Computational Intelligence for the IoT

Project 1.1

Classification using NN

Professor:

João P. Carvalho

Students (G2) :

Aurora Nora | 93573

João Lopes | 93584

In this report we will explain how we developed two neural-network models with the purpose of knowing how many students are inside a room, based on data received from several sensors inside that specific classroom.

Initially, we analyzed the given dataset and the problems that we would have to solve in the development of the models.

We noticed that there were some features that weren't relevant for data evaluation ('Date', 'Time') so we removed them and used them as x-axis values. Also, we noticed that some rows in the dataset that had missing values and noisy values and treated them in different ways which are explained below:

- Missing values:
  - When detected, we inserted the last valid value of that feature
- Noisy values:
  - When detected an invalid value (value type is different from the one in the specification) deletes row
  - When detected a negative value (doesn't make sense in the given dataset) deletes row
  - When detected movement (PIRi = 1) and the room is empty deletes row

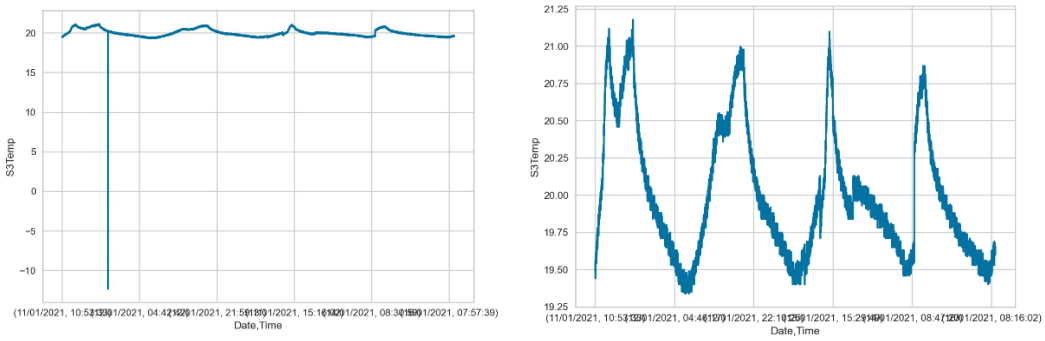


Figure 1: 'S3Temp' representation before (with negative value) and after Noisy Values correction.

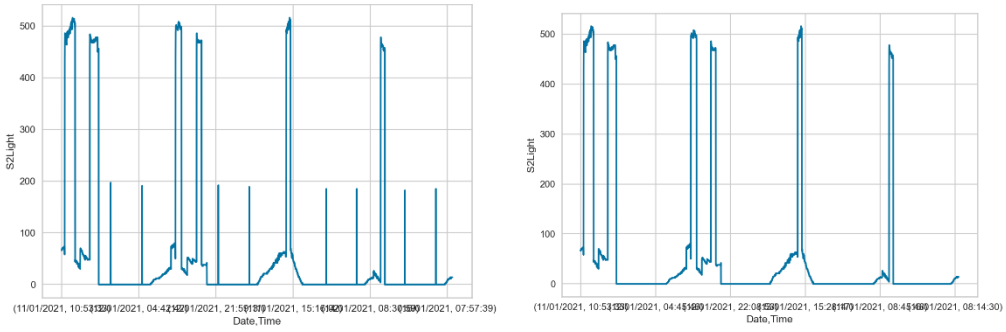


Figure 2: 'S2Light' before (empty room and movement detected) and after Noisy Values correction.

After finding and deleting the noisy values in the dataset we proceeded to find the outliers. For that we made a function that considers outliers a value that are included in the following conditions:

- Value < Q1 / 20
- Value > Q3 * 20

Being Q1 and Q3 the first and third quartiles of the dataset, respectively.

We chose '20' as the limit value because it allows a broader data interval, removing only the obvious outliers.

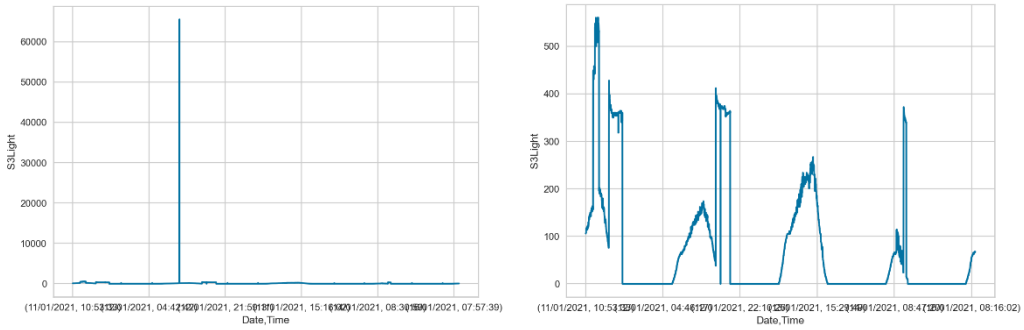


Figure 3: 'S3Light' before and after outlier detection and correction.

After removing the outliers, we normalized the data of all features except for the 'PIR1', 'PIR2', and 'Persons' features. For that we used *min-max normalization* ($\frac{x-min}{\max - min}$) so that we have a scale that goes from 0 to 1 (represented in Figure 4).

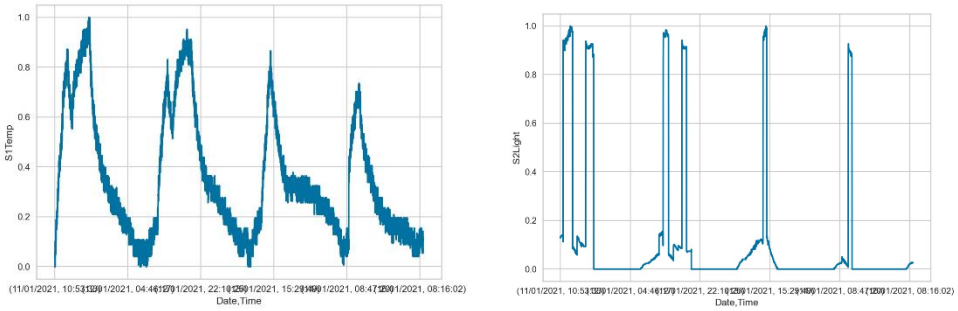


Figure 4: 'S1Temp' and 'S2Light' representation after applying min-max normalization.

As the 'preprocessing' phase of the dataset was completed, we started the neural-network model training.

For starters we divided the dataset in three different sets: data_train (60% of original dataset), data_validation (20% of original dataset) and data_test (20% of original dataset) using the 'train_test_split' function (of *sklearn* library) twice.

For the NN model we used MLPClassifier with the hyper-parameters *solver='sgd', activation='logistic'* (which is equivalent to the sigmoid function) and *hidden_layer_sizes=(5,2)*. The results weren't the best so we started by changing *solver='lbfgs'* which immediately showed improvements in the results.

For the first exercise, we started giving random values in *hidden_layer_sizes* but noticed that this wasn't the best strategy as we kept getting random results so, instead, we made a function based on for loops where we had the same hyper-parameters, changing only the neurons amount per hidden layer, starting at *hidden_layer_sizes=(1,1)* and finishing at *hidden_layer_sizes=(8,8)* and storing the corresponding f1 score in a dictionary, in order to find the best combination. The result of this function can be seen below in Figure 5.

```
(3,1) 0.6394396551475894
(2,3) 0.6419999765542406
(4,2) 0.6777216604029163
(2,2) 0.7280580480017615
(5,3) 0.7840627938597605
(1,4) 0.8249101920245427
(3,3) 0.851207949191797
(3,5) 0.8528680526308058
(1,3) 0.8543034176774915
(4,1) 0.8695302028980615
(4,4) 0.9080283996783073
(5,4) 0.9577520064334435
(4,5) 0.9748231442652546
```

Figure 5: Results of f1 score by different neuron combinations.

We tested the combination of *hidden_layer_sizes=(2,2)* because we tried to use the least amount of neurons and its *f1 score* was decent, however, by looking at it's Classification Report (plot of model's *precision*, *recall* and *f1 score*) we weren't satisfied so we changed the amount of neurons to *hidden_layer_sizes=(4,2)* which proved to be a better solution (classification report shown in Figure 6).

In order to improve the results we tried to increase the number of iterations but, for this exercise, it didn't make a difference so we went with the default value of *max_iter=200*. For this, we made a similar function of the one used for the neurons per hidden layer, using a for loop that kept the same hyper-parameters, changing only the value of *max_iter* from 200 to 400.

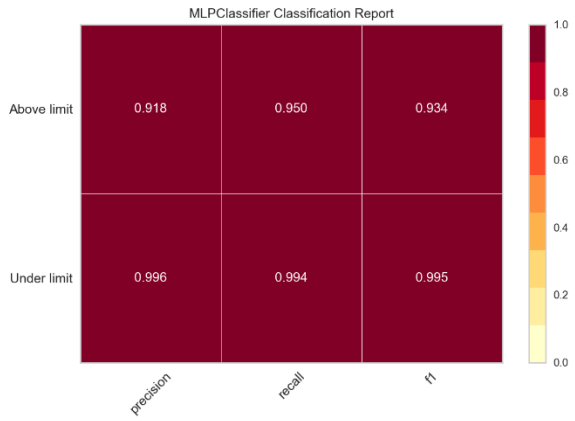


Figure 6: Representation of the classification report of the first exercise comparing the training data with the validation set.

As for the second exercise (multiclass) we used the same methodology as we used for the first exercise and we figured that the best combination of neurons, based on our function, were *hidden_layer_sizes=(4,5)* (as you can observe in Figure 7). We still tried other combinations we lesser neurons, however it kept being the best solution, shown in Figure 8.

```
(2,3) 0.6419999765542406
(3,1) 0.6419999765542406
(4,2) 0.6777216604029163
(2,2) 0.7280580480017615
(5,3) 0.7840627938597605
(1,4) 0.8249101920245427
(3,3) 0.851207949191797
(3,5) 0.8533796861253725
(1,3) 0.8559923316686096
(4,1) 0.8682783343017544
(4,4) 0.9357582300597006
(5,4) 0.9406978351372793
(4,5) 0.9738975501717445
```

Figure 7: Results of f1 score by different neuron combinations.

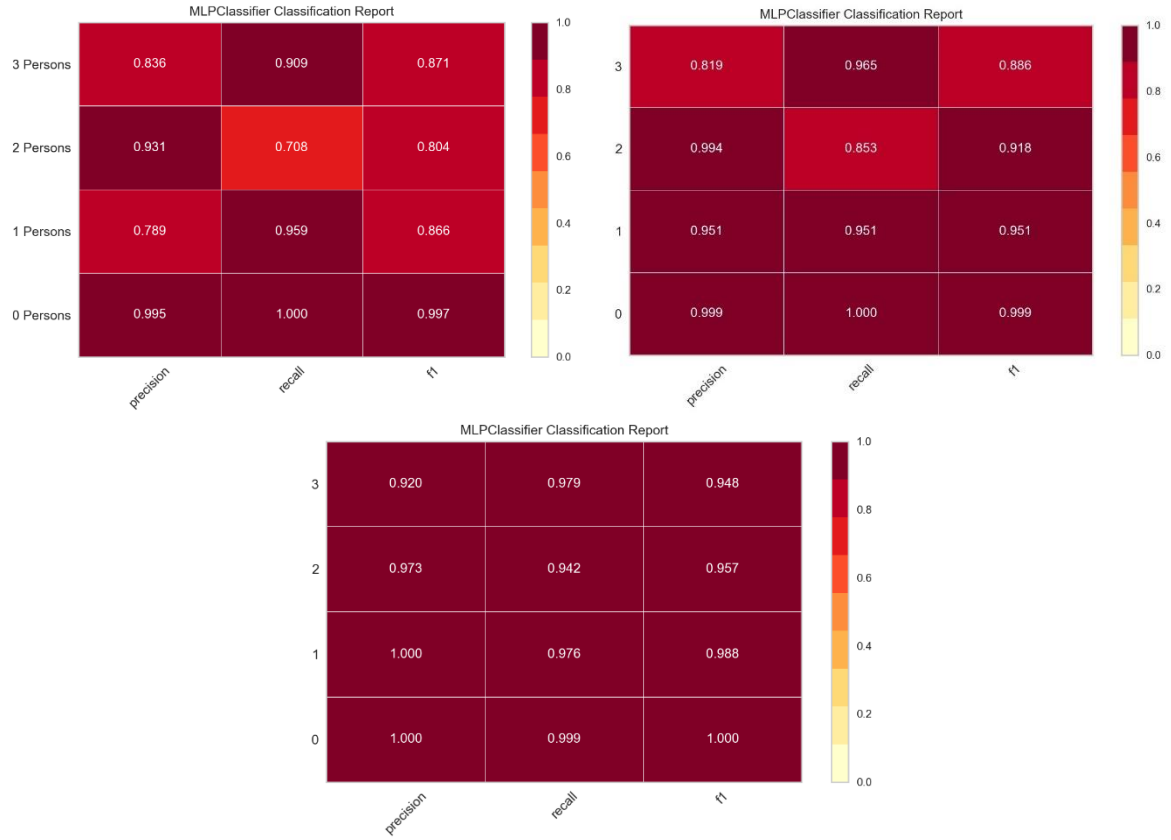


Figure 8: Representation of the classification report of the second exercise comparing the training data with the validation set. combination of (4,1), (4,4) and (4,5) respectively.

As for the amount of iterations, we used the same function as in the previous exercise. We noticed that the model achieved its best result at *max_iter=296* however, the f1 score of *max_iter=256* were almost the same (Figure 9), so we used *max_iter=256* in order to use the least number of iterations, as it didn't make a big difference.

```
(261) 0.9371930057793769
(262) 0.9371930057793769
(263) 0.9371930057793769
(278) 0.9372223689902198
(280) 0.9372223689902198
(378) 0.9380750875848916
(397) 0.938103098862064
(398) 0.938103098862064
(399) 0.938103098862064
(369) 0.9381346456729867
(379) 0.9381346456729867
(256) 0.9386094363742777
(296) 0.9396383773217424
```

Figure 9: Results of f1 score by different max iterations.

Finally, we tested our models with the testing set and we were very satisfied with the results we got, as we could check that overfitting hadn't happened (Figure 10 and 11).

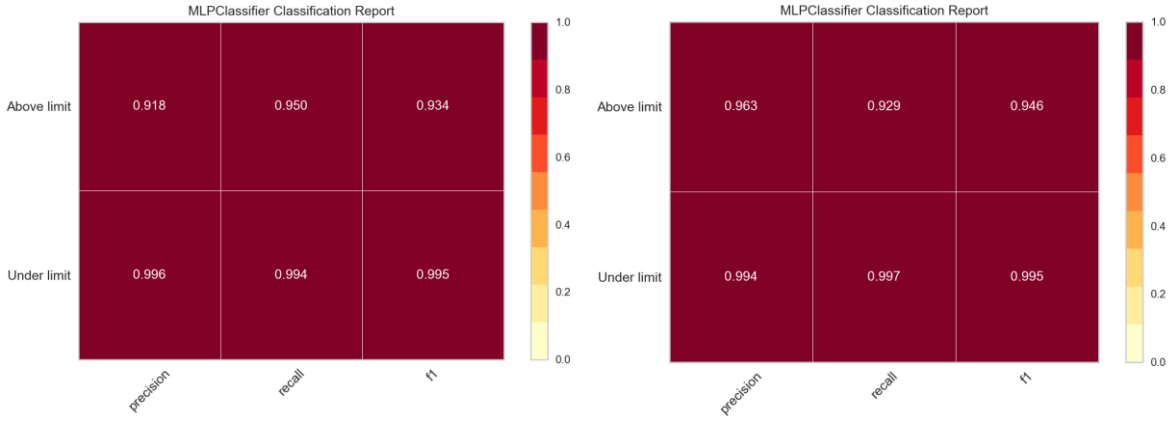


Figure 10: Representation of classification report of validation and testing phase in the first exercise, respectively.

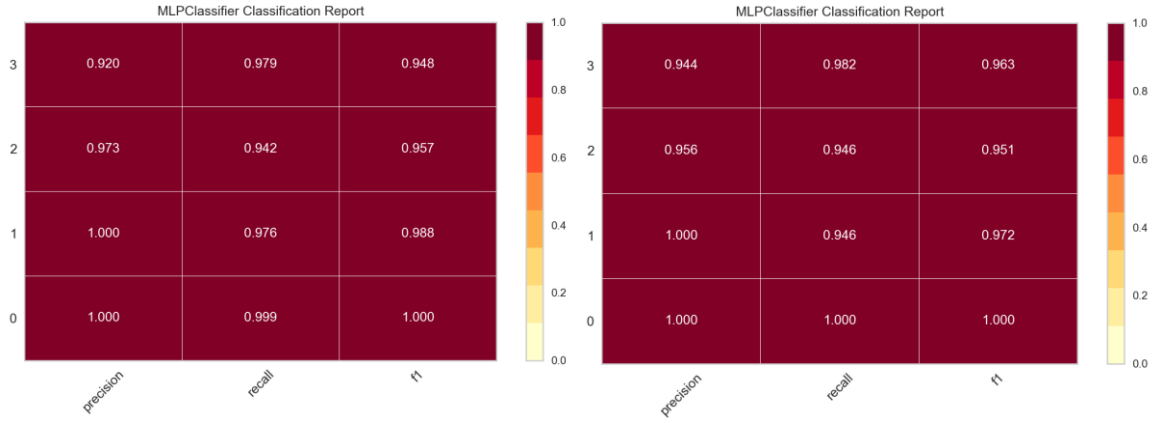
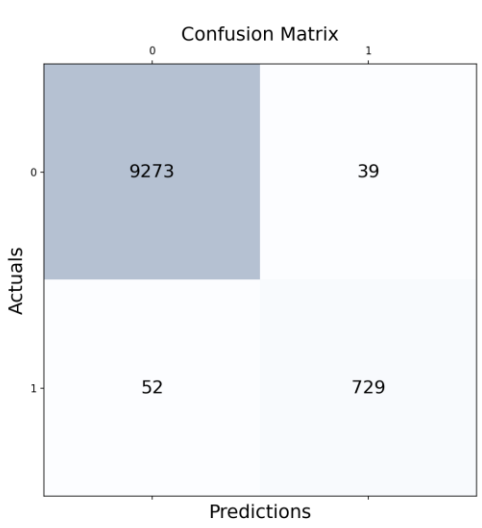


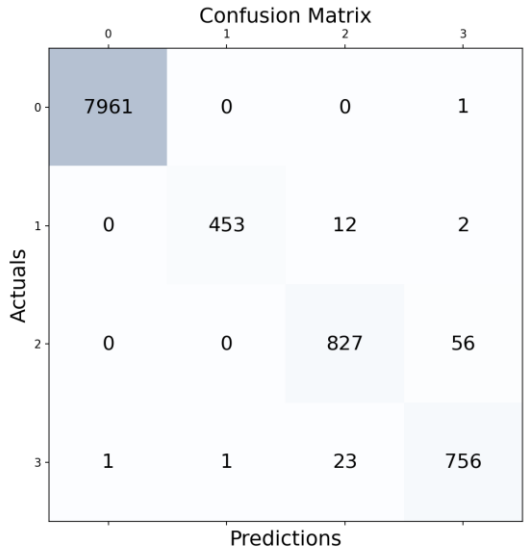
Figure 11: Representation of classification report of validation and testing phase in the second exercise, respectively.

After exporting the models using *pickle* library, we created the TestMe code, which calls the ProcessData class that has all our code that processes data. We tested with our dataset and got the following results.




```
#################
#  Exercise 1   #
#################
Precision
        Under Limit: 0.9944235924932976
        Above Limit: 0.94921875
        macro: 0.9718211712466488
Recall
        Under Limit: 0.9958118556701031
        Above Limit: 0.9334186939820742
        macro: 0.9646152748260887
F1
        macro: 0.9681848304079228
```

Figure 12: Results of the first exercise.




```
################
#  Exercise 2  #
################
Precision
        0 Persons: 0.9998744034162271
        1 Persons: 0.9977973568281938
        2 Persons: 0.9593967517401392
        3 Persons: 0.9276073619631902
        macro: 0.9711689684869376
Recall
        0 Persons: 0.9998744034162271
        1 Persons: 0.9700214132762313
        2 Persons: 0.9365798414496036
        3 Persons: 0.967989756722151
        macro: 0.9686163537160533
F1
        macro: 0.969701795595762
```

Figure 13: Results of the second exercise.

The code we used to train the models is in files *p1.py* and *p1_aux.py*. The first contains the model training and the second contains the data preprocessing.

To run *TestMe.py* to run in the terminal *TestMe.py [name of the dataset .csv file]*, having the .csv file inside the same directory as the *TestMe.py* file.