ML Project #1 Report

Regression

Author: 資工三 406410035 秦紫頤

Setup

Components

- data/: 3 csv files (**data.csv, data_320.csv, sin_data.csv**) as the data be used in the project
- figure/: 7 jpg image files (**data-m-kf.jpg, data-m-loo.jpg, linear-kf.jpg, linear-loo.jpg, regularization-kf.jpg, sine-kf.jpg, sine-kf.jpg, sine-loo.jpg**). These are all the fitting plot results
- dataset.py: To generate data and Dataset class
- main.py: main program
- opt.py: some options you can set prior to the execution of the program
- README.md: README file
- ML-proj1_report.pdf: this report file

Requirements

- Pytorch 1.4.0
- Scikit-learn 0.22.1
- Matplotlib 3.1.3
- Pillow 6.2.1

Execution

1. Locate to the project folder "**406410035_hw1_v1/**"
2. Ensure all the requirements are installed in your environment
3. python3 main.py
4. The error of training, validation and testing will be shown on your screen
5. The fitting plots will all be saved in the folder "**figure/**"

Method Description

Generate Data: data_generator()

1. Choose to generate linear data or sine data
2. **np.linespce()** to generate data points between a specific range
3. **gauss()** to generate random Gaussian noise that will be added to the data points as well
4. y=f(x) -> f(x) based on linear or sine data
5. Save to csv file (**data/**) to ensure every time gets the same data not regenerating new data points

Prepare dataset: class RegressionDataset

Pass in the generate data points to RegressionDataset class to bind each x with its corresponding y as one data. This is for the convenience of splitting the data to training, validation, and testing later on

Split the Data to training, validation, and testing

1. **torch.utils.data.random_split()**: Split the data to training and testing (75% training, 25% testing). Using random_split() is because I want the selection of training and testing data to be random not sequential. Because I write this is the main function of my code so **every time re-execute the code the results will be different**
2. Cross-validation
    - Leave one out: **sklearn.model_selection.LeaveOneOut()**
    - Five folds: **sklearn.model_selection.KFold()**
3. **torch.utils.data.Subset()** split the data by index chosen by LeaveOneOut() or KFold()

Polynomial Term: get_poly_features()

1. Linear Regreesion (degree=1), polynomial regression (degree=5,10,14)
2. Generate all the polynomial feature, ex:degree=5 -> $[x^5, x^4, x^3, x^2, x^1, 1]$ as X

Get the best weight: get_best_weight()

1. Reshape x and y in the convenience of matrix calculation when calculating the best weight
2. $w_{LIN} = \left(X^T X\right)^{-1} X^T y$
3. Implement using two methods:
    - QR decomposition to compute the pseudo-inverse and do the rest the same as the formula says. There is paper saying using QR decomposition can have better results: Regularization Using QR Factorization and the Estimation of the Optimal Parameter
    - Do all the same as the formula says
4. Average all the best weight compute in the training state (try and get a generalize best weight that can fit all the data)
5.

Linear Regression: linear_regression()

Perform linear regression with the training x, and the best weight I compute

Loss: RMSE_loss()

1. $RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$
2. Why RMSE: I search online and see RMSE loss is suitable for calculating the loss for regression problem
3. RMSE loss is simply the square root of MSE loss
4. Add the loss computer in every iteration to total training loss

5. After the training complete average the loss as the output training loss

Fitting plot: draw_fitting_plot()

1. **plt.scatter()**: draw the target data points on the graph
2. **plt.plot()**: plot the predicted curve on to the graph by passing into the weights
3. Save the plot as figure save in **"figure/"**

Detail Procedure Step

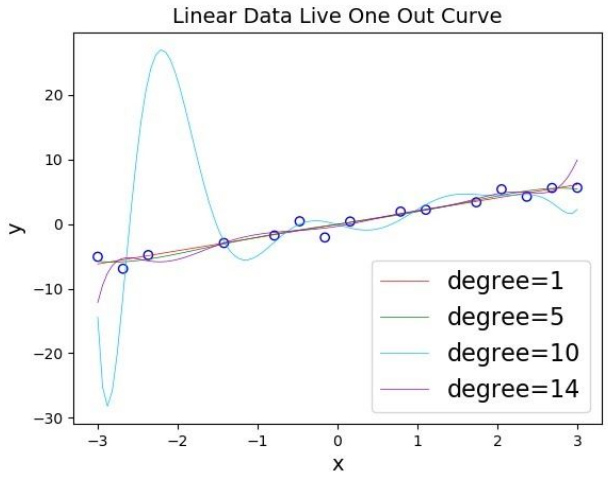
1. Generate the data
    - 20 data points will work on most problem
    - When comparing how will the number of data points affects the accuracy: generate 320 data points and choose the number of data points (m) needed from it
2. Prepare Dataset
3. Split the data to training, validation, and testing
4. Perform training
    - Compute best weight (without regularization: lambda=0, with regularization: lambda>0)
    - Linear regression to predict output
    - Compute loss
5. Validation and Testing: same as training except no need to compute the best weight
6. Draw fitting plots
7. All the training, validation and testing error will show in the terminal while executing

Results

Regression on linear data with degree=1,5,10,14

Leave One Out

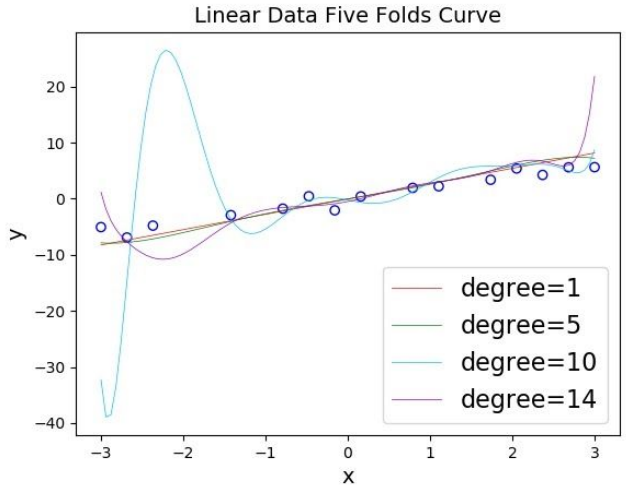
Degree	Training Loss	Validation Loss	Testing Loss
1	0.0591	0.6960	0.1779
5	0.0538	1.1657	0.1513
10	0.2542	35.4669	2.9379
14	0.1185	15.1026	0.1128



Figure(1.1) linear-loo.jpg

Five Fold

Degree	Training Loss	Validation Loss	Testing Loss
1	0.5350	0.8923	0.2477
5	0.5363	0.9266	0.2648
10	0.5517	7.8812	2.8535
14	0.5611	9.5415	0.5596

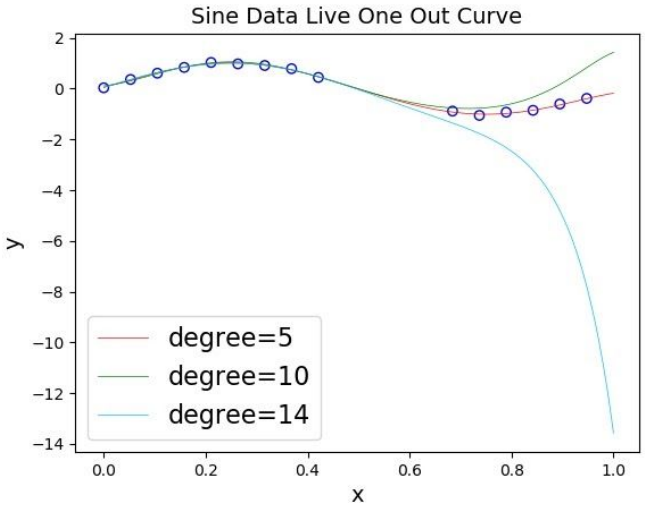


Figure(1.2) linear-kf.jpg

Regression on sine curve data with degree=5,10,14

Live One Out

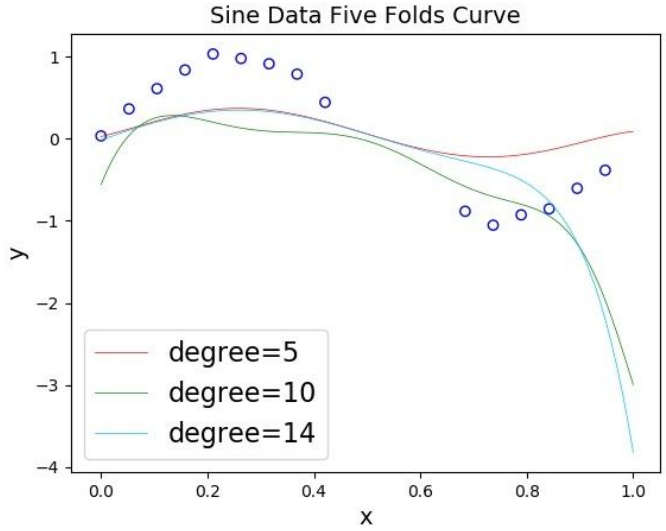
Degree	Training Loss	Validation Loss	Testing Loss
5	0.0027	0.0554	0.0136
10	0.0801	1.1556	0.1330
14	0.3034	4.3546	1.2098



Figure(2.1) sine-loo.jpg

Five Fold

Degree	Training Loss	Validation Loss	Testing Loss
5	0.0999	0.1106	0.0628
10	0.3230	0.3568	0.2660
14	0.1606	0.7992	0.3423

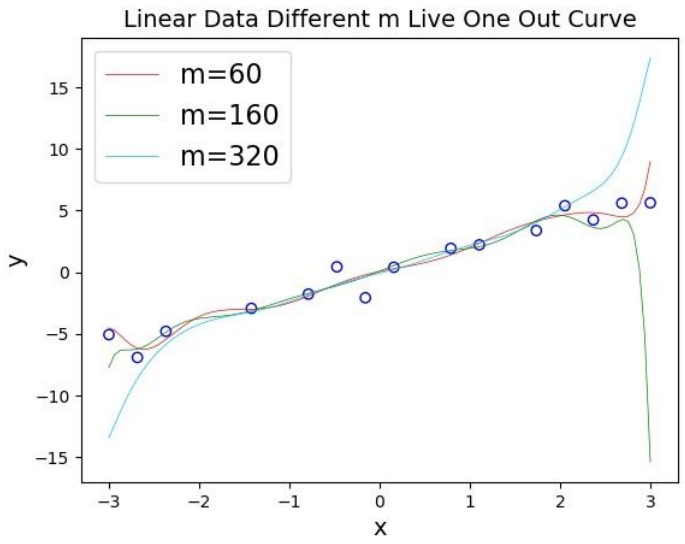


Figure(2.2) sine-kf.jpg

Regression on different training data size m

Live One Out

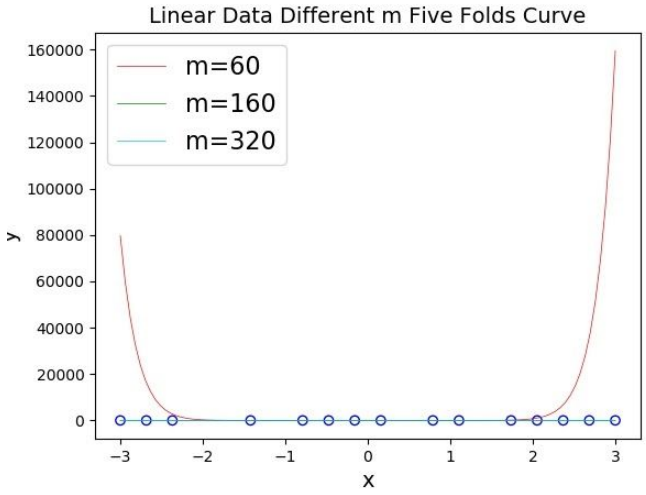
m	Training Loss	Validation Loss	Testing Loss
60	0.0306	2.0622	0.0928
160	0.0190	3.3542	0.0258
320	0.0464	1.1980	0.0322



Figure(3.1) data-m-loo.jpg

Five Fold

m	Training Loss	Validation Loss	Testing Loss
60	0.1279	0.2387	2139.6487
160	0.0205	0.0724	0.0256
320	0.0077	0.0156	0.0328



Figure(3.2) data-m-kf.jpg

Regression with regularization λ=0.001/m, 1/m, 1000/m

λ	Training Loss	Validation Loss	Testing Loss
0.001/m	0.5578	9.6800	966.1201
1/m	5.2470	45.2341	966.3683
1000/m	0.8210	255.8676	964.6821

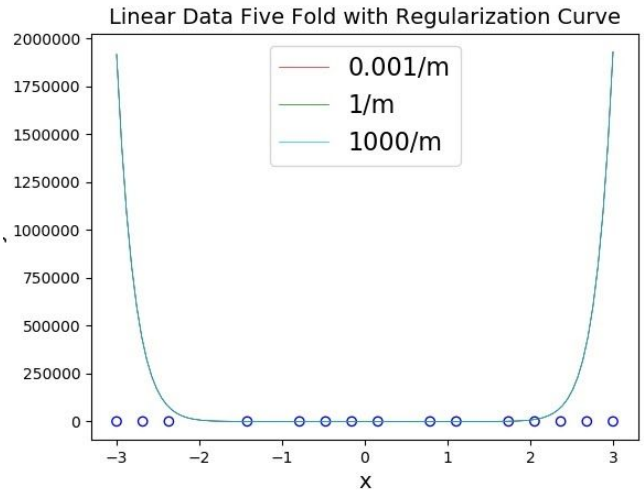


Figure (4) regularization-kf.jpg

Discussion

When testing on different data sizes, I've been comparing the results when m=20,60,160,320. When using five fold as the cross-validation method, there is an unusual result. When m=60 the testing error increase to 2139.6487, it is higher than when m=20. I have re-run the program many times but the results look similar. I couldn't find the reason why.

When adding regularization to regression. The testing loss is all really high no matter which λ I chose. The purpose of regularization is to prevent overfitting that causes the model to perform badly on the testing set. When adding regularization the training error may be higher, both validation and testing error should be lower. But in my case, the training error is higher than original but both the validation and testing error aren't lower, instead, they are much higher than the original. My conclusion for this is maybe the data size isn't large enough to make regularization working.

When seeing the formula $w_{LIN} = \left(X^T X\right)^{-1} X^T y$. I thought of the linear algebra course I took in my freshman year. And I found out that this formula can be solved by QR decomposition also there is some research support that using QR decomposition may lead to a better result: Regularization Using QR Factorization and the Estimation of the Optimal Parameter. But after implementing, I don't find this really better than the original method which I'm really disappointing. If you want to test it out than you can test it by uncommenting the other get_best_weight() function and comment the original one. I think maybe this method will work better on higher-dimensional data.