import time

import board

import busio

import pickle

import numpy as np

import pandas as pd

from adafruit\_adxl34x import ADXL343

# Load the pre-trained models

with open('dt\_model.pkl', 'rb') as model\_file:

dt\_model = pickle.load(model\_file)

with open('knn\_model.pkl', 'rb') as model\_file:

knn\_model = pickle.load(model\_file)

with open('rf\_model.pkl', 'rb') as model\_file:

rf\_model = pickle.load(model\_file)

# Setup I2C connection and ADXL343 sensor

i2c = busio.I2C(board.SCL, board.SDA)

accelerometer = ADXL343(i2c)

accelerometer.range = ADXL343.Range.RANGE\_4\_G

# Feature extraction function

def extract\_features(data):

if len(data) > 0:

df = pd.DataFrame(data, columns=['x', 'y', 'z'])

df['magnitude'] = np.sqrt(df['x']\*\*2 + df['y']\*\*2 + df['z']\*\*2)

# Extract features

features = {

'mean\_x': df['x'].mean(),

'mean\_y': df['y'].mean(),

'mean\_z': df['z'].mean(),

'magnitude': df['magnitude'].mean(),

'std\_x': df['x'].std(),

'std\_y': df['y'].std(),

'std\_z': df['z'].std(),

'var\_x': df['x'].var(),

'var\_y': df['y'].var(),

'var\_z': df['z'].var(),

'range\_x': df['x'].max() - df['x'].min(),

'range\_y': df['y'].max() - df['y'].min(),

'range\_z': df['z'].max() - df['z'].min()

}

return pd.DataFrame([features])

return None

# Collect data and make predictions

data\_buffer = []

try:

while True:

x, y, z = accelerometer.acceleration

data\_buffer.append((x, y, z))

if len(data\_buffer) >= 10: # Adjust this number based on your needs

features\_df = extract\_features(data\_buffer)

if features\_df is not None:

# Predict using all models

dt\_prediction = dt\_model.predict(features\_df)

knn\_prediction = knn\_model.predict(features\_df)

rf\_prediction = rf\_model.predict(features\_df)

# Print predictions from all models

print(f'Decision Tree Prediction: {dt\_prediction[0]}')

print(f'KNN Prediction: {knn\_prediction[0]}')

print(f'Random Forest Prediction: {rf\_prediction[0]}')

# Clear the buffer

data\_buffer = []

time.sleep(0.1) # Adjust sleep for desired sampling rate

except KeyboardInterrupt:

print("Stopping data collection.")

finally:

print("Data collection complete.")

FOR CONFUSION MATRIX (DRAFT)

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

# Generate confusion matrix for each model

dt\_cm = confusion\_matrix(y\_true, dt\_predictions)

knn\_cm = confusion\_matrix(y\_true, knn\_predictions)

rf\_cm = confusion\_matrix(y\_true, rf\_predictions)

# Print classification reports

print("Decision Tree Classification Report:")

print(classification\_report(y\_true, dt\_predictions))

print("KNN Classification Report:")

print(classification\_report(y\_true, knn\_predictions))

print("Random Forest Classification Report:")

print(classification\_report(y\_true, rf\_predictions))

VISUALISING CONFUSION MATRIX (I think pick which one is most accurate?idk)

import matplotlib.pyplot as plt

import seaborn as sns

def plot\_confusion\_matrix(cm, model\_name):

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['walking', 'sitting', 'running'], yticklabels=['walking', 'sitting', 'running'])

plt.title(f'Confusion Matrix for {model\_name}')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Plot confusion matrices for each model

plot\_confusion\_matrix(dt\_cm, 'Decision Tree')

plot\_confusion\_matrix(knn\_cm, 'KNN')

plot\_confusion\_matrix(rf\_cm, 'Random Forest')