CODE BOOKLET

TEXT CLASSIFICATION FOR MAJOR DEPRESSIVE DISORDER (MDD) SYMPTOMS AND TREATMENTS USING CONVOLUTIONAL NEURAL NETWORKS (CNN)

ACADEMIC SESSION: 2023/2024

STUDENT: FELICIA CHIN HUI FEN (A20EC0037)

PROGRAM: Bachelor of Computer Science (Bioinformatics)

SUPERVISOR: DR SHARIN HAZLIN BINTI HUPSI
CO-SUPERVISOR: DR AHMAD NAJMI BIN AMERHAIDER NUAR

SUMMARY:

This document holds all the coding for the final year project TEXT CLASSIFICATION FOR MAJOR DEPRESSIVE DISORDER (MDD) SYMPTOMS AND TREATMENTS USING CONVOLUTIONAL NEURAL NETWORKS (CNN), which include all process and functions used to analyze the dataset, building the model, evaluate the model's performance and classifying the MDD symptoms and treatments words from medical journals. The issues of classifying the MDD symptoms and treatments from text data (medical journals) can be solved.

Table of Contents

Project Hardware and Software List

HARDWARE	VERSION	EXPLANATION OF USAGE
CPU	Intel Core i5 or above	Use for Data Processing and multitasking that help to run multiple application efficiently.
processor with a minimum of 1 GHz	-	Ensures that the system can handle basic operations and application
RAM with a minimum of 4GB	-	Provides enough memory to run the development environments, browsers, and other necessary applications without significant slowdowns.
Google Drive	-	Backup storage for project data.

SOFTWARE	VERSION	EXPLANATION OF USAGE
Microsoft Word	2406	Use to write the report.
Microsoft Excel	2406	Save the dataset extracted.
Bio	1.84	Used to access the NCBI database for data collection.
Gensim	4.3.2	Used to build the Word2vec model.
Google Colaboratory Tools	1.0.0	Work as Integrated Development Environment for building the Python code.
Keras	2.15.0	Used to build the CNN model.
Matplotlib	3.7.1	Used in data visualization for the results.
Nltk	3.8.1	Used in text preprocessing.

Numpy	1.25.2	Provides support for large and multi-dimensional arrays that help in data analysis.
Os	-	Used to interact with the operating system.
Pandas	2.0.3	Used for data manipulation and analysis.
re	2.2.1	Used for regular expression.
Seaborn	0.13.1	Used in exploratory data analysis.
Sklearn	1.2.2	Used to split the dataset ratio by importing the train_test_split.
string	-	Common string operations.
Tensorflow	2.15.0	Used as machine learning framework.
Wordcloud	1.9.3	Used to create word cloud.

Module List and Explanation

MODULE	FUNCTION	USERS
PACKAGE INSTALLATION	• Install necessary package for the project. • Import necessary libraries.	• Package Installation. • Import libraries.
DATA COLLECTION	• Extract medical journal from the NCBI website.	• Data Collection.
DATA PREPROCESSING	• Split the abstract into sentences. • Dataset undergoes a series of text preprocessing to produce clean dataset, which include removal of empty row, removal of stopwords, removal of digits, removal of punctuation, convert text to lowercase, tokenization, and lemmatization. • Labelling the dataset based on the MDD keywords defined. • Utilise visualization tools to understand the dataset better. • Handle the imbalanced dataset.	• Data preprocessing. • Data Labelling. • Exploratory Data Analysis. • Handle imbalanced dataset.
MODEL BUILDING	• Building three types of CNN model using three different dataset split ratio. • Utilise visualization tool to view the model results.	• CNN model building. • Model result visualization.

MODEL EVALUATION	• Evaluate the CNN model performance using performance metrics (accuracy, recall, precision, F1-Score, and confusion matrix).	• Model's performance evaluation.

Code for Package Installation

```
# Install necessary packages
!pip install biopython
!pip install nltk
```

```
# Import libraries
import pandas as pd
import numpy as np
import re
import string
import nltk
import gensim
import os
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.util import ngrams

# Dataset Collection
from Bio import Entrez
from google.colab import files

# importing libraries
import tensorflow as tf
from keras import initializers, regularizers, constraints, optimizers,
layers, callbacks
from keras.callbacks import EarlyStopping,ModelCheckpoint
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing import text, sequence
from keras import initializers, regularizers, constraints, optimizers,
layers
from keras.layers import Dense, Input, Embedding, Dropout,
SpatialDropout1D, Activation, Conv1D, GlobalMaxPool1D, BatchNormalization,
Add
from keras.optimizers import Adam
from keras.models import Model, Sequential
from sklearn.model_selection  import train_test_split

# For custom metrics
import keras.backend as K
from keras.utils import plot_model
```

```python
from keras.callbacks import EarlyStopping

# Visualization
from sklearn.metrics import classification_report, confusion_matrix
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import seaborn as sns

# Download required resources for NLTK
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")
```

Code for Data Collection

```
# Data Collection

# Set your email address (required by NCBI)
Entrez.email = "feliciac552@gmail.com"

# Define search terms
search_terms = "major depressive disorder AND symptoms AND treatments"
start_year = "2019"
end_year = "2023"

# Perform the search
handle = Entrez.esearch(db="pubmed", term=search_terms, retmax=5000,
mindate=start_year, maxdate=end_year)
record = Entrez.read(handle)
handle.close()

id_list = record["IdList"]
summary_records = []

# Create a list to store the extracted data
data = []

# Create a set to store the processed PMIDs
processed_pmids = set()

for article_id in id_list:
  if article_id in processed_pmids:
      continue

  try:
      handle = Entrez.efetch(db='pubmed', id=article_id, retmode='xml')
      article = Entrez.read(handle)['PubmedArticle'][0]
      handle.close()

      title = article['MedlineCitation']['Article']['ArticleTitle']
      abstract = article['MedlineCitation']['Article'].get('Abstract',
{}).get('AbstractText', '')

      # Convert abstract to string
      abstract = str(abstract)

      # Append the extracted data to the list
      data.append({'Title': title, 'Abstract':abstract})

  except Exception as e:
        print(f"Error occurred for article ID {article_id}: {str(e)}")
        continue

# Save the data into CSV file
import csv
# Specify the output CSV file path
csv_file = 'major_depressive_disorder_datasets.csv'

# Write the data to the CSV file
with open(csv_file, 'w', newline='') as file:
    fieldnames = ['Title', 'Abstract']
    writer = csv.DictWriter(file, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerows(data)

print("Data saved to:", csv_file)
```

```
from google.colab import files

# Download the file
files.download('major_depressive_disorder_datasets.csv')
```

Code for Data Preprocessing

```python
# Split abstract into sentences
sentences_list = []
for abstract in df["Abstract"]:
    sentences = nltk.sent_tokenize(str(abstract))
    sentences_list.extend(sentences)

# Create a new DataFrame with only the sentences column
sentences_df = pd.DataFrame(sentences_list, columns=["sentences"])

# Save the sentences DataFrame to a new CSV file
sentences_df.to_csv("mdd_sentences_dataset.csv", index=False)
```

```python
# Check for NaN values in the DataFrame
nan_values = sentences_df.isnull().sum()

# Print the number of NaN values for each column
print(nan_values)
```

```python
# Preprocessing function
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))

    # Remove digits
    text = re.sub(r'\d+', '', text)

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stop words
    stop_words = set(stopwords.words("english"))
    tokens = [token for token in tokens if token not in stop_words]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Join the tokens back into a string
    processed_text = " ".join(tokens)

    return processed_text

# Apply preprocessing to the "sentences" column
sentences_df["sentences"] =
sentences_df["sentences"].apply(preprocess_text)

# Save the preprocessed data to a CSV file
sentences_df.to_csv("mdd_preprocessed_data.csv", index=False)

# Chekcing null values in the dataset
preprocessed_df.isnull().sum()

# Drop the empty rows
preprocessed_df.dropna(subset=['sentences'], inplace=True)

# Resetting the index
preprocessed_df.reset_index(drop=True, inplace=True)

# recheck null values in the dataset
preprocessed_df.isnull().sum()
```

```
# Labelling Process
# Define the symptoms and treatments keywords
symptoms_keywords = [
    "Aches",
    "pains",
    "Anxious",
    "Appetite changes",
    "Cramps",
    "decreased concentration",
    "Decreased energy",
    "depressed mood",
    "Difficulty concentrating",
    "Difficulty making decisions",
    "Difficulty Remembering",
    "Difficulty sleeping",
    "digestive problems without a clear physical cause",
    "disrupted sleep",
    "do not ease even with treatment",
    "early-morning awakening",
    "empty mood",
    "excessive guilt",
    "fatigue",
    "Feeling restless",
    "feeling very tired",
    "feelings of excessive guilt",
    "Feelings of guilt",
    "Feelings of hopelessness",
    "having trouble sitting still",
    "headaches",
    "helplessness",
    "hopelessness about the future",
    "hypersomnia",
    "Insomnia",
    "Irritability",
    "Loss of interest",
    "loss of pleasure",
    "low in energy",
    "low self-worth",
    "oversleeping",
    "Persistent sadness",
    "pessimism",
    "poor concentration",
    "psychomotor agitation",
    "retardation",
    "suicide attempts",
    "thoughts about dying",
    "thoughts of suicide",
    "weight changes",
    "weight gain",
    "weight loss",
    "Worthlessness"
]


treatments_keywords = [
    "antidepressants",
    "Atypical antidepressants",
    "behavioural activation",
    "cognitive behavioural therapy",
    "Electroconvulsive therapy",
    "ECT"
```

```python
    "fluoxetine",
    "interpersonal psychotherapy",
    "Monoamine oxidase inhibitors",
    "MAOIs",
    "neuromodulation",
    "problem-solving therapy",
    "Psychotherapy",
    "Selective serotonin reuptake inhibitors",
    "SSRIs",
    "Serotonin-norepinephrine reuptake inhibitors",
    "SNRIs",
    "Transcranial magnetic stimulation",
    "Tricyclic antidepressants"
]

# Preprocessing keywords
def preprocess_keyword(text):
    # Convert text to lowercase
    text = text.lower()

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stop words
    stop_words = set(stopwords.words("english"))
    tokens = [token for token in tokens if token not in stop_words]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Join the tokens back into a string
    processed_keywords = " ".join(tokens)

    return processed_keywords


# Preprocess the keywords using the preprocess_text function
symptoms_keywords = [preprocess_keyword(keyword) for keyword in
symptoms_keywords]
treatments_keywords = [preprocess_keyword(keyword) for keyword in
treatments_keywords]
```

```
# Exploratory Data Analysis
# Flatten the symptom_words_dict into a list
symptom_words_flat = [word for words in symptom_words_dict.values() for
word in words]
treatment_words_flat = [word for words in treatment_words_dict.values()
for word in words]

# Remove duplicate occurrences
symptom_words_unique = list(set(symptom_words_flat))
treatment_words_unique = list(set(treatment_words_flat))

# Generate word cloud for symptom words without duplicates
symptom_wordcloud_unique = WordCloud(width=800, height=400,
background_color='white').generate(' '.join(symptom_words_unique))
treatment_wordcloud_unique = WordCloud(width=800, height=400,
background_color='white').generate(' '.join(treatment_words_unique))

# Plot the symptoms word cloud
plt.figure(figsize=(6, 4))
plt.imshow(symptom_wordcloud_unique, interpolation='bilinear')
plt.title('Symptom Words')
plt.axis('off')
plt.show()

# Plot the treatments word cloud
plt.figure(figsize=(6, 4))
plt.imshow(treatment_wordcloud_unique, interpolation='bilinear')
plt.title('Treatment Words')
plt.axis('off')
plt.show()
```

```
# Count the occurrences of each label
label_counts = labeled_data['label'].value_counts()
print(label_counts)

# Create a bar chart with different colors for each label
colors = ['blue', 'green', 'orange', 'grey']
plt.figure(figsize=(10, 6))
bars = plt.bar(label_counts.index, label_counts.values, color=colors,
alpha=0.7)
plt.xlabel('Labels')
plt.ylabel('Count')
plt.title('Number of Labels for Each Category')
plt.xticks([0, 1, 2, 3], ['None', 'Symptoms', 'Treatments', 'Both'])

# Add counts above the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.05, yval,
ha='center', va='bottom')

plt.show()
```

```
# Calculate word count
word_count = labeled_data['sentences'].apply(lambda x:
len(str(x).split()))

# Create histogram
plt.figure(figsize=(10,8))
hist = sns.histplot(data=labeled_data, x=word_count, binwidth=1)

plt.title('Word Count of Sentences of Abstract in Dataset')
plt.xlabel('Word Count')
plt.ylabel('Sentences Count')
plt.show()

max_word_count = word_count.max()
print("The maximum word count is:", max_word_count)

sentences = (sum(word_count < 100)/labeled_data.shape[0])*100
print('Percentage of sentences having less than 100
Words:{:.2f}%'.format(sentences))
```

```
# Handle Imbalanced Dataset
# remove the 'Both' category inside the dataset
df = df[df['label'] != 3]

# Count the occurrences of each label
label_counts = df['label'].value_counts()
print(label_counts)

none_treatment_symptoms = df[(df['label'] == 0)]
other_classes = df[(df['label'] != 0)]
```

```
# Randomly sample instances from 'none treatment and symptoms' class
reduced_none_treatment_symptoms = none_treatment_symptoms.sample(n=5000,
random_state=1)

# Concatenate the reduced 'none treatment and symptoms' DataFrame wth the
other classes DataFrame
balanced_df = pd.concat([reduced_none_treatment_symptoms, other_classes])

# Shuffle the dataset
balanced_df = balanced_df.sample(frac=1,
random_state=1).reset_index(drop=True)

# Count the occurrences of each label
counts = balanced_df['label'].value_counts()
print(counts)

# Save the DataFrame to a CSV file
balanced_df.to_csv('balanced_labeled_data.csv', index=False)
from google.colab import files

# Download the file
files.download('balanced_labeled_data.csv')
```

Code for Model Building

```
# Read the CSV file
df = pd.read_csv(file_name)

# Assuming your DataFrame 'df' has the structure as described
X = df['sentences']
y = df['label'].values
```

```
num_words = 100 #Max. words to use per sentences
max_features = 100 #Max. number of unique words in embeddinbg vector
max_len = 100 #Max. number of words per sentences to be use
embedding_dims = 100 #embedding vector output dimension
num_epochs = 15
batch_size2 = 32
```

```
#sentences Tokenization
tokenizer = tokenizer = Tokenizer(num_words)
tokenizer.fit_on_texts(list(X))

#Convert tokenized sentences to sequnces
X = tokenizer.texts_to_sequences(X)

# padding the sequences
X = sequence.pad_sequences(X, max_len)
```

```
#split the dataset into train and test (train 80%, val 10% and test 10%).
X_train_1, X_temp, y_train_1, y_temp = train_test_split(X, y,
test_size=0.2, random_state=42)
X_val_1, X_test_1, y_val_1, y_test_1 = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

# print the train 80%, val 10% and test 10%.
print(X_train_1.shape)
print(X_val_1.shape)
print(X_test_1.shape)
```

```
#split the dataset into train and test (train 70%, val 15% and test 15%).
X_train_2, X_temp, y_train_2, y_temp = train_test_split(X, y,
test_size=0.3, random_state=42)
X_val_2, X_test_2, y_val_2, y_test_2 = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)
```

```
# print the train 70%, val 15% and test 15%.
print(X_train_2.shape)
print(X_val_2.shape)
print(X_test_2.shape)
```

```
#split the dataset into train and test (train 60%, val 20% and test 20%).
X_train_3, X_temp, y_train_3, y_temp = train_test_split(X, y,
test_size=0.4, random_state=42)
X_val_3, X_test_3, y_val_3, y_test_3 = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

# print the train 60%, val 20% and test 20%.
print(X_train_3.shape)
print(X_val_3.shape)
print(X_test_3.shape)
```

```
# callback used by all three models
early = EarlyStopping(monitor="val_loss", mode="min", patience=4)
```

Code for Benchmark CNN Model

```
# Set 1 dataset split ratio
CNN_model_author_1 = Sequential([
    Embedding(input_dim=max_features, input_length=max_len,
output_dim=embedding_dims),
    SpatialDropout1D(0.5),
    # ... 100 filters with a kernel size of 4 so that each convolution
will consider a window of 4 word embeddings
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    #**batch normalization layer** normalizes the activations of the
previous layer at each batch,
    #i.e. applies a transformation that maintains the mean activation
close to 0 and the activation standard deviation close to 1.
    #It will be added after the activation function between a
convolutional and a max-pooling layer.
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu'),
    Dense(3, activation = 'sigmoid')
])
CNN_model_author_1.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.01), metrics=['accuracy'])
CNN_model_author_1.summary()
CNN_model_author_fit_1 = CNN_model_author_1.fit(X_train_1, y_train_1,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_1,
y_val_1), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_model_author_fit_1.history['accuracy'])
plt.plot(CNN_model_author_fit_1.history['val_accuracy'])
plt.title('Author CNN Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()


# Plot training & validation loss values
plt.plot(CNN_model_author_fit_1.history['loss'])
plt.plot(CNN_model_author_fit_1.history['val_loss'])
plt.title('Author CNN Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

```
# Set 2 dataset split ratio
CNN_model_author_2 = Sequential([
    Embedding(input_dim=max_features, input_length=max_len,
output_dim=embedding_dims),
    SpatialDropout1D(0.5),
    # ... 100 filters with a kernel size of 4 so that each convolution
will consider a window of 4 word embeddings
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    #**batch normalization layer** normalizes the activations of the
previous layer at each batch,
    #i.e. applies a transformation that maintains the mean activation
close to 0 and the activation standard deviation close to 1.
    #It will be added after the activation function between a
convolutional and a max-pooling layer.
```

```
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu'),
    Dense(3, activation = 'sigmoid')
])
CNN_model_author_2.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.01), metrics=['accuracy'])
CNN_model_author_2.summary()
CNN_model_author_fit_2 = CNN_model_author_2.fit(X_train_2, y_train_2,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_2,
y_val_2), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_model_author_fit_2.history['accuracy'])
plt.plot(CNN_model_author_fit_2.history['val_accuracy'])
plt.title('Author CNN Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_model_author_fit_2.history['loss'])
plt.plot(CNN_model_author_fit_2.history['val_loss'])
plt.title('Author CNN Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

```
# Set 3 dataset split ratio
CNN_model_author_3 = Sequential([
    Embedding(input_dim=max_features, input_length=max_len,
output_dim=embedding_dims),
    SpatialDropout1D(0.5),
    # ... 100 filters with a kernel size of 4 so that each convolution
will consider a window of 4 word embeddings
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    #**batch normalization layer** normalizes the activations of the
previous layer at each batch,
    #i.e. applies a transformation that maintains the mean activation
close to 0 and the activation standard deviation close to 1.
    #It will be added after the activation function between a
convolutional and a max-pooling layer.
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu'),
    Dense(3, activation = 'sigmoid')
])
CNN_model_author_3.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.01), metrics=['accuracy'])
CNN_model_author_3.summary()
CNN_model_author_fit_3 = CNN_model_author_3.fit(X_train_3, y_train_3,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_3,
y_val_3), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_model_author_fit_3.history['accuracy'])
plt.plot(CNN_model_author_fit_3.history['val_accuracy'])
plt.title('Author CNN Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_model_author_fit_3.history['loss'])
plt.plot(CNN_model_author_fit_3.history['val_loss'])
plt.title('Author CNN Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

Code for Proposed CNN Model

```
# Set 1 dataset split ratio
CNN_proposed_model_1 = Sequential([
    Embedding(input_dim=max_features, input_length=max_len, 
output_dim=embedding_dims),
    SpatialDropout1D(0.5),
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu', 
kernel_regularizer=regularizers.l2(0.01)),  # Added L2 regularization here
    Dense(3, activation = 'sigmoid', 
kernel_regularizer=regularizers.l2(0.01))  # And here
])

CNN_proposed_model_1.compile(loss='sparse_categorical_crossentropy', 
optimizer=Adam(0.001), metrics=['accuracy'])
CNN_proposed_model_1.summary()
CNN_proposed_model_fit_1 = CNN_proposed_model_1.fit(X_train_1, y_train_1, 
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_1, 
y_val_1), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_proposed_model_fit_1.history['accuracy'])
plt.plot(CNN_proposed_model_fit_1.history['val_accuracy'])
plt.title('Proposed CNN Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower 
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_proposed_model_fit_1.history['loss'])
plt.plot(CNN_proposed_model_fit_1.history['val_loss'])
plt.title('Proposed CNN Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

```
# Set 2 dataset split ratio
CNN_proposed_model_2 = Sequential([
    Embedding(input_dim=max_features, input_length=max_len,
output_dim=embedding_dims),
    SpatialDropout1D(0.5),
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu',
kernel_regularizer=regularizers.l2(0.01)),  # Added L2 regularization here
    Dense(3, activation = 'sigmoid',
kernel_regularizer=regularizers.l2(0.01))  # And here
])

CNN_proposed_model_2.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.001), metrics=['accuracy'])
CNN_proposed_model_2.summary()

CNN_proposed_model_fit_2 = CNN_proposed_model_2.fit(X_train_2, y_train_2,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_2,
y_val_2), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_proposed_model_fit_2.history['accuracy'])
plt.plot(CNN_proposed_model_fit_2.history['val_accuracy'])
plt.title('Proposed CNN Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_proposed_model_fit_2.history['loss'])
plt.plot(CNN_proposed_model_fit_2.history['val_loss'])
plt.title('Proposed CNN Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

```
# Set 3 dataset split ratio
CNN_proposed_model_3 = Sequential([
    Embedding(input_dim=max_features, input_length=max_len,
output_dim=embedding_dims),
    SpatialDropout1D(0.5),
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu',
kernel_regularizer=regularizers.l2(0.01)),  # Added L2 regularization here
    Dense(3, activation = 'sigmoid',
kernel_regularizer=regularizers.l2(0.01))  # And here
])

CNN_proposed_model_3.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.001), metrics=['accuracy'])
CNN_proposed_model_3.summary()
CNN_proposed_model_fit_3 = CNN_proposed_model_3.fit(X_train_3, y_train_3,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_3,
y_val_3), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_proposed_model_fit_3.history['accuracy'])
plt.plot(CNN_proposed_model_fit_3.history['val_accuracy'])
plt.title('Proposed CNN Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_proposed_model_fit_3.history['loss'])
plt.plot(CNN_proposed_model_fit_3.history['val_loss'])
plt.title('Proposed CNN Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

Code for Proposed CNN Model + Word2vec

```
# Set 1 dataset split ratio
# Define the Word2Vec model
sentences = [tokenizer.sequences_to_texts([sequence])[0].split() for
sequence in X_train_1]
#train word2vec model
embedding_dims = 100 #embedding vector output dimension
max_len = 100 #Max. number of words to be use
word2VecModel = gensim.models.Word2Vec(sentences, vector_size=
embedding_dims, window=5, workers=4, min_count=1)

# Get the vocabulary size
words = list(word2VecModel.wv.key_to_index)
print('Vocabulary size: %d' % len(words))
# Get vocabulary from Word2Vec model
vocabulary = word2VecModel.wv.index_to_key

# Initialize embedding matrix
num_words = len(vocabulary) + 1  # Add 1 for padding token
embedding_matrix = np.zeros((num_words, embedding_dims))

# Fill embedding matrix
for i, word in enumerate(vocabulary):
    if word in word2VecModel.wv:
        embedding_matrix[i] = word2VecModel.wv[word]

# Optionally handle missing words (out of vocabulary)

# Print shape of embedding matrix
print("Shape of embedding matrix:", embedding_matrix.shape)
```

```
CNN_Word2Vec_model_1 = Sequential([
    Embedding(input_dim =embedding_matrix.shape[0], input_length=max_len,
output_dim=embedding_matrix.shape[1],weights=[embedding_matrix],
trainable=True),
    SpatialDropout1D(0.5),
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu',
kernel_regularizer=regularizers.l2(0.01)),  # Added L2 regularization here
```

```
    Dense(3, activation = 'sigmoid',
kernel_regularizer=regularizers.l2(0.01))  # And here
])

CNN_Word2Vec_model_1.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.001), metrics=['accuracy'])
CNN_Word2Vec_model_1.summary()
CNN_Word2Vec_model_fit_1 = CNN_Word2Vec_model_1.fit(X_train_1, y_train_1,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_1,
y_val_1), callbacks=[early])
CNN_Word2Vec_model_1.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.001), metrics=['accuracy'])
CNN_Word2Vec_model_1.summary()
CNN_Word2Vec_model_fit_1 = CNN_Word2Vec_model_1.fit(X_train_1, y_train_1,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_1,
y_val_1), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_Word2Vec_model_fit_1.history['accuracy'])
plt.plot(CNN_Word2Vec_model_fit_1.history['val_accuracy'])
plt.title('Proposed CNN + Word2Vec Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_Word2Vec_model_fit_1.history['loss'])
plt.plot(CNN_Word2Vec_model_fit_1.history['val_loss'])
plt.title('Proposed CNN + Word2Vec Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

```
# Set 2 dataset split ratio
# Define the Word2Vec model
sentences = [tokenizer.sequences_to_texts([sequence])[0].split() for
sequence in X_train_2]
#train word2vec model
embedding_dims = 100 #embedding vector output dimension
```

```
max_len = 100 #Max. number of words per toxic comment to be use
word2VecModel = gensim.models.Word2Vec(sentences, vector_size=
embedding_dims, window=5, workers=4, min_count=1)
# Get the vocabulary size
words = list(word2VecModel.wv.key_to_index)
print('Vocabulary size: %d' % len(words))

# Get vocabulary from Word2Vec model
vocabulary = word2VecModel.wv.index_to_key

# Initialize embedding matrix
num_words = len(vocabulary) + 1  # Add 1 for padding token
embedding_matrix = np.zeros((num_words, embedding_dims))

# Fill embedding matrix
for i, word in enumerate(vocabulary):
    if word in word2VecModel.wv:
        embedding_matrix[i] = word2VecModel.wv[word]

# Optionally handle missing words (out of vocabulary)

# Print shape of embedding matrix
print("Shape of embedding matrix:", embedding_matrix.shape)
```

```
CNN_Word2Vec_model_2 = Sequential([
    Embedding(input_dim =embedding_matrix.shape[0], input_length=max_len,
output_dim=embedding_matrix.shape[1],weights=[embedding_matrix],
trainable=True),
    SpatialDropout1D(0.5),
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu',
kernel_regularizer=regularizers.l2(0.01)),  # Added L2 regularization here
    Dense(3, activation = 'sigmoid',
kernel_regularizer=regularizers.l2(0.01))  # And here
])
CNN_Word2Vec_model_2.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.001), metrics=['accuracy'])
CNN_Word2Vec_model_2.summary()
CNN_Word2Vec_model_fit_2 = CNN_Word2Vec_model_2.fit(X_train_2, y_train_2,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_2,
y_val_2), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_Word2Vec_model_fit_2.history['accuracy'])
plt.plot(CNN_Word2Vec_model_fit_2.history['val_accuracy'])
plt.title('Proposed CNN + Word2Vec Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_Word2Vec_model_fit_2.history['loss'])
plt.plot(CNN_Word2Vec_model_fit_2.history['val_loss'])
plt.title('Proposed CNN + Word2Vec Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

```
# Set 3 dataset split ratio
# Define the Word2Vec model
sentences = [tokenizer.sequences_to_texts([sequence])[0].split() for
sequence in X_train_3]
#train word2vec model
embedding_dims = 100 #embedding vector output dimension
max_len = 100 #Max. number of words per toxic comment to be use
word2VecModel = gensim.models.Word2Vec(sentences, vector_size=
embedding_dims, window=5, workers=4, min_count=1)
# Get the vocabulary size
words = list(word2VecModel.wv.key_to_index)
print('Vocabulary size: %d' % len(words))

# Get vocabulary from Word2Vec model
vocabulary = word2VecModel.wv.index_to_key

# Initialize embedding matrix
num_words = len(vocabulary) + 1  # Add 1 for padding token
embedding_matrix = np.zeros((num_words, embedding_dims))

# Fill embedding matrix
for i, word in enumerate(vocabulary):
    if word in word2VecModel.wv:
        embedding_matrix[i] = word2VecModel.wv[word]
```

```
# Optionally handle missing words (out of vocabulary)

# Print shape of embedding matrix
print("Shape of embedding matrix:", embedding_matrix.shape)
CNN_Word2Vec_model_3 = Sequential([
    Embedding(input_dim =embedding_matrix.shape[0], input_length=max_len,
output_dim=embedding_matrix.shape[1],weights=[embedding_matrix],
trainable=True),
    SpatialDropout1D(0.5),
    Conv1D(filters=100, kernel_size=4, padding='same', activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    GlobalMaxPool1D(),
    Dense(50, activation = 'relu',
kernel_regularizer=regularizers.l2(0.01)),  # Added L2 regularization here
    Dense(3, activation = 'sigmoid',
kernel_regularizer=regularizers.l2(0.01))  # And here
])
CNN_Word2Vec_model_3.compile(loss='sparse_categorical_crossentropy',
optimizer=Adam(0.001), metrics=['accuracy'])
CNN_Word2Vec_model_3.summary()
CNN_Word2Vec_model_fit_3 = CNN_Word2Vec_model_3.fit(X_train_3, y_train_3,
batch_size=batch_size2, epochs=num_epochs, validation_data=(X_val_3,
y_val_3), callbacks=[early])
```

```
# Plot training & validation accuracy values
plt.plot(CNN_Word2Vec_model_fit_3.history['accuracy'])
plt.plot(CNN_Word2Vec_model_fit_3.history['val_accuracy'])
plt.title('Proposed CNN + Word2Vec Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower
right')
plt.show()

# Plot training & validation loss values
plt.plot(CNN_Word2Vec_model_fit_3.history['loss'])
plt.plot(CNN_Word2Vec_model_fit_3.history['val_loss'])
plt.title('Proposed CNN + Word2Vec Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```

Code for Model Evaluation

* The code for the confusion matrix and classification report are same for all model built, by changing the respective model variable defined.

```
# Make predictions on the testing data
y_pred = CNN_model_author_1.predict(X_test_1)

# Ensure y_pred is an array of predicted probabilities and get predicted
classes
y_pred_classes = np.argmax(y_pred, axis=1)

# Use y_val directly as the true labels (assuming y_val is already in the
correct shape)
y_true_classes = y_test_1

if len(y_true_classes) != len(y_pred_classes):
    print("Error: The lengths of y_true_classes and y_pred_classes do not
match.")
else:
    # Define the mapping from numerical labels to string labels
    label_mapping = {0: 'none', 1: 'symptoms', 2: 'treatments'}
    label_names = ['none', 'symptoms', 'treatments']

    # Map the numerical labels to string labels
    y_true_mapped = [label_mapping[label] for label in y_true_classes]
    y_pred_mapped = [label_mapping[label] for label in y_pred_classes]

    # Compute confusion matrix
    conf_matrix = confusion_matrix(y_true_mapped, y_pred_mapped,
labels=label_names)

    # Plot confusion matrix
    fig, ax = plt.subplots(figsize=(8, 6))
    cax = ax.matshow(conf_matrix, cmap=plt.cm.Blues)
    plt.title('\nConfusion Matrix\n')
    fig.colorbar(cax)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    ax.set_xticklabels([''] + label_names)
    ax.set_yticklabels([''] + label_names)
    # Add counts to each cell
    for (i, j), val in np.ndenumerate(conf_matrix):
        ax.text(j, i, val, ha='center', va='center', color='red')

    plt.show()

    # Print classification report
    print(classification_report(y_true_classes, y_pred_classes,
target_names=label_names))
```

CODE BOOKLET

SCHOOL OF COMPUTING

2024

CB_V1_2024