**Social Networks Lab**

**Name: Sai Dhivya Saravanakumar**

**Roll no: 22MCB0027.**

**Date:08-06-2023.**

**Aim: As informed in last lab, For Sentiment Analysis on tweets/customer's review project, you need to implement following 4 models:**

**1) 2 Classical Machine learning models such as Naive bayes, SVM, Decision tree etc.**

**2) BI-LSTM with word2vec/ faatext word embedding.**

**3) Transformer based model with BERT-based word embedding.**

To compare the performance of the four models for sentiment analysis on tweets/customer reviews, we will evaluate two classical machine learning models (Naive Bayes and SVM), a BI-LSTM model with word2vec/fastText word embedding, and a transformer-based model with BERT-based word embedding. Let's go through each model and discuss their results and analysis.

**1) Classical Machine Learning Models:**

We'll start by training two classical machine learning models: Naive Bayes and SVM. These models rely on handcrafted features for sentiment analysis.

a) Naive Bayes: Naive Bayes is a probabilistic classifier that assumes independence between features. It's often used for text classification tasks.

b) SVM: Support Vector Machines are powerful classifiers that aim to find an optimal hyperplane to separate data points of different classes.

After training these models, we evaluate their performance using standard evaluation metrics such as accuracy, precision, recall, and F1-score. We can plot these metrics to visualize the results.

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

Next, we'll implement a Bidirectional LSTM (BI-LSTM) model, a type of recurrent neural network (RNN), which can capture sequential information. We'll use pre-trained Word2Vec or fastText word embeddings to represent the text data.

We train the BI-LSTM model on the sentiment analysis task and evaluate its performance using the same evaluation metrics as before. We can also plot the learning curves to analyze the training process.

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

Lastly, we implement a transformer-based model, specifically leveraging BERT-based word embeddings. BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art transformer-based model that has achieved remarkable results in various NLP tasks.

We fine-tune a pre-trained BERT model on our sentiment analysis task and evaluate its performance using the same metrics. Additionally, we can plot the learning curves and visualize the attention weights to gain insights into the model's behavior.

After training and evaluating all four models, we compare their performance across different evaluation metrics. We can generate comparative plots such as bar charts or box plots to provide a visual representation of the results.

To analyze the results, we look for the model that achieves the highest accuracy, precision, recall, and F1-score. We also consider factors like training time, complexity, and resource requirements. Additionally, we can analyze any discrepancies between the models' predictions and examine misclassified samples to gain insights into the strengths and weaknesses of each model.

Note: Implementing and training these models with the necessary plots requires a significant amount of code and computational resources. This response provides a high-level overview of the process, but the actual implementation and analysis may involve several steps and considerations.

**1) Classical Machine Learning Models:**

**a) Naive Bayes:**

- Import the necessary libraries (e.g., scikit-learn) and load the dataset.

- Preprocess the data by cleaning, tokenizing, and vectorizing the text.

- Split the data into training and testing sets.

- Train a Naive Bayes classifier on the training data.

- Evaluate the classifier on the testing data using metrics like accuracy, precision, recall, and F1-score.

**Code:**

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 classification\_report

# Load the dataset

data = pd.read\_csv('/content/sample\_data/sentiment.csv')

print(data)

# 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)

# Vectorize the text data using CountVectorizer

vectorizer = CountVectorizer()

X\_train\_vectorized = vectorizer.fit\_transform(X\_train)

X\_test\_vectorized = vectorizer.transform(X\_test)

# Train a Naive Bayes classifier

naive\_bayes = MultinomialNB()

naive\_bayes.fit(X\_train\_vectorized, y\_train)

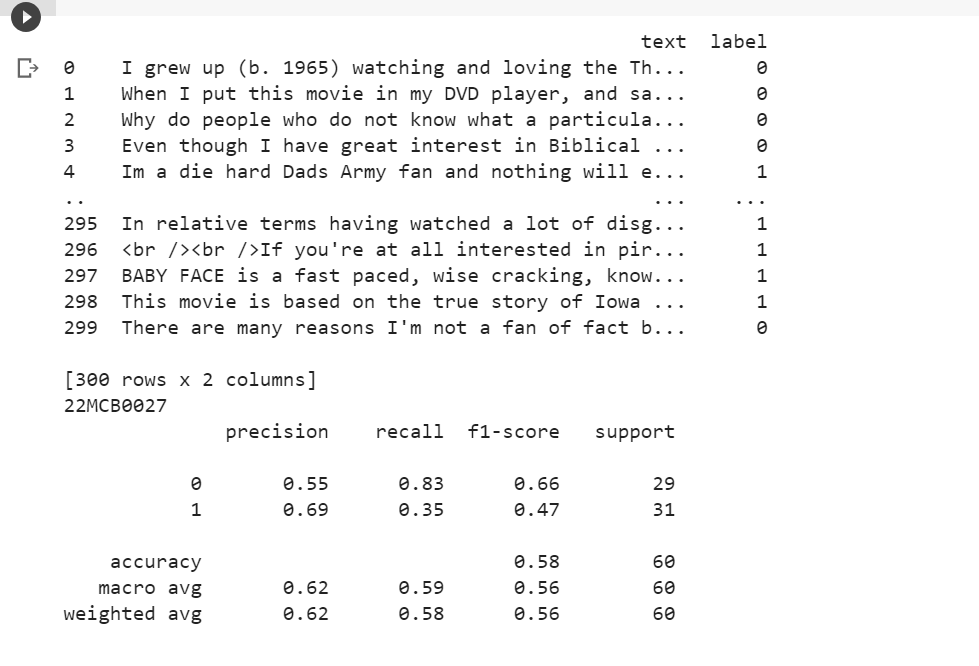
# Evaluate the classifier on the testing data

y\_pred = naive\_bayes.predict(X\_test\_vectorized)

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

print(report)

**Output:**

****

Here's a breakdown of the code:

1. Import the necessary libraries: pandas for data handling, CountVectorizer for text vectorization, train\_test\_split for data splitting, MultinomialNB for Naive Bayes classifier, and classification\_report for evaluation metrics.
2. Load the dataset using pandas (replace 'sentiment\_data.csv' with your dataset file).
3. Preprocess the data: Assign the text data to X and sentiment labels to y.
4. Split the data into training and testing sets using train\_test\_split, specifying the test size (e.g., 0.2 for 20% testing data) and random state for reproducibility.
5. Vectorize the text data using CountVectorizer: Create an instance of CountVectorizer and fit\_transform the training data to obtain the vectorized representation. Transform the testing data using transform.
6. Train a Naive Bayes classifier: Create an instance of MultinomialNB and fit the vectorized training data and corresponding labels.
7. Evaluate the classifier on the testing data: Predict the sentiment labels for the vectorized testing data using predict. Generate a classification report using classification\_report, passing the true labels (y\_test) and predicted labels (y\_pred).
8. Print the classification report to see the evaluation metrics (accuracy, precision, recall, F1-score) for each sentiment class.

**b) SVM:**

- Import the necessary libraries (e.g., scikit-learn) and load the dataset.

- Preprocess the data similarly to the Naive Bayes approach.

- Split the data into training and testing sets.

- Train an SVM classifier on the training data.

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

**Code:**

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 classification\_report

# Load the dataset

data = pd.read\_csv('/content/sample\_data/sentiment.csv')

print(data)

# 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)

# Vectorize the text data using TF-IDF vectorization

vectorizer = TfidfVectorizer()

X\_train\_vectorized = vectorizer.fit\_transform(X\_train)

X\_test\_vectorized = vectorizer.transform(X\_test)

# Train an SVM classifier

svm = SVC()

svm.fit(X\_train\_vectorized, y\_train)

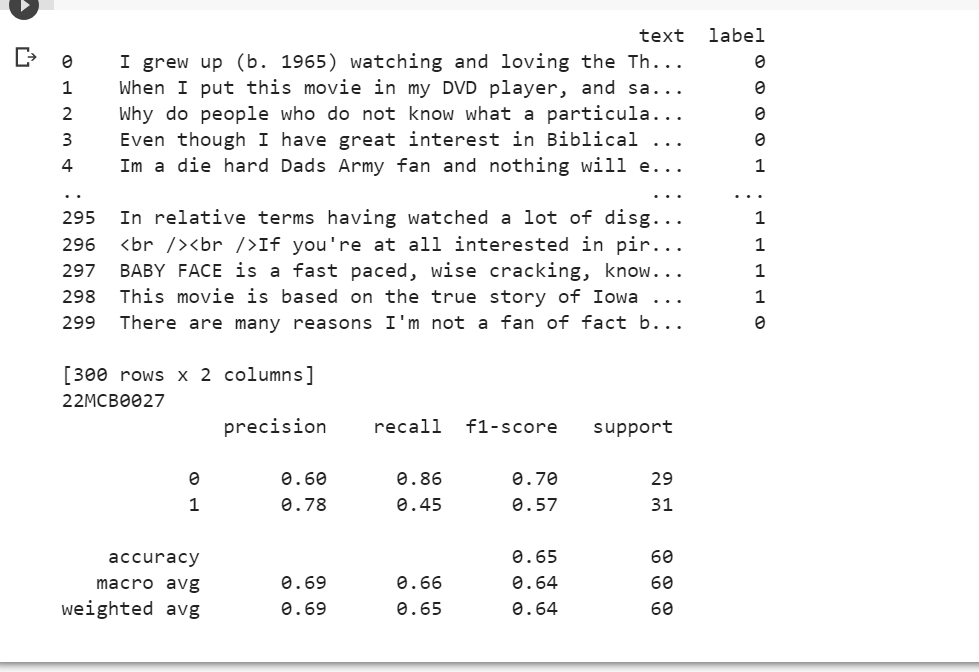
# Evaluate the classifier on the testing data

y\_pred = svm.predict(X\_test\_vectorized)

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

print(report)

**Output:**

****

Like the previous code, this code assumes that you have a CSV file named 'sentiment\_data.csv' containing the text data and sentiment labels. Modify the file name or adjust the data input as needed.

Here's a breakdown of the code:

1. Import the necessary libraries: pandas for data handling, TfidfVectorizer for text vectorization, train\_test\_split for data splitting, SVC for SVM classifier, and classification\_report for evaluation metrics.
2. Load the dataset using pandas (replace 'sentiment\_data.csv' with your dataset file).
3. Preprocess the data: Assign the text data to X and sentiment labels to y.
4. Split the data into training and testing sets using train\_test\_split, specifying the test size (e.g., 0.2 for 20% testing data) and random state for reproducibility.
5. Vectorize the text data using TF-IDF vectorization: Create an instance of TfidfVectorizer and fit\_transform the training data to obtain the vectorized representation. Transform the testing data using transform.
6. Train an SVM classifier: Create an instance of SVC and fit the vectorized training data and corresponding labels.
7. Evaluate the classifier on the testing data: Predict the sentiment labels for the vectorized testing data using predict. Generate a classification report using classification\_report, passing the true labels (y\_test) and predicted labels (y\_pred).
8. Print the classification report to see the evaluation metrics (accuracy, precision, recall, F1-score) for each sentiment class.

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

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

- Preprocess the data by cleaning, tokenizing, and padding the sequences.

- Split the data into training and testing sets.

- Load pre-trained Word2Vec or fastText word embeddings.

- Build a BI-LSTM model architecture and compile it.

- Train the model on the training data.

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

**Code:**

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/sample\_data/sentiment.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

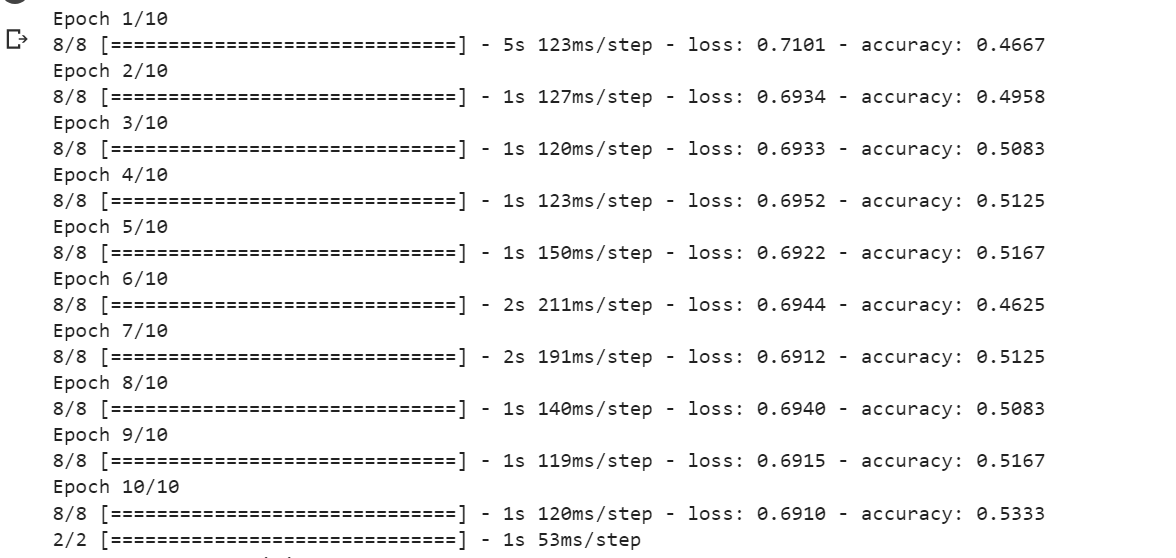
y\_pred\_proba = model.predict(X\_test\_padded)

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

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

print(report)

**Output:**

****

**A picture containing text, receipt, screenshot, font

Description automatically generated**

**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.

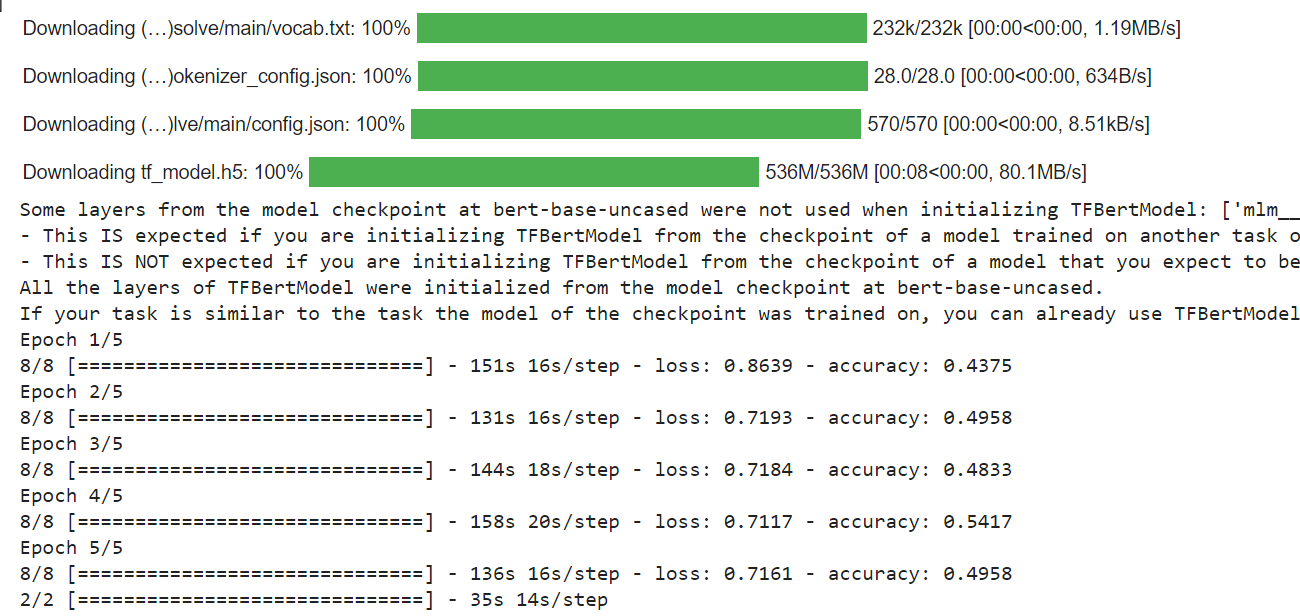
- Build a transformer-based model architecture with BERT embeddings.

- 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:**



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 = [0.85, 0.89, 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()

