import streamlit as st

import pandas as pd

import numpy as np

from Chronic\_disease\_prediction import DiseasePrediction # Import your DiseasePrediction class from your module

# Create an instance of the DiseasePrediction class

dp = DiseasePrediction(model\_name='decision\_tree') # Specify the model name without the '.joblib' extension

# Define the Streamlit app

def main():

st.title("Chronic Disease Prediction")

# User input for symptoms

symptoms = st.text\_input("Enter a list of symptoms (comma-separated):")

if st.button("Predict Disease"):

# Make a prediction

predicted\_disease = dp.make\_prediction(saved\_model\_name='decision\_tree', test\_data=symptoms)

if predicted\_disease:

st.success(f"Predicted Disease: {predicted\_disease[0]}")

else:

st.error("Unable to make a prediction. Please check your input.")

if \_name\_ == "\_\_main\_\_":

main()# Import Dependencies

import yaml

from joblib import dump, load

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

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

# Naive Bayes Approach

from sklearn.naive\_bayes import MultinomialNB

# Trees Approach

from sklearn.tree import DecisionTreeClassifier

# Ensemble Approach

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

import seaborn as sn

import matplotlib.pyplot as plt

class DiseasePrediction:

# Initialize and Load the Config File

def \_init\_(self, model\_name=None):

# Load Config File

try:

# Load Configuration

with open("C:\\Users\\Gandhana\\Desktop\\Varcons\\config.yaml", 'r') as f:

self.config = yaml.safe\_load(f)

except FileNotFoundError as e:

raise ConfigurationError("Configuration file not found. Please make sure 'config.yaml' is in the correct location.") from e

except yaml.YAMLError as e:

raise ConfigurationError("Error reading the configuration file. Please check the file format and content.") from e

# Verbose

self.verbose = self.config['verbose']

# Load Training Data

self.train\_features, self.train\_labels, self.train\_df = self.\_load\_train\_dataset()

# Load Test Data

self.test\_features, self.test\_labels, self.test\_df = self.\_load\_test\_dataset()

# Feature Correlation in Training Data

self.\_feature\_correlation(data\_frame=self.train\_df, show\_fig=False)

# Model Definition

self.model\_name = model\_name

# Model Save Path

self.model\_save\_path = self.config['model\_save\_path']

self.clf = None

# Function to Load Train Dataset

def \_load\_train\_dataset(self):

df\_train = pd.read\_csv(self.config['dataset']['training\_data\_path'])

cols = df\_train.columns

cols = cols[:-2]

train\_features = df\_train[cols]

train\_labels = df\_train['prognosis']

# Check for data sanity

assert (len(train\_features.iloc[0]) == 132)

assert (len(train\_labels) == train\_features.shape[0])

if self.verbose:

print("Length of Training Data: ", df\_train.shape)

print("Training Features: ", train\_features.shape)

print("Training Labels: ", train\_labels.shape)

return train\_features, train\_labels, df\_train

# Function to Load Test Dataset

def \_load\_test\_dataset(self):

df\_test = pd.read\_csv(self.config['dataset']['test\_data\_path'])

cols = df\_test.columns

cols = cols[:-1]

test\_features = df\_test[cols]

test\_labels = df\_test['prognosis']

# Check for data sanity

assert (len(test\_features.iloc[0]) == 132)

assert (len(test\_labels) == test\_features.shape[0])

if self.verbose:

print("Length of Test Data: ", df\_test.shape)

print("Test Features: ", test\_features.shape)

print("Test Labels: ", test\_labels.shape)

return test\_features, test\_labels, df\_test

# Features Correlation

def \_feature\_correlation(self, data\_frame=None, show\_fig=False):

# Get Feature Correlation

numeric\_cols = data\_frame.select\_dtypes(include=['float64', 'int64'])

corr = numeric\_cols.corr()

sn.heatmap(corr, square=True, annot=False, cmap="YlGnBu")

plt.title("Feature Correlation")

plt.tight\_layout()

if show\_fig:

plt.show()

plt.savefig('feature\_correlation.png')

# Dataset Train Validation Split

def \_train\_val\_split(self):

X\_train, X\_val, y\_train, y\_val = train\_test\_split(self.train\_features, self.train\_labels,

test\_size=self.config['dataset']['validation\_size'],

random\_state=self.config['random\_state'])

if self.verbose:

print("Number of Training Features: {0}\tNumber of Training Labels: {1}".format(len(X\_train), len(y\_train)))

print("Number of Validation Features: {0}\tNumber of Validation Labels: {1}".format(len(X\_val), len(y\_val)))

return X\_train, y\_train, X\_val, y\_val

# Model Selection

def select\_model(self):

if self.model\_name == 'mnb':

self.clf = MultinomialNB()

elif self.model\_name == 'decision\_tree':

self.clf = DecisionTreeClassifier(criterion=self.config['model']['decision\_tree']['criterion'])

elif self.model\_name == 'random\_forest':

self.clf = RandomForestClassifier(n\_estimators=self.config['model']['random\_forest']['n\_estimators'])

elif self.model\_name == 'gradient\_boost':

self.clf = GradientBoostingClassifier(n\_estimators=self.config['model']['gradient\_boost']['n\_estimators'],

criterion=self.config['model']['gradient\_boost']['criterion'])

return self.clf

# ML Model

def train\_model(self):

# Get the Data

X\_train, y\_train, X\_val, y\_val = self.\_train\_val\_split()

classifier = self.select\_model()

# Training the Model

classifier = classifier.fit(X\_train, y\_train)

self.clf = classifier

# Trained Model Evaluation on Validation Dataset

confidence = classifier.score(X\_val, y\_val)

# Validation Data Prediction

y\_pred = classifier.predict(X\_val)

# Model Validation Accuracy

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

# Model Confusion Matrix

conf\_mat = confusion\_matrix(y\_val, y\_pred)

# Model Classification Report

clf\_report = classification\_report(y\_val, y\_pred)

# Model Cross Validation Score

score = cross\_val\_score(classifier, X\_val, y\_val, cv=3)

if self.verbose:

print('\nTraining Accuracy: ', confidence\*100)

print('\nValidation Prediction: ', y\_pred)

print('\nValidation Accuracy: ', accuracy\*100)

print('\nValidation Confusion Matrix: \n', conf\_mat)

print('\nCross Validation Score: \n', score)

print('\nClassification Report: \n', clf\_report)

# Save Trained Model

dump(classifier, str(self.model\_save\_path + self.model\_name + ".joblib"))

self.clf = classifier

# Function to Make Predictions on Test Data

def make\_prediction(self, saved\_model\_name=None, test\_data=None):

try:

# Load Trained Model

if self.clf is None:

print(f"Model {saved\_model\_name} not found...")

self.clf = load(str(self.model\_save\_path + saved\_model\_name + ".joblib"))

return None

if test\_data is not None:

preprocessed\_data = self.preprocess\_text\_input(test\_data)

result = self.clf.predict(preprocessed\_data)

return result

else:

print("Test data not provided. Please provide test data.")

return None

except Exception as e:

print("Model not found...")

return None

def preprocess\_text\_input(self, text\_input):

# Tokenize the text (split into words)

words = text\_input.split(',')

# Convert words into a numerical format (e.g., using word embeddings or TF-IDF)

# Here, we'll just use a simple one-hot encoding as an example

unique\_words = list(set(words))

word\_indices = {word: i for i, word in enumerate(unique\_words)}

num\_features = len(unique\_words)

# Create a one-hot encoded vector

one\_hot\_vector = np.zeros(num\_features)

for word in words:

if word in word\_indices:

index = word\_indices[word]

one\_hot\_vector[index] = 1

# Return the preprocessed data as a DataFrame or a NumPy array

return pd.DataFrame([one\_hot\_vector], columns=unique\_words)

if \_name\_ == "\_\_main\_\_":

# Model Currently Training

current\_model\_name = 'decision\_tree'

# Instantiate the Class

dp = DiseasePrediction(model\_name=current\_model\_name)

# Train the Model

dp.train\_model()

# Get Model Performance on Test Data

test\_accuracy, classification\_report = dp.make\_prediction(saved\_model\_name=current\_model\_name)

print("Model Test Accuracy: ", test\_accuracy)

print("Test Data Classification Report: \n", classification\_report

symptoms = input()

dp = DiseasePrediction(model\_name='decision\_tree') # Specify the model name without the '.joblib' extension

# Make a prediction

predicted\_disease = dp.make\_prediction(saved\_model\_name='decision\_tree', test\_data=symptoms)

print("Predicted Disease:", predicted\_disease)