Maching Learning - Project 1

Pierre Thevenet, Buisson Nicolas, Arsalan Syed
Department of Computer Science, EPFL, Switzerland

I. Introduction

This report concerns the first project of maching learning. It consists in using knowledge from machine learning in order to solve a problem of the Higgs Boson Machine Learning Challenge. Data from the CERN is provided, with inputs and the corresponding outputs, and the goal is to find a way to build a correlation between input and output. The final test to establish a score will be done on an untrained dataset, to see if the team has found a way to build a good correlation model.

II. Data analysis

A. Initial data

The initial data given to us to train our models consists in two arrays of 250,000 elements. In the first array are inputs, with each inout having 30 features, with real values. The second array holds the corresponding output, which have a binary value: +1 or -1. The initial data is raw and was not preprocessed in any way. There are a lot of features having the value -999, which actually corresponds to an undefined value (in other words, this physical quantity was not measured during the experiment), so this will have to be sorted out after.

B. Pre-processing

a. *Undefined data*
As stated above, data with value -999 actually corresponds to missing information. In fact, as one can read in the Higgs Boson Machine Leanring Challenge description, we can split our data in 6 different classes (M0 to M5 in the code), depending on the missing information it has. So we decided to divide our model into 6 submodels, therefore depending on which variables are defined or not.

b. *Test data set*
We separated our training data into two sets, in order to exclusively train on one set and test on the other. This enables us to really check if we might not be overfitting too much, and to have a good first overview of our possible score when submitting on Kaggle. Of course we used all the training data before submitting to have a better fit.

c. *Normalizing the data sets*
We standardized both the training and testing datasets, as the values from the different features had different physical units. With this done on our dataset, we had a first clean base to work on.

III. Building other features

Our dataset has 30 features (a little less depending on the class), but we could have a better model with other features. We used in our model polynomials and also inverses, since some features might be better related to the output this way. In fact, this is a common way to proceed for simpler problems in physics, when someone wants to find a correlation in his experimental data: maybe there's a logarithm realtion between the input and the output, so we compute the logarithm of the input before trying to do a linear regression. In our model however, polynomials and inverses gave the best approximations.

IV. Ridge regression model

We used ridge regression to compute a solution for our data, with cross validation to run smaller batches. We used different parameters based on what was discused previously in order to choose the best model, based on its validation score (the lower the RMSE the better). We used the following formula for our ridge regression:

$$\min_{\mathbf{w}} \frac{1}{2N} \sum_{n=1}^{N} [y_n - \mathbf{x}_n^\top \mathbf{w}]^2 + \lambda ||\mathbf{w}||_2^2$$

The different parameters were the following:

1. $\lambda \in \{\log_{10}(-5), ..., \log_{10}(-2)\}$
The λ parameter helps to counter overfitting with a linear model, particularly when adding polynomials of high dimension. In the tests we've went through we could clearly see that with $\lambda = 0$ the validation data set had a much higher RMSE value than the train dataset. By testing different values for lambda, we could lower the error when evaluating the validation dataset.
2. *Polynomials:* $x_i^d, d \in \{1, ..., 8\}$
We tested different values for d in order to find a good value, keeping the same objective: minimizing the validation RMSE.
3. *Inverses:* $(\frac{1}{1+x_i})^d, d \in \{1, ..., 8\}$
For these new features, we decided to compute $\frac{1}{1+x_i}$ after testing different inverse functions because it gave us the best result. Like for polynomials, we tested this for different d values. It is to be noted that we only

applied this function to features which have exlusively positive values.

V. Results

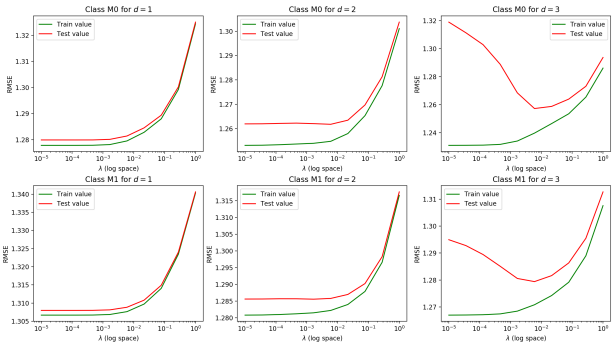

Figure 1. RMSE values depending on λ for classes M0 and M1 and degrees 1 to 3

a. *Small degree*

In the figure above we represented the RMSE values for two of the six classes (M0 and M1), with degrees $d \in \{1, 2, 3\}$ and their evolution with λ. Overall, the validation (or test) set (red curve) has a higher RMSE than the train set (green curve), which is of course to be expected. However it is interesting to see that for lower degrees, the best value is $\lambda = 0$ (as we can see on figure 2.), which actually corresponds to a least squares model.

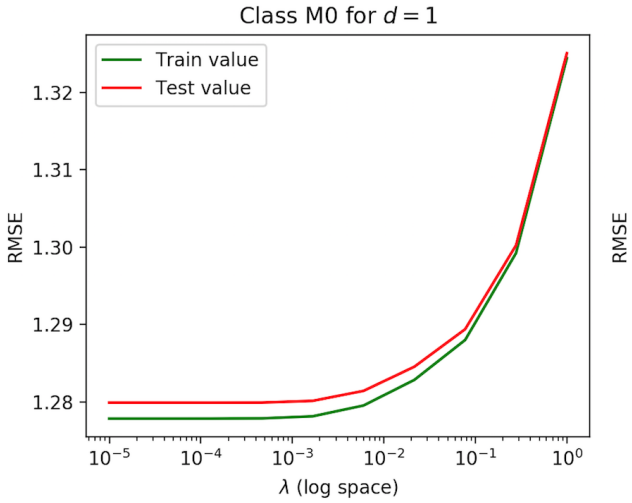

Figure 2. RMSE value for class M0 and $d = 1$

b. *Higher degrees*

For $d = 3$ and class M0, we can see that a certain λ reduces the RMSE value for the validation data, which is for $\lambda = 5.10^{-3}$. So with this information, for every class we can determine an optimal value for λ, which helps reduce the overfitting factor (the red line tends to be closer to the green line, with a smaller error).

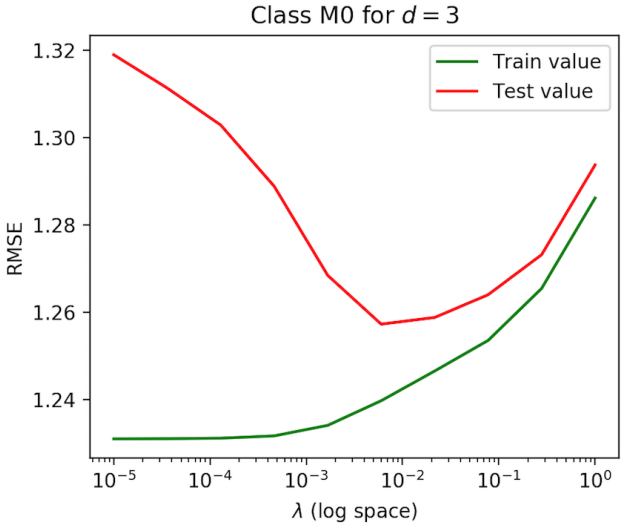

Figure 3. RMSE value for class M0 and $d = 3$

c. *Final result*

Overall, with our final model, we were able to achieve around a score of around 78.8% (this score corresponds to the amount of good guesses on the final test set, which the model could not train on).