Machine Learning
Università della Svizzera italiana

Assignment 1

De Castelli Fabrizio

May 5, 2023

Tasks

Task 1

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

a. Write in the report the formula of the model substituting parameters $\theta_0, \ldots, \theta_4$ with the estimates you've found:

$$f(\mathbf{x}, \boldsymbol{\theta}) = 1.31635 - 0.05128 \cdot x_1 - 0.57659 \cdot x_2 + 0.42026 \cdot \sin(x_2) + 0.03686 \cdot x_1 \cdot x_2$$

b. Evaluate the test performance of your model using the mean squared error as performance measure.

$$\textit{MSE on test set}: \ 0.7516362518990539$$

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.

$$\textit{MSE on test set}: \ 0.07060475599724211$$

b. Compare your model with the linear regression of Task 1. Which one is statistically better?

Statistically, it is better the non-linear family because the linear family implicitly assumes that there is a linear relationship between features and targets and it is not able to learn it. Anyway, I implemented k-fold cross validation to prove that the second model is better than the first one. In the first cell we can see a section that shows this statistical comparison on the whole dataset, based on the Root Mean Squared Error **(RMSE)**.

$$RMSE\ for\ Linear\ Regression : 0.83018$$

$$RMSE\ for\ Polynomial\ Fitting : 0.26014$$

I got the score on MSE with a degree 11 polynomial fitting and since it is smaller by a factor of 4 statistically and by a factor of 10 on the test errror, it is clear that is statistically better. Still, it can be worth considering to implement a Neural Network, to learn complex scenarios where the relationship between features and their targets is not linear.

Task 3 (Bonus)

In the **GitHub repository of the course**, you will find a trained Scikit-learn model that we built using the same dataset you are given. This *baseline* model is able to achieve a MSE of **0.022**, 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.

My Solution
I implemented a Neural Network with two hidden layers:

- Layer 1: 20 neurons and *ReLU* as activation function;
- Layer 2: 13 neurons and *tanh* as activation function.

I trained the model for 1200 epochs without early stopping. I got to the optimal number of epochs after having tried different options. This number, for my NN, was outperforming all other configurations I tried.
I managed to get **0.0148** MSE on the whole dataset after having transformed input data from 2 to 5 dimensions, as suggested by you in this assignment.
Probably this model performs better because a NN is more flexible at learning features than other models. Plus, the baseline model is a simple reference model used as a benchmarks for other trained models.

Questions

Q1. Training versus Validation

Q1.1 What is the whole figure about?

A1.1 The figure refers to a method called *Early Stopping* used to stop the training once the model is no longer improving its performance on a validation set.

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 The smooth red curve is the average of the validation errors generated by random configuration of initial parameters. We aim to find a model which complexity minimizes the red curve. The green curve is generally decreasing with along the x-axis and represent the error, evaluated with a metric, on the training set during the training of our model. The blue curve is the error we get on a reserved portion of the dataset (validation set). It is decreasing with the improvement of underfitting and increasing the more overfitting we have.

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 The more the number of iteration grows, the more the risk of overfitting increases. That is because we are training our model too much, minimizing its error on the training set and losing generalization power. In the figure, overfitting can be seen in the section (c). In underfitting we are in the opposite situation: the model is not able to fit data in the correct way (it fits only few of them) and we are getting a very high training, validation and expected test error as well. Thus, underfitting can be seen in section (a).
In section (b) there is a compromise of overfitting and underfitting, leading to a better performing model that should be able to generalize more on unseen data.

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

A1.2.b One could say that the optimal model is the one lying on the blue dotted line because it minimizes the difference between the observed validation error and observed training error. But this is not completely true as that model is biased towards some configuration of the dataset, depending on how it is shuffled. The optimal model is probably the one closer to the red dotted line because it is not biased.
Generally, the optimal model is the one that has the best loss performance before the error on the validation set starts increasing (towards section (c)).

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

A1.3 Yes, the approximation risk refers to the model complexity, that is strictly related to overfitting. In the figure we can see that a certain point both the validation and test error are increasing, while the training error is decreasing. This happens because the model is starting to overfit data and lose its generalization power. High overfitting increases the approximation risk.

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 can bring the training error to zero, but with a low probability because all data points are affected by noise. The structural risk is the expected difference between the empirical risk (i.e. the training error) and the true risk (i.e the generalization error), so even if the training error is zero, it is unlikely that the model will perform the same way for never seen data. Because the more the model gets complex, the more overfitting we have and this will affect the test error, so the structural risk.

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)

A1.5 No, because the model didn't stop early and lost its generalization power. Anyway, it can be shown that the number of iteration represent the model complexity. This means that training (if it stops at the end of the plot) made the model become too complex.

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 In this case, we are expecting both the training and test error not to decrease because another linear component have been introduced in the model and it will just offset or change the slope of the line. Still, they might decrease a little bit as we don't know the task but they will not statistically improve the model.
Coefficients will adjust based on the relevance of the new regressor. If the regressor is relevant it will have a larger coefficients, while if it's not relevant it will have a small coefficient. Other coefficients will adjust based on the other features' coefficients.

Q2.2 $x_3 = x_1 \cdot x_2 \cdot x_2$

A2.2 In the case the model can learn non linear features and thus decrease both the test and training error. Obviously, it depends on data... θ_3 in this will have high absolute value if that feature can correctly approximate the unknown function g.

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

A2.3 In this case, since x_3 is independent from y, adding it to the model will not help to improve the model. The coefficient of x_3 is going to be very close to zero because x_3 has nothing to do with y.

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

A2.3 If we were using lasso regression, all irrelevant coefficient would be shrunk to zero as irrelevant to the model so the training procedure will automatically cut off irrelevant parameters for the model. This happens because we want to find a compromise between the training and parameter vector with small magnitude by minimizing the function

$$E(\boldsymbol{\theta}) = \sum_{i \leq n} L(y(x_i), f(x_i, \boldsymbol{\theta})) + \lambda ||\boldsymbol{\theta}||_1$$

λ is the weight we want to give to the regularization term.
Now, since the error depends on the sum of the absolute values of the parameter vector, some coefficients might go to zero and so any regressor we added will automatically be ignored thank to this regularization term.

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

A2.4 Both in Lasso and Ridge regression, we want to regularize parameter vectors with low magnitude to control overfitting. In Ridge regression we shrink the squares of components of the parameter vector, and the error becomes:

$$E(\boldsymbol{\theta}) = \sum_{i \leq n} L(y(x_i), f(x_i, \boldsymbol{\theta})) + \lambda ||\boldsymbol{\theta}||_2$$

This won't totally shrink irrelevant features to zero as $\theta_i^2 < \theta_i, \forall \theta_i \in [0, 1]$, for some component θ_i (we are looking at squares). Additionally, this kind of regularization (L_2) penalizes large components to reduce overfitting (the square of a large number is even higher).
On the other hand, Lasso regularization term is used to avoid overfitting by setting irrelevant coefficients to zero. More clearly, *irrelevant parameters* are the ones that have small values.
Usually, Ridge is used in large models were the number of features is high and Lasso is used in smaller models were the number of features is low.

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?

A3.1 The perceptron is not performing well because it can only solve linearly separable problems. In particular, the perceptron cannot solve the *xor* problem but only the *or* problem. Clearly, the one showed in the picture does not have linearly separable classes.
Generally, if we have a network composed of linearities only, we lose power of computation because their composition is still a linear combination of the input.

Q3.2 Would you expect better luck with a neural network with the activation function $h(x) = -x * e^{(-2)}$ for the hidden units?

A3.2 This activation function is still linear, so I am not expecting explicitly better result. In fact:

$$h(ax + b) = -(ax + b) * e^{(-2)} = -ax * e^{(-2)} - b * e^{(-2)} = a * h(x) + h(b)$$

We are not sure it will worsen or improve the model, but if it improves it, the gain might not be too evident. It might be the case to introduce another activation function that is non-linear.

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

A3.3 The main differences are:

- They have different activation functions. The perceptron has a Step activation function, which is not continuous thus not differentiable:

$$f : \mathbb{R} \longrightarrow \{0, 1\},$$

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

while the logistic regression neuron has a sigmoid as activation function, which is differentiable:

$$sigmoid(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

– The role of the neurons are different. The perceptron returns as output a value that represents a class assignment if it is higher than a certain threshold, but it doesn't have other interpretation for the output.
  On the other hand, the logistic regression neuron output the probability that the given input can be assigned to a certain class. This comes from the construction of the sigmoid, because $\forall x \in \mathbb{R} \,.\, \sigma(x) \in (0, 1)$

– The perceptron neuron is trained by updating weights iteratively, while the other one can be trained with *Maximum Likelihood* estimation.

The main similarities are:

– They are both used in architectures to solve binary classification tasks;

– They both generate a decision boundary for the classes.