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% |

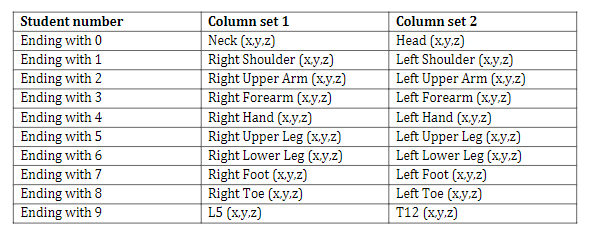
# Portfolio Assessment

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

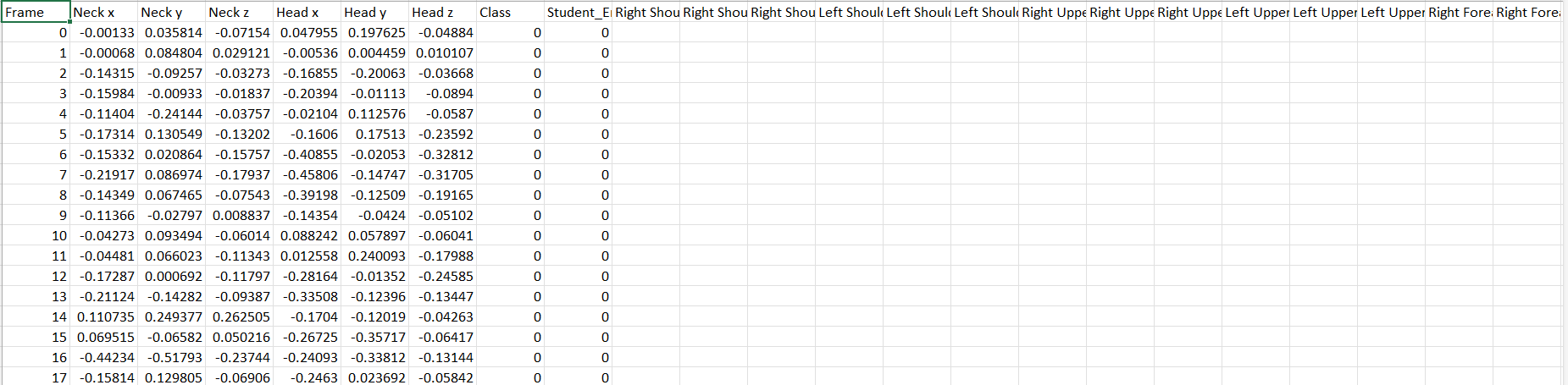
## Data collection

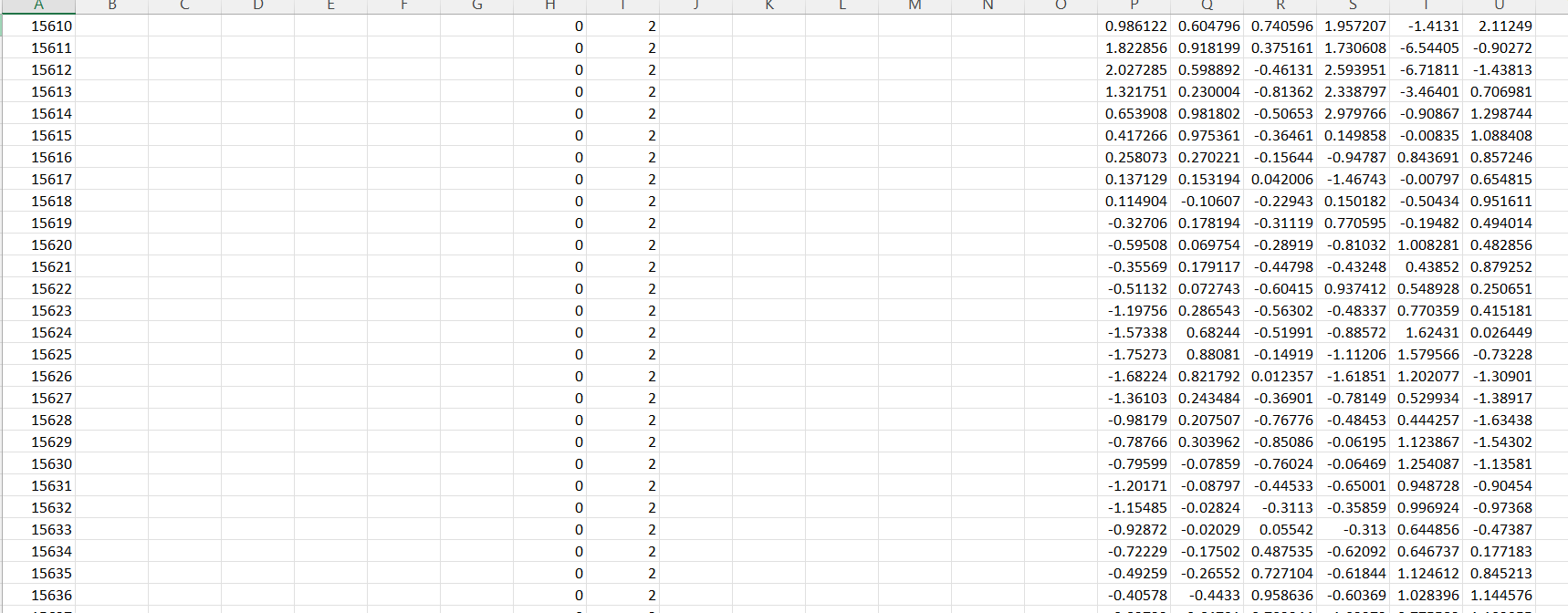
For this task, I have implemented a general solution for all cases first, then used the output to extract the data specifically for student number ending with 2. Here is the breakdown:

* 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

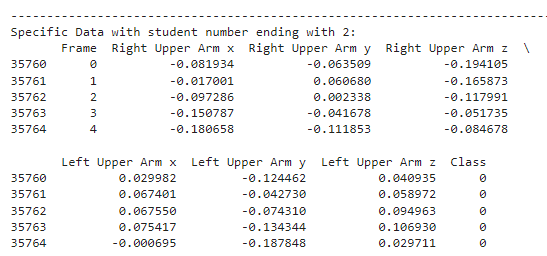


* I have written a method process\_data that extracts the necessary columns from a DataFrame based on the mappings and labels the data with the appropriate class (0 for boning, 1 for slicing).
* After processing each file with the function, the results are concatenated into one DataFrame:
  + For each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

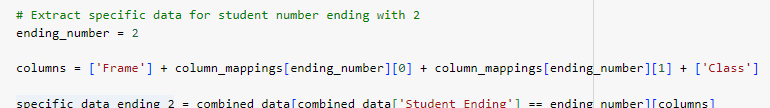




* + Link for this data: <https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%203/ampc2/combined_data.csv>
* After combining all data, I filters out entries specifically for student number ending with **2**. Here is the output of that 8-columns dataframe:



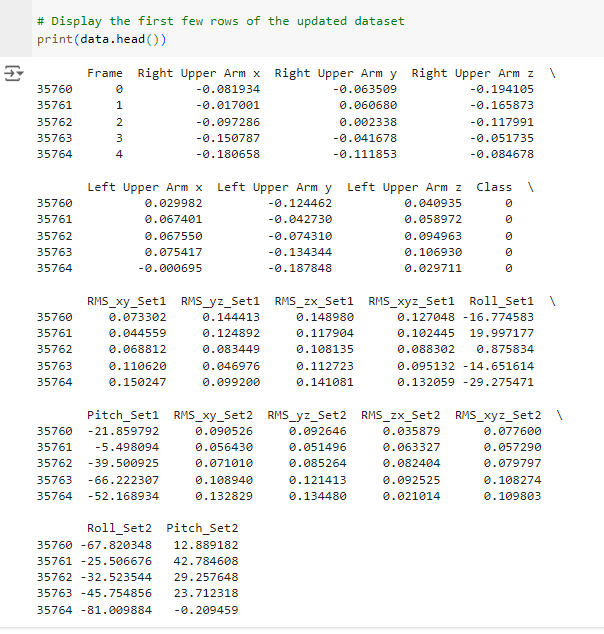
* + Link for this data: <https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%203/ampc2/specific_data_ending_2.csv>
  + Note that, if you want to test with other student ending, just reassign the value of variable ending\_number:



## 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:

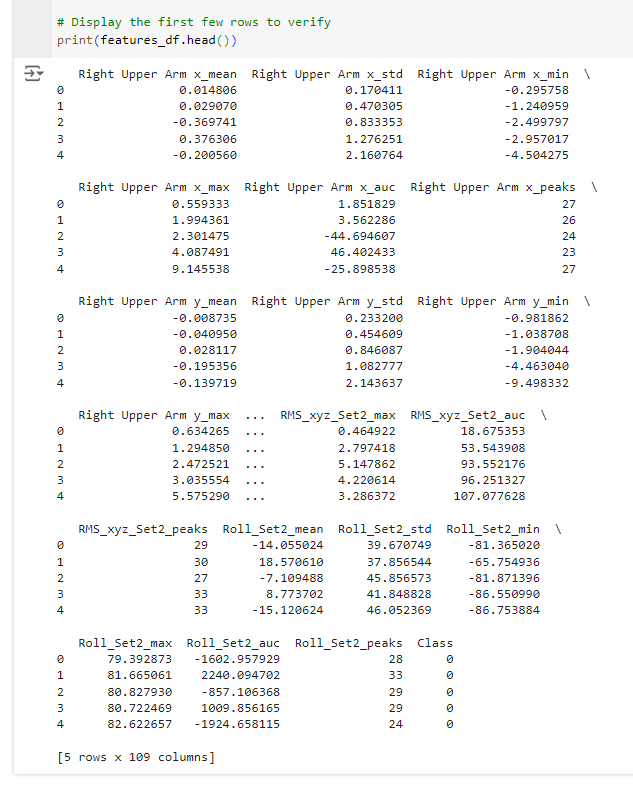


* + Link to this data with compisited columns: <https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%203/ampc2/specific_data_ending_2_with_composites.csv>

## 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:



* + 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.