CSC311 HW1

Jiakai Shi

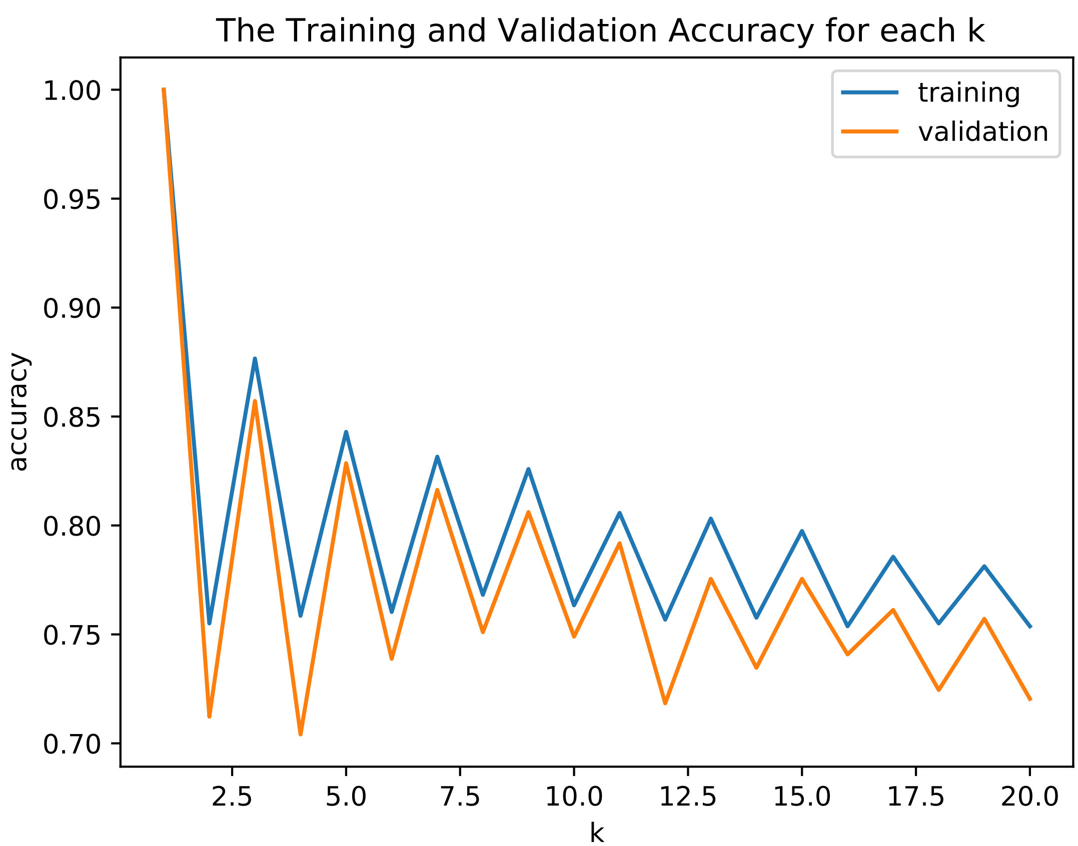
Student number: 1003986760

1 Classification with Nearest Neighbours

1. a) Code is provided in the file called "hw1_q1_code.py".

2. b) Code is provided in the file called "hw1_q1_code.py".

   Here is the plot showing the training and validation accuracy for each k.



   As we can see in the plot, the model with the best validation accuracy is $k = 1$, when we have $accuracy = 1.0$.

3. c) When we are passing argument $metric =' cosine'$ to the $KNeighborsClassifier$, we are calculating the cosine distance instead of Euclidean distance.

Here is the equation for cosine between vector **a** and vector **b**:

$$cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}$$

Comparing to using Euclidean metric, there are some special cases in cosine results, they are: -1, 0, 1.

If the cosine value is 1, the vectors are in the same direction, which means there are similarities between data points.

If the cosine value is 0, we have orthogonal vectors, in which data points are unrelated but have some similarities.

If the cosine value if -1, then vectors are pointing in opposite directions, which means there are no similarity between data points.

Since the distance is greater than or equal to zero, the cosine distance is:

$$cos_D(\mathbf{a}, \mathbf{b}) = 1 - \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}$$

Considering the special cases of cosine results, we can find the similarities between data samples faster and clearer. Thus, this might perform better than the Euclidean metric.

2 Regularized Linear Regression

1. a) Since we have

$$\begin{aligned}
\mathcal{J} &= \frac{1}{2N} \sum_{i-1}^{N} (y^{(i)} - t^{(i)})^2 \\
\mathcal{R} &= \frac{1}{2} \sum_{j=1}^{D} \beta_j w_j^2 \\
\mathcal{J}_{reg}^{\beta} &= \mathcal{J} + \mathcal{R}
\end{aligned}$$

And

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial w_j} &= \frac{\partial}{\partial w_j} (\frac{1}{2N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})^2) \\
&= \frac{\partial}{\partial w_j} (\frac{1}{2N} \sum_{i=1}^{N} (\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} + b - t^{(i)})^2) \\
&= \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} (\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} + b - t^{(i)}) \\
&= \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} (y^{(i)} - t^{(i)}) \\
\frac{\partial \mathcal{R}}{\partial w_j} &= \frac{\partial}{\partial w_j} (\frac{1}{2} \sum_{j=1}^{D} \beta_{j'} w_{j'}^2) \\
&= \beta_j w_j
\end{aligned}$$

We can determine the gradient descent update rules for $\mathcal{J}_{reg}^{\beta}$,

$$\begin{aligned}
w_j &\leftarrow w_j - \alpha \frac{\partial}{\partial w_j}(\mathcal{J} + \mathcal{R}) \\
&= w_j - \alpha(\frac{\partial \mathcal{J}}{\partial w_j} + \frac{\partial \mathcal{R}}{\partial w_j}) \\
&= w_j - \alpha(\frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)}) + \beta_j w_j) \\
&= (1 - \alpha\beta_j)w_j - \frac{\alpha}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)})
\end{aligned}$$

Since

$$\begin{aligned}
y &= \sum_{j=1}^{D} w_j x_j^{(i)} + b \\
b &= y - \sum_{j=1}^{D} w_j x_j^{(i)}
\end{aligned}$$

We have

$$\begin{aligned}
b &\leftarrow y - \sum_{j=1}^{D}((1 - \alpha\beta_j)w_j - \frac{\alpha}{N}\sum_{i=1}^{N} x_j^{(i)}(y^{(i)} - t^{(i)}))x_j^{(i)} \\
&= y - \sum_{j=1}^{D}((1 - \alpha\beta_j)w_j x_j^{(i)} - \frac{\alpha}{N}\sum_{i=1}^{N}(x_j^{(i)})^2(y^{(i)} - t^{(i)}))
\end{aligned}$$

We need the "weight decay" in order to minimize the regularized cost function $\mathcal{J}_{reg}^{\beta}$.

2. b) We have

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

We can define an indicator function

$$\begin{aligned}
i' &= \begin{cases} 1 & if\ i' = j \\ 0 & if\ i' \neq j \end{cases} \\
&= \mathbb{1}[i' = j]
\end{aligned}$$

Then

$$\frac{\partial}{\partial w_j}\mathcal{J}_{reg}^{\beta} = \frac{1}{N}\sum_{j'=1}^{D}(\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} + N\beta_{j'}\mathbb{1}[i' = j])w_{j'} - \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} t^{(i)}$$

Then $\frac{\partial}{\partial w_j}\mathcal{J}^{\beta}_{reg} = \sum_{j'=1}^{D}\mathbf{A}_{jj'}\mathbf{w}_{j'} - \mathbf{c}_j = 0$, where

$$\mathbf{A}_{jj'} = \frac{1}{N}\sum_{i=1}^{N}(x_j^{(i)}x_{j'}^{(i)} + N\beta_{j'}\mathbb{1}[i' = j])$$
$$\mathbf{c}_j = \frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}t^{(i)}$$

3. c) Here is the formulas for $\mathbf{A}$ and $\mathbf{c}$

$$\mathbf{A} = \mathbf{X}^T\mathbf{X} + N\beta\mathbf{I}$$
$$\mathbf{c} = \mathbf{X}^T\mathbf{t}$$

Then we have

$$\begin{aligned}\mathbf{A}\mathbf{w} - \mathbf{c} &= 0\\ \mathbf{A}\mathbf{w} &= \mathbf{c}\\ \mathbf{w} &= (\mathbf{A})^{-1}\mathbf{c}\\ &= (\mathbf{X}^T\mathbf{X} + N\beta\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}\end{aligned}$$

3 Loss Functions

For y, we have

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

For $\frac{\partial \mathcal{J}}{\partial y}$, we have

$$\begin{aligned}\frac{\partial \mathcal{J}}{\partial y} &= \frac{\partial}{\partial y}(\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y^{(i)}, t^{(i)}))\\ &= \frac{\partial}{\partial y}(\frac{1}{N}\sum_{i=1}^{N}(1 - cos(y^{(i)} - t^{(i)})))\\ &= \frac{1}{N}\sum_{i=1}^{N}(-(-sin(y^{(i)} - t^{(i)})))\\ &= \frac{1}{N}\sum_{i=1}^{N}sin(y^{(i)} - t^{(i)})\end{aligned}$$

For $\frac{\partial \mathcal{J}}{\partial w_j}$, we have

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial w_j} &= \frac{\partial}{\partial w_j}(\frac{1}{N}\sum_{i=1}^{N}(1 - cos(\sum_{j=1}^{D} w_j x_j^{(i)} + b - t^{(i)}))) \\
&= \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}(-(-sin(\sum_{j=1}^{D} w_j x_j^{(i)} + b - t^{(i)}))) \\
&= \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} sin(y^{(i)} - t^{(i)})
\end{aligned}$$

For $\frac{\partial \mathcal{J}}{\partial w}$, we have

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix} = \begin{pmatrix} \frac{1}{N}\sum_{i=1}^{N} x_1^{(i)} sin(y^{(i)} - t^{(i)}) \\ \vdots \\ \frac{1}{N}\sum_{i=1}^{N} x_D^{(i)} sin(y^{(i)} - t^{(i)}) \end{pmatrix}$$

For $\frac{\partial \mathcal{J}}{\partial b}$, we have

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial w_j} &= \frac{\partial}{\partial w_j}(\frac{1}{N}\sum_{i=1}^{N}(1 - cos(\mathbf{w}^T\mathbf{x} + b - t^{(i)}))) \\
&= \frac{1}{N}\sum_{i=1}^{N}(-(-sin(\mathbf{w}^T\mathbf{x} + b - t^{(i)}))) \\
&= \frac{1}{N}\sum_{i=1}^{N} sin(y^{(i)} - t^{(i)})
\end{aligned}$$

4 Cross Validation

1. a) Code is provided in the file called "hw1_q4_code.py".

2. b) Code is provided in the file called "hw1_q4_code.py".

3. c) Code is provided in the file called "hw1_q4_code.py".

   Here is the training and test errors corresponding to each λ in *lambd_seq*:

```
lambda: 5e-05
    train error: 0.02382739814261592
    test error: 2.601391644823139
lambda: 0.00015102040816326530
    train error: 0.0589127457391339
    test error: 1.728767799404156
lambda: 0.00025204081632653066
    train error: 0.08829510341160735
    test error: 1.449041107721427
lambda: 0.000353061224489796
    train error: 0.11469870650014886
```

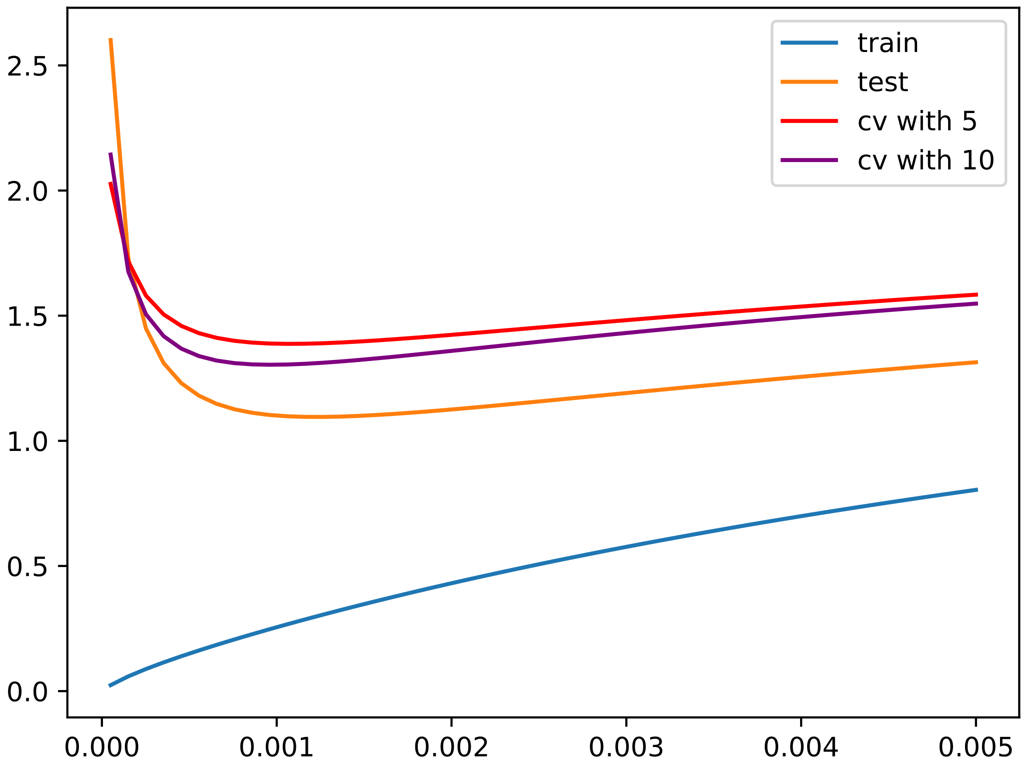
```
    test error: 1.3106086900637666
lambda: 0.0004540816326530613
    train error: 0.13923988850774663
    test error: 1.230498596164056
lambda: 0.0005551020408163266
    train error: 0.16248341903497207
    test error: 1.1803892554728757
lambda: 0.000656122448979592
    train error: 0.1847466058035631
    test error: 1.1477791855291437
lambda: 0.0007571428571428573
    train error: 0.20622199234382144
    test error: 1.1262495170677957
lambda: 0.0008581632653061226
    train error: 0.22703389988077488
    test error: 1.1121518828937422
lambda: 0.0009591836734693879
    train error: 0.24726692464648706
    test error: 1.1032544106716153
lambda: 0.001060204081632653
    train error: 0.266981325079457
    test error: 1.098113519790282
lambda: 0.0011612244897959184
    train error: 0.28622180470505526
    test error: 1.095753848148265
lambda: 0.0012622448979591838
    train error: 0.3050227607306164
    test error: 1.0954928990341022
lambda: 0.001363265306122449
    train error: 0.32341154386293686
    test error: 1.0968392292395672
lambda: 0.0014642857142857144
    train error: 0.34141055052213554
    test error: 1.0994304538825792
lambda: 0.0015653061224489796
    train error: 0.35903860424252554
    test error: 1.102993965144239
lambda: 0.001666326530612245
```

```
    train error: 0.3763118905865588
    test error: 1.1073211894755193
lambda: 0.0017673469387755104
    train error: 0.3932446038612772
    test error: 1.1122502216496664
lambda: 0.0018683673469387756
    train error: 0.40984940331483316
    test error: 1.1176538117834809
lambda: 0.001969387755102041
    train error: 0.4261377407022197
    test error: 1.1234308705075875
lambda: 0.0020704081632653064
    train error: 0.44212009936214175
    test error: 1.1295003442370222
lambda: 0.002171428571428572
    train error: 0.45780617138961804
    test error: 1.1357967224805827
lambda: 0.002272448979591837
    train error: 0.47320499084143147
    test error: 1.142266691137402
lambda: 0.0023734693877551023
    train error: 0.4883250352812423
    test error: 1.1488666047195637
lambda: 0.002474489795918368
    train error: 0.5031743042367001
    test error: 1.1555605531143072
lambda: 0.002575510204081633
    train error: 0.5177603806219528
    test error: 1.1623188662165589
lambda: 0.0026765306122448983
    train error: 0.5320904794538495
    test error: 1.1691169452851022
lambda: 0.0027775510204081635
    train error: 0.5461714869921586
    test error: 1.1759343410147318
lambda: 0.002878571428571429
    train error: 0.5600099925916273
    test error: 1.1827540199582742
```

```
lambda: 0.0029795918367346943
    train error: 0.5736123149542308
    test error: 1.1895617761946025
lambda: 0.0030806122448979595
    train error: 0.5869845240389067
    test error: 1.1963457560476192
lambda: 0.003181632653061225
    train error: 0.6001324595729604
    test error: 1.2030960715562673
lambda: 0.0032826530612244903
    train error: 0.6130617468798987
    test error: 1.2098044841755542
lambda: 0.0033836734693877555
    train error: 0.6257778105688379
    test error: 1.216464144465608
lambda: 0.003484693877551021
    train error: 0.638285886504309
    test error: 1.2230693767226455
lambda: 0.0035857142857142863
    train error: 0.6505910323804636
    test error: 1.2296154999172526
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: 0.6863369856278859
    test error: 1.248864348839178
lambda: 0.003989795918367347
    train error: 0.697877736093214
    test error: 1.2551423617905229
lambda: 0.004090816326530612
    train error: 0.7092384724728255
    test error: 1.26134830577148
lambda: 0.004191836734693878
    train error: 0.7204233532423233
```

```
    test error: 1.2674810372558627
lambda: 0.004292857142857143
    train error: 0.7314364093542272
    test error: 1.2735397425155068
lambda: 0.004393877551020408
    train error: 0.7422815494277315
    test error: 1.279523890635272
lambda: 0.004494897959183674
    train error: 0.7529625646135095
    test error: 1.2854331932216376
lambda: 0.004595918367346939
    train error: 0.763483133169546
    test error: 1.2912675697720937
lambda: 0.004696938775510204
    train error: 0.7738468247780174
    test error: 1.2970271178472275
lambda: 0.00479795918367347
    train error: 0.7840571046284169
    test error: 1.3027120873299656
lambda: 0.004898979591836735
    train error: 0.7941173372883514
    test error: 1.3083228581730004
lambda: 0.005
    train error: 0.8040307903801959
    test error: 1.3138599211313071
```

4. d) Here is the plot containing training error, test error, and 5-fold and 10-fold cross validation errors on the same plot for each value in *lambd_seq*:



Here are the λ proposed by the cross validation procedure:

```
5-fold cross validation lambda: 0.001060204081632653
10-fold cross validation lambda: 0.0009591836734693879
```

From the plot, decreasing trends can be observed in the test error, 5-fold cv error, and 10-fold cv error, and the error of 5-fold cv stays lowest among those three errors.

Since we are finding the value of λ that produce the lowest error in 5-fold cv or in 10-fold cv, we need to keep increasing λ to find the critical point(s) in the plot.