# Machine Learning - Project 1

Julie Djeffal & Loïs Huguenin-Dumittan & Fabien Zellweger
*École Polytechnique Fédérale de Lausanne - Switzerland*

***Abstract*—This report resumes the work and result of a prediction made from a dataset extracted from the Higgs boson machine learning challenge. Some feature processing has been made to the dataset, followed by the application of several models, to predict the data reasonably well. We explore multiple classification methods, parameter values and feature transforms to finally reach our most accurate model by doing average on data, scaling them, applying log, polynomial and square transformations and logistic regression.**

## I. Introduction

The purpose of this project is to predict with maximal accuracy if a set of features represents a signal or a background noise. To do this, two sets of data are provided by The Higgs boson machine learning challenge [1].

## II. Data Exploratory

The first data set is a train data set, composed of 31 variables with a total of 250′000 samples. The output variable $y_{train}$, is a binary variable that can take the values $'s'$ or $'b'$, depending on the output (signal or background noise). The other columns form the matrix $X$. One variable is categorical, taking value from 0 to 4, the 29 others are real values. Missing values are set to $-999$, replacing them with the mean of their respective column gave the best results.

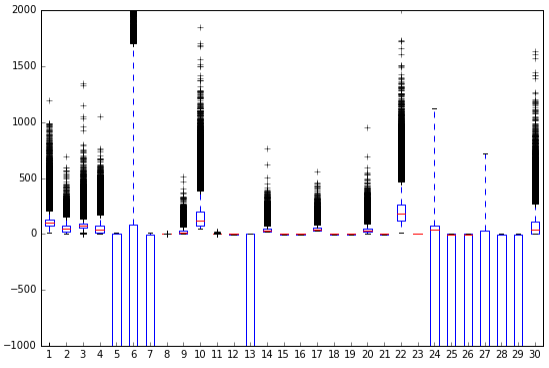


Figure 1. Boxplots of features

Figure 1 shows the boxplots of the features, we can see that the range of each variable vary greatly, thus we scaled the data.

## III. Models and Method

Six methods were implemented to complete the task:

1) *Least Squares by Gradient Descent (GD)* This algorithm approximates the weights that minimize the MSE cost function by computing the gradient and updating the weights at each step.
2) *Least Squares by Stochastic Gradient Descent (SGD)* Same as the previous algorithm, but the gradient is computed on batches of data and not on all training set.
3) *Least Squares* This method solves a linear equations system to find the best weights (i.e. $weights_{tr} = (X^T X)^{-1} X^T y_{tr}$).
4) *Ridge Regression* This method minimizes the MSE cost function but add a small amount $\lambda$ of a full rank matrix to $X$. This can reduce the multicolinearity of $X$ and provide a more stable model.
5) *Logistic Regression by SGD* This algorithm approximates weights that minimize a cost function that takes into account the binarity of $y$. It projects a prediction $x_n^T weights$ into the interval $[0, 1]$, amongst other things. As this method is slow and memory consuming, we use mini-batch.
6) *Penalized Logistic Regression by SGD* This method is the same as the previous one, but add a penalty factor $\lambda$ to the cost function. As for ridge regression, this allows to reduce multicolinearity problems.

We built several models, from the simplest to the most sophisticated ones. First, we simply tried the *Least Squares* and *Ridge Regression* methods on the scaled matrix $X$, then tried the cross validation method (CV) to estimate the test and train error (i.e. $RMSE$). This allowed us to have a reference error for the subsequent models. To obtain better models and take into account polynomial relations, we transformed $X$ by adding the square and cubic powers of all variables. We applied *Least Squares*, *Ridge Regression* and *Logistic Regression* to this transformed matrix, then performed CV on the first two methods. As *Logistic Regression* is slow to compute and converge (i.e. CV would have been difficult to perform), we trained on 70% of the data and tested on the remaining data to obtain $RMSE_{te}$.

In order to find the best model, we had the idea to add a transformation of a feature $f_1(x_n)$ only if $corr(f_1(x_n), y) > corr(x_n, y)$. For a second transforma-

tion, we add $f_2(x_n)$ if $corr(f_2(x_n), y) > corr(f_1(x_n), y)$ and so on (see *add_feature* in *build_polynomial.py*). Note that the order of the transformations is important here. After trying different transformations and permutations, we kept two models with this algorithm: the first one with transformations $log, P_2, sqrt, P3$ and the second one with transformations $log, P_2, sqrt, P_{3-7}$, where $P_i$ indicates polynomial transformation of degree $i$. Then we tried to apply all the methods on these two models, with CV for the first two and training on 70% of data for logistic regressions. For logistic regressions, we used the following parameters: $\gamma = 0.005, batch\_size = 3000$. We did not use *Least Squares GD* and *Least Squares SGD* since they approximate the best $w$ where *Least Squares* computes it analytically.

## IV. Results

The error obtained from applying the weights computed from a training set on a test set is noted $RMSE_{te}$. For the first model, we obtained a $RMSE_{te} = 1.02$ after applying the *Least Squares* algorithm with 4-fold CV. The matrix X is not ill-conditioned since the ridge regression method with CV gives a $RMSE_{te} > 1.02$ for all $\lambda$. However, the $RMSE_{te}$ approaches 1.02 as $\lambda$ goes to zero. When we apply a polynomial transformation of degree 2, we obtain a $RMSE_{te} = 0.95$ with *Least Squares* method but *Ridge Regression* does not give a better result (same as in the first model). So we can see that applying a square transformation to all features gives more accurate predictions.

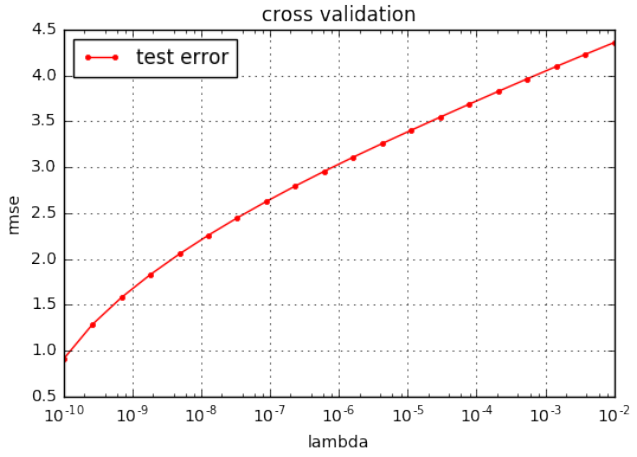


Figure 2. Boxplots of features

Now we give the results for our most interesting models, built with the algorithm described above. If we apply *Least Squares* with 10-fold CV to the first one, we obtain $RMSE_{te} = 0.884$, a great improvement in regard of the first models. With ridge regression, we can see on Figure 2 that the $RMSE_{te}$ is larger and goes down as $\lambda$ becomes smaller but *Least Squares* seems to give better results.

Then we applied *Logistic Regression with SGD* on 70% of this model's data, with maximal iterations = 750, as the algorithm converges around 700 steps. When we compute the error on the 30% remaining data we obtain $RMSE_{te} = 0.8586$ which is the lowest error we found and the best model we submitted on Kaggle. In order to improve the result, we tried *Penalized Logistic Regression* with $\lambda = 10^i, i \in \{-1, -3, -6, -10, -12, -15\}$. As for all the other models, adding penalization does not improve the error. With this technique, we obtain errors that are close (but not lower) to the best we got.

Finally, we implemented the last model, with the 7 transformations, which gives a large matrix. We applied *Least Squares* on it to find $RMSE_{te} = 0.88$, which is slightly better than the one found with the same method on the previous model. *Ridge Regression* gives no better error. We did not achieve to make this model work with logistic regression, as the computation was extremely slow and did not converge despite trying to vary the method parameters.

## V. Discussion

Our best model gives good predictions and is right more than 80% of the time. As we computed our model on 70% of the data and obtained a low test error we can assume that the model does not overfit the test data.

We see that our transformation algorithm works as expected. However, a better model could probably be found by trying more permutations of transformations. Higher degree transformations could also achieve a lower error with logistic regression, but more computational power would be required. It is interesting to note that adding a L2 penalty factor to our models did not give better results, which indicates that the features provided were not too highly correlated between each other. Also, we note that logistic regression gives better results than least squares method, as $y$ is a binary variable.

## VI. Summary

In this project, we were able to experience a dataset without any information about it. We tested methods and get confident about our prediction for regression and classification. We estimate our average test error (rmse) is on average 0.86 for the regression dataset. We estimate the average 0-1 loss for the classification dataset at 0.81702.

## References

[1] "Learning to discover: the higgs boson machine learning challenge," 2014.