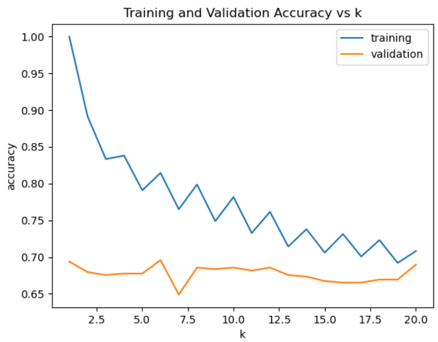
Q1 i

(a) In hw1_q1_code.py

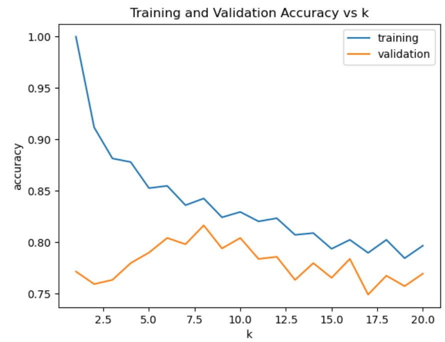
(b) I generate two plots . one with metric = 'cosine' . one not

Plot without metric = 'cosine'




```
Best validation accuracy:0.6959183673469388
Best accuracy on the test data: 0.6428571428571429
```

Plot with metric = 'cosine'




```
Best validation accuracy:0.8163265306122449
Best accuracy on the test data: 0.7591836734693878
```

(c) metric = 'cosine' calculates the distance using the following formula

$$\vec{x}\cdot\vec{y} = |\vec{x}|\cdot|\vec{y}|\cdot\cos\langle\vec{x},\vec{y}\rangle \Rightarrow \cos\langle\vec{x},\vec{y}\rangle = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}||\vec{y}|}$$

Since the value of $\cos\langle\vec{x},\vec{y}\rangle$ can let us know the angle between those two vectors , it is easier to know whether those two data sets are related with metric = 'cosine' than the Euclidean metric

Q3:

(a) Similar to what we learned from class

$$w_j \leftarrow w_j - \frac{\partial J^\beta}{w_j}$$

Now calculate $\frac{\partial J^\beta}{\partial w_j} = \frac{\partial (J+R)}{\partial w_j} = \frac{\partial J}{\partial w_j} + \frac{\partial}{\partial w_j}\left(\frac{1}{2}\sum_{j=1}^{D} \beta_j w_j^2\right)$

$$= \underbrace{\frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)})}_{\text{from lecture}} + \beta_j w_j$$

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

Since $y = \sum_{j=1}^{D} w_j x_j^{(i)} + b \Rightarrow b = y - \sum_{j=1}^{D} w_j x_j^{(i)}$ (2)

plug (1) into (2), we got

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

Substitute y with $b + \sum_{j=1}^{D} w_j x_j^{(i)}$

We got

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

So we have $b \leftarrow b + \sum_{j=1}^{D} \alpha\left(\frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)})x_j^{(i)} + \beta_j w_j x_j^{(i)}\right)$

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

(b): By part (a)

We know $\frac{\partial J_{reg}^\beta}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)}) + \beta_j w_j$

substitute with $y = \sum_{j=1}^{D} w_j x_j^{(i)}$

So $\frac{\partial J_{reg}^{\beta}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} \left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} - t^{(i)} \right) + \beta_j w_j$

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

$= \frac{1}{N}\sum_{i=1}^{N} \left(x_j^{(i)} \sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} + N \cdot \beta_j w_j \right) - \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} t^{(i)} = A_{jj'} w_{j'} - c_j$.

define an indicator function $I_{j'} = \begin{cases} 1, & j = j' \\ 0, & \text{otherwise} \end{cases}$

So $\frac{\partial J_{reg}^{\beta}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} \left(x_j^{(i)} \sum_{j'=1}^{D} x_{j'}^{(i)} w_{j'} + N \cdot \sum_{j'=1}^{D} \beta_{j'} I_{j'} w_{j'} \right) - \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} t^{(i)}$

Hence, $A_{jj'} = \frac{1}{N}\sum_{i=1}^{N} \left(x_j^{(i)} \cdot x_{j'}^{(i)} + N \cdot \beta_{j'} I_{j'} \right)$

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

(c):

let $\beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_D \end{pmatrix}$, β_i are vectors, $\beta_i = \beta_{j'}$ if $i = j'$

$A = (X^T X + N\beta) \cdot \frac{1}{N}$

$C = (X^T \vec{t}) \cdot \frac{1}{N}$

$A \cdot W - C = 0 \Rightarrow W = A^{-1} \cdot C = (X^T X + N\beta)^{-1} \cdot X^T \vec{t}$

Q3: $\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y^{(i)}, t^{(i)}) = \frac{1}{N}\sum_{i=1}^{N} 1-\cos(y^{(i)}-t^{(i)})$

So

$$y = w^T x + b = \sum_{j=1}^{D} w_j x_j^{(i)} + b$$

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

$$\frac{\partial \mathcal{J}}{\partial w_j} = \frac{\partial \mathcal{J}}{\partial y} \cdot \frac{\partial y}{\partial w_j} = \left(\frac{1}{N}\sum_{i=1}^{N} \sin(y^{(i)} - t^{(i)})\right) \cdot x_j^{(i)}$$

So $\frac{\partial \mathcal{J}}{\partial W} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix} = \begin{pmatrix} \frac{1}{N}\sum_{i=1}^{N} (\sin(y^{(i)} - t^{(i)}) x_j^{(i)}) \\ \vdots \\ \frac{1}{N}\sum_{i=1}^{N} (\sin(y^{(i)} - t^{(i)}) x_D^{(i)}) \end{pmatrix}$

$$\frac{\partial \mathcal{J}}{\partial b} = \frac{\partial \mathcal{J}}{\partial y} \cdot \frac{\partial y}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} \sin(y^{(i)} - t^{(i)}) \cdot 1 = \frac{1}{N}\sum_{i=1}^{N} \sin(y^{(i)} - t^{(i)})$$

Q4:
(a) code both shown in hw1_q4_code.py

(b): Code shown in hw1_q4_code.py

The order should be shuffle_data
$\downarrow$
split_data → train_model → predict → loss → cross_validation.

The argumenes are: data,
number of folds,
a sequence of lambda

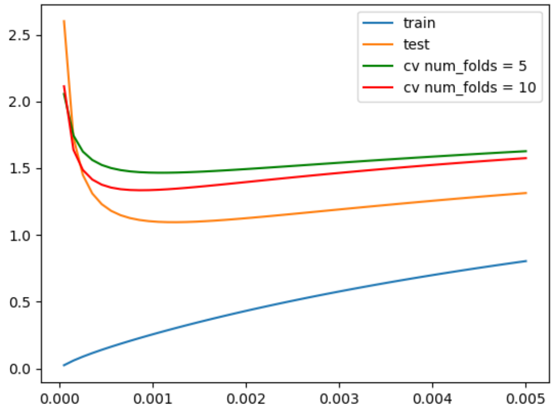
(c): The output is shown below:

```
lambda: 5e-05
     train error: 0.023827398142615853
     test error: 2.601391644823133
lambda: 0.0001510204081632653
     train error: 0.05891274573913389
     test error: 1.7287677994041546
lambda: 0.00025204081632653066
     train error: 0.08829510341160723
     test error: 1.449041107721426
lambda: 0.000353061224489796
     train error: 0.11469870650014892
     test error: 1.3106086900637661
lambda: 0.0004540816326530613
     train error: 0.13923988850774655
     test error: 1.2304985961640553
lambda: 0.0005551020408163266
     train error: 0.16248341903497207
     test error: 1.180389255472875
lambda: 0.000656122448979592
     train error: 0.18474660580356297
     test error: 1.1477791855291435
lambda: 0.0007571428571428573
     train error: 0.20622199234382155
     test error: 1.1262495170677962
lambda: 0.0008581632653061226
     train error: 0.22703389988077502
     test error: 1.1121518828937422
lambda: 0.0009591836734693879
     train error: 0.2472669246464869
     test error: 1.1032544106716156
lambda: 0.001060204081632653
     train error: 0.26698132507945677
     test error: 1.098113519790282
lambda: 0.0011612244897959184
     train error: 0.28622180470505515
     test error: 1.0957538481482645
lambda: 0.0012622448979591838
     train error: 0.3050227607306162
     test error: 1.095492899034103
lambda: 0.001363265306122449
     train error: 0.3234115438629369
     test error: 1.0968392292395663
lambda: 0.0014642857142857144
     train error: 0.34141055052213537
     test error: 1.09943045388258
lambda: 0.0015653061224489796
     train error: 0.35903860424252554
     test error: 1.1029939651442386
lambda: 0.001666326530612245
     train error: 0.3763118905865588
     test error: 1.1073211894755193
lambda: 0.0017673469387755104
     train error: 0.3932446038612771
     test error: 1.1122502216496666
lambda: 0.0018683673469387756
     train error: 0.4098494033148334
     test error: 1.1176538117834807
```

```
lambda: 0.001969387755102041
     train error: 0.4261377407022199
     test error: 1.1234308705075875
lambda: 0.0020704081632653064
     train error: 0.44212009936214197
     test error: 1.129500344237022
lambda: 0.002171428571428572
     train error: 0.4578061713896179
     test error: 1.1357967224805834
lambda: 0.002272448979591837
     train error: 0.4732049908414316
     test error: 1.1422666911374024
lambda: 0.0023734693877551023
     train error: 0.488325035281242
     test error: 1.1488666047195637
lambda: 0.002474489795918368
     train error: 0.5031743042367003
     test error: 1.155560553114307
lambda: 0.002575510204081633
     train error: 0.5177603806219526
     test error: 1.1623188662165589
lambda: 0.0026765306122448983
     train error: 0.5320904794538496
     test error: 1.1691169452851025
lambda: 0.0027775510204081635
     train error: 0.5461714869921581
     test error: 1.1759343410147318
lambda: 0.002878571428571429
     train error: 0.5600099925916275
     test error: 1.1827540199582744
lambda: 0.0029795918367346943
     train error: 0.5736123149542308
     test error: 1.1895617761946022
lambda: 0.0030806122448979595
     train error: 0.5869845240389066
     test error: 1.196345756047619
lambda: 0.003181632653061225
     train error: 0.6001324595729604
     test error: 1.2030960715562675
lambda: 0.0032826530612244903
     train error: 0.6130617468798986
     test error: 1.2098044841755544
lambda: 0.0033836734693877555
     train error: 0.6257778105688381
     test error: 1.2164641444656075
lambda: 0.003484693877551021
     train error: 0.6382858865043091
     test error: 1.2230693767226455
lambda: 0.0035857142857142863
     train error: 0.6505910323804636
     test error: 1.229615499917253
lambda: 0.0036867346938775514
     train error: 0.6626981371520451
     test error: 1.2360986781407932
lambda: 0.003787755102040817
     train error: 0.6746119295199954
     test error: 1.2425157951688124
```

```
lambda: 0.0038887755102040822
     train error: 0.6863369856278856
     test error: 1.2488643488391782
lambda: 0.003989795918367347
     train error: 0.6978777360932145
     test error: 1.2551423617905224
lambda: 0.004090816326530612
     train error: 0.7092384724728255
     test error: 1.26134830577148
lambda: 0.004191836734693878
     train error: 0.7204233532423231
     test error: 1.2674810372558627
lambda: 0.004292857142857143
     train error: 0.731436409354227
     test error: 1.2735397425155066
lambda: 0.004393877551020408
     train error: 0.7422815494277314
     test error: 1.2795238906352722
lambda: 0.004494897959183674
     train error: 0.7529625646135095
     test error: 1.2854331932216374
lambda: 0.004595918367346939
     train error: 0.7634831331695463
     test error: 1.2912675697720937
lambda: 0.004696938775510204
     train error: 0.7738468247780177
     test error: 1.297027117847228
lambda: 0.00479795918367347
     train error: 0.7840571046284165
     test error: 1.302712087329965
lambda: 0.004898979591836735
     train error: 0.7941173372883514
     test error: 1.3083228581730006
lambda: 0.005
     train error: 0.8040307903801956
     test error: 1.3138599211313071
5-fold cross validation lambda: 0.001060204081632653
10-fold cross validation lambda: 0.0008581632653061226
```

(d): The plot is shown below.



The λ proposed is :

```
5-fold cross validation lambda: 0.0010602040816326530
10-fold cross validation lambda: 0.0008581632653061226
```

Comments: As we can see from the plot.

The trend of test, cv num_folds = 5 and cv num_folds = 10 are close to each other, they all decrease first then increase.

While for the train error line, the curve always increase.