Log File

Task: Create a system to authenticate users based on the way they perform touch interactions on their devices.

We were provided with a dataset with 4 users and 26 different features that were recorded for each user.

Problems we had:

- Some values were printed in console with an unnecessary number of decimal places. This made it harder to read. To resolve this we used round([number], [no. of dp]) to round these values to 3dp.
- Some values in the data set were missing which caused an error in the python script as the missing data was displayed as "NaN". To remove these we used dropna() in the format: dataset.dropna()
- As we used multiple metrics within each classifier, we had multiple return values that were similar (percentage accuracy and precision values). We decided to take an average of these values using a string assigned to a variable which was then printed in console.
- The code began to get harder to understand so we used comments and white space to allow us (and anyone else reading the script) to navigate around the script with ease.
- When tweaking the values of the number of columns that the script should read I found myself having to scroll up and down constantly to replace each value. To resolve this we added a variable called "limit" and assigned it a number. The variable was used within the script in multiple places to reference that limit number. This sped up the process of optimising the script.
- We originally reset y_pred to become a variable that represented the new predicted values that had been generated by that specific classifier as we had multiple classifiers because we would be inputting the same predicted values each time if we didn't reset them.

  However, after beginning to make another part of the script further on, we realised that it would be easier if we renamed each y_pred to y_pred1, y_pred2 and y_pred3 which would allow us to use each dataset from each variable separately.
- After assigning each predicted value sets for each classifier with a unique variable, we were able to use a "for" statement which used zip() to create a list of y_pred1, y_pred2, y_pred3 and y_test. We could assign each of these columns a variable (we used: a, b, c, d). This enabled us to use some basic python conditions which tested to see if a, b and c were equal to d. Each time it found for example, a equal to d, then it would add 1 to the counter (counter is a variable that we set to 0). It then sees if the score on the counter is >= 2 and if it is then it adds a point to another counter (variable) called "correct answers" which we defined OUTSIDE of the "for" loop so that it didn't get reset after each row. We could then take this value of number of correct answers to find a percentage (multiply by 100 then

divide by the number of rows (this can be found by using “len(y_test)”). We found that this percentage was less volatile than the other metrics from the other classifiers.

The large data set was over 10 times larger than the original data set.

Majority vote less volatile

CLASSIFIERS

k-nearest neighbors (KNN)

Support-vector machine (SVM)

Decision Tree (DT)

METRICS

balanced_accuracy_score

precision_score

PROGRAM

JetBrains