Machine Learning CS-433 - Project I

Andrea Giovanni Perozziello, Henrik Øberg Myhre, Jurriaan Marcus Hubertus Schuring
Department of Computer Science, EPFL Lausanne, Switzerland

***Abstract*—In this project we predict the creation of a Higgs boson using machine learning methods. Data processing and feature selection were fundamental, both to improve the training process and to reduce computational costs. Various implementation techniques and the use of machine learning methods on real data allowed us to train a model and generate accurate predictions. This model can be used to predict the previous unknown occurrences of a Higgs boson by using features about the event.**

I. Introduction

Researchers at CERN tried to find Higgs bosons by smashing protons into each other with extremely high speeds. In these collisions, the protons break down into smaller particles. Sometimes a Higgs boson is one of these particles. However, the boson decays into other particles extremely rapidly and is therefore not directly detectable. Rather, the Higgs boson is found indirectly by looking at the "signature" of the decay event, meaning the combination of a variety of features which are measured in the collision event. In this report, we will present a machine learning approach to detect signal from a given "signature".

II. Data pre-processing

The first step is to properly process the data to improve the quality of training. The data itself consists of 30 features. However, for some measurements, certain features were not measured or were not calculable. Additionally, one feature is the number of jets created in a collision, either 0, 1, 2 or 3. For jets 2 or 3 all features could be calculated or measured, while jet 1 had less and jet 0 the least. It was therefore decided to split the data in two groups based on these features, since the usefulness of a feature for a prediction depends on the number of jets.

After this division we normalized the data so features of different orders of magnitude can be more easily used together in the training process. Furthermore, we removed columns with zero variance (for example unmeasured features in the jet0 and jet1 group). Then we replaced -999 values that remained by the median of each column. The median was chosen since we expected that replacement by the median would disturb the distribution of values in a column the least[1]. Additionally, it is fairly robust to outliers, as opposed to the mean. Finally, we removed columns with low correlation (less than 0.01) with the output to reduce computational complexity. This can be done since such lowly correlated features will barely/not affect the quality of the training. The result both with and without removing these features was the same. For an overview of the final set of features used in training, see **table** I.

Feature \ Set	1	2	3
DER_mass_MMC	✓	✓	✓
DER_mass_transverse_met_lep	✓	✓	✓
DER_mass_vis	✓	✓	✓
DER_pt_h		✓	✓
DER_deltaeta_jet_jet	✓	✓	✓
DER_mass_jet_jet		✓	✓
DER_prodeta_jet_jet	✓	✓	✓
DER_deltar_tau_lep	✓	✓	✓
DER_pt_tot	✓	✓	✓
DER_sum_pt	✓	✓	
DER_pt_ratio_lep_tau			✓
DER_met_phi_centrality			✓
DER_lep_eta_centrality	✓	✓	✓
PRI_tau_pt	✓	✓	✓
PRI_tau_eta			✓
PRI_tau_phi	✓	✓	
PRI_lep_pt			✓
PRI_lep_eta	✓	✓	
PRI_lep_phi		✓	
PRI_met			✓
PRI_met_phi			
PRI_met_sumet		✓	✓
PRI_jet_num			✓
PRI_jet_leading_pt			✓
PRI_jet_leading_eta			
PRI_jet_leading_phi			
PRI_jet_subleading_pt			✓
PRI_jet_subleading_eta			
PRI_jet_subleading_phi			
PRI_jet_all_pt			✓

TABLE I
Final features in each set

III. Data augmentation

We augmented the data by adding new features to the data in order to be able to train the model more. Firstly we added $\log(1+x)$ for those columns without any values smaller or equal than minus one. Some features have long tail distributions, applying a log to this will equalize these, which will improve the learning process. Therefore we decided to apply all following expansions also to the added log terms.

Secondly, we applied the square root onto all columns by using the absolute values of the elements in order to prevent imaginary numbers. Thirdly we added cross terms of the different columns. Fourthly, we added an offset column of ones. Finally, we added a polynomial expansion of degree 2 up to degree n. To find the right n, we ran an optimization algorithm (see V).

[1] "Feature Engineering Part-1 Mean/Median Imputation". by Arun Amballa, Analytics Vidhya, Medium. https://medium.com/analytics-vidhya/feature-engineering-part-1-mean-median-imputation-761043b95379

So in summary we added:

- $\{\log(1+x)\}$ for all with x all columns with all values bigger than -1.
  - $\{\sqrt{|x|}\}$
  - offset column of ones
  - $\{\sum_i^N \sum_{j=i+1}^N x_i x j\}$ with x_i and x_j being feature columns
  - $\{x^2, x^3, ..., x^n\}$ for all feature columns x.

Method \ Set	1	2	3
processed data	14	16	22
$\log(1+x)$	13	15	21
$\sqrt{\lvert x \rvert}$	27	31	43
offset	1	1	1
cross terms	351	465	903
polynomial	$27 \times (n-1)$	$31 \times (n-1)$	$43 \times (n-1)$

TABLE II
NUMBER OF COLUMNS ADDED PER METHOD

IV. Regression techniques

Since the training data sets have a lot of features (see **table** II) we expected that a penalized regression technique would produce better test results, since it penalizes high values in the model and therefore prevents any feature from becoming "over important". This will avoid over fitting, which would be a risk due to the high number of features.

Additionally, the problem at hand is a classification problem. Therefore we expected that a logistic regression would give better results than linear regression. However, after some initial tries, we saw that this was not necessarily the case. In the end we tried to optimize the parameters for both techniques and pick the most accurate one.

In our implementation of penalized logistic regression, we initialized w by taking random values from a normal distribution and dividing this by 1000. The rationale for this is that we hoped that we would not get stuck at the same local minimum all the time, since that happened when we initialized w by a zero-vector.

V. Optimization of hyperparameters

After initial testing we found that each subgroup has a different optimal range for their parameters (see table **table** III). With further tests we provided a specific range of hyperparameters for each subgroup in order to improve training and reduce time spent for optimizing. We tested both for logistic regression and ridge regression.

VI. Results

Ridge regression proved to be the best method to use for the classification problem. Using the optimization script to find the best hyperparameters, we found that the best degrees are: 3, 6 and 9 and the best lambdas are: 10^{-9}, 10^{-5} and 10^{-9} for the respective groups jet0, jet1 and jet2&3. Using these hyperparameters we created a model with an accuracy of 0.833 and an F-score of 0.746. The result was uploaded to Aicrowd.

Set \ Param	lambda	degree
1	logspace$(-14, -9, 7)$	linspace$(3, 9, 7)$
2	logspace$(-14, -5, 10)$	linspace$(6, 14, 9)$
3	logspace$(-14, -9, 7)$	linspace$(9, 17, 8)$

TABLE III
RANGES USED TO TEST FOR HYPERPARAMETERS

VII. Discussion

Using ridge regression and optimizing for lambda and the size of the polynomial expansion, we created a model which creates a decent prediction, with accuracy 0.833 and F-score 0.746. The difference between the accuracy score and the F-score seems to be due to a high precision rate of the model, but a lower recall rate. Meaning that the model correctly identifies signal from background, but it misses a lot of signal and mistakenly classifies it as background. This behaviour is less "visible" in the accuracy parameter since it also counts all the true negatives as a good result. And since the background is a large group (26%, 36%, 45% in the respective groups in the training data), the true negatives will "drive up" the accuracy number. For this problem, the F-score seems therefore to be a better measure. This effect is the strongest in the first group, with the lowest percentage of signal. Here, for a precision of 0.73, the recall is typically 0.58, so the difference is around 0.15. For the second group the difference is around 0.9 and for the last around 0.1. The difference seems thus to be caused by the different numbers of signal and background events.

While fine tuning the parameters, we noted that the lowest loss did not always correspond to the highest accuracy and F-score. For example, degree 3 and lambda 10^{-9} for jet0 gave an accuracy of 0.852667, F-score of 0.680209 and loss of 532. Degree 5, lambda 10^{-9} on the other hand had accuracy of 0.853168, F-score of 0.680879 and loss of 702269. So while the loss is way higher, the accuracy and F-score were better. We are not sure why this happened, but for future use it might be better to select the hyperparameters based on F-score or accuracy instead of the loss, since those measures are in the end the most important for the performance of the model.

We could not avoid getting stuck at local minimum for logistic regression. Solving this would probably improve the results because it generally works better as a classifier than linear regression. Especially since the data is dominated by background events, which is the strongest in the jet0 group. It is also in this group where we expect most improvement if a suitable logistic regression technique was found. However, we note that ridge regression still gives decent results, given that right preprocessing and data augmentation is applied.

VIII. Summary

In conclusion, this paper has shown that it is possible to classify Higgs bosons using different machine learning techniques. Many of the provided features are not important, as shown in **table** I. The research also highlights the importance of the F-score when the data is not evenly distributed between the classes.