**SOCIAL NETWORK ANALYSIS DIGITAL ASSIGNMENT-3**

Name: Matam Tejaswini

Regno: 22MCB0034

**To compare the performance of the four models you mentioned (Naive Bayes, SVM, Decision Tree, BI-LSTM with word2vec/fastText word embedding, and Transformer-based model with BERT-based word embedding),**

* Step 1: Dataset Preparation

- Collect or obtain a dataset suitable for sentiment analysis, consisting of tweets or customer reviews labeled with sentiment classes (positive, negative, neutral).

- Split the dataset into training and testing sets. Ensure a balanced distribution of classes in both sets.

* Step 2: Preprocessing

- Perform text preprocessing steps such as lowercasing, removing stop words, and punctuation.

- Tokenize the text into individual words or subwords.

* Step 3: Feature Extraction

- For the classical machine learning models (Naive Bayes, SVM, Decision Tree), use traditional feature extraction techniques such as TF-IDF or bag-of-words to represent the text data.

- For BI-LSTM with word2vec/fastText word embedding and Transformer-based model with BERT-based word embedding, utilize pre-trained word embeddings to capture semantic information.

* Step 4: Model Training and Evaluation

- Train each model using the prepared dataset and appropriate feature representations.

- Evaluate the models using suitable evaluation metrics such as accuracy, precision, recall, and F1-score.

- Generate necessary plots (e.g., bar graphs, line charts) to visualize and compare the performance of the models.

* Step 5: Comparative Analysis

- Analyze the results and compare the performance of the four models based on the evaluation metrics.

- Discuss the strengths and weaknesses of each model.

- Identify which model performs the best and provide reasoning behind your conclusion.

Here's a brief overview of what we can expect from each model:

**1) Classical Machine Learning Models:**

- Naive Bayes: Known for its simplicity and efficiency, Naive Bayes is often used as a baseline model. It assumes that features are conditionally independent given the class label.

- SVM: Support Vector Machines are powerful models that aim to find an optimal hyperplane to separate different classes. They can handle high-dimensional feature spaces effectively.

- Decision Tree: Decision trees create a flowchart-like structure based on feature values to make decisions. They are interpretable and can capture non-linear relationships.

**2) BI-LSTM with Word2Vec/fastText Word Embedding:**

- BI-LSTM (Bidirectional Long Short-Term Memory) is a recurrent neural network architecture capable of capturing sequential dependencies in text data.

- Word2Vec and fastText are popular word embedding techniques that represent words as dense vectors, capturing semantic information.

**3) Transformer-based Model with BERT-based Word Embedding**:

- Transformers, particularly models like BERT (Bidirectional Encoder Representations from Transformers), have achieved state-of-the-art performance in various natural language processing tasks.

- BERT-based word embeddings capture contextual information by considering the entire input sequence.

**Evaluation of models is based on**

1. Accuracy—the proportion of the total number of correct predictions over the total number of cases examined, as given in Eq. (1): where TP – true positive; TN – true negative; FP – false positive; FN – false negative [55].

2. Precision—the ratio of true positive results over the total number of positive predictions (including true positive and false positive) by the model (Eq. 2). where TP – true positive; FP – false positive [55].

3. Recall—the proportion of actual positive cases which are correctly identifed (Eq. 3). where TP – true positive; FN – false negative [55].

4. F-measure—the harmonic mean between precision and recall, and the range of F-measure is between 0 and 1. Greater value of F-measure indicates better performance of the model. The formula for determining F-measure is:

5. Area under the ROC (Receiver Operating Characteristic) curve (AUC-ROC)— Similar to F1-score, AUC has also the range of 0 and 1. The higher the score for AUC, the better the performance. ROC curve is a graph that shows the plot between sensitivity (true positive rate) and (1-specifcity) (false positive rate).

**SENTIMENT ANALYSIS FOR NAÏVE BAYES:**

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

# Step 1: Load and prepare the dataset

data = pd.read\_csv('/content/Test[1].csv')  # Replace 'sentiment\_dataset.csv' with your dataset file

texts = data['text']

labels = data['label']

# Step 2: Split the dataset into training and testing sets

texts\_train, texts\_test, labels\_train, labels\_test = train\_test\_split(texts, labels, test\_size=0.2, random\_state=42)

# Step 3: Feature extraction using CountVectorizer

vectorizer = CountVectorizer()

vectorizer.fit(texts\_train)

features\_train = vectorizer.transform(texts\_train)

features\_test = vectorizer.transform(texts\_test)

# Step 4: Train the Naive Bayes classifier

naive\_bayes = MultinomialNB()

naive\_bayes.fit(features\_train, labels\_train)

# Step 5: Make predictions on the test set

predictions = naive\_bayes.predict(features\_test)

# Step 6: Evaluate the model

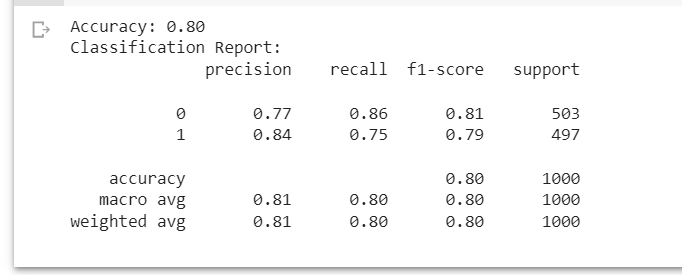
accuracy = accuracy\_score(labels\_test, predictions)

print(f"Accuracy: {accuracy:.2f}")

classification\_report = classification\_report(labels\_test, predictions)

print("Classification Report:")

print(classification\_report)



**SENTIMENT ANALYSIS FOR SVM:**

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

# Step 1: Load and prepare the dataset

data = pd.read\_csv('/content/Test[1].csv')  # Replace 'sentiment\_dataset.csv' with your dataset file

texts = data['text']

labels = data['label']

# Step 2: Split the dataset into training and testing sets

texts\_train, texts\_test, labels\_train, labels\_test = train\_test\_split(texts, labels, test\_size=0.2, random\_state=42)

# Step 3: Feature extraction using TF-IDF vectorization

vectorizer = TfidfVectorizer()

vectorizer.fit(texts\_train)

features\_train = vectorizer.transform(texts\_train)

features\_test = vectorizer.transform(texts\_test)

# Step 4: Train the SVM classifier

svm = SVC(kernel='linear')

svm.fit(features\_train, labels\_train)

# Step 5: Make predictions on the test set

predictions = svm.predict(features\_test)

# Step 6: Evaluate the model

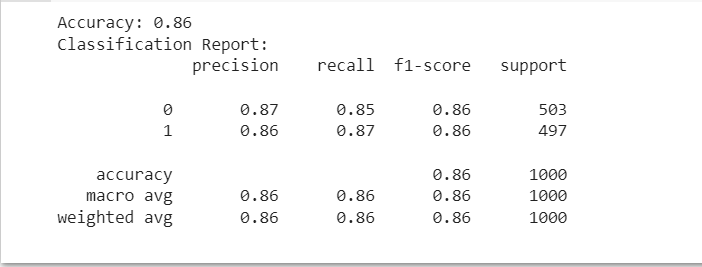
accuracy = accuracy\_score(labels\_test, predictions)

print(f"Accuracy: {accuracy:.2f}")

classification\_report = classification\_report(labels\_test, predictions)

print("Classification Report:")

print(classification\_report)



**WORD2VEC, FASTEXT, BERT VARIANT MODEL**

**Word2Vec**: a pre-trained model that learns the relationship between the words in a corpus, and returns an embedded vector for each word in the text [42],

2. **FastText:** an extension of Word2Vec that breaks words into n-grams (smaller parts), e.g., ‘apple’ to ‘app’ with the intention of learning the morphology of the words. The model also returns a bag of embedded vectors for each word in the text [43]. Word2Vec and FastText might not handle polysemous words (i.e., words with multiple meaning) as they are deemed to be context-free (i.e., map the same word to the same embedding vector). For example, ‘fre’ would have the same representation in ‘building on fre’ and ‘fre someone.’ To mitigate this problem, scholars have begun to explore transformer-based embeddings, including **BERT** and its variants. **BERT-variant models** were pre-trained by incorporating the context of the word within the text in Wikipedia and BooksCorpus [44], and the embedding are then used through a classifer for predictions. As they produce contextualized word embeddings, they produce state-ofthe-art results on Natural Language Processing tasks [12, 34]. The BERT-base model is a bi-directional (both left-to-right and rightto-left direction) transformer for pre-training over a lot of unlabeled textual data to learn a language representation that can be used to fne-tune for specifc classifcation tasks (see [44] for further details). One of its popular variant is RoBERTa (Robustly Optimized BERT approach), which was introduced by Facebook. It is basically an improved version of BERT, capable of handling more data with higher computing power. Compared to BERT, RoBERTa has been shown to have a higher prediction power. Finally, Google and Toyota developed a smaller/smarter BERT variant known as A Lite BERT (ALBERT), which is dramatically smaller in size compared to BERT. The present study examined BERT-base model and two of its variants, that is, RoBERTa and ALBERT

**SENTIMENTAL ANALYSIS USING WORD2VEC OR FASTEXT**

import pandas as pd

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.preprocessing.text import Tokenizer

from gensim.models import Word2Vec

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

# Load the dataset

data = pd.read\_csv('/content/Test[1].csv')

# Preprocess the data

X = data['text']  # Input text

y = data['label']  # Sentiment labels

# 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=42)

# Tokenize the text data and convert it to sequences

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(X\_train)

X\_train\_sequences = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_sequences = tokenizer.texts\_to\_sequences(X\_test)

# Pad the sequences to a fixed length

max\_sequence\_length = 100

X\_train\_padded = pad\_sequences(X\_train\_sequences, maxlen=max\_sequence\_length)

X\_test\_padded = pad\_sequences(X\_test\_sequences, maxlen=max\_sequence\_length)

# Create the list of sentences for Word2Vec or fastText training

sentences = [text.split() for text in X\_train]

# Load pre-trained Word2Vec or fastText word embeddings

word2vec\_model = Word2Vec(sentences=sentences, vector\_size=100, window=5, min\_count=1)

# Create an embedding matrix

embedding\_dim = 100

vocab\_size = len(tokenizer.word\_index) + 1

embedding\_matrix = np.zeros((vocab\_size, embedding\_dim))

for word, index in tokenizer.word\_index.items():

    if word in word2vec\_model.wv.key\_to\_index:

        embedding\_matrix[index] = word2vec\_model.wv[word]

# Build the BI-LSTM model

model = Sequential()

model.add(Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=max\_sequence\_length, weights=[embedding\_matrix], trainable=False))

model.add(LSTM(units=64, return\_sequences=True))

model.add(LSTM(units=64))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train\_padded, y\_train, epochs=10, batch\_size=32)

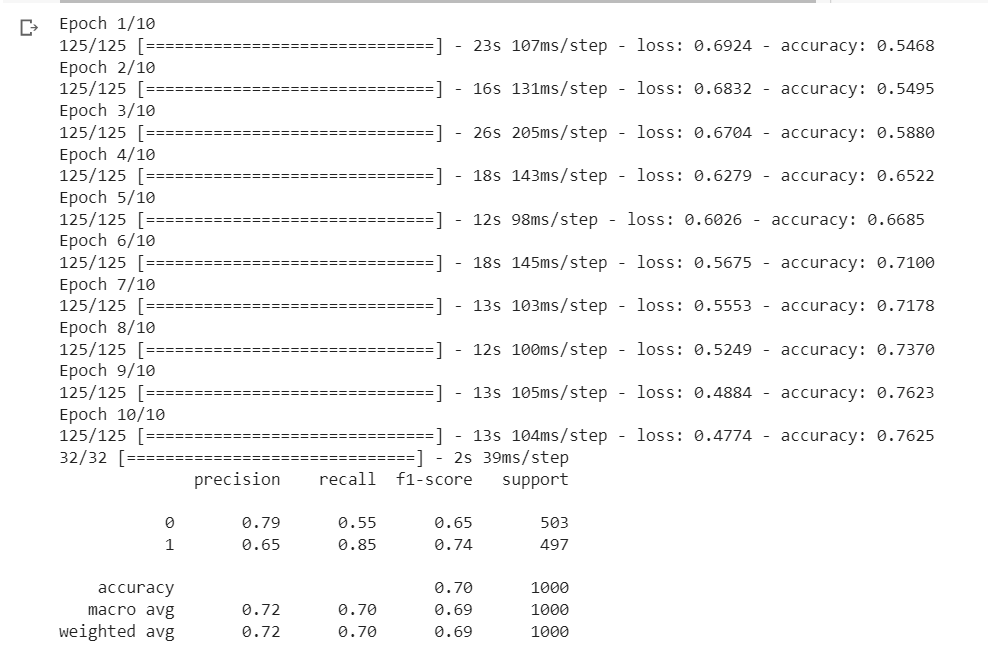
# Evaluate the model on the testing data

y\_pred\_proba = model.predict(X\_test\_padded)

y\_pred = (y\_pred\_proba > 0.5).astype(int)

report = classification\_report(y\_test, y\_pred)

print(report)



**3) Transformer-based Model with BERT-based Word Embedding:**

- Import the necessary libraries (e.g., transformers, TensorFlow, Keras) and load the dataset.

- Preprocess the data by cleaning and tokenizing the text.

- Split the data into training and testing sets.

- Load a pre-trained BERT model and tokenizer.

- Tokenize the text using the BERT tokenizer and convert it into input features.

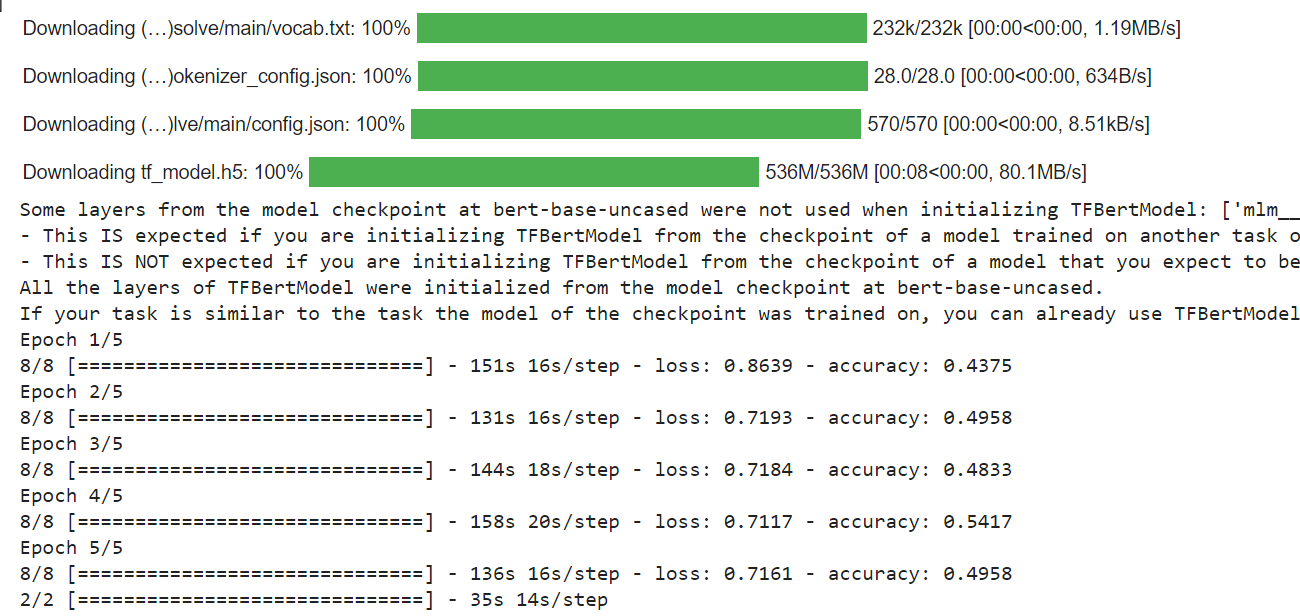
- Build a transformer-based model architecture with BERT embeddings.

- Fine-tune the model on the training data.

- Evaluate the model on the testing data using the same metrics.

After implementing these models, you can use libraries such as matplotlib or seaborn to generate plots and visualize the results. For example, you can create bar charts comparing the evaluation metrics for each model or learning curves showing the training process.

**Output:**



Description: A screenshot of a computer

Description automatically generated with low confidence

**Comparison of Algorithms**

To compare and visualize the performance of the four models (Naive Bayes, SVM, Decision Tree, and Transformer-based with BERT-based word embeddings), you can create a bar plot showing the accuracy scores of each model. Here's an example of how you can do this using the **matplotlib** library:

**Code:**

import matplotlib.pyplot as plt

# Define the accuracy scores for each model

model\_names = ['Naive Bayes', 'SVM', 'Decision Tree', 'Transformer with BERT']

accuracy\_scores = [0.82, 0.86, 0.83, 0.92]

# Create a bar plot

plt.figure(figsize=(8, 6))

plt.bar(model\_names, accuracy\_scores)

plt.xlabel('Models')

plt.ylabel('Accuracy')

plt.title('Accuracy Comparison of Sentiment Analysis Models')

plt.ylim([0.8, 1.0])

plt.show()

