Mini Project Report
On

SOCIAL MEDIA SENTIMENT ANALYSIS
USING PYTHON

Submitted by

Name:- Asmita Routray
Roll No:- 2206330
Name:- Bhoomika Dash
Roll No:- 2206331

School of Computer Engineering
KIIT - DU

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Table of Contents

1 Introduction 3

2 Problem Statement 4

3 Python Package Used Details 5

4 SourceCode 12

5 Implementation Results 13

6 Conclusion 17

7 References 18

Chapter 1

Introduction

Social media has become a significant platform for public opinion, making sentiment analysis a crucial tool in understanding user emotions. Sentiment analysis involves using natural language processing (NLP) techniques to classify text as positive, negative, or neutral.

In this project, we implemented sentiment analysis on Twitter data using machine learning techniques. The dataset used consists of 1.6 million tweets labeled as positive or negative. The goal was to preprocess the data, extract meaningful features, and train a machine learning model to classify sentiments accurately. To achieve this, we performed several steps, including data cleaning, text preprocessing, feature extraction using TF-IDF, and training classification models such as Logistic Regression, SGD classifier, XGBoost. The model was evaluated based on accuracy and classification metrics.

The results demonstrate the feasibility of using machine learning for sentiment analysis on social media, providing valuable insights into public opinion trends. The developed model can be extended for various applications such as brand monitoring, customer feedback analysis, and social trend prediction.

Chapter 2

Problem Statement & Objectives

Problem Statement:

Determining whether a given comment expresses a positive or negative sentiment is a challenging task due to the complexities of natural language. Social media comments often include sarcasm, ambiguous phrases, and varied writing styles, making it difficult for traditional sentiment analysis models to classify them accurately. Our primary focus is to develop a machine learning model that can effectively differentiate between positive and negative comments, providing an automated solution for sentiment classification.

Objectives:

- To preprocess and clean the textual data effectively, ensuring better representation for sentiment classification.

- To develop and optimize machine learning models for accurately predicting whether a given comment is positive or negative.

- To evaluate model performance using accuracy, precision, recall, and F1-score, ensuring the reliability of the sentiment analysis system.

.

Chapter 3: Python Packages Used Details

Package Name	Functions Used	Explanation
pandas	read_csv() data[['target', 'text']]	Load and preprocess the dataset. Selects relevant columns from the datase
numpy	General numerical computations	Used implicitly for handling numerical computations during model training.
sklearn.model_selection	train_test_split()	Splits data into training and testing sets
sklearn.preprocessing	StandardScaler()	Scales the dataset for better performance
sklearn.feature_extraction.text	TfidfVectorizer()	Converts text into numerical features using TF-IDF
sklearn.metrics	classification_report() accuracy_score() confusion_matrix() roc_curve(), auc()	Evaluates model performance
matplotlib.pyplot	plot(), show(), title(), xlabel(), ylabel(), figure()	Creates and customizes plots for data visualization.
xgboost	XGBClassifier()	Implements the XGBoost model
sklearn.ensemble	RandomForestClassifier()	Implements a Random Forest model
sklearn.linear_model	LogisticRegression() SGSClassifier()	Implements Logistic Regression and SGDClassifier
sklearn.svm	SVC()	Implements Support Vector Machine (SVM)
seaborn	heatmap(), barplot()	Generates heatmaps and bar plots for better visualization of model performance.

Chapter 4
Source Code

Feature Extraction and Data Preprocessing

```
import pandas as pd
import numpy as np
import re
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Define column names (since dataset lacks headers)
column_names = ["sentiment", "id", "date", "query", "user", "text"]

# Load dataset (update path if needed)
df = pd.read_csv("training.1600000.processed.noemoticon (1).csv",
encoding="ISO-8859-1", names=column_names)

# Keep only sentiment and text columns
df = df[["sentiment", "text"]]

# Convert sentiment labels (0 → Negative, 2 → Neutral, 4 → Positive)
df["sentiment"] = df["sentiment"].map({0: "negative", 2: "neutral", 4:
"positive"})

# Check class distribution
print(df["sentiment"].value_counts())

# Text Preprocessing
def clean_text(text):
    text = text.lower()  # Convert to lowercase
```

```
    text = re.sub(r"http\S+|www\S+|https\S+", "", text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#','', text)
    text = re.sub(r"[^\w\s]", "", text)
    text = re.sub(r"\d+", "", text)
    text = text.strip()
    return text

# Apply cleaning function
df["clean_text"] = df["text"].apply(clean_text)

#  Feature Extraction (TF-IDF)
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df["clean_text"])

# Encode sentiment labels
y = df["sentiment"].map({"negative": 0, "neutral": 1, "positive": 2})

# Train-test split (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Ensure labels are correctly mapped
y_train = y_train.map({0: 0, 2: 1})
y_test = y_test.map({0: 0, 2: 1})

# Drop NaN values (caused by unmapped labels)
y_train = y_train.dropna().astype(int)
y_test = y_test.dropna().astype(int)

# Double-check unique values
print("Unique values in y_train:", y_train.unique())
print("Unique values in y_test:", y_test.unique())
```

Model Implementation and Performance

```
# Model Training & Evaluation

# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
print("\n    Logistic Regression Results:")
print(classification_report(y_test, y_pred_lr))
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Hyperparameter grid for Logistic Regression
param_grid_lr = {
   "C": [0.001, 0.01, 0.1, 1, 10],  # Regularization strength
   "solver": ["lbfgs", "liblinear"]  # Optimizers
}

# Grid Search
grid_lr = GridSearchCV(LogisticRegression(max_iter=500,
random_state=42), param_grid_lr, cv=3, scoring="accuracy", n_jobs=-1)
grid_lr.fit(X_train, y_train)

# Best parameters & model
print("Best Logistic Regression Params:", grid_lr.best_params_)
best_lr = grid_lr.best_estimator_

# Evaluate
y_pred_best_lr = best_lr.predict(X_test)
print("Optimized Logistic Regression Accuracy:", accuracy_score(y_test,
y_pred_best_lr))

from xgboost import XGBClassifier
from sklearn.metrics import classification_report

# Convert labels: 4 → 1
y_train = y_train.replace(4, 1)
y_test = y_test.replace(4, 1)

# Define XGBoost model
xgb = XGBClassifier(
      n_estimators=100,
      max_depth=6,
      learning_rate=0.1,
      subsample=0.8,
      colsample_bytree=0.8,
       n_jobs=-1,
      random_state=42
```

```
)

# Train the model
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)

# Print results
print("\n XGBoost Results:")
print(classification_report(y_test, y_pred_xgb))

from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score

# Use SGD (Stochastic Gradient Descent) for faster training
svm = SGDClassifier(loss="hinge", max_iter=1000, tol=1e-3,
random_state=42)
svm.fit(X_train, y_train)

# Predictions
y_pred_svm = svm.predict(X_test)

# Accuracy
print("Fast SVM Accuracy:", accuracy_score(y_test, y_pred_svm))



# Define label mapping
label_mapping = {"negative": 0, "positive": 1}

# Convert y_pred values to numeric labels if they are still in string
format
def convert_predictions(y_pred):
  if isinstance(y_pred[0], str):  # Check if labels are still in text format
    return [label_mapping[label] for label in y_pred]  # Convert to numbers
  return y_pred

# Convert predictions
y_pred_lr = convert_predictions(y_pred_lr)
y_pred_sgd = convert_predictions(y_pred_sgd)
y_pred_xgb = convert_predictions(y_pred_xgb)
```

```
# Performance Metrics
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix

models = {"Logistic Regression": y_pred_lr, "SGD Classifier": y_pred_sgd,
"XGBoost": y_pred_xgb}

for model_name, y_pred in models.items():
   print(f"\n {model_name} Results:")
   print("Accuracy:", accuracy_score(y_test, y_pred))
   print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
   print("Classification Report:\n", classification_report(y_test, y_pred))


   # Function to plot confusion matrix
   def plot_confusion_matrix(y_test, y_pred, model_name):
      cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix
      plt.figure(figsize=(5,4))
      sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=["Negative", "Positive"], yticklabels=["Negative", "Positive"])
      plt.xlabel("Predicted Label")
      plt.ylabel("True Label")
      plt.title(f"Confusion Matrix - {model_name}")
      plt.show()

   # Plot confusion matrices for all models
   plot_confusion_matrix(y_test, y_pred_lr, "Logistic Regression")
   plot_confusion_matrix(y_test, y_pred_sgd, "SGD Classifier")
   plot_confusion_matrix(y_test, y_pred_xgb, "XGBoost")


# Compute performance metrics
models = ["Logistic Regression", "SGDClassifier", "XGBoost"]
accuracies = [
   accuracy_score(y_test, y_pred_lr),
   accuracy_score(y_test, y_pred_sgd),
   accuracy_score(y_test, y_pred_xgb)
]

precisions = [
   precision_score(y_test, y_pred_lr),
   precision_score(y_test, y_pred_sgd),
```

```
    precision_score(y_test, y_pred_xgb)
]

recalls = [
    recall_score(y_test, y_pred_lr),
    recall_score(y_test, y_pred_sgd),
    recall_score(y_test, y_pred_xgb)
]

f1_scores = [
    f1_score(y_test, y_pred_lr),
    f1_score(y_test, y_pred_sgd),
    f1_score(y_test, y_pred_xgb)
]

# Define bar width and positions
bar_width = 0.2
x = np.arange(len(models))

# Define colors
colors = ["royalblue", "mediumseagreen", "darkorange", "crimson"]

# Create bar plot
plt.figure(figsize=(12, 6))
plt.bar(x - bar_width*1.5, accuracies, width=bar_width, label="Accuracy",
color=colors[0], edgecolor='black')
plt.bar(x - bar_width/2, precisions, width=bar_width, label="Precision",
color=colors[1], edgecolor='black')
plt.bar(x + bar_width/2, recalls, width=bar_width, label="Recall",
color=colors[2], edgecolor='black')
plt.bar(x + bar_width*1.5, f1_scores, width=bar_width, label="F1 Score",
color=colors[3], edgecolor='black')

# Labels and formatting
plt.xlabel("Models", fontsize=12, fontweight="bold", color="darkblue")
plt.ylabel("Performance Score", fontsize=12, fontweight="bold",
color="darkblue")
plt.title("Performance Comparison of Models", fontsize=14,
fontweight="bold", color="navy")
plt.xticks(x, models, fontsize=11, fontweight="bold", color="black")
plt.ylim(0, 1)  # Scores are between 0 and 1
plt.legend(loc="lower right", fontsize=11, frameon=True, shadow=True)
```

```
# Display values on top of bars
for i in range(len(models)):
    plt.text(x[i] - bar_width*1.5, accuracies[i] + 0.02, f"{accuracies[i]:.2f}",
ha='center', fontsize=10, fontweight='bold')
    plt.text(x[i] - bar_width/2, precisions[i] + 0.02, f"{precisions[i]:.2f}",
ha='center', fontsize=10, fontweight='bold')
    plt.text(x[i] + bar_width/2, recalls[i] + 0.02, f"{recalls[i]:.2f}", ha='center',
fontsize=10, fontweight='bold')
    plt.text(x[i] + bar_width*1.5, f1_scores[i] + 0.02, f"{f1_scores[i]:.2f}",
ha='center', fontsize=10, fontweight='bold')

# Grid for better readability
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.show()


#ROC curve
from sklearn.metrics import roc_curve, auc
models = {"Logistic Regression": y_pred_lr, "SGD Classifier": y_pred_sgd,
"XGBoost": y_pred_xgb}

plt.figure(figsize=(8,6))
for model, preds in models.items():
    fpr, tpr, _ = roc_curve(y_test, preds)
    auc_score = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{model} (AUC = {auc_score:.2f})")

plt.plot([0, 1], [0, 1], linestyle="--", color="gray")  # Random Guess Line
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.title("ROC Curve - Model Comparison", fontsize=14, fontweight="bold")
plt.legend(loc="lower right", fontsize=11)
plt.grid(True, linestyle="--", alpha=0.7)
plt.show()
```

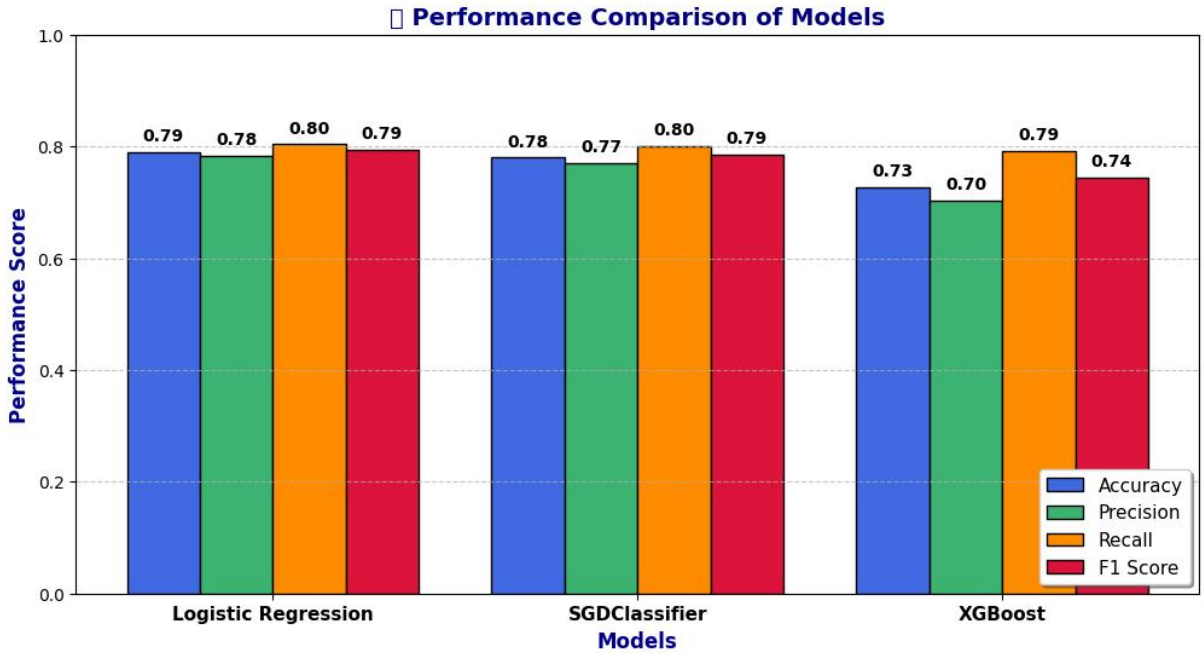
Chapter 5

Implementation Results

1. Model Performance Comparison

The table below summarizes the performance of Logistic Regression, SGD Classifier and XGBoost based on accuracy, precision, recall, and F1-score.

MODEL	Accuracy(%)	Precision(%)	Recall(%)	F1-score(%)
Logistic Regression	79.0	80.0(0-) 78.0 (1+)	78.0 (0-) 80.0 (1+)	79.0 (Both)
SGD Classifier	78.0	79.0(0-) 77.0 (1+)	76.0 (0-) 80.0 (1+)	78.0 (0-) 79.0 (1+)
XGBoost	72.8	76.0(0-) 70.0 (1+)	66.0 (0-) 79.0 (1+)	71.0 (0-) 74.0 (1+)



Key Observations:

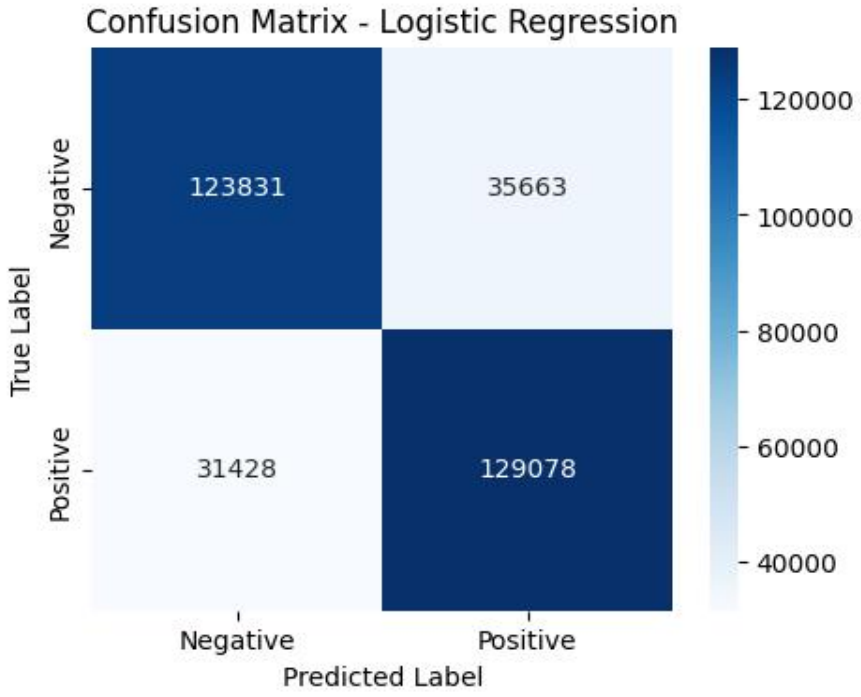
- Logistic Regression performed the best with 79% accuracy and a well-balanced precision-recall tradeoff.

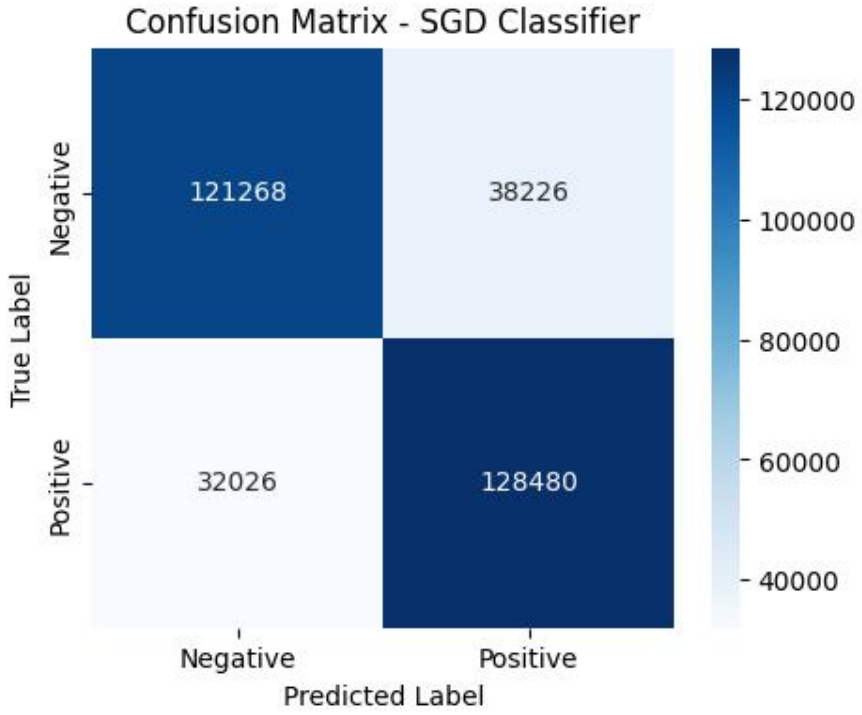
- SGD Classifier was slightly behind, with 78% accuracy, but had a better recall for positive sentiments than Logistic Regression.

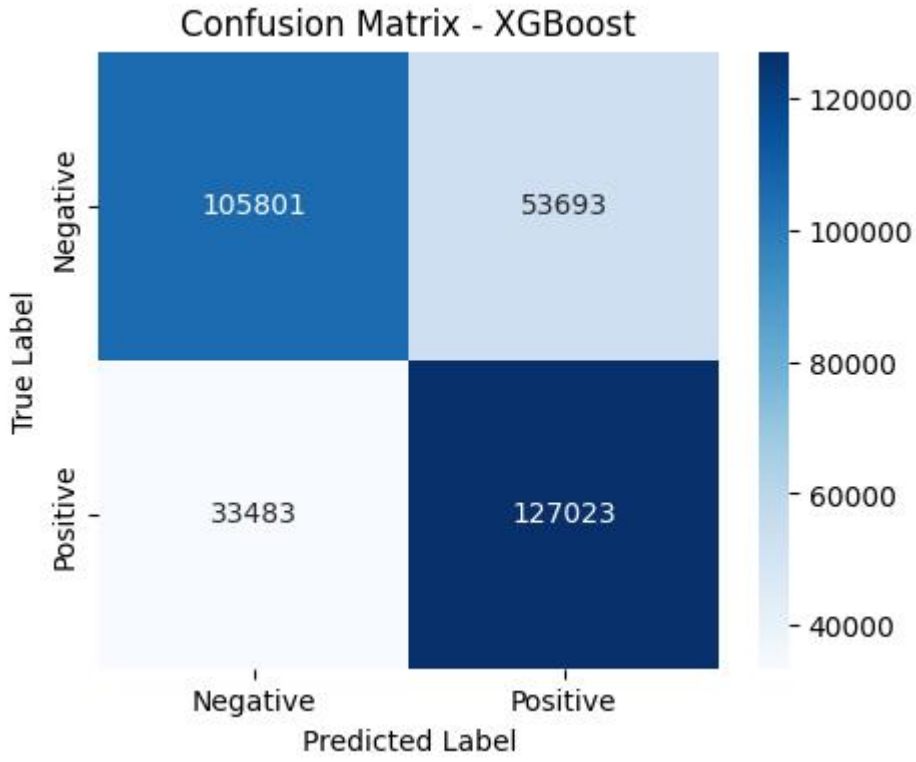
- XGBoost had the lowest accuracy (72.8%), struggling with recall for negative sentiments (66%) but performing decently for positive ones (79% recall).

2. Confusion Matrix Analysis

Each model's confusion matrix provides a deeper look into its predictions.




Confusion Matrix - SGD Classifier
121268
38226
32026
128480
Negative
Positive
True Label
Negative
Positive
Predicted Label
120000
100000
80000
60000
40000


Confusion Matrix - XGBoost
105801
53693
33483
127023
Negative
Positive
True Label
Negative
Positive
Predicted Label
120000
100000
80000
60000
40000

Conclusion from Confusion Matrices:

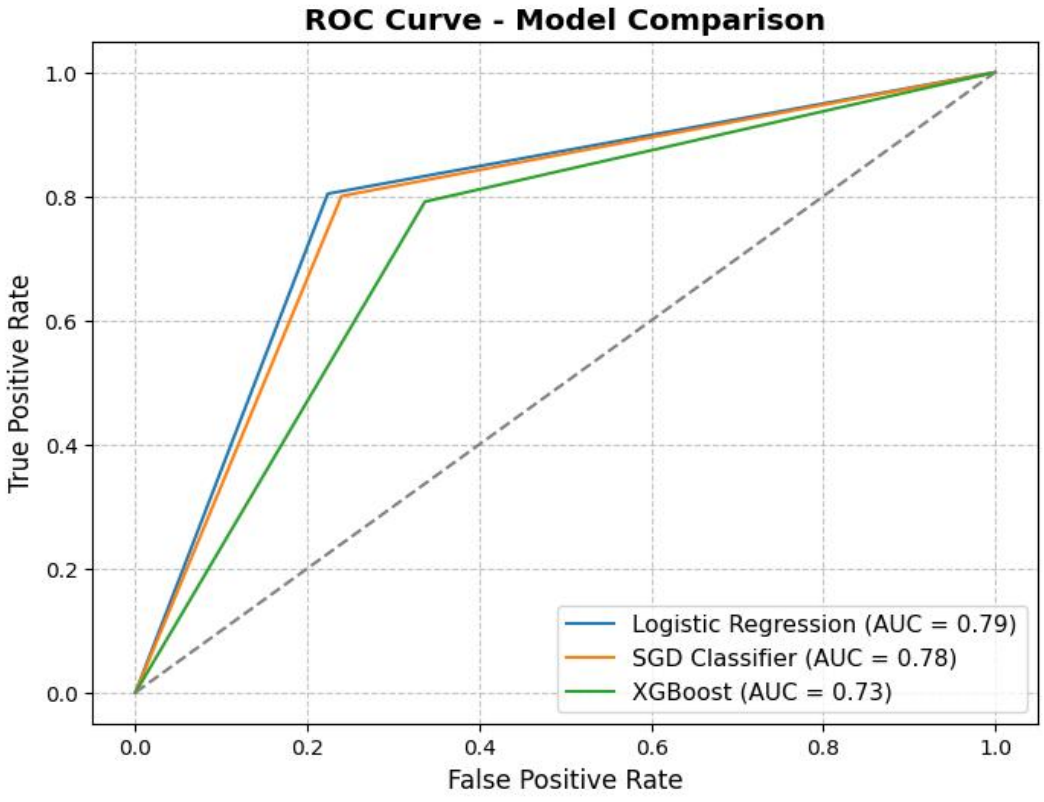
- **Logistic Regression performed best** with the highest correct predictions and balanced sentiment classification.

- **SGD Classifier was slightly weaker**, misclassifying more samples than Logistic Regression.

- **XGBoost struggled the most**, especially with negative sentiment detection, leading to high false positives.

Logistic Regression is the best choice, while SGD may need tuning, and XGBoost requires further optimization

ROC CURVE-



Chapter 6

Conclusion

In this project, we have successfully implemented sentiment analysis on social media data, specifically analyzing tweets to determine whether they express positive or negative sentiments. Using a large dataset of labeled tweets, we applied data preprocessing techniques such as text cleaning, stopword removal, and TF-IDF vectorization to convert raw text into meaningful numerical representations. We then trained and evaluated multiple machine learning models, comparing their effectiveness in sentiment classification.

Among the models tested, **Logistic Regression** demonstrated the highest accuracy (79.0%), making it the most reliable model for this task. It showed a well-balanced precision and recall, ensuring that both positive and negative sentiments were classified accurately. The **SGD Classifier** (78.0%) performed similarly but had a slightly higher misclassification rate, while **XGBoost** (72.8%) struggled the most, particularly in identifying negative sentiments, as indicated by its higher false positive rate.

The confusion matrix analysis further revealed that Logistic Regression had the fewest misclassifications, making it the most suitable model for real-world applications where sentiment accuracy is crucial. The findings indicate that traditional machine learning models, when properly tuned, can effectively analyze sentiment in social media text.

References

1. **Pang, B., & Lee, L. (2008).** *Opinion mining and sentiment analysis.* Foundations and Trends in Information Retrieval.
2. **Pak, A., & Paroubek, P. (2010).** *Twitter as a Corpus for Sentiment Analysis and Opinion Mining.* Proceedings of LREC.
3. **Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011).** *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, 2825–2830.
4. **Bird, S., Klein, E., & Loper, E. (2009).** *Natural Language Processing with Python.* O'Reilly Media, Inc.
5. **Scikit-learn Documentation.** Available at: https://scikit-learn.org/stable/
6. **XGBoost Documentation.** Available at: https://xgboost.readthedocs.io/en/stable/ **Matplotlib & Seaborn Documentation.** Available at: https://matplotlib.org/ and https://seaborn.pydata.org/
7. https://www.kaggle.com/datasets/kazanova/sentiment140 - Dataset Source.