Computational Quantum Physics

Christos Papazois

[1]Physics Department
Aristotle University of Thessaloniki

Abstract. *Machine Learning Applications on Signal-Background Separation in Exotic Higgs Scenarios*

1. Introduction

Supersymmetry (SUSY) is a prominent extension of the Standard Model (SM) that predicts a richer Higgs sector, comprising five Higgs bosons instead of one. This extended structure includes two charged and three neutral Higgs states, leading to a variety of possible decay channels not present in the SM. In this study, we focus on a specific SUSY-inspired decay mode where a heavy neutral Higgs boson decays into a pair of W bosons and a Standard Model-like Higgs boson, following the process $H^0 \to W^+W^-h$. The final state consists of two leptons (from $W^\pm \to \ell^\pm + \nu$) and two b-jets (from $h \to b\bar{b}$), resulting in a signature that can be targeted in LHC experiments.

To enhance the separation between signal and background events in this complex final state, we apply and compare three machine learning classification algorithms: K-Nearest Neighbors (KNN), SVM Algorithm, and Artificial Neural Networks (ANN). The models are trained on a dataset that includes both low-level variables and high-level variables (composite features constructed from physical observables). Our objective is to evaluate the effectiveness of each method in identifying signal events (labeled as 1) against background events (labeled as 0), and to understand the impact of feature selection on classification performance.

2. Data

In every method before running the algorithm we pre processing the data. First of all, we separate them according to the orders of the file as following: The first column was for classification as 0 indicates "background" and 1 "signal" , the next 21 columns where for low level and from 22^{nd} column till the end where the high ones. Also we observed due to the codes bug that the data are not "clean". In the 17^{th} column there is a value that is classified as an object and we replaced it with the value of 0.

3. K-Nearest Neighbors-KNN

After defining the data as we mentioned before, we applied the KNN classifier initially for the low level and then for the high. We run everything inside a loop that runs for different values that represent the number of neighbors. The data were spitted into training and test sets by the ratio of 80%-20% .

Inside the loop the algorithm works as: First using a pseudo number splits the data with the ratio we mentioned. Then uses a "scaler" to standardize features (make them have mean 0 and standard deviation 1) and then applies this scaler to the spitted data. This part puts all features on the same scale and it is important because the algorithm is based on the calculation of the points between themselves. Meaning uneven and unscaled data might create serious problem to the calculation. Then the algorithm stores the training data and in order to predict a new point measure the Euclidean distance between a certain point and all training points. For example if the neighbours were 20, it would find the 20 closest points and count the classes among those 20 points then it would assign the most frequent class as the prediction. We recorded the confusion matrix, plotted the roc and auc graphs and repeated the process for the high level. The k neighbors we used were [1,5,10,20,50,100].Below are the results of each:

k-neighbors	Low Level Acc.	Low Level AUC	High Level Acc.	High Level AUC
1	0.525	0.48	0.610	0.39
5	0.560	0.43	0.676	0.27
10	0.573	0.41	0.672	0.27
20	0.577	0.39	0.673	0.26
50	0.596	0.36	0.673	0.25
100	0.571	0.38	0.674	0.25

As k neighbors are increased the accuracy is getting better which is of course obvious and also better results are shown in High Level. Overall these results along with the AUC scores are not that good for our model showing that are getting almost random values in both situations.

4. SVM Algorithm Method

After using K-Nearest Neighbors (KNN), which predicts a point's label based on the closest neighbors measured by Euclidean distance, we will try Support Vector Machine (SVM) Algorithm. Unlike KNN, which looks at local neighbors, SVM finds a single best boundary that separates the classes. It uses only the most important points near this boundary, called support vectors, to create a model that can better generalize to new data. Here it do not compute distances to every point when predicting but it creates a hyperplane of the closest points. It focuses on the support vectors — the training points closest to the boundary — and tries to maximize the space between the boundary and those points. When trained, it uses this boundary to classify new points. The results are below:

Low Level Acc.	Low Level AUC	High Level Acc.	High Level AUC
0.573	0.60	0.621	0.67

The interesting part here is that, although both models achieve nearly the same accuracy, the SVM model shows a significantly higher AUC—almost double that of the KNN model. So even if both models predict the correct labels at a similar rate, the SVM provides more confident probability estimates, leading to a stronger overall classification performance.

5. ANN-Artificial Neural Network

We first split the data into training and test sets, then applied feature scaling using StandardScaler to normalize the inputs. The ANN model was built using TensorFlow/Keras with an input layer (matching the 20 columns), two hidden layers (with 12 and 8 neurons respectively, using ReLU activation), and a sigmoid-activated output layer for binary output. The model was compiled with the Adam optimizer and binary cross-entropy loss function, then trained over 200 and 300 epochs by alternating the batch size as 6,8,16 and 32. Finally, we evaluated its performance using a confusion matrix and accuracy score on the test set and also with the ROC curve. We repeat for the high level.The results are below:

Batch	Epoch	Low Level Acc	High Level Acc.
32	200	0.593	0.702
32	300	0.595	0.698
16	200	0.600	0.695
16	300	0.594	0.688
8	200	0.609	0.688
6	200	0.597	0.688

We also plotted the ROC graph and found for Low Level AUC values either 0.36 or 0.37 and for High Level either 0.23 or 0.24. We can clearly see that the best results are with 8 batches and 200 epochs for low level and with 32 batches and 200 epochs for high level.Generally smaller batch sizes help in more noisy data, while larger batches may work better with cleaner or more abstracted data. However AUC value indicates here that our model has a serious issue and is not learning meaningful discrimination, despite these accuracy improvements.

6. Conclusion

We clearly saw that in each of the three Methods the High Level outperformed the Low one with the ANN Method to give the most accurate results. However accuracy is not enough to mark a method as meaningful or sufficient since low AUC values show that the model cannot distinguish between the positive and negative classes.Overall the performance of the methods caused in a bigh part from the Data itself since they were intuitive and not so many to let us simulate a "real" problem. Also we saw that except that , proper hyperparameters may improve even here the performance significantly .