Machine Learning Project 1 : Higgs Boson

Mehdi Zayene, Augustin Henry, Juliette Challot
Department of Computer Science, EPFL, Switzerland

***Abstract*—This project relies on data analysis and machine learning techniques to generate predictions on a data set.**

I. Introduction

The Higgs boson is an elementary particle. Although it is not possible to observe it directly, physicists at CERN have been able to detect it's decay signature in their particle accelerator. For this project we use actual data from the CERN to reproduce its discovery ie we try to predict whether a collision event resulted in a Higgs boson or not based on the decay signature of the event. In order to do this, we firstly explore the data, then process it and finally, we apply machine learning algorithms to generate predictions.

II. Exploring data Analysis

A necessary first step is exploring and understanding the data given. In our case the data is split into 2 sets : a training set (250 000 events) and a testing set (568 238 events). In both sets, each event has 30 features; taking a closer look at each of them and exploring the relationships that binds some of them together was the basis of our data processing.

III. Processing and transforming the data

Processing is key to reveal the information that lies within the data. In the following sections, we describe all the techniques that were used, tried and possibly dropped to improve the quality of the information fed to our model.

A. Data cleaning

*1) **Separating the data:*** In the dataset, only one feature is categorical : the PRI_jet_num, taking value in 0,1,2,3. We noticed that this value was directly linked to the total absence of values of certain features. Namely, when this feature is 0, eleven features have value -999 for all events. Similarly when it is 1, seven features are completely filled with -999. When it is 2 or 3 however, no such pattern was observed. We thus decided to split the data into three groups: one containing events with PRI_jet_num equal to 0, one with PRI_jet_num equal to 1 and the rest together. For the first two groups, 12 and 8 columns were dropped respectively (missing value and PRI_jet_num column). From this point on, the data is processed separately as tx1,tx2 and tx3.

*2) **Dealing with missing values:*** To remove punctual missing values, two strategies where explored : replacing them with the mean of the feature's values (-999 values excepted), or taking the median. We later chose the median, based on the accuracy of our predictions in both cases.

*3) **Dealing with correlated features:*** Having highly correlated features in a data set is not desirable as it does not bring additional information and complexifies the model. Keeping them can therefore increasing the risk of error. Thus, we checked the correlation factor between features and dropped features that were more than 80% correlated with another feature to keep only one of them. The correlation matrix of the tx2 is given as an example in Figure 1.

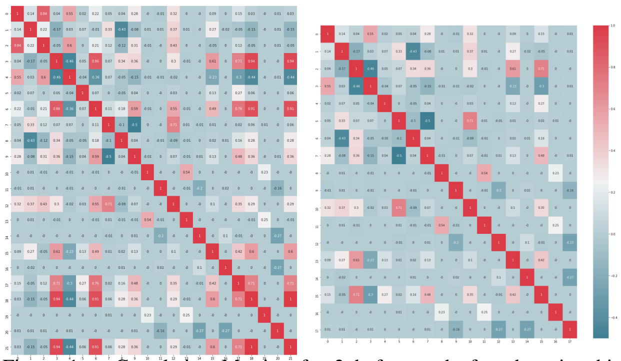
Figure 1. Correlation Matrix of tx2 before and after dropping highly correlated features

B. Feature engineering

*1) **Standardising:*** To make our dataset is consistent, we standardise the data ie for each feature x_i we perform the following transformation :

$$x_i \rightarrow \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{1}$$

This redistributes the data in the range [0,1], an example is shown in Figure 2.

*2) **Reducing skewness:*** Linear regression assumes normal distribution of the data. When plotting the features, we noticed that some of them were highly skewed. This is not desirable as it violates the model's assumption so we applied the log function on all skewed feature values x_i as follows:

$$x_i \rightarrow log(1 + x_i) \tag{2}$$

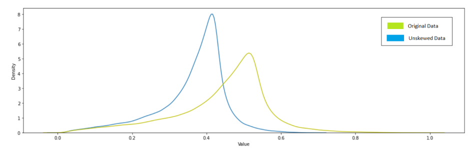

Figure 2. Feature DER_deltaeta_jet_jet before and after skew reduction

*3) **Normalising:*** Normalising means giving all the features a common scale, it is done by performing the following transformation :

$$x_i \rightarrow \frac{x_i - mean(x)}{std(x)} \tag{3}$$

*4) **Performing feature expansion:*** A way to account for non-linearity in a data set is to perform feature expansion. Although this method is very efficient to avoid underfitting, it needs to be carefully implemented so as not to overfit the data either. We implemented several methods of feature expansion and tested their impact on train and test accuracy through cross validation. In our final model we use pairwise product feature expansion which is a kind of feature expansion that uses both polynomial and cross terms expansions:

$$(x_i, x_j \rightarrow x_i * x_j \forall i, j 1 \leq i < j \leq D) \tag{4}$$

This technique exploits the pairwise relationships between features. We also tried $sin(x_i)$ or $cos(x_i)$ feature expansion but although it slightly increases our results on cross validation, it didn't improve our accuracy on AIcrowd.

*5) **Data leakage avoidance:*** We were very aware about data leakage during our cross validations and testing process.In order to prevent this phenomenon, we stored all our transformations parameters (median and mean for normalisation, min and max per feature for standardising...) in order to apply the exact same transformations on our testing set with the same parameters. Doing this improved a lot our results (order of 0.2 depending on the model used).

IV. Machine learning techniques

To train our model, we try different linear regression algorithms and choose the one gives the best results. This means that for each one, we optimise the hyper-parameters and performing cross validation multiple times to evaluate both the average and standard deviation of the accuracy on the validation set. This ensures that our model is stable.

*A. **Least squares***

This is a simple yet powerful method to establish a model by solving the equation: $\tilde{X}w = \hat{y}$, where $\tilde{X}$ is the data with additional offset term. This technique gave great and fast results (see Table II). However, we noticed that although the best accuracy was as high as 0.85, the standard deviation of the accuracy was also a little high meaning the model wasn't very stable. We thus went on to use other methods.

*B. **Linear regression with GD and stochastic GD***

Both techniques were tried in cross validation with several degrees of expansion but the result were not as good as the least square algorithm (around 74% as shown in Table II), we therefore decided not to continue with these methods.

*C. **Ridge regression***

Ridge regression is a model that penalises complex model and favors simpler ones to mitigate overfitting. To optimise the hyper parameter λ of the regularizer, we performed cross validation on forty lambdas spaced evenly on a logarithmic scale going from 10^-4 to 1. We then tried all these lambdas for degrees of feature expansion between 1 and 17. Based on these results, we checked the stability of the model by computing the average and standard deviation of the accuracy on 50 cross validations for the five (λ,degree) pairs per groups that gave the best test accuracy. The results of this optimisation process is detailed in Table I where accuracy is the mean accuracy over the 50 cross validations and std is the corresponding standard deviation. As the standard deviation is small we can say that are model is stable.

Ridge Regression				
batch	λ	degree	accuracy	std
1	0.0001603718743751331	12	0.84530	7.4176e-4
2	0.0001603718743751331	14	0.80644	1.8879e-4
3	0.002154434690031882	13	0.83947	3.5708e-4

Table I
FINAL MODEL PARAMETER

*D. **Logistic and Regularized Logistic Regression***

Naturally, we expect logistic regression (or the regularized version of it) to clearly outperform all other algorithms. However on cross validation, we got results lower than ridge regression (see table II) for all the hyper-parameters combinations tried so we decided not continue with this method. This is due to the long training duration that prevented us from finding the best parameters since logistic regression is computationally heavier than the other methods.

Methods	λ	γ	deg	iter	acc
Least Squares GD	NA	0.01	2	100	0.729
Least Squares SGD	NA	0.01	2	100	0.703
Least Squares	NA	NA	9	NA	0.829
Ridge Regression	0.00016037	NA	12	NA	0.833
Logistic Regression GD	NA	0.1	3	1000	0.733
Reg Logistic Regression GD	1.0e-6	0.1	4	1000	0.721

Table II
COMPARISON OF METHODS

Note that the λ and degree given in table II is the one for tx_1, this is for simplicity but those parameters were optimised on the 3 batches.

V. Results

Using ridge regression and the parameters shown in table I, we achieved an accuracy of 0.833 on AIcrowd.

Conclusion and future work

This project gave us a better understanding of the difference linear regression techniques that are used in machine learning. It also showed us the importance of data processing and feature engineering to reveal the meaningful information contained within the data. Overall, our model achieved great results both in terms of stability and accuracy going up to 0.823. The model is still not perfect and could probably be further improved by processing the data more; we chose not to remove outliers (as we opted mostly for regularised models) but this could be considered. Another idea would be to use more advanced machine learning methods like model voting over all our 3 different datasets or even by implementing a neural network which is possible using only Numpy (even though it would take a lot of time).