**1 Import necessary libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import TfidfVectorizer

**2 Get the dataset**

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

df.head()

**3Clearly there are no missing values in the dataset. But in case if missing values occur we can use certain methods like:**

**- deleting the rows with missing values**

**- deleting the column with too many missing values**

**- filling the missing values with mean, median or mode of the column**

**- using machine learning algorithms to predict the missing values for ex using regression to predict the missing values of a column.**

**- using clustering algorithms to group the data points with missing values and then using the mean of the cluster to fill the missing values.**

**- flag the missing values as a separate category.**

**Now let's drop irrelevant columns from the dataset. The 'Unnamed: 0' here is the extra column which is not required for our analysis. So we will drop it. Also the 'id' column is not required for our analysis as it is just a unique identifier for each row. So we will drop it too.**

df.drop(columns=['Unnamed: 0', 'post\_id', 'user\_id'], inplace=True)

df.head()

**4 Convert the time to post creation to datetime format and use seperate columns for year, month and day.**

df.post\_created=df.post\_created.apply(pd.to\_datetime)

df["month"]=df.post\_created.dt.month

df["year"]=df.post\_created.dt.year

df["day"]=df.post\_created.dt.day

df.drop("post\_created", axis=1)

df.head()

**5 Find the pearson correlation coefficient between the variables.**

**- For r = 1, there is a perfect positive correlation between the variables.**

**- For 0 < r < 1 there is a positive correlation between the variables.**

**- For r = 0, there is no correlation between the variables.**

**- For -1 < r < 0 there is a negative correlation between the variables.**

**- For r = -1, there is a perfect negative correlation between the variables.**

# Select only the desired columns for correlation

selected\_columns = ['followers', 'friends', 'favourites', 'statuses', 'retweets', 'label', 'month', 'year', 'day']

corr\_matrix = df[selected\_columns].corr()

# Display the correlation matrix

print(corr\_matrix)

**6 Heat map**

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')

plt.show()

**7 .**

def correlation(dataset, threshold):

col\_corr = set() # Set of all the names of correlated columns

corr\_matrix = dataset.corr()

for i in range(len(corr\_matrix.columns)):

for j in range(i):

if abs(corr\_matrix.iloc[i,j]) > threshold: # we are interested in absolute coeff value

colname = corr\_matrix.columns[i] # getting the name of column

col\_corr.add(colname)

return col\_corr

**8**.

import pandas as pd

import numpy as np

# Select the specific columns for correlation

selected\_columns = ['followers', 'friends', 'favourites', 'statuses', 'retweets', 'label', 'month', 'year', 'day']

df\_selected = df[selected\_columns]

# Calculate the correlation matrix

corr\_matrix = df\_selected.corr().abs()

# Set the correlation threshold

threshold = 0.85

# Select the upper triangle of the correlation matrix to avoid redundancy

upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

# Find columns with correlation above the threshold

corr\_features = [column for column in upper.columns if any(upper[column] > threshold)]

# Display correlated features and correlation values

print("Highly correlated features (above threshold of 0.85):")

for column in corr\_features:

highly\_correlated = upper[column][upper[column] > threshold]

print(f"\nFeature '{column}' is highly correlated with:")

print(highly\_correlated)

**9** df = df.drop(columns=corr\_features)

df

**Since Friends and Followers are highly correlated, we have dropped the Friends column.**

**10 PREPROCESSING**

#remove numbers

df["post\_text"] = df["post\_text"].str.replace('\d','')

import nltk

from nltk.corpus import stopwords

# Download stopwords

nltk.download('stopwords')

# Define stopwords

sw = stopwords.words("english")

# Remove stopwords from "post\_text" column

df["post\_text"] = df["post\_text"].apply(lambda x: " ".join(word for word in x.split() if word not in sw))

**11** #Stemming

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

def stem\_text(text):

stemmed\_text = " ".join([stemmer.stem(word) for word in text.split()])

return stemmed\_text

df['post\_text'] = df['post\_text'].apply(stem\_text)

**12 !pip install textblob**

**13.**

import nltk

from textblob import TextBlob

# Download punkt

nltk.download("punkt")

# Tokenization

df["tokens"] = df["post\_text"].apply(lambda x: TextBlob(x).words)

**14** #TF-IDF

vectorizer\_tf = TfidfVectorizer(stop\_words="english", max\_features=1000)

X\_tf = vectorizer\_tf.fit\_transform(df["post\_text"])

**15** vectorizer\_tf.vocabulary\_

**16** #idf of each word

all\_feature\_names = vectorizer\_tf.get\_feature\_names\_out()

for word in all\_feature\_names:

#let's get the index in the vocabulary

indx = vectorizer\_tf.vocabulary\_.get(word)

#get the score

idf\_score = vectorizer\_tf.idf\_[indx]

print(f"{word} : {idf\_score}")

#output of tf-idf

X\_tf.toarray()

**17 Cosine Similarity measure**

**Implementing Similarity Measure for Text Processing(SMTP).**

**18.**

#getting the unique features in the document(tweet)and creating another column

def get\_unique\_words(tweet):

words = tweet.split()

unique\_words = list(set(words))

return unique\_words

df1 = df[:1000]

for i in range(1000):

df1['unique\_words'] = df1['post\_text'].apply(lambda x: get\_unique\_words(x))

df1.head()

**19.**

def get\_features(lists):

features = [word for sublist in lists for word in sublist if len(word) > 3]

return features

df2 = df[:500]

df3 = df[19500:]

print(df2.head())

df4 = pd.concat([df2, df3], axis=0)

print(df4.head())

req\_lst = df4['tokens'].tolist()

req\_lst[:5]

#selected\_features = get\_features(req\_lst)

#print(len(selected\_features))

#selected\_features

**20.**

tweet\_column = []

for sublist in req\_lst:

tweet = ' '.join(sublist) # Join the words in each sublist to form a single string

tweet\_column.append(tweet)

print(len(tweet\_column))

tweet\_column[:5]

**21.**

from collections import Counter

# Calculate word count vectors for pairs of two documents

def calculate\_word\_count\_vectors(documents):

word\_count\_vectors = []

for i in range(len(tweet\_column)):

for j in range(0, len(tweet\_column)):

document1 = tweet\_column[i]

document2 = tweet\_column[j]

# Create a set of unique words from both documents

words = set(document1.split()) | set(document2.split())

# Calculate word count vectors for the selected words

d1 = [Counter(document1.split()).get(feature, 0) for feature in words]

d2 = [Counter(document2.split()).get(feature, 0) for feature in words]

# Format the word count vectors

d1\_formatted = "".join(str(count) for count in d1)

d2\_formatted = "".join(str(count) for count in d2)

# Print the word count vectors

#print(f"d{i+1}, d{j+1} =", d1\_formatted, d2\_formatted)

#print()

word\_count\_vectors.append(f"d{i+1} = {d1\_formatted}")

word\_count\_vectors.append(f"d{j+1} = {d2\_formatted}")

return word\_count\_vectors

document\_pairs = calculate\_word\_count\_vectors(tweet\_column)

print(len(document\_pairs))

document\_pairs[:10]

**22 .**

#F1(di,dj) function to calculate

import math

def calculate\_similarity\_score(d1, d2, sigma, lambd):

num = 0

den = 0

for d1j, d2j in zip(d1, d2):

num += calculate\_N\_star(d1j, d2j, sigma, lambd)

den += calculate\_N\_union(d1j, d2j)

if den == 0:

return 0

else:

return num / den

def calculate\_N\_star(d1j, d2j, sigma, lambd):

if d1j == 0 and d2j == 0:

return 0

if d1j > 0 and d2j > 0:

return 0.5 \* (1 + math.exp(-1 \* ((d1j - d2j) / sigma)\*\*2))

return -lambd

def calculate\_N\_union(d1j, d2j):

if d1j == 0 and d2j == 0:

return 0

return 1

**23.**

# Example usage

d1 = [0, 2, 1, 1, 0, 0, 1]

d2 = [3, 1, 1, 1, 1, 0, 0]

sigma = 2

lambd = 1

result = calculate\_similarity\_score(d1, d2, sigma, lambd)

print(result)

**24.**

#SMTP

def calculate\_SMTP(d1, d2, sigma, lambd):

f\_score = calculate\_similarity\_score(d1, d2, sigma, lambd)

smtp\_score = (f\_score + lambd) / (1 + lambd)

return smtp\_score

**25.**

result = calculate\_SMTP(d1, d2, sigma, lambd)

print(result)

**26 .**

len(document\_pairs)

**27.**

lst1 = []

lst2 = []

lst = []

for i in range(0, len(document\_pairs), 2):

dx = document\_pairs[i].split(' = ')[1]

dy = document\_pairs[i + 1].split(' = ')[1]

dxi = [int(x) for x in dx]

dyi = [int(y) for y in dy]

smtp\_score = calculate\_SMTP(dxi, dyi, sigma, lambd)

lst.append(smtp\_score)

#op = (f"SMTP score = {smtp\_score}")

#print(op)

#lst1.clear()

#lst2.clear()

print(len(lst))

lst[:5]

**28.**

lst\_matrix = np.array(lst).reshape(1000, 1000)

lst\_matrix

**29.**

sns.heatmap(lst\_matrix, cmap='hot')

plt.show()

**30 .SVM Model**

from sklearn import svm

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

s = df4['label']

# Convert the series to a matrix

label\_matrix = s.values

# Reshape the matrix if needed

label\_matrix = label\_matrix.reshape((1000,))

unique\_val = np.unique(label\_matrix)

print(unique\_val)

**31.**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(lst\_matrix, label\_matrix, test\_size=0.2, random\_state=42)

clf = svm.SVC(kernel='linear')

clf.fit(X\_train, y\_train)

pred = clf.predict(X\_test)

svm\_acc = clf.score(X\_test, y\_test)

print("Accuracy of the model is:", svm\_acc)

**32 Naive Bayes Classifier**

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score

nb\_model = MultinomialNB()

nb\_model.fit(X\_train, y\_train)

nb\_pred = nb\_model.predict(X\_test)

nb\_acc = accuracy\_score(y\_test, nb\_pred)

print("Accuracy of the model is:", nb\_acc)

**33. Logistic Regression**

from sklearn.linear\_model import LogisticRegression

log\_model = LogisticRegression()

log\_model.fit(X\_train, y\_train)

log\_pred = log\_model.predict(X\_test)

log\_acc = accuracy\_score(y\_test, log\_pred)

print("Accuracy of the model is:", log\_acc)

**34 Random Forest Classifier**

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier()

rf\_model.fit(X\_train, y\_train)

rf\_pred = rf\_model.predict(X\_test)

rf\_acc = accuracy\_score(y\_test, rf\_pred)

print("Accuracy of the model is:", rf\_acc)

**35. !pip install tensorflow**

**36.**

import numpy as np

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.utils import to\_categorical

# Assuming 'label' is the target column with categorical labels for classification

y = df['label'] # Target labels

# Convert target labels to categorical if they are not binary

y = to\_categorical(y)

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_tf.toarray(), y, test\_size=0.2, random\_state=42)

# Define the ANN model

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(X\_train.shape[1],))) # Input layer

model.add(Dropout(0.5)) # Dropout layer for regularization

model.add(Dense(256, activation='relu')) # Hidden layer

model.add(Dropout(0.5))

model.add(Dense(y.shape[1], activation='softmax')) # Output layer

# Compile the model

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

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {test\_accuracy}")

print(f"Test Loss: {test\_loss}")

**37 !pip install optuna**

**38.**

import optuna

from tensorflow.keras.optimizers import Adam

# Define the objective function for Optuna

def objective(trial):

# Define the hyperparameters to be tuned

learning\_rate = trial.suggest\_loguniform("learning\_rate", 1e-5, 1e-2)

neurons\_layer1 = trial.suggest\_int("neurons\_layer1", 50, 512)

neurons\_layer2 = trial.suggest\_int("neurons\_layer2", 50, 512)

dropout1 = trial.suggest\_uniform("dropout1", 0.1, 0.5)

dropout2 = trial.suggest\_uniform("dropout2", 0.1, 0.5)

# Build the model with suggested hyperparameters

model = Sequential([

Dense(neurons\_layer1, activation='relu', input\_shape=(X\_train.shape[1],)),

Dropout(dropout1),

Dense(neurons\_layer2, activation='relu'),

Dropout(dropout2),

Dense(y.shape[1], activation='softmax')

])

# Compile the model

model.compile(optimizer=Adam(learning\_rate=learning\_rate), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model with early stopping

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2, verbose=0)

# Evaluate the model

\_, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

# Return accuracy for Optuna to maximize

return accuracy

# Step 3: Run the Optuna optimization

study = optuna.create\_study(direction="maximize")

study.optimize(objective, n\_trials=20)

# Display the best hyperparameters and best accuracy found

print("Best hyperparameters:", study.best\_params)

print("Best accuracy:", study.best\_value)

**39**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.optimizers import Adam

# Best hyperparameters from Optuna

best\_params = {

'learning\_rate': 0.0008749334351850515,

'neurons\_layer1': 161,

'neurons\_layer2': 437,

'dropout1': 0.40449289524206694,

'dropout2': 0.13053022899986386

}

# Define and compile the final model with optimized hyperparameters

final\_model = Sequential([

Dense(best\_params['neurons\_layer1'], activation='relu', input\_shape=(X\_train.shape[1],)),

Dropout(best\_params['dropout1']),

Dense(best\_params['neurons\_layer2'], activation='relu'),

Dropout(best\_params['dropout2']),

Dense(y.shape[1], activation='softmax')

])

# Compile with optimized learning rate

final\_model.compile(optimizer=Adam(learning\_rate=best\_params['learning\_rate']), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the final model

history = final\_model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the final model on the test set

test\_loss, test\_accuracy = final\_model.evaluate(X\_test, y\_test)

print(f"Final Test Accuracy: {test\_accuracy}")

print(f"Final Test Loss: {test\_loss}")

**40.**

from tensorflow.keras.layers import LeakyReLU

# Example of adding layers and using LeakyReLU

model = Sequential([

Dense(200, input\_shape=(X\_train.shape[1],)),

LeakyReLU(alpha=0.1),

Dropout(0.3),

Dense(150),

LeakyReLU(alpha=0.1),

Dropout(0.3),

Dense(y.shape[1], activation='softmax')

])

**41.**

import pandas as pd

import numpy as np

import matplotlib as plt

import seaborn as sns

import tensorflow as tf

import keras

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Model, Sequential

from keras.layers import GRU, Input, Dense, Activation, RepeatVector, Bidirectional, LSTM, Dropout, Embedding

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

from keras.losses import sparse\_categorical\_crossentropy

from keras.preprocessing.text import Tokenizer

from keras.preprocessing import sequence

from keras.callbacks import EarlyStopping

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

from sklearn.metrics import confusion\_matrix, classification\_report

import collections

from tensorflow.python.client import device\_lib

import matplotlib.pyplot as plt

import seaborn as sns

tf.compat.v1.logging.set\_verbosity(tf.compat.v1.logging.ERROR)

SEED = 10

**42.**

from keras.layers import Embedding

**43.**

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

X = df['post\_text']

y = df['label']

**44.**

plt.figure(figsize = (10,6))

sns.countplot(x = df['label'], palette = 'Set1', alpha = 0.8)

plt.title('Distribution of Target')

**45.**

df['num\_words'] = df['post\_text'].apply(lambda x: len(x.split()))

plt.figure(figsize = (20,6))

sns.histplot(df['num\_words'], bins = range(1, 40, 2), palette = 'Set1', alpha = 0.8)

plt.title('Distribution of the Word Count')

**46.**

#split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify = y, random\_state = SEED)

**47.**

#define Keras Tokenizer

tok = Tokenizer()

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

#return sequences

sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

#print size of the vocabulary

print(f'Train vocabulary size: {len(tok.word\_index)}')

**48.**

#maximum sequence length (512 to prevent memory issues and speed up computation)

MAX\_LEN = 40

#padded sequences

X\_train\_seq = sequence.pad\_sequences(sequences,maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences,maxlen=MAX\_LEN)

**49.**

#define the model

model = tf.keras.Sequential([

Input(name='inputs',shape=[MAX\_LEN]),

Embedding(len(tok.word\_index), 128),

Bidirectional(tf.keras.layers.LSTM(64, return\_sequences=True)),

Bidirectional(tf.keras.layers.LSTM(32)),

Dense(64, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

#compile model

model.compile(loss=tf.keras.losses.BinaryCrossentropy(),

optimizer=tf.keras.optimizers.Adam(1e-4),

metrics=['accuracy'])

#model summary

model.summary()

**50.**

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from tensorflow.keras.layers import Input, Embedding, Bidirectional, LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Check the data structure and information

print(df.info())

# Extract features and target variable

X = df['post\_text']

y = df['label']

# Visualize the distribution of labels

plt.figure(figsize=(10,6))

sns.countplot(x=df['label'], palette='Set1', alpha=0.8)

plt.title('Distribution of Target')

plt.show()

# Calculate and visualize the distribution of word count per post

df['num\_words'] = df['post\_text'].apply(lambda x: len(x.split()))

plt.figure(figsize=(20,6))

sns.histplot(df['num\_words'], bins=range(1, 40, 2), palette='Set1', alpha=0.8)

plt.title('Distribution of the Word Count')

plt.show()

# Split the dataset into training and test sets

SEED = 42 # Set a seed for reproducibility

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=SEED)

# Define and fit the Keras Tokenizer

tok = Tokenizer()

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

# Convert texts to sequences

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

# Print the size of the vocabulary

vocab\_size = len(tok.word\_index) + 1 # Adding 1 to include zero padding in the vocabulary

print(f'Train vocabulary size: {vocab\_size}')

# Define the maximum sequence length based on distribution analysis

MAX\_LEN = 40 # Or adjust based on the word distribution plot

# Pad sequences to ensure uniform input size

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define the model architecture

model = tf.keras.Sequential([

Input(name='inputs', shape=[MAX\_LEN]),

Embedding(input\_dim=vocab\_size, output\_dim=128),

Bidirectional(LSTM(64, return\_sequences=True)),

Bidirectional(LSTM(32)),

Dense(64, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# Print the model summary

model.summary()

# Define early stopping

early\_stopping = EarlyStopping(monitor='val\_accuracy', mode='max', patience=3,

verbose=1, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=10,

validation\_split=0.2,

batch\_size=64,

callbacks=[early\_stopping])

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

**51.**

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from tensorflow.keras.layers import Input, Embedding, Bidirectional, LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Check the data structure and information

print(df.info())

# Extract features and target variable

X = df['post\_text']

y = df['label']

# Visualize the distribution of labels

plt.figure(figsize=(10,6))

sns.countplot(x=df['label'], palette='Set1', alpha=0.8)

plt.title('Distribution of Target')

plt.show()

# Calculate and visualize the distribution of word count per post

df['num\_words'] = df['post\_text'].apply(lambda x: len(x.split()))

plt.figure(figsize=(20,6))

sns.histplot(df['num\_words'], bins=range(1, 40, 2), palette='Set1', alpha=0.8)

plt.title('Distribution of the Word Count')

plt.show()

# Split the dataset into training and test sets

SEED = 42 # Set a seed for reproducibility

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=SEED)

# Define and fit the Keras Tokenizer

tok = Tokenizer()

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

# Convert texts to sequences

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

# Print the size of the vocabulary

vocab\_size = len(tok.word\_index) + 1 # Adding 1 to include zero padding in the vocabulary

print(f'Train vocabulary size: {vocab\_size}')

# Define the maximum sequence length based on distribution analysis

MAX\_LEN = 40 # Or adjust based on the word distribution plot

# Pad sequences to ensure uniform input size

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define the model architecture

model = tf.keras.Sequential([

Input(name='inputs', shape=[MAX\_LEN]),

Embedding(input\_dim=vocab\_size, output\_dim=128),

Bidirectional(LSTM(64, return\_sequences=True)),

Bidirectional(LSTM(32)),

Dense(64, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# Print the model summary

model.summary()

# Define early stopping

early\_stopping = EarlyStopping(monitor='val\_accuracy', mode='max', patience=3,

verbose=1, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=10,

validation\_split=0.2,

batch\_size=64,

callbacks=[early\_stopping])

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

**52.**

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Predictions and classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32") # Convert probabilities to binary predictions

# Print classification report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(cm).plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Plot training and validation accuracy and loss

history\_dict = history.history

# Accuracy plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['accuracy'], label='Training Accuracy')

plt.plot(history\_dict['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

# Loss plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['loss'], label='Training Loss')

plt.plot(history\_dict['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

**53.**

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

from tensorflow.keras.layers import Input, Embedding, Bidirectional, LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Check the data structure and information

print(df.info())

# Extract features and target variable

X = df['post\_text']

y = df['label']

# Visualize the distribution of labels

plt.figure(figsize=(10,6))

sns.countplot(x=df['label'], palette='Set1', alpha=0.8)

plt.title('Distribution of Target')

plt.show()

# Calculate and visualize the distribution of word count per post

df['num\_words'] = df['post\_text'].apply(lambda x: len(x.split()))

plt.figure(figsize=(20,6))

sns.histplot(df['num\_words'], bins=range(1, 40, 2), palette='Set1', alpha=0.8)

plt.title('Distribution of the Word Count')

plt.show()

# Split the dataset into training and test sets

SEED = 42 # Set a seed for reproducibility

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=SEED)

# Define and fit the Keras Tokenizer

tok = Tokenizer()

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

# Convert texts to sequences

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

# Print the size of the vocabulary

vocab\_size = len(tok.word\_index) + 1 # Adding 1 to include zero padding in the vocabulary

print(f'Train vocabulary size: {vocab\_size}')

# Define the maximum sequence length based on distribution analysis

MAX\_LEN = 40 # Or adjust based on the word distribution plot

# Pad sequences to ensure uniform input size

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define the model architecture

model = tf.keras.Sequential([

Input(name='inputs', shape=[MAX\_LEN]),

Embedding(input\_dim=vocab\_size, output\_dim=128),

Bidirectional(LSTM(64, return\_sequences=True)),

Bidirectional(LSTM(32)),

Dense(64, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# Print the model summary

model.summary()

# Define early stopping

early\_stopping = EarlyStopping(monitor='val\_accuracy', mode='max', patience=3,

verbose=1, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=10,

validation\_split=0.2,

batch\_size=64,

callbacks=[early\_stopping])

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Predictions and classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32") # Convert probabilities to binary predictions

# Print classification report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(cm).plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Plot training and validation accuracy and loss

history\_dict = history.history

# Accuracy plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['accuracy'], label='Training Accuracy')

plt.plot(history\_dict['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

# Loss plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['loss'], label='Training Loss')

plt.plot(history\_dict['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

**54.**

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

from tensorflow.keras.layers import Input, Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dropout, Flatten

from tensorflow.keras.callbacks import EarlyStopping

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Extract features and target variable

X = df['post\_text']

y = df['label']

# Split the dataset into training and test sets

SEED = 42 # Set a seed for reproducibility

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=SEED)

# Define and fit the Keras Tokenizer

tok = Tokenizer()

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

# Convert texts to sequences

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

# Set vocabulary size and padding length

vocab\_size = len(tok.word\_index) + 1 # Adding 1 to include zero padding in the vocabulary

MAX\_LEN = 40 # Adjust based on your dataset

# Pad sequences to ensure uniform input size

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define the hybrid model architecture

model = tf.keras.Sequential([

Input(name='inputs', shape=[MAX\_LEN]),

Embedding(input\_dim=vocab\_size, output\_dim=128),

# Add convolutional layers for feature extraction

Conv1D(filters=64, kernel\_size=3, activation='relu'),

MaxPooling1D(pool\_size=2),

# Add LSTM layers for sequence processing

Bidirectional(LSTM(64, return\_sequences=True)),

Bidirectional(LSTM(32)),

# Fully connected layers

Dense(128, activation='relu'),

Dropout(0.5),

Dense(64, activation='relu'),

Dropout(0.3),

Dense(1, activation='sigmoid') # Binary classification

])

# Compile the model

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# Print the model summary

model.summary()

# Define early stopping

early\_stopping = EarlyStopping(monitor='val\_accuracy', mode='max', patience=3,

verbose=1, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=15,

validation\_split=0.2,

batch\_size=64,

callbacks=[early\_stopping])

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Predictions and classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32") # Convert probabilities to binary predictions

# Print classification report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(cm).plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Plot training and validation accuracy and loss

history\_dict = history.history

# Accuracy plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['accuracy'], label='Training Accuracy')

plt.plot(history\_dict['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

# Loss plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['loss'], label='Training Loss')

plt.plot(history\_dict['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

**55.**

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

from tensorflow.keras.layers import Input, Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dropout, Flatten

from tensorflow.keras.callbacks import EarlyStopping

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Extract features and target variable

X = df['post\_text']

y = df['label']

# Split the dataset into training and test sets

SEED = 42 # Set a seed for reproducibility

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=SEED)

# Define and fit the Keras Tokenizer

tok = Tokenizer()

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

# Convert texts to sequences

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

# Set vocabulary size and padding length

vocab\_size = len(tok.word\_index) + 1 # Adding 1 to include zero padding in the vocabulary

MAX\_LEN = 40 # Adjust based on your dataset

# Pad sequences to ensure uniform input size

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define the hybrid model architecture

model = tf.keras.Sequential([

Input(name='inputs', shape=[MAX\_LEN]),

Embedding(input\_dim=vocab\_size, output\_dim=128),

# Add convolutional layers for feature extraction

Conv1D(filters=64, kernel\_size=3, activation='relu'),

MaxPooling1D(pool\_size=2),

# Add LSTM layers for sequence processing

Bidirectional(LSTM(64, return\_sequences=True)),

Bidirectional(LSTM(32)),

# Fully connected layers

Dense(128, activation='relu'),

Dropout(0.5),

Dense(64, activation='relu'),

Dropout(0.3),

Dense(1, activation='sigmoid') # Binary classification

])

# Compile the model

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# Print the model summary

model.summary()

# Define early stopping

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=35,

validation\_split=0.2,

batch\_size=64)

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Predictions and classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32") # Convert probabilities to binary predictions

# Print classification report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(cm).plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Plot training and validation accuracy and loss

history\_dict = history.history

# Accuracy plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['accuracy'], label='Training Accuracy')

plt.plot(history\_dict['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

# Loss plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['loss'], label='Training Loss')

plt.plot(history\_dict['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

**56.**

import tensorflow as tf

from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Bidirectional, LSTM, Dense, Dropout, BatchNormalization, Multiply, Activation

from tensorflow.keras.models import Model

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.optimizers import Adam, AdamW, RMSprop, Nadam

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Prepare the data

X = df['post\_text']

y = df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=42)

# Tokenize and pad sequences

tok = Tokenizer()

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

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

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

MAX\_LEN = 40

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define attention mechanism

def attention\_block(inputs):

attention = Dense(1, activation='tanh')(inputs)

attention = tf.keras.layers.Flatten()(attention)

attention = Activation('softmax')(attention)

attention = tf.keras.layers.RepeatVector(inputs.shape[-1])(attention)

attention = tf.keras.layers.Permute([2, 1])(attention)

output = Multiply()([inputs, attention])

return output

# Define the hybrid model architecture

inputs = Input(shape=(MAX\_LEN,), name="inputs")

x = Embedding(input\_dim=vocab\_size, output\_dim=128)(inputs)

# Convolutional Block

x = Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same')(x)

x = BatchNormalization()(x)

x = Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same')(x)

x = GlobalMaxPooling1D()(x)

# LSTM Block with Attention

x = tf.keras.layers.Reshape((1, -1))(x) # Reshape for attention compatibility

x = Bidirectional(LSTM(64, return\_sequences=True))(x)

x = attention\_block(x)

x = tf.keras.layers.Flatten()(x)

# Fully connected layers with dropout

x = Dense(128, activation='relu')(x)

x = Dropout(0.5)(x)

x = Dense(64, activation='relu')(x)

x = Dropout(0.3)(x)

outputs = Dense(1, activation='sigmoid')(x)

# Model creation

model = Model(inputs, outputs)

# Choose an optimizer

# optimizer = Adam(learning\_rate=1e-4)

optimizer = AdamW(learning\_rate=1e-4) # Uncomment for AdamW

# optimizer = RMSprop(learning\_rate=1e-4) # Uncomment for RMSprop

# optimizer = Nadam(learning\_rate=1e-4) # Uncomment for Nadam

# Compile the model

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

# Model summary

model.summary()

# Define early stopping

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=30,

validation\_split=0.2,

batch\_size=64,

callbacks=[early\_stopping])

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Predictions and classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(cm).plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Plot training and validation accuracy and loss

history\_dict = history.history

# Accuracy plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['accuracy'], label='Training Accuracy')

plt.plot(history\_dict['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

# Loss plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['loss'], label='Training Loss')

plt.plot(history\_dict['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

**57.**

import tensorflow as tf

from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Bidirectional, LSTM, Dense, Dropout, BatchNormalization, Multiply, Activation

from tensorflow.keras.regularizers import l2

from tensorflow.keras.models import Model

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.optimizers import AdamW

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Prepare the data

X = df['post\_text']

y = df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=42)

# Tokenize and pad sequences

tok = Tokenizer()

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

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

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

MAX\_LEN = 40

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define attention mechanism

def attention\_block(inputs):

attention = Dense(1, activation='tanh')(inputs)

attention = tf.keras.layers.Flatten()(attention)

attention = Activation('softmax')(attention)

attention = tf.keras.layers.RepeatVector(inputs.shape[-1])(attention)

attention = tf.keras.layers.Permute([2, 1])(attention)

output = Multiply()([inputs, attention])

return output

# Define the hybrid model architecture with L2 regularization and higher dropout

inputs = Input(shape=(MAX\_LEN,), name="inputs")

x = Embedding(input\_dim=vocab\_size, output\_dim=128)(inputs)

# Convolutional Block

x = Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same', kernel\_regularizer=l2(0.01))(x)

x = BatchNormalization()(x)

x = Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same', kernel\_regularizer=l2(0.01))(x)

x = GlobalMaxPooling1D()(x)

# LSTM Block with Attention

x = tf.keras.layers.Reshape((1, -1))(x) # Reshape for attention compatibility

x = Bidirectional(LSTM(64, return\_sequences=True, kernel\_regularizer=l2(0.01)))(x)

x = attention\_block(x)

x = tf.keras.layers.Flatten()(x)

# Fully connected layers with increased dropout

x = Dense(128, activation='relu', kernel\_regularizer=l2(0.01))(x)

x = Dropout(0.6)(x) # Increased dropout

x = Dense(64, activation='relu', kernel\_regularizer=l2(0.01))(x)

x = Dropout(0.6)(x) # Increased dropout

outputs = Dense(1, activation='sigmoid')(x)

# Model creation and compilation

model = Model(inputs, outputs)

optimizer = AdamW(learning\_rate=1e-5) # Reduced learning rate

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

# Model summary

model.summary()

# Define early stopping with increased patience

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=50, # Increased epochs to give more time for improvement

validation\_split=0.2,

batch\_size=64,

callbacks=[early\_stopping])

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Predictions and classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(cm).plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Plot training and validation accuracy and loss

history\_dict = history.history

# Accuracy plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['accuracy'], label='Training Accuracy')

plt.plot(history\_dict['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

# Loss plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['loss'], label='Training Loss')

plt.plot(history\_dict['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

**58.**

import tensorflow as tf

from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Bidirectional, LSTM, Dense, Dropout, BatchNormalization, Multiply, Activation

from tensorflow.keras.regularizers import l2

from tensorflow.keras.models import Model

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.optimizers import AdamW

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Prepare the data

X = df['post\_text']

y = df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=42)

# Tokenize and pad sequences

tok = Tokenizer()

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

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

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

MAX\_LEN = 40

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define attention mechanism

def attention\_block(inputs):

attention = Dense(1, activation='tanh')(inputs)

attention = tf.keras.layers.Flatten()(attention)

attention = Activation('softmax')(attention)

attention = tf.keras.layers.RepeatVector(inputs.shape[-1])(attention)

attention = tf.keras.layers.Permute([2, 1])(attention)

output = Multiply()([inputs, attention])

return output

# Define the hybrid model architecture with L2 regularization and higher dropout

inputs = Input(shape=(MAX\_LEN,), name="inputs")

x = Embedding(input\_dim=vocab\_size, output\_dim=128)(inputs)

# Convolutional Block

x = Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same', kernel\_regularizer=l2(0.01))(x)

x = BatchNormalization()(x)

x = Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same', kernel\_regularizer=l2(0.01))(x)

x = GlobalMaxPooling1D()(x)

# LSTM Block with Attention

x = tf.keras.layers.Reshape((1, -1))(x) # Reshape for attention compatibility

x = Bidirectional(LSTM(64, return\_sequences=True, kernel\_regularizer=l2(0.01)))(x)

x = attention\_block(x)

x = tf.keras.layers.Flatten()(x)

# Fully connected layers with increased dropout

x = Dense(128, activation='relu', kernel\_regularizer=l2(0.01))(x)

x = Dropout(0.6)(x) # Increased dropout

x = Dense(64, activation='relu', kernel\_regularizer=l2(0.01))(x)

x = Dropout(0.6)(x) # Increased dropout

outputs = Dense(1, activation='sigmoid')(x)

# Model creation and compilation

model = Model(inputs, outputs)

optimizer = AdamW(learning\_rate=1e-5) # Reduced learning rate

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

# Model summary

model.summary()

# Define early stopping with increased patience

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=100, # Increased epochs to give more time for improvement

validation\_split=0.2,

batch\_size=64,

callbacks=[early\_stopping])

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Predictions and classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

ConfusionMatrixDisplay(cm).plot(cmap="Blues")

plt.title('Confusion Matrix')

plt.show()

# Plot training and validation accuracy and loss

history\_dict = history.history

# Accuracy plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['accuracy'], label='Training Accuracy')

plt.plot(history\_dict['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.show()

# Loss plot

plt.figure(figsize=(12, 5))

plt.plot(history\_dict['loss'], label='Training Loss')

plt.plot(history\_dict['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.show()

**59.**

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Get model predictions as probabilities

y\_prob = model.predict(X\_test\_seq).ravel() # Flatten the probabilities for ROC curve

# Calculate the ROC curve and AUC score

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

# Plot ROC curve

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

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--') # Diagonal line for random chance

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

**60.**

import tensorflow as tf

from tensorflow.keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Bidirectional, LSTM, Dense, Dropout, LayerNormalization, MultiHeadAttention, Add, Flatten, concatenate

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import AdamW

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

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

import numpy as np

import pandas as pd

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Prepare the data

X = df['post\_text']

y = df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=42)

# Tokenize and pad sequences

tok = Tokenizer()

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

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

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

MAX\_LEN = 40

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define the Transformer encoder block

def transformer\_encoder(inputs, head\_size, num\_heads, ff\_dim, dropout=0):

x = LayerNormalization(epsilon=1e-6)(inputs)

x = MultiHeadAttention(key\_dim=head\_size, num\_heads=num\_heads, dropout=dropout)(x, x)

x = Add()([x, inputs])

x = LayerNormalization(epsilon=1e-6)(x)

x = Dense(ff\_dim, activation="relu")(x)

x = Dropout(dropout)(x)

x = Dense(inputs.shape[-1])(x)

x = Add()([x, inputs])

return x

# CNN Model

def create\_cnn\_model():

inputs = Input(shape=(MAX\_LEN,))

x = Embedding(input\_dim=vocab\_size, output\_dim=128)(inputs)

x = Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same')(x)

x = Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same')(x)

x = GlobalMaxPooling1D()(x)

x = Dense(64, activation='relu', kernel\_regularizer=tf.keras.regularizers.l2(0.01))(x)

x = Dropout(0.5)(x)

outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs, outputs)

return model

# Bi-LSTM Model

def create\_lstm\_model():

inputs = Input(shape=(MAX\_LEN,))

x = Embedding(input\_dim=vocab\_size, output\_dim=128)(inputs)

x = Bidirectional(LSTM(64, return\_sequences=True))(x)

x = GlobalMaxPooling1D()(x)

x = Dense(64, activation='relu', kernel\_regularizer=tf.keras.regularizers.l2(0.01))(x)

x = Dropout(0.5)(x)

outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs, outputs)

return model

# Transformer Model

def create\_transformer\_model():

inputs = Input(shape=(MAX\_LEN,))

x = Embedding(input\_dim=vocab\_size, output\_dim=128)(inputs)

x = transformer\_encoder(x, head\_size=64, num\_heads=4, ff\_dim=128, dropout=0.2)

x = GlobalMaxPooling1D()(x)

x = Dense(64, activation='relu', kernel\_regularizer=tf.keras.regularizers.l2(0.01))(x)

x = Dropout(0.5)(x)

outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs, outputs)

return model

# Instantiate models

cnn\_model = create\_cnn\_model()

lstm\_model = create\_lstm\_model()

transformer\_model = create\_transformer\_model()

# Compile models

optimizer = AdamW(learning\_rate=1e-5)

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

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

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

# Early stopping

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

# Train each model

cnn\_model.fit(X\_train\_seq, y\_train, epochs=20, validation\_split=0.2, batch\_size=64, callbacks=[early\_stopping])

lstm\_model.fit(X\_train\_seq, y\_train, epochs=20, validation\_split=0.2, batch\_size=64, callbacks=[early\_stopping])

transformer\_model.fit(X\_train\_seq, y\_train, epochs=20, validation\_split=0.2, batch\_size=64, callbacks=[early\_stopping])

# Generate predictions and ensemble them by averaging

cnn\_pred = cnn\_model.predict(X\_test\_seq)

lstm\_pred = lstm\_model.predict(X\_test\_seq)

transformer\_pred = transformer\_model.predict(X\_test\_seq)

# Averaging the predictions from each model

ensemble\_pred = (cnn\_pred + lstm\_pred + transformer\_pred) / 3

ensemble\_pred = (ensemble\_pred > 0.5).astype("int32")

# Evaluate ensemble performance

print("\nEnsemble Classification Report:")

print(classification\_report(y\_test, ensemble\_pred))

accuracy = accuracy\_score(y\_test, ensemble\_pred)

print(f"Ensemble Test Accuracy: {accuracy \* 100:.2f}%")

**61.**

import tensorflow as tf

from tensorflow.keras.layers import Input, Embedding, SimpleRNN, LSTM, Dense, Dropout, GlobalMaxPooling1D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing import sequence

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

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

import pandas as pd

# Load the dataset

df = pd.read\_csv('/content/Mental-Health-Twitter.csv')

# Prepare the data

X = df['post\_text']

y = df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=42)

# Tokenize and pad sequences

tok = Tokenizer()

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

train\_sequences = tok.texts\_to\_sequences(X\_train)

test\_sequences = tok.texts\_to\_sequences(X\_test)

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

MAX\_LEN = 40

X\_train\_seq = sequence.pad\_sequences(train\_sequences, maxlen=MAX\_LEN)

X\_test\_seq = sequence.pad\_sequences(test\_sequences, maxlen=MAX\_LEN)

# Define the RNN-LSTM hybrid model architecture

inputs = Input(shape=(MAX\_LEN,), name="inputs")

x = Embedding(input\_dim=vocab\_size, output\_dim=128)(inputs)

# RNN layer

x = SimpleRNN(64, return\_sequences=True)(x)

# LSTM layer

x = LSTM(64, return\_sequences=True)(x)

# Global Max Pooling to reduce dimensions and flatten

x = GlobalMaxPooling1D()(x)

# Fully connected layers with dropout for regularization

x = Dense(64, activation='relu')(x)

x = Dropout(0.5)(x)

x = Dense(32, activation='relu')(x)

x = Dropout(0.3)(x)

outputs = Dense(1, activation='sigmoid')(x)

# Model creation and compilation

model = Model(inputs, outputs)

optimizer = Adam(learning\_rate=1e-4)

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

# Model summary

model.summary()

# Define early stopping to prevent overfitting

# Train the model

history = model.fit(X\_train\_seq, y\_train,

epochs=100,

validation\_split=0.2,

batch\_size=64

)

# Evaluate the model on test data

test\_loss, test\_accuracy = model.evaluate(X\_test\_seq, y\_test)

print(f'\nTest Loss: {test\_loss}, Test Accuracy: {test\_accuracy}')

# Generate predictions and display a classification report

y\_pred = (model.predict(X\_test\_seq) > 0.5).astype("int32")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Print accuracy score

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Test Accuracy: {accuracy \* 100:.2f}%")