**Applying BERT Model on the Clift Dataset**

The process of applying the BERT (Bidirectional Encoder Representations from Transformers) model to the Clift dataset for sentiment analysis. Sentiment analysis involves determining the sentiment or emotional tone of text data, which can be positive, negative, or neutral.

# **Data Preparation**

We start by importing the necessary libraries and loading the Clift dataset. The dataset contains various columns, but for sentiment analysis, we focus on the text data and sentiment labels.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder import pandas as pd import nltk from nltk.corpus import sentiwordnet as swn from sklearn.preprocessing import LabelEncoder from sklearn.model\_selection import train\_test\_split import matplotlib.pyplot as plt from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Dropout from tensorflow.keras.optimizers import Adam from nltk.corpus import stopwords from nltk.stem import WordNetLemmatizer from nltk.tokenize import word\_tokenize from textblob import TextBlob from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix import numpy as np from tensorflow.keras.utils import to\_categorical import seaborn as sns  # Download NLTK resources nltk.download('punkt') nltk.download('sentiwordnet') nltk.download('stopwords') nltk.download('wordnet') nltk.download('averaged\_perceptron\_tagger')  # Load the dataset df = pd.read\_excel('/content/Clift Data.xlsx')  x = df.iloc[:, 3].values y = df.iloc[:, 1].values  x = df.iloc[:, 3].astype(str) |

## **Data Preprocessing**

### **Text Preprocessing**

To prepare the text data for sentiment analysis, we perform the following steps:

1. **Text Cleaning**: We remove any special characters, digits, and punctuation from the text. Additionally, we tokenize the text.
2. **Stopword Removal**: Common English stopwords are removed to focus on meaningful words.
3. **Lemmatization**: We lemmatize the words to reduce them to their base form.
4. **Sentiment Analysis**: We use TextBlob to perform sentiment analysis and assign sentiment scores and labels (positive, negative, neutral) to each text.

|  |
| --- |
| # Initialize stopwords, lemmatizer, and POS tagger stop\_words = set(stopwords.words('english')) lemmatizer = WordNetLemmatizer()  # Initialize TextBlob for sentiment analysis def analyze\_sentiment(text):     blob = TextBlob(text)     sentiment\_score = blob.sentiment.polarity      if sentiment\_score > 0:         sentiment\_label = "Positive"     elif sentiment\_score < 0:         sentiment\_label = "Negative"     else:         sentiment\_label = "Neutral"      return sentiment\_score, sentiment\_label  def preprocess\_text\_with\_pos(text):     # Tokenization     tokens = word\_tokenize(text)      # Removing stopwords and non-alphabetic tokens     tokens = [token.lower() for token in tokens if isinstance(token, str) and token.isalpha() and token.lower() not in stop\_words]      # POS tagging     pos\_tags = nltk.pos\_tag(tokens)      # Lemmatization using POS tags     lemmatized\_tokens = []     for token, pos\_tag in pos\_tags:         pos = 'n'  # Default POS tag for lemmatization is 'n' (noun)         if pos\_tag.startswith('J'):             pos = 'a'  # Adjective         elif pos\_tag.startswith('V'):             pos = 'v'  # Verb         elif pos\_tag.startswith('R'):             pos = 'r'  # Adverb          lemmatized\_token = lemmatizer.lemmatize(token, pos)         lemmatized\_tokens.append(lemmatized\_token)      return lemmatized\_tokens |

## **Data Transformation**

The preprocessed data is then transformed into a DataFrame for further analysis and saved to an Excel file.

|  |
| --- |
| # Initialize lists to store preprocessed data preprocessed\_data = []  # Process each text for i, text in enumerate(x):     # ...  # Create a DataFrame from the preprocessed data preprocessed\_df = pd.DataFrame(preprocessed\_data)  # Save the preprocessed DataFrame to an Excel file preprocessed\_df.to\_excel('Clift\_Preprocessed.xlsx', index=False)  # Display the preprocessed data print(preprocessed\_df) |

## **BERT Model Training**

We continue by training a BERT-based model on the preprocessed text data.

|  |
| --- |
| # Convert each element in x to string x = [str(text) for text in x]  # Tokenize the text using the BERT tokenizer tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased', do\_lower\_case=True) encoded\_inputs = tokenizer(x, padding=True, truncation=True, max\_length=20, return\_tensors='tf')  # Convert the labels to categorical format label\_encoder = LabelEncoder() y = label\_encoder.fit\_transform(y) num\_classes = len(label\_encoder.classes\_) y = to\_categorical(y, num\_classes)  # Convert TensorFlow tensor to NumPy array X = encoded\_inputs['input\_ids'].numpy()  # Reshape X to match the expected shape X = X.reshape(X.shape[0], -1)  # Split the data into training and testing sets X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2)  # Create the neural network model model = Sequential() model.add(Dense(units=64, activation='relu', input\_dim=X\_train.shape[1])) model.add(Dropout(0.5)) model.add(Dense(units=num\_classes, activation='softmax'))  # Compile the model model.compile(optimizer=Adam(learning\_rate=2e-5), loss='categorical\_crossentropy', metrics=['accuracy'])  # Train the model history = model.fit(X\_train, y\_train, epochs=30, batch\_size=32, validation\_data=(X\_test, y\_test)) |

## **Model Evaluation**

After training the model, we evaluate its performance using various metrics such as accuracy, precision, recall, F1-score, and confusion matrix.

|  |
| --- |
| # Evaluate the model y\_pred = model.predict(X\_test) y\_pred = y\_pred.argmax(axis=1) y\_test = y\_test.argmax(axis=1)  accuracy = accuracy\_score(y\_test, y\_pred) precision = precision\_score(y\_test, y\_pred, average='weighted') recall = recall\_score(y\_test, y\_pred, average='weighted') f1 = f1\_score(y\_test, y\_pred, average='weighted') support = len(y\_test)  print("Test accuracy:", accuracy) print("Precision:", precision) print("Recall:", recall) print("F1 score:", f1) print("Support:", support) |

## **Conclusion**

In this document, we have applied the BERT model to the Clift dataset for sentiment analysis. We performed data preprocessing to clean and transform the text data and then trained a neural network model using BERT embeddings. The model was evaluated for its performance on sentiment classification, and various evaluation metrics were reported. This demonstrates the application of state-of-the-art models like BERT in real-world sentiment analysis tasks. Further fine-tuning and experimentation can lead to improved model performance.