Higgs Boson

Jacopo Ferro, Selima Khabthani, Matthew Dupraz
Department of Computer Science, EPF Lausanne, Switzerland

I. Introduction

The Higgs boson challenge aims to classify decay signatures of collisions resulting from protons-protons collisions, using a real-world dataset obtained from CERN, to predict if they correspond to a Higgs boson particle or not. The Higgs boson decays very rapidly into other particles, and the dataset consists of the observations of these remnants.

To come up with a good model, we first explored the data from various perspectives, used our findings to process the data accordingly, and then experimented to find the model giving us optimal accuracy.

II. Processing the dataset

The training set we are dealing with is composed of 250.000 samples and 30 features, each representing measures of physical quantities and some derived quantities selected by CERN physicists. The samples are labeled by s, standing for "signal", or b, standing for "background" according to whether the observation corresponds to the observation of a Higgs boson or not.

We processed the data in various ways in order to obtain a nice training set for our model.

A. Undefined values

The first difficulty this dataset presents is the presence of numerous undefined values. The nature of these undefined values is described in detail in the physical description of the project. In essence, most of the undefined features appear regularly in the dataset depending on the value of `PRI_jet_num`, which represents the number of measured jets. The feature takes values in $\{0, 1, 2, 3\}$, and in each case, we know exactly which features will be defined and which not. This naturally led us to split the dataset according to its values. One feature deviates from this rule, called `DER_mass_MMC` – this one can be undefined independently on the number of jets. Our solution was to again split the data depending whether this feature was defined or not. As a result, we split the data in 8 disjoint subsamples on which we train our model separately. For each of these subsamples we exclude the features we don't need and pick the best-suited hyperparameters.

One nice side-effect of this split is that in the case where `DER_mass_MMC` is not defined the model we trained was able to predict signal and background samples with very high accuracy. We discovered this correlation 'by hand' as well in our preliminary data exploration and visualization, which highlighted how an undefined value gave substantial information about the predicted label.

B. Outliers

To handle outliers, we implemented an algorithm which works similarly to the usual method for univariate datasets. We define

$$q = 1/2(1 - \sqrt[N]{1/2}),$$

where N is the number of features. For each feature, the number of data points lying between the quantiles q and $1-q$ corresponds to $\sqrt[N]{1/2}$ times the total number of data points. Hence if we filter the data points lying in this range in each feature, we are left with half of all the data. For any feature, we say a point is an outlier if it lies outside the range

$$[A - 1.5(B - A), B + 1.5(B - A)],$$

where A and B are the q-th and $(1-q)$-th quantile for that feature respectively. This method of finding outliers is entirely analogous to the classical IQR method for univariate datasets.

An advantage of this method, is that we do not need to assume much about the distribution of the data. Furthermore, quantiles are robust to outliers, unlike other estimators (like the mean).

C. Data Standardization

We standardized each sub-sample separately. This allowed us to take into account the fact that each of the sub-samples are different in nature. Furthermore we no longer had to worry about the influence of undefined values on the statistics (mean, standard deviation) as our sub-samples no longer had any undefined features.

We standardized the data using the usual formula

$$\tilde{x} = \frac{x - \mu}{\sigma},$$

where μ, σ are the mean and standard deviation of the respective feature.

D. Feature expansion

To add complexity to the model we expanded the features of the data with several additions:

- Powers of each feature of degree up to d.
- Products of pairs of features
- Square and cube root of each feature (sign is preserved)
- Exponential of each feature

Here $d \in \mathbb{N}$ is a hyperparameter, set to 10 in all but one subsample, set to 7 in our final Ideally, products of more than just two features would be added, but the resulting feature set would be too big to handle.

To find the optimal degrees we used for each subsample, we implemented a variant of grid search (see method `degree_search` in the file `run.py`).

E. Data augmentation

Since we split our data in 8 sub-samples, naturally the training set became much sparser. With all the added features this led to overfitting. During the training of the data we can apply regularization to tackle this issue, however it turns out to be a good idea to augment the data by adding new data points, which are obtained by perturbing existing data points by random noise.

To find the optimal amount by which we would augment the data, we implemented a variant of grid search (see method `lambda_search` in the file `run.py`). It turns out some of the subsamples were already dense enough as they were, and others had to be augmented considerably before being usable (even with regularization).

III. Models and Methods

During the "exploration phase" we were testing various regression techniques to see which one suited the data best.

We had trouble getting logistic regression to work, we were getting very low accuracy as well as very long computation times. Instead we set for ridge regression (and by extension least squares).

A. Model testing

To test our model, we used k-fold cross-validation. This technique proved to be very useful throughout the project as it allowed us to test the data without having to submit to the challenge website. The accuracies we estimated using k-fold cross-validation were very close to the actual values obtained after submitting, so we could safely estimate the strength of our model.

Through k-fold cross-validation we would obtain the average testing and training accuracies of our model. By comparing those, we could detect possible issues with our model, such as whether it is overfitting. To see this, we looked at the ratio

$$\frac{\text{testing accuracy}}{\text{training accuracy}}.$$

The closer this ratio was to 1, the less there was an issue with overfitting.

B. Model selection

We augmented the data as explained before, until we could get this ratio past a certain threshold with ridge regression (while maintaining the best possible testing accuracy).

Each time we augmented the dataset, we would look at the accuracy ratio for various values of the hyperparameter λ for ridge regression. The curve obtained would in general be concave with an apparent maximum, however there was some variation which made it difficult to estimate the best value of λ exactly. For example, take a look at Figure 1.

For this reason, the optimal values were selected by hand using the information gathered with the grid search for each sub-sample.

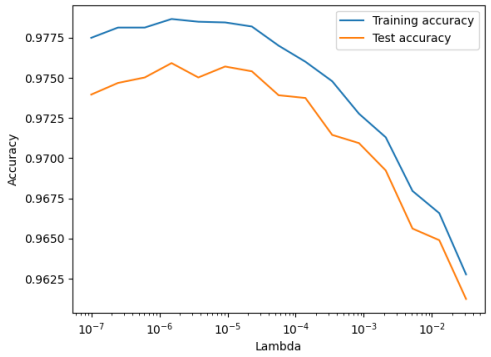

Fig. 1. Accuracy for different values of lambda, `(with_mass_MMC, jet_num) = (False, 3)`

IV. Conclusion

In the end, using our model we achieved an accuracy score of 0.828. While not the best, it is a good result, which shows the approach we have taken worked rather well. One key take away from this project is that the way we worked with the data was heavily shaped by the nature of the data. It turned out to be very important to understand the relationships between the features by reading the physical description as well as experimenting with the data ourselves.