Reading Training & Testing Data

We have used the **np.load()** function from the NumPy library to read the training data from various sensor sources. The training data is stored in numpy arrays for further processing and analysis.

The **np.load()** function is specifically designed to load data stored in the .npy format, which is a file format specific to NumPy. This format enables efficient storage and retrieval of large arrays of numerical data.

We use **the np.load()** function to load the data from the following training data files:

train_Accelerometer.npy: Contains accelerometer data for training.

train_Gravity.npy: Contains gravity data for training.

train_Gyroscope.npy: Contains gyroscope data for training.

train_JinsAccelerometer.npy: Contains Jins accelerometer data for training.

train_JinsGyroscope.npy: Contains Jins gyroscope data for training.

train_LinearAcceleration.npy: Contains linear acceleration data for training.

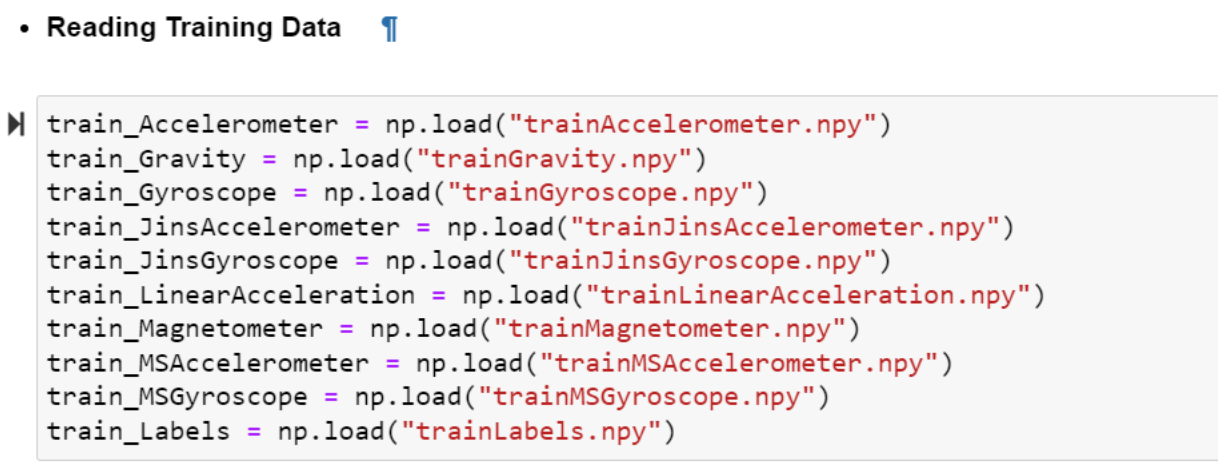
train_Magnetometer.npy: Contains magnetometer data for training.

train_MSAccelerometer.npy: Contains MS accelerometer data for training.

train_MSGyroscope.npy: Contains MS gyroscope data for training.

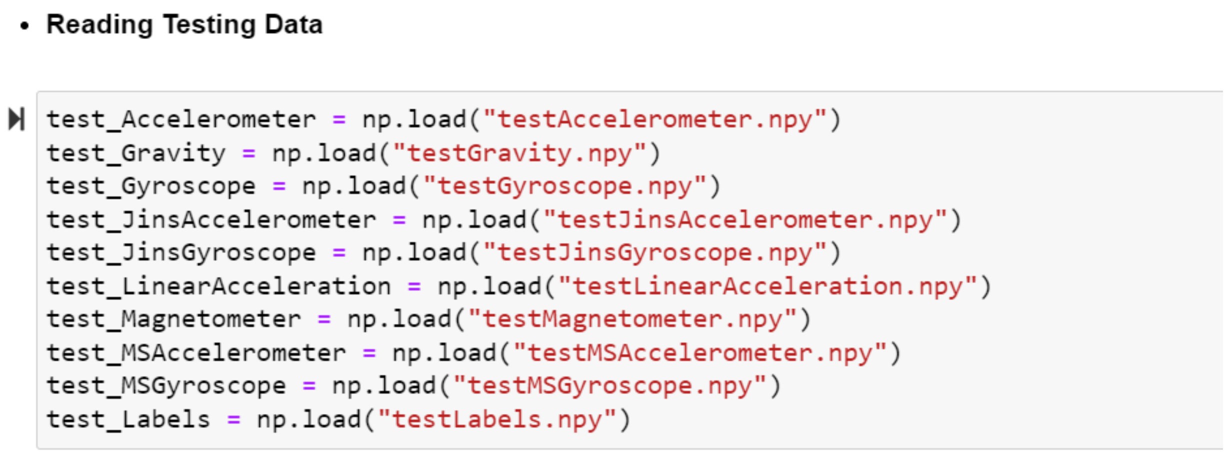
train_Labels.npy: Contains labels or target values for training.

By loading the training data into separate numpy arrays, we can easily access and manipulate the data during subsequent steps such as feature extraction and classification.



- **Reading Training Data** ¶

```
train_Accelerometer = np.load("trainAccelerometer.npy")
train_Gravity = np.load("trainGravity.npy")
train_Gyroscope = np.load("trainGyroscope.npy")
train_JinsAccelerometer = np.load("trainJinsAccelerometer.npy")
train_JinsGyroscope = np.load("trainJinsGyroscope.npy")
train_LinearAcceleration = np.load("trainLinearAcceleration.npy")
train_Magnetometer = np.load("trainMagnetometer.npy")
train_MSAccelerometer = np.load("trainMSAccelerometer.npy")
train_MSGyroscope = np.load("trainMSGyroscope.npy")
train_Labels = np.load("trainLabels.npy")
```



- **Reading Testing Data**

```
test_Accelerometer = np.load("testAccelerometer.npy")
test_Gravity = np.load("testGravity.npy")
test_Gyroscope = np.load("testGyroscope.npy")
test_JinsAccelerometer = np.load("testJinsAccelerometer.npy")
test_JinsGyroscope = np.load("testJinsGyroscope.npy")
test_LinearAcceleration = np.load("testLinearAcceleration.npy")
test_Magnetometer = np.load("testMagnetometer.npy")
test_MSAccelerometer = np.load("testMSAccelerometer.npy")
test_MSGyroscope = np.load("testMSGyroscope.npy")
test_Labels = np.load("testLabels.npy")
```

Storing Dataset in Data Structures

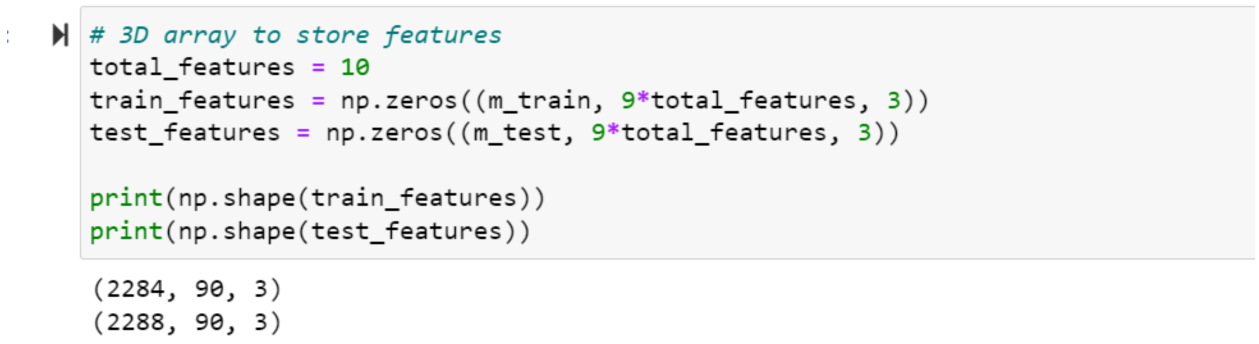
We have used numpy arrays to store the sensory data. There are total 18 numpy arrays for testing and training.

Feature Extraction & Learning

We initialize empty arrays to store the extracted features from the sensor data. These arrays are structured based on the number of training and testing examples and the total number of feature.

We have calculated statistical features and assign them to specific positions in the feature arrays for each sensor type. The extracted features include:

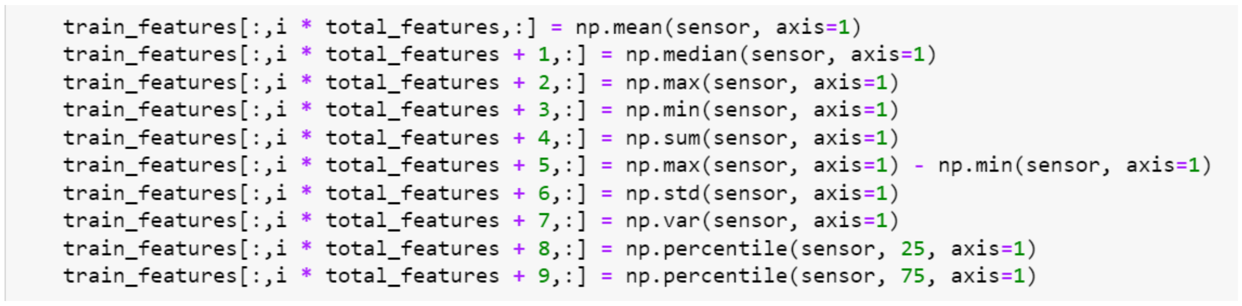
- Mean
- Median
- Maximum
- Minimum
- Sum
- Range
- Standard Deviation
- Variance
- Percentile-25
- Percentile-75



```
# 3D array to store features
total_features = 10
train_features = np.zeros((m_train, 9*total_features, 3))
test_features = np.zeros((m_test, 9*total_features, 3))

print(np.shape(train_features))
print(np.shape(test_features))
```

```
(2284, 90, 3)
(2288, 90, 3)
```



```
train_features[:,i * total_features,:] = np.mean(sensor, axis=1)
train_features[:,i * total_features + 1,:] = np.median(sensor, axis=1)
train_features[:,i * total_features + 2,:] = np.max(sensor, axis=1)
train_features[:,i * total_features + 3,:] = np.min(sensor, axis=1)
train_features[:,i * total_features + 4,:] = np.sum(sensor, axis=1)
train_features[:,i * total_features + 5,:] = np.max(sensor, axis=1) - np.min(sensor, axis=1)
train_features[:,i * total_features + 6,:] = np.std(sensor, axis=1)
train_features[:,i * total_features + 7,:] = np.var(sensor, axis=1)
train_features[:,i * total_features + 8,:] = np.percentile(sensor, 25, axis=1)
train_features[:,i * total_features + 9,:] = np.percentile(sensor, 75, axis=1)
```

Reshaping to 2D

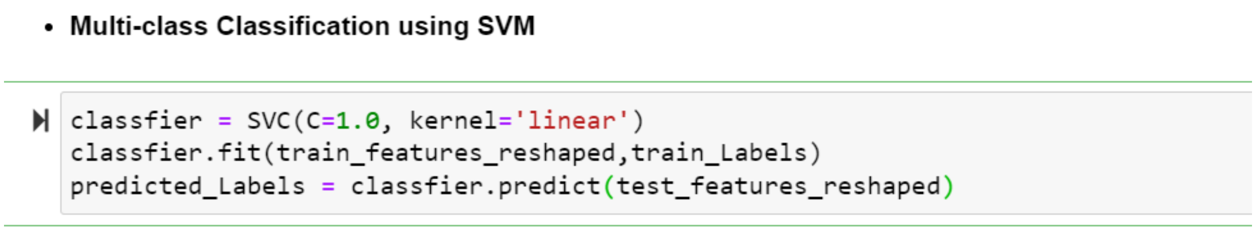
We have reshaped the feature arrays into a 2D format. This reshaping process converts the multi-dimensional feature arrays into a flat structure, where each row represents an example and each column represents a specific feature.

Reshaping the feature arrays to a 2D format is necessary to prepare the features for classification algorithms that expect a 2D input format.

Multi-class Classification using SVM

In this section, we perform multi-class classification using a Support Vector Machine (SVM) classifier with a linear kernel. We instantiate the SVM classifier using the **SVC** class from the **sklearn.svm** module. It is trained on the reshaped training feature array, and predictions are made for the reshaped testing feature array.

SVM is a powerful classification algorithm that aims to find an optimal hyperplane in the feature space that separates different classes. By using SVM with a linear kernel, we classify the activity recognition based on the extracted features.



- **Multi-class Classification using SVM**

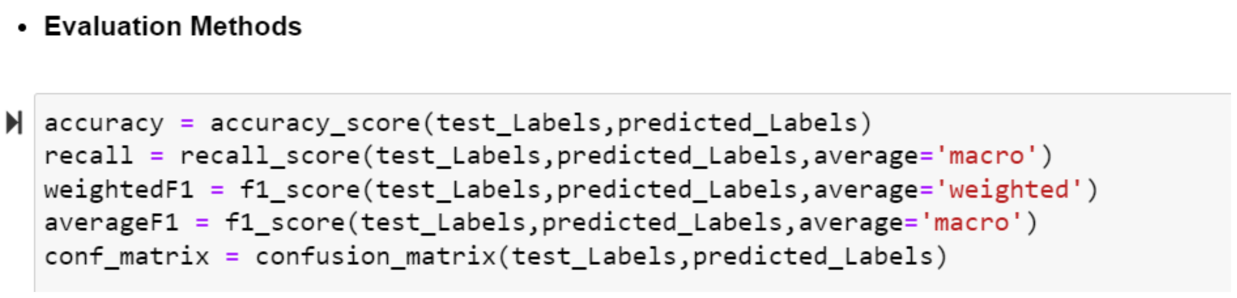
```
classfier = SVC(C=1.0, kernel='linear')
classfier.fit(train_features_reshaped,train_Labels)
predicted_Labels = classfier.predict(test_features_reshaped)
```

Evaluation Methods

We evaluate the performance of the classification model using various evaluation metrics in this section of the code. The evaluated metrics include:

- Accuracy
- Recall
- Weighted F1-score
- Average F1-score

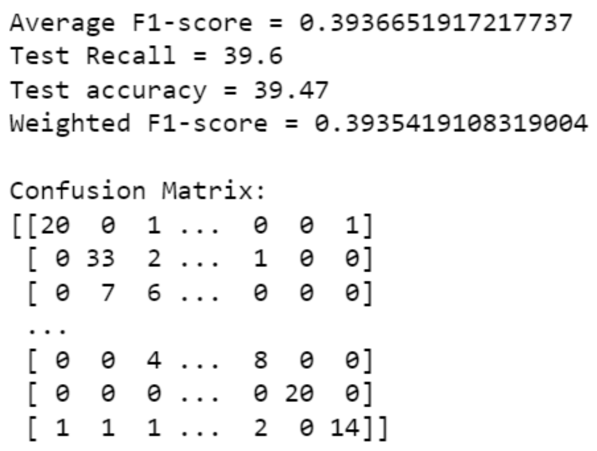
Additionally, a confusion matrix is generated to provide a visual representation of the classification results.



- **Evaluation Methods**

```
accuracy = accuracy_score(test_Labels,predicted_Labels)
recall = recall_score(test_Labels,predicted_Labels,average='macro')
weightedF1 = f1_score(test_Labels,predicted_Labels,average='weighted')
averageF1 = f1_score(test_Labels,predicted_Labels,average='macro')
conf_matrix = confusion_matrix(test_Labels,predicted_Labels)
```

Results



```
Average F1-score = 0.3936651917217737
Test Recall = 39.6
Test accuracy = 39.47
Weighted F1-score = 0.3935419108319004

Confusion Matrix:
[[20  0  1 ...  0  0  1]
 [ 0 33  2 ...  1  0  0]
 [ 0  7  6 ...  0  0  0]
 ...
 [ 0  0  4 ...  8  0  0]
 [ 0  0  0 ...  0 20  0]
 [ 1  1  1 ...  2  0 14]]
```