Machine learning for Higgs boson classification

Marie-Alix GILLYBOEUF, Baptiste HERNETTE, Gaspard VILLA
EPFL, CS-433, Machine Learning

***Abstract*—In this project, the main objective is to apply machine learning methods to "discover" the Higgs particles using only "nympy" library from python, *i.e.* without using external libraries, code or data. The idea is to use CERN particle accelerator data to build a model that predict if after a collisions, the resulting particles are either Higgs bosons or a background particles.**

I. INTRODUCTION

In this project, we aim to build a python code to create a model that predict which particles are produced after collisions events. This problem is a classification problem, *i.e.* in our case we classify particles: either Higgs boson or background particle. Specifically, the particles should have a label s (signal) if it is a Higgs boson and b if it is a background particle. Our executable code `run.py` will build different machine learning methods for binary classification, evaluate them and select the best model for our problem. Here we will present our initial data set and how we performed exploratory data analysis (EDA) on it, then models for binary classification will be examined and tested with regards to the loss functions.

II. EXPLORATORY DATA ANALYSIS (EDA)

The data set we work with contains thirty features (from 0 to 29) that should predict a binary output with label s for Higgs boson and b for background particle. For easier data manipulation the labels are redefined as 1 for Higgs boson and 0 for background particles. By looking each features individually, we observe that feature 22, which correspond to the number of jets (PRI_jet_num) is a categorical features that can take only four integer values (0, 1, 2, 3). To deal with that, we decide to classify our data into four classes (*class_0*, *class_1*, *class_2* and *class_3*) according to the value of the feature 22. Then, we perform EDA and try different machine learning methods for binary classification on each of these classes.

For each class, we first delete irrelevant features, *i.e.* features that should not be taken into account in the model of the class according to the documentation. Secondly, we delete features that don't provide any information in predicting the output. To do so, we plot histograms of the density of output 0 and 1 for each feature. Thereafter, by looking at them individually, we remove the features for which there is an overlap of the histograms for the output response (either 1 or 0), see Figure 1.

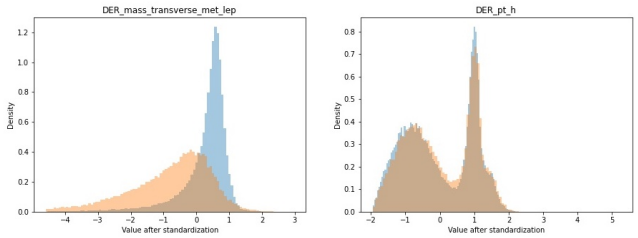

Figure 1: Histograms of the density of feature 1 (DER_mass_transverse_met_lep) and 3 (DER_pt_h) for *class_0* with respect to the output response. Total overlap of both of the histograms in feature 3 reveals that it doesn't provide any useful information in predicting the output and thus we should delete it in our model. On the contrary, for feature 1 we have two distinct histograms meaning that we should keep it in our model.

To get an idea of the distribution of our data into each features, we plot the boxplot. This enables us to notice that many features contain more or less outliers and undefined values (represented by -999 values). To deal with these oultiers, we first select them using the interquartile range method and then we put them to the median of the feature (without considering oultiers). After doing so, we notice that in the four classes, feature 0 is the only one which still have -999 values so we decide to delete it in all classes. In order to recenter the data and to respond to unbalanced large values, we apply a logarithmic filter ($f(x) = log(1 + x).sign(x)$) that normalize the data. To compare scores of different types of features we put them on the same scale using standardization technique. Then, after deleting constant features (*i.e.* features that contain the same value for all the data points), we delete those who are highly correlated (*i.e.* when the correlation coefficient between two features is higher then a certain threshold (0.95) or according to the correlation matrix plot).

Finally, to obtain better results, we split the data set into a training set and a testing set according to a 0.8 ratio. We use the training set to build models and the testing set to select the best one according to the accuracy.

III. CROSS-VALIDATION

A. Hyper-parameters tuning

To compute a model, we need different optimization methods to train our model. We implemented six of them that are the following: *least square regression*, *linear regression* using either *gradient descent (GD)* or *stochastic gradient descent (SGD)*, *ridge regression*, *logistic regression* and *regularized logistic regression* both using *GD*. The goal

is to find the best model among all, for each of the four classes introduced in Section II. As each method uses some parameters, we do sort of "hyper-parameters tuning" using *Cross-Validation* methods. The idea is to go over a large set of possible values for the parameters and keep the ones that give the lowest error on the test set. Note that the error of the test is computed as seen in the course of Machine Learning by folding the given train set into k folds. Then, we train our model with $k-1$ folds and test it on the last stayed fold and we do this for all possible combinations of $k-1$ trained sets. Finally, the returned error is the average error obtained over all the tested subsets. This is how we select the best parameter for methods that have only one parameter (`gamma` or `lambda`). For methods that requires two parameters, as regularized logistic regression, we first do the previous method for one parameter. Then, we fix the first parameter to the best found before, and we use this method for the second parameter. It is not exactly the optimal way to find the best combination of the two parameters, but it is a way that keeps the computation time relatively acceptable.

B. Degrees of features

After tuning parameters for the different methods, we have to deal with the possibility that our model can be non-linear. The idea behind that is to see if a feature x_i that is transformed to x_i^d, for d (an integer we called "degree"), can increase the accuracy of our model when it is trained. To check if some feature needed to be increase to some power d, we use some statistics methods such as *forward selection* and *backward selection*. The idea of the first method is to begin with an empty model (*i.e.* only a column of ones), iterate over the different values of degree d, iterate over the different features, and finally check if adding the i^{th} feature to the power d increases the accuracy of our model. If it is the case, we add this new feature to our model and we do the same for the next feature and so on. If it is not, we do again the previous step with the old model for the next feature or degree. For the backward selection, contrary to the forward one, we begin with the largest possible model, *i.e.* all the features to the power d for $d = 1, \dots, d_{max}$. Then, we iterate over all features in this model and we check if the removal of this feature either increases the accuracy of our model or not. If it increases the results, we continue the process with the reduce model (*i.e.* without the tested feature) otherwise we keep it and continue.

Now that we have two different ways to increase the accuracy of the model, we just have to test them both and keep the best one. Another thing we also need to consider is the possible interaction between some features. For example, the model can be more accurate if the feature $x_i x_j$ is added to it for i not equals to j. Then, we apply a forward pass over all the possible pairs of feature and check which one of them is worth to keep in our final model. All of this is registered in the *Cross-Validation* process for a fixed method.

IV. Model selection

In Section III, we have designed tools that allow us to optimize one of the six main methods presented. We will now compare the accuracy between all those methods after using *Cross-Validation* on each of them. These results, for $class\_0$, are resumed in the following Table I:

Methods	Train error [%]	Test error [%]
`Least squares`	15.23	15.30 ± 1.13
`Gradient descent`	16.60	17.07 ± 1.16
`Stochastic Gradient descent`	19.64	19.78 ± 2.43
`Ridge regression`	15.20	15.25 ± 1.04
`Logistic regression`	16.60	17.07 ± 1.16
`Regularized logistic regression`	16.89	16.94 ± 1.22

Table I: Train and test errors for the six considered methods for $class\_0$ with the standard deviation of the test error. These results reveals that *Ridge regression* method seems to be the best method for minimizing the test error on $class\_0$.

For each class we pick the best model to build the best model for any data point. We split the test set into the four classes using our classification presented in Section II, and then applied the best method found for each class individually. This method gives us an accuracy of 81.2% when submitted on AI Crowd.

V. Conclusion

To conclude, through this study we build a model for binary classification of particles resulting form collisions (either Higgs bosons or a background particles) using CERN particle accelerator data. To do so, we classify our data into four classes and perform EDA on each of them. This EDA treats outliers, cleans irrelevant, highly correlated and constant features, normalizes and standardizes data in order to have the more accurate data set. Then, we use *least square regression*, *linear regression* using either *GD* or *SGD*, *ridge regression*, *logistic regression* and *regularized logistic regression* both using *GD* to individually fit our classes. For each of these methods we tune hyper-parameters using *Cross-Validation*. This enables us to select the best parameters for each methods according to the lowest testing error. Then, to deal with possible non-linearity of our model and possible interaction between features, we introduce transformed features that are selected based on either a *forward* or a *backward selection*. Finally, among all the models obtained with such processes, we selected for each class the best model and we group them to create a model for the whole data set.