Machine Learning
Università della Svizzera italiana

Assignment 1

Morales Mariciano Jeferson

May 3, 2024

!! !! Note that this file is just meant as a template for the report, in which we reported *part of* the assignment text for convenience. You must always refer to the text in the README.md file as the assignment requirements !! !!.

Tasks

This section should contain a detailed description of how you solved the assignment, including all required statistical analyses of the models' performance and a comparison between the linear regression and the model of your choice. Limit the assignment to 8-10 pages and do not include any code in the report.

Task 1

Use the family of models $f(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot \cos(x_1) + \theta_4 \cdot x_2 \cdot x_2 + \theta_5 \cdot \tanh(x_1)$ to fit the data.

a. Write in the report the formula of the model substituting parameters $\theta_0, \ldots, \theta_5$ with the estimates you've found:

   $$f(\mathbf{x}, \boldsymbol{\theta}) = 0 + 5.019 \cdot x_1 - 4.000 \cdot x_2 + 6.983 \cdot \cos(x_1) + 1.997 \cdot x_2 \cdot x_2 - 0.088 \cdot \tanh(x_1)$$

   The coefficient parameters $\hat{\theta}$ are shown until the 3rd decimal place.

b. Evaluate the test performance of your model using the mean squared error as performance measure. The test performance of the model is **1.4662**. The value is shown until the 4rd decimal place.

c. Implement Lasso Regression, what do you observe? What can you infer about the given family of models? Lasso is generally worses than the simple linear model, although by tweaking the hyperparameter λ it could be reach similar results.

Task 2

Consider any family of non-linear models of your choice to address the above regression problem.

a. Evaluate the test performance of your model using the mean squared error as performance measure (same data as Task 1). The test performance of the model is still **1.4664**. The value is shown until the 4rd decimal place.

b. Compare your model with the linear regression of Task 1. Which one is statistically better? By doing the statistical comparison I get the following values from the confront between the square error of the 2 models. The comparison is in the jupiter notebook.

Task 3 (Bonus)

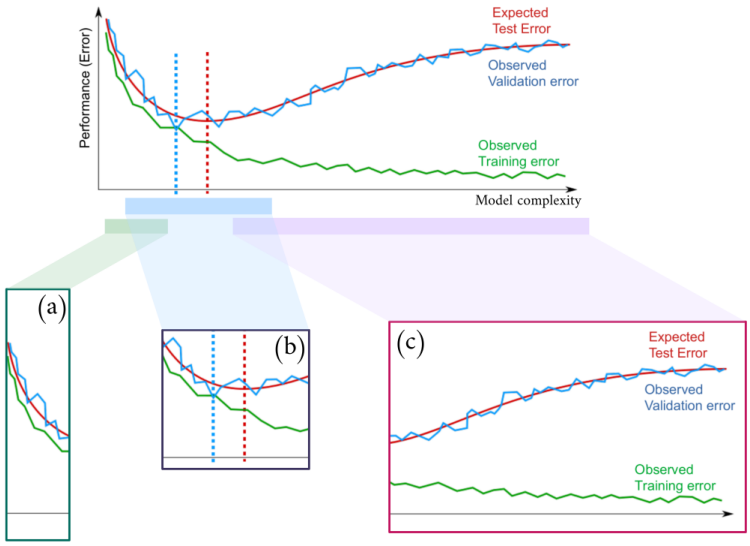
In the **GitHub repository of the course**, you will find a trained Torch learn model that we built using the same dataset you are given (**data_bonus**). This **baseline** model is able to achieve a MSE of **0.013**, when evaluated on the test set. You will get extra points if you provide a model of your choice whose test performance is **better** (i.e., the MSE is lower) than ours. Of course, you must also tell us why your model is performing better.

The model I built is a Feed Forward Neural Network with 2 hidden layers with 30 and 13 neurons each, using the sigmoidal and ReLU activation function. In general, from the Universal Approximation Theorem (1991), FFNNs are able to approximate virtually any function. The algorithm quits as soon as it reaches a better test performance than the model implemented by the course staff.

Questions

Q1. Training versus Validation

Figure 1: Training versus validation exercise image



Q1.1 What is the whole figure about?

A1.1

The Figure 1 shows the bias-variance tradeoff for a model within the context of the regression problem. The bias gives errors due to erroneous assumptions in the learning procedure. In contrast, the variance yields errors due to the sensitivity of the model to the training set fluctuations i.e. noise in the training dataset. The three error lines displayed in the plot correspond to:

- Observed training error, from training set used to fit the model on training data
- Observed validation error, from test set assessing performance meaning how well the model generalizes to unseen data
- Expected test error, from an ideal unbiased dataset assessing performance on unseen data meaning how well the model would ideally generalize to unseen data

Along the x-axis, model complexity is intended as more hidden layers or neurons in the model. In this particular model we can see that the observed training error is decreasing as the model complexity increases, this will lead to overfitting. Along the y-axis, the lower the performance metric, the better the model since it measures the error of the model.

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

A1.2

In Figure 1:

a. window is associated with the start of the training process, when the model is *underfitting* the data, has high bias and low variance. The model is excessively simple to capture the underlying patterns from the dataset, that is evident from the high observed training error, and also the validation error is high as a consequence of the non-generalization of the model with unseen data from the test set. We talk about test set and not validation set because there is no model selection process involved. The model has high bias and low variance at start. As the model complexity increases, the training error decreases and the model starts to learn the patterns in the data. However, the validation error decreases at the beginning but after a certain point starts to increase.

b. window is associated with the optimal bias-variance tradeoff for a model. The ideal optimal model complexity θ^o resulting from the expected test error and the observed training error references the red dotted vertical line, and the optimal model complexity point according to the validation set θ_m resulting from the observed training and validation error references the blue dotted line.

c. window is associated with the erroneous assumptions in the learning procedure, when the model is *overfitting* the data, has low bias and high variance. The model is excessively complex to capture the underlying patterns from the dataset, that is evident from the low observed training error as it started to learn from the fluctuations of the independent input variables i.e. their variance, and also the validation error is high as a consequence of the non-generalization of the model with unseen data from the test set due to the

noise learnt. As the model complexity increases, the training error decreases and the model starts to learn the noise patterns in the data. The more you continue once overfitting started, the worse the performance assessment result will be for the model, as the training error and validation error diverge in performace metrics.

Q1.2.a Can you identify any signs of overfitting or underfitting in the plot? If yes, explain which sections correspond to which concept.

A1.2.a

As already vastly explained in previous answer A1.2, the signs of underfitting are evident in Figure 1 (a.) at the start of the training process.

The signs of overfitting are evident in Figure 1 (c.) after the optimal model complexity point θ^o, when the model starts to learn the noise patterns in the data.

Q1.2.b How can you determine the optimal complexity of the model based on the given plot?

A1.2.b

The optimal complexity of the model can be determined based on Figure 1 window (b.) of given plot: between underfitting and overfitting. Among the two optimal model complexity points θ^o and θ_m delimited by their dotted vertical lines. Where both the ideal expected test error and the observed validation error are minimized.

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

A1.3

The approximation risk is defined as $\overline{V}(\theta^o) - V_I$ where $\overline{V}(\theta^o)$ is the structural risk and V_I is the inherent risk. It depends on how well $f(\hat{\theta}, x) \approx g(x)$. It can be improved by choosing a more appropriate model family $f'(\hat{\theta}, x)$ reducing the risk. Figure 1 shows the relationship between model complexity and error in performance. The evidence of high approximation risk is present in the subfigures (a.) and (c.). They are associated with underfitting and overfitting respectively. High approximation risk for underfitting is due to the model being too simple to capture underlying patterns, while for overfitting is the poor generalization behavior on unseen data. The spot with minimum approximation risk is the optimal model complexity point θ^o where the data generation function $g(x)$ is best approximated.

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

A1.4

Increasing the model complexity could maybe bring the training error close or even to zero. From the analysis of Figure 1 (c.) we can see that achieving 0 training error is not a good goal, as it would lead to overfitting, where the model learns the noise patterns in the data. It is not sufficient to increase model complexity to reduce the structural risk.
The structural risk is defined as a sum of 3 terms:
$\overline{V}(\hat{\theta}) = [\overline{V}(\hat{\theta}) - \overline{V}(\theta^o)] + [\overline{V}(\theta^o) - V_I] + V_I$

The structural risk represents the model performance in the optimal case as it describes the generalization abitility of the model, and shows the discrepancy

through loss function $\mathcal{L}$. Improving the structural risk means bringing each term to 0:

$\overline{V}(\hat{\theta}) - \overline{V}(\theta^o)$ estimation risk can reach 0 if the model family chosen reach the optimal θ^o coefficient parameters.

$\overline{V}(\theta^o) - V_I$ approximation risk can reach 0 if model family $f(\theta, x)$ exactly approximates the data generating function $g(x)$.

V_I inherent risk can reach 0 if no noise is present in the dataset. Since uncertainty is inherently present in the every physicl sensor, data always has it.

In conclusion, increasing the model complexity could potentially bring the training error to 0 but it is not a good goal and the structural risk is not reduced by purely increasing model complexity.

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? Explaib why. (**NB**: ignore the subfigures and the dashed vertical lines)

A1.5

If early stopping was used, then the plot should have ended when it reached the optimal model θ_m according to the validation set in window (b.), before the validation error started to increase. Since it reached way far in the training iterations in window (c.), reaching overfitting of training data and learning noise from it, we can state that the procedure generating Figure 1 did not used early stopping regularization technique.

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 $x_3 = x_1 + 0.2 \cdot x_2$.

A2.1

When adding the regressor to the model family formula

$$\begin{aligned} f(x, \theta) &= \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3 \\ &= \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot (x_1 + 0.2 \cdot x_2) \\ &= \theta_0 + (\theta_1 + \theta_3) \cdot x_1 + (\theta_2 + 0.2 \cdot \theta_3) \cdot x_2 \\ &= \theta_0 + \theta_1' \cdot x_1 + \theta_2' \cdot x_2 \end{aligned}$$

The new regressor addition did not change the model family structure. Th new feature predictor x_3 is a linear combination of x_1, x_2, hence is redundant and increases the multicollinearity in the model. Such property could lead to overfitting and make the model learn noise patterns. In windows:

a. the train error could either remain the same or decrease due to multicollinearity
b. as consequence of multicollinearity, thus overfitting, the test error could increase
c. since θ_3 is linear dependent, it lead to multiple solutions for θ_1, θ_2 and the coefficients could be unstable and not unique, as a result of overfitting

Q2.2 $x_3 = x_1 * *2$ (in Python ** is the "power" operator, so 3 ** 2 = 3 * 3 = 9).

A2.2

When adding the regressor to the model family formula

$$\begin{aligned} f(x,\theta) &= \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3 \\ &= \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_1^2 \\ &= \theta_0 + (\theta_1 + \theta_3 \cdot x_1) \cdot x_1 + \theta_2 \cdot x_2 \\ &= \theta_0 + \theta_3' \cdot x_1 + \theta_2 \cdot x_2 \end{aligned}$$

The new regressor addition added a non-linear combination of existing features. The model complexity is higher and the model family structure has changed. Assuming the relationship between y and predictors is non-linear, the new regressor could capture the non-linear patterns in the data making better predictions. If the assumption does not hold, then Occam's razor principle applies, and the new predictor could decrease overall performance. In windows:

a. the train error could decrease if the data is linear, otherwise it could increase
b. the test error could decrease if the data is linear, otherwise it could increase
c. the coefficients could be more stable and unique as the model is able to capture the non-linear patterns. This depends on regressor relevance, multicollinearity, the relationship linearity among input independent variables x_i and the dependent output variable y.

Q2.3 x_3 is a random variable independent from y.

A2.3

When adding the regressor to the model family formula

$$f(x,\theta) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3$$

Predictors should have a consequences on the dependent output variable y. If that's not the case for x_3, then it does not provide any additional information to the model and it is not useful for the prediction, bringing up noise problems and causing overfitting. Overall, the model family structure change, its complexity is higher, and the new regressor could capture noise fluctuations making worse predictions.

a. initially, the train error could decrease due to multicollinearity and noise overfitting
b. the test could increase as a consequence of overfitting the training dataset and not generalizing to unseen data
c. after regression, the independent random variable x_3 could either be close or set to 0

Q2.4 How would your answers change if you were using Lasso Regression?

A2.4

Q2.1 since the main problem of x_3 is being a linear combination of x_1, x_2 and the multicollinearity it brings could be solved by Lasso regression as the regularization mechanism could lower or even set to 0 the coefficient parameter θ_3 or the couple θ_1, θ_2. So, it depends on the relevance of the predictor x_3 and the couple x_1, x_2 over the dependent output variable y.

Q2.2 as the model goes to a higher complexity order with $x_3 = x_1^2$, the Lasso regression would generally decrease training error and increase test error. Since x_3 is a non-linear combination of x_1, I would speculate that the coefficient parameter θ_3 of x_3 not be set to 0 since x_1 is a relevant predictor. The redundant nature of x_3 could probably lead to overfitting since it amplifies x_1 fluctuation noise in the model learning process.

Q2.3 the coefficient parameter θ_3 of x_3 random variable input would be set to 0 as Lasso regression purpose is to allow irrelevant predictors to be set to 0. The model will then be the same as the original one without x_3. So the expected behavior on (a.), (b.) and (c.) would be the same as in the original model with $\theta_3 = 0$.

Q2.5 Explain the motivation behind Ridge and Lasso regression and their principal differences.

A2.5

Both Ridge and Lasso regression are regularization techniques used to prevent overfitting and improve generalization in linear regression models. They add a penalty term λ to the loss function to prevent the irrelevant coefficients of feature variables to become too dominant by setting them either close or to 0. High values of λ lead parameter shriking i.e. to few relevant parameters in the model. Small values of λ lead to accuracy in parameter values, unaltering the model. Both of them solve/mitigate multicollinearity in indipendent predictor variable inputs. Both of them usually implement preprocessing techniques to normalize the input data and easy the regularization process: *centering inputs* to have mean 0 and *scaling* them to have unit variance. With MSE loss function $\mathcal{L} = V_n(\theta)$ for both regression techniques, ridge regression is defined as:

$$V_{Ridge}(\theta) = V_n(\theta) + \lambda||\theta_i||^2$$

it penalizes the square of parameters and it solve the rank deficiency problem for multicollinearity in independent input variables in matrix, i.e.

$$\exists(X^T X + \lambda I)^{-1} \text{ for } \lambda > 0$$

Although, LASSO regression defined as:

$$V_{LASSO}(\theta) = V_n(\theta) + \lambda \sum_{i=2}^{d} |\theta_i|$$

as the interesting property of setting irrelevant parameters to 0, while ridge cannot set them to 0, but only close. Such property is due to Lasso penalizing the absolute value of parameters where the original becomes non-convex and would require quadratic programming optimization tecniques. LASSO potentially solves multicollinearity and promotes sparse coefficient solutions.

Q3. Logistic Regression

Q3.1 What are the main differences between the logistic-regression and the perceptron?

A3.1

The main differences between logistic regression and perceptron are:

activation function: logistic regression commonly uses a sigmoidal function as its activation function allowing a non-linear decision boundary. Instead, perceptron uses a step function as its activation function which limits the decision boundary to be linear.

learning procedure: logistic regression leverages gradient descent to minimize the loss function $\mathcal{L}$, thus optimizing the coefficient parameter values θ. On the other hand, percetrons use the single layer neural network called delta rule to update the weights θ of the input features.

output probability: logistic regression yields the estimated probability of the dependent output variable y for classification tasks. Instead, the discrete nature of the perceptron output $y \in \{0, 1\}$ is a consequence of the limits in its decision boundary.

Q3.2 Discuss the major limit they share and how neural networks can solve it.

A3.2

The major limit of both logistic regression and perceptron is the inability to capture non-linear relationships. The logic within them is to create decision boundaries with hyperplanes. The XOR problem is a classic example where both logistic regression and perceptron fail. Neural networks can solve this problem by using the hidden layers in the model which are able to capture complex non-linear relationship in data.

Q3.3 What is the role of activation functions in feedforward neural networks.

A3.3

The role of the activation function in FFNN is to determine which neurons should be activated. Allowing subsets of neurons from the hidden layer to be either activated or not based on the activation function, permits the model to learn complex non-linear patterns in the data. E.g. the XOR problem becomes solvable. Continous activation functions e.g. the sigmoidal, are typically usually used in the hidden layers of the FFNN to allow the model to approximate virtually any function, as stated by the Universal Approximation Theorem (1991). The activation function is applied to the output of each neuron in the hidden layer(s) and it transforms the input into a non-linear output.

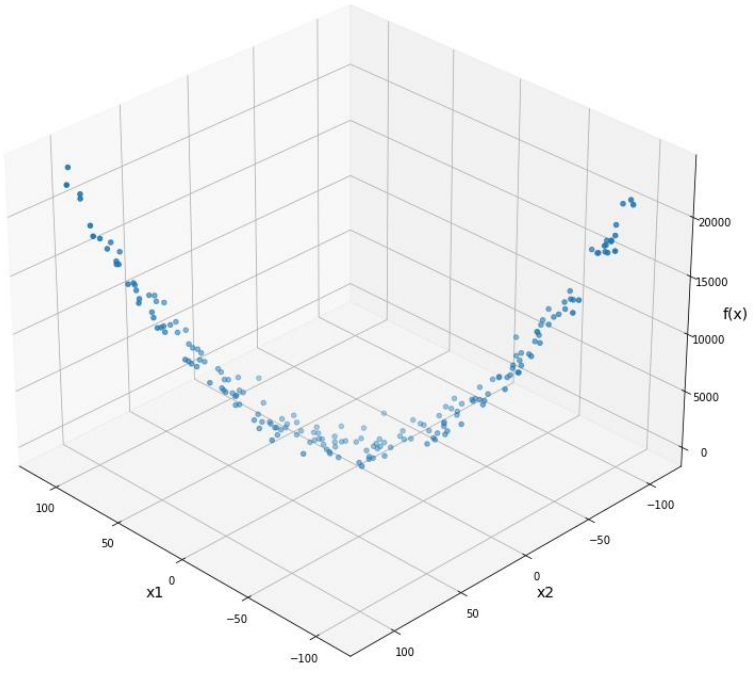
Q4. Consider the regression problem shown in the picture 2 below and answer each point.

Q4.1 Do you think a model of the family f(x, theta) = $\theta_0 + \theta_1 * x_1 + \theta_2 * x_2$ is a good choice for such task? Why?

A4.1 The simple linear model $f(x, \theta)$ would not be a good fit for the data. It is evident that the data does not follow a linear trend. The linear model would lead to underfitting where the model is not able to capture the underlying patterns. The relationship between the inputs variables and the dependent output variable seems to be non-linear, describing a quadratic interaction on both x_1, x_2 resulting in a parabolic shape. A linear relationship among input variables as in $f(x, \theta)$ would not capture the curvature form of the data.

Q4.2 Do you think using a feed-forward neural network would improve the results?

Figure 2: Regression problem



A4.2 Using an FFNN would be a good choice for this problem task. It is for sure better than the simple model with linear relationship between independent input variable x_1, x_2 and dependent output variable $f(x, \theta)$. In general, with FFNNs we can leverage the *Universal Approximation Theorem (1991)* which states that an FFNN with a single hidden layer containing a finite number of neurons and a linear ouput neuron can approximate any continuous function on compact subsets of $\mathbb{R}^n$. Moreover, FFNN are highly flexible and can approximate virtually any function, so complex non-linear relationship patterns, which are hard for linear and polynomial regression, are handled through the encapsulated abstraction of the neurons in the hidden layer(s). In this case, the parabolic shape of the data would be captured, as FFNN are a common choice for regression tasks with non-linear data.