The Higgs boson machine learning challenge
Project 1

César Descalzo B, Gabriel Juri Á, and Cao Khanh N.
Machine Learning (CS-433), School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne

***Abstract*—In this report, we present our results for the project 1 of the Machine Learning course. The task was to utilize classification techniques to predict whether a collision event can produce Higgs boson particles, given the measured "decay signature". In our work, we investigated a variety of classification techniques and experimented different approach to preprocess the data. Finally, we introduced a pipeline that can accurately predict 81.8% of the testing datasets with an F1-score of 0.727, using least squares method with linear equation.**

I. INTRODUCTION

The Higgs boson is an elementary particle in the Standard Model of physics which explains why other particles have mass. To regenerate the Higgs boson, physicists produced experiments by smashing protons at high speed. As the resulting Higgs boson decays rapidly, physicists could only measure the "decay signature" that results from the decay process. Based on the data collected, we estimated the likelihood whether a collision results in a signal of Higgs boson or just to a normal background phenomena. The data preprocessing methods are documented in the first section. Follow up is our methods of testing and selecting models. Finally, we present the quantitative outcomes of our experiments and the final predictions we derived.

II. PRE-PROCESSING

A. Data exploration

Two datasets were provided, training and test data, which contains 30 columns for both files, and one extra column for training data (output column). We first notice, in the original datasets, there are the presence of missing values represented by -999 values. We also recognized the existence of a categorical features, namely *PRI_jet_num*, suggesting there are four different categories within the data. We would name these four categories: group 0, 1, 2, and 3. By dividing the datasets into this four groups, we realized the patterns of the missing values. Most of the missing values are presented only in group 0 and 1, and the entities belong to group 2 and 3 are perfectly recorded. We decided to individually handle each group by separately processing features and building models. By doing so, we could safely remove the columns that contains only missing or uniform values ($std = 0$) across the whole group. The result was we dropped 11 columns from group 0, 7 columns from group 1, and none from group 2 and 3.

B. Data pre-processing

Having divided the datasets into four groups, we proceed with some typical data preprocessing steps. First, we realized that there are still missing values within the data even after the removal of the columns. We dealt with this missing values by replacing them with the median values of the corresponding columns, as the median value would be less sensitive to outliers. We then identified the columns that are highly correlated by using `np.corrcoef` to generate a correlation matrix and kept columns that only have correlation coefficient of less than 0.9. Note that, we kept track of these columns to apply the removal to the test datasets.

Afterwards, we removed all the rows that contain outliers to improve the generalization capability of our model. For a standard distribution, we followed the default value of 3 times `std` to detect the outliers. The final step was normalization using the standard normalization techniques with the `mean` and `std`. However, during our analysis, we found that normalization step reduced the accuracy for *least_squares* and *ridge_regression*. On the other hands, using raw data would sometimes lead to regression calculation overflow as weights exploded or vanished. We decided to take the non-normalization approach. Further details can be found on the models analysis part.

III. MODELS SELECTION

A. Methods

We implemented the six different methods as requested: *least_squares_GD*, *least_squares_SGD*, *least_squares*, *ridge_regression*, *logistic_regression*, and *reg_logistic_regression*. The implementation can be found on `implementations.py`.

B. Models Testing and Selection

We followed the following pipeline to select the suitable methods to build our models:

1) Test the pure methods with heuristic hyper-parameters to the raw and normalized datasets. Select the best performing method.
2) Tune the hyper-parameters for the selected method using cross validation.
3) Apply polynomial feature augmentation.

	Accuracy					
	least_squares_GD	**least_squares_SGD**	**least_squares**	**ridge_regression**	**logistic_regression**	**reg_logistic_regression**
Normalized	*0.6590*	*0.6784*	*0.6769*	*0.6769*	*0.7353*	*0.6523*
Raw	-	-	*0.7694*	*0.7694*	-	-

TABLE I. ACCURACIES OF DIFFERENT METHODS APPLIED

C. Cross Validation

We utilized cross validation to find the optimized hyper-parameter to our model. In our case, cross validation was used to determine the best lambda for ridge regression. the validation was a 5-fold, with four subsets being the train data and one being the validation data. The average accuracies across five trials was recorded to find the optimized lambda.

D. Polynomial basis

As we finished with the model selection, we realized that we can further improve our model by augmenting the features vector. The standard method was to build polynomial basis for each feature. We once again used 5-fold cross validation to visualize the effect of each degree between 1 and 12.

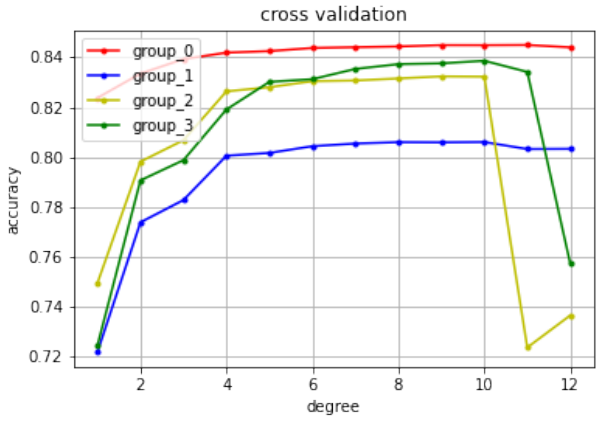
IV. MODELS ANALYSIS

A. Methods analysis

The accuracies of applying the methods to normalized and raw data was recorded in TABLE I. We saw that while the accuracies of logistic regression is superior to other methods, *least_squares* and *ridge_regression* achieved better result when applied to raw, non-normalized data. As these two methods are also efficient speed-wise, we decided to use them and applied directly to non-normalized data.

B. Cross Validation

We used cross validation to select our hyper-parameter for *ridge_regression* by simply selecting the best performing one. However, for polynomial basis, we had to select proper number so as to avoid overfitting. The accuracies resulting from each degree was recorded below.



Polynomial basis cross validation

We decided to choose a degree of 6, which is quite heuristic.

V. DISCUSSION

There were several challenges we met during our project. At first, we were unable to realize the categorical feature *PRI_jet_num*, leading to a number of trials and errors without actual improvement surpassing 0.74 accuracy benchmark. As soon as we discovered this pattern and divided our dataset, we experimented and implemented several methods of cleaning the data, such as eliminating correlation, removing outliers, and standardizing. At this point, the model achieved somewhere around 0.77 accuracy. As we were exhaustive in refining the data, augmentation came into our mind. With the help of polynomial basis, we were able to improve our model performance to achieve an accuracy of 0.828 on the training set. *Least_squares* and *ridge_regression* were our methods of choice since they are faster and somewhat more accurate as they directly solved linear matrix equations.

We also had the issue of calculation overflow during our experiments. While we believed our implementations are correct, with the large amount of data for training, we were unable to avoid overflow. Based on our educated guess, the overflow was because of vanishing gradient, which we did not handle within our implementation. This could be fixed by normalizing our data. However, for some reasons, our selected methods performed worse on normalized data. While we understand there is no guarantee that standardization would improve classification accuracy, we were surprised by the noticeable difference in performances. A possible reason is that standardization might have changed the distribution of the data, leading to worse classification accuracy.

VI. FINAL RESULT

For the final submission, we chose a polynomial basis of 6 to augment the features and applied *least_squares* to build our models. We achieved an accuracy of 0.8278 for the training set. We then made our predictions for the test set and submitted to *aicrowd* challenge. For the challenge, we achieved a decent accuracy of 0.818 with an F-1 score of 0.727.

The project provided us with an opportunity to understand theoretical machine learning model. At the same time, we had chance to participate in a data science challenge and gained practical skills that are need for a data scientist. It also paved our way to become more knowledgeable and efficient in handling future projects and studies.