AIML - CS 335

Lab 5: Intro to Linear Regression

01 September 2022

Lecturer: Abir De

Important: Please read the instructions mentioned in the questions carefully. We have provided boilerplate code for each question. Please ensure that you make changes in the areas marked with TODO.

Please **read the comments in the code carefully** for detailed information regarding input and output format.

1 Linear Regression

We studied Linear Regression model in the class. The setup is as follows

Dataset : $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^{n}$

was the set of all points over which the analysis is done

Input vector for i^{th} sample $x^{(i)} \in \mathrm{R}^d$

is the vector denoting the features of the i^{th} sample of data

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \\ . \\ . \\ x_d^{(i)} \end{bmatrix}$$

d denotes the no of features in each vector $x^{(i)}$

Label $y^{(i)} \in \mathbb{R}$ denotes the associated label to each sample $x^{(i)}$

weight $w \in \mathrm{R}^d$ was the vector related to the weights assigned to each individual feature

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ . \\ . \\ w_d \end{bmatrix}$$

Hypothesis function $h_w(x) = w^T x$ was the hypothesis used to predict the label of any general point x

For the mathematical convenience its useful to define the following two quantities, which will be used in this question

1. **Design Matrix** $X \in \mathbb{R}^{nXd}$ which is nothing but all the input samples of the set stacked one below the other. More specifically

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & . & . & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & . & . & x_d^{(2)} \\ x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & . & . & x_d^{(3)} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & . & . & x_d^{(n)} \end{bmatrix}$$

2. $Y \in \mathrm{R}^n$ which is nothing but all the labels of the set stacked one over the other in a vector. More specifically

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ . \\ . \\ y^{(n)} \end{bmatrix}$$

In order to find the optimum value of w we needed to minimise a given loss function, which is generally denoted by $\mathcal{L}(w)$

There can be many loss functions that will yield different w^* corresponding to their respective minimised loss. However what interests us here is the implementation of two specific losses namely

L-1 Loss

$\mathcal{L}_1(w) = \frac{1}{n}\sum_{i=1}^{N} |y_i - w^T x^{(i)}|$

L-2 Loss

$\mathcal{L}_2(w) = \frac{1}{n}\sum_{i=1}^{N} (y_i - w^T x^{(i)})^2$

Note that the losses are normalised

However doing the calculations one sample at a time might take a lot of time if the dataset is large.

Worry Not !!! You already know the solution. $Torch$ allows us to perform these calculations in parallel. In fact you even had a lab question regarding this property earlier.

So all you have to do for this question is
In the file **Q1.py** complete the following tasks

TASK 1

```
function : def w_closed_form(X, Y)

Write the closed form solution for l-2 loss in terms of
X(design matrix) and Y(labels vector). Do not use a loop.
Use torch vectorization
```

TASK 2

```
function : def l1_loss(X, Y, w)

Write the l-1 loss in terms of X(design matrix) and
Y(labels vector). Do not use a loop.
Use torch vectorization
```

TASK 3

```
function : def l2_loss(X, Y, w)

Write the l-2 loss in terms of X(design matrix) and
Y(labels vector). Do not use a loop.
Use torch vectorization
```

TASK 4

function : def l2_loss_derivative(X, Y, w)

Derive the gradient of the loss in pen and paper by differentiating l2-loss(You might need a little matrix calculus)
Write the derivative of l-2 loss in terms of X(design matrix) and Y(labels vector). Do not use a loop. Use torch vectorization
Please note that the return value will be a [dx1] vector

TASK 5

function : def train_model(X_train, Y_train, X_test, Y_test)

The function runs a gradient descent algorithm to return:
w : np.float64 array(vector) of size(d,1) ,
the final optimised w
epochs : Total iterations it take for algorithm to converge
test_err : python list containing the l2-loss at each epoch

For a sanity check you can run this code without any changes to check whether the closed form solution you calculated in TASK 1 matches with the w found by the gradient descent.

Your task however is to append the python list : test_arr at each epoch with the l2-loss calculated at test dataset. Once you do this you can move on to:

TASK 6

function : In the main code itself

We will observe the strategy of EARLY STOPPING here. While training the algorithm the test error reduces upto certain epochs but then it starts increasing. In the earlier task you have all the test errors for all the epochs.

```
We run a code to find e_star: the epoch after which test
error increases If everything was calculated correctly in
TASK5 then you just need to uncomment the code of TASK 6 to
find the early stopping epoch. See the pdf plot that is
generated
```

2 Ridge Regression

Ridge regression is a model tuning method that is used to analyse any data that suffers from multi-collinearity. This method performs L2 regularization. When the issue of multi-collinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

In the file **Q2.py** complete the functions (marked by TO DO). Through the functions you are expected to:

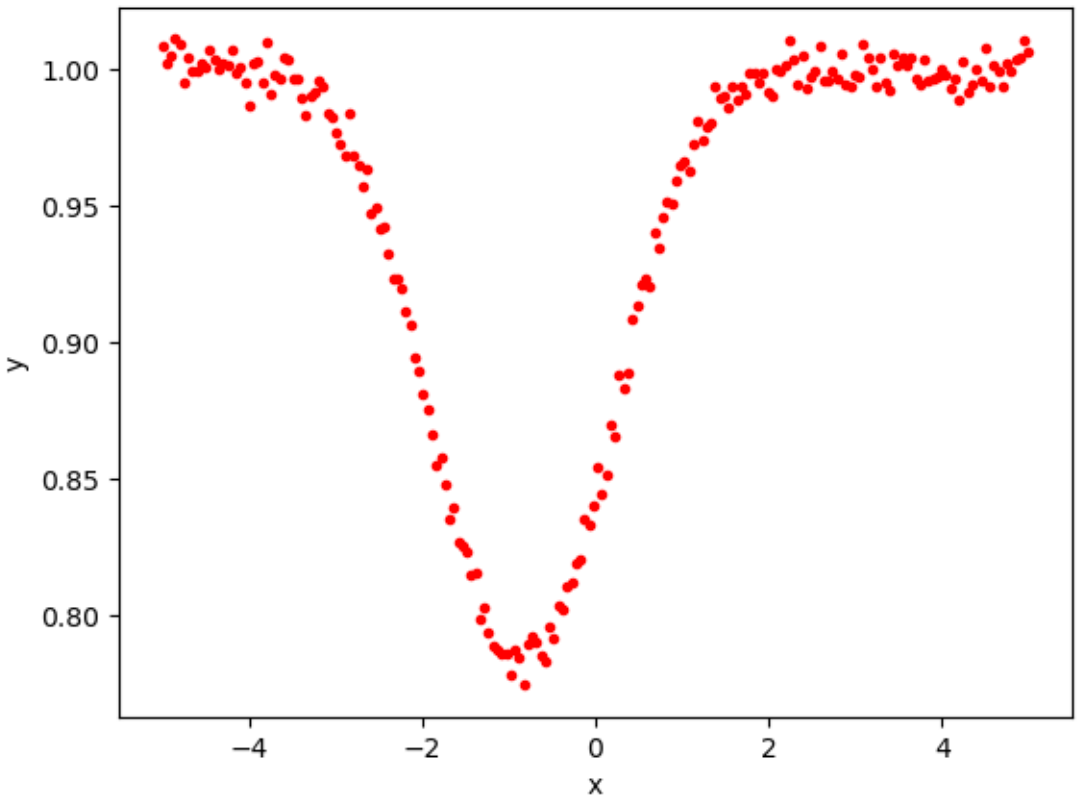
1. Find the closed form solution of the weight vector for ridge regression.
2. Find the test errors corresponding for a particular weight vector. Please use MSE (Mean-swuared error) while computing the test error.
3. Plot the test-errors versus λ.

Ridge regression uses a hyperparameter. λ. In this exercise you are also expected to try out different values of λ with the aim of minimizing the error on the test-set. Plot the test-error versus the lambda value and save the generated plot as **Q2.png**. The plot should be saved in the same directory as **Q2.py** Be sure to label the plot well. Choose the values of λ such that the underlying trend is clearly visible.

Note: There is some code to be written in the *main* function too.
Note: Students are to use PyTorch and Matplotlib for the above implementations. They aren't allowed to use torch.

3 Non Linearity

Observe the following plot of x vs y:



The data that produces this plot is given in *data.npy inside the Q3 directory*. This data is not suitable for performing linear regression, even after addition of features. Instead, it is known that the data comes from a function of the following form with some noise:

$$y = f(x) = w_4 \cdot tanh(w_1 x + w_2 x^2 + w_3) + w_5$$

Along with it are given two files - *nonlinear_regression.py* and *print_result.py*. The first file implements fitting a function of the form given above to the data given. You have to complete the code wherever it is indicated by comments.

For purposes of verification, the code produces 11 plots. If you've implemented everything correctly, these should show the predictions gradually moving towards the data. These plots are for you and you need not include them in your final submission - but we won't penalise you if you do.

Make sure that running *print_result.py* outputs 2 lists. These are the reproducible results you've obtained. We should obtain the same results on running your code as well.

NOTE : You need not perform any optimization. The fitting (optimizing) code is handled inside the `train` function. Notice the `nn.Parameter` declarations in the constructor for `NLModule` - these register declared variables as parameters in `nn.Module` - from which `NLModule` is derived. Once `forward` is implemented, the dependence of output of the Model on these parameters and the `loss_function` are used to find the optimal parameter values.

4 Assessment

We will be checking the following:

- Correctness of your implementations
- Consistency of plots with hypothesis
- Efficiency of your code - **vectorize wherever possible**
- Reproducibility of your results wherever expected

5 Submission instructions

Make changes only in the places mentioned in comments. Do not modify anything else. Finally, place the 3 folders, namely `Q1`, `Q2` and `Q3` inside a folder named `<ROLL_NUMBER>_L5` and compress it to a tar file named `<ROLL_NUMBER>_L5.tar.gz` using the command
`tar -zcvf <ROLL_NUMBER>_L5.tar.gz <ROLL_NUMBER>_L5`
Submit the tar file on Moodle. The directory structure should be -

```
<ROLL_NUMBER>_L5
| - - - - Q1
| - - - - -|- - - - Q1.py
| - - - - Q2
| - - - - -|- - - - Q2.py
| - - - - Q3
| - - - - -|- - - - nonlinear_regression.py
| - - - - -|- - - - print_result.py
| - - - - -|- - - - losses.npy
| - - - - -|- - - - weights.npy
```

Replace `ROLL_NUMBER` with your own roll number. If your Roll number has alphabets, they should be in *"small"* letters.