Machine Learning
Università della Svizzera italiana

Assignment 1

Maggioni Claudio

May 7, 2021

The assignment is split into two parts: you are asked to solve a regression problem, and answer some questions. You can use all the books, material, and help you need. Bear in mind that the questions you are asked are similar to those you may find in the final exam, and are related to very important and fundamental machine learning concepts. As such, sooner or later you will need to learn them to pass the course. We will give you some feedback afterwards.
!! 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.

Regression problem

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 2500 words (formulas, tables, figures, etc., do not count as words) and do not include any code in the report.

Premise on data splitting

In order to perform a correct statistical analysis, I have splitted the given datapoints in a training set, a validation set and a test set. The test set has 200 points (10% of the given datapoints), the validation set has 180 data points (10% of the remaining 90% of the given datapoint) while the training set has 1620 datapoints (all the remaining datapoints).

For the linear model, I will perform the linear regression using both the training and the validation datapoints (since there is no need to choose hyperparameters since the given family of models requires none). For the nonlinear model, which is a feedforward NN in my case, I used the training set to fit the model and I have used the validation set in order to choose the architecture of the neural network.

Task 1

Use the family of models $f(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_1 \cdot x_2 + \theta_4 \cdot \sin(x_1)$ to fit the data. Write in the report the formula of the model substituting parameters $\theta_0, \ldots, \theta_4$ with the estimates you've found:

$$f(\mathbf{x}, \boldsymbol{\theta}) = 3.13696 - 0.468921 \cdot x_1 - 0.766447 \cdot x_2 + 0.0390513 \cdot x_1 x_2 - 0.918346 \cdot \sin(x_1)$$

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

On the 200-datapoint test set described before, the MSE of this linear model is 1.22265. This result can be reproduced by running `src/statistical_tests.py`.

Task 2

Consider any family of non-linear models of your choice to address the above regression problem. Evaluate the test performance of your model using the mean squared error as performance measure. Compare your model with the linear regression of Task 1. Which one is **statistically** better?

The model I've chosen is a feed-forward neural network with 2 hidden layers:

- one 22-neuron dense layer with hyperbolic tangent activation function;
- one 15-neuron dense layer with sigmoidal activation function;

Finally, the output neuron has a linear activation function. As described before, the validation set was used to manually tailor the NNs architecture.

The network was trained using the `adam` optimizer with 5000 maximum number of epochs and with an early stopping procedure that has a patience of 120 epochs.

The fitted model has these final performance parameters:

- Final MSE on validation set ≈ 0.0143344
- Final MSE on the test set ≈ 0.0107676

I conclude the NN non-linear model is better than the linear regression model because of the lower test set MSE. Since the test set is 200 elements big, we can safely assume that the loss function applied on the test set is a resonable approximation of the structural risk for both models.

In order to motivate this intuitive argument, I will perform a student's T-test to show that the non-linear model is indeed statistically better with 95% confidence. The test was implemented in `src/statistical_tests.py` and it has been performed by spitting the test set, assuming both resulting sets have an expectation of the noise equal to 0, and by computing mean, variance and T-score.

The results of the test are the following:

- T-test result: 5.37324
- Linear model noise variance: 6.69426
- Feedforward NN noise variance: $0.348799 \cdot 10^{-3}$

The test is clearly outside of the 95% confidence interval of $[-1.96, 1.96]$ and thus we reject the null hypothesis. Since the variance is lower for the NN we conclude that this model is indeed the better one statistically speaking.

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 **0.0194**, when evaluated on the test set. You will get extra points if the test performance of your model is better (i.e., the MSE is lower) than ours. Of course, you also have to tell us why you think that your model is better.

In order to understand if my model is better than the baseline model, I performed another Student's T-test re-using the subset of the test set I've used for my linear model. The results of the new test are the following:

- T-test result: 1.1777 (for a p value of 0.2417396)
- Baseline model noise variance: $0.366316 \cdot 10^{-3}$
- Feedforward NN noise variance: $0.348799 \cdot 10^{-3}$

This test would accept my model as the better one (as the variance is lower) with a 75% confidence interval, which is far from ideal but still can be significant and could be a good chance at having a better model on the hidden test set.

To further analyze the performance of my NN w.r.t. the baseline model, I have computed the MSE over my entire 200-datapoints test set for both models. Here are the results:

- MSE on entire test set for baseline model: 0.0151954
- MSE on entire test set for feedforward NN model: 0.0107676

My model shows lower MSE for this particular test set, but even if this test wasn't used to train both sets of course the performance may change for a different test set like the hidden one. The T-test performed before suggests that my model might have a decent chance at performing better than the baseline model, but I cannot say that for sure or with a reasonable (like 90% or 95%) confidence.

Questions

Q1. Training versus Validation

1. **Explain the curves' behavior in each of the three highlighted sections of the figures, namely (a), (b), and (c).**

   In the highlighted section (a) the expected test error, the observed validation error and the observed training error are significantly high and close toghether. All the errors decrease as the model complexity increases. In (c), instead, we see a low training error but high validation and expected test error. The last two increase as the model complexity increases while the training error is in a plateau. Finally, in (b), we see the test and validation error curves reaching their respectively lowest points while the training error curve decreases as the model complexity increases, albeit in a less steep fashion as its behaviour in (a).

2. **Is any of the three section associated with the concepts of overfitting and underfitting? If yes, explain it.**

   Section (a) is associated with underfitting and section (c) is associated with overfitting.

   The behaviour in (a) is fairly easy to explain: since the model complexity is insufficient to capture the behaviour of the training data, the model is unable to provide accurate predictions and thus all MSEs we observe are rather high. It's worth to point out that the training error curve is quite close to the validation and the test error: this happens since the model is both unable to learn accurately the training data and unable to formulate accurate predictions on the validation and test data.

   In (c) instead, the model complexity is higher than the intrinsic complexity of the data to model, and thus this extra complexity will learn the intrinsic noise of the data. This is of course not desirable, and the dire consequences of this phenomena can be seen in the significant difference between the observed MSE on training data and MSEs for validation and test data. Since the model learns the noise of the training data, the model will accurately predict noise fluctuations on the training data, but since this noise is completely meaningless information for fitting new datapoints, the model is unable to accurately predict for validation and test datapoints and thus the MSEs for those sets are high.

   Finally in (b) we observe fairly appropriate fitting. Since the model complexity is at least on the same order of magnitude of the intrinsic complexity of the data the model is able to learn to accurately predict new data without learning noise. Thus, both the validation and the test MSE curves reach their lowest point in this region of the graph.

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

   Depending on the scale and magnitude of the x axis, there could be significant approximation risk. This can be observed in subfigure (b), namely by observing the difference in complexity between the model with lowest validation error and the optimal model (the model with lowest expected test error). The distance between the two lines indicated that the currently chosen family of models (i.e. the currently chosen gray box model function, and not the value of its hyperparameters) is not completely adequate to model the process that generated the data to fit. High approximation risk would cause even a correctly fitted model to have high test error, since the inherent structure behind the chosen family of models would be unable to capture the true behaviour of the data.

4. **Do you think that by further increasing the model complexity you will be able to bring the training error to zero?**

   Yes, I think so. The model complexity could be increased up to the point where the model would be so complex that it could actually remember all x-y pairs of the training data, thus turning the model function effectively in a one-to-one direct mapping between input and output data of the training set. Then, the loss on the training dataset would be exactly 0. This of course would mean that an absurdly high amount of noise would be learned as well, thus making the model completely useless for prediction of new datapoints.

5. **Do you think that by further increasing the model complexity you will be able to bring the structural risk to zero?**

   No, I don't think so. In order to achieve zero structural risk we would need to have an infinite training dataset covering the entire input parameter domain. Increasing the model's complexity would actually make the structural risk increase due to overfitting.

Q2. Linear Regression

Comment and compare how the (a.) training error, (b.) test error and (c.) coefficients would change in the following cases:

1. x_3 **is a normally distributed independent random variable** $x_3 \sim \mathcal{N}(1, 2)$

   With this new variable, the coefficients θ_1 and θ_2 will not change significantly for the new optimal model. Training and test error behave similarly, although the training error may be higher in the first iteration of the learning procedure. All this variations are due to the fact that the new variable x_3 is completely independent from x_1 and x_2, and consequently from y. Therefore, the model will "understand" that x_3 contains no information at all and thus set θ_3 to 0. This effect would be achieved even more quickly by using Lasso instead of linear regression, since Lasso tends to set parameters to zero when their linear regression optimal value would be already close to 0.

2. $x_3 = 2.5 \cdot x_1 + x_2$

   With this new variable, the coefficients would indeed change but test and training error would stay the same. Since x_3 is a linear combination of x_1 and x_2, then we can rewrite the model function in the following way:

   $$f(x, \theta) = \theta_1 x_1 + \theta_2 x_2 + \theta_3(2.5x_1 + x_2) = (\theta_1 + 2.5\theta_3)x_1 + (\theta_2 + \theta_3)x_2$$

   This shows that even if the value of θ_1 and θ_2 would change if this term is introduced, the solution that would be found through linear regression would still be effectively equivalent w.r.t. effectiveness and MSE to the optimal model for the original family of models.

3. $x_3 = x_1 \cdot x_2$

   If the underlying process generating the data would also depend on an $x_1 \cdot x_2$ operation, then this additional input variable would change the parameters, improve the training error, and depending on if the impact of this quadratic term on the original data-generating process is small or big, it would slighty or considerably improve the test error.

Essentially, this parameter would had useful complexity to the model, which may be beneficial if the model is underfitted w.r.t. number of variables in the linear regression function, or otherwise detrimental if the model is correctly fitted or overfitted already.

Q3. Classification

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?**

   The classification problem in the graph, according to the data points shown, is quite similar to the XOR or ex-or problem. Since in 1969 that problem was proved impossible to solve by a perceptron model by Minsky and Papert, then that would be quite a motivation in front of my boss.

   On a more general (and more serious) note, the perceptron model would be unable to solve the problem in the picture since a perceptron can solve only linearly-separable classification problems, and even through a simple graphical argument we would be unable to find a line able able to separate yellow and purple dots w.r.t. a decent approximation simply due to the way the dots are positioned.

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

   First of all we notice that that activation function is still linear, even if a $-e^{-2}$ scaling factor is applied to the output. Since the problem is not linearly separable, we would still have bad luck since the new family of models would still be inadequate for solving the problem.

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

   The perceptron neuron and the logistic regression neuron both are dense neurons that recieve as parameters all values from the previous layer as single scalars. Both require a number of parameters equal to the number of inputs, which are used as scaling factors for the inputs recieved. The main difference between the neurons is the activation function: the perceptron uses an heavyside activation function, while the logistic regression neuron uses a sigmoidal function, providing a probabliistic interpretation of the output w.r.t. the classification problem (i.e. by instead of providing just a "in X class" or "not in X class" indication, the logistic regression neuron can output in the continuous $[0, 1]$ range a probabliity of said element being in the class).