Xiang Chen 1004704992

Instructor's Name : Chris J. Maddison

September 28, 2020

CSC311 Homework One

1.Classification with Nearest Neighbours

(a).

See **hw1_q1_code.py**

(b).

When k is 1, the training accuracy is 1.0, the validation accuracy is 0.6693877551020408.

When k is 2, the training accuracy is 0.8678915135608049, the validation accuracy is 0.673469387755102.

When k is 3, the training accuracy is 0.8171478565179353, the validation accuracy is 0.6551020408163265.

When k is 4, the training accuracy is 0.8040244969378828, the validation accuracy is 0.6918367346938775.

When k is 5, the training accuracy is 0.7825896762904637, the validation accuracy is 0.6530612244897959.

When k is 6, the training accuracy is 0.7769028871391076, the validation accuracy is 0.6632653061224489.

When k is 7, the training accuracy is 0.768591426071741, the validation accuracy is 0.6653061224489796.

When k is 8, the training accuracy is 0.7668416447944006, the validation accuracy is 0.689795918367347.

When k is 9, the training accuracy is 0.7585301837270341, the validation accuracy is 0.6959183673469388.

When k is 10, the training accuracy is 0.7471566054243219, the validation accuracy is 0.6714285714285714.

When k is 11, the training accuracy is 0.752843394575678 , the validation accuracy is 0.7.

When k is 12, the training accuracy is 0.747594050743657, the validation accuracy is 0.6918367346938775.

When k is 13, the training accuracy is 0.7506561679790026, the validation accuracy is 0.6959183673469388.

When k is 14, the training accuracy is 0.7480314960629921, the validation accuracy is 0.6979591836734694.

When k is 15, the training accuracy is 0.7497812773403325, the validation accuracy is 0.7040816326530612.

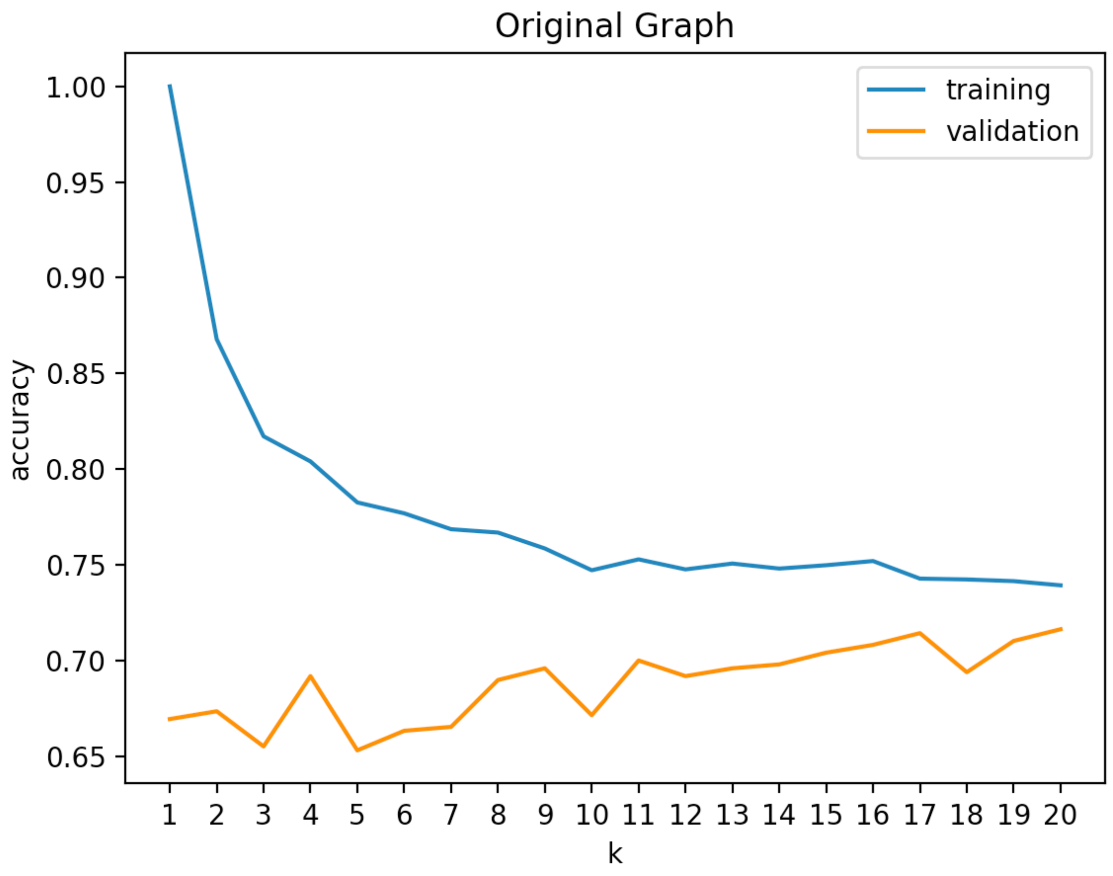
When k is 16, the training accuracy is 0.7519685039370079, the validation accuracy is 0.7081632653061225.

When k is 17, the training accuracy is 0.7427821522309711, the validation accuracy is 0.7142857142857143.

When k is 18, the training accuracy is 0.742344706911636, the validation accuracy is 0.6938775510204082.

When k is 19, the training accuracy is 0.7414698162729659, the validation accuracy is 0.710204081632653.

When k is 20, the training accuracy is 0.7392825896762905, the validation accuracy is 0.7163265306122449.



When k=20, it has the best performance in validation set. For the testing set, it is 0.6612244897959184.

(c).

Based on the documents on the source from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html, we know that cosine similarity is computed as

$$\frac{x \cdot y}{\sqrt{x \cdot x}\sqrt{y \cdot y}}$$

Based on the dataset provided in hint, [‘cat’,‘bulldozer’,’‘cat cat cat’], and after vectorization, we can get features of ['bulldozer','cat'], and [[0,1],[1,0],[0,3]]. If we use Euclidean distance measurement, we can see [0,1] is more closer with [1,0] instead of [0,3], which is obviously wrong since [cat] should be closer with [cat cat cat]. Therefore, the magnitude in text (word counts) does not matter too much. We should consider the direction of the vector which we can use cosine similarity to compute the angle. In the extreme case, when the angle between tow vector is zero, the cosine similarity is 1(distance is 0), which is perfect matching like two vectors [1,0] and [100,0](e.g. [cat] and [cat … cat]). Therefore, cosine metric can have better performance than Euclidean metric.

2. Regularized Linear Regression

(a).

Based on the course notes, we can get the following equation

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}^{\beta}_{reg}}{\partial w_j} \quad (1)$$

Then we need to solve $\alpha \frac{\partial \mathcal{J}^{\beta}_{reg}}{\partial w_j}$

Since we already know

$$\mathcal{J}^{\beta}_{\text{reg}}(\mathbf{w}) = \underbrace{\frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2}_{=\mathcal{J}} + \underbrace{\frac{1}{2}\sum_{j=1}^{D}\beta_j w_j^2}_{=\mathcal{R}},$$

We expand it and do the partial differential to w_j

$$\alpha \frac{\partial \mathcal{J}^{\beta}_{reg}}{\partial w_j} = \frac{\alpha}{N}\sum_{i=1}^{N} x_j^{(i)}(\sum_{j'=1}^{D} w_{j'}x_{j'}^{(i)} + b - t^{(i)}) + \alpha\beta_j w_j = \alpha(\frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)}) + \beta_j w_j) \quad (2)$$

We plugin (2) into (1). Then, we can get

$$w_j \leftarrow w_j - \alpha(\frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)}) + \beta_j w_j) = (1 - \alpha\beta_j)w_j - \frac{\alpha}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)})$$

Based on the course notes, we can get the following equation

$$b \leftarrow b - \alpha \frac{\partial \mathcal{J}^{\beta}_{reg}}{\partial b} \quad (3)$$

We can apply chain rule to the last part

$$\frac{\partial \mathcal{J}^{\beta}_{reg}}{\partial b} = \frac{\partial J}{\partial b} + \frac{\partial R}{b} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial b} + \frac{\partial R}{\partial b} \quad (4)$$

We will do $\frac{\partial J}{\partial y}\frac{\partial y}{\partial b}$ first,

$$\frac{\partial J}{\partial y}\frac{\partial y}{\partial b} = \frac{\partial}{\partial y}(\frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})^2) * \frac{\partial y}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)}) \quad (5)$$

$$\frac{\partial R}{\partial b} = \frac{\partial}{\partial b}(\frac{1}{2}\sum_{j=1}^{D}\beta_j w_j^2) = 0 \ (6)$$

We plugin (4), (5) and (6) into (3), then

$$b \leftarrow b - \alpha\frac{\partial \mathcal{J}_{reg}^{\beta}}{\partial b} = b - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)}) \ (7)$$

Therefore, the final answer for the part a is

$$w_j \leftarrow (1-\alpha\beta_j)w_j - \frac{\alpha}{N}\sum_{i=1}^{N}x_j^{(i)}(y^{(i)} - t^{(i)})$$

$$b \leftarrow b - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})$$

Based on the update rule we get above, we can compare it with the rule without regularization. We can find that we have $(1-\alpha\beta)$ before and $(1-\alpha\beta)$ can be less than one, which is a decay effect in weight (make the weight smaller). Therefore, it should be called weight decay.

(b).

Based on part a, we can get the equation that

$$\frac{\partial \mathcal{J}_{reg}^{\beta}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}(\sum_{j'}w_{j'}x_{j'}^{(i)} + b - t^{(i)}) + \beta_j w_j = \frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}(y^{(i)} - t^{(i)}) + \beta_j w_j$$

We can further expand it.

$$\frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}(y^{(i)} - t^{(i)}) + \beta_j w_j = \frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}(\sum_{j'=1}^{D}w_{j'}x_{j'}^{(i)} + b - t^{(i)}) + \beta_j w_j$$

Since in this part we drop the bias term, then we can get

$$\frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}(\sum_{j'=1}^{D}w_{j'}x_{j'}^{(i)} - t^{(i)}) + \beta_j w_j \ (8)$$

We can expand (8)

$$\frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}(\sum_{j'=1}^{D}w_{j'}x_{j'}^{(i)} - t^{(i)}) + \beta_j w_j = \frac{1}{N}(\sum_{i=1}^{N}\sum_{j'=1}^{D}x_j^{(i)}w_{j'}x_{j'}^{(i)}) - \frac{1}{N}(\sum_{i=1}^{N}x_j^{(i)}t^{(i)}) + \beta_j w_j \ (9)$$

We can reorganize sigmas in (9) and get

$$\frac{1}{N}(\sum_{j'=1}^{D}\sum_{i=1}^{N}x_j^{(i)}w_{j'}x_{j'}^{(i)}) - \frac{1}{N}(\sum_{i=1}^{N}x_j^{(i)}t^{(i)}) + \beta_j w_j = \sum_{j'=1}^{D}w_{j'}(\frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}x_{j'}^{(i)}) + \beta_j w_j - \frac{1}{N}(\sum_{i=1}^{N}x_j^{(i)}t^{(i)}) \ (10)$$

Therefore, we can get the final answers based on (10)

$$A_{jj'} = \left\{ \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} + \beta_j \mid j = j' \right\} \cup \left\{ \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \mid j \neq j' \right\}$$

$$c_j = \frac{1}{N}(\sum_{i=1}^{N} x_j^{(i)} t^{(i)})$$

(c).

Let's define $diag(\beta_1, \ldots, \beta_D)$ is a diagonal matrix with $\beta_1, \ldots, \beta_D$ as the numerical value on the diagonal. (11)

Based on (b), we have got the formulas for $A_{jj'}$ and c_j. With what I have defined in (11), we can our formulas for **A** and **c**.

$$A = \frac{1}{N}X^T X + diag(\beta_1, \ldots, \beta_D) \text{ (12)}$$

$$c = \frac{1}{N}X^T t \text{ (13)}$$

We have $Aw - c = 0$, and put (12) and (13) into the equation.

$$(\frac{1}{N}X^T X + diag(\beta_1, \ldots, \beta_D)) * w - \frac{1}{N}X^T t = 0 \text{ (14)}$$

After we simplify (14), we will get

$$w = (X^T X + N diag(\beta_1, \ldots, \beta_D))^{-1} X^T t$$

which is our final answer.

3. Loss Functions

Based on the question description, we have $y = w^T x + b$ and X is a design matrix which is originated with one row per training example. Therefore, we can easily get $y = Xw + b$.

From the question description, we know $\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y^{(i)}, t^{(i)})$

$$\frac{\partial \mathcal{J}}{\partial y_i} = \frac{\partial}{\partial y_i}(\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y^{(i)}, t^{(i)})) = \frac{1}{N}\frac{\partial}{\partial y_i}(\mathcal{L}(y^{(i)}, t^{(i)})) \ (15)$$

We know $\mathcal{L}(y, t) = 1 - cos(y - t)$. Therefore, we can calculate (15) and get

$$\frac{1}{N}sin(y^{(i)} - t^{(i)})$$

Therefore, we can get $\frac{\partial \mathcal{J}}{\partial y} = \frac{1}{N}sin(\mathrm{y} - \mathrm{t})$ (16).

From the question description, we know

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

Therefore, we can apply chain rule and then get

$$\frac{\partial \mathcal{J}}{\partial w} = (\frac{\partial \mathcal{J}}{\partial y})^T \frac{\partial y}{\partial w}$$

By plugin (16), we can get

$$\frac{\partial \mathcal{J}}{\partial w} = (\frac{1}{N}sin(y - t))^T X$$

We can apply chain rule to $\frac{\partial \mathcal{J}}{\partial b}$ and then get

$$\frac{\partial \mathcal{J}}{\partial b} = (\frac{\partial \mathcal{J}}{\partial y})^T \frac{\partial y}{\partial b} = (\frac{1}{N}sin(y - t))^T \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Therefore, the final results are

$$y = Xw + b$$

$$\frac{\partial \mathcal{J}}{\partial y} = \frac{1}{N}sin(y - t)$$

$$\frac{\partial \mathcal{J}}{\partial w} = (\frac{1}{N} sin(y - t))^T X$$

$$\frac{\partial \mathcal{J}}{\partial b} = (\frac{1}{N} sin(y - t))^T \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

4. Cross Validation

(a) & (b).

See **hw1_q4_code.py.**

(c)

The error analysis for training & testing is:

When lambd is 5e-05 , training error is 0.024868245773721747 , testing error is 2.553479891795417

When lambd is 0.0001510204081632653 , training error is 0.06137693545770471 , testing error is 1.697008161887628

When lambd is 0.00025204081632653066 , training error is 0.09189949933629438 , testing error is 1.425641340029266

When lambd is 0.000353061224489796 , training error is 0.11933116280251348 , testing error is 1.2926871564164

When lambd is 0.0004540816326530613 , training error is 0.14484130634919565 , testing error is 1.2165510007564098

When lambd is 0.0005551020408163266 , training error is 0.16901265605197027 , testing error is 1.1695172228820245

When lambd is 0.000656122448979592 , training error is 0.19216904055777656 , testing error is 1.139396486362229

When lambd is 0.0007571428571428573 , training error is 0.21450555118836645 , testing error is 1.1199475211008287

When lambd is 0.0008581632653061226 , training error is 0.23614765421037193 , testing error is 1.1076307890971744

When lambd is 0.0009591836734693879 , training error is 0.2571806283747777 , testing error is 1.1002851517053027

When lambd is 0.001060204081632653 , training error is 0.2776653041102488 , testing error is 1.0965152691373765

When lambd is 0.0011612244897959184 , training error is 0.2976469736809816 , testing error is 1.0953800251692092

When lambd is 0.0012622448979591838 , training error is 0.3171606784698311 , testing error is 1.0962220699056795

When lambd is 0.001363265306122449 , training error is 0.33623447266715434 , testing error is 1.0985689651588966

When lambd is 0.0014642857142857144 , training error is 0.35489150654474494 , testing error is 1.1020730492698234

When lambd is 0.0015653061224489796 , training error is 0.373151395233175 , testing error is 1.10647336737211

When lambd is 0.001666326530612245 , training error is 0.39103114104238607 , testing error is 1.1115707422434107

When lambd is 0.0017673469387755104 , training error is 0.4085457690032087 , testing error is 1.1172109697194246

When lambd is 0.0018683673469387756 , training error is 0.42570877369469184 , testing error is 1.12327320240217

When lambd is 0.001969387755102041 , training error is 0.4425324392154308 , testing error is 1.129661741216066

When lambd is 0.0020704081632653064 , training error is 0.4590280722511183 , testing error is 1.1363001214551582

When lambd is 0.002171428571428572 , training error is 0.47520617458927134 , testing error is 1.14312677794992

When lambd is 0.002272448979591837 , training error is 0.4910765727898202 , testing error is 1.150091818477592

When lambd is 0.0023734693877551023 , training error is 0.5066485171140065 , testing error is 1.157154588712396

When lambd is 0.002474489795918368 , training error is 0.5219307581089103 , testing error is 1.1642818115297286

When lambd is 0.002575510204081633 , training error is 0.5369316067546627 , testing error is 1.1714461490862453

When lambd is 0.0026765306122448983 , training error is 0.5516589823819626 , testing error is 1.1786250801849985

When lambd is 0.0027775510204081635 , training error is 0.5661204513915442 , testing error is 1.1858000155809332

When lambd is 0.002878571428571429 , training error is 0.5803232589831643 , testing error is 1.1929555948262478

When lambd is 0.0029795918367346943 , training error is 0.594274355517434 , testing error is 1.2000791230204302

When lambd is 0.0030806122448979595 , training error is 0.6079804187152036 , testing error is 1.2071601163801449

When lambd is 0.003181632653061225 , training error is 0.6214478725962772 , testing error is 1.2141899331770818

When lambd is 0.0032826530612244903 , training error is 0.6346829038380558 , testing error is 1.2211614721781823

When lambd is 0.0033836734693877555 , training error is 0.6476914760718074 , testing error is 1.2280689248550711

When lambd is 0.003484693877551021 , training error is 0.6604793425133655 , testing error is 1.2349075707171393

When lambd is 0.0035857142857142863 , training error is 0.6730520572346608 , testing error is 1.2416736074511963

When lambd is 0.0036867346938775514 , training error is 0.6854149853144172 , testing error is 1.2483640093220445

When lambd is 0.003787755102040817 , training error is 0.6975733120547548 , testing error is 1.2549764086469561

When lambd is 0.0038887755102040822 , training error is 0.7095320514110905 , testing error is 1.261508996207127

When lambd is 0.003989795918367347 , training error is 0.7212960537525347 , testing error is 1.267960437276635

When lambd is 0.004090816326530612 , training error is 0.7328700130466711 , testing error is 1.274329800590167

When lambd is 0.004191836734693878 , training error is 0.7442584735445273 , testing error is 1.280616498076194

When lambd is 0.004292857142857143 , training error is 0.755465836027404 , testing error is 1.286820233583379

When lambd is 0.004393877551020408 , training error is 0.7664963636661716 , testing error is 1.2929409591481675

When lambd is 0.004494897959183674 , training error is 0.7773541875348714 , testing error is 1.2989788376084361

When lambd is 0.004595918367346939 , training error is 0.7880433118135346 , testing error is 1.304934210575325

When lambd is 0.004696938775510204 , training error is 0.798567618709579 , testing error is 1.3108075709433944

When lambd is 0.00479795918367347 , training error is 0.8089308731227057 , testing error is 1.3165995392560388

```
When lambd is 0.004898979591836735 , training error is 0.819136727074616 , testing error is 1.3223108433550168
When lambd is 0.005 , training error is 0.8291887239219635 , testing error is 1.3279423008348683
The error analysis for 5_folds is:
When lambd is 5e-05 , 5_folds error 2.034419129566172
When lambd is 0.0001510204081632653 , 5_folds error 1.7158718422493606
When lambd is 0.00025204081632653066 , 5_folds error 1.5942889369197708
When lambd is 0.000353061224489796 , 5_folds error 1.5326883765056514
When lambd is 0.0004540816326530613 , 5_folds error 1.4983259698135734
When lambd is 0.0005551020408163266 , 5_folds error 1.4786949145134665
When lambd is 0.000656122448979592 , 5_folds error 1.46786337689146
When lambd is 0.0007571428571428573 , 5_folds error 1.462632764986267
When lambd is 0.0008581632653061226 , 5_folds error 1.461124165558918
When lambd is 0.0009591836734693879 , 5_folds error 1.462164127105551
When lambd is 0.001060204081632653 , 5_folds error 1.4649853320122292
When lambd is 0.0011612244897959184 , 5_folds error 1.4690678624287172
When lambd is 0.0012622448979591838 , 5_folds error 1.4740493645999122
When lambd is 0.001363265306122449 , 5_folds error 1.479671507281292
When lambd is 0.0014642857142857144 , 5_folds error 1.4857466977679072
When lambd is 0.0015653061224489796 , 5_folds error 1.4921366539664247
When lambd is 0.001666326530612245 , 5_folds error 1.4987381958210702
When lambd is 0.0017673469387755104 , 5_folds error 1.5054735829699768
When lambd is 0.0018683673469387756 , 5_folds error 1.5122837991377218
When lambd is 0.001969387755102041 , 5_folds error 1.5191237949948841
When lambd is 0.0020704081632653064 , 5_folds error 1.5259590615632304
When lambd is 0.002171428571428572 , 5_folds error 1.5327631252759921
When lambd is 0.002272448979591837 , 5_folds error 1.5395156925700788
When lambd is 0.0023734693877551023 , 5_folds error 1.5462012593550807
When lambd is 0.002474489795918368 , 5_folds error 1.5528080578528523
When lambd is 0.002575510204081633 , 5_folds error 1.559327251366604
When lambd is 0.0026765306122448983 , 5_folds error 1.5657523133378592
When lambd is 0.0027775510204081635 , 5_folds error 1.572078544814552
When lambd is 0.002878571428571429 , 5_folds error 1.5783026968641771
When lambd is 0.0029795918367346943 , 5_folds error 1.5844226732514222
When lambd is 0.0030806122448979595 , 5_folds error 1.5904372949952736
When lambd is 0.003181632653061225 , 5_folds error 1.5963461129827379
When lambd is 0.0032826530612244903 , 5_folds error 1.6021492581569547
When lambd is 0.0033836734693877555 , 5_folds error 1.6078473212672684
When lambd is 0.003484693877551021 , 5_folds error 1.6134412560113056
When lambd is 0.0035857142857142863 , 5_folds error 1.6189323007850878
When lambd is 0.0036867346938775514 , 5_folds error 1.6243219153080137
When lambd is 0.003787755102040817 , 5_folds error 1.6296117291920305
When lambd is 0.0038887755102040822 , 5_folds error 1.6348035001414136
When lambd is 0.003989795918367347 , 5_folds error 1.6398990799471431
When lambd is 0.004090816326530612 , 5_folds error 1.6449003868117174
When lambd is 0.004191836734693878 , 5_folds error 1.6498093828314182
When lambd is 0.004292857142857143 , 5_folds error 1.6546280556922774
When lambd is 0.004393877551020408 , 5_folds error 1.659358403817389
When lambd is 0.004494897959183674 , 5_folds error 1.6640024243474016
When lambd is 0.004595918367346939 , 5_folds error 1.6685621034512308
When lambd is 0.004696938775510204 , 5_folds error 1.6730394085563698
When lambd is 0.00479795918367347 , 5_folds error 1.6774362821625997
When lambd is 0.004898979591836735 , 5_folds error 1.681754636963019
When lambd is 0.005 , 5_folds error 1.685996352045105
The error analysis for 10_folds is:
When lambd is 5e-05 , 10_folds error 1.9239292575036178
When lambd is 0.0001510204081632653 , 10_folds error 1.5615284495031705
When lambd is 0.00025204081632653066 , 10_folds error 1.4439918340971738
When lambd is 0.000353061224489796 , 10_folds error 1.3872953688999878
When lambd is 0.0004540816326530613 , 10_folds error 1.3569617948430923
When lambd is 0.0005551020408163266 , 10_folds error 1.3406766387332607
When lambd is 0.000656122448979592 , 10_folds error 1.3327334282670855
When lambd is 0.0007571428571428573 , 10_folds error 1.3300796677027655
When lambd is 0.0008581632653061226 , 10_folds error 1.3309263395652142
When lambd is 0.0009591836734693879 , 10_folds error 1.3341562876642188
When lambd is 0.001060204081632653 , 10_folds error 1.3390386236640859
When lambd is 0.0011612244897959184 , 10_folds error 1.3450778488745987
When lambd is 0.0012622448979591838 , 10_folds error 1.3519285467134965
When lambd is 0.001363265306122449 , 10_folds error 1.359344518360419
When lambd is 0.0014642857142857144 , 10_folds error 1.3671471290807893
When lambd is 0.0015653061224489796 , 10_folds error 1.3752049022383501
When lambd is 0.001666326530612245 , 10_folds error 1.3834199687421078
When lambd is 0.0017673469387755104 , 10_folds error 1.3917188388030495
When lambd is 0.0018683673469387756 , 10_folds error 1.4000459789949002
When lambd is 0.001969387755102041 , 10_folds error 1.4083592563008995
When lambd is 0.0020704081632653064 , 10_folds error 1.4166266522559847
When lambd is 0.002171428571428572 , 10_folds error 1.424823858027696
When lambd is 0.002272448979591837 , 10_folds error 1.4329324911319536
When lambd is 0.0023734693877551023 , 10_folds error 1.4409387576173236
When lambd is 0.002474489795918368 , 10_folds error 1.4488324379305013
When lambd is 0.002575510204081633 , 10_folds error 1.4566061109350446
When lambd is 0.0026765306122448983 , 10_folds error 1.4642545551567019
When lambd is 0.0027775510204081635 , 10_folds error 1.4717742832861986
When lambd is 0.002878571428571429 , 10_folds error 1.4791631778288987
When lambd is 0.0029795918367346943 , 10_folds error 1.4864202041939905
When lambd is 0.0030806122448979595 , 10_folds error 1.4935451835434947
When lambd is 0.003181632653061225 , 10_folds error 1.5005386120938025
When lambd is 0.0032826530612244903 , 10_folds error 1.5074015167674042
When lambd is 0.0033836734693877555 , 10_folds error 1.5141353394643646
When lambd is 0.003484693877551021 , 10_folds error 1.5207418439943727
When lambd is 0.0035857142857142863 , 10_folds error 1.5272230410440097
When lambd is 0.0036867346938775514 , 10_folds error 1.5335811275661793
```

When lambd is 0.003787755102040817 , 10_folds error 1.5398184377525443

When lambd is 0.0038887755102040822 , 10_folds error 1.545937403345572

When lambd is 0.003989795918367347 , 10_folds error 1.551940521508311

When lambd is 0.004090816326530612 , 10_folds error 1.5578303288297721

When lambd is 0.004191836734693878 , 10_folds error 1.5636093803257605

When lambd is 0.004292857142857143 , 10_folds error 1.56928023251725

When lambd is 0.004393877551020408 , 10_folds error 1.574845429844395

When lambd is 0.004494897959183674 , 10_folds error 1.5803074938143544

When lambd is 0.004595918367346939 , 10_folds error 1.5856689143930822

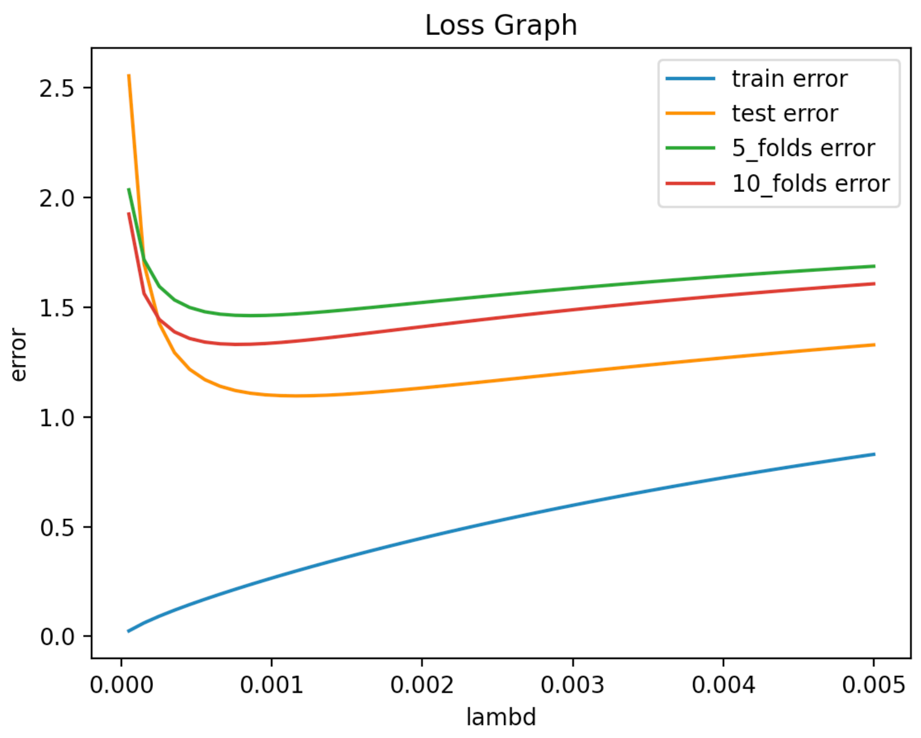
When lambd is 0.004696938775510204 , 10_folds error 1.5909321432411332

When lambd is 0.00479795918367347 , 10_folds error 1.59609958846598

When lambd is 0.004898979591836735 , 10_folds error 1.6011736106219572

When lambd is 0.005 , 10_folds error 1.606156519736528

(d).



Based on the Loss Graph, we can see that the curves of 5_folds error, 10-_folds_error, and test error all go down first and then go up, which means that if we have lambd with too smaller value or too large value, we will get a large validation error. The curves of 5_folds error, 10-_folds_error, and test error have the trend to converge to some point and have really similar shapes. Therefore, we know the cross validation actually perform well when we do not have enough data.