Name: Trung Kien Nguyen

Student ID: 104053642

Studio: 1 - 3

PORTFOLIO REPORT – WEEK 3

Studio 3

For this studio, I have used Jupiter Notebook instead of merely Python file to implement the solution. This is for easier code management and execution, as I don't have to re-execute all the sollution for each activity from 2 to 5. Here is the link ro my solution:

https://colab.research.google.com/drive/1VAsLZw93XLWyyAW-fvXAiyv_vKKAt6IK#scrollTo=SEbnURRJnRg1

The results for the activities from 2 to 5, which have been presented in the Notebook, are also summerised in the following table:

Activity 6 - Summary Table

SVM model	Train-Test split	Test Accuracy	Cross validation (mean)
Original features	70 – 30	88.93%	89.18%
With hyper parameter tuning		89.71%	90.29%
With feature selection and hype parameter tuning		90.11%	89.59%
With PCA and hyper parameter tuning		89.33%	90.04%

Activity 7 - Other classifiers

For this activity, I will choose the best-performed SVM model among the four above scenarios, which is ***using feature selection and hyperparameter tuning***, and compare it with the other 3 classifiers, including SGD (Stochastic Gradient Descent), RandomForest, and MLP (Multi-layer Perceptron). The implementation is also demonstrated in the above Notebook. Here is the summary of the results:

Model	Train-Test split	Test Accuracy	Cross Validation
SVM	70 – 30	90.11%	89.59%
SGD		87.85%	86.93%
RandomForest		92%	92.6%
MLP		87.41%	85.86%

Portfolio Assessment

Here is the link of code (Notebook) for my solution of this week portfolio:
https://colab.research.google.com/drive/1C9hU4yxXyajywaxhhaIQxFLb5mFMRkEq#scrollTo=Kqe6OD8Yz-Bz

Data collection

For this task, I have implemented a general solution for all cases first, then used the output to extract the data specifically for student number ending with 2. Here is the breakdown:

- Column Mappings: A dictionary (named column_mappings) that associate each student number ending (from 0 to 9) with the respective columns for sets 1 and 2, based on the given table

Student number	**Column set 1**	**Column set 2**
Ending with 0	Neck (x,y,z)	Head (x,y,z)
Ending with 1	Right Shoulder (x,y,z)	Left Shoulder (x,y,z)
Ending with 2	Right Upper Arm (x,y,z)	Left Upper Arm (x,y,z)
Ending with 3	Right Forearm (x,y,z)	Left Forearm (x,y,z)
Ending with 4	Right Hand (x,y,z)	Left Hand (x,y,z)
Ending with 5	Right Upper Leg (x,y,z)	Left Upper Leg (x,y,z)
Ending with 6	Right Lower Leg (x,y,z)	Left Lower Leg (x,y,z)
Ending with 7	Right Foot (x,y,z)	Left Foot (x,y,z)
Ending with 8	Right Toe (x,y,z)	Left Toe (x,y,z)
Ending with 9	L5 (x,y,z)	T12 (x,y,z)

- I have written a method `process_data` that extracts the necessary columns from a DataFrame based on the mappings and labels the data with the appropriate class (0 for boning, 1 for slicing).
- After processing each file with the function, the results are concatenated into one DataFrame:
    - For each record, the data columns corresponding to the student ending in the map will be saved, while other columns will be left blank, for example:

Frame	Neck x	Neck y	Neck z	Head x	Head y	Head z	Class	Student_E	Right Shou	Right Shou	Right Shou	Left Shoul	Left Shoul	Left Shoul	Right Uppe	Right Uppe	Right Uppe	Left Upper	Left Upper	Left Upper	Right Fore	Right Fore
0	-0.00133	0.035814	-0.07154	0.047955	0.197625	-0.04884	0	0														
1	-0.00068	0.084804	0.029121	-0.00536	0.004459	0.010107	0	0														
2	-0.14315	-0.09257	-0.03273	-0.16855	-0.20063	-0.03668	0	0														
3	-0.15984	-0.00933	-0.01837	-0.20394	-0.01113	-0.0894	0	0														
4	-0.11404	-0.24144	-0.03757	-0.02104	0.112576	-0.0587	0	0														
5	-0.17314	0.130549	-0.13202	-0.1606	0.17513	-0.23592	0	0														
6	-0.15332	0.020864	-0.15757	-0.40855	-0.02053	-0.32812	0	0														
7	-0.21917	0.086974	-0.17937	-0.45806	-0.14747	-0.31705	0	0														
8	-0.14349	0.067465	-0.07543	-0.39198	-0.12509	-0.19165	0	0														
9	-0.11366	-0.02797	0.008837	-0.14354	-0.0424	-0.05102	0	0														
10	-0.04273	0.093494	-0.06014	0.088242	0.057897	-0.06041	0	0														
11	-0.04481	0.066023	-0.11343	0.012558	0.240093	-0.17988	0	0														
12	-0.17287	0.000692	-0.11797	-0.28164	-0.01352	-0.24585	0	0														
13	-0.21124	-0.14282	-0.09387	-0.33508	-0.12396	-0.13447	0	0														
14	0.110735	0.249377	0.262505	-0.1704	-0.12019	-0.04263	0	0														
15	0.069515	-0.06582	0.050216	-0.26725	-0.35717	-0.06417	0	0														
16	-0.44234	-0.51793	-0.23744	-0.24093	-0.33812	-0.13144	0	0														
17	-0.15814	0.129805	-0.06906	-0.2463	0.023692	-0.05842	0	0														

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
15610							0	2							0.986122	0.604796	0.740596	1.957207	-1.4131	2.11249
15611							0	2							1.822856	0.918199	0.375161	1.730608	-6.54405	-0.90272
15612							0	2							2.027285	0.598892	-0.46131	2.593951	-6.71811	-1.43813
15613							0	2							1.321751	0.230004	-0.81362	2.338797	-3.46401	0.706981
15614							0	2							0.653908	0.981802	-0.50653	2.979766	-0.90867	1.298744
15615							0	2							0.417266	0.975361	-0.36461	0.149858	-0.00835	1.088408
15616							0	2							0.258073	0.270221	-0.15644	-0.94787	0.843691	0.857246
15617							0	2							0.137129	0.153194	0.042006	-1.46743	-0.00797	0.654815
15618							0	2							0.114904	-0.10607	-0.22943	0.150182	-0.50434	0.951611
15619							0	2							-0.32706	0.178194	-0.31119	0.770595	-0.19482	0.494014
15620							0	2							-0.59508	0.069754	-0.28919	-0.81032	1.008281	0.482856
15621							0	2							-0.35569	0.179117	-0.44798	-0.43248	0.43852	0.879252
15622							0	2							-0.51132	0.072743	-0.60415	0.937412	0.548928	0.250651
15623							0	2							-1.19756	0.286543	-0.56302	-0.48337	0.770359	0.415181
15624							0	2							-1.57338	0.68244	-0.51991	-0.88572	1.62431	0.026449
15625							0	2							-1.75273	0.88081	-0.14919	-1.11206	1.579566	-0.73228
15626							0	2							-1.68224	0.821792	0.012357	-1.61851	1.202077	-1.30901
15627							0	2							-1.36103	0.243484	-0.36901	-0.78149	0.529934	-1.38917
15628							0	2							-0.98179	0.207507	-0.76776	-0.48453	0.444257	-1.63438
15629							0	2							-0.78766	0.303962	-0.85086	-0.06195	1.123867	-1.54302
15630							0	2							-0.79599	-0.07859	-0.76024	-0.06469	1.254087	-1.13581
15631							0	2							-1.20171	-0.08797	-0.44533	-0.65001	0.948728	-0.90454
15632							0	2							-1.15485	-0.02824	-0.3113	-0.35859	0.996924	-0.97368
15633							0	2							-0.92872	-0.02029	0.05542	-0.313	0.644856	-0.47387
15634							0	2							-0.72229	-0.17502	0.487535	-0.62092	0.646737	0.177183
15635							0	2							-0.49259	-0.26552	0.727104	-0.61844	1.124612	0.845213
15636							0	2							-0.40578	-0.4433	0.958636	-0.60369	1.028396	1.144576

    - Link for this data: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%203/ampc2/combined_data.csv

- After combining all data, I filters out entries specifically for student number ending with **2**. Here is the output of that 8-columns dataframe:

```
---------------------------------------------------------------------------
Specific Data with student number ending with 2:
       Frame  Right Upper Arm x  Right Upper Arm y  Right Upper Arm z  \
35760      0          -0.081934          -0.063509          -0.194105
35761      1          -0.017001           0.060680          -0.165873
35762      2          -0.097286           0.002338          -0.117991
35763      3          -0.150787          -0.041678          -0.051735
35764      4          -0.180658          -0.111853          -0.084678

       Left Upper Arm x  Left Upper Arm y  Left Upper Arm z  Class
35760          0.029982         -0.124462          0.040935      0
35761          0.067401         -0.042730          0.058972      0
35762          0.067550         -0.074310          0.094963      0
35763          0.075417         -0.134344          0.106930      0
35764         -0.000695         -0.187848          0.029711      0
```

  - Link for this data: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%203/ampc2/specific_data_ending_2.csv

  - Note that, if you want to test with other student ending, just reassign the value of variable `ending_number`:

```
# Extract specific data for student number ending with 2
ending_number = 2

columns = ['Frame'] + column_mappings[ending_number][0] + column_mappings[ending_number][1] + ['Class']

specific_data_ending_2 = combined_data[combined_data['Student Ending'] == ending_number][columns]
```

Composite columns

To do this, I have implemented a method calle `calculate_composites` that computes several composite metrics, including root mean square values for various combinations of x, y, and z coordinates, and angles like roll and pitch based on the motion data. Again, this method can also be used for other student ending, you only need to change the value of `ending_number`

Here is the output for this task:

```python
# Display the first few rows of the updated dataset
print(data.head())
```

```
       Frame  Right Upper Arm x  Right Upper Arm y  Right Upper Arm z  \
35760      0          -0.081934          -0.063509          -0.194105
35761      1          -0.017001           0.060680          -0.165873
35762      2          -0.097286           0.002338          -0.117991
35763      3          -0.150787          -0.041678          -0.051735
35764      4          -0.180658          -0.111853          -0.084678

       Left Upper Arm x  Left Upper Arm y  Left Upper Arm z  Class  \
35760          0.029982         -0.124462          0.040935      0
35761          0.067401         -0.042730          0.058972      0
35762          0.067550         -0.074310          0.094963      0
35763          0.075417         -0.134344          0.106930      0
35764         -0.000695         -0.187848          0.029711      0

       RMS_xy_Set1  RMS_yz_Set1  RMS_zx_Set1  RMS_xyz_Set1  Roll_Set1  \
35760     0.073302     0.144413     0.148980      0.127048 -16.774583
35761     0.044559     0.124892     0.117904      0.102445  19.997177
35762     0.068812     0.083449     0.108135      0.088302   0.875834
35763     0.110620     0.046976     0.112723      0.095132 -14.651614
35764     0.150247     0.099200     0.141081      0.132059 -29.275471

       Pitch_Set1  RMS_xy_Set2  RMS_yz_Set2  RMS_zx_Set2  RMS_xyz_Set2  \
35760  -21.859792     0.090526     0.092646     0.035879      0.077600
35761   -5.498094     0.056430     0.051496     0.063327      0.057290
35762  -39.500925     0.071010     0.085264     0.082404      0.079797
35763  -66.222307     0.108940     0.121413     0.092525      0.108274
35764  -52.168934     0.132829     0.134480     0.021014      0.109803

       Roll_Set2  Pitch_Set2
35760 -67.820348   12.889182
35761 -25.506676   42.784608
35762 -32.523544   29.257648
35763 -45.754856   23.712318
35764 -81.009884   -0.209459
```

- Link to this data with compisited columns: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%203/ampc2/specific_data_ending_2_with_composites.csv

Data pre-processing

To do this, I have done the following steps:

- Define the time segment: According to the requirement, the dataset must be segmented into 1-min intervals.
- Calcualte the features: For each one-min segment, I have computed the following features, as listed in the reuiqrements:
    - Mean: The average value of the data points in the segment, providing a central tendency.
    - Standard Deviation: Measures the amount of variation or dispersion of the data points.
    - Minimum (Min) and Maximum (Max) values in the datasets, respectively, offering insights into the range of the data.
    - Area Under the Curve is calculated using **Simpson's rule** for numerical integration. This feature sums the area under the data plot, which can be useful for understanding the overall magnitude of the data over time.
    - The number of peaks in the data, detected using the find_peaks function from scipy.signal. Peaks can indicate critical points in the data, such as local maxima which may represent significant events or changes.
- Finally, I have combine the features above to generate a new dataframe. Here is the outcome:

```
# Display the first few rows to verify
print(features_df.head())
```

```
   Right Upper Arm x_mean  Right Upper Arm x_std  Right Upper Arm x_min  \
0                0.014806               0.170411              -0.295758
1                0.029070               0.470305              -1.240959
2               -0.369741               0.833353              -2.499797
3                0.376306               1.276251              -2.957017
4               -0.200560               2.160764              -4.504275

   Right Upper Arm x_max  Right Upper Arm x_auc  Right Upper Arm x_peaks  \
0               0.559333               1.851829                       27
1               1.994361               3.562286                       26
2               2.301475             -44.694607                       24
3               4.087491              46.402433                       23
4               9.145538             -25.898538                       27

   Right Upper Arm y_mean  Right Upper Arm y_std  Right Upper Arm y_min  \
0               -0.008735               0.233200              -0.981862
1               -0.040950               0.454609              -1.038708
2                0.028117               0.846087              -1.904044
3               -0.195356               1.082777              -4.463040
4               -0.139719               2.143637              -9.498332

   Right Upper Arm y_max  ...  RMS_xyz_Set2_max  RMS_xyz_Set2_auc  \
0               0.634265  ...          0.464922         18.675353
1               1.294850  ...          2.797418         53.543908
2               2.472521  ...          5.147862         93.552176
3               3.035554  ...          4.220614         96.251327
4               5.575290  ...          3.286372        107.077628

   RMS_xyz_Set2_peaks  Roll_Set2_mean  Roll_Set2_std  Roll_Set2_min  \
0                  29      -14.055024      39.670749     -81.365020
1                  30       18.570610      37.856544     -65.754936
2                  27       -7.109488      45.856573     -81.871396
3                  33        8.773702      41.848828     -86.550990
4                  33      -15.120624      46.052369     -86.753884

   Roll_Set2_max  Roll_Set2_auc  Roll_Set2_peaks  Class
0      79.392873   -1602.957929               28      0
1      81.665061    2240.094702               33      0
2      80.827930    -857.106368               29      0
3      80.722469    1009.856165               29      0
4      82.622657   -1924.658115               24      0

[5 rows x 109 columns]
```

- Link to this dataframe: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%203/ampc2/processed_features_per_minute.csv

Training:

Different scenarios for SVM

For this task, I have implemented similarly to what I have done in Studio 3:

- The data was split into training (70%) and testing (30%) subsets.
- SVM:
  - 10-folds cross-validation is used to evaluate the model's stability and generalizability across the whole dataset.
  - Hyperparameter Tuning: I have used GridSearchCV to find optimal SVM parameter
  - Feature Selection. I have used SelectKBest to find the most 10 best features for the model
  - PCA for Dimensionality Reduction: I have used the class PCA from sklearn.decomposition to reduce the dataset to the top 10 principal components.
  - Here is the summary of outcome for SVM:

SVM model	Train-Test split	Test Accuracy	Cross validation (mean)
With hyper parameter tuning	70 - 30	66.93%	66.96%
With feature selection and hype parameter tuning		100%	100%
With PCA and hyper parameter tuning		66.78%	66.96%

To summary,

- With Hyperparameter Tuning: Achieving a test accuracy of approximately 66.93% and a very similar crossvalidation mean value indicate a consistent performance of the model across different subsets of the data. However, this rate may be far from a good accuracy in my opinion (I expected it was >=80%).
- With Feature Selection and Hyperparameter Tuning: A perfect score of 100% for both test accuracy and cross-validation mean is remarkable but might indicated **overfitting**. So, I would not use this result for the comparison to other models section.
- With PCA and Hyperparameter Tuning: The accuracy slightly dips compared to just hyperparameter tuning (the first scenario). This could suggest that the PCA might be discarding some important features that contribute positively to the model's predictive power. Nevertheless, overally the model is still a bit weak.

Because of the concern of Overfitting in Scenario 2 – Feature selection andHyperparameter tuning, I am not sure this is the best SVM model for the problem, despite its perfect accuracy. I believe the best option is the SVM with Hyperparameter Tuning Only. It offers a balanced performance, perhaps reflecting the genuine generalisation power of the model independent of any overfitting effects.

Compare with other classifiers

- SGD classifier: I have use the SGDClassifier class from sklearn.linear_model module
- RandomForest: I have used the class RandomForestClassifier from sklearn.ensemble module
- MLP classifier: I have used the MLPClassifie from sklearn.neural_network moduel
- Here is the summary of the best cases using SVM compared to other classifers

Model	Train-Test split	Test Accuracy	Cross Validation (mean value)
SVM	70 – 30	66.93%	66.96%
SGD		99.21%	96.23%
RandomForest		100%	100%
MLP		91.46%	96.23%

Although the statistics suggest that RandomForest is the best among the listed models, and this is also match with the example in Activity 7, I still have a concern of overfitting.

With a test accuracy of 99.21% and cross-validation mean of 96.23%, SGD is presented to perform well. Although, compared to the Random Forest model, it mays be less likely to be overfitting and more resilient, we should not ignore doubts about its almost perfect test accuracy/cross-validation mean.

The SVM models, except for With feature selection and hype parameter tuning, which is considered overfitting, appear to be quite weak when compared to other classifiers. Most likely, the solution to my data processing encountered a problem that was beyond my understanding. Anyway, based on the results I have gathered, I will temporarily say that SVM is the weakest of the four models.

Based on the above analysis, along with the fairly reliable accuracy rate/Cross Validation, I assume that the MLP (Multi-layer Perceptron) classifier is the most optimal one.