Icon

Description automatically generatedIcon

Description automatically generated

BARANZINI Guillaume 22-992-549

**Machine Learning:**

**Assignment 1**

Evaluator: Pr. Alippi Cesare

Year 2022 - 2023

## Introduction

In this report, you will find the answers to the various tasks on Python and questions asked in the first assignment in Machine learning. The objective of this assignment is to review all the concepts that we saw during the first part of our semester.

Task 1

This first task it’s asked using the family of models f (x, theta) = theta\_0 + theta\_1 \* x\_1 + theta\_2 \* x\_2 + theta\_3 \* sin(x\_2) + theta\_4 \* x\_1 \* x\_2 to find the best estimator’s theta that fit the data.

After the computation of the Python code, we find this result:

**Optimal parameters:** [ 1.24205736 -0.04503563 -0.56462773 0.47663575 0.04029328]

**f(x, θ):** 1.242057355919393 + -0.04503563207911075 \*x1 + -0.5646277318069006 \*x2 + 0.4766357540857314 \*sin(x2) + 0.04029327643848045 \*x1\*x2

**1D Train performance:**   0.7103436190376121

**Test MSE**: 0.7296013997475126

The modus operandi is the following:

We started the process by splitting our data into 2 parts the test one and the training one using the function “train\_test\_split”. I decided to use 90% of the data for the training and only 10% for the test from a database of 2000 (knowing that x is bi-dimensional) data because it permits the model to learn more and as consequence be more accurate regarding the relationship between the data.

(+ random\_state parameter is set to 2, which means that the same random seed will be used each time the code is run, ensuring that the split is consistent.)

Then we decided to define our function as a linear function and his parameters x and theta, to apply the regression well we prepared our X (dim = ) matrix which is a compact form of different vectors. This matrix is composed of a vector of ones (for the training data is dim = ) and 4 other vectors: x\_1 = the first raw of our data, x\_2 = the second raw of or data x\_3 = sin(x\_2) and x\_4 = x\_1\*x\_2 with the same dimension than the ones vector.

After the preparation of our data, we pass at the calculation part, we solve our function in a linear way (ax = b) where in this case the “a” is our vector of estimators. So, we solve to find our vector of thetas. Considering that X is the matrix that we compose before and Y is a vector of dim ( that represent our y\_train.

Now that we find our estimators, we can evaluate the performance of our model first on our training data and then on our test data using the mean square error:

where n is the number of samples in the dataset, yi is the actual value of the ith sample, and ŷi is the predicted value of the ith sample.

Thus, in general, the lower the MSE the better our model perform with our result we can say that our model is better on the test data which is a good point it means that the model is generalizing well and is not overfitting to the training data.

Task 2

In this second task, we will use the same function and derive our estimators from it but using a model which is more appropriate for a non-linear function. The model that we decided to use is gradient descent.

Here are the results obtained for theta and the MSE of the train and test data:

**Optimal theta:** [ 1.2420569 -0.04503555 -0.56462758 0.47663592 0.04029325]

**Mean Squared Error:** 0.71034362

**Mean Squared Error (test):** 0.72960141

First of all, before the definition of the function, we don’t forget that the data that we use are split into 90% used to train our model and 10% to apply the test data. Then we decided to define our function as a non-linear function we keep the same structure of X composed of our various vectors and 1 vector of ones.

Here to optimize our estimators we use gradient descent will minimizing the loss function by taking into account X the input data, the theta and Y. In a few words, the gradient decent will minimise the loss function thank the information that will update the parameters until the loss function is minimized. So, at each iteration, the gradient will take the steeper way to reach the minimum updating of the parameters.

In our code we initialized our vector theta randomly taking a length of 5 the same as our matrix X it will change until the loss function will be minimized. Then we will optimize the hyperparameter to obtain the more minimized loss function here we input:

eps = 0.0003

eps0 = 0.01

steps = 250000

Eps are the step size of the gradient descent it will determine how quickly we will converge to the optimal.

Eps0 gives momentum to our step size it will more rapidly converge to the optimal.

Steps are the iteration of the gradient descent will do before stopping.

Then we will create a vector to save the history of the theta from the iteration that will proceed with the calculation of the gradient descent without forgetting a vector that will also store the value of the loss function computed during these iterations. Moreover, we imply a mask to avoid the intercept (theta0) update too quickly to optimize the process indirectly we give less weight to the intercept using the mask.

Finally, after all the iterations our estimators are printed, and we can compute the different MSE for the training and the test one.

Task 3

In Task 3, we have to create a model that has a better performance than 0.022 as MSE. Thus, we decided to first create our model by defining the architecture of our neuron network (don’t forget to install the library keras). Thus, we decided to create 4 hidden layers, the first one is the input layer composed of 16 neurones and has an input dimension equal to 2 because our data are bi-dimensional.

Then we have 2 other layers composed of 16 neurones also that compute calculation and their outputs give the model information about the relation of the data. The last layer is the output layer composed of only one neuron that will give us the target data. In what it concerns the activation functions we have the ReLu (rectified linear unit) in our first 3 hidden layers which introduces the non-linearity to the neuron network that permits the computation of a more complex data relationship. Moreover, for our output layer we desired some continuous value this is why we use a linear activation function.

After that, we need to train our model so that we will save the result into the variable “history”, we use an epochs of 126 which represent the number of iterations that our model trains all the training data set. So, if the epochs are too low it could lead to an underfitting the model is not enough trained on the other hand, if the epochs are too high it can lead to an overfitting. Then we have the batch size where the batchs are small subsets of the dataset used to train the neurons. (validation\_split = % of training data used for validation, we prefer a low validation split to permit at the model better training.)

Finally, we evaluate the model calculating the MSE on the test data which is equal on our cell to 0.0167.

In what it concerns the use of the baseline model I use the script given and I add the path of my model « './model\_task\_3.pickle' » and add the input which are the data given.

The result of the Mean squared error that we have equal to 0.01448861059050486 which is better.

## Theoretical Questions

Q1. Training versus Validation

Q1.1 What is the whole figure about?

The image represents the early stopping which allows us to improve the model performance avoiding the overfitting of the training data. In other words, it’s a graph of the approximation performance so the error against the complexity of our model where we observed the test error expectation in red (average), the observed validation error in reality and the observed training error in reality. (Gradient base architecture)

Knowing that the model is trained with the training data while the early stopping uses the validation data to evaluate the performance of the model. Thus, the early stopping permits us to avoid overfitting the training data which will prevent a decrease in performance on the validation data.

Q1.2 Explain the behaviours of the curves in each of the three highlighted sections in the figure, namely (a), (b), and (c).

Q1.3 Is there any evidence of high approximation risk? Why? If yes, in which of the below subfigures?

In the image a) we can identify an underfitting is a situation where our model is not enough complex to consider all the values of the data. It means that there are few estimators in our model, and it implies a poor performance that can occur more error.

In the image b) we can identify the optimal model for the validation set and the test set in this situation our model has the optimal complexity to capture all the values from the data.

In the image c) we can identify overfitting is a situation where our model is too complex meaning that we use too many resources to capture the value of the data. In this model, there are too many parameters that fit the training data and so will not be able to perform on the test and validation data. This is why we can see the training error decreasing on the graph.

Images a) and c) are both subjects at high approximation risk.

Q1.4 Do you think that increasing the model complexity can bring the training error to zero? And the structural risk?

Increasing the complexity of our model will lead that the model fitting the noise from the training data. Knowing that the structural risk is the ability of our model to perform well with unseen data the risk structural will be high if our model is too complex because he will not generalize them.

Q1.5 If the X axis represented the training iterations instead, would you think that the training procedure that generated the figure used early stopping? Explain why. (NB: ignore the subfigures and the dashed vertical lines)

If the X axis represented the training iterations instead of the model complexity, the training procedure would generate the figure because we can see the up and down on the different line that starts and stop the process to adapt the model.

Q2. Linear Regression

Comment and compare how the (a.) training error, (b.) test error and (c.) coefficients would change in the following cases: Q2.1 x3 = x1 + 0.2 · x2

Considering that y=g(x) + eta where g(.) is unknow and eta follow a normal distribution with input x which is a bi-dimensional vector x = [x\_1,x\_2]. Suppose having n training samples and a linear model f(x,theta) = theta\_0 + theta\_1\*x\_1 + theta\_2\*x\_2.

Now we had another regressor x\_3 and our model is f(x, theta) = theta\_0 + theta\_1 \* x\_1 + theta\_2 \* x\_2 + theta\_3 \* x\_3).

So, if we consider that **x\_3 = x\_1 + 0.2\*x\_2** our model will still be linear and will look like at f(x, theta) = theta\_0 + theta\_1 \* x\_1 + theta\_2 \* x\_2 + theta\_3 \*( x\_1 + 0.2\*x\_2). In what it concerns the training error, and the test error will normally decrease as we capture more information between the input and the target by adding a new regressor.

Nevertheless, if the regressor that we add is irrelevant could have an inverse effect in other words our model could be less performant due to an important complexity (overfitting). Moreover, the addition will affect the coefficient theta\_1 theta\_2 due to the fact that theta\_3 has a relation with x\_1 and x\_2.

Q2.2 x3 = x1 · x2 · x

Then if we consider that **x\_3 = x\_1\*x\_2^2**our linear model becomes a non-linear model. Regarding the training and test errors our model will capture more information, it could decrease the errors depending on if the data are relevant. Moreover, the coefficients will change to optimize the model considering the new coefficients.

Q2.3 x3 is a random variable independent from y.

Finally, if **x\_3 is a random variable independent from y** it means that x\_3 is not correlated to the target variable so will not influence it. In other words, the performance won’t change it could have a little change in the coefficients, but the training and test errors would be quite similar.

Q2.3 How would your answers change if you were using Lasso Regression?

However, if we use the Lasso regression our coefficients the less important having less impact on our target are going to be shrinking to 0 which will reduce the complexity of our model. The Lasso regression applies a penalty to the loss functions to avoid overfitting.

Q2.4 Explain the motivation behind Ridge and Lasso regression and their principal differences.

The lasso regression and the rigide regression are two techniques of regularization that have as objective to avoid the overfitting of our model by avoiding that some estimators get to much importance. They both add a penalty on the mean square error which is normally equal to the loss function. However, the lasso regression adds an absolute value of the parameters while the rigide regression add the square of the parameters as penalty to the loss function. Moreover, they depend on the hyperparameter lambda is a penalty hyperparameter weighting the two contributions (accuracy vs. parameter shrinking). Thus, a small lambda gives more value to accuracy; a large lambda privileges a small number of parameters in the model.

Q3. Classification

Q3.1 Your boss asked you to solve the problem using a perceptron, and now he’s upset because you are getting poor results. How would you justify the poor performance of your perceptron classifier to your boss?

Based on the graph we have the reason that can explain the poor performance of our perceptron is that it’s impossible to perfectly separate our class points with a linear line. So, our perceptron is not able to correctly classify our classes.

Q3.2 Would you expect better luck with a neural network with the activation function h(x) = −x ∗ e ( − 2) for the hidden units?

Theoretically, the neural network will be more adapt to the non-linear classes because it has more hidden layers with non-linear activation functions, allowing them to learn more complex and non-linear decision boundaries. However, here the given activation function is a linear function we will be in the same case that in the perceptron.

Q3.3 What are the main differences and similarities between the perceptron and the logistic regression neuron?

Linear classifiers derived as extensions of the regression methods neither provide a bounded output nor a probabilistic interpretation of it. Logistic regression aims at training the network parameters so that the sigmoidal output is supported by a probabilistic framework. While the perceptron works well on the linear separable classes has an architecture of a single neuron with a Heaviside activation function. After a random assignment for weights (small values) it trains weights that are updated by iteratively presenting instances of the training set. Moreover, the algorithm converges to the optimal parameter configuration if the problem is linearly separable.