Machine Learning Assignment 1

Nikolaos Gialitsis (Erasmus)

May 2019

1 Linear Regression

1.1 Preprocessing

First ,I perform a normalization over the data in order to simplify the calculations .Also, I split the input data into a training and a validation set , the validation set containing 10% of the initial number of samples

1.2 K-fold cross-validation

Next, I perform a k-fold validation with k=10 by partitioning the initial data into ten sets, using one of them for validation and the rest of them for training. I decided to use a k-fold validation in order to get a rough estimate of the model's accuracy and to have enough measurements to assess the model's performance when I will have to compare it with another model. For every round of training I perform the following procedure:

1. Transform each point to an instance, corresponding to the family of models that could describe it , using the format specified in the exercise's description $f(x, \theta) = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \theta_3 * x_1 * x_2$ (with unknown parameters θ)

2. Estimate the optimal parameters $[\theta_0, \theta_1, \theta_2, \theta_3]$ by fitting the family of models to the data instances using the linear regression function that exists in sklearn.The metric used to decide which family performs the best is the Mean Square Error

3. After completing the procedure , we have reduced the family of models to a single linear model with known parameters $\theta = [\theta_0, \theta_1, \theta_2, \theta_3]$

4. Output learned function that represents the data with estimated parameters

5. Predict new estimated values by running the model with the parameters derived from training to the validation set.

6. Print accuracy score and mean square error

7. Store the model's accuracy and error in a vector

At the end of the k-fold validation , I have produced two 10-dimensional vectors, containing the model accuracies and mean square errors, one entry for each run. Then , I print the average mean square error as well as the variance of my estimations and I can make an assumption about the quality of my model by inducing that the error will be within the confidence interval $ConfidenceInterval = [\mu - \sigma, \mu + \sigma]$

1.3 Training and storing the model

Finally I use the whole input set as a training set in order to create a plot that shows the difference between the plane that best describes the input data and the estimated one. I print the mean square error of my model and its accuracy by comparing the estimated values with the real ones. Finally, I save the trained sklearn model in a .pickle format.

2 Non-linear regression

2.1 Choice of non-linear model

For this non-linear family of models I chose to use a Feedforward Neural Network with the following architecture:

1. one input layer

2. one output layer with one neuron

3. eleven hidden layers with rectified linear units (ReLU)as activation functions :

I chose Neural Networks since because of the Universal approximation theorem which states that "a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function".
Universal Approximation Theorem

2.2 Neural Network Architecture

Even though the approximation theorem states that one hidden layer is enough in order to approximate most functions, I decided to use a deep architecture with 11 hidden layers for the following reasons. (The following information I found on *How to configure a neural network* encouraged my decision.)

- A single-layer neural network can only be used to represent linearly separable functions. This means very simple problems where, say, the two classes in a classification problem can be neatly separated by a line. If your problem is relatively simple, perhaps a single layer network would be sufficient. Most problems that we are interested in solving are not linearly separable.Since the performance of the linear model is generally considered bad , I have strong feelings that the problem is not linearly separable.This is why I chose to increase the non-linear behavior of my model by adding more layers.

- Empirically, greater depth does seem to result in better generalization for a wide variety of tasks. [...] This suggests that using deep architectures does indeed express a useful prior over the space of functions the model learns.

- Although a single hidden layer is optimal for some functions, there are others for which a single-hidden-layer-solution is very inefficient compared to solutions with more layers.

2.3 Parameter Tuning

2.3.1 Nodes per Layer

I decided to start with a quite high number of nodes for the first hidden layer , about 50% of the input size , but I intentionally rounded the number to a power of 2 (2^9), in an attempt to simplify the matrix multiplications performed by the neurons.

Then, for the deeper levels I choose lesser powers of 2 , since the goal is to end on a single output layer , so it only makes sense that the number of nodes per layer will be reducing at each hidden level. Then I found out that adding intermediate layers produces a better result, so I added layers with 200 and 100 nodes.

2.3.2 Activation Function

I chose ReLU because

1. ReLu is nonlinear in nature(And combinations of ReLu are also non linear, so this means that it is possible to stack layers)
2. it is a good approximator. Any function can be approximated with combinations of ReLu)
3. ReLU has sparse activation(fewer neurons are firing) than other activation functions(e.g sigmoid) and this means that is faster and easier to compute

2.3.3 Optimizer

I chose adam as an optimizer because

1. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods.
2. In practice Adam is currently recommended as the default algorithm to use and this is the reason it is the default optimizer for keras

2.3.4 Epochs and Batches

In order to choose the number of epochs I used the Early Stopping method ,where the metric is the validation score on the testing set. If the model had stopped improving , it would run for 300 more epochs and if it did not find any better configuration it would halt. I ran the model multiple times with different number for batches and I concluded that 32 batches is a good deal between speed and efficiency.

2.4 Cross Validation

For the implementation:

I used the keras regressor with 10-fold cross validation.
I reported the mean MSE,the variance and the expected MSE(confidence interval)
Finally I fit the whole dataset into my model and predict $\hat{Y}$
I compare the predicted values with the true ones (Y) and I print the accuracy.

2.5 Training and testing

At first , I performed a Principal Component Analysis on the input dataset , which shows that 1 parameter is able to explain the data. So I reduce the input data from two dimensions to one. Then I split the data into a training and validation set , fitting the model into the training set and prediction the values for the validation set. I output the training error,the test error and the error on the whole input set.

3 Model Comparison

Finally, in order to compare my models (the linear regressor and the non-linear regressor) I compute a t-test.

Using the values I have collected from the two 10 k-fold validations , I conclude that the probability that the Feedforward Neural Network has the same mean accuracy as the Linear model is less than 0.05 ($\approx$0.02 , I reject the Null Hypothesis.

This way I have shown that one of the models outperforms the other one with statistical significance.
By comparing the means of the scores I conclude that the linear model outperforms the non-linear one.