Machine Learning

Università della Svizzera italiana

Assignment 1

Hatas Tomas

May 3, 2022

1 REGRESSION PROBLEM

SPLIT DATA

At the beginning of the analysis was necessary to split the data. Therefore, I created the new split_data.py file which was responsible for splitting the data into training and test set. The split was done by provided train_test_split() function. Consequently, 4 new numpy files were created in the src folder, namely test_set_x.npy, test_set_y.npy, training_set_x.npy and training_set_y.npy.

1. 1 Task 1 - LINEAR REGRESSION

From the beginning I was using the training set. From the training set I extracted x1 vector and x2 vector. I needed these 2 vectors in order to create another 2 vectors representing cos(x2) and $x1^2$. It was just needed these created vectors to reshape for next step. After reshaping these 2 additional vectors, I packed all these vectors into 4-D matrix thanks to the numpy hstack() function. The preprocessing stage was done.

Now, I was ready to process the Linear Regression itself. First of all, I needed to create an instance of class LinearRegression(). This class has needed functions for solving this linear problem. I was ready to use the fit() function to get optimal needed thetas.

As the provided formula had one theta parameter without any variable, I had 2 options how to figure out this theta parameter, either to add additional column of 1 to the left of numpy hstack() or to use intercept_ and coef_ in order to avoid creating this additional column of

1. I chose the second option. The first theta parameter without any variable is represented by the intercept_. Other theta parameters with different formation of variables are represented by coef_. The result is:

$$f(x, \text{theta}) = 1.764 + (-0.164) * x1 + (-0.652) * x2 + (-0.019) * \cos(x2) + (0.042) * x1^2$$

After fitting my linear regression model I saved it as linear_regression.pickle in deliverable folder.

Everything was ready to predict the linear function. This prediction is done by using provided predict() function. Now, the test set comes into play. I needed to do the same preprocessing procedure with the **test set**. After packing all the vectors into 4-D matrix thanks to the numpy hstack() function, I was able to obtain the y predict function by using predict() function where the input parameter was the 4-D matrix from test set.

In the figure 1: linear model we can see the visualization of this regression model after applying the linear model with the above-mentioned formula. I used elevation 5. in the view_init() function.

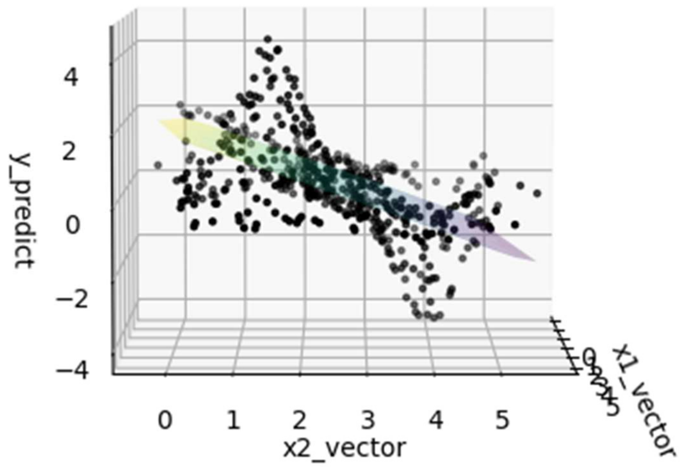

Figure 1: Linear model

Now, I was able to evaluate the test performance of my linear regression model using the mean squared error (MSE in text) as performance measure. To do so, I used the provided mean_squared_error() function from sklearn.metrics. This function takes 2 parameters. The first parameter is the test set y and the second parameter is the y predict function. This procedure takes the difference between y predict function and the provided test data and sum these differences. The lower mean squared error, the more accurate my linear regression model. The MSE of my linear regression model is 1.1810, which is not horrible.

1. 2 Task 2 – NON-LINEAR MODEL FOR REGRESSION PROBLEM

As I use the Visual Studio Code to solve this assignment, I needed to import all the necessary libraries for neural networks. While installing the Keras libraries into my virtual environment, I got an error saying that I do not have sufficient graphic driver. The recommendation was to install a certain driver from NVIDIA, but I only have iRIS xe graphics. Therefore, I tried to do it in the colab Google. But it would not be easy to compare my linear regression model which was created in my Visual Studio Code and the neural network model created in the colab Google. For this reason, I decided to try another non-linear model. I was looking for some non-linear models in Google. I found Random Forest Regressor as reliable option. I was surprised since I only know Random Forest for classification problem. As a matter of fact, I was able to learn something new because I already learned the neural network in the lab.

Random Forest consists of many decision trees. In other words, random forest combines the predictions made by many decision trees. These decision trees use the same data set. The idea behind it is that if we have more predictions, then we can have more accurate single model. In fact, the Random Forest Regressor takes the average of all the predictions from the decision trees and put it as a prediction of the Random Forest. Consequently, the Random Forest Regressor model is more robust and prone less to overfitting than the decision tree. Moreover, it is possible to find out what variables contribute to the predictions more than other variables. However, the last advantage would be beneficial in case of more variables. The disadvantage is that the Random Forest Regressor cannot extrapolate, but neural network neither.

I created an instance from the class RandomForestRegressor(). There are many possible input parameters such as n_estimators representing basically the number of decision trees, max_depth representing the maximum depth of the tree and etc. I left every parameter default except of n_estimators. I put different numbers for n_estimators and I came to the conclusion that 650 would be good number. Even bigger number of decision trees led to higher MSE.

After that I trained my Random Forest Regressor model by using fit() function, where input parameters were training set x and training set y. In this case it was not necessary to change or modify the training set compared to the 1. 1 Task 1. Then, I saved the model as random_forest_regressor.pickle.

Next, I predict the y function by using predict() function, where input parameter was the test set x.

In the figure 2: nonlinear model we can see the visualization of this regression model after applying the Random Forest Regressor. Again, I used elevation 5. in the view_init() function.

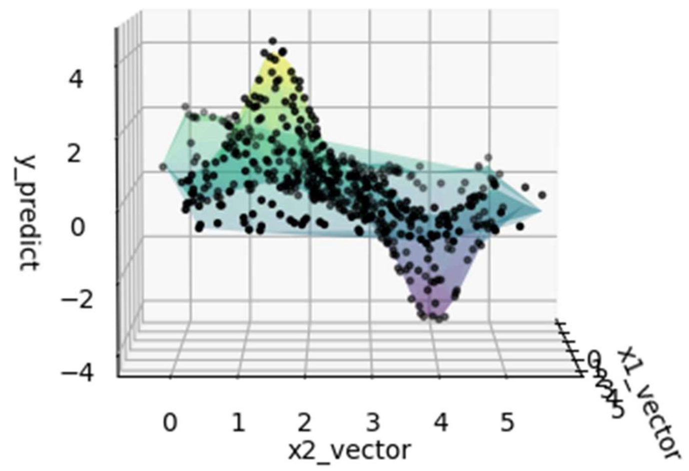

Figure 2: non-linear model – Random Forest Regressor

After predicting the y function, I calculated the MSE by using mean_squared_error(). The input parameters are the test set y and the y predict function. The MSE of my Random Forest Regressor model is 0.0248, which is remarkable.

COMPARISON

In order to compare the linear regression model and random forest regressor I used the paired t-test like in the Lab.

First of all, I used KFold() function which is for regression problem. I put 2 input parameters into this function: n_splits=10 and shuffle=True. It is necessary to shuffle the data set. n_splits=10 represents the fact that the data set will be split into 10 parts. 1 part represents the validation set and 9 parts represent the training set. But the validation set is going to rotate through the complete data set and the training set do it as well. For this reason, I needed the for loop.

Before using the for loop, I needed to use split() function which was assigned to the fold_iterator. Now, I was able to create the for loop.

In each iteration are created new training set x, training set y, validation set x and validation set y. Moreover, the Random Forest Regressor and Linear Regression have to be trained on the training data by using fit() function. In order to measure accuracy, which will be needed for the paired two sample t-test, I used score() function on the validation set which gave me the accuracy metric. In each iteration the accuracy for Linear Regression and the Random Forest Regressor was appended to the appropriate list.

Once the code gets out of the for loop, the paired two sample test is calculated. I used the ttest_rel() function, where the input parameters are the lists of the measured accuracy for Linear Regression and Random Forest Regressor. The size of this list depends on the number of n_splits. As I set n_splits=10, I get 10 values in each list for accuracy. The ttest_rel gave me t and p value.

If I run my compare.py file multiple times, I always get different t values because of splitting in each iteration in the for loop. Nevertheless, t value is always outside of the 95 % confidence interval (-1.96, 1.96). This means, that I can reject the null hypothesis. In fact, I cannot say which model is statistically better. If I can not say which model is statistically better, then I select model with smaller variance. Variance is basically same like MSE.

In conclusion, my Random Forest Regressor model is better than my Linear Regression model on the data set. Nevertheless, the Random Forest Regressor may deliver higher MSE on the unseen data as prone more to overfitting. But my guess is that the MSE from Random Forest Regressor should not be higher than Linear Regression model in terms of the unseen data.

1. 3 Task 3 (Bonus)

Your baseline model achieves a MSE of 0.0197 on the test data but on the entire data set your model achieve 0.01503 which is just slightly better than on the test set. My Random Forest Regressor has higher MSE on the test data which is 0.0248 than yours. However, if I test my model on the entire data set, I got only 0.0086 which is much better than in your case.

Therefore, my Random Forest Regressor will achieve lower MSE on unseen data than your baseline model.

2 QUESTIONS

Q1. Training versus Validation

1. We have a data set. This data set is split into 3 parts: training set, validation set and test set. The way how to split the data depends on the number of samples in the set. Let' s assume that the number of samples is 1M, which leads to training set 99 %, validation set 0.5 % and test set 0.5 %. Following bullet points represent the sections which should be described.
   a) Model complexity is very low, which leads to high MSE. As the model becomes more complex, the MSE decreases. We train a model with training data. The training error is calculated based on the training data. But if we take this model and calculate the MSE with test or validation set, the MSE will be higher because of different data.
   Let's assume, I draw a line through the sample set, I will get many noises because the function does not fit well to the data set. Due to low complexity, the model does not explain the regression problem well as we talk about underfitting.

   b) Model complexity is higher than in the figure (a). As the model becomes more complex, the MSE decreases to the certain point. This certain point represents the lowest point for test error and for validation error. The dashed vertical blue line represents this lowest validation error. Based on the lowest validation error we decide to select the model. But the dashed vertical red line represents the lowest test error which serves as performance measure of the model. Therefore, the optimal model is the one on the dashed vertical red line.

   c) Model complexity is higher than in the previous figures. This leads to the fact that now we can see a big difference between training error and test/validation error. In terms of training error, as we have higher model complexity, this model goes through more and more samples in the training set which leads to lower and lower training error. The reason is that model goes through huge number of samples in the training set which leads to overfitting. As a result, if we calculate MSE on the validation/test error with the same model, we obtain higher and higher MSE. The reason is that model was too perfect for the training data, but for validation/test data is useless. In fact, the model depends too much/strictly on training data. Consequently, if the model gets the different data set, the model cannot handle it well.

   For the above-mentioned reasons, we must find an optimal model by gradient descent.

2. Approximation and estimation risk – we have unknown g(x) function and we try to find optimal function which is close to the unknown g(x).
    a. Approximation risk is a distance between error (function loss) from g(x) and error from the optimal function f(theta0, x). If this distance is small, we have low risk. On the other hand, if this distance is big, we talk about high approximation risk. Approximation risk can be reduced by more appropriate family model. In other words, more complex model reduces the approximation risk. The goal is to have low approximation risk.
    The figure does not provide us the unknown g(x) function. It means that the figure gives just some useful information to reduce the approximation risk because with more model complexity, we achieve lower error (approximation risk) to the certain point. This certain point (vertical dashed red line) represents the optimal function with lowest error. In other words, the figure provides us the error from the optimal function but not the unknown g(x) function.

    b. Estimation risk is related to the learning procedure so that the estimation function is close to optimal function. In other words, the estimation risk is a difference between error from the estimation function and error from the optimal function. The goal is to find out the family of model which is close to the unknown g(x) function.
    The figure gives useful information to reduce the estimation risk because the vertical dashed red line represents the optimal function. The optimal function gives us the lowest error.

3. No, increasing model complexity would not be able to bring the structural risk to zero. Increasing model complexity would lead to overfitting. Overfitting means that the model is too strong/to perfect for the training set, but poor for unseen data. If data was not affected by noise, we would still not be able to bring the structural risk to zero due to overfitting.

4. Yes, I am convinced that the training procedure used an early stopping. Early stopping reduces overfitting and is applied in the validation set. It is necessary to define the stopping criterion which is usually minimum validation loss. It means that if the training procedure (calculation validation loss) does not change over some iterations, we can be safe and stop here and select it as lowest loss of the optimal model according to the validation set.

Q2. Linear Regression

1. After adding x_3 (x_3 = x_1 + 3.0 * x_2), we have more complex model. More complex would lead to lower training error and test error as well. We can not claim that this model is too complex which would lead to difference between training error (still decreasing) and test error (increasing after optimal model). Coefficients for thetas would change because of additional regressor x_3.

2. After adding x_3 regressor (x_3 = x_1 * x_2 * x_2), we have slightly more complex model than x_3 = x_1 + 3 * x_2 because of power and multiplication between regressor x_1 and x_2. This should affect the training and test error so that the MSE is a bit lower than in the case above. Again, coefficients for thetas would change because of additional regressor x_3.

3. Lasso means Least Absolute Shrinkage and Selection Operator. Lasso Regression is mostly used when we talk about overfitting. Overfitting means to have too perfect/complex model. The Lasso penalize the coefficients by reducing coefficients towards to zero so that we can still use the complex model and avoid overfitting. Lasso can even reduce the coefficients exactly to zero if the coefficient does not contribute to the explanation of regression problem at all. More precise explanation is below in the point 4.
   Looking at the linear family model, which is not complex at all, we can not talk about overfitting. Therefore, I would not use the Lasso Regression. If anyway the Lasso Regression is used, then the value of theta_3 would change because less contributive coefficients would be set exactly to 0. After running the Lasso Regression in VS code, the theta_1 and theta_2 was set exactly to 0 and the theta_3 had different value than in case of Linear Regression. In both Lasso Regression cases the MSE was higher than in case of Linear Regression.

4. There is a common problem called overfitting. This means that the model is too perfect/too complex for the training data but performs very poor on the test data. It means that the model uses more parameters/coefficients than is necessary/optimal. This is a violation of Occam's razor.
   For this reason, the Ridge and Lasso Regression regularize these coefficients. In fact, they penalize the overfitted linear regression by reducing the coefficients towards zero. As a result, it allows to use complex model and still avoid overfitting.
   a. In terms of Ridge regression, the penalty function consists of lambda and the sum of the squared coefficients. The Ridge regression can only shrink the coefficients close to zero. As a matter of fact, the coefficients cannot be set exactly to zero.
   b. On the other hand, the penalty of the Lasso regression consists of lambda and the absolute value of the coefficients. The advantage over Ridge is the fact that less contributive variables can be set exactly to zero. Consequently, the model has a reduced set of regressor due to zero coefficients.

Q3. Non-Linear Regression

1. No. The model of the family: f(x, theta) = theta_0 + theta_1 * x_1 + theta_2 * x_2 is linear model. However, we can see non-linear problem in the picture because the y prediction would not be straight line if our goal is to minimalize the MSE. In other words, to better reflect the regression problem, the y prediction will not be a straight line.

2. Yes. Any family nonlinear model should improve the result. A feed forward neural network with the mentioned activation function would definitely improve the result.

3. Yes. It would be possible to transform the data set into higher dimension. Then it may be possible to draw a straight line in higher dimension to achieve a good performance. Anyway, if we transform the data set back to original domain, the straight line would not be any more straight but curve line.

4. First of all, it is necessary to define what is a hidden layer. Hidden layer receives input from the input vector/layer or previous hidden layer. Consequently, the hidden layer provides output to next hidden layer or output layer.
   The activation function in the hidden layer controls how well the network model learns the input vector/layer or previous hidden layer. Choosing different activation functions will lead to different results/output. In fact, the word “activation” in the activation function already says a lot. The activation function makes a decision whether the previous neuron’s input to the hidden layer with this activation function is important to the network or not.
   Nowadays, the popular activation functions in hidden layers are Rectified Linear Activation (ReLU), Sigmoid function and Tanh function. However, the preferred activation function in hidden layers prevails ReLU.