Using Machine Learning Regression to understand the Higgs Boson dataset

Alleon Antoine
Computational Science & Engineering
Email: antoine.alleon@epfl.ch

Cleres David
Computational Science & Engineering
Email: david.cleres@epfl.ch

Lesimple Nicolas
Computational Science & Engineering
Email: nicolas.lesimple@epfl.ch

***Abstract*—The aim of this first machine learning project was to participate in an online competition based on one of the most popular machine learning challenges recently - finding the Higgs boson. In order to do so one was provided with the original data from the CERN. The team implemented a various range of machine learning functions based on different types of regressions (linear, least-squares, ...) with the help of the Python numpy library.**

I. Introduction

The Higgs Boson has many different processes through which it can decay. When it decays, it produces other particles. Nowadays, it is a great challenge to be able to predict their behavior. The subject of the Higgs Boson machine learning challenge was to try and improve on this analysis by promoting collaboration between high energy physicists and data scientists. In this project, basic regressions that were taught during the first weeks of the EPFL Machine Learning class. A major part of the project was to understand the provided data set. Six different regression methods were used after preprocessing of the raw data and using a k-folds cross-validation to choose the best parameter(s) for each regression method. A solid build-create-train-evaluate-predict-score pipeline was implemented to make the iterative process repeatable, and reliable. Finally, built-in feature analysis and engineering were developed to wrap up the entire development process.

II. Dataset

The dataset consisted of several thousands of samples containing thirty features such as particle mass and a number of jets. One of the aim was to transform the dataset in order to lead to the highest binary classification accuracy using different machine learning regressions.

Preprocessing of the data: The data from the CERN contained lots of missing data which followed a specific pattern depending on the number of jets described in the ”PRI-jet-num” feature. One could split the training data set into different subsets corresponding to zero jets, one jet, and multiple jets and train a separate classifier on each subset with a different number of initial dimensions. However, fitting convenient parameters was more difficult than excepted, especially due to the computation time required. Thus, the best results were achieved by replacing the missing data by various values (see Fig. 1). For instance, replacing them with a 0 value appeared to be the most efficient solution in comparison to the results that emerged for replacing by the mean or median of the selected features.

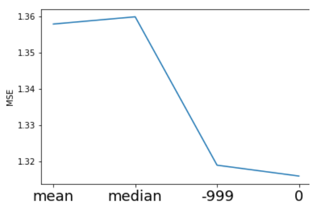

Fig. 1: MSE when replacing missing data by different values

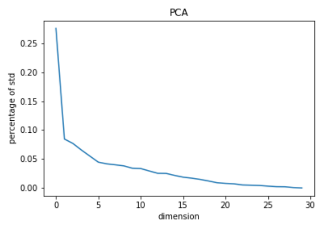

Fig. 2: Principal Component Analysis of the training dataset

Moreover, by using PCA, it was possible to assess which of the features were giving the most information to the model. Indeed, PCA showed that derived mass and label were strongly correlated. This observation was confirmed by the papers that the group read about the Higgs Boson. The PCA is then quite helpful to find parameters that would fit the high dimensional data well but based on a lower dimensional dataset. This process reduces drastically the computation time. In our case, only 8 dimensions were preserved from the initial 30 dimensions while keeping 70% of the relative cumulated standard deviation, see Fig. 2. Moreover, a polynomial basis was constructed based on the remaining dimensions and different combinations of powers of these dimensions. Finally the data was standardized by column as the different features did not represent the same quantities.

III. Methods

To satisfy the grading criteria of this first project six machine learning regressions were implemented. In these functions, two were **linear regressions** using either, gradient descent (GD) or stochastic gradient descent (SGD). The four other functions were **least squares regression**, **ridge regression**, both using normal equations, **logistic regression** using GD or SGD and finally a **regularized logistic regression** also using GD or SGD. These six functions were implemented with the only help of the numpy library as described in the rules of the project.

Tuning of the hyper-parameters: The most common method for selecting algorithm parameters is **grid-search**. In this project, Grid-search was performed by picking a range of values for each hyperparameter, trying out all possible combinations of these hyperparameters and selecting the one that was minimizing the loss function. However, the computational cost of this method is massive. In addition, by coupling a k-folds cross-validation on each hyperparameter in the grid-search, it is possible to obtain more information on the error as seen in Fig. 4, especially if the error overfits or not the dataset.

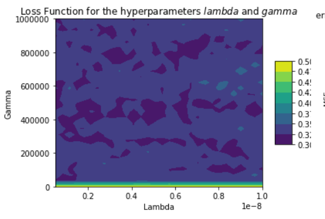

Fig. 3: Grid search for the Regularized Logistic Regression

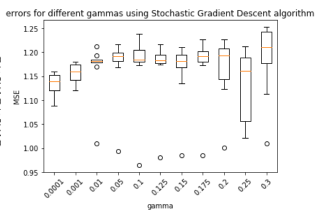

Fig. 4: boxplot of error for Stochastic Gradient Descent

Learning: In the first week of the project, the focus was trying all implemented regressions, starting with the simplest, most widely used and easily generalized to non-linear models, Linear regression. This model assumes a linear relationship between input and the output. Given data, we would like to find $w = [w_0, w_1, ..., w_D]$, such that the predicted output is: $y = Xw$ for any X. This is called learning. This is usually accomplished by optimizing a cost function. In the scope of this project, we have chosen two cost functions. The first one is the Mean Square Error, the second is the mean absolute error; in both cases, we wish to minimize the error.

The **Least Square** method, was the first linear regression that was used. The solution could be obtained explicitly, by solving a linear system of equations. This model was used to predict a label for an unclassified test point x_m: $y_m = x_m^T w^* = x_m^T (X^T X)^{-1} X^T y$.

Moreover, by increasing the polynomial degree of the feature vector we could build powerful linear models. Unfortunately, this leads to the problem of over-fitting. Regularization is a way to mitigate this undesirable behavior. The most frequently used regularizer is the standard Euclidean norm (L_2-norm): $\omega(w) = \lambda ||w||_2^2$. The main advantage of this regression was that large model weights were penalized (avoided), since they were considered "unlikely" in comparison to the smaller ones. By minimizing the cost function, we get: $w*_{ridge} = (X^T X + 2N\lambda)^{-1} X^T y$. Thus least squares is simply a special case where $\lambda = 0$.

The third and fourth methods were based on the gradient of the loss functions: **Gradient Descent** and Stochastic Gradient Descent methods aim to iteratively minimize the loss function to reach a local minimum. The difference between GD and SGD is that SGD creates and uses stochastic batches of training points instead of the whole training dataset. The weights for the gradient descent (GD) and subgradient descent (SGD) algorithms are given by the following update rule, at step t: $w^{t+1} = w^t - \lambda \nabla_n (w^t)$, where the gradient is based on subsets of the points in SGD. Because the target values are binary (either 0 or 1), the use of **Logistic Regression** makes sense. This technique uses the sigmoid function which given a real number, returns a value between 0 and 1 which corresponds to the probability that the point is classified as 1. The **Regularized Logistic Regression (RLR)** uses the same principle as the aforementioned method, adding a penalty term to its cost function, as to penalize large weights. Once the regression weights are determined, we can predict from the test dataset the probability of each point belonging to class 0 or 1 using the sigmoid function, as was done with the training dataset.

IV. Results

The best score on kaggle was obtained using the regularized logistic regression where we replaced the missing data with zeros.

To obtain the best hyperparameters, a grid search was performed on the training dataset with fewer samples ($10'000$ instead of $250'000$) along with a 5-folds cross-validation to better determine the loss for each hyperparameter γ and λ pair. This allowed to save computing time without losing too much accuracy as the difference between the RMSE is about 1% with and without the whole dataset, while the time is nearly 60 times greater with the full dataset. To obtain the final weights w for the prediction, a 5-folds cross-validation was finally applied on the full dataset ($N = 250'000$ samples) using the best parameters found with the previous step. The best hyperparameters for this regression were obtained by setting $\gamma = 10^5$ and $\lambda = 3.73 \cdot 10^{-9}$, as shown in Fig 3. Then the final prediction for Kaggle submission was made for the testing dataset using the computed weights w.

Other techniques gave a higher error on the regressed testing set.

V. Discussion

One of the major drawbacks of the RLR was its high time consumption, especially as the number of data increases. On one hand, an alternative for very large datasets could be to use ridge, or least squares regressions as long as an amount of false positives or negatives is not extremely important. On the other hand, to make a compromise, the GD methods showed good accuracy with a lower computing time since the cost function $\mathcal{L}$ was simpler than the logistic regressions ones. However, the penalized regression gave us the best accuracy and enabled us to get a Kaggle submission score of 0.83794 at the end. Finally, the accuracy highly depends on the dataset one is regressing, some data are more easily regressed than others. The *My Little Pony* group also tried to implement some algorithm that aimed to take in consideration the nature of the data during the preprocessing. As already described in the *Dataset* paragraph, the data could be split into three different groups according to the number of jets. It was possible to reach an accuracy of 0.848 for the samples with $jets = 0$ and $jets = 1$ but it was too time-consuming to find the right hyperparameters for the two other groups.

VI. Appendix

References

[1] Aad, G., Abajyan, T., Abbott, B., Abdallah, J., Abdel Khalek, S., Abdelalim, A. A., … Zwalinski, L. (2012). Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Physics Letters B, 716(1), 1–29. https://doi.org/https://doi.org/10.1016/j.physletb.2012.08.020

[2] Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I. (2015). The Higgs boson machine learning challenge. NIPS Workshop on High-Energy Physics and Machine Learning, 19–55. https://doi.org/10.1088/1742-6596/664/7/072015

[3] Collaboration, T. C. M. S. (2014). Evidence for the direct decay of the 125 GeV Higgs boson to fermions. Nature Physics, 10(8), 557–560. https://doi.org/10.1038/nphys3005

[4] Ho Fai Wong, Wanda Wang, R. C. and Y. K. (2016). Higgs Boson Kaggle Machine Learning Competition. Retrieved from https://blog.nycdatascience.com/student-works/centaurs-higgs-boson-kaggle-competition/