Project 1 - Higgs boson binary classification

Clémence Barsi (CS), Pauline Conti (DS) and Younes Moussaif (GM)
clemence.barsi@epfl.ch, pauline.conti@epfl.ch, younes.moussaif@epfl.ch

***Abstract*—The objective of this project is to test different models with pre-processing steps and find the one that performs the best on a specific dataset. This project uses solely the NumPy library. An accuracy of 82.9% has been achieved using ridge regression.**

I. Introduction

The Higgs Boson is a particle that has a very important scientific significance as it explains why other elementary particles have mass. In short, detecting this particle requires a large number of sensors, to measure a lot of raw data, analyse it and find signatures of its decay. Machine learning methods shall be used to attempt and detect said decay.

II. Exploratory data analysis & preprocessing

The Dataset contains 30 features that describe processes resulting from the decay of the Higgs boson particle. As stated in the documentation, features can be undefined (in which case the value -999.0 is used). Apart from the ones in `DER_mass_MMC`, all the invalid values depend on the value of the only categorical feature `PRI_jet_num`, which can take 4 different values.

First and foremost, we obtained the basic statistics (mean, median, standard deviation, minimum and maximum) of all the features. We then looked at the distributions of the features which are in Figure 1. As we can see, many features have a very high proportion of values equal to -999.0.

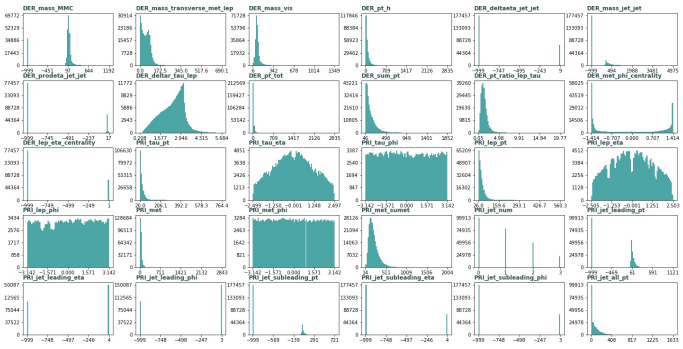

Figure 1. Histogram plots of all 30 features over the training set

1) Handling of invalid values by data splitting: Our first focus was then to handle the many invalid values. In the challenge's documentation, it was mentioned that most features were defined depending on `PRI_jet_num`'s value. Accordingly, we decided to divide the dataset into three separate sub-datasets depending on said value (`PRI_jet_num` = 0, `PRI_jet_num` = 1 and `PRI_jet_num`= 2 or 3). After the separation, almost all features depending on it were either fully defined or undefined, allowing us to simply remove the undefined columns. However, `DER_mass_MMC` still presented some invalid values (15.25% of the data points), that we chose to replace by the feature's median (computed when they are not taken into account). We chose the median over the mean as it is more robust to outliers. There were no Null or NaN values in the data, hence no further cleaning was needed.

2) Removal of redundant features: After plotting a correlation matrix heat map on each sub-dataset, we realized that some features were very similar. Hence, we chose to remove one of the two features when they had over 95% of correlation. This allowed us to limit the number of features the models were trained on, and as a result, slightly reduce the risk of overfitting by limiting the variance of results from one training to the other. This gave us the final number of features for each sub dataset, summarized in the following table.

Number of features in each sub dataset after preprocessing

	PRIjetnum value	# valid features	# final features
1	0	18	17
2	1	22	21
3	2&3	29	28

3) Standardizing the data: Finally, after removing redundant features, we normalized column-wise the data by subtracting the mean and dividing it by the standard deviation for each feature: $\frac{x_{in}-mean_i}{std_i}$ (i=feature index and n=datapoint index)

III. Presentation of all implemented models

1) Linear Regression with gradient descent: Linear regression tries to find a linear relationship between the inputs and the outputs. The gradient descent (GD) optimization algorithm tries to find a local minimum by using the gradient (slope of the tangent of the function) at each step. The update rule is : $w^{(t+1)} = w^{(t)} - \gamma * \bigtriangledown L(w^{(t)})$ where γ is the learning rate.

2) Linear Regression with stochastic gradient descent: This model is still a linear regression but uses stochastic gradient descent (SGD). Whereas in GD we use all the training set's samples, in SGD we use only one randomly selected data point to update the weights at each iteration. The update rule is : $w^{(t+1)} = w^{(t)} - \gamma * \bigtriangledown L_n(w^{(t)})$

3) Least squares with normal equations: This method tries to find the optimum of the cost function in our case MSE by solving a linear system of equations (i.e. normal equations). The weights to achieve the optimal cost are computed as follow : $w^* = (X^TX)^{-1}X^Ty$ where X contains the data points and y the data labels.

4) Ridge Regression: Ridge regression uses a regularizer along MSE Loss to mitigate the effect of overfitting. To do so it penalizes large model weights by adding the square of the norm of it to each weight. $w^* = (X^TX + \lambda' I)^{-1}X^Ty$ It has the same cost function as least squares plus a L_2 regularization term.

5) Logistic regression with negative log-likelihood: Logistic regression is used to transform the predictions into a probability distribution by using the sigmoid funcion.

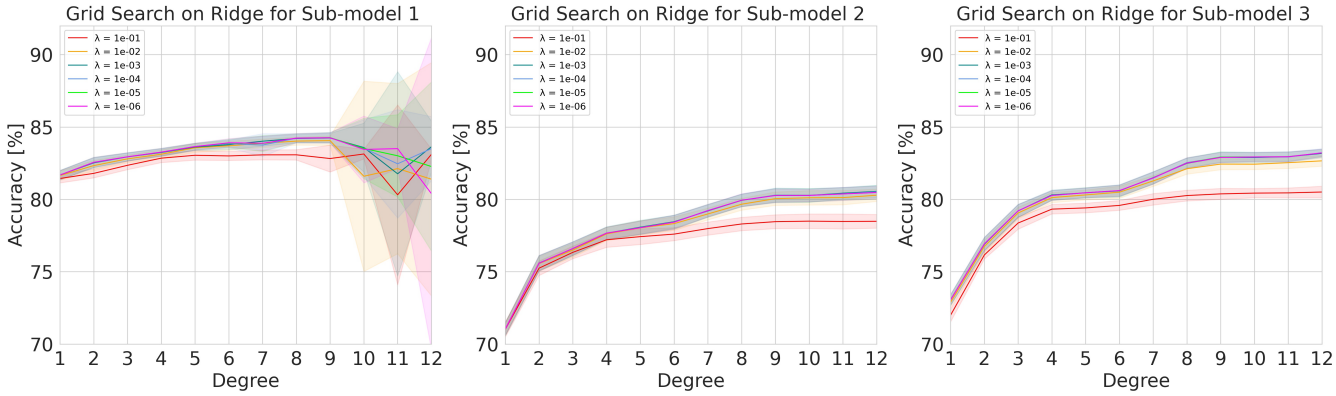

Figure 2. Mean accuracy (± std) of the three models obtained by 10-fold cross-validation in function of the degree of polynomial expansion. Note: The y axis goes from 70 to 92% (for increased visibility of the superimposed areas)

6) Penalized logistic regression with negative log likelihood: A version of logistic regression but with the L_2 regularization term $\lambda\|\mathbf{w}\|^2$ in the cost function. A higher λ might make the model underfit, decreasing it can make it overfit.

IV. Description & fitting of final model

A. Model description

Since the dataset was separated in three, three models were trained on each sub-section of the dataset. We first applied the same preprocessing on each sub-dataset as presented previously, before performing independently the hyperparameter optimization with cross-validation on each sub-dataset in order to optimize the results for each of them. Once the best hyperparameters were chosen, we trained each sub-section with them, and obtained their respective predictions. The final predictions for our model were simply the predictions of the three-submodels put back together using their original indices in the complete dataset. Thanks to its regularization, we obtained the best results with the ridge regression model, as it helps counteract the overfitting caused by high dimensionality in the input data. We will explain our optimization procedure in more detail in the following section.

B. Model fitting, hyperparameter optimization

In order to select the hyperparameters, we performed grid-search on the model's hyperparameters. To be able to use the entirety of the training data, we used a 10-fold cross-validation which also allowed us to limit the impact of overfitting on our results. To measure a hyperparameter's performance, we used the mean accuracy across the folds as well as its standard deviation as it allowed us to monitor the variance of our parameters across different trainings. For ridge regression, we optimized two hyperparameters: the degree of polynomial expansion d, and the regularization term λ. For the polynomial degree, we performed grid search for $d \in range(1, 15)$. As of λ, we tried values $\lambda \in range(1e^{-1}, 1e^{-10})$, only changing the exponent from one value to the next.

We plotted the means and standard deviations of the accuracy over 10-folds for each d, λ pair in Figure 2. In order to visualize better, we showed only six values of λ and twelve degrees. The optimal parameters chosen are $(d_1 = 9; \lambda_1 = 1e-5), (d_2 = 12; \lambda_1 = 1e-3) and (d_3 = 12; \lambda_1 = 1e-4)$. It is observed that the standard deviations for these values are small. As the degree increases, for model 2 and 3, the accuracy doesn't get much higher, going past these values of d doesn't seem useful as the model would be very prone to overfitting.

V. Results comparison & interpretation

The Least Squares, Ridge regression and Regularized Logistic Regression models were evaluated on AIcrowd (test accuracy). Their train accuracy was obtained by averaging the 10 accuracies from the cross-validation. The values are summarized in Table I. The other models were not included in the comparison of the results as their train or test accuracy were not as satisfying. This can be explained by our use of polynomial feature expansion and the resulting high number of dimensions of our train data. Indeed, distances in big dimensions cannot be understood in the way we do in lower ones, causing weights to explode to very high values. As a result, regularization like L2 becomes a necessity when training on such input data. As for Least Squares, the normal equations should output an "optimal" solution so long as the problem is convex, which made the model more robust to the dimension problem.

Table I
Accuracies of the different models

model	Accuracy (%)	
	test (ai-crowd)	train (local)
Least Squares	82.8	82.563
Ridge Regression	82.9	82.764
Penalised Logistic Regression	71.3	71.433

VI. Conclusion

The best performing model is Ridge Regression, with very similar results to Least Squares. Our final test accuracy was 82.9% which is fairly satisfying given the high dimensionality of the train set. This project showcases perfectly how extended pre-processing is a crucial part of a machine learning model's pipeline, even more so than the actual model. In addition to that, it also shows how a certain algorithm can perform very poorly if it is not adapted to the learning problem or the training data's dimensionality.