INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN

RESEARCH PROJECT FILE

Tweeting the Future: Predicting Public Opinion and Trends through Sentiment Analysis of Twitter Data

BTECH, ENGINEERING in ELECTRONICS AND COMMUNICATION with ARTIFICIAL INTELLIGENCE

ECEAI-2 (2021-2025)

Submitted by: **KHUSHI**

11501182021

Index

1.Certificate

2. Introduction

3. Technical Approach

- Pre-processing of data
- Exploratory Data Analysis (EDA)
- TF-IDF (Term Frequency-Inverse Document Frequency)
- Model building
- Model evaluation
- Model improvement
- Model Deployment
- Visual model performance

4.Scope of improvement

5.References

Certificate

This is to certify that the work presented in this B. Tech research project tilted ***"Tweeting the Future: Predicting Public Opinion and Trends through Sentiment Analysis of Twitter Data"*** is an authentic record of my own work under the supervision of Dr. Ritu, **Department of Electronics and Communication Engineering**.

It is submitted in partial fulfilment of the requirements for the award of the **Bachelors of Technology in Electronics and communication with artificial intelligence** at **Department of Electronics and communication**, **Indira Gandhi Delhi Technical University for Women.**

This is to certify that this work has been done under my supervision and guidance. It has not been submitted elsewhere either in part or full, for award of any other degree or diploma to the best of my knowledge and belief.

Signature of Candidate

KHUSHI

11501182021

Introduction

Identifying opinions and sentiments from tweets is called as “Twitter Sentiment Analysis”.

- It involves detecting opinions and emotions expressed in tweets. The primary goal is to ascertain the sentiment of a tweet, categorizing it as either positive or negative.
- The Internet's increasing prominence has made it a crucial and cost-effective platform for various activities, particularly social media, which has become an essential part of daily life for social interaction.
- Numerous social media platforms such as Facebook, Instagram, blogs, reviews, and tweets are being processed to extract users' opinions on products, organizations, medical experiences, professional knowledge, or specific situations.
- The underlying attitude and emotions behind these opinions, known as sentiment, are key in assessing individual behavior.
- The code leverages `re` and `nltk` libraries for text preprocessing, removing URLs, mentions, and non-alphabetic characters.
- It standardizes text by lowercasing, filters stopwords using `nltk`, and applies `PorterStemmer` for stemming, thus refining textual data for enhanced efficacy in subsequent natural language processing tasks.
- The model was trained using the training data, fine-tuned, and then evaluated on the test dataset. Predictions were made using the test data, and performance metrics were employed to assess the model's effectiveness.

The objective of this project was to build an effective approach based on the re and nltk model for Twitter sentiment analysis.

Technical Approach

Pre-processing of data

The code utilizes the re and nltk libraries to preprocess text, eliminating URLs, mentions, and nonalphabetic characters. It normalizes the text by converting it to lowercase, removes stopwords using nltk, and employs the PorterStemmer for stemming.

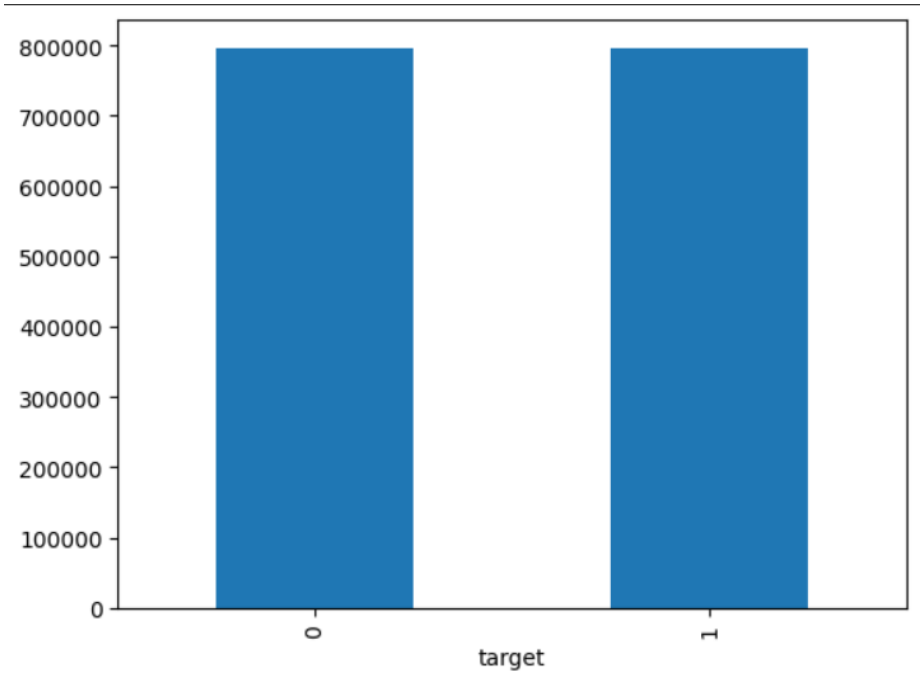
Step 3: Data Preprocessing

```
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def clean_text(text):
    text = re.sub(r'http\S+', '', text)
    text = re.sub(r'@\w+', '', text)
    text = re.sub(r'#', '', text)
    text = re.sub(r'[^A-Za-z\s]', '', text)
    text = text.lower()
    text = ' '.join([ps.stem(word) for word in text.split() if word not in stop_words])
    return text

df['text'] = df['text'].apply(clean_text)
df = df[df['text'].str.strip() != '']
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[2]  df['target'] = df['target'].apply(lambda x: 1 if x == 4 else 0)
```

```
[3]  df['target'].value_counts().plot(kind='bar')
```



EDA (Exploratory Data Analysis)

EDA (Exploratory Data Analysis) is the preliminary examination and visualization of data sets to uncover patterns, anomalies, and relationships, essential for informing subsequent NLP, ML, and AI modelling and decision-making processes.

It was required due to the following reasons-

1. Data Understanding: Helps in comprehending the data's structure, patterns, and relationships.

2. Data Quality: Identifies data quality issues such as missing values, outliers, and inconsistencies.

3. Hypothesis Generation: Facilitates the formulation of hypotheses and guides further analysis.

4. Feature Selection: Aids in selecting relevant features for modelling.

Step 4: Exploratory Data Analysis (EDA)

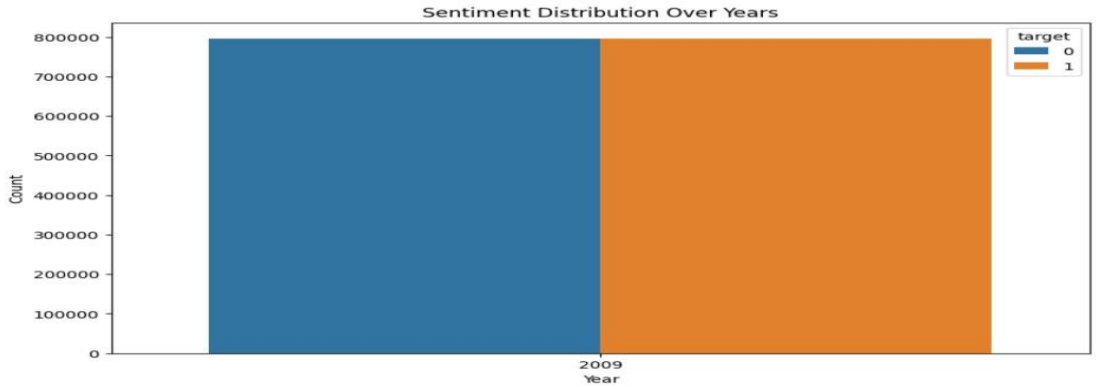
4.1 Basic Statistics and Word Clouds

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

positive_texts = ' '.join(df[df['target'] == 1]['text'])
negative_texts = ' '.join(df[df['target'] == 0]['text'])

wordcloud_positive = WordCloud(max_words=1000, background_color='white').generate(positive_texts)
wordcloud_negative = WordCloud(max_words=1000, background_color='black').generate(negative_texts)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(wordcloud_positive, interpolation='bilinear')
plt.axis('off')
plt.title('Positive Sentiments')

plt.subplot(1, 2, 2)
plt.imshow(wordcloud_negative, interpolation='bilinear')
plt.axis('off')
plt.title('Negative Sentiments')

plt.show()
```

4.2 Sentiment Distribution Over Time

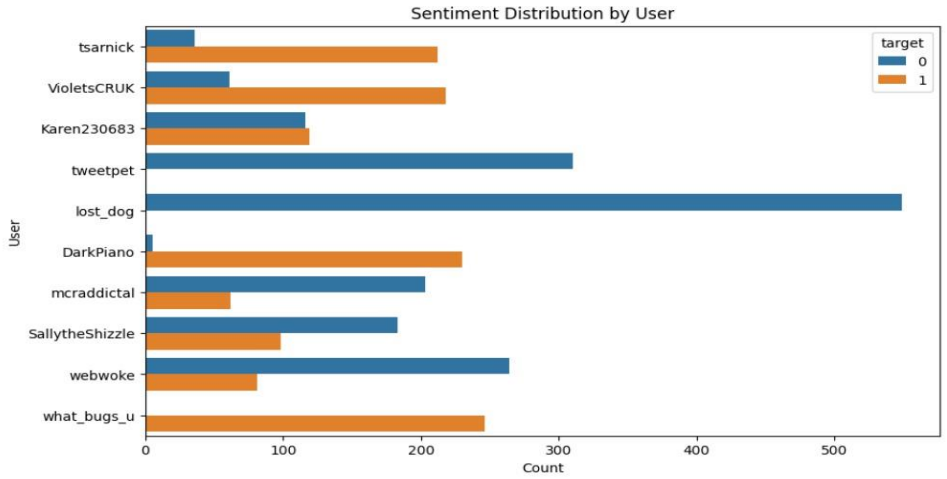
```
# Extract year from the date column
df['date'] = pd.to_datetime(df['date'], errors='coerce')
df['year'] = df['date'].dt.year

# Plot sentiment distribution over years
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='year', hue='target')
plt.title('Sentiment Distribution Over Years')
plt.xlabel('Year')
plt.ylabel('Count')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/dateutil/parser/_parser.py:1207: UnknownTimezoneWarning: tzname PDT identi
  warnings.warn("tzname {tzname} identified but not understood.  "
```



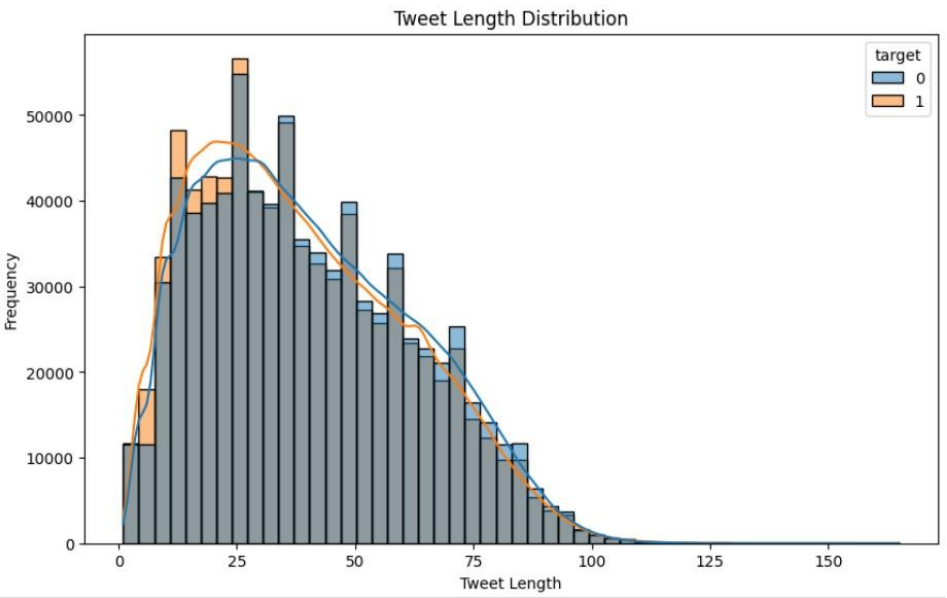
4.3 Sentiment Distribution by User

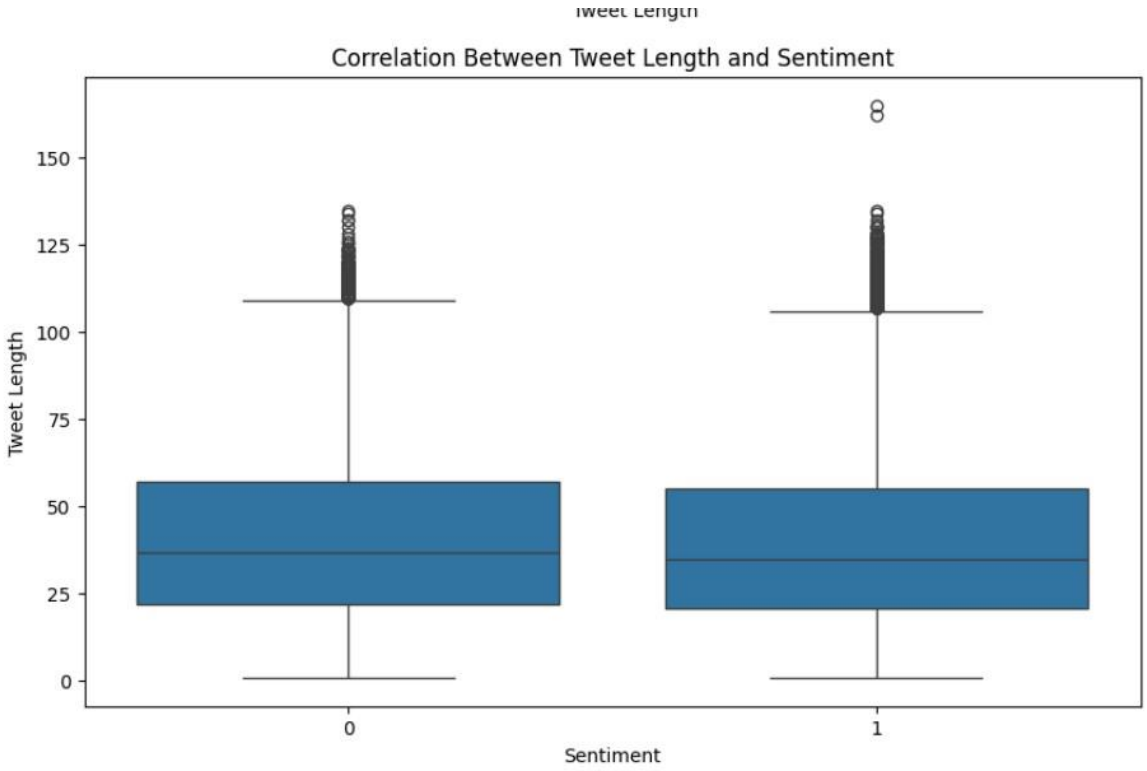
```
# Plot the top 10 users with the most tweets
top_users = df['user'].value_counts().head(10)

plt.figure(figsize=(10, 6))
sns.countplot(data=df[df['user'].isin(top_users.index)], y='user', hue='target')
plt.title('Sentiment Distribution by User')
plt.xlabel('Count')
plt.ylabel('User')
plt.show()
```



4.4 Analyze and Visualize the Length of Tweets

```
# Calculate tweet lengths
df['tweet_length'] = df['text'].apply(len)

# Plot distribution of tweet lengths
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='tweet_length', hue='target', bins=50, kde=True)
plt.title('Tweet Length Distribution')
plt.xlabel('Tweet Length')
plt.ylabel('Frequency')
plt.show()

# Correlation between tweet length and sentiment
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='target', y='tweet_length')
plt.title('Correlation Between Tweet Length and Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Tweet Length')
plt.show()
```





- The code, specifically focuses on text preprocessing to prepare textual data for subsequent analysis or machine learning models.
- Within nltk, the stopwords corpus is downloaded to filter out common words that do not contribute significant semantic value.
- Additionally, the PorterStemmer algorithm is utilized to perform stemming, which reduces words to their base or root form.
- The clean_text function defined in the code systematically removes unwanted elements from the text. It employs regular expressions to strip URLs, user mentions (e.g., @username), hashtags, and any non-alphabetic characters, thereby retaining only letters and spaces.
- The text is then converted to lowercase to ensure uniformity. The function proceeds by tokenizing the text into individual words, filtering out stopwords, and applying stemming to the remaining tokens.
- This process effectively reduces the dimensionality of the text data while preserving the core meaning of the words.
- Subsequently, the clean_text function is applied to the 'text' column of a DataFrame df using the .apply() method, transforming each entry in the column.
- To ensure that no rows with empty text are included in the cleaned dataset, any rows where the cleaned text results in an empty string (after stripping whitespace) are removed.

This comprehensive preprocessing step is crucial for enhancing the quality and effectiveness of the text data, facilitating more accurate and efficient downstream NLP tasks.

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used in natural language processing and information retrieval to evaluate the importance of a word within a document relative to a collection of documents.

It was required due to the following reasons-

1. Feature Importance: Highlights words that are distinctively frequent in a document but relatively rare across the entire corpus, aiding in identifying meaningful terms.

2. Noise Reduction: Mitigates the influence of commonly occurring words (e.g., "the", "is") that offer little discriminatory power in distinguishing documents.

3. Versatility: Applicable to various NLP tasks such as document classification, information retrieval, and keyword extraction, enhancing the effectiveness of text analysis.

4. Normalization: Balances differences in document lengths and word frequencies, ensuring fair comparisons across documents of varying sizes.

5. Interpretability: Produces interpretable results by assigning higher weights to terms that are more indicative of document content, facilitating intuitive understanding and analysis.

Model building

The following methods were utilised:

- **Data Splitting**: The train_test_split function from sklearn.model_selection divides the data (X for text features and y for target labels) into training (X_train, y_train) and testing (X_test, y_test) sets.
- **Text Vectorization**: TfidfVectorizer from sklearn.feature_extraction.text converts text data into numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency), capturing word importance relative to documents.
- **Model Training**: LogisticRegression from sklearn.linear_model is employed for classification. It learns from the TF-IDF transformed training data (X_train_tfidf) to predict the labels (y_train).

Step 5: Model Building

5.1 Train-Test Split

```
from sklearn.model_selection import train_test_split

X = df['text']
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5.2 Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

5.3 Model Training with Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression()
```

Model evaluation

Evaluating the performance of a trained logistic regression model is essential for several reasons:

1. **Model Validation**: It helps in assessing how well the model generalizes to unseen data, ensuring that it is not just performing well on the training data but also on new, unseen data.
2. **Performance Metrics**: Evaluation metrics such as accuracy, precision, recall, and F1 score provide insights into different aspects of the model's performance:

    - **Accuracy**: Measures the proportion of correctly classified instances among all instances.
    - **Precision**: Measures the proportion of true positive instances among the instances predicted as positive.
    - **Recall**: Measures the proportion of true positive instances among all actual positive instances.
    - **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two metrics.

3. **Model Comparison**: Allows for comparing the logistic regression model with other models, facilitating the selection of the best model for the task.

4. **Hyperparameter Tuning**: Evaluation results can guide the tuning of hyperparameters to improve model performance.

5. **Identifying Issues**: Helps in detecting issues such as overfitting, underfitting, or bias in the model, enabling corrective measures to be taken.

6. **Informed Decision-Making**: Provides stakeholders with quantitative measures of model performance, aiding in making informed decisions about deploying the model in a real-world application.

Step 6: Model Evaluation

```
[ ] from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

    y_pred = model.predict(X_test_tfidf)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f'Accuracy: {accuracy}')
    print(f'Precision: {precision}')
    print(f'Recall: {recall}')
    print(f'F1 Score: {f1}')
```

```
Accuracy: 0.7738848403595638
Precision: 0.7654240441713938
Recall: 0.7924474019963476
F1 Score: 0.7787013449938389
```

Model improvement

- The provided code snippet utilizes `GridSearchCV` from `sklearn.model_selection` to perform hyperparameter tuning for a logistic regression model. It defines a grid of hyperparameters, specifically different values for `C` (regularization strength) and `solver` (optimization algorithm).
- The `GridSearchCV` object is created with this parameter grid, set to evaluate models using 5-fold cross-validation and the F1 score as the evaluation metric.
- The grid search is then fit to the training data transformed by TF-IDF. The best model found through this process is selected and used to make predictions on the test data.
- The performance of this best model is evaluated using accuracy, precision, recall, and F1 score metrics, which are then printed along with the best hyperparameters.

This approach ensures that the model is optimized for the best combination of hyperparameters, enhancing its performance on unseen data.

Step 7: Model Improvement

7.1 Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10],
    'solver': ['newton-cg', 'lbfgs', 'liblinear']
}

grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring='f1')
grid_search.fit(X_train_tfidf, y_train)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_tfidf)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Best Model Parameters: {grid_search.best_params_}')
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
```

Model Deployment

- The code demonstrates the use of joblib for saving and loading a trained machine learning model and its corresponding TF-IDF vectorizer. Initially, the joblib.dump function is used to serialize and save the best_model and vectorizer objects to files named 'sentiment_model.pkl' and 'tfidf_vectorizer.pkl', respectively.
- Later, these objects are deserialized and loaded back into memory using joblib.load. A function named predict_sentiment is defined to preprocess new text input using the previously defined clean_text function, transform it using the loaded TF-IDF vectorizer, and then make predictions using the loaded logistic regression model.

The function returns 'Positive' if the prediction is 1, otherwise 'Negative'.

This setup allows for efficient and reusable sentiment prediction on new text data. For instance, the example usage demonstrates predicting the sentiment of the sample text "I love this product!" and printing the result.

Step 8: Model Deployment

8.1 Save the Model

```
[ ] import joblib

    joblib.dump(best_model, 'sentiment_model.pkl')
    joblib.dump(vectorizer, 'tfidf_vectorizer.pkl')
```

```
['tfidf_vectorizer.pkl']
```

8.2 Create a Prediction Script

```
import joblib

# Load model and vectorizer
model = joblib.load('sentiment_model.pkl')
vectorizer = joblib.load('tfidf_vectorizer.pkl')

def predict_sentiment(text):
    text = clean_text(text)
    text_tfidf = vectorizer.transform([text])
    prediction = model.predict(text_tfidf)
    return 'Positive' if prediction == 1 else 'Negative'

# Example usage
sample_text = "I love this product!"
print(predict_sentiment(sample_text))
```

```
Positive
```

Model performance

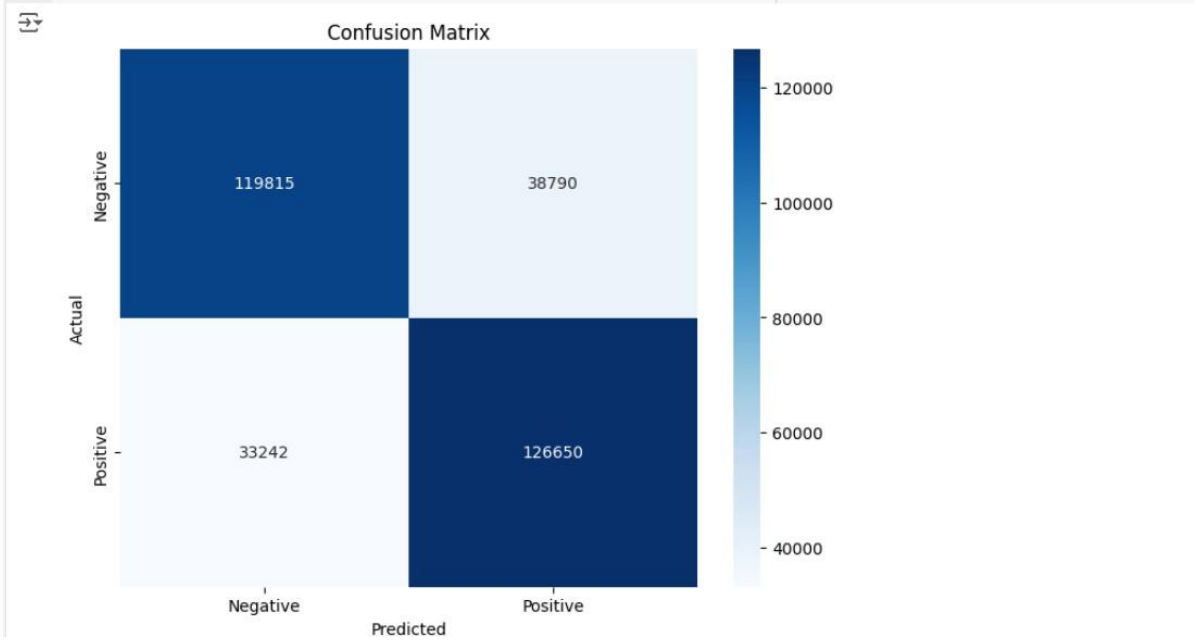
- The code is used to compute and visualize the confusion matrix for a classification model. First, it imports the necessary libraries: `confusion_matrix` from `sklearn.metrics` to compute the confusion matrix, `seaborn` for creating the heatmap, and `matplotlib.pyplot` for plotting. The confusion matrix is computed using the true labels (`y_test`) and the predicted labels (`y_pred`).
- The resulting matrix is then visualized as a heatmap using `seaborn`. The `plt.figure` function sets the figure size, and `sns.heatmap` generates the heatmap with annotations and a blue color map. The axes are labeled with 'Predicted' and 'Actual', and the plot is titled 'Confusion Matrix'.

This visualization helps in understanding the performance of the model by showing the counts of true positive, true negative, false positive, and false negative predictions in a clear and interpretable format.

Step 9: Visualize Model Performance

9.1 Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Confusion matrix -

It is a table with 4 different combinations of predicted and actual values.

True Positives: Model predicted positive and it's true

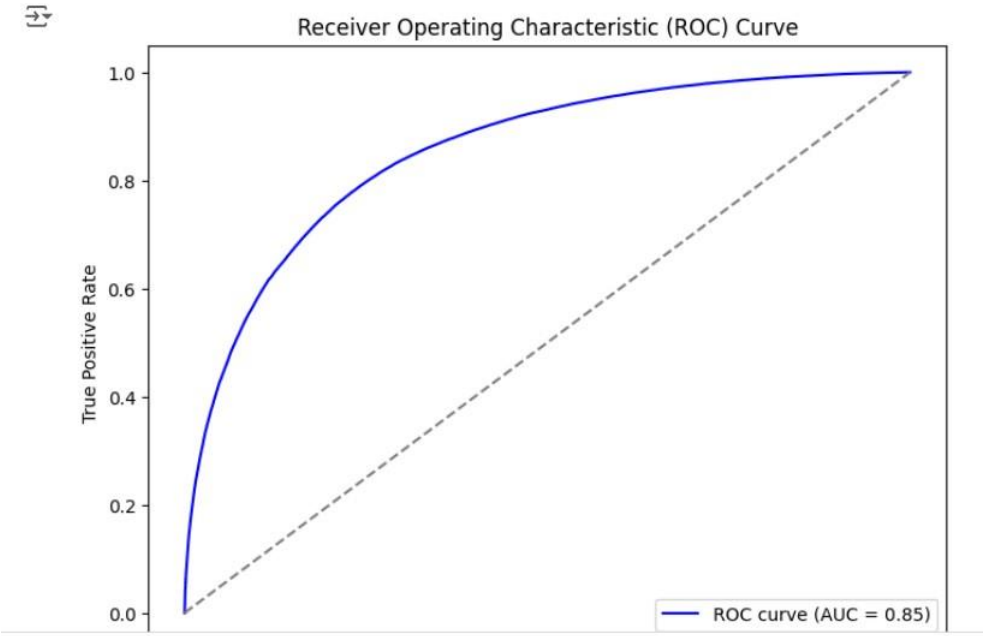
True Negatives: Model predicted negative and it's true.

False Positives: Model predicted positive and it's false.

False Negatives: Model predicted negative and it's false.

9.2 ROC Curve and AUC

```
from sklearn.metrics import roc_curve, roc_auc_score

# Compute predicted probabilities
y_prob = model.predict_proba(X_test_tfidf)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Compute AUC
roc_auc = roc_auc_score(y_test, y_prob)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



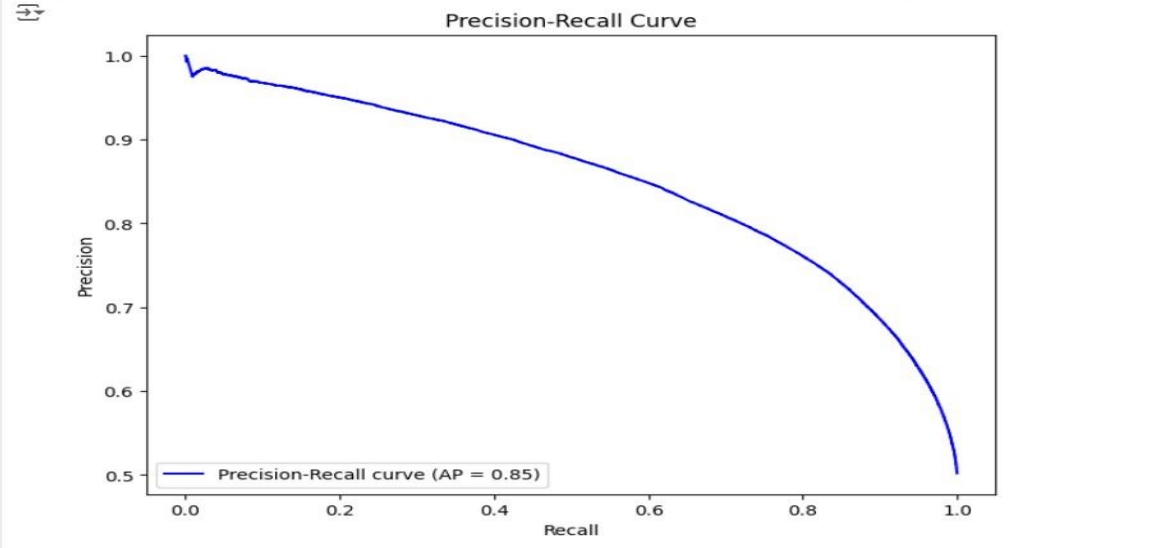
1. The code snippet evaluates the performance of a classification model by computing and visualizing the Receiver Operating Characteristic (ROC) curve and calculating the Area Under the Curve (AUC).
2. The ROC curve is computed using the true labels (`y_test`) and the predicted probabilities (`y_prob`) for the positive class, obtained from the model's `predict_proba` method.
3. The false positive rates (FPR) and true positive rates (TPR) are calculated using the `roc_curve` function, while the AUC, which summarizes the overall performance of the model, is calculated using `roc_auc_score`.
4. The ROC curve is then plotted with FPR on the x-axis and TPR on the y-axis, including a diagonal line representing a random classifier for comparison. The plot is annotated with the AUC value, providing a visual and quantitative assessment of the model's ability to distinguish between positive and negative classes across different thresholds.
5. The provided code snippet evaluates the performance of a classification model by computing and visualizing the precision-recall curve and calculating the average precision score.

6. It first imports the necessary functions, `precision_recall_curve` and `average_precision_score`, from `sklearn.metrics`.
7. The `precision_recall_curve` function calculates precision and recall values for different classification thresholds based on the true labels (`y_test`) and predicted probabilities (`y_prob`) for the positive class. The `average_precision_score` function then computes the average precision, which summarizes the precision-recall curve as a single value.

   This visualization helps assess the trade-off between precision and recall for different thresholds, providing insights into the model's ability to correctly identify positive instances while minimizing false positives.

9.3 Precision-Recall Curve

```
from sklearn.metrics import precision_recall_curve, average_precision_score

# Compute precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_prob)

# Compute average precision
average_precision = average_precision_score(y_test, y_prob)

# Plot precision-recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', label=f'Precision-Recall curve (AP = {average_precision:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```



Conclusion:

The model predicted sentiment of tweets with

- Best Model Parameters: {'C': 1, 'solver': 'newton-cg'}
- Accuracy: 0.7738377441545761
- Precision: 0.7655343326885881
- Recall: 0.7920971655867711
- F1 Score: 0.7785892565133464– which is reasonably good compared random model with auc of 0.50.

Scope of improvement

- To enhance the performance of the model built using TF-IDF vectorization with TfidfVectorizer, several strategies can be employed.
- Firstly, optimizing hyperparameters such as max_features within TfidfVectorizer can significantly impact model performance by finding the optimal balance between feature representation and computational efficiency.
- Utilizing grid search (GridSearchCV) to fine-tune other model-specific parameters like regularization strength (C in logistic regression) can also refine model accuracy.
- Secondly, exploring advanced text preprocessing techniques such as alternative stopword handling methods, stemming, or employing more sophisticated vectorization approaches like word embeddings (e.g., Word2Vec, GloVe) can capture deeper semantic relationships within the text data.

References

Reference has been taken from the following websites:

1. https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/
2. https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-usecase-for-beginners/
3. https://www.kaggle.com/code/paoloripamonti/twitter-sentiment-analysis
4. https://www.youtube.com/watch?v=ng6L_wvREB4
5. https://www.kaggle.com/code/gauravchhabra/nlp-twitter-sentiment-analysis-project
6. https://arxiv.org/pdf/1509.04219