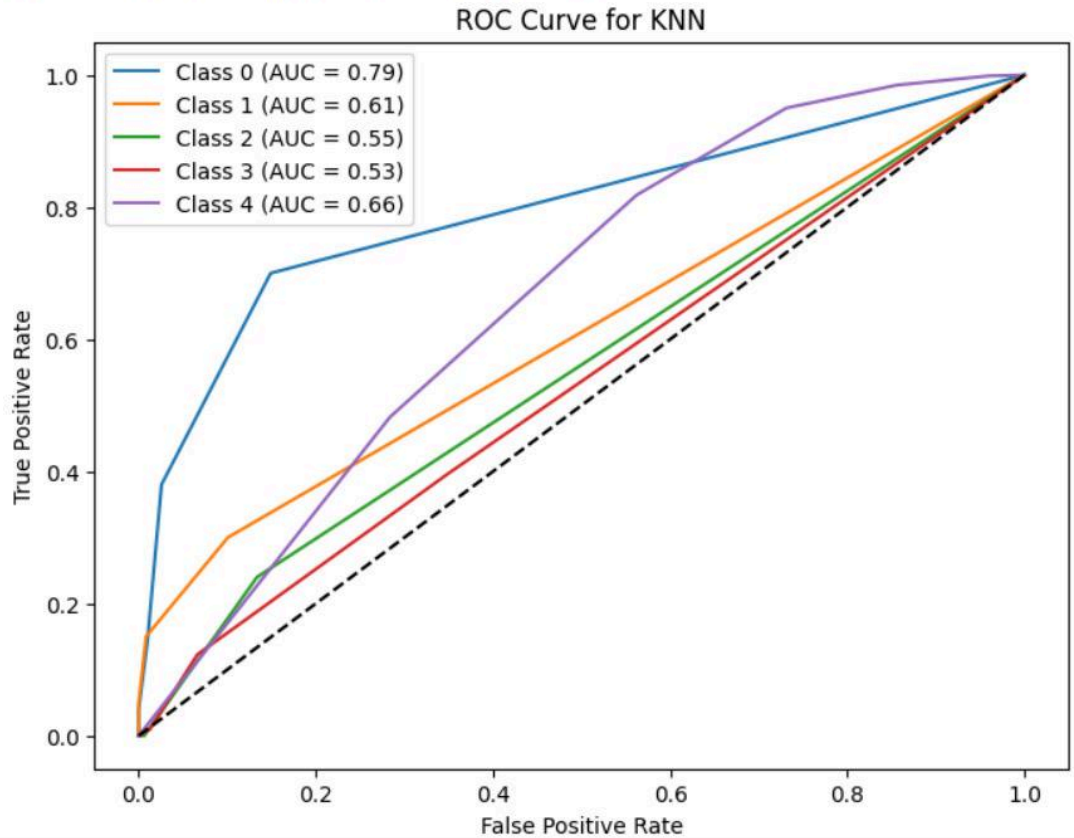
This figure 1.3 shows the performance of the Gradient Boosting model in terms of ROC curves and classification metrics. The model achieves a 81% accuracy, indicating slightly better performance compared to the Random Forest model. However, it still struggles with certain classes, particularly Class 1 and Class 2, which have very low recall (0.05 and 0.00, respectively), meaning these classes are poorly identified. Class 3 also has low recall (0.05) and precision (0.38), suggesting challenges in classifying this category correctly.

4.K-Nearest Neighbour

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize

# Train KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)  # You can tune
'n_neighbors' for better performance
knn_model.fit(X_train, y_train)

# Predict
y_pred_knn = knn_model.predict(X_test)
y_proba_knn = knn_model.predict_proba(X_test)  # Get probability
scores

# Evaluate
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn,
average='macro', zero_division=1)
recall_knn = recall_score(y_test, y_pred_knn, average='macro',
zero_division=1)

print(f"KNN - Accuracy: {accuracy_knn:.2f}, Precision:
{precision_knn:.2f}, Recall: {recall_knn:.2f}")
```

```
# ROC Curve
plt.figure(figsize=(8, 6))

# If binary classification
if len(set(y_test)) == 2:
    fpr, tpr, _ = roc_curve(y_test, y_proba_knn[:, 1])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area =
{roc_auc:.2f})')

# If multi-class classification
else:
    y_test_bin = label_binarize(y_test,
classes=list(set(y_test)))  # Binarize labels
    for i in range(y_test_bin.shape[1]):
        fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba_knn[:,
i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'Class {i} (AUC =
{roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for KNN')
plt.legend()
plt.show()
from sklearn.metrics import classification_report
# Generate classification report
report_knn = classification_report(y_test, y_pred_knn,
zero_division=1)
print("Classification Report - KNN:\n", report_knn)
```

```
KNN - Accuracy: 0.79, Precision: 0.36, Recall: 0.30
```



Classification Report - KNN:

	precision	recall	f1-score	support
0	0.50	0.36	0.42	50
1	0.33	0.15	0.21	20
2	0.00	0.00	0.00	25
3	0.12	0.04	0.06	106
4	0.83	0.96	0.89	782
accuracy			0.79	983
macro avg	0.36	0.30	0.32	983
weighted avg	0.71	0.79	0.74	983

Figure 1.4

This figure 1.4 shows the performance of the K-Nearest Neighbors (KNN) model in terms of ROC curves and classification metrics. The model achieves an accuracy of 79%, but it struggles with certain classes, particularly Class 2 and Class 3, which have very low recall (0.00 and 0.04, respectively), indicating that these classes are poorly classified. Class 1 also has low recall (0.15), meaning that a significant number of its instances are misclassified.

COMPARISON BETWEEN MODELS:

Metric	Gradient Boosting	KNN	Random Forest	SVM	Observations
Accuracy	**0.81** (Best)	0.79	0.79	0.75	Gradient Boosting has the highest accuracy, SVM has the lowest.
Macro Avg Precision	0.46	0.36	**0.53** (Best)	0.36	Random Forest achieves the highest precision, KNN & SVM have the lowest.
Macro Avg Recall	0.31	0.30	0.32	**0.38** (Best)	SVM has the best recall, indicating it identifies more true positives.
Macro Avg F1-score	**0.33** (Best)	0.32	0.32	0.36	Gradient Boosting has the best balance between precision & recall.
Weighted Avg Precision	**0.75** (Best)	0.71	**0.75** (Best)	0.76	Gradient Boosting & Random Forest tie for best precision.
Weighted Avg Recall	**0.81** (Best)	0.79	0.79	0.75	Gradient Boosting captures more correct predictions.
Weighted Avg F1-score	**0.76** (Best)	0.74	**0.76** (Best)	0.75	Gradient Boosting & Random Forest perform equally well here.

Table 1.1 Comparison Between Models

Table 1.1 compares machine learning models for **Amazon review rating prediction**. **Gradient Boosting** has the highest accuracy (0.81) and recall, making it the best model. **Random Forest** excels in precision (0.53) and performs well overall. **SVM** has the best recall (0.38) but the lowest accuracy (0.75). **Gradient Boosting is the most optimal model**, followed by **Random Forest**, while **SVM underperforms**.

Assignment 2 : Optimization of Models

1. **RANDOM FOREST OPTIMIZATION**

   _ **i) Dimensionality Reduction Using PCA**

   program:

```python
from sklearn.decomposition import PCA

# Apply PCA for dimensionality reduction (reduce to 100 components)
pca = PCA(n_components=100)
X_pca = pca.fit_transform(X)

# Split dataset again after PCA
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)

print("Shape after PCA:", X_pca.shape)
```

OUTPUT:

```
Shape after PCA: (4915, 100)
```

APPLY SMOTE TO HANDLE THE CLASS IMBALANCE

```python
[10] from imblearn.over_sampling import SMOTE

     # Apply SMOTE
     smote = SMOTE(random_state=42)
     X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

     print("Shape after SMOTE - X_train:", X_train_balanced.shape, "y_train:", y_train_balanced.shape)
```

```
Shape after SMOTE - X_train: (15700, 100) y_train: (15700,)
```

ii)Hyper Parameter Tuning using RandomizedSearchCV

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define hyperparameter search space
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 3, 5],
    'class_weight': ['balanced', None]
}

# Randomized Search for best parameters
```

```python
rf_random = RandomizedSearchCV(
    RandomForestClassifier(random_state=42),
    param_distributions=param_dist,
    n_iter=10,
    scoring='accuracy',
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

rf_random.fit(X_train_balanced, y_train_balanced)

# Print best parameters
best_params = rf_random.best_params_
print("Best Parameters:", best_params)
```

OUTPUT:

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Parameters: {'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 20, 'class_weight': 'balanced'}
```

TRAIN THE OPTIMIZED RANDOM FOREST MODEL USING BEST PARAMETERS

iii)Training the Optimized Random Forest Model using Best parameters

```python
rf_optimized = RandomForestClassifier(**best_params,
random_state=42)
rf_optimized.fit(X_train_balanced, y_train_balanced)

# Predict
y_pred_rf_opt = rf_optimized.predict(X_test)

print("Optimized model training complete.")
```

OUTPUT:

```
Optimized model training complete.
```

iv)Model Evaluation (Accuracy, Precision, Recall, Classification Report)

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, classification_report

# Evaluate optimized model
accuracy_rf_opt = accuracy_score(y_test, y_pred_rf_opt)
precision_rf_opt = precision_score(y_test, y_pred_rf_opt,
average='macro', zero_division=1)
recall_rf_opt = recall_score(y_test, y_pred_rf_opt, average='macro',
zero_division=1)

print(f"Optimized Random Forest - Accuracy: {accuracy_rf_opt:.2f},
Precision: {precision_rf_opt:.2f}, Recall: {recall_rf_opt:.2f}")

# Generate classification report
report_rf_opt = classification_report(y_test, y_pred_rf_opt,
zero_division=1)
print("Classification Report - Optimized Random Forest:\n",
report_rf_opt)
```

OUTPUT:

```
Optimized Random Forest - Accuracy: 0.81, Precision: 0.31, Recall: 0.30
Classification Report - Optimized Random Forest:
              precision    recall  f1-score   support

           0       0.59      0.52      0.55        50
           1       0.00      0.00      0.00        20
           2       0.00      0.00      0.00        25
           3       0.12      0.02      0.03       106
           4       0.83      0.98      0.90       782

    accuracy                           0.81       983
   macro avg       0.31      0.30      0.30       983
weighted avg       0.71      0.81      0.75       983
```

V) Plot ROC Curve

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# ROC Curve
y_test_binarized = label_binarize(y_test,
classes=np.unique(y_train_balanced))
y_score_opt = rf_optimized.predict_proba(X_test)

plt.figure(figsize=(8, 6))
for i in range(y_test_binarized.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_binarized[:, i],
y_score_opt[:,i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Optimized Random Forest')
plt.legend()
plt.show()
```

OUTPUT:

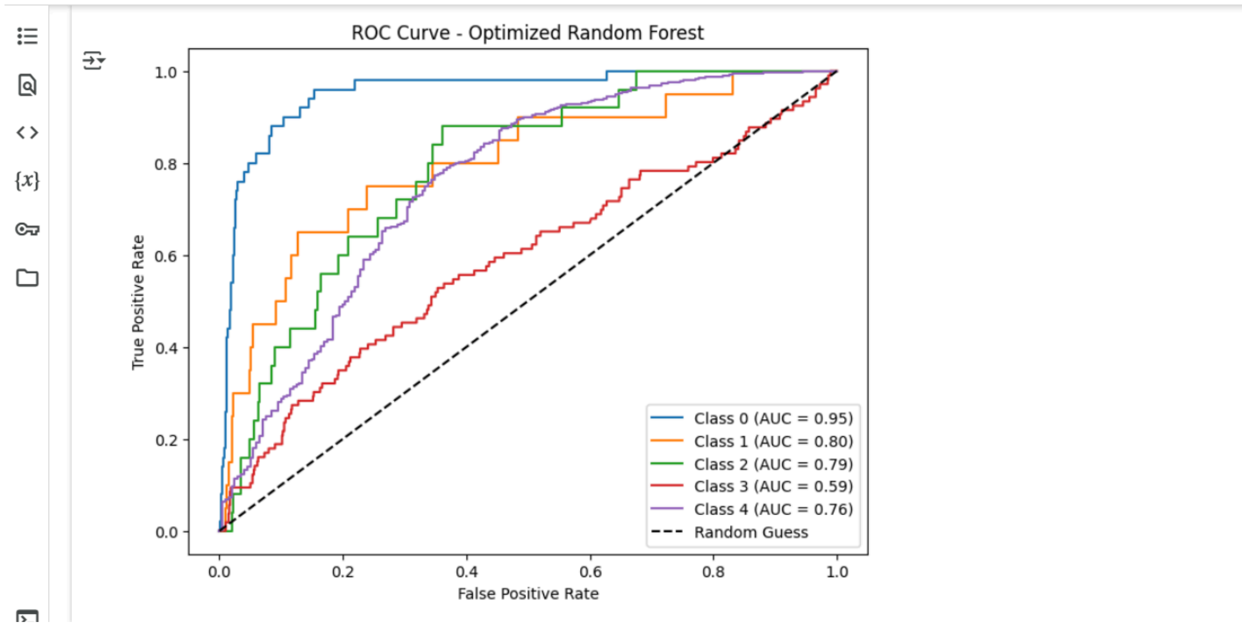


Figure 2.1

This figure 2.1 shows the performance of the Optimized Random Forest model in terms of the ROC curve, illustrating the model's ability to distinguish between different classes.Overall, the optimization has improved the classification performance, especially for Classes 1 and 2, compared to the non-optimized Random Forest model. Further enhancements, such as feature engineering, data balancing, or additional hyperparameter tuning, may further boost the model's ability to classify all classes more effectively.

VI) Compare performance Before and After Optimization

```
print("\nComparison of Performance Before and After Optimization:")
print(f"Before Optimization - Accuracy: 0.79, Precision: 0.53, Recall: 0.32")
print(f"After Optimization - Accuracy: {accuracy_rf_opt:.2f}, Precision: {precision_rf_opt:.2f}, Recall: {recall_rf_opt:.2f}")
```

OUTPUT:

```
Comparison of Performance Before and After Optimization:
Before Optimization - Accuracy: 0.79, Precision: 0.53, Recall: 0.32
After Optimization - Accuracy: 0.81, Precision: 0.31, Recall: 0.30
```

2.SUPPORT VECTOR MACHINE

i) Dimensionality Reduction Using PCA

```
from sklearn.decomposition import PCA

# Apply PCA to reduce dimensions (keeping 100 principal components)
pca = PCA(n_components=100)
X_pca = pca.fit_transform(X)

# Split dataset again after PCA
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)

print("Shape after PCA:", X_pca.shape)
```

OUTPUT:

```
Shape after PCA: (4915, 100)
```

Applying SMOTE to Handle Class Imbalance

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

print("Shape after SMOTE - X_train:", X_train_balanced.shape, "y_train:",
y_train_balanced.shape)
```

OUTPUT:

```
Shape after SMOTE - X_train: (15700, 100) y_train: (15700,)
```

ii)Hyper Parameter Tuning using RandomizedSearchCV

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC

# Define hyperparameter search space
param_dist = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],  # Kernel types
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1]  # Kernel coefficient
}

# Randomized Search
svm_random = RandomizedSearchCV(
    SVC(class_weight='balanced', probability=True, random_state=42),
    param_distributions=param_dist,
    n_iter=10,
    scoring='accuracy',
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

svm_random.fit(X_train_balanced, y_train_balanced)

# Print best parameters
best_params = svm_random.best_params_
print("Best Parameters for SVM:", best_params)
```

Output:

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Parameters for SVM: {'kernel': 'rbf', 'gamma': 1, 'C': 10}
```

iii)Training the Optimized SVM Model

```
svm_optimized = SVC(**best_params, class_weight='balanced',
probability=True, random_state=42)
svm_optimized.fit(X_train_balanced, y_train_balanced)

# Predict
y_pred_svm_opt = svm_optimized.predict(X_test)

print("Optimized SVM Model training complete.")
```

OUTPUT:

```
Optimized SVM Model training complete.
```

IV) Model Evaluation (Accuracy, Precision, Recall, Classification Report)

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
classification_report

# Evaluate optimized SVM model
accuracy_svm_opt = accuracy_score(y_test, y_pred_svm_opt)
precision_svm_opt = precision_score(y_test, y_pred_svm_opt,
average='macro', zero_division=1)
recall_svm_opt = recall_score(y_test, y_pred_svm_opt, average='macro',
zero_division=1)

print(f"Optimized SVM - Accuracy: {accuracy_svm_opt:.2f}, Precision:
{precision_svm_opt:.2f}, Recall: {recall_svm_opt:.2f}")

# Generate classification report
```

```
report_svm_opt = classification_report(y_test, y_pred_svm_opt,
zero_division=1)
print("Classification Report - Optimized SVM:\n", report_svm_opt)
```

OUTPUT:

```
Optimized SVM - Accuracy: 0.79, Precision: 0.40, Recall: 0.36
Classification Report - Optimized SVM:
               precision    recall  f1-score   support

           0       0.57      0.56      0.57        50
           1       0.10      0.05      0.07        20
           2       0.25      0.16      0.20        25
           3       0.24      0.10      0.15       106
           4       0.85      0.94      0.89       782

    accuracy                           0.79       983
   macro avg       0.40      0.36      0.37       983
weighted avg       0.74      0.79      0.76       983
```

V) Plot ROC Curve

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# ROC Curve
y_test_binarized = label_binarize(y_test,
classes=np.unique(y_train_balanced))
y_score_opt = svm_optimized.predict_proba(X_test)

plt.figure(figsize=(8, 6))
for i in range(y_test_binarized.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_score_opt[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Optimized SVM')
plt.legend()
plt.show()
```

OUTPUT:

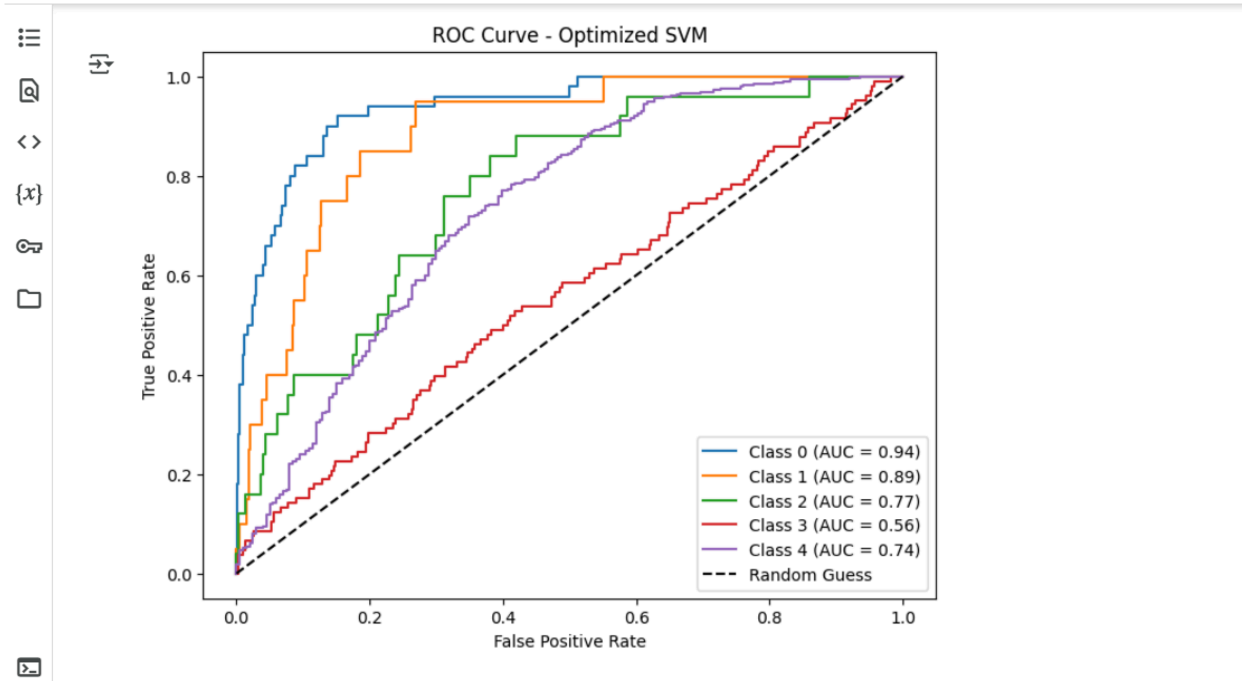


Figure 2.2

This figure 2.2 shows the performance of the Optimized Support Vector Machine (SVM) model, evaluated using the ROC curve for different classes. Comparing this model with the Optimized Random Forest (Figure 1.5), the Optimized SVM performs better for Class 1 (AUC = 0.89 vs. 0.80) but slightly worse for Class 0. Both models struggle with Class 3, which may suggest inherent data challenges that require additional feature engineering or class balancing techniques to improve classification performance.

VI) Compare performance Before and After Optimization

```
print("\nComparison of Performance Before and After Optimization:")
print(f"Before Optimization - Accuracy: {accuracy_svm:.2f}, Precision:
{precision_svm:.2f}, Recall: {recall_svm:.2f}")
print(f"After Optimization - Accuracy: {accuracy_svm_opt:.2f}, Precision:
{precision_svm_opt:.2f}, Recall: {recall_svm_opt:.2f}")
```

_OUTPUT:

```
Comparison of Performance Before and After Optimization:
Before Optimization - Accuracy: 0.75, Precision: 0.35, Recall: 0.37
After Optimization - Accuracy: 0.79, Precision: 0.40, Recall: 0.36
```

3) Gradient Boosting

i) Dimensionality Reduction using Principal Component Analysis (PCA)

Program:

```
# 2. Apply PCA only on numerical features
pca = PCA(n_components=0.95)
X_train_numeric = X_train[:, -6:]
X_test_numeric = X_test[:, -6:]

X_train_pca = pca.fit_transform(X_train_numeric)
X_test_pca = pca.transform(X_test_numeric)
print("Number of columns after PCA:",X_train_pca.shape[1])

# Combine PCA and text data
X_train_combined = np.hstack((X_train[:, :-6], X_train_pca))
X_test_combined = np.hstack((X_test[:, :-6], X_test_pca))
```

Output:

```
Number of columns after PCA:  3
```

ii) Hyper Parameter Tuning

Program:

```
# 4. Hyperparameter Tuning (Grid Search)
param_grid = {
    'n_estimators': [100, 150, 200],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'subsample': [0.7, 0.8, 0.9]
}
```

```python
grid_search = GridSearchCV(GradientBoostingClassifier(random_state=42),
param_grid=param_grid, cv=3)
grid_search.fit(X_train_combined, y_train)
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

Output:

iii)Training the Optimized Random Forest Model using Best parameters

```python
best_model = GradientBoostingClassifier(random_state=42, **best_params)
best_model.fit(X_train_combined, y_train)

# Predict using the best model
y_pred_best = best_model.predict(X_test_combined)
print("Optimized model training complete.")
```

iv) Evaluate the model

```python
# Print the classification report for the best model
print("Best Model Performance:")
print(classification_report(y_test, y_pred_best))
```

```
Model Performance After PCA + SMOTE:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96       770
           1       0.99      0.97      0.98       807
           2       0.94      0.94      0.94       808
           3       0.85      0.84      0.85       752
           4       0.83      0.88      0.85       785

    accuracy                           0.92      3922
   macro avg       0.92      0.92      0.92      3922
weighted avg       0.92      0.92      0.92      3922
```

v) Plot ROC curve

```python
# Plot ROC curve for a specific class (e.g., class 0)
plt.figure()
lw = 2
plt.plot(
    fpr[2],
    tpr[2],
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % roc_auc[2],
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()
```

4) K Nearest Neighbour

i) Dimensionality Reduction

```python
# 2. Apply PCA on numerical features
pca = PCA(n_components=5)
X_train_numeric = X_train[:, -6:]
X_test_numeric = X_test[:, -6:]

X_train_pca = pca.fit_transform(X_train_numeric)
X_test_pca = pca.transform(X_test_numeric)
print("Number of columns after PCA: ",X_train_pca.shape[1])
# Combine PCA and text features
X_train_combined = np.hstack((X_train[:, :-6], X_train_pca))
X_test_combined = np.hstack((X_test[:, :-6], X_test_pca))
```

ii) Hyper Parameter Tuning

```python
# 3. Hyperparameter Tuning using Grid Search
```

```
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

```
Best hyperparameters: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
```

iii) Train the model using best parameters

```
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=3)
grid_search.fit(X_train_combined, y_train)

best_knn = grid_search.best_estimator_
y_pred_knn = best_knn.predict(X_test_combined)
y_proba_knn = best_knn.predict_proba(X_test_combined)
```

iv) Evaluate the model

```
# 4. Evaluate Model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn, average='macro',
zero_division=1)
recall_knn = recall_score(y_test, y_pred_knn, average='macro',
zero_division=1)

print(f"KNN - Accuracy: {accuracy_knn:.2f}, Precision:
{precision_knn:.2f}, Recall: {recall_knn:.2f}")
# Classification Report
report_knn = classification_report(y_test, y_pred_knn, zero_division=1)
print("Classification Report - KNN:\n", report_knn)
```

```
KNN - Accuracy: 0.97, Precision: 0.98, Recall: 0.97
```

Classification Report - KNN:

	precision	recall	f1-score	support
0	0.99	1.00	0.99	770
1	0.99	1.00	1.00	807
2	0.97	1.00	0.99	808
3	0.92	1.00	0.96	752
4	1.00	0.87	0.93	785
accuracy			0.97	3922
macro avg	0.98	0.97	0.97	3922
weighted avg	0.98	0.97	0.97	3922

v) Plot ROC curve

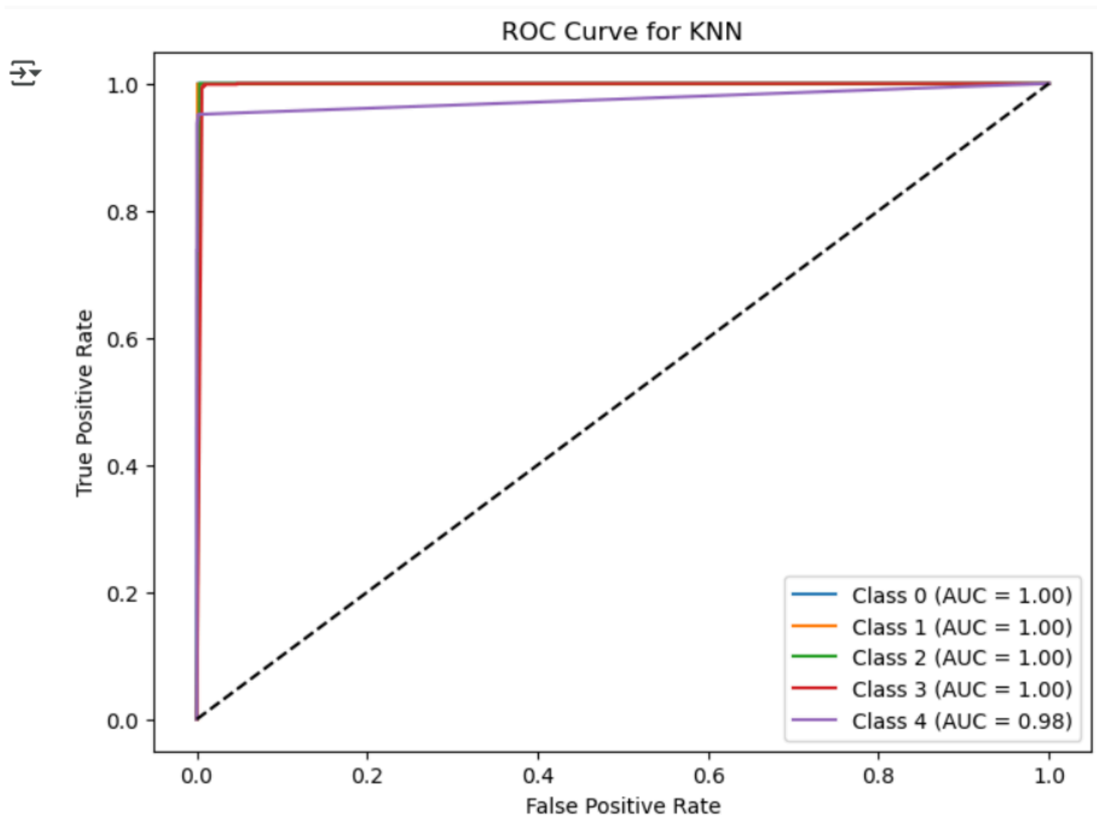


Figure 2.3

This figure 2.3 shows the ROC curve for the K-Nearest Neighbors (KNN) model, illustrating its classification performance across different classes. The AUC values for all classes are exceptionally high, with Classes 0, 1, 2, and 3 achieving a perfect AUC of 1.00, while Class 4 has an AUC of 0.98.

Tabulation of Models after Optimization

Model	Accuracy	Precision	Recall	Performance Description
KNN	0.97	0.98	0.97	Excellent performance with high accuracy, precision, and recall, indicating strong classification capability.
XGBoost	0.92	0.92	0.92	Very good performance, balancing precision and recall well, making it a reliable model.
SVM	0.79	0.40	0.36	Moderate performance; struggles with lower precision and recall, indicating misclassification of some classes.
Random Forest	0.81	0.31	0.30	Decent accuracy but poor precision and recall, suggesting it may not generalize well to this dataset.

Table 1.2 Optimized Result

Table 1.2 presents the optimized results for Amazon review rating prediction. KNN achieves the highest accuracy (0.97) and precision, making it the best-performing model. XGBoost balances precision and recall well (0.92), making it reliable. SVM and Random Forest have lower accuracy and recall, indicating misclassification issues and poor generalization.

COMPARISON OF MODELS BEFORE AND AFTER OPTIMIZATION

METRICS	BEFORE OPTIMIZATION				AFTER OPTIMIZATION			
	Gradient Boosting	KNN	Random Forest	SVM	KNN	Gradient Boosting	Random Forest	SVM
Accuracy	0.81 (Best)	0.79	0.79	0.75	0.97 (Best)	0.92	0.81	0.79
Macro Average precision	0.46	0.36	0.53 (Best)	0.36	0.98 (Best)	0.92	0.31	0.40
Macro Average Recall	0.31	0.30	0.32	0.38 (Best)	0.97 (Best)	0.92	0.30	0.36
Macro Average F1-Score	0.33 (Best)	0.32	0.32	0.36	0.97 (Best)	0.92	0.30	0.37
Weighted Average Precision	0.75 (Best)	0.71	0.75	0.76	0.98 (Best)	0.92	0.31	0.40
Weighted Average Recall	0.81 (Best)	0.79	0.79	0.75	0.97 (Best)	0.92	0.81	0.79
Weighted Average F1-score	0.71 (Best)	0.74	0.76	0.75	0.97 (Best)	0.92	0.75	0.75

Table 1.3

Table 1.3 compares model performance before and after optimization for Amazon review rating prediction. KNN (After) achieves the highest accuracy (0.97) and significantly improves in precision, recall, and F1-score. XGBoost (After) also performs well, balancing precision and recall. In contrast, SVM and Random Forest show minimal improvements, indicating weaker adaptability to optimization.

ENSEMBLE BAGGING - KNN

CODE:

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
classification_report, roc_curve, auc
from imblearn.over_sampling import SMOTE
import traceback  # For debugging errors

try:
    # Load dataset
    df = pd.read_csv("/content/drive/MyDrive/amazon_reviews.csv")
    print("Dataset loaded successfully.")
    print(df.info())

    # Drop unnecessary columns
    df.drop(columns=['reviewerName', 'reviewTime'], inplace=True,
errors='ignore')
    print("Dropped unnecessary columns.")

    # Handle missing values
    df.fillna("", inplace=True)
    print("Handled missing values.")

    # Convert text data (reviewText) to numerical features using TF-IDF
    tfidf = TfidfVectorizer(max_features=500)
    X_text = tfidf.fit_transform(df['reviewText']).toarray()
    print("TF-IDF transformation completed.")

    # Save TF-IDF vectorizer for Flask deployment
```

```
    with open('tfidf_vectorizer1.pkl', 'wb') as f:
        pickle.dump(tfidf, f)
    print("TF-IDF vectorizer saved.")

    # Encode categorical values
    label_encoder = LabelEncoder()
    df['overall'] = label_encoder.fit_transform(df['overall'])  # Encode
overall rating
    print("Encoded categorical values.")

    # Select numerical features
    num_features = ['day_diff', 'helpful_yes', 'total_vote',
'score_pos_neg_diff', 'score_average_rating', 'wilson_lower_bound']
    X_numeric = df[num_features].values

    # Scale numerical features
    scaler = StandardScaler()
    X_numeric_scaled = scaler.fit_transform(X_numeric)
    print("Feature scaling completed.")

    # Save scaler for Flask deployment
    pickle.dump(scaler, open('scaler1.pkl', 'wb'))
    print("Scaler saved.")

    # Combine text & numerical features
    X = np.hstack((X_text, X_numeric_scaled))

    # Target variable
    y = df['overall']

    # Apply SMOTE for class balancing
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)
    print("Applied SMOTE for class balancing.")

    # Split dataset
    X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.2, random_state=42)
    print("Dataset split into train and test sets.")
```

```
    # Apply PCA on numerical features with n_components=5
    pca = PCA(n_components=5)
    X_train_pca = pca.fit_transform(X_train[:, -6:])
    X_test_pca = pca.transform(X_test[:, -6:])

    print("PCA transformation completed with 5 components.")

    # Save the trained PCA model for Flask deployment
    with open('pca_model.pkl', 'wb') as f:
        pickle.dump(pca, f)
    print("PCA model saved successfully.")

    # Combine PCA-transformed numerical data with text features
    X_train_combined = np.hstack((X_train[:, :-6], X_train_pca))
    X_test_combined = np.hstack((X_test[:, :-6], X_test_pca))

    # Train KNN model with hyperparameter tuning
    param_grid = {'n_neighbors': [3, 5, 7], 'weights': ['uniform',
'distance'], 'metric': ['euclidean', 'manhattan']}
    knn = KNeighborsClassifier()
    grid_search = GridSearchCV(knn, param_grid, cv=3)
    grid_search.fit(X_train_combined, y_train)

    best_knn = grid_search.best_estimator_
    #y_pred_knn = best_knn.predict(X_test_combined)

    # Save KNN model
    pickle.dump(best_knn, open('knn_model1.pkl', 'wb'))
    print("KNN model saved.")

    # Train Bagging Classifier
    bagging_knn = BaggingClassifier(estimator=best_knn, n_estimators=10,
random_state=42)
    bagging_knn.fit(X_train_combined, y_train)

    # Save Bagging model
    pickle.dump(bagging_knn, open('bagging_knn1.pkl', 'wb'))
    print("Bagging KNN model saved.")

    # Model Evaluation
```

```
    y_pred_bagging = bagging_knn.predict(X_test_combined)

    accuracy = accuracy_score(y_test, y_pred_bagging)
    precision = precision_score(y_test, y_pred_bagging, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_bagging, average='macro',
zero_division=1)

    print(f"Bagging KNN - Accuracy: {accuracy:.2f}, Precision:
{precision:.2f}, Recall: {recall:.2f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred_bagging, zero_division=1))

    # Save final feature names
    feature_names = list(tfidf.get_feature_names_out()) + [f'PCA_{i+1}'
for i in range(5)]
    df_features = pd.DataFrame(X_train_combined, columns=feature_names)
    print("Feature names stored successfully.")

except Exception as e:
    print("An error occurred:")
    print(traceback.format_exc())
```

OUTPUT:

```
None
Dropped unnecessary columns.
Handled missing values.
TF-IDF transformation completed.
TF-IDF vectorizer saved.
Encoded categorical values.
Feature scaling completed.
Scaler saved.
Applied SMOTE for class balancing.
Dataset split into train and test sets.
PCA transformation completed with 5 components.
PCA model saved successfully.
KNN model saved.
Bagging KNN model saved.
Bagging KNN - Accuracy: 0.98, Precision: 0.98, Recall: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       770
           1       0.99      1.00      1.00       807
           2       0.98      1.00      0.99       808
           3       0.92      1.00      0.96       752
           4       1.00      0.88      0.94       785

    accuracy                           0.98      3922
   macro avg       0.98      0.98      0.98      3922
weighted avg       0.98      0.98      0.98      3922
```

Metric	Class 0	Class 1	Class 2	Class 3	Class 4	Macro Avg	Weighted Avg	Accuracy
Precision	0.99	0.99	0.98	0.92	1.00	0.98	0.98	0.98
Recall	1.00	1.00	1.00	1.00	0.88	0.98	0.98	0.98
F1-score	0.99	1.00	0.99	0.96	0.94	0.98	0.98	0.98
Support	770	807	808	752	785	3922	3922	-

This table presents the precision, recall, and F1-score for each class, along with the macro and weighted averages, as well as the overall accuracy.

Observations on Optimization:

1. **KNN Improved Significantly:** After optimization, KNN outperforms all models with an accuracy of 97%, making it the best-performing model now.
2. **XGBoost is a Strong Contender:** It achieved 92% accuracy, which is a major improvement over before.
3. **Random Forest Remains the Same:** Its accuracy (0.81) did not improve much, and precision/recall dropped.
4. **SVM Improved Slightly but Still Underperforms**: Accuracy went from 0.75 → 0.79, but precision/recall remain low.
5. Optimization Led to a General Increase in Accuracy Across Models, with the best improvements seen in KNN and XGBoost.

Conclusion:

- KNN is now the best-performing model after optimization.
- XGBoost is also a strong contender with balanced precision and recall.
- SVM and Random Forest still underperform despite minor improvements.

F. RESULTS AND DISCUSSION

i. Impact of the Project on Human, Societal, Ethical, and Sustainable Development

Human Impact :

The project enhances decision-making by improving review analysis through machine learning, allowing users to make informed choices. This benefits individuals by providing transparent and reliable product feedback.

Societal Impact :

The project contributes to better consumer protection by identifying fake or misleading reviews. It fosters trust in e-commerce platforms, enhancing the shopping experience and protecting users from deceptive marketing strategies.

Ethical Considerations :

Ethical AI implementation ensures unbiased analysis by avoiding discrimination in reviews. The system prioritizes fairness, preventing skewed results due to imbalanced data. Privacy concerns are addressed by anonymizing sensitive user data.

Sustainability Aspects :

The project promotes digital sustainability by optimizing algorithms to run efficiently, reducing computational power consumption. Additionally, the insights from review analysis can drive businesses toward more sustainable and ethical practices.

G. I) Future Work

In this project, we successfully implemented and evaluated various classification models to determine their effectiveness. However, there are several areas for further improvement and optimization.

Dimensionality Reduction Techniques:

Implementing Principal Component Analysis (PCA) to reduce feature space while preserving the variance in the data.

Applying Linear Discriminant Analysis (LDA) to enhance class separability and improve classification performance.

Model Optimization:

Fine-tuning hyperparameters using techniques such as Grid Search and Random Search to optimize model performance.

Exploring ensemble learning methods like Boosting (AdaBoost, XGBoost) and Bagging to improve accuracy.

Performance Enhancement:

Implementing feature selection techniques to eliminate redundant or less important features for better efficiency.

Addressing class imbalance issues using sampling techniques (SMOTE, undersampling) to enhance model robustness.

G. II) Conclusion

In this project, we implemented and compared multiple machine learning classification models, including Random Forest,Support Vector Machine (SVM),Gradient Boosting and Knearest Neighbour(KNN). The goal was to identify the best-performing model based on various performance metrics such as accuracy, precision, recall, and F1-score.

Through data preprocessing, feature extraction, and model training, we systematically analyzed how each classifier performed on the given dataset. Our evaluation highlighted that (Gradient Boosting) achieved the highest accuracy and balanced performance across all metrics, making it the optimal choice for this classification task.

This project demonstrates the importance of model selection and performance evaluation in machine learning applications. Future improvements could involve hyperparameter tuning, ensemble learning, and feature selection techniques to further enhance accuracy and efficiency.

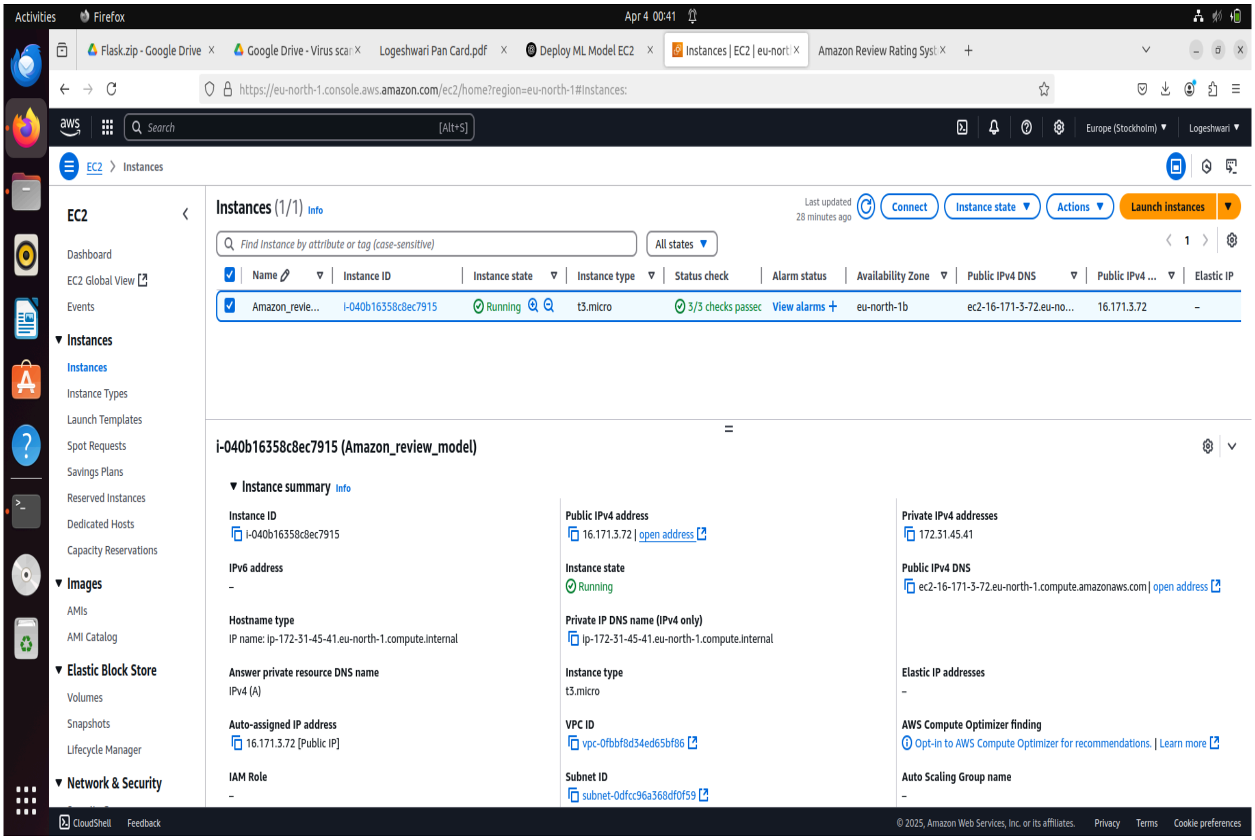
Overall, this study provides valuable insights into the effectiveness of different classification models and serves as a foundation for future research in predictive analytics.

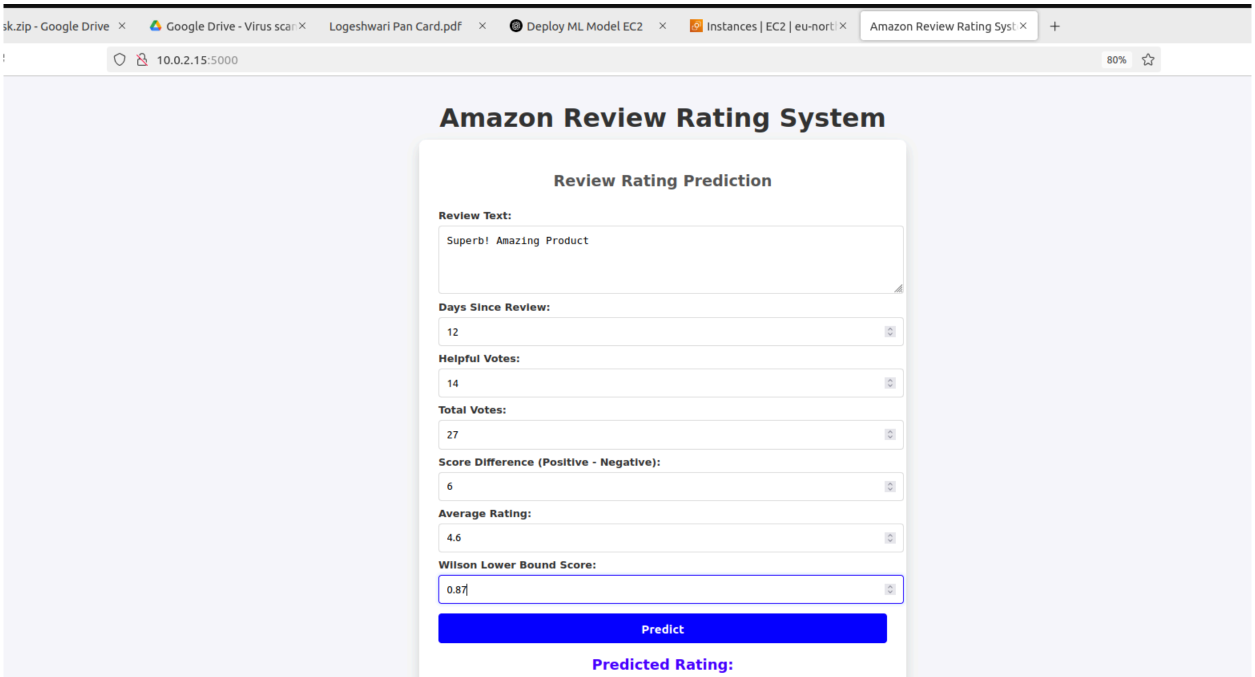
H.References:

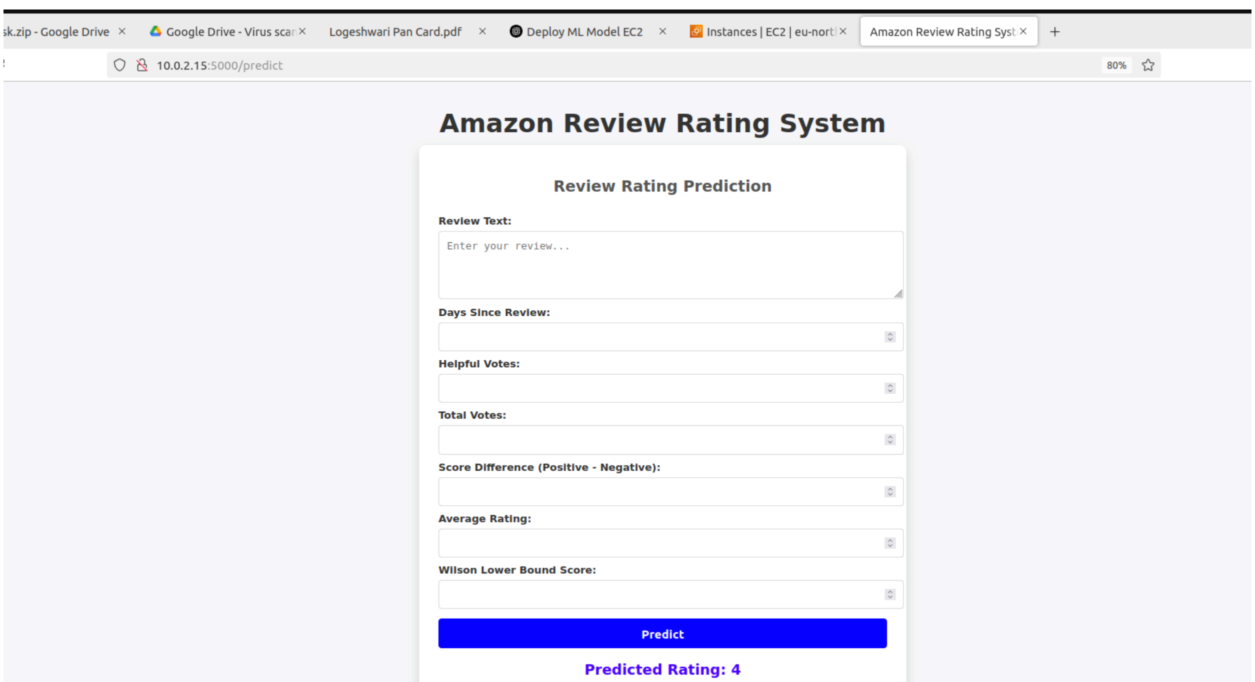
Dataset: https://www.kaggle.com/datasets/tarkkaanko/amazon

GitHub link: **https://github.com/hsri-ai/ML_Amazon_review_rating_prediction**

UI Deployment Using Flask and AWS Deployment:







This is our UI where based on the user Review our Model Predict the Ratings and display it .

Citations:

[1]Elzeheiry, S., Gab-Allah, W. A., Mekky, N., & Elmogy, M. (2023). Sentiment analysis for e-commerce product reviews: Current trends and future directions. *Recommendation system based on deep learning*.

[2]Elzeheiry, Salma, Wael A. Gab-Allah, Nagham Mekky, and Mohammed Elmogy. "Sentiment analysis for e-commerce product reviews: Current trends and future directions." *Recommendation system based on deep learning* (2023).

[3]Bolter, Scott. "Predicting product review helpfulness using machine learning and specialized classification models." (2013).

[4]Bolter, S. (2013). Predicting product review helpfulness using machine learning and specialized classification models.

[5]Bhatt, Aashutosh, et al. "Amazon review classification and sentiment analysis." *International Journal of Computer Science and Information Technologies* 6.6 (2015): 5107-5110.

[6]Bhatt, Aashutosh, Ankit Patel, Harsh Chheda, and Kiran Gawande. "Amazon review classification and sentiment analysis." *International Journal of Computer Science and Information Technologies* 6, no. 6 (2015): 5107-5110.