**Assignment 0. Human Action Recognition using Machine Learning**

**Student ID:** m5281022 **Name:** LAI HUI SHAN

**Programming Language Used:** Python

**Required Libraries:**

* numpy
* matplotlib
* scikit-learn
* pandas

**How to Run the Program:**

1. **Environment Setup**:
   * Ensure Python 3.x is installed on your system.
   * Install the required libraries using the following commands:

pip install numpy matplotlib scikit-learn pandas

1. **Executing the Script**:
   * Save the code in a file named **action\_recognition.py**.
   * Open a command line or terminal window.
   * Navigate to the directory where the script is saved.
   * Run the script using the command:

python action\_recognition.py

**Code Explanation:**

* **read\_data**: This function reads the acceleration data from a text file and returns it as a NumPy array.
* **plot\_and\_save\_data**: This function generates a line plot for each axis of the acceleration data and saves it as an image file.
* **calculate\_rms**: This function calculates the Root Mean Square (RMS) of the acceleration data for each axis.
* **plot\_and\_save\_3d\_features**: This function creates a 3D scatter plot of the RMS features and saves it to a file.
* **classify\_actions**: This function trains a K-Nearest Neighbors (KNN) classifier with the training features and labels, and then predicts the class labels for the test features.
* **create\_result\_table**: This function creates a result table with the test sample numbers and their predicted class labels, and saves it as a CSV file.
* **main**: This is the main function that orchestrates the reading of data, feature calculation, plotting, classification, and generation of the result table.

|  |
| --- |
| **Source Code:** |
| import os  import numpy as np  import matplotlib.pyplot as plt  from mpl\_toolkits.mplot3d import Axes3D  from sklearn.neighbors import KNeighborsClassifier  from sklearn.metrics import classification\_report, accuracy\_score  import pandas as pd # Import pandas for data manipulation  # Function to read data from a file  def read\_data(filepath):  """Reads data from a given filepath and returns it as a NumPy array."""  data = np.loadtxt(filepath)  return data  # Function to plot data and save the plot as an image  def plot\_and\_save\_data(data, title, save\_path):  """Generates and saves a plot for given data."""  plt.figure(figsize=(10, 4))  plt.plot(data[:, 0], label='X-axis')  plt.plot(data[:, 1], label='Y-axis')  plt.plot(data[:, 2], label='Z-axis')  plt.title(title)  plt.legend()  plt.savefig(save\_path) # Save the figure  plt.close() # Close the plot to avoid displaying it  # Function to calculate RMS from data  def calculate\_rms(data):  """Calculates the Root Mean Square (RMS) for each axis of the data."""  return np.sqrt(np.mean(data\*\*2, axis=0))  # Function to plot 3D features and save the plot  def plot\_and\_save\_3d\_features(features, labels, save\_folder):  """Generates a 3D scatter plot of features and saves it."""  fig = plt.figure()  ax = fig.add\_subplot(111, projection='3d')  colors = ['b', 'g', 'r']  for i, action\_features in enumerate(features):  xs = [f[0] for f in action\_features]  ys = [f[1] for f in action\_features]  zs = [f[2] for f in action\_features]  ax.scatter(xs, ys, zs, c=colors[i % len(colors)], label=labels[i])  ax.set\_xlabel('RMS Feature of X-axis')  ax.set\_ylabel('RMS Feature of Y-axis')  ax.set\_zlabel('RMS Feature of Z-axis')  ax.legend()  plt.title('3D Feature Plot')  plt.savefig(os.path.join(save\_folder, '3D\_Features.png')) # Save the 3D features plot  plt.close()  # Function to classify actions using KNN  def classify\_actions(train\_features, train\_labels, test\_features):  """Train KNN classifier and predict classes for test features."""  classifier = KNeighborsClassifier(n\_neighbors=3)  classifier.fit(train\_features, train\_labels)  predictions = classifier.predict(test\_features)  return predictions  # Function to create a result table and save it as a CSV  def create\_result\_table(test\_files, predictions, action\_names):  """Creates a pandas DataFrame with the test files and their predicted class labels."""  results = pd.DataFrame({  'Sample #': [os.path.splitext(file)[0] for file in test\_files], # Extracts the base filename without the extension  'Class': [f'Act0{label+1}: {action\_names[label]}' for label in predictions] # Formats the label as per given style  })  results.sort\_values(by='Sample #', inplace=True) # Sorts the results by the sample number  print(results)  results.to\_csv("classification\_results.csv", index=False)  # The main function of the script  def main():  """Main function to execute the action recognition tasks."""  base\_path = 'C:/Users/HuiShan/Documents/HAPP/Assignment0'  actions = ['act01', 'act02', 'act03']  action\_names = ['Walking', 'Sitting', 'Jogging']  all\_features = []  # Loop over each action, read data, calculate features, plot data, and save plots  for action\_folder, action\_name in zip(actions, action\_names):  action\_path = os.path.join(base\_path, action\_folder)  file\_names = os.listdir(action\_path)  save\_folder = os.path.join(base\_path, action\_folder + '\_plots')  os.makedirs(save\_folder, exist\_ok=True) # Ensures that the directory exists  action\_features = []  for file\_name in file\_names:  file\_path = os.path.join(action\_path, file\_name)  data = read\_data(file\_path)  rms = calculate\_rms(data)  action\_features.append(rms)  plot\_title = f'{action\_name} Sample {file\_name.split(".")[0]}'  save\_path = os.path.join(save\_folder, f'{file\_name}.png')  plot\_and\_save\_data(data, plot\_title, save\_path)  all\_features.append(action\_features)  # Combine features from all actions for training  train\_features = np.concatenate(all\_features[:-1])  train\_labels = np.concatenate([[i] \* len(features) for i, features in enumerate(all\_features[:-1])])  # Read, calculate RMS for test data, and classify actions  test\_folder = os.path.join(base\_path, 'test')  test\_files = os.listdir(test\_folder)  test\_features = []  for test\_file in test\_files:  test\_file\_path = os.path.join(test\_folder, test\_file)  test\_data = read\_data(test\_file\_path)  test\_rms = calculate\_rms(test\_data)  test\_features.append(test\_rms)  test\_features = np.array(test\_features)  predictions = classify\_actions(train\_features, train\_labels, test\_features)  # Generate a table of results and save it  create\_result\_table(test\_files, predictions, action\_names)  # Save the 3D features for all actions  plot\_and\_save\_3d\_features(all\_features, action\_names, base\_path)  if \_\_name\_\_ == '\_\_main\_\_':  main() |