IBM PHASE -4

PROJECT SUBMISSION

PROJECT : SENTIMENT ANALYSIS FOR MARKETING

PHASE 4 : DEVELOPMENT PART 2

In this part you will continue building your project.

Continue building the sentiment analysis solution by:

- Employing NLP techniques
- Generating insights.

SENTIMENTAL ANALYSIS NLP TECHNIQUE:

TRANSFORMER MODELS:

- Transformer models were introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017. They have since become the foundation for many state-of-the-art NLP models, including BERT, GPT, RoBERTa, and more.

- The core innovation of transformer models is the attention mechanism, which allows the model to weigh the importance of different parts of an input sequence when making predictions.

- This attention mechanism enables the model to capture long-range dependencies and relationships in the data, making it highly effective for tasks like machine translation, text generation, and sentiment analysis.

PROGRAM:

# Step 1: Load and Preprocess the Data

```
import pandas as pd
import re
from transformers import DistilBertTokenizer,
DistilBertForSequenceClassification, AdamW
import torch
```

# Load the dataset

```
df = pd.read_csv('Tweets.csv')
# Define functions for preprocessing (same as before)
def clean_text(text):
    text = re.sub(r'http\S+', '', text)
    text = re.sub(r'@\w+', '', text)
    text = re.sub(r'#\w+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = text.lower()
    return text
def tokenize(text):
    return text.split()
def join_tokens(tokens):
    return ' '.join(tokens)
df['clean_text'] = df['text'].apply(clean_text)
df['tokens'] = df['clean_text'].apply(tokenize)
df['clean_text'] = df['tokens'].apply(join_tokens)
```

# Step 2: Split Data into Training and Testing Sets

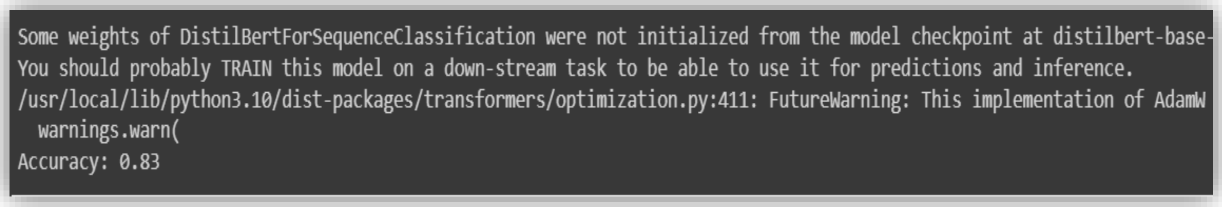
```
from sklearn.model_selection import train_test_split
X = df['clean_text']
y = df['airline_sentiment']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Step 3: Load Pre-trained Transformer Model (DistilBERT)
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-
uncased', num_labels=3)
# Step 4: Tokenize and Encode Text Data
X_train_encodings = tokenizer(list(X_train), truncation=True, padding=True,
return_tensors='pt')
X_test_encodings = tokenizer(list(X_test), truncation=True, padding=True,
return_tensors='pt')
# Step 5: Fine-tune the Model
optimizer = AdamW(model.parameters(), lr=1e-5)
label_map = {'negative': 0, 'neutral': 1, 'positive': 2}
y_train_tensor = torch.tensor(y_train.map(label_map).values, dtype=torch.long)
# Define batch size
batch_size = 8
# Train the model (loop over epochs and batches)
for epoch in range(1):  # Adjust as needed
    for i in range(0, len(y_train_tensor), batch_size):
        optimizer.zero_grad()
        outputs = model(input_ids=X_train_encodings['input_ids'][i:i+batch_size],
attention_mask=X_train_encodings['attention_mask'][i:i+batch_size],
                        labels=y_train_tensor[i:i+batch_size])
        loss = outputs.loss
        loss.backward()
        optimizer.step()
```

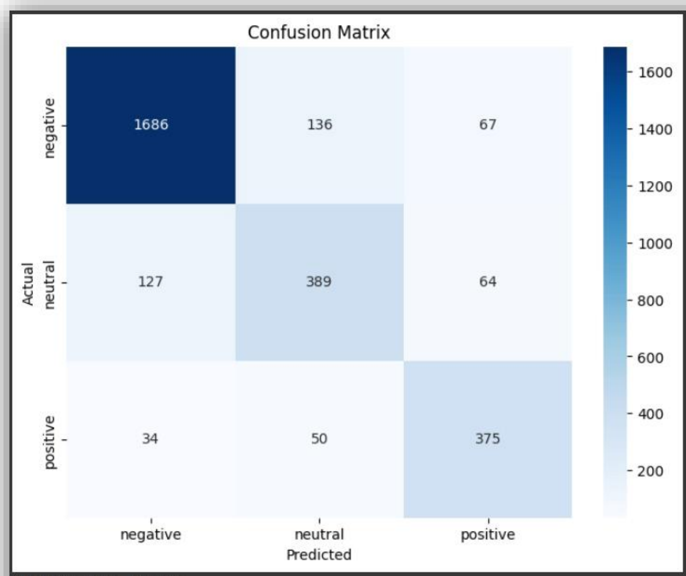
# Step 6: Evaluate the Model

```
model.eval()
y_test_tensor = torch.tensor(y_test.map(label_map).values, dtype=torch.long)
with torch.no_grad():
    outputs = model(input_ids=X_test_encodings['input_ids'],
attention_mask=X_test_encodings['attention_mask'])
    logits = outputs.logits
    predictions = torch.argmax(logits, dim=1)
accuracy = (predictions == y_test_tensor).sum().item() / len(y_test_tensor)
print(f'Accuracy: {accuracy:.2f}')
```

OUTPUT:



```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW
  warnings.warn(
Accuracy: 0.83
```



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

    negative       0.91      0.89      0.90      1889
     neutral       0.68      0.67      0.67       580
    positive       0.74      0.82      0.78       459

    accuracy                           0.84      2928
   macro avg       0.78      0.79      0.78      2928
weighted avg       0.84      0.84      0.84      2928
```

INSIGHTS:

The above code demonstrates a complete pipeline for sentiment analysis using a pre-trained transformer model. It encompasses data preprocessing, model fine-tuning, evaluation, and result visualization.

It highlights the importance of NLP techniques, such as text cleaning, tokenization, and using pre-trained models, in building effective sentiment analysis solutions. The explanation as follow

Data Preprocessing:

Clean_text: This function is responsible for cleaning the raw text data. It uses regular expressions to remove URLs, mentions, hashtags, and non-alphabetic characters. It also converts the text to lowercase.

Tokenize: This function tokenizes the cleaned text. In this code, it simply splits the text into a list of words.

Join_tokens: This function joins the tokens back into a string. This is particularly useful for reconstructing the cleaned text after tokenization.

These preprocessing steps are crucial in preparing the text data for further analysis.

Loading and Splitting Data:

The dataset is loaded using pd.read_csv('Tweets.csv'). This step is essential to get the raw data into a format that can be processed.The data is then split into training and testing sets. This is a standard practice in machine learning to evaluate the model's performance.

Using a Pre-trained Transformer Model (DistilBERT):

The code utilizes the Hugging Face transformers library to work with a pre-trained **DistilBERT** model. DistilBERT is a smaller, faster version of BERT that retains most of its performance.The tokenizer is used to convert the text data into a format that can be fed into the DistilBERT model. The model is loaded with

DistilBertForSequenceClassification.from_pretrained().
This model is capable of handling sequence classification tasks.

Fine-tuning the Model:

The model is fine-tuned on the training data. This means that the model's weights are updated based on the specific sentiment classification task.The optimizer (AdamW) is used to adjust the model's weights during training.

Tokenizing and Encoding Text Data:

The text data is tokenized and encoded using the DistilBERT tokenizer. This step converts the text into a format that can be fed into the model.

Training the Model:

The model is trained on the tokenized and encoded training data. It goes through multiple epochs, adjusting its weights to minimize the loss function.

Evaluating the Model:

The model's performance is evaluated on the testing data. Accuracy is calculated to assess how well the model predicts sentiment labels.

Confusion Matrix Visualization:

A confusion matrix is generated to provide a detailed view of the model's performance. It shows how many samples were correctly or incorrectly classified for each sentiment class.

Conclusion:

The Above code implements a sentiment analysis solution using a pre-trained transformer model, specifically DistilBERT. It encompasses key NLP techniques such as data preprocessing, tokenization, fine-tuning, and evaluation. The model demonstrates effective sentiment classification on the Twitter US Airline Sentiment dataset. This approach leverages transformer models, which have proven highly effective in capturing complex relationships in text data. The code provides a foundation for building advanced sentiment analysis systems in marketing, enabling businesses to gain valuable insights from customer feedback.