Project 1: the Higgs boson machine learning challenge

Lazare Girardin, Mateusz Paluchowski and Radmila Popovic
EPFL, Switzerland

***Abstract*—This paper is to present process of building machine learning classifier for Higgs boson detection on a dataset provided by CERN. Exploratory data analysis to understand the dataset and features is included. Last but not least feature processing and engineering with use of machine learning methods on real data is described.**

I. Introduction

Dataset provided to us by CERN and distributed on Keggle platform consists of training set of 250,000 labeled events and test set of 568,238 unlabeled events. Both data sets contain 30 feature columns on which model can be trained and predicted. Feature columns consists of actual physical properties of particle physics and even though it is not required to understand these properties, it is worth noting that some variables are meaningless or cannot be computed. We have addressed this problem ad described our approach in next section.

II. Methodology

A. Preprocessing the data

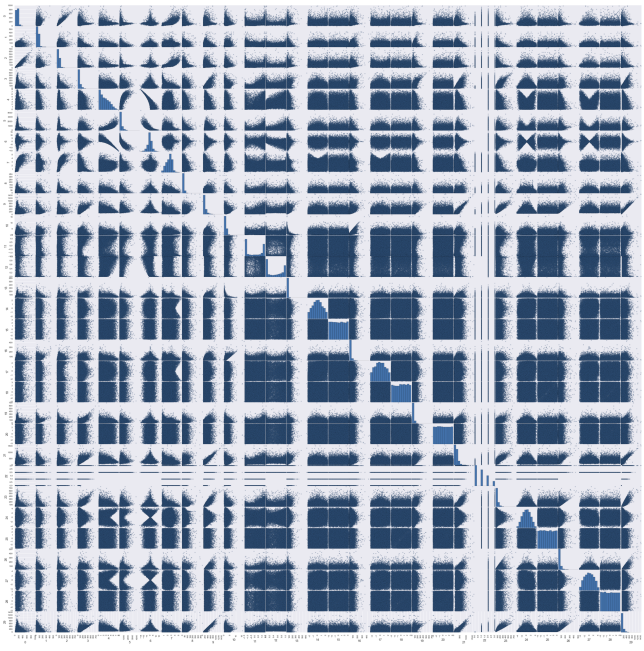

Figure 1. Scatter matrix of all features with feature histogram on main diagonal.

	0	4	5	6
count	42435.000000	14394.000000	14394.000000	14394.000000
mean	121.514904	2.402741	371.627493	-0.818595
std	57.121667	1.746793	393.423709	3.579605
min	10.499000	0.000000	18.593000	-15.783000
25%	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN
max	985.102000	7.971000	4543.913000	16.690000

Figure 2. Statistical analysis of few sparse features.

Reading the documentation and exploring the test data through statistic and visualization just as visible in Figure 1 leads to some early observations. Multiple columns are clearly correlated with one another as various patterns become visible. Statistical analysis partially presented in Figure 2 revealed the fact that set is not normalized and data points contain a lot of non-measured features, which are represented by -999 values. While advanced classifiers should be able to determine that these values shouldn't be taken into account, simple ones as least squares may not give good results, because they are extremely sensitive to outliers. Dealing with these non-defined characteristics is one of the first challenge of this project. We tried many different approaches in order to correctly address this problem. Our first approach which is probably most intuitive, but not necessarily most accurate, consisted of simple substitution of NaNs with 0 values. This approach did not yield a good result because as we can see in Figure 2 zero value is within set of values for some features, thus we are distorting our dataset. We also tried similar approach by filling NaNs with mean feature value but it failed for the very same reason as before. Unsurprisingly simple dropping of columns with NaNs is nor a right approach either, as there are 10 sparse columns in our dataset and it would cripple it significantly. Finally we decided to do preprocessing in two steps. Firstly we standardize by subtracting from each column its mean value and divide by its standard deviation. Secondly we fill our NaNs with distinct negative value but much lower magnitude, for example -5. Such approach yields significantly better results than any previously mentioned and unprocessed dataset, because classifiers can distinguish these unnormalized values while their impact as outliers is

reduced.

B. Dimensionality reduction or augmentation?

An eigenvalue decomposition (using for instance the SVD decomposition of the numpy package) reports that only the a few features contain almost all the variance of the data points. One could therefore think of performing dimensionality reduction such as PCA (Principal Component Analysis) to compute the classification on a lower dimension and reduce the computational cost. However, this doesn't yield any better results, and the projection on the PCA space is extremely dependent on how one treats the outliers mentioned in section II-A.
Another idea is to add some features based on the existing ones. As the computation cost comes mainly from the number of data points and not from the dimensionality of the data, this doesn't increase drastically the cost. Of course it depends on which algorithm is chosen as seen in class. But why adding some dimensions? The answer is that in higher dimensions, a linear classifier allows non-linear classification of the lower dimensions (an widely used example is the kernel trick). Choosing which dimensions should be used to create new features is another challenge. We addressed this problem by thoroughly analyzing Figure 1 and adding additional dimensions for features which appeared to be somewhat correlated with one another. As a result we ended up with dataset containing all 30 base features, an additional 240 features which are base features taken to the power of 2 to 9 and combinations of features. Such feature polynomial yields better results than raw dataset, however it should be noted that this tendency reverts if we decide to build higher order polynomial such as base features to the power of 10 or higher.

C. Classification and validation methods

- *Least Squares Regression* is one of the most simple regression in here, used to divide the data into two regions of the space in order to classify them. The loss function to be optimized is the square distance of points with respect to the dividing hyperplane.
- *Ridge Regression* uses basis function to construct better features and linearize the problem. In addition, a penalization term dependent on the L2-distance is added to avoid overfitting the data.
- *Gradient Descent* uses the derivative of the loss function to optimize the weights.
- *Stochastic Gradient Descent* randomly chose a subset of the data to perform the gradient descend in order to reduce the computational cost.
- *Newton's Method (Logistic Regression)* uses sigmoid function to transform the interval of the predicted values on a smaller interval corresponding to probabilities. An iterative process is used to minimize the negative logarithm of the likelihood with first and second derivative of the loss function.
- *Penalized Logistic Regression* is the same algorithm as described above but in which a penalty term is added to deal with linearly separable data.
- *K-Fold Cross Validation* is one of the most important tool when it comes to choosing or comparing methods. Performing a single evaluation of parameters does not hold any information over repeatability nor overfit of the classifier on the dataset. By performing the optimization on different train and test subsets of the data, one can then use the variance and mean of the results to assess the method.

III. Results

In order to obtain good results, two aspects had to be considered: engineering the features and choosing a prediction method mentioned in Section II. The best submission was obtained using a degree nine polynomial base and combining all the features which each other. The chosen method is the Ridge regression, which yielded good results and most importantly a small variance over 10 folds as compared to Least Squares or other classification methods.

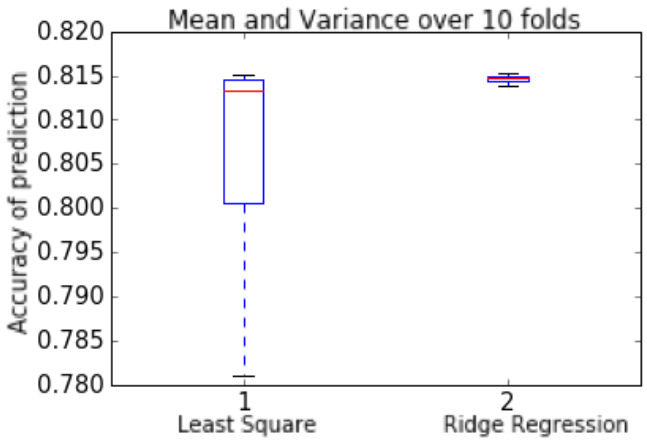

Figure 3. Boxplot representing the mean and variance of Least Square and Ridge Regression respectively for a feature matrix of degree 9 polynomial basis without feature combination

IV. Discussion

Since our result is not the top one on Kaggle leader board we could have done more. There is no significantly better method when it comes to train error, however not every one of them fits our task. We decided to use quite simple and reasonably fast Ridge regression, but probably more advanced and well suited classification method could have been used. What definitely could have been improved is feature engineering as it seemed to have most impact on final result. Additional effort in order to find the most meaningful features could have been made.