The Higgs Boson Machine Learning Challenge

Charles Beauville - 283103
Data Science

Robin Debalme - 282406
Data Science

Théo Patron - 284555
Comp. Science & Eng.

***Abstract*—The goal of this project was to apply machine learning techniques to actual CERN particle accelerator data to recreate the process of "discovering" the Higgs particle. Using the different physical properties of the particles' collisions, we had to determine whether or not a Higgs boson was present.**

To solve this binary classification task we first preprocessed the data, by removing irrelevant features and transforming others, while splitting the dataset in relevant categories. We then used different regression methods, hyper-parameter tuning and cross-validation to find the most accurate model.

We surprisingly achieved the best results using *Least Squares* and *Ridge Regression*, but we think that some more computing power could have allowed us to better tune our logistic models.

I. Introduction

The Higgs boson is an elementary particle in the Standard Model of physics which explains why other particles have mass. It was discovered at the Large Hadron Collider at CERN in March 2013. To observe a boson, physicists at CERN smash protons into one another at high speeds to generate even smaller particles as by-products of the collisions. Rarely, these collisions can produce a Higgs boson. Since the Higgs boson decays rapidly into other particles, scientists don't observe it directly, but rather measure its "decay signature", or the products that result from its decay process. Since many decay signatures look similar, we want to estimate the likelihood that a given event's signature was the result of a Higgs boson (signal) or some other process/particle (background).

II. Dataset

A. Original

We had access to a training and a testing dataset. The training dataset is comprised of 250'000 samples composed of 30 features each, with the label for the detection ('b' for background, therefore no detection, and 's' for signal, therefore detection of a Higgs boson). The test dataset contains 568'238 samples with no labels. Each feature corresponds to a physical property of the decay signature of the different particles that resulted from the collision.

B. Preprocessing

1) Splitting the data: In [1] we can see that depending on the value of the feature *PRI_jet_num* (which is categorical), some other features are undefined. This allowed us to create 3 datasets depending on which features were undefined (those features are dropped afterwards). In the first column of these datasets there was still a non negligible number of -999 values and therefore we decided to split in 2 each of our 3 created datasets based on whether or not the value of the first column was undefined. This left us with a total of 6 datasets, with different numbers of features. We also thought to fill-in those values with the median of the feature, but the results were less accurate.

2) Data manipulation: Having separated the data in 6 distinct datasets, we then deleted the column corresponding to *PRI_jet_num*. We also deleted the columns that were entirely filled with the same number (0 or -999).

As the data was acquired through measurement machines, there could be some calibration errors and inaccuracies leading to outliers in some features. That's why we decided to delete the outlier values. We achieved this using the Chebyshev's inequality that states that for all distributions, the proportion of the data within $1/\sqrt{1-l}$ standard deviations from the mean is at least l. We then deleted the data outside $[\mu - \sigma/\sqrt{1-l}, \mu + \sigma/\sqrt{1-l}]$, with μ and σ the mean and the standard deviation. We chose $l = 0.98$.

By taking a look at the distributions of the features using histograms, we have been able to see that some features had no significant difference in distribution whether a particle is detected or not. As those features are unhelpful for a classification task, we remove them from the dataset. Below, you can see an example of a feature that have no difference in distribution (Fig. 1, left), and of one that have a significant difference (Fig. 1, right).

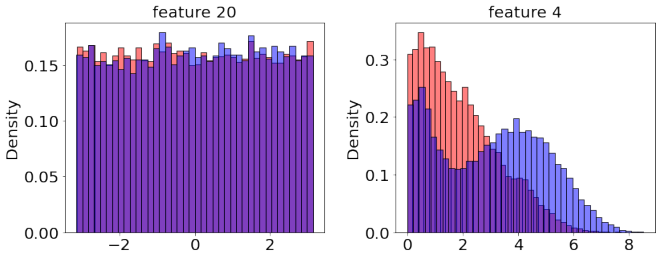

Fig. 1: On the left: no significant change between detection (blue) and no detection (red). This feature is then deleted. On the right: significant change between detection and no detection. This feature is kept.

Further looking at the distributions, we realised that some of the features' distribution had positive skewness (right long tail). Because machine learning models assume that the residuals (used to estimate the error) are normally distributed, we decided to add the unskewed distribution of these features to our data (as a new feature). To do so, we simply applied the

logarithmic function to the skewed features.
After doing all of that, we standardized our training and test sets using only the training set's mean and standard deviation. Finally, we performed polynomial expansion of varying degree and added a bias term (a column of ones). The polynomial expansion is performed only on the base features (not the previously added log features). The functions we use to preprocess the data are stored in the file *helpers_data.py*.

III. Methods

A. Implementations

We implemented the different methods required in the project description in the file *implementation.py*: *least squares gradient descent*, *least squares stochastic gradient descent*, *least squares*, *ridge regression*, *logistic regression* and *regularized logistic regression*.

To compute the loss of the logistic models without overflowing, we had to implement a component wise log-sigmoid function based on [2] and [3], which prevents NaNs from arising in calculations. Indeed, the numpy function `numpy.exp()` overflows, that is we get `numpy.log(numpy.exp(710)) = inf` instead of 710. Therefore to compute the log of $\sigma(x) = 1/(1+\exp(-x))$ we use the following log-sigmoid function:

$$\log(\sigma(x)) = \begin{cases} x & \text{if } x < -33 \\ x - \exp(x) & \text{if } -33 \leq x \leq -18 \\ -\log(1+\exp(-x)) & \text{if } -18 < x \leq -37 \\ -\exp(-x) & \text{if } 37 < x \end{cases}$$

For the same reasons as above, to compute the gradient of the logistic models, we also implemented a function that compute $\sigma(x) - y$ element-wise:

$$\sigma(x) - y = \begin{cases} ((1-y)e^x - y)/(1+e^x) & x \leq 0 \\ ((1-y) - ye^{-x})/(1+e^{-x}) & x > 0 \end{cases}$$

B. Hyper-parameter tuning

For hyper-parameter tuning, we performed grid searches on the different functions listed in the previous section. The grid searches allowed us to test the different parameters of the functions inside a given range and to determine which ones yielded the best results.

In order to find the best hyper-parameters for a method, we compute the accuracy of the model using the `compute_score` function that return the number of good predictions. Maximizing the score instead of minimizing the loss allowed us to increase the test accuracy obtained on aicrowd.com by nearly 3%.

We can visualize the results of our grid search with ridge regression by plotting the accuracy of the model on the test sample of our cross validation process in function of the hyper-parameters, as you can see in Fig. 2.

C. Validation

To be able to accurately estimate the performance of our models, and to avoid overfitting, we implemented 4-fold cross validation.

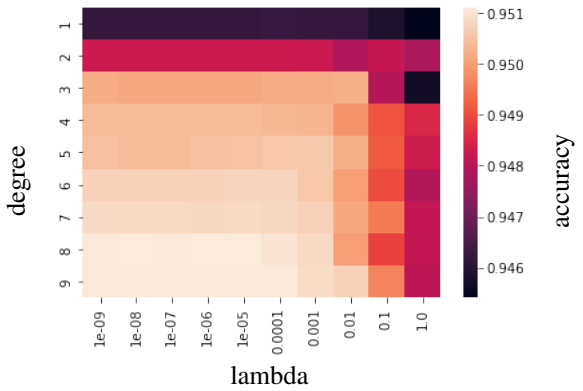

Fig. 2: Accuracy on the test sample in function of the hyper-parameters lambda and degree of polynomial expansion for the ridge regression grid search. Best result is achieved for degree = 8 and λ = 1e-5.

IV. Results

The results we will discuss in this section are those from the aicrowd.com challenge submission scores (accuracy and F1 score). Our best results for all the regression methods are summarized in Table I.

Regression method	Accuracy	F1 score
Least Squares Gradient Descent	0.787	0.662
Least Squares Stochastic Gradient Descent	0.741	0.569
Least Squares	0.828	0.741
Ridge Regression	0.828	0.741
Logistic Regression	0.815	0.687
Regularized Logistic Regression	0.815	0.687

TABLE I: Results of the different methods using the submission system of aircrowd.com

To further validate our methods we can compare our results for *Least Squares* when gradually adding the different techniques : when only separating the dataset and removing the -999 values, we had an accuray of 0.569 and a F1 score of 0.313. With the log expansion added and the removal of the irrelevant features, we observed an accuracy improvement of about 20%, with an accuracy of 0.755 and a F1 score of 0.605. Finally, with the polynomial expansion and the grid search, using cross-validation, we achieved an accuracy of 0.828 (an improvement of about 7 %) and a F1 score of 0.741.

We were expecting *Regularized Logistic Regression* to perform best, as it is better suited for binary classification tasks, but we think that a lack of computing power did not allow us to do a more extensive grid search and to find the best parameters. More computing power would also have allowed us to add l (used for outliers removal) as a hyper-parameter.

From the table, we also see that we have the same scores for *Ridge Regression* and *Least Squares*, this is because we had the best *Ridge Regression* results using a very small regularization factor, which essentially makes the model equivalent to *Least Squares*. The same happens for *Logistic Regression*.

Finally, we reached a great accuracy of 82.8%.

REFERENCES

[1] Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kégl, B., Rousseau, D.. (2015). *The Higgs boson machine learning challenge*. Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning, in Proceedings of Machine Learning Research, 42:19-55.

[2] Mächler, Martin (2012). *Accurately Computing* $log(1 - exp(-|a|))$ *Assessed by the Rmpfr package*. The Comprehensive R Archive Network

[3] Fabian Pedregosa, Bart van Merriënboer (2019). *How to Evaluate the Logistic Loss and not NaN trying*. Retrived from http://fa.bianp.net/blog/2019/evaluate_logistic/