# PCML Project 1: Report

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

## 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. Each of these methods creates a model using the training data available.

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

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

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_)`: This method is essentially like least squares except a weight is introduced into the equation to penalize complex models that could have a tendency to overfit.

Logistic regression
: `logistic_regression(y, tx, initial_w, max_iters, gamma)`: This method is designed for categorical classifications, e.g. a binary classification as is the case in this problem. The goal is reducing the loss from the cost function, to fit a logistic function in the data.

Regularised logistic regression
: `reg_logistic_regression(y, tx, tambda_, initial_w, max_iters, gamma)`: This method is essentially like the one above it, except it is penalized so that the weights do not tend to infinity when doing classifications that could be resolved linearly.

### B. Feature processing

Besides model selections, feature engineering on the raw data is also a key part that contributes to a better classification result. We compared the classification results from some of the methods above on raw data and feature engineered data. We tested this using three models:

- Least squares

- Ridge regression
- Regularized logistic regression

The results across all three models consistently showed that a better classification result was found when using data with engineered features.

*1) Data splitting:* After a close examination on the data, we observed that the `PRI_jet_num` feature only took 4 possible integer values. Many other features depended directly on the value of `PRI_jet_num`. For instance, if `PRI_jet_num = 0`, then only 18 out of 30 features will be defined, and all others will be meaningless or undefined (0 or -999).

Splitting the data with respect to this feature, and training different models accordingly proved to be a good idea. We split the data three parts:

- `PRI_jet_num = 0`: 18 meaningful features
- `PRI_jet_num = 1`: 22 meaningful features
- `PRI_jet_num = 2 or 3`: all features defined

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, getting rid of the useless features is considered as part of standardization. Only meaningful features 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) and the outliers are replaced by the mean. The standard deviation is normalized after the above two procedures.

*3) Additional features:* Besides altering the features originally included in the dataset, we conducted some experiments using additional calculated features as well. Specifically, we added new features such as the $\sin$ and $\tanh$ of the original features. However, these additional features didn't improve the prediction quality when tested. Thus, they were not used.

Overall, our feature engineering efforts significantly improved our model. From a result perspective, they increased our prediction rate from 77% to 81%. These results reflect the efforts on Gamma search as well.

### C. Gamma search

Through our implementation and experimentation, we observed that the step size is a crucial parameter for the training, especially when using regularised logistic regression. If Gamma was too big, the convergence would be insufficient for a good model, and small Gamma values meant expensive computations.

The Gamma problem could be solved with Newtons method, as Newtons method is considered as a complement to a fixed Gamma value. However, due to the amount of data, computing the hessian needed too much memory and took too long. Thus, we didn't use Newtons method in our implementation.

Another option was to do a work around using the Armijo Rule, which can work well in practice. However, the implementation of Armijo Rule was no easy matter. Several tests were conducted, but none of them actually worked as expected or gave any significant improvement.

We decided to employ a naive way to define a dynamic Gamma value. Whenever the loss value was worse (larger) than in the previous step, Gamma was reduced to 2/3 of its previous value, to allow convergence. It might not be the optimal approach, but our implementation produced good results with regularised logistic regression. Having a dynamic Gamma value gave us a better indication on how well we are doing, and helped us to find a solution that are relatively close to the exact result for logistic regression.

## III. Conclusion