Higgs boson classification using machine learning implementation

Aline Brunner, Louise Font and Julie Korber
Machine Learning CS-433, EPFL

***Abstract*—In this project, using the python numpy library, we implemented a machine learning program in order to determine whether particles providing from proton collisions are Higgs bosons or not. We did so using data provided by the CERN particle accelerator. Good results have be obtained with some preprocessing and Ridge Regression.**

I. Introduction

The Higgs boson is an elementary particle which was theorized by physicists about 50 years ago. It was only in 2012 that scientists at the CERN finally discovered a particle showing the required properties. In this project, we aim to predict what type of particles are produced after a collision event using data from the CERN. In particular, we want to determine whether after a proton collision, the measured particles are Higgs bosons or simply background noises. To address this classification problem, we implemented a machine learning code in python. We explain here the exploratory data analysis, methods and results we obtained.

II. Models and Methods

A. Exploratory Data Analysis

First, we decided to explore the data to see what preprocessing could be applied. This is called exploratory data analysis (EDA): modify and clean the data in order to obtain better results later on.

The data set is composed of 30 features for each measured particle and an output label indicating whether it is a boson (*s* for *signal*) or noise (*b* for *background*). We began by replacing this binary classification label by $\{0;+1\}$ or $\{-1;+1\}$ depending on the method used.

Then a basic preprocessing has been applied, which consists of the following:

- The non defined values (-999 in dataset) were replaced with the median of its feature (computed after removing the undefined values).
- Furthermore, some features are angle values. Since these values are contained between $-\pi$ and π, two very close angle can have two very different values. We therefore decided to replace these angle values by their *sin* and *cos* to give some more linearity.
- Then, the data have been standardized. The mean and standard deviation have been computed for each individual feature.

Some more elaborated preprocessing could also be tested:

- After looking at the correlation heatmap (see notebook) we observed highly correlated features that can have a negative impact for two reasons : firstly, many features imply a greater training time and secondly, two highly correlated features can obtain some contradictory weights. Therefore, if two features have a coefficient above 0.95, one of them was deleted.
- Principal Component Analysis (PCA) was applied. This method allows to determine new "features" which have the maximum variance across the data. With this, we reduced the dimensionality of the data by selecting the first n^{th} components that explain 95% of the variance across our data set.
- To deal with non linear data, polynomial feature expansion has been performed. The degree expansion is a hyperparameter that has to be chosen carefully.
- Based on one hot encoding of the categorical feature *PRI_jet_num*, the dataset could be split into four different sets, trained and predicted separately.

All these transformations were first applied on the training set and then reproduced with the same parameters on the test set. The EDA function is encoded in a modular way so that the user can decide which different steps of the elaborated preprocessing to apply.

B. Apply method function

We conceptualized one function (*apply_method*) that allows to compute the wanted method based on its name. The goal is to have a modular code. This allows to call any implementation specifying the parameters, the chosen loss function and whether to use a validation set. The different methods used are least squares (LS), ridge regression (RR), gradient descent (GD), subgradient descent (SGD), logistic regression (LR) and regularized logistic regression (RLR). The four first methods use mean squared errors as measure, whereas the last two use the negative loss likelihood. Concerning the labels, we determined that the LS, the RR, the GD and the SGD give better scores when the input y $\in \{-1, 1\}$ instead of $\{0,1\}$. LR (and RLR) need y $\in \{0, 1\}$.

C. Cross Validation

A big risk in machine learning is to overfit the training set, which means that the model performs very well on the training data but performs badly on an unseen dataset. To avoid this, k-fold cross-validation can be used:
The dataset is split in k subsets, trained on all sets except

the k^{th}, and then a validation is performed on the k^{th} set (using the weights obtained with the training sets, predicting an output and computing the error on the validation set). If both training and validation errors are near and low, the model is good. If the training error is much below the validation error, the model is overfitting. In order to compute the best hyperparameters, the model is trained on each sets (except the k^{th}) and validated on the k^{th} for each k in k-fold. Then the mean of the validation error is taken for each parameter. Finally, the hyperparameters are chosen based on the best mean validation errors. We use the MSE, respectively negative log-likelyhood, for all parameter tuning except for the polynomial degree for which we chose acuracy. We used two functions for cross validation: the first one only tunes one parameter at the time, and the second one tunes up to three parameters at the time. The advantage of the second one is to select the real best parameters, as they are interdependent (e.g. for gradient descent, a small gamma should lead to a higher maximum number of iterations, whereas a bigger gamma to a smaller maximum number of iterations), but its computational cost is higher.

III. Results

A. Preprocessing

Across all methods, different preprocessings have been tested (example for RR in *Table 1*). They all showed the same tendency: one hot encoding class separation with polynomial give the best results.

	Training accuracy	Validation accuracy
Basic preprocessing	0.72207	0.71892
Delete correlated features + PCA	0.71219	0.71086
Polynomial (degree 8)	0.81484	0.81402
Polynomial (degree 6) & hot encoding	0.82422	0.82596

Table I
RESULTING CATEGORICAL ACCURACY BY TUNING RIDGE REGRESSION IN DIFFERENT PREPROCESSING CONDITIONS.

B. Methods comparison

Our best results were obtained with the ridge regression method (*Table 2*), where we obtained a validation accuracy of 0.82596. The results obtained on AICrowd (Accuracy: 0.825, F1-score: 0.729) confirmed this observation.

	Training accuracy	Validation accuracy
Least squares	0.82424	0.82560
Ridge regression	0.82422	0.82596
Gradient descent	0.80834	0.80858
Stochastic gradient descent	0.57678	0.57812
Logistic regression	0.81131	0.81314
Regularized logistic regression	0.81553	0.81664

Table II
RESULTING CATEGORICAL ACCURACY FOR EACH METHOD USING BASIC PREPROCESSING, POLYNOMIAL FEATURE EXPANSION AND ONE-HOT ENCODING

IV. Discussion

A. Preprocessing

The basic preprocessing helped a lot to improve the predictions. Concerning the elaborated preprocessing, PCA & deleting correlated features haven't helped at all. All the different methods didn't show much differences between them, so we concluded that the preprocessing could be improved. Most of the data aren't linearly distributed, and that seems to be the case with the Higg's boson dataset too. This is the reason why polynomial features expansion helped a lot. One hot encoding class separation showed also significant differences.

Actually, the best obtained results are on RR. We think that better predictions can be made with RLR, but as this method has 3 parameters, which choice are dependant on each other, we didn't have enough time to tune this method properly. We could also chose to tune one parameter at the time, but this doesn't lead to as good results as by tuning them together.

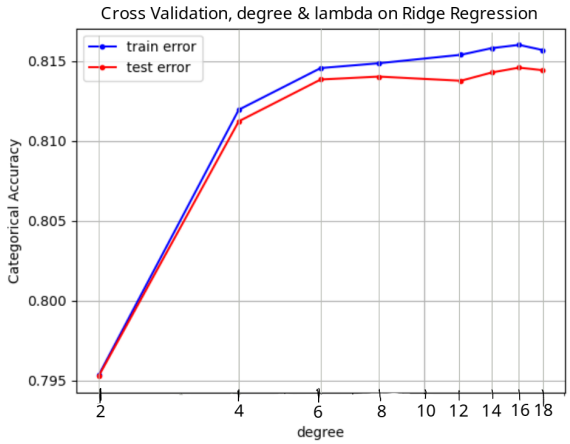

Figure 1. Figure illustrating one training on ridge regression, tuning the degree and lambda. Surprisingly, very high degrees, even until 16, give better accuracy results. The difference is sufficiently small to let the computational cost disadvantage take over the accuracy improving.

B. Methods

RR, as expected, gave better results than LS, thanks to the regularization parameter. GD, and especially SGD, didn't show good performance. This made us think that the dataset has a distribution close to a polynomial one. Also, we noticed that GD and SGD diverge with a too high gamma; depending on the method and the polynomial degree, the limit value fluctuates.

V. Conclusion

After exploring the data, preprocessing them and testing many machine learning methods, we could finally obtain quite good results using ridge regression, with a validation categorical accuracy of 0.826, which is a quite high value.

References

[1] Claire Adam-Bourdarios Glen Cowan Cécile Germain Isabelle Guyon Balázs Kégl David Rousseau. *Learning to discover: the Higgs boson machine learning challenge*. [Online; accessed 31-October-2022]. 2014. URL: https://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf.