MACHINE LEARNING: Assignment 1

Davide Ieraci

N. MATRICOLA: 22992283 EMAIL davide.ieraci@usi.ch

TASK 1:

To solve the regression task with 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, I started splitting the data into training and test set for x and y with the function ' train_test_split ' from sklearn.model_selection, and in particular I used the 80% of the data for the training and 20% for the test. Once I did it, I defined the array of x that contains x_1 +x_2 + sin(x_2)+ x_1*x_2and I called it ' x_train_transformed ' , and then I used it with the ' y_train ' to train the linear regression model using ' model.fit() '. I also created ' x_test_transformed', so I made the prediction using ' model.predict() '. Given the prediction I computed the MSE for both training and test set with ' mean_squared_error(y_train, train_predictions) ' and the same for the test. I obtained the following results:

Training MSE: 0.7231636462303825
Testing MSE: 0.670288340179917
Model formula: f(x) = 1.31 + -0.05*x1 + -0.58*x2 + 0.47*sin(x2) + 0.04*x1*x2.

TASK 2:
For the task 2 I decided to build a neural network model to solve the nonlinear problem after I tried differe nts models such as gradient descent because it gives me the smallest mean squared error. To build it I start ed splitting data into training and test set with 80% and 20% of the total respectively. Then I defined ' x_com pact_train ' and ' x_compact_test ' that contain the family of models, and then I built I created the neural ne twork using the imput layer with 20 neurons and 5 input dimension, with ' relu ' activation function that we have seen in class, one hidden layer with 10 neurons and the output layer with a single neuron. I reduced th e number of neurons layer by layer to reduce the complexity of the model avoiding overfitting. After that I c ompiled the model with the loss equal to the mean squared error and for the optimizer I choose the stochas tic gradient descent that is an iterative optimization algorithm that updates the parameters of the neural ne twork in the direction of the negative gradient descent of the loss function. Then I fitted the model with 100 epochs which is the single procedure over the data and batch size equal to 32 which means that the training set is divided into 32 batches and the gradient is computed based on this and then they are averaged to get the estimated one. Finally, I made the predictions and I plotted the mean squared error for the training and the test set. I found the following results:

Mean Squared Error on training set: 0.10
Mean Squared Error on test set: 0.12.
Compared to the model of the task one this is statistically better, having a mean squared error that is more or less 7 times less and it also has a difference that is lower between training and test set.

TASK 3
For the task 3 I decided to build a neural network model because with multiple layers allows me to reduce t he mean squared error as much as possible until I obtain the results lower than 0.022 that is required. I star ted splitting data with 70% for training and 30% for test, and then I defined the neural network with one inp ut layer with two dimensions and ' relu ' activation function and 64 neurons, the hidden layer with 32 neuro ns and ' relu ' activation function and in the end the output layer with one neuron, the choice of the numbe r of neurons that are decreasing is because I tried to ' shrink ' the features and to reduce the model avoiding overfitting, as the model in the task 2 . Then I compiled using mean squared error and ' adam ' optimizer, th at uses the mean and the variance of the gradient to adjust the learning rates for each parameter in the neu

ral network, so it converges fastly toward the optimal solution. I fitted data using 300 epochs that is a single procedure over the data and ' batch size ' equal to 32 because searching on internet I found that is like a rul e of thumb to use 32 or 64. In conclusion I made the predictions with ' model.predict() ' for the training and the test set and I printed the mean squared error for both.
The results I obtained are:
Training MSE: 0.014624534459571237
Test MSE: 0.01762934118616797

If I run again the code I obtain different results but are almost always lower than 0.022 so the model is perf orming well for the problem.

QUESTIONS Q1

1)

The figures are about the prediction error. In particular, the y-axes is the performance and the x-axes represent in general the model complexity. The blue line is the observed validation error, the red line is the expected test error and the green one is the observed training error. We move around the model complexity to find the one that gives the optimal model with the lowest prediction error.

2)

In the figure a) we are in the underfitting area, so the complexity is not enough and the model have both training and test error too high. The figure b) is the area in which we find the optimal model for the validation set and the optimal model according to the test set. Here is where we stop if we are using for example ' early stopping ' method. If we increase the number of iterations/model complexity we arrive in the area of the figure c). In this area we have overfitting, which means that the model is too complex and the model fit perfect the data and the training performance is very high but the test performance is poor because there is too much noise on the new data.

3)

The approximation risk depends on the difference between the optimal model that is estimated and the real model the it unknown, and we can control it for example by changing hypothesis, so it is not something that can be seen on the given graphs.

4) The training error can be brought to zero which means that the model fit perfectly the training data, and the structural risk cannot be computed, but if we think about empirical risk it can increase due to overfitting, having poor performances on the new data because the model learned all the noise of the training set.

5) Yes, early stopping can be used to find the optimal complexity of the model, moving among the iterations to find which is the one that optimize the error before having overfitting.

QUESTIONS Q2:

Answer 1: Adding a new feature x_3 = x_1 + 0.2 * x_2 will probably decrease the training error because we are using a more complex model to fit the data but maybe the test error could increase due to overfitting or maybe could decrease, it depends on where we are with the performance-complexity chart described before. The coefficient of x_3 in this case can be positive as it brings more information and it is a sum of x_1 and x_2.

Answer 2: For the training and test error I can say the same thing of the answer 1. It probably decrease the training error but for the test error it can have a positive or negative effect. The coefficient x_3 may be positive or negative it depends on the relationship with y.

Answer 3: if x_3 is a random variable independent from y basically it gives no information for the model, so if the model does not change significantly, also the training and test error will be the same and the value of x_3 will be equal or close to zero.

Answer 4: Lasso regression aims to reduce the model by pushing some parameters to zero. In this case with Lasso regression the parameters can go to zero due to the penalty parameters .In particular, in the case x_3 is a random variable independent from y.

Answer 5: the motivation under these two type of regression is to intervene on the learning procedure to put the parameter to zero in order to shrink the model, for example to avoid overfitting. Ridge regression use a shrinking penalty for the loss function and it penalize the square of the parameter, while the Lasso regression penalize the parameter adding a shrinking penalty. The higher the penalty parameter, the lower parameters the model will have.

QUESTIONS Q3

Answer 1: the problem of the perceptron is that is able to solve only linearly separable problem, in the sense that we can draw a hyperplane that separate the classes. In this case we are not able to draw a line that split in a right way the classes.

Answer 2: The activation function is not linear and the neural networks can gives us better results but in order to have an activation function that assign a confidence level to my classes so I need the logistic regression as activation function. This activation function does not produce an output between 0 and 1.

Answer 3: Both perceptron and logistic regression neuron takes a set of inputs and produce a binary output for classification. The perceptron uses a Heaviside activation function which gives 0 or 1 while the logistic regression neuron use a sigmoid function, that allows to have an output in terms of probability between 0 and 1. The perceptron learning procedure is not coming from minimization of error, it updates the weights looking at the error between the real value and the classification value provided by the neuron to converge to the optimal hyperplane, instead for the and logistic regression neuron we want to maximize the cross entropy function or minimize the opposite because we want to have a probabilistic interpretation of the output.