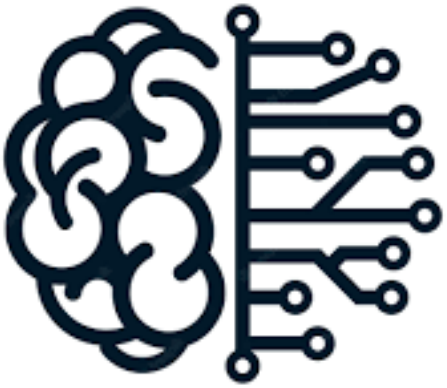
BARANZINI Guillaume 22-992-549

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 theoretical 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.

Be aware if you didn't install these different packages numpy, pandas, sklearn, pickle, io, requests, keras and joblib it could have an error in jupiterlab.

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): 1.24205735591937 + -0.04503563207910319 *x1 + -0.5646277318068914 *x2 + 0.4766357540857329 *sin(x2) + 0.040293276438477466 *x1*x2
Train performance: 0.7103436190376121
Test MSE: 0.7296013997475128

The modus operandi is the following:

First, we load the different libraries that we will need in this cell. Then, we load the data from an URL given using the library request we create x and y that will store the data from the URL. x is bi-dimensional (input) while y is one dimension vector (output).

We started the process by splitting our data into 2 parts the test one and the training one using the function "train_test_split" from the package Scikit-learn's. 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. Actually, it permits the model to learn more and consequently 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.)

After that, we decided to prepare the input features for the test and training set, to apply the regression well. First, we prepared our matrix X (dim = 1800*5) which is a compact form of different vectors. This matrix is composed of a vector of ones 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. We do the same work for the matrix X_test(dim = 200*5) using the x_test.

Then, the LinearRegression class allow us to consider all the features to solve the regression problem. As we can see in the code, we use our matrix X which is composed of our input variables, and we also use a vector of the target values. Using the fit model of our class we can find the estimators that minimize the mean squared error between the relationship of the inputs and the outputs by training them. Moreover, we decide to set the parameter of the fit intercept to "False" which allows us a better understanding of the influence of each feature on the target value.

Now we will create a vector theta_hat (dim = 5*1) to contain the coefficients of our model the ones that minimize the MSE.

After having determined the estimators and printed them we compute the prediction for the training and the test set with the function model.predict from the library sklearn.metrics using our trained

model. It will allow us to determine the mean squared error of each set by using the function mean_squared_error from the package Sklearn.

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 train data. Nevertheless, the difference between the test set and the training set is not huge it suggests that the model performs well and there is no overfitting or underfitting.

Finally, we save our training model into a pickle file with the name “linear_regression. pickle” that can be loaded and evaluated into cell 2.

Task 2

In this second task, we will use the family model 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:

Train performance: 0.0302
Test MSE: 0.0327

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 3 hidden layers, the first one is the input layer composed of 16 neurones and has an input dimension equal to 5 because our matrix.

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 100 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 equal to 8 where the batchs are small subsets of the dataset used to train the neurons. (validation_split = percentage of training data used for validation, we prefer a low validation split to permit at the model better training.)

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 train data. Nevertheless, the difference between the test set and the training set is not huge it suggests that the model performs well and there is no overfitting or underfitting.

Finally, we save our training model into a pickle file with the name “nonlinear_model.pickle” that can be loaded and evaluated into cell 2.

In what it concerns the comparison with the 2 models we can say that the non-linear model performs significantly better than the linear model if we compare the 2 means squared errors.

Moreover, we can explain this by the fact that the family model gets a nonlinear relationship between the features and the target values with the sinus. As the neuronal network is a non-linear model it will capture better the data relationship.

Task 3

In the task 3 our job is to find a model that perform better than the assistants one we need a model that have a MSE < 0.022. So, we decided to use the Random Forest model to train our model.

The random forest model is composed of a multiple of decision trees that will be trained by various of subsets from the training data that are randomly selected for each of them (bootstrapping). To find the final prediction for a regression we take the average of all the prediction calculated by the trees.

For the code we first the libraries that we will use in this task that are RadomForestRegressor and mean_squared_error from the packages Sklearn. Then we impose the different hyperparameters of our random forest model:

The number of estimators equal to 100 which is the number of decision trees in our random forest model and the max depth equal to 10 which controls the max depth of each tree it represents the maximum of splits between each part of the tree and limit the complexity of our model. This hyperparameters are used to optimize our model.

Then we compute the prediction for the training and the test set with the function model.predict from the library sklearn.metrics using our trained model. It will allow us to determine the mean squared error of each set by using the function mean_squared_error from the package Sklearn.

Finally, we save our training model into a pickle file with the name “model_task_3.pickle” that can be loaded and evaluated into cell 2.

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. Nevertheless, I add flatten to the predict output because is a method to convert our input in a 1D array.

The result of the Mean squared error that we have is equal to 0. 0.008566 which is better.

*(In addition, if you want to evaluate the two other models, we create a matrix X_data (dim 2000*5) to match the output y_pred (dim 2000*1). There are also the two paths scripted you just have to cancel the # to evaluate them.)*

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.