CONTENTS

TRANSFORMER-BASED SENTIMENT ANALYSIS PROJECT DOCUMENTATION

1. INTRODUCTION

This document provides comprehensive documentation for a Twitter Sentiment Analysis project that utilizes a BERT-based transformer model to classify text into three sentiment categories: positive, negative, and neutral. The project covers the entire machine learning pipeline, from data preparation to deployment, including exploratory data analysis (EDA), model training, evaluation, visualization, inference, and model persistence.

The dataset used consists of 15,000 tweets with columns for the full tweet text, a selected substring driving the sentiment, and the target sentiment label. The project leverages the Hugging Face Transformers library and PyTorch to implement the sentiment analysis system.

2. DATA PREPARATION

• DATA LOADING AND INITIAL EXPLORATION

- The dataset is loaded into a pandas DataFrame from a CSV file named **`DATASET.CSV`**. The DataFrame contains three columns: **`TEXT`**, **`SELECTED_TEXT`**, and **`SENTIMENT`**. The **`TEXT`** column holds the full tweet, **`SELECTED_TEXT`** contains a substring of the tweet that drives the sentiment, and **`SENTIMENT`** is the target label with values "positive", "negative", or "neutral".
- The dataset has 15,000 entries with no missing values, as confirmed by checking for null values in each column.
- An initial inspection of the first five rows is performed to verify the data structure.

• DATA CLEANING

Text preprocessing is crucial for effective sentiment analysis. The following steps are applied to clean the text data:

1. Convert text to lowercase.
2. Remove text in square brackets.
3. Remove links.
4. Remove punctuation.
5. Remove stop words.
6. Remove words containing numbers.

```python
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'\[.*?\]', '', text)  # Remove brackets content
    text = re.sub(r'http\S+', '', text)  # Remove links
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\w*\d\w*', '', text)  # Remove words with numbers
    return text
```

Stop Words Removal:

- Using NLTK's English stop words:

```python
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
def remove_stopword(x):
    return [y for y in x.split() if y not in stop_words]
train['temp_list'] = train['text'].apply(lambda x: remove_stopword(clean_text(x)))
```

• Data Splitting

Features and Target:

- X = train.drop(columns=['sentiment']): Features (text data).
- y = train['sentiment']: Target variable.

Train-Test Split:

- 80/20 split with stratification to maintain class distribution

• Exploratory Data Analysis (EDA)

Sentiment Distribution: Visualized using a bar chart to check for class imbalance.

Statistics:

- text: 15,000 unique entries.
- selected_text: 12,583 unique entries, "good" most frequent (109 times).
- sentiment: 3 unique values, "neutral" most frequent (6,116 times).

Common Words: Analyzed for positive, negative, and neutral categories (e.g., "love" for positive, "hate" for negative).

3. Model Selection and Setup

- ### Library Installation

- **Transformers**: Installed via: !pip install transformers.
- **Dependencies**: PyTorch, NumPy, SciPy for model operations.

- ### Model Choices

Options:

- cardiffnlp/twitter-roberta-base-sentiment-latest
- distilbert-base-uncased
- bert-base-uncased (selected).

Reason: bert-base-uncased chosen for its balance of performance and resource efficiency.

- ### Tokenizer and Model Initialization

- **Tokenizer**:
  Pads/truncates text to 128 tokens.

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
def tokenize_function(text):
    return tokenizer(text, padding="max_length", truncation=True, max_length=128,
return_tensors="pt")
```

- **Model**:
  Configured for 3-class classification (positive, negative, neutral).

```
import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

4. Model Training

- ### Data Preparation for Training

- **Test Data**:
  Similar process for training data (not explicitly shown but implied).

```
test_text = X_test["text"].tolist()
encoded_test = tokenize_function(test_text)
test_input_ids = encoded_test['input_ids']
test_attention_masks = encoded_test['attention_mask']
test_labels = torch.tensor(y_test.values)
from torch.utils.data import TensorDataset, DataLoader
test_dataset = TensorDataset(test_input_ids, test_attention_masks, test_labels)
test_dataloader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

- ### Training Loop

- **Scheduler**:

```
from transformers import get_linear_schedule_with_warmup
epochs = 3
total_steps = len(train_dataloader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,
num_training_steps=total_steps)
```

- **Training**:
  Trains for 3 epochs with batch processing, backpropagation, and learning rate scheduling.

```
for epoch in range(epochs):
    model.train()
    for batch in train_dataloader:
        batch_input_ids, batch_attention_masks, batch_labels = [b.to(device) for b
in batch]
        outputs = model(batch_input_ids, attention_mask=batch_attention_masks,
labels=batch_labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()
    print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss.item()}")
```

5. Model Evaluation

- ### Simple Evaluation

- **Code**:
- **Metrics**: Accuracy and weighted F1 score.

```
model.eval()
predictions, true_labels = [], []
with torch.no_grad():
    for batch in test_dataloader:
        input_ids, attention_mask, labels = [b.to(device) for b in batch]
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions.extend(torch.argmax(logits, dim=-1).cpu().numpy())
        true_labels.extend(labels.cpu().numpy())
from sklearn.metrics import accuracy_score, f1_score
print("Accuracy:", accuracy_score(true_labels, predictions))
print("F1 Score:", f1_score(true_labels, predictions, average='weighted'))
```

- ### Comprehensive Evaluation

- **Function**:
- **Output**: Average loss, accuracy, F1 score, predictions, and true labels.

```
def evaluate(model, dataloader, device):
    model.eval()
    predictions, true_labels, total_loss = [], [], 0.0
    with torch.no_grad():
        for batch in tqdm(dataloader, desc="Evaluating"):
            input_ids, attention_mask, labels = [b.to(device) for b in batch]
            outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
            logits = outputs.logits
            loss = outputs.loss
            total_loss += loss.item()
            predictions.extend(torch.argmax(logits, dim=1).cpu().numpy())
            true_labels.extend(labels.cpu().numpy())
    avg_loss = total_loss / len(dataloader)
    accuracy = accuracy_score(true_labels, predictions)
    f1 = f1_score(true_labels, predictions, average='weighted')
    return avg_loss, accuracy, f1, predictions, true_labels
```

6. Visualization

Confusion Matrix:

- **Purpose**: Visualizes classification performance and provides precision, recall, and F1 scores per class.

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
cm = confusion_matrix(true_labels, predictions)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative',
'Positive', 'Neutral'], yticklabels=['Negative', 'Positive', 'Neutral'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
print("Classification Report:")
print(classification_report(true_labels, predictions, target_names=['Negative',
'Positive', 'Neutral']))
```

7. Inference

- Inference Function
- **Code**:

Features: Tokenizes input, performs inference, and maps numeric outputs to labels (e.g., 0 → "negative").

```
def predict_sentiment(text, model, tokenizer, device, label_mapping=None):
    model.eval()
    encoded = tokenizer(text, padding='max_length', max_length=128,
truncation=True, return_tensors='pt')
    input_ids = encoded['input_ids'].to(device)
    attention_mask = encoded['attention_mask'].to(device)
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        prediction = torch.argmax(logits, dim=1).cpu().numpy()[0]
    if label_mapping:
        reverse_mapping = {v: k for k, v in label_mapping.items()}
        return reverse_mapping.get(prediction, prediction)
    return prediction
```

- EXAMPLE USAGE
- **Code**:

```
label_mapping = {'negative': 0, 'neutral': 1, 'positive': 2}
test_examples = [
    "I absolutely love this product, it's amazing!",
    "This is the worst experience I've ever had.",
    "The movie was okay, nothing special."
]
for text in test_examples:
    predicted_label = predict_sentiment(text, model, tokenizer, device,
label_mapping)
    print(f"Text: {text}")
    print(f"Predicted Sentiment: {predicted_label}")
```

8. MODEL PERSISTENCE

- **Saving:**

```
model.save_pretrained("sentiment_model")
tokenizer.save_pretrained("sentiment_model")
```

- **Simplified Prediction**:

Purpose: Saves model and tokenizer for reuse; provides a lightweight inference function.

```
def predict_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True,
max_length=128)
    inputs = {key: val.to(device) for key, val in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    return torch.argmax(outputs.logits, dim=1).item()
```

9. Deployment

- ### Streamlit App

- **Code**:

Features: Interactive web interface with text input and example predictions.

```
import streamlit as st
model = BertForSequenceClassification.from_pretrained("sentiment_model")
tokenizer = BertTokenizer.from_pretrained("sentiment_model")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.eval()
def predict_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True,
max_length=128)
    inputs = {key: val.to(device) for key, val in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    prediction = torch.argmax(outputs.logits, dim=1).item()
    label_mapping = {0: "negative", 1: "neutral", 2: "positive"}
    return label_mapping[prediction]
st.title("Sentiment Analysis with BERT")
user_input = st.text_area("Input Text", "I love this product!")
if st.button("Predict"):
    prediction = predict_sentiment(user_input)
    st.success(f"Predicted Sentiment: **{prediction}**")
```

- ### Gradio App

- **Code**:

Features: Simple UI with emoji-enhanced outputs.

```
import gradio as gr
def predict(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=128)
    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    pred = torch.argmax(outputs.logits).item()
    return ["Negative 😠", "Neutral 😐", "Positive 😊"][pred]
interface = gr.Interface(
    fn=predict,
    inputs=gr.Textbox(label="Enter your text"),
    outputs=gr.Label(label="Predicted Sentiment"),
    title="Twitter Sentiment Analysis 🐦"
)
interface.launch()
```

10. SUMMARY

This project implements a robust sentiment analysis system using BERT, covering:

1. **Data Preparation**: Cleaning and splitting 15,000 tweets.
2. **Model Training**: Fine-tuning BERT for 3-class classification.
3. **Evaluation**: Comprehensive metrics and visualizations.
4. **Inference**: Efficient prediction on new inputs.
5. **Deployment**: Web-based access via Streamlit and Gradio.

The system leverages modern NLP tools (Hugging Face, PyTorch) and provides a practical solution for sentiment analysis, with potential applications in social media monitoring and beyond.

11. TEAM MEMBERS:

1. Ibrahim Abdelsattar (Team Leader) (Analysis, Documentation, Presentation)
2. Amr Belal (Modeling)
3. Nour Mostafa (Modeling)
4. Moaz Ramadan (Deployment)
5. Noran Alaa (Deployment)