CSE 353
HomeWork 6

JeongYoon Lee (114133199)

a) Introduction. Brief summary of what you think the assignment is about

This assignment is about showing overfitting and underfitting using Q-th order polynomial transform. We can see that the model will be affected to the number of Q, and I am expecting overfitting as Q gets bigger. And also, we will trying regularization to avoid overfitting. There are various ways to regularize W, but I will try to use lambda value while calculating new weight value.

Lastly, I will try to apply cross validation while training to reduce overfitting when training dataset is not enough. I will compare Leave-one-out algorithm and V-fold algorithm, and I will find the optimal model.

I'm expecting that as the value of V decreases, the computational cost decreases, but the error value will increases.

The number of training set is 20, and the number of testing set is 40, and we should figure out how this training data looks like.

b) Method. Brief outline of your (algorithmic) approach

- Q-th order polynomial transform (Q = {2,3,…19})

  $\boldsymbol{z}n = \phi Q(xn)$ = [1, xn, xn^2 , xn^3 , … xn^Q] T

- Linear regression on a transformed dataset {($\boldsymbol{z}n$, yn)}n=1.

  N: $\boldsymbol{w}Poly$ = ($\boldsymbol{Z}.T\boldsymbol{Z}$)^(-1)$\boldsymbol{Z}.T\boldsymbol{y}$, where $\boldsymbol{Z}$ = [$\boldsymbol{z}1.T$; $\boldsymbol{z}2.T$;… ; $\boldsymbol{z}N.T$]; and the estimated $\boldsymbol{w}poly$ = [$w0$, $w1$, … , wQ].T .

  * For regularization, W_poly* = (Z.T * Z + Lamda*I)^(-1) Z.T*y

- Find err_square with W_poly

  $errsqr$ = ∑($\boldsymbol{w}poly.T$ $\phi(xn)$ − yn)^2

- Cross validation

  Leave-one-out

  → split training set to train set and validation set, use validation set as each one data.

V-fold

→ split training set to train set and validation set into v folds, and then use validation set as each one folds.

Choose model with minimum validation error, and check test error to avoid overfitting.

```
def print_plot(W):
  W_poly = W
  plt.scatter
  Wx = np.arange(-10, 10, 0.0001)
  # Wy = [((-1)*Wm*num)-w[0]/w[2] for num in Wx]
  Wy = 0
  for i in range(Q+1):
    Wy += W_poly[i] * np.power(Wx,i)
  plt.plot(Wx, Wy)

  for i in range(len(X)):
    plt.plot(X[i], Y[i], 'o', color='blue');
  for i in range(len(Test_X)):
    plt.plot(Test_X[i], Test_Y[i], 'o', color='red');
  plt.show();
  return
```

[print_plot function]

I draw plot with blue dots for the train data, and red dots for the Test data.

c) Experiments. Tables and/or pictures of intermediate and final results that convince us that the program does what you think it does.

1. Nonlinear transform and overfitting/underfitting

```
z = [X**0]
X_next = X
for i in range(Q):
  z = np.append(z,[X_next],axis = 0)
  X_next= np.multiply(X,X_next)
z = z.T
```

[Find z value]

```
W_poly =  np.linalg.inv(z.T @ z) @ z.T@Y
```

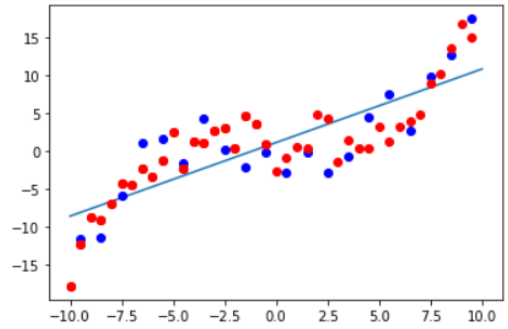
[W_poly]

```
err_sqr = ((W_poly.T)@Test_z.T - Test_Y)**2
```

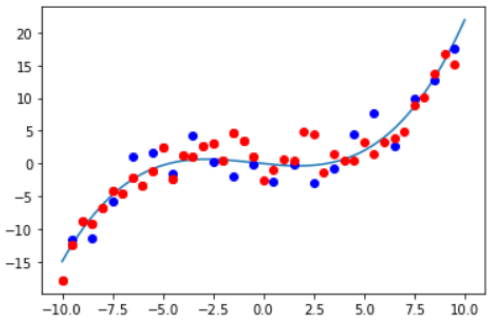
[Find testing error]

- Result

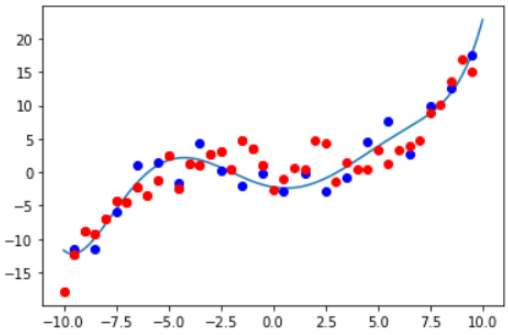
Training Err_square for Q= 1 : 341.02214142047364
Testing Err_square for Q= 1 : 558.5515760833837



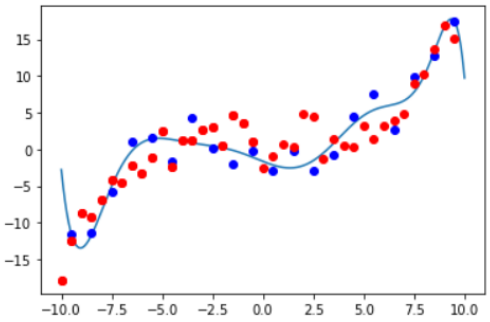
Training Err_square for Q= 3 : 112.7394032059552
Testing Err_square for Q= 3 : 158.18832808759208



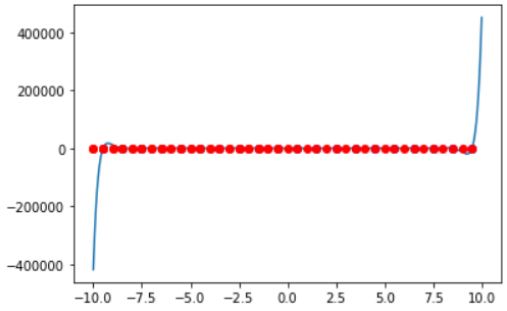
Training Err_square for Q= 6 : 74.72322354660918
Testing Err_square for Q= 6 : 306.29570705425897



Training Err_square for Q= 10 : 60.85094298220477
Testing Err_square for Q= 10 : 523.9301945905759



Training Err_square for Q= 19 : 1443.96760187201
Testing Err_square for Q= 19 : 174564256583.7809



➔ It seems underfitting when Q = 1 since the training error and testing error are both big, and overfitting when Q = 19 because the testing error is too big than the training error, and I can see the W in the plot is quite same with every dots.

➔ When Q is 6, the training error is quite small, ant testing error is acceptable. We can find that the testing error for Q = 3 is also small, but the training error is bigger than model with Q = 6.

2. Regularization

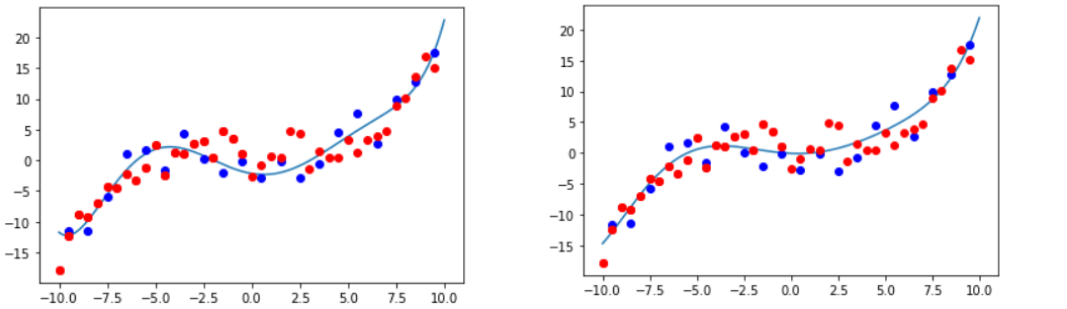
The only different thing between Part 1 and this part is to get new regularized weight.

```
W_poly_star = np.linalg.inv((z.T @ z + L *I)) @ z.T @ Y
```

[How to get regularized Weight]

- Result

```
Training Err_square for Q= 6 , L= 0.1  :  74.73519896431631 Training Err_square for Q= 6 , L= 100  :  99.01336690628663
Testing Err_square for Q= 6 , L= 0.1  :  302.27005458491044 Testing Err_square for Q= 6 , L= 100  :  174.65196623523957
```



➔ We can see that as the value of Lambda increases, the curve of the graph becomes smoother and the testing error decreases.

➔ So it means that the regularization can prevent overfitting in some way.

3. Cross validation

- Leave-one-out

```
def leave_one_out():
  global Y
  start = -1 * num_of_iter
  end = 0
  TrainX = X
  TrainY = Y
  err_sqr_cross = 0
  err_sqr_cross_sum = 0
  err_sqr_training = 0
  err_sqr_training_sum = 0
  for i in range(Y):
    start += num_of_iter
    end += num_of_iter
    start = int(start)
    end = int(end)
    val_arr_X = TrainX[start:end]
    val_index_X = np.arange(start, end, 1)
    train_arr_X = np.delete(TrainX, val_index_X)
    val_arr_Y = TrainY[start:end]
    val_index_Y = np.arange(start, end, 1)
    train_arr_Y = np.delete(TrainY, val_index_X)
    z_cross = [train_arr_X**0]
    X_next_cross = train_arr_X
```

[Split Train dataset to Trainset and validation set into len(Y) pieces]

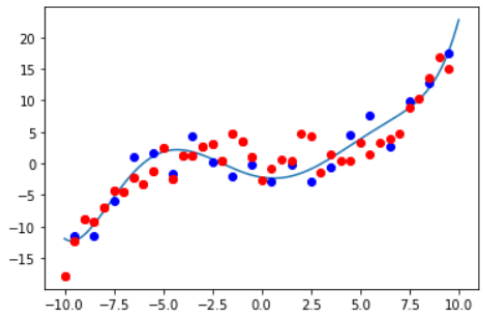
- Result (When Q = 6)

```
Training Err_square for Q= 6 , L= 0.01  :  69.22895281469457
Validation Err_square for Q= 6 , L= 0.01  :  11.959704528259365

Training Err_square for Q= 6 , L= 0.1  :  69.24171721778775
Validation Err_square for Q= 6 , L= 0.1  :  11.978504970916902

Training Err_square for Q= 6 , L= 1  :  70.07790143482315
Validation Err_square for Q= 6 , L= 1  :  12.243661391794266

Training Err_square for Q= 6 , L= 10  :  79.83241070754715
Validation Err_square for Q= 6 , L= 10  :  14.265325821511187

Training Err_square for Q= 6 , L= 100  :  92.34648989108426
Validation Err_square for Q= 6 , L= 100  :  17.217027151067402

Training Err_square for Q= 6 , L= 1000  :  103.68015154680225
Validation Err_square for Q= 6 , L= 1000  :  19.61844546691453

Training Err_square for Q= 6 , L= 1000000  :  128.94806813175865
Validation Err_square for Q= 6 , L= 1000000  :  19.876740966049503
```

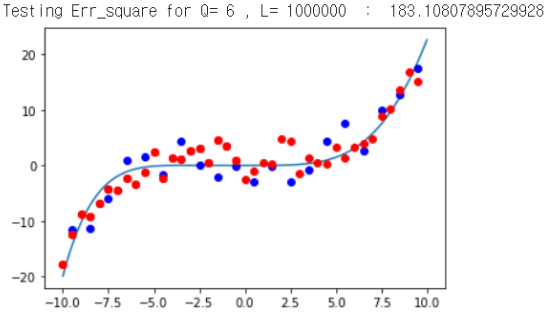
[Training and Validation error result]

➔ Validation error is biggest when L = 1000000, and smallest when L = 0.01

```
Testing Err_square for Q= 6 , L= 0.01  :  302.12862243895813
```




```
Testing Err_square for Q= 6 , L= 1000000  :  183.10807895729928
```



➔ We usually select the weight with smallest validation error, so when L = 0.01 is optimal model.

➔ But as we can see in the upper picture, the testing error is smaller when L = 100000 which has bigger validation error. So this should be considered in other way.

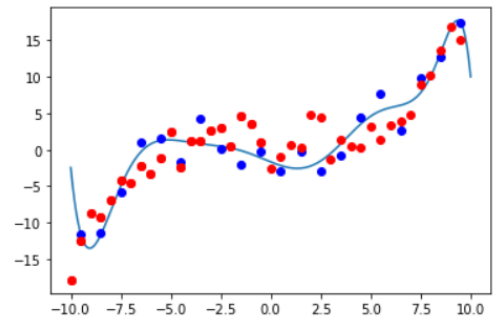
- Result(When Q = 10)

```
Training Err_square for Q= 10 , L= 0.01  :  54.68898161646153
Validation Err_square for Q= 10 , L= 0.01  :  185.27490126735583

Training Err_square for Q= 10 , L= 0.1  :  54.70248060035202
Validation Err_square for Q= 10 , L= 0.1  :  180.22032376859536

Training Err_square for Q= 10 , L= 1  :  55.44453017816907
Validation Err_square for Q= 10 , L= 1  :  141.7169821488414

Training Err_square for Q= 10 , L= 10  :  62.158684554081596
Validation Err_square for Q= 10 , L= 10  :  52.086194851664686

Training Err_square for Q= 10 , L= 100  :  70.45554311089899
Validation Err_square for Q= 10 , L= 100  :  17.349151945126216

Training Err_square for Q= 10 , L= 1000  :  74.23660974421918
Validation Err_square for Q= 10 , L= 1000  :  13.976245196842726

Training Err_square for Q= 10 , L= 1000000  :  84.8909578922808
Validation Err_square for Q= 10 , L= 1000000  :  46.31711999706338
```

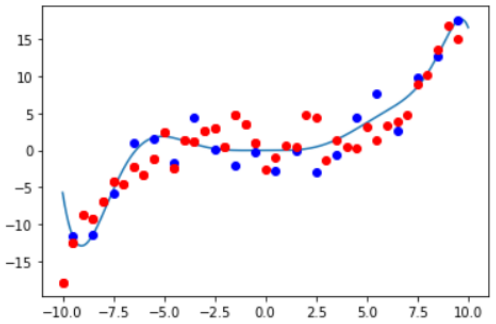
[Training and Validation error result]

➔ Validation error is biggest when L = 0.01, and smallest when L = 1000

```
Testing Err_square for Q= 10 , L= 0.01  :  531.0069535089756
```



```
Testing Err_square for Q= 10 , L= 1000  :  368.85992027134944
```



➔ We can find that when we choose model with small validation error, we can prevent overfitting.

➔ I think the reason why when we get model with Q = 6 is not that efficient in this cross validation is that when we make model with Q = 6, there are not much overfitting, so I think this doesn't make big difference when we set Q = 6. But as you can see, we can know that cross validation can prevent overfitting in Q = 10 model.

- V-fold

```
def slice_array():
  global Y
  start = -1 * num_of_iter
  end = 0
  TrainX = X
  TrainY = Y
  err_sqr_cross = 0
  err_sqr_cross_sum = 0
  err_sqr_training = 0
  err_sqr_training_sum = 0
  err_sqr_cross_min = 10000000000000000000
  err_sqr_cross_index = 0
  min_W = 0
  for i in range(Y):
    # train_arr_X, val_arr_X, train_arr_Y, val_arr_
    start += num_of_iter
    end += num_of_iter
    start = int(start)
    end = int(end)
    val_arr_X = TrainX[start:end]
    val_index_X = np.arange(start, end, 1)
    train_arr_X = np.delete(TrainX, val_index_X)
    val_arr_Y = TrainY[start:end]
    val_index_Y = np.arange(start, end, 1)
    train_arr_Y = np.delete(TrainY, val_index_X)
    z_cross = [train_arr_X**0]
    X_next_cross = train_arr_X
```

[Split Train dataset to Trainset and validation set into len(Y) pieces]

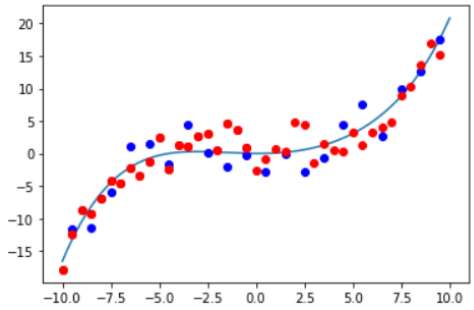
- Result (When Q = 6)

```
Training Err_square for Q= 6 , L= 0.01  :  54.55153485001254
Validation Err_square for Q= 6 , L= 0.01  :  7131.6116899246135

Training Err_square for Q= 6 , L= 0.1  :  54.590319687956914
Validation Err_square for Q= 6 , L= 0.1  :  7314.874160599093

Training Err_square for Q= 6 , L= 1  :  55.93670173722734
Validation Err_square for Q= 6 , L= 1  :  8798.355547209332

Training Err_square for Q= 6 , L= 10  :  64.35558323761245
Validation Err_square for Q= 6 , L= 10  :  13077.252456171544

Training Err_square for Q= 6 , L= 100  :  73.769753524043
Validation Err_square for Q= 6 , L= 100  :  15838.13734297254

Training Err_square for Q= 6 , L= 1000  :  80.28924100937536
Validation Err_square for Q= 6 , L= 1000  :  15891.45366156143

Training Err_square for Q= 6 , L= 1000000  :  98.26275047546656
Validation Err_square for Q= 6 , L= 1000000  :  3043.6252856001224
```

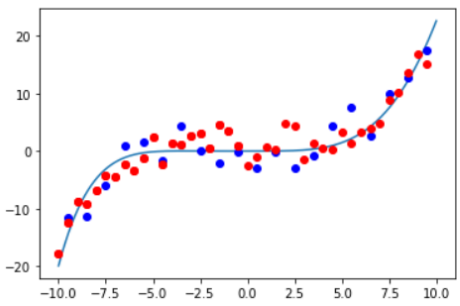
[Training and Validation error result]

➔ Validation error is largest when L = 1000, and smallest when L = 1000000

```
Testing Err_square for Q= 6 , L= 1000  :  160.93981993653867
```




```
Testing Err_square for Q= 6 , L= 1000000  :  183.18363213955303
```



➔ Similar to leave-one-out, there was no significant difference in cross validation using V-fold when Q was 6.

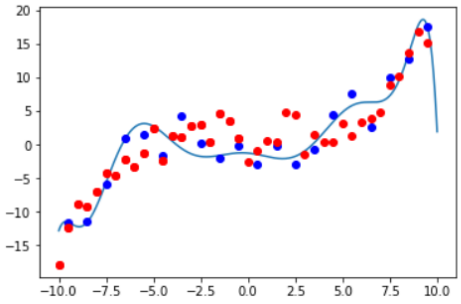
- Result(When Q = 10)

```
Training Err_square for Q= 10 , L= 0.01  :  33.052738665382385
Validation Err_square for Q= 10 , L= 0.01  :  199312531.7023789

Training Err_square for Q= 10 , L= 0.1  :  33.19217824753819
Validation Err_square for Q= 10 , L= 0.1  :  194999144.81961828

Training Err_square for Q= 10 , L= 1  :  34.14569097385878
Validation Err_square for Q= 10 , L= 1  :  166230205.5076119

Training Err_square for Q= 10 , L= 10  :  39.139387736165816
Validation Err_square for Q= 10 , L= 10  :  82925527.08482367

Training Err_square for Q= 10 , L= 100  :  47.508074560365856
Validation Err_square for Q= 10 , L= 100  :  26823270.154681634

Training Err_square for Q= 10 , L= 1000  :  52.78082725398881
Validation Err_square for Q= 10 , L= 1000  :  16833332.747765012

Training Err_square for Q= 10 , L= 1000000  :  64.27358853922786
Validation Err_square for Q= 10 , L= 1000000  :  652025.0975851931
```

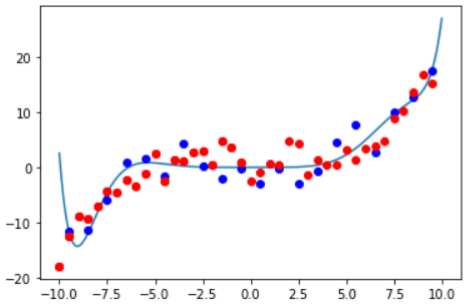
[Training and Validation error result]

➔ Validation error is biggest when L = 0.01, and smallest when L = 1000000

Testing Err_square for Q= 10 , L= 0.01 : 415.06464435695364



Testing Err_square for Q= 10 , L= 1000000 : 651.0580627450136



➔ In v-fold, even when Q was 10, the test error of the case where the validation error was small did not appear small. Rather, both the validation error and the testing error were larger than the leave-one-out result, so the model showed poor performance than using leave-one out, but still not bad error square value.

d) **Discussions and Conclusions. Any design decisions you had to make and your experimental observations. What do you observe about the behavior of your program when you run it? Does it seem to work the way you think it should? Play around a little with different setting to see what happens. Note, your open-ended exploration is highly valued.**

Cross validation is used to make training a little more accurate when there is a risk of overfitting because there is little training data. As a result of the task, it was confirmed that overfitting was reduced when cross validation was applied, unlike the model where perfect overfitting occurred if cross validation was not actually used in Q = 10. Among applying cross validation, leave-one-out has a large computational cost, but is one of the effective training methods to prevent overfitting. V-fold costs less computational cost than leave-one-out, but is less effective in preventing overfitting. However, as a result of this task, there is not much difference in testing error for the model using the two algorithms, so I think it is a good way to save cost by using enough v-fold.

It was confirmed that regularization was also effectively applied.

One question is that when performing v-fold, the validation set is determined by splitting the folds sequentially from the beginning. I wondered if this would cause any bias in the data while training and validating with validation set.