DataAnalysis_HrridayWork

January 24, 2024

1 Human Activity Recognition (Mini Project)

1.1 Team: TensionFlow

1.2 Importing Libraries and Preprocessing

[2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
from latex import latexify, format_axes
import numpy as np
import tsfel
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn import tree
import graphviz
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
import seaborn as sns
from MakeDataset import *
%matplotlib inline
# Retina
%config InlineBackend.figure_format = 'retina'
```

Dataset Generated from `MakeDataset.py`

[ ]:
```
X_train,X_test,y_train,y_test
X_test,X_val,y_test,y_val
```

[4]:
```
X_train
```

[4]:
```
array([[[ 0.9736077 , -0.1844755 , -0.2821974 ],
        [ 0.9760866 , -0.1867793 , -0.2848794 ],
        [ 0.977865  , -0.191836  , -0.2891687 ],
        …,
        [ 0.9779202 , -0.1834941 , -0.2829651 ],
        [ 0.9796224 , -0.1832831 , -0.279844  ],
        [ 0.9775468 , -0.1833646 , -0.2764387 ]],
```

```
[[ 1.00564   , -0.1732591 , -0.2299191 ],
 [ 1.006267  , -0.1727248 , -0.2516695 ],
 [ 1.004331  , -0.1783138 , -0.2447012 ],
 …,
 [ 0.9963187 , -0.165975  , -0.2166365 ],
 [ 0.998345  , -0.1662256 , -0.2176124 ],
 [ 1.00105   , -0.1642913 , -0.2210956 ]],

[[ 0.784794  , -0.2597323 , -0.2317497 ],
 [ 0.8028195 , -0.2151319 , -0.2276441 ],
 [ 0.7250539 , -0.2064177 , -0.2095281 ],
 …,
 [ 0.6540971 , -0.140727  , -0.2860766 ],
 [ 0.6268603 , -0.2748843 , -0.2455943 ],
 [ 0.6052588 , -0.3292142 , -0.1952567 ]],

…,

[[ 1.013856  , -0.08463204, -0.1833906 ],
 [ 1.018295  , -0.08470217, -0.1755404 ],
 [ 1.017008  , -0.08552211, -0.1765002 ],
 …,
 [ 1.009988  , -0.09727326, -0.1556776 ],
 [ 1.009527  , -0.1015205 , -0.1545634 ],
 [ 1.012576  , -0.1036466 , -0.1521443 ]],

[[ 1.479893  , -0.4783501 , -0.1348317 ],
 [ 1.417838  , -0.5264301 , -0.02401967],
 [ 1.12661   , -0.4468493 , -0.05762023],
 …,
 [ 0.8351439 , -0.1607123 , -0.1592067 ],
 [ 0.7713394 , -0.1248221 , -0.1388356 ],
 [ 0.7175624 , -0.1262066 , -0.1470366 ]],

[[ 0.7170605 , -0.02063687, -0.1085871 ],
 [ 0.7705297 , -0.0618509 , -0.1241441 ],
 [ 0.7802221 , -0.0469533 , -0.1191337 ],
 …,
 [ 0.7315363 , -0.1621981 , -0.04988996],
 [ 0.7622148 , -0.1765388 , -0.03800902],
 [ 0.7644377 , -0.2050919 , -0.02824738]]])
```

Extract a_x, a_y, a_z from `X_train`

[5]:
```
aXYZ_Xtrain = X_train[:, :, 0], X_train[:, :, 1], X_train[:, :, 2]
```

Get Total Acceleration $(a_x^2 + a_y^2 + a_z^2)$ Time Series from `X_train`, `X_test`, `X_val`

[6]:
```
X_train_TS = np.sum(np.square(X_train), axis = -1)
X_test_TS = np.sum(np.square(X_test), axis = -1)
X_val_TS = np.sum(np.square(X_val), axis = -1)
```

[7]:
```
print(X_train_TS.shape, X_test_TS.shape, X_val_TS.shape)
```

```
(108, 500) (36, 500) (36, 500)
```

[8]:
```
y_train
```

[8]:
```
array([5, 5, 2, 3, 6, 4, 1, 5, 5, 4, 3, 1, 6, 5, 4, 1, 5, 1, 4, 6, 2, 4,
       6, 3, 6, 1, 1, 6, 6, 2, 5, 1, 4, 2, 6, 4, 3, 3, 6, 4, 2, 3, 6, 3,
       5, 5, 3, 1, 3, 1, 4, 6, 4, 5, 4, 4, 3, 4, 2, 4, 2, 1, 2, 5, 4, 2,
       5, 2, 3, 6, 1, 3, 2, 2, 1, 3, 6, 2, 5, 1, 3, 2, 4, 5, 4, 2, 1, 1,
       1, 3, 5, 5, 6, 1, 4, 6, 6, 5, 3, 3, 2, 6, 6, 3, 1, 5, 2, 2])
```

The Sort Order

[9]:
```
df = pd.DataFrame(X_train_TS)
df["Label"] = y_train
df.sort_values(by = "Label", inplace = True)
S = np.array(df.index)
S
```

[9]:
```
array([ 86,  26,  25,  87,  93,  47,  49,  17,  15,  31,  61,  88,  70,
        74,  79,   6,  11, 104,  33,  73,  72,  40,  62, 106,  58,  67,
        65,  81,  85,  60, 107,  77,   2, 100,  29,  20,  75,   3,  71,
       103,  68,  80,  10,  99,  98,  56,  48,  89,  41,  36,  37,  43,
        23,  46,  32,  35,   5,  39,   9,  64,  21,  54,  59,  57,  55,
        84,  18,  52,  82,  50,  94,  14,  97,  78,  91,  90, 105,  83,
         0,  53,  66,  63,   1,  45,  44,   7,  30,   8,  13,  16,   4,
       102, 101,  12,  96,  69,  19,  76,  92,  24,  27,  28,  34,  38,
        42,  51,  95,  22], dtype=int64)
```

[10]:
```
df
```

[10]:

	0	1	2	3	4	5	6	\
86	0.714053	0.732246	0.768960	0.853281	0.950569	1.001927	1.025067	
26	0.751970	0.651734	0.579656	0.587043	0.606218	0.727218	0.797291	
25	0.596975	0.577279	0.564058	0.773096	0.871679	1.057734	1.427447	
87	1.447083	1.243813	0.999385	0.811583	0.683452	0.727751	0.759291	
93	0.678714	0.700478	1.148305	2.166170	2.921367	2.662965	1.954290	
..	…	…	…	…	…	…	…	
38	0.996194	0.991963	0.984747	0.987995	0.992127	0.994089	0.995440	
42	1.030783	1.005243	0.994559	0.997765	1.005931	1.004629	0.970352	
51	1.017841	1.016995	1.024412	1.029757	1.026955	1.028478	1.026419	
95	1.003037	1.000143	1.002048	0.993890	0.997013	1.012518	1.025586	
22	1.027639	1.033972	1.033586	1.024781	1.022987	1.022073	1.025497	

```
            7         8         9  …       491       492       493       494  \
86   1.039051  1.033828  1.021974  …  1.477209  1.245564  0.884933  0.751383
26   0.794672  0.834353  0.888909  …  0.396283  0.573048  0.702095  0.827120
25   1.728494  1.920396  1.673412  …  0.961942  1.053653  1.100745  1.275938
87   0.818850  0.947241  0.960450  …  1.395899  1.404201  1.532946  2.060462
93   1.440979  1.186594  1.258637  …  1.112212  1.326475  1.516445  1.761912
..        …         …         …  …         …         …         …         …
38   0.998901  1.003637  1.002344  …  1.010409  1.008286  1.004564  1.001250
42   0.929467  0.959656  1.012440  …  0.978252  1.000408  1.027792  1.028454
51   1.026562  1.034466  1.037380  …  1.035634  1.040731  1.038952  1.027263
95   1.026735  1.021032  1.016388  …  1.010742  1.016365  1.022836  1.018828
22   1.025356  1.019591  1.021240  …  1.031073  1.032912  1.032843  1.028517

          495       496       497       498       499  Label
86   0.895341  1.111554  1.159284  1.061964  0.962311      1
26   0.867516  0.805562  0.813949  0.771488  0.791860      1
25   1.522039  1.531372  1.548665  1.541446  1.412323      1
87   2.916117  3.367976  2.650901  1.300202  0.516547      1
93   2.096122  1.985628  1.334452  0.939196  0.868019      1
..        …         …         …         …         …    …
38   0.997999  0.996779  0.991216  0.983254  0.985567      6
42   1.006944  0.987428  0.983824  0.990666  0.990392      6
51   1.011913  1.013323  1.020733  1.019209  1.022797      6
95   1.014693  1.025160  1.033535  1.029625  1.023284      6
22   1.027242  1.030718  1.032342  1.033870  1.031274      6

[108 rows x 501 columns]
```

Sorting the a_x, a_y, a_z also as according to the Label sort index

[11]:
```
aXYZ_Xtrain = aXYZ_Xtrain[0][S], aXYZ_Xtrain[1][S], aXYZ_Xtrain[2][S]
```

Sort the total acceleration $(a_x^2 + a_y^2 + a_z^2)$ according to label sort index and get the FINAL TIME SERIES for all test data subjects 108

[12]:
```
df_Xtrain = pd.DataFrame(X_train_TS)
df_Xtrain["Label"] = y_train
df_Xtrain.sort_values(by = "Label", inplace = True)
df_Xtrain.set_index(pd.Series(range(108)), inplace = True)
df_Xtrain
```

[12]:
```
           0         1         2         3         4         5         6  \
0   0.714053  0.732246  0.768960  0.853281  0.950569  1.001927  1.025067
1   0.751970  0.651734  0.579656  0.587043  0.606218  0.727218  0.797291
2   0.596975  0.577279  0.564058  0.773096  0.871679  1.057734  1.427447
3   1.447083  1.243813  0.999385  0.811583  0.683452  0.727751  0.759291
4   0.678714  0.700478  1.148305  2.166170  2.921367  2.662965  1.954290
..       …         …         …         …         …         …         …
```

```
103  0.996194  0.991963  0.984747  0.987995  0.992127  0.994089  0.995440
104  1.030783  1.005243  0.994559  0.997765  1.005931  1.004629  0.970352
105  1.017841  1.016995  1.024412  1.029757  1.026955  1.028478  1.026419
106  1.003037  1.000143  1.002048  0.993890  0.997013  1.012518  1.025586
107  1.027639  1.033972  1.033586  1.024781  1.022987  1.022073  1.025497

            7         8         9  …       491       492       493  \
0    1.039051  1.033828  1.021974  …  1.477209  1.245564  0.884933
1    0.794672  0.834353  0.888909  …  0.396283  0.573048  0.702095
2    1.728494  1.920396  1.673412  …  0.961942  1.053653  1.100745
3    0.818850  0.947241  0.960450  …  1.395899  1.404201  1.532946
4    1.440979  1.186594  1.258637  …  1.112212  1.326475  1.516445
..        …         …         …  …         …         …         …
103  0.998901  1.003637  1.002344  …  1.010409  1.008286  1.004564
104  0.929467  0.959656  1.012440  …  0.978252  1.000408  1.027792
105  1.026562  1.034466  1.037380  …  1.035634  1.040731  1.038952
106  1.026735  1.021032  1.016388  …  1.010742  1.016365  1.022836
107  1.025356  1.019591  1.021240  …  1.031073  1.032912  1.032843

          494       495       496       497       498       499  Label
0    0.751383  0.895341  1.111554  1.159284  1.061964  0.962311      1
1    0.827120  0.867516  0.805562  0.813949  0.771488  0.791860      1
2    1.275938  1.522039  1.531372  1.548665  1.541446  1.412323      1
3    2.060462  2.916117  3.367976  2.650901  1.300202  0.516547      1
4    1.761912  2.096122  1.985628  1.334452  0.939196  0.868019      1
..        …         …         …         …         …         …    …
103  1.001250  0.997999  0.996779  0.991216  0.983254  0.985567      6
104  1.028454  1.006944  0.987428  0.983824  0.990666  0.990392      6
105  1.027263  1.011913  1.013323  1.020733  1.019209  1.022797      6
106  1.018828  1.014693  1.025160  1.033535  1.029625  1.023284      6
107  1.028517  1.027242  1.030718  1.032342  1.033870  1.031274      6

[108 rows x 501 columns]
```

[13]:
```
df_Xtrain["Label"].value_counts()
```

[13]:
```
Label
1    18
2    18
3    18
4    18
5    18
6    18
Name: count, dtype: int64
```

[14]:
```
classes
```

[14]:
```
{'WALKING': 1,
 'WALKING_UPSTAIRS': 2,
 'WALKING_DOWNSTAIRS': 3,
 'SITTING': 4,
 'STANDING': 5,
 'LAYING': 6}
```

[15]:
```
classesN = {1 : 'WALKING', 2 : 'WALKING_UPSTAIRS', 3 : 'WALKING_DOWNSTAIRS', 4 :
 ↪ 'SITTING', 5 : 'STANDING', 6 : 'LAYING'}
classesN
```

[15]:
```
{1: 'WALKING',
 2: 'WALKING_UPSTAIRS',
 3: 'WALKING_DOWNSTAIRS',
 4: 'SITTING',
 5: 'STANDING',
 6: 'LAYING'}
```

1.3 Questions/Tasks

1.3.1 1. Plot the waveform for data from each activity class. Are you able to see any difference/similarities between the activities? You can plot a subplot having 6 columns to show differences/similarities between the activities. Do you think the model will be able to classify the activities based on the data? [1 mark]

Plot of each (a_x, a_y, a_z) for all subjects $ (a_x, a_y, a_z) = [Red, Green, Blue] $

[97]:
```
fig, axes = plt.subplots(18, 6, figsize = (250, 275))
fig.suptitle(r"Time Series of Acceleration $(acc_x, acc_y, acc_z)$", size = 240)
sortedY_train = np.array(df_Xtrain["Label"])
colors = ["red", "green", "blue"]
c = 0
for i in range(6):
    for j in range(18):
        for k in range(3):
            time_series = aXYZ_Xtrain[k][c]
            axes[j, i].plot(time_series, color = colors[k], linewidth = 3)
            axes[j, i].set_title(f"Subject {c + 1}, Class =␣
 ↪{classesN[sortedY_train[c]]}", fontsize = 90)
            axes[j, i].set_ylabel(r"Acceleration in $ms^{-2}$", fontsize = 60)
            axes[j, i].set_xlabel("Time Samples", fontsize = 60)
        c += 1


plt.subplots_adjust(wspace = 0.5, hspace = 0.5)
plt.savefig("Activity_Individual.png")
plt.close()
```

Answer: #### 1. From the above plots, one can easily differentiate between Static Activities

(like Laying, Sitting, Standing) and Dynamic Activities (Walking, Walking Downstairs, Walking Upstairs) as the Dynamic Activities are more volatile and seem to have greater variance as compared to the Static Activities.

2. Among subjects belonging to the same class, there is a certain degree of similarity in the trends of time series data.

3. The model may be capable of classifying the activites into Static and Dynamic, though further classification may depend on many factors like model hyperparameters, feature matrix, etc.

1.3.2 2. Do you think we need a machine learning model to differentiate between static activities (laying, sitting, standing) and dynamic activities(walking, walking_downstairs, walking_upstairs)? Look at the linear acceleration $(acc_x^2+acc_y^2+acc_z^2)$ for each activity and justify your answer. [1 mark]

[98]:
```
latexify()
fig, axes = plt.subplots(18, 6, figsize = (250, 275))
fig.suptitle(r"Time Series of Total Acceleration $(acc_x^2 + acc_y^2 +␣
 ↪acc_z^2)$", size = 240)
sortedY_train = np.array(df_Xtrain["Label"])
colors = ["red", "deeppink", "purple", "teal", "green", "blue"]
c = 0
for i in range(6):
    for j in range(18):
        time_series = df_Xtrain.iloc[c, : -1]
        axes[j, i].plot(time_series, color = colors[i], linewidth = 4)
        axes[j, i].set_title(f"Subject {c + 1}, Class =␣
 ↪{classesN[sortedY_train[c]]}", fontsize = 90)
        axes[j, i].set_ylabel(r"Total Acceleration in  $m^{2}s^{-4}$", fontsize␣
 ↪= 60)
        axes[j, i].set_xlabel("Time Samples", fontsize = 60)
        c += 1

plt.subplots_adjust(wspace = 0.5, hspace = 0.5)
plt.savefig("Activity_Tot.png")
plt.close()
```

Answer: #### 1. For differentiating between Static and Dynamic Activities, implementation of a machine learning model may not be necessary but it may be a sufficient condition.

2. One can always extract time series features from the samples of activities and draw meaningful inferences out of it.

3. A more automated way of the same would be to implement a machine learning model that learns the distribution of data.

4. A simple machine learning model like a Decision Tree Classifier might work well in classifying between Static and Dynamic activites as the Dynamic Activities are more volatile and seem to have greater variance as compared to the Static Activities.

1.3.3 3. Train Decision Tree using trainset and report Accuracy and confusion matrix using testset. [1 mark]

Raw Time Series for 108 Training Subjects and 500 features/samples

[16]:
```
model = DecisionTreeClassifier(random_state=42)
clfg = model.fit(X_train_TS, y_train)
y_pred = clfg.predict(X_test_TS)
y_pred
```

[16]:
```
array([6, 1, 6, 1, 6, 5, 6, 1, 1, 1, 4, 6, 6, 5, 2, 6, 5, 5, 3, 6, 4, 1,
       4, 5, 2, 1, 3, 6, 1, 4, 6, 6, 6, 1, 1, 3])
```

Classification Report

Accuracy $= \frac{||y=\hat{y}||}{||y||}$

Precision $= \frac{||y=\hat{y}=\text{Class}||}{||\hat{y}=\text{Class}||} = \frac{\text{T.P.}}{\text{T.P.+F.P.}}$

Recall $= \frac{||y=\hat{y}=\text{Class}||}{||y=\text{Class}||} = \frac{\text{T.P.}}{\text{T.P.+F.N.}}$

F-Score $= \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision+Recall}}$

[17]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.40      0.67      0.50         6
           2       0.50      0.17      0.25         6
           3       0.33      0.17      0.22         6
           4       0.50      0.33      0.40         6
           5       1.00      0.83      0.91         6
           6       0.50      1.00      0.67         6

    accuracy                           0.53        36
   macro avg       0.54      0.53      0.49        36
weighted avg       0.54      0.53      0.49        36
```

Confusion Matrix

[18]:
```
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index = [classT for classT in classes], columns =␣
↪[classT for classT in classes])
```

Confusion Matrix Plot Handler

[19]:
```
## flag = 1 for a single plot and 0 for subplots for 2 - 8 depths
def confMatrix(dataFrame, flag = 1, accuracies = None):
    if flag:
        plt.figure(figsize = (15, 15))
        ax = sns.heatmap(dataFrame, annot = True, cmap = "PuBu")
        plt.setp(ax.get_xticklabels(), rotation = 45, fontsize = 8)
        plt.setp(ax.get_yticklabels(), fontsize = 8)
        plt.ylabel("True label", fontsize = 18)
        plt.xlabel("Predicted label", fontsize = 18)
        plt.title(f"Accuracy = {accuracy_score(y_test, y_pred)*100: .4f}%",␣
 ↪fontweight = "bold", fontsize = 13)
        plt.savefig("Single_ConfusionM.png")
        plt.show()
    else:
        fig, axes = plt.subplots(3, 3, figsize = (25, 25))
        axes = axes.flatten()

        for i, df in enumerate(dataFrame):
            ax = sns.heatmap(df, annot = True, ax = axes[i], cbar = False, cmap␣
 ↪= "PuBu")

            plt.setp(ax.get_xticklabels(), rotation = 45, fontsize = 6)
            plt.setp(ax.get_yticklabels(), fontsize = 8)
            ax.set_title(f"Depth = {i + 2}\nAccuracy = {accuracies[i] * 100: .
 ↪4f}%", fontsize = 10)
            ax.set_ylabel("True label", fontsize = 12)
            ax.set_xlabel("Predicted label", fontsize = 12)

        plt.delaxes(axes[7])
        plt.delaxes(axes[8])
        plt.tight_layout()
        plt.subplots_adjust(wspace = 1.1, hspace = 1.1)
        plt.savefig("DepthConfusionM.png")
        plt.show()
```

Confusion Matrix Image

[ ]:
```
confMatrix(df_cm, 1)
```

Answer: #### 1. The overall performance of the Decision Tree Classifier model trained on the raw total acceleration $(acc_x^2 + acc_y^2 + acc_z^2)$ dataset was on average barely better than a fair coin toss.

2. The model performed decently well in classifying between Static and Dynamic Activities.

3. Classification among the Static Activities:

- The model performed the best in classifying the activity STANDING, while it mispredicted LAYING as SITTING.
- On the contrary, classifications among the Dynamic Activities showed confusion.

4. NOTE: The above observations are specific to a single random model and cannot be generalized for any model. Additionally, the number of samples available for training and testing are quite few hence, nothing much can be said about the distributions of activities.

1.3.4 4. Train Decision Tree with varying depths (2-8) using trainset and report accuracy and confusion matrix using Test set. Does the accuracy change when the depth is increased? Plot the accuracies and reason why such a result has been obtained. [1 mark]

Varying Tree Depths $(2-8)$ and looking at the results for each depth

[20]:
```
confusion_matrices, class_reports, class_reports_dict, accuracies = [], [], [],␣
↪[]
for i in range(2, 9):
    model = DecisionTreeClassifier(max_depth = i,random_state=42)
    clfg = model.fit(X_train_TS, y_train)
    y_pred = clfg.predict(X_test_TS)

    pred, actual = y_pred, y_test

    cm = confusion_matrix(actual, pred)

    confusion_matrices.append(pd.DataFrame(cm, index = [classT for classT in␣
↪classes], columns = [classT for classT in classes]))
    class_reports.append(classification_report(actual, pred, labels = np.
↪unique(pred)))
    class_reports_dict.append(classification_report(actual, pred, labels = np.
↪unique(pred), output_dict = True))
    accuracies.append(accuracy_score(actual, pred))
```

Confusion Matrix Image (for varying tree depths)

[ ]:
```
confMatrix(confusion_matrices, 0, accuracies = accuracies)
```

Classification reports

[21]:
```
for i in range(2,9):
    print(f'Depth = {i}\n{class_reports[i-2]}\n\n')
```

```
Depth = 2
              precision    recall  f1-score   support

           1       0.38      0.50      0.43         6
           3       0.50      0.17      0.25         6
```

5	0.75	1.00	0.86	6
6	0.33	1.00	0.50	6
micro avg	0.44	0.67	0.53	24
macro avg	0.49	0.67	0.51	24
weighted avg	0.49	0.67	0.51	24

Depth = 3

	precision	recall	f1-score	support
1	0.50	0.50	0.50	6
2	1.00	0.17	0.29	6
3	0.50	0.33	0.40	6
4	0.29	0.33	0.31	6
5	0.86	1.00	0.92	6
6	0.55	1.00	0.71	6
accuracy			0.56	36
macro avg	0.61	0.56	0.52	36
weighted avg	0.61	0.56	0.52	36

Depth = 4

	precision	recall	f1-score	support
1	0.50	0.67	0.57	6
2	0.50	0.17	0.25	6
3	0.33	0.17	0.22	6
4	0.29	0.33	0.31	6
5	1.00	0.83	0.91	6
6	0.55	1.00	0.71	6
accuracy			0.53	36
macro avg	0.53	0.53	0.49	36
weighted avg	0.53	0.53	0.49	36

Depth = 5

	precision	recall	f1-score	support
1	0.45	0.83	0.59	6
2	0.50	0.17	0.25	6
3	0.50	0.17	0.25	6
4	0.40	0.33	0.36	6

```
              5       1.00      0.83      0.91         6
              6       0.55      1.00      0.71         6

       accuracy                           0.56        36
      macro avg       0.57      0.56      0.51        36
   weighted avg       0.57      0.56      0.51        36
```

```
Depth = 6
              precision    recall  f1-score   support

              1       0.56      0.83      0.67         6
              2       0.33      0.17      0.22         6
              3       0.50      0.17      0.25         6
              4       0.50      0.33      0.40         6
              5       0.83      0.83      0.83         6
              6       0.50      1.00      0.67         6

       accuracy                           0.56        36
      macro avg       0.54      0.56      0.51        36
   weighted avg       0.54      0.56      0.51        36
```

```
Depth = 7
              precision    recall  f1-score   support

              1       0.40      0.67      0.50         6
              2       0.50      0.17      0.25         6
              3       0.33      0.17      0.22         6
              4       0.50      0.33      0.40         6
              5       1.00      0.83      0.91         6
              6       0.50      1.00      0.67         6

       accuracy                           0.53        36
      macro avg       0.54      0.53      0.49        36
   weighted avg       0.54      0.53      0.49        36
```

```
Depth = 8
              precision    recall  f1-score   support

              1       0.40      0.67      0.50         6
              2       0.50      0.17      0.25         6
              3       0.33      0.17      0.22         6
              4       0.50      0.33      0.40         6
```

```
           5       1.00      0.83      0.91         6
           6       0.50      1.00      0.67         6

    accuracy                           0.53        36
   macro avg       0.54      0.53      0.49        36
weighted avg       0.54      0.53      0.49        36
```

Answer: #### 1. From the classification reports, one can infer that the accuracy of classifying activities is in general increasing.

2. At Depth $= 2$, the model seems to underfit the data which is evident from its large number of misclassifications of Dynamic Activities as Static ones and classification of only a few number of activities (model did not make a single prediction for acivities WALKING UPSTAIRS and SITTING).

3. The accuracy peaks at Depth $= 5$ and Depth $= 6$ equivalent to an expected peak in accuracy at higher depths.

4. ***NOTE: The above observations are specific to a selected random model and cannot be generalized for any model. Additionally, the number of samples available for training and testing are quite few hence, nothing much can be said about the distributions of activities.***

1.3.5 5. Use PCA (Principal Component Analysis) on Total Acceleration $(acc_x^2 + acc_y^2 + acc_z^2)$ to compress the acceleration timeseries into two features and plot a scatter plot to visualize different class of activities. Next, use TSFEL (a featurizer library) to create features (your choice which ones you feel are useful) and then perform PCA to obtain two features. Plot a scatter plot to visualize different class of activities. Are you able to see any difference? [2 marks]

2 REFER TO `FeaturesExtr.ipynb` for further