# PCML Project 1: Report

**Group #23**
Chenfan Sun
Valentin Le Tourneux De La Perraudi$\grave{e}re$
Ireneu Pla

***Abstract*—TBW**

## I. Introduction

This is the report for the EPFL Pattern Classification and Machine Learning course's first project: to observe the effect of the Higgs boson in some real datasets by using machine learning concepts learned in class.
The aim of the project was to build a binary classifier that predicts if each feature vector represents a signal (Higgs Boson decay signature) or only background noise.

## II. Models and Methods

### A. ML techniques

We first implemented six machine learning methods (as per project description). Each of these methods creates a model using the training data available.

The first three functions minimise the Mean Square Error (MSE) cost function:

$$MSE(w) := \frac{1}{N}\sum_{n=1}^{N}\left[y_n - f(x_n)\right]^2$$

Here is a description of the implemented Machine Learning methods:

Gradient descent

`least_squares_GD(y, tx, initial_w, max_iters, gamma)`: this method is a simple gradient descent, i.e. at each step we adapt each weight to minimise the loss. This is done by computing the gradient of the MSE cost function, and then updating the weights by following it. We can tweak this method using the gamma parameter, which corresponds to the step size. With a large gamma, we risk to exceed the minimum, and then oscillate around it, but with a small gamma, the descent may be very slow, and then very long to compute.

Stochastic gradient descent

`least_squares_SGD(y, tx, initial_w, max_iters, gamma)`: This method is very similar to the one above. Instead of calculating the gradient for every dimension, it is only calculated on one randomly chosen dimension. In expectation, we get the same gradient direction as the previous method, but for a much lower computational cost. This method, however can quite inefficient, so we chose to implement an intermediate version, which is the mini-batch SGD : rather than computing the gradient on all or just one dimensions, a random subset of dimensions is used. This way, the method is more efficient compared to the classic gradient descend and it is robust enough (depending on the size of the mini-batch).

Least squares

`least_squares(y, tx)`: This method implements the normal equation, which gives the optimal solution of the minimum loss:

$$w^* = (X^\top X)^{-1}X^\top y$$

Ridge regression

`ridge_regression(y, tx, lambda_)`:

Logistic regression

`logistic_regression(y, tx, initial_w, max_iters, gamma)`:

Regularised logistic regression

`reg_logistic_regression(y, tx, tambda_, initial_w, max_iters, gamma)`:

### B. Feature processing

Besides model selections, feature engineering on the raw data is also a key part that contributes to a better classification result. A comparison was done for classification on raw data and feature engineered data, with respect to three models, least_squares, ridge_regression and reg_logistic_regression. The results across all three models showed that a better classification result could be found through feature engineered data. This section divides into three subsections: data splitting, standardising data, additional features.

*1) Data splitting:* After a close examination on the data, PRI_jet_num is a special feature that only contains 4 possible integer values. Besides, a lot of the features varies with respect to PRI_jet_num. For instance, if PRI_jet_num is 0, then only 18 features will be left while all others will either be 0 or -999, which are meaningless or undefined. Splitting the data with respect to this feature, and train different models accordingly seems like a good idea. The data is split into three parts, the first part is when PRI_jet_num equals 0, which the corresponding data is left with 18 useful features. The second part is when PRI_jet_num equals 1, which leaves 22 useful features. The final part contains PRI_jet_num with value either 2 or 3. After splitting the data, we choose to standardise the data first instead of go directly training them.

*2) Data standardising:* As mentioned above, since the training data is split into three parts, get rid of the useless features is considered as part of standardisation. Only meaningful features are kept. For instance, when PRI_jet_num is 0, only 18 features from the training set are kept. The combination of data splitting and standardisation reduce the amount of outliers (-999) in the overall training dataset. After the above feature selection, we normalise the data to be mean 0, standard deviation 1. The mean is calculated without the outliers (-999). The outliers are filled with the mean. The standard deviation is normalised after the above two procedures.

*3) Additional features:* Besides the original features that are included in the dataset, experiments is conducted using additional random features as well. Specifically, we add the sin and tanh operations on the original dataset as additional features, and trained afterwards. Experiments were conducted after the splitting and standardisation of the data. The result showed that the additional feature did not improve the quality of classification. Thus, the additional features experimented were not used.

Overall, only a limited feature engineering efforts were performed but better results were found. From a result perspective, we increased from 0.77 to 0.81 on the leaderboard with feature engineering. This also included efforts on Gamma search as well, which will be introduced in the next section.

### C. Gamma search

Through our implementation and experiment, we found that the step size is a very important parameter through the training, especially with regularised logistic regression. If Gamma is too big, then the chance of converge is hard to tell, as the function might always go to the other side of the convex function. If Gamma is too small, then the convergence will definitely conduct, but the efficiency will be low. Gamma problem can be solved with Newtons method, as Newtons method is considered as a complement to a fixed Gamma value. However, due to our data size, the step to compute the hessian is too memory consuming and takes too much time. Thus, Newtons method is not used in the implementation. A work around is using the Armijo Rule, which is a unsound mathematics theory but works well in practice. However, the Armijo Rule is easy to understand but hard to implement. Experiments were done with several versions of Armijo Rule implemented. None of them actually worked as expected. Then we thought of a very naive way to achieve a dynamic Gamma value. Instead of using all those complicated algorithm, if the current loss value is larger than the previous loss value, then Gamma is reduced to be 2 / 3 or the current Gamma size. This method enforces the convergence. It might take some iterations to converge to a reasonable result, but our implementation found that this works the best with regularised logistic regression. We also tried to increase the speed of convergence by increasing Gamma, but it was not working as expected, as increasing Gamma to a reasonable value at any stage was different, which might influence the convergence a lot. Thus, increasing the Gamma was not used in our implementation. Dynamic Gamma value gives us a better indication on how well we are doing, as it helps us to find a solution that are relatively close to the exact result for logistic regression.

## III. Results

## IV. Discussion

## V. Summary