**CMPS 261 Project**

**Group name: UltraHackers**

**Group members: Joudy Al Ashkar, Khaled Gheith, Nour Obeid**

Link to the Github repository: <https://github.com/Khaled-Gheith/Ultrahackers_Machine-Learning.git>

**Data Cleaning:**

First, we imported the data using pandas. We noticed that the data is missing headers, so we added labels for each column. We noticed that there are some duplicated rows that we dropped down. We searched for rows with missing data, and we got their indices and dropped them accordingly. We noticed that the data is not of the same type. We got the indices of the columns and the rows where the data is of type string. We looped over the cells with string data and converted them to float if applicable. If not, we replaced them with the mode of the column. Finally, we checked if the data is skewed, and it turned out that the data is almost equally distributed between the two classes.

**Model Training:**

1. Logistic regression:

We split the data into training (80%) and testing (20%), and we used a logistic regression model from SkLearn, but the performance was very low, and we ended up with an accuracy of 64%.

1. XGboost:

We also tried the XGBclassifier with different parameters, namely: number of estimators, max depth and learning rate. After different models, we got an accuracy of 74.07% by setting the following parameters:

* max\_depth = 5
* eta = 0.3
* n\_estimators= 290
* min\_child\_weight= 10
* subsample=0.9
* colsample\_bytree = 0.9

1. Neural Networks:

The activation function for all the below models is the Relu function. Both Tensorflow and Sklearn’s MLP classifier were used to train these models.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models | Number of neurons | Number of hidden layers | Solver | Accuracy |
| Model 1 | (500,100,1) | 3 | X | 72.76 |
| Model 2 | (28,14,1) | 3 | X | 73.39 |
| Model 3 | (50,25,1) | 3 | X | 73.94 |
| Model 4 | (99, 88, 77, 66, 55, 44, 33, 22, 11, 1) | 10 | X | 74.15 |
| Model 5 | (100,100,80,66,50,25,1) | 7 | X | 74.31 |
| Model 6 | (75,50,1) | 3 | X | 74.47 |
| Model 7 | (90,60,1) | 3 | X | 74.76 |
| Model 8 | (28,25, 25, 20, 17,13,5,3) | 8 | X | 74.89 |
| Model 9 | (50,40, 30,15,10,5) | 6 | X | 74.97 |
| Model 10 | (80,50, 40,30,15,10,5) | 7 | X | 74.95 |
| Model 11 | (80,60,40,20,5) | 5 | X | 75.01 |
| Model 12 | (28,80,20,10,5,1) | 6 | X | 75.11 |
| Model 13 | (80,60,40,20,5) | 5 | Adam | 75.19 |
| Model 14 | (28, 18, 8) | 3 | X | 75.05 |
| Model 15 | (100,80,60,40,20,5) | 6 | Adam | 75.09 |
| Model 16 | (77, 53, 15, 6) | 4 | X | 75.10 |
| Model 17 | (28,90,20,10,5,1) | 6 | X | 75.21 |

We did train many more models, but the performance was not significantly different from the ones mentioned in the table.

After all these models, we decided to change the architecture, namely we tried to train two models, where each model takes a subset of the features, and the features are grouped based on similarities between them. Then, a third model consisting of one layer did the dot product between the outputs of the two first models.

The first model had 4 hidden layers with the numbers of neurons respectively 3000, 1000, 900, 200 and the second model’s layers consisted of 1000, 500, 128 and 200 neurons. We used Tensorflow’s regularizers.L2 with a value of 0.0005 in the first layer of the first model since it helped us to reduce overfitting. Finally, we set the batch size to 50, the number of epochs to 100 and we set an early stopping with patience = 15.

The training accuracy turned out to be 77.33 and the testing accuracy was 75.697.