Machine learning for the Higgs Boson Challenge

Florence Osmont, Benoît Müller, Armelle Hours
Master students at EPFL, Switzerland

***Abstract*—Using machine learning on a dataset from the CERN, we aimed at predicting if a collision event between two protons resulted in the creation of a Higgs boson. To do so, we first cleaned and modified our data and then tried different models and methods. In this report, we present gradient descent, stochastic gradient descent, Ridge regression, logistic regression with stochastic gradient descent (not regularized) and regularized logistic regression with gradient descent. In order to optimize our different methods, we used cross-validation to choose our hyper-parameters. In the end, we found that our best model is Ridge regression, with an accuracy of 0.776.**

I. Introduction

In this project, we want to recreate the validation of the discovery of the Higgs boson. The Higgs boson is detected by smashing protons onto one another which leads to the creation of smaller particles and, by observing their decay, we can see if a Higgs boson was created or not. In this project, using supervised machine learning, we will predict if a certain decay signature of a collision event was 'signal' (Higgs boson) or 'background' (something else).

II. Models and methods

A. Preparing our data set

From our train dataset, we extracted the train input, a collection of vectors sample where each one represents a collision event. We extracted the train output containing strings 'b' and 's' indicating if the collisions in the train input were respectively a background or a signal. Our first step was in the train output to replace the signal values by 1 and the background values by 0 or -1 depending on the context.

To better understand the train input, we visualized all its features using box plots. We realized the data was skewed for some features and that there were many outliers. Thus, we chose to keep and treat them, instead of deleting them which would have lost too much information. We noticed a feature seemed to be categorical but decided to treat it like the others at the moment.

In the train input, missing values were encoded by -999 so we changed their representation to avoid them acting as outliers and leading to miscomputations. At first, we thought about erasing them but the dataset is made of circa 21.067 % Nan values, so that would've made us lose too much information. Thus, we decided to replace all missing values by the median of the column it lies in as it is robust to outliers. For consistency (to insure our train and test input have the same distribution), we replaced the missing values in the test input by the median of the corresponding feature computed for the train input.

Lastly, we normalized our data so that our algorithms would perform better on them. Indeed, normalizing puts them all on the same scale. And again for consistency, we also "normalized" the test input by subtracting and dividing it by the mean and the std of our train input.

B. Models

Let X be a vector of data features, Y' the vector approximating the labels and w the weight parameter. We considered three different models: Model A, a simple linear model without biais, Model B, a polynomial expansion of model A of degree n to be chosen where the exponentiation is element-wise, and Model C, a logistic model with a biais.

Model A: $Y' = Xw$
Model B: $Y' = X^n w_n + \cdots + Xw_1 + w_0$
Model C: $Y' = \sigma(Xw + w_0)$

We then turned Y' into the vector containing the label encoded by integer values Y.

C. Methods

NB: If we write a hyperparameter was found, it means it was found using cross-validation on 5-folds (see II-D)

Methods for the linear model A: The linear model methods we implemented are gradient descent and stochastic gradient descent.

In both of them, we used an arbitrary initial weight vector and had stopping criterion on both the number of iterations and the norm of the gradient. For gradient descent, we chose a maximum number of iterations of 100 and we found an optimal step hyperparameter of value 0.27. In stochastic gradient descent, we chose to increase the maximum number of iterations to 600 as it does smaller steps, thus needs more to converge. Our optimal step hyperparameter found was 0.007. Then, for both, we computed the label with the sign function.

Methods for the polynomial model B: The method we used to select a polynomial model is the Ridge regression method because it has fast computation and regularization.

The linear system in ridge regression has been solved with the Cholesky decomposition which is behind the Numpy function numpy.linalg.solve and the label was computed with the sign function. It rapidly gives us a good first result.

method name	computed accuracy	AI crowd's accuracy	F1	hyper-parameters
gradient descent	0.72	0.697	0.661	step-size=0.27
stochastic gradient descent	0.687	0.644	0.612	step-size=0.007
Ridge regression	0.78	0.776	0.647	degree=3, lambda=2 e-6
logistic regression with stochastic gradient descent without regularizer	0.74	0.69	0.39	step-size:0.08
regularized logistic regression with gradient descent	0.74	0.71	0.363	step-size=0.2 lambda_reg=0.01

Table I
ACCURACY AND HYPERPARAMETERS OF OUR METHODS

We found optimal degree and regularization term hyperparameters of value 3 and $2 * 10^{-6}$.

We didn't try other methods on this model as computing the gradient for powers of X was very high consuming and easily created inf or Nan values.

Methods for the logistic model C: We first applied stochastic gradient descent on the model. The setting was the same as in linear model A. We found a step hyperparameter of 0.08 and chose to do 1700 ± 300 iterations.

Then, we did a regularized gradient descent. We found a step size hyperparameter of 0.2 and a regularization hyperparameter of $\lambda = 0.01$. The number of iterations chosen was 5000 with the same stopping condition and initial parameter as before.

Eventually, we did some changes to try to improve the accuracy. First, we took a dynamic step parameter of the form $1/(iter + 1)$ at first, then of the form $1/\sqrt{iter + 1}$ with $iter$ the counter of the current number of iterations. Additionally, we tried to take a random normal distribution for the initial weight, and to take the final weight w as the mean or the median of the ones found by running the regularized gradient descent multiple times. None of this made improvement on our computed accuracy. The label were then obtained with a threshold. By default, we used 0.5. After looking at the distribution of 0's and 1's in the train output, it made sense to try a threshold value around 0.6. But it didn't change the accuracy, so we kept the one by default.

D. Research protocol and measurements

The best hyperparameters of each of the methods were found using cross-validation on 5-folds. The hyperparameter was then selected as the one minimizing the loss function associated while taking into account risks of overfitting or mistake made by noise error.

To compare the different methods and models, we separated the train data into a training part (90%) and a validation part(10%) (we also tried an 80-20 ratio but it made no significant change). The separation was done randomly using seed. Then, we trained the parameter on the training part and computed the accuracy on the validation part. Notice that the data was prepared as described in II-A.

III. RESULTS

All our results are resumed in I. The "computed-accuracy" is computed as in II-D to select the model. The AI-accuracy and F1 score are the ones given by AI-crowd on the test set. They are given for reference's sake as we aren't selecting our model based on them to avoid overfitting on the test output.

After computing the accuracy of our models, the one with the highest one turned out to be Ridge regression with a score of 0.78. The rapidity of execution allowed us to do multiple tests and precise hyperparameter selection with cross-validation. We obtained a mean square error (MSE) of 0.315. Our Ridge regression submission on AI-crowd also got an accuracy of 0.776.

IV. DISCUSSION

In the gradient descent methods, we could've used a time criterion, since our number of maximum iteration is often reached or we could have found the optimal step with line search.

The limit of the Ridge regression method and the gradient descent method is that they optimize the mean square error, which is not directly linked to the accuracy. On the same note, when doing cross-validation, we could've used the accuracy of our hyperparameter as criterion instead of its associated maximum likelihood estimator.

Moreover, we could've done more specific data processing. Indeed, we noticed that the presence of -999 was sometimes related to the value of another indicator feature.

V. SUMMARY

To summarize, after cleaning our data by replacing -999 values by the median, we applied gradient descent, stochastic gradient descent, Ridge regression, logistic regression with stochastic gradient descent without a regularizer and regularized logistic regression with gradient descent on it. We used cross validation to find our parameters using the MLE as criterion. We also tried polynomial fitting where it was suitable. We tried playing around with the threshold, changing the size of the step per iteration, taking the mean or median of the weights... Some issues we encountered were that some methods took a very long time to run, the norm of our gradient was very large from the start so we sometimes couldn't try polynomial fit... In the end, our optimal hyperparameters and the resulting model accuracy per method can be found in Table I. We found that Model B with Ridge regression was the optimal one with hyperparameters degree=3, $\lambda = 2e - 6$ and computed accuracy 0.78.

References

[1] C. G. I. G. B. K. D. R. Claire Adam-Bourdariosa, Glen Cowanb, “Learning to discover: the higgs boson machine learning challenge,” 2014.