Computational Quantum Physics and Applications: Machine Learning Project

Nikolaos Zafeiriadis

July 2025

Introduction

The aim of this project was to implement **machine and deep learning methods** to create a **classification model** that accurately predicts whether a set of experimental data from the LHC, studying the Higgs Boson, corresponds to a signal or background noise.

Dataset

The dataset is composed of 29 columns. The first column is the one stating the type of the row, 1 corresponding to signal and 0 corresponding to background. The rest 28 columns represent the independent variables of each row. The independent variables are split into two categories, the low level (columns 1-22) and high level (23-29) quantities. Although the original dataset contained over a million rows, the one used in this project was a smaller version of it, containing only 8000 rows.

Before implementing the machine and deep learning methods, the **cleansing** of the dataset was required. This cleansing of data was implemented using the script:

data_cleansing.ipynb

Machine Learning

The first step of the analysis was to implement two machine learning classifiers. The two methods was the **K-Nearest-Neighbors** classifier and the **Random Forest** classifier. Both methods were implemented 3 times, the first one the whole dataset, the second on the low level quantities and the third on the high level quantities.

KNN Classifier

The first step of the KNN method was to split the dataset into train and test parts, with the test part being the 25% of the dataset. The second step was to scale both the train and the test part using the Standard Scaler method. The scaling parameters were tuned using only the training part of the dataset. In order to fine tune the KNN algorithm, the function GridSearch was used so that multiple hyperparameters were tested and specify those that correspond to the highest accuracy. This implementation is found on the file:

classifier_knn.ipynb

The notebook also contains the resulting models, the confusion matrices and the ROC curves of each part of the dataset. The achieved accuracies on the test dataset were the following:

```
Every Column Accuracy score = 0.6116941529235382
Low Level Accuracy score = 0.5627186406796602
High Level Accuracy score = 0.6646676661669165
```

The best model out of the three is the one using only the High Level quantities, according to their accuracy.

Random Forest Classifier

The first part of the Random Forest method was once again splitting the dataset into train and test parts, exactly as in the case of the KNN classifier. In this implementation, no scaling was required, so the following step was the fine tuning. Instead of using the GridSearch, the function RandomSearch was used in order to specify the hyperparameters that returned the highest accuracy. This implementation is found on the file:

classifier_random_forest.ipynb

The notebook once again contains the resulting models, the confusion matrices and the ROC curves of each part of the dataset. The achieved accuracies on the test dataset were the following:

```
Every Column Accuracy score = 0.6976511744127936
Low Level Accuracy score = 0.5897051474262869
High Level Accuracy score = 0.6806596701649176
```

Deep Learning

The second part of the analysis was to implement a deep learning method in order to create the classification model. The way that this was implemented was via building and training a **sequential Artificial Neural Network** (ANN)

using the **Tensorflow/Keras** Python library. As in the machine learning classifiers, three different ANNs were created, the first for every quantity, the second for the low quantities and the third for the high quantities.

The first step was once again the splitting of the dataset into train and test, proceeding with the scaling using the Standard Scaler method. After fine tuning every ANN, the resulting accuracies on predicting the test dataset were the following:

```
Every Column | Validation accuracy score is: 0.6226886510848999
Low Level | Validation accuracy score is: 0.530734658241272
High Level | Validation accuracy score is: 0.6966516971588135
```

Every ANN implementation is found on the following script, along with their confusion matrices and ROC curves.

ann.ipynb

Results and Discussion

According to the results, all the models that were created using the Low Level quantities clearly were the least efficient in the classification problem, having an accuracy slightly better than randomly choosing between 0 and 1.

Using only the High Level quantities resulted into the best accuracies in every case, except the case of the Random Forest classifier, where using every column as input resulted into slightly better predictions.

The worst method seemed to be the KNN classifier, while the Random Forest classifier and the ANNs provided similar outcomes.

The best model out of the total nine, seemed to be the ANN trained using only the High Level quantities. Even though, the accuracy of the Random Forest classifier has a slightly higher value, if one considers the fact that it requires 28 variables as input instead of only 7, it is marked as a "worst" model.

The confusion matrix of the ANN model with only the High Level Quantities is the following:

$$\begin{bmatrix} 688 & 317 \\ 290 & 706 \end{bmatrix}$$

The AUC score of the ROC Curve is:

```
0.7609352834222463
```

and the ROC curve is presented on the following figure:

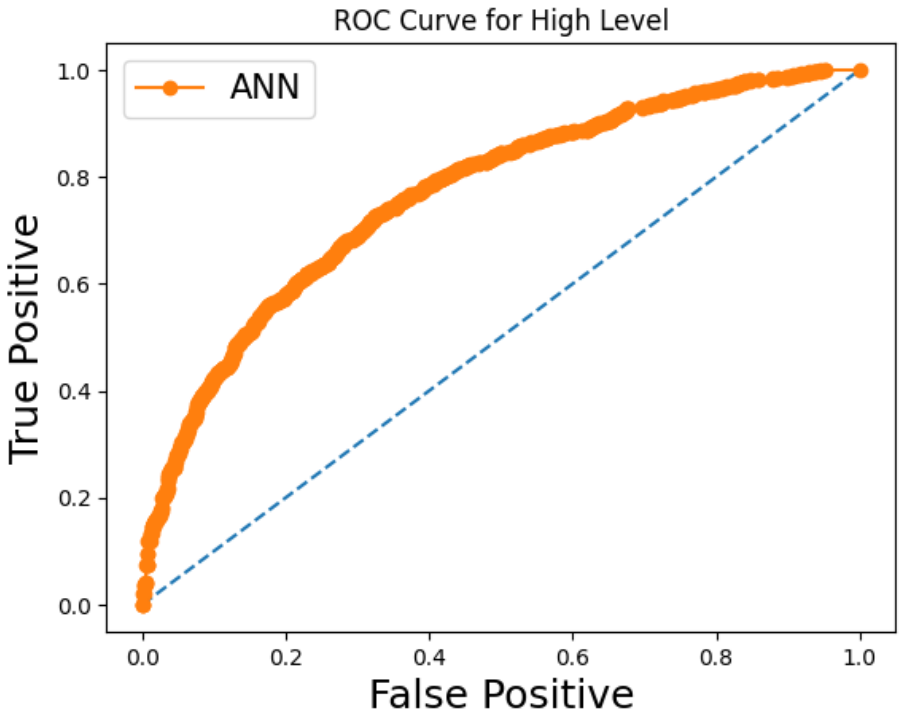

Figure 1: *ROC Curve of the ANN model using only the High Level quantities.*

In conclusion, the best model was the ANN using only the High Level quantities. On the other hand, all 9 models did not reach an accuracy that would make them considered efficient at predicting if a case is a signal or background. The most probable reason of that is the size of the dataset. There is a high probability that if the original dataset was used in training, the resulting ANNs would reach accuracies that could be considered sufficient.