Q1

$$E_{aug}(W) = E_{in}(W) + \frac{\lambda}{N} W^T W, \quad \lambda > 0$$

Optimize the error:

$$w(t+1) = w - \eta \nabla E_{aug}(w(t))$$

$\Updownarrow$ equivalent

$$w(t+1) = s \cdot w(t) - \eta \nabla E_{in}(w(t))$$

s:

$$\nabla E_{aug}(w) = \nabla E_{in}(w) + \frac{2\lambda}{N} w$$

$$w - \eta \nabla E_{aug}(w)$$

$$= w - \eta \nabla E_{in}(w) - \eta \frac{2\lambda}{N} w$$

$$= \left(1 - \eta \frac{2\lambda}{N}\right) w - \eta \nabla E_{in}(w)$$

$$s = 1 - \eta \frac{2\lambda}{N}$$

#

Q2

N labels : $\{y_n\}_{n=1}^{N}$, $y_n \in R$

One - variable regularized problem :

$$\min_{w \in R} \boxed{\frac{1}{N}\sum_{n=1}^{N}(w-y_n)^2 + \frac{\lambda}{N}w^2} \quad \text{— A}$$

$\Updownarrow$ equivalent $\boxed{w^*,\ (w^*)^2 = C}$

$$\min_{w \in R} \frac{1}{N}\sum_{n=1}^{N}(w-y_n)^2 \quad \text{s.t.} \quad w^2 \le C$$

Relationship :

$$\frac{d}{dw}A = 0 \Rightarrow \frac{d}{dw}E_{in} + \frac{2\lambda}{N}w = 0$$

$$\frac{d}{dw}E_{in}$$

$$= \frac{1}{N}\sum_{n=1}^{N}(2w - 2y_n)$$

$$= \frac{2}{N}\left(Nw - \sum_{n=1}^{N}y_n\right)$$

$$= 2w - \frac{2}{N}\sum_{n=1}^{N}y_n$$

$$\begin{cases} (w^*)^2 = C \\ 2w^* - \frac{2}{N}\sum_{n=1}^{N} y_n + \frac{2\lambda}{N} w^* = 0 \end{cases}$$

$$\Downarrow$$

$$w^*\left(2 + \frac{2\lambda}{N}\right) = \frac{2}{N}\sum_{n=1}^{N} y_n$$

$$w^* = \frac{\frac{2}{N}\sum_{n=1}^{N} y_n}{2 + \frac{2\lambda}{N}}$$

$$= \frac{2\sum_{n=1}^{N} y_n}{2N + 2\lambda}$$

$$= \frac{\sum_{n=1}^{N} y_n}{N + \lambda}$$

$$(w^*)^2 = C = \left(\frac{\sum_{n=1}^{N} y_n}{N + \lambda}\right)^2$$

#

Q3

$$\{(x_n, y_n)\}_{n=1}^{N}$$

$\downarrow$ $\Phi(x) = Vx$, V: diagonal matrix,

diagonal component $\rightarrow$ positive value

$$\{(\Phi(x_n), y_n)\}_{n=1}^{N}$$

$$\min_{\tilde{w} \in R^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{w}^T \Phi(x_n) - y_n)^2 + \frac{\lambda}{N} (\tilde{w}^T \tilde{w}) \quad \text{——} ①$$

$\Updownarrow$ equivalent

$$\min_{w \in R^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (w^T x_n - y_n)^2 + \frac{\lambda}{N} (w^T U w) \quad \text{——} ②$$

For ①:

E_{in}

$$= \frac{1}{N} \sum_{n=1}^{N} (\tilde{w}^T V x_n - y_n)^2$$

$$= \frac{1}{N} \left\| \begin{matrix} (Vx_1)^T \tilde{w} - y_1 \\ \cdots \\ (Vx_N)^T \tilde{w} - y_N \end{matrix} \right\|^2$$

$$= \frac{1}{N} \left\| \begin{matrix} x_1^T V^T \tilde{w} - y_1 \\ \cdots \\ x_N^T V^T \tilde{w} - y_N \end{matrix} \right\|^2$$

$V^T = V$

$$= \frac{1}{N} \| XV\tilde{w} - y \|^2$$

Let $\tilde{w} = Mw$

$$= \frac{1}{N} \| XVMw - y \|^2$$

$$E_{aug}\ ① = \frac{1}{N} \| XVMw - y \|^2 + \frac{\lambda}{N} w^T M^T M w$$

$$E_{aug}\ ② = \frac{1}{N} \| Xw - y \|^2 + \frac{\lambda}{N} w^T U w$$

If $M = V^{-1}$ and $U = (V^{-1})^T V^{-1} = \boxed{(V^{-1})^2}$, the equivalence between two E_{aug} above will hold.

#

Q4

$$\Phi(x) = x + \varepsilon, \quad \varepsilon = (\varepsilon_0, \dots, \varepsilon_d), \quad \varepsilon \sim N(0, \sigma^2 I)$$

$$\min_{w \in R^{d+1}} \mathbb{E}\left\{ \frac{1}{N} \sum_{n=1}^{N} \left[w^T \Phi(x_n) - y_n \right]^2 \right\} \quad \text{— ①}$$

$$\downarrow \text{equivalent}$$

$$\min_{w \in R^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (w^T x_n - y_n)^2 + \frac{\lambda}{N} \|w\|_2^2 \quad \text{— ②}$$

For ①

Let $\Phi(x) = \tilde{x}$

$$\|\tilde{X} w - y\|^2, \quad \tilde{X} = \begin{bmatrix} \tilde{x}_1^T \\ \vdots \\ \tilde{x}_N^T \end{bmatrix} = \begin{bmatrix} (\tilde{x}_1 + \varepsilon_1)^T \\ \vdots \\ (\tilde{x}_N + \varepsilon_N)^T \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} \tilde{x}_1^T \\ \vdots \\ \tilde{x}_N^T \end{bmatrix}}_{X} + \underbrace{\begin{bmatrix} \varepsilon_1^T \\ \vdots \\ \varepsilon_N^T \end{bmatrix}}_{E}$$

$$\Downarrow$$

$$\frac{1}{N}\sum_{n=1}^{N}\left[w^T\Phi(x_n) - y_n\right]^2$$

$$= \frac{1}{N}\left\|\tilde{X}w - y\right\|^2$$

$$= \frac{1}{N}\left\|(X+E)w - y\right\|^2$$

$$= \frac{1}{N}\left(w^T(X+E)^T(X+E)w - 2w^T(X+E)^Ty + y^Ty\right)$$

$$= \frac{1}{N}\left[w^T(X^TX + X^TE + E^TX + E^TE)w - 2w^T(X^T+E^T)y + y^Ty\right]$$

$$= \frac{1}{N}\left[w^TX^TXw + w^TX^TEw + w^TE^TXw + w^TE^TEw - 2w^TX^Ty - 2w^TE^Ty + y^Ty\right]$$

$$\mathbb{E}\left\{\frac{1}{N}\left[w^TX^TXw + w^TX^TEw + w^TE^TXw + w^TE^TEw - 2w^TX^Ty - 2w^TE^Ty + y^Ty\right]\right\}$$

$$= \frac{1}{N}\left[w^T X^T X w + \underline{w^T X^T \mathbb{E}(\varepsilon) w} + \underline{w^T \mathbb{E}(\varepsilon^T) X w} + \underline{w^T \mathbb{E}(\varepsilon^T \varepsilon) w} - 2 w^T X^T y - \underline{2 w^T \mathbb{E}(\varepsilon^T) y} + y^T y \right]$$

$$\underbrace{\begin{bmatrix} \varepsilon_1 & \cdots & \varepsilon_N \end{bmatrix}}_{\varepsilon^T} \underbrace{\begin{bmatrix} \varepsilon_1^T \\ \vdots \\ \varepsilon_N^T \end{bmatrix}}_{\varepsilon}$$

$$\varepsilon^T \varepsilon = \varepsilon_1 \varepsilon_1^T + \cdots + \varepsilon_N \varepsilon_N^T$$

$$\mathbb{E}(\varepsilon^T \varepsilon) = \mathbb{E}(\varepsilon_1 \varepsilon_1^T + \cdots + \varepsilon_N \varepsilon_N^T)$$
$$= \mathbb{E}(\varepsilon_1 \varepsilon_1^T) + \cdots + \mathbb{E}(\varepsilon_N \varepsilon_N^T)$$
$$= N\sigma^2 I_{(d+1)\times(d+1)}$$

$$\mathbb{E}(\varepsilon) = 0_{N\times(d+1)}$$

$$\mathbb{E}(\varepsilon^T) = 0_{(d+1)\times N}$$

① $(d+1) \cdot (d+1)\times N \cdot N\times(d+1) \cdot (d+1)$
$= 1\times N \cdot 0_{N\times 1} = 0$

② $1\times(d+1) \cdot 0_{(d+1)\times N} \cdot N\times(d+1) \cdot (d+1)\times 1$
$= 0_{1\times N} \cdot N\times(d+1) \cdot (d+1)\times 1 = 0$

③ $1\times(d+1) \cdot N\sigma^2 \cdot I_{(d+1)\times(d+1)} \cdot (d+1)\times 1$
$= N\sigma^2 w^T w$

④ $1\times(d+1) \cdot 0_{(d+1)\times N} \cdot N\times 1$
$= 0$

$$= \frac{1}{N}\left[w^T X^T X w + \underbrace{0}_{①} + \underbrace{0}_{②} + \underbrace{N\sigma^2 w^T w}_{③} - 2 w^T X^T y - \underbrace{0}_{④} + y^T y \right]$$

$$= \frac{1}{N}\left(w^T X^T X w - 2 w^T X^T y + y^T y + N\sigma^2 w^T w \right)$$

$$= \frac{1}{N} \| XW - y \|^2 + \sigma^2 W^T W$$

$$\Rightarrow \frac{\lambda}{N} = \sigma^2, \quad \lambda = N\sigma^2$$

#

Q5

Head probability estimated by $\longrightarrow \dfrac{\sum_{n=1}^{N} y_n + \alpha}{N + \alpha k}$

Let y that we estimate be "w". And y be $\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$.

Let $x = [x_0] = [1]$, $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$.

Rewrite optimization problem:

$$\min_{w \in R} \underbrace{\frac{1}{N} \sum_{n=1}^{N} (wx_n - y_n)^2 + \frac{\alpha k}{N} \Omega(w)}_{E_{aug}}$$

$$E_{aug} = \frac{1}{N} \| Xw - y \|^2 + \frac{\alpha k}{N} \Omega(w)$$

$$= \frac{1}{N} \left(w^T x^T x w - 2 w^T x^T y + y^T y \right) + \frac{\alpha k}{N} \Omega(w)$$

∇E_{aug}

$= \dfrac{\partial}{\partial w} E_{aug}$

$= \dfrac{2}{N} \left(x^T x w - x^T y \right) + \dfrac{\alpha k}{N} \times \dfrac{\partial}{\partial w} \Omega(w)$

Let $\nabla E_{aug} = 0$:

$$\frac{2}{N}(X^T X w - \underbrace{X^T y}_{\Sigma y_n}) + \frac{\alpha k}{N}\frac{\partial \Omega(w)}{\partial w} = 0$$

$$2X^T X w + \alpha k \frac{\partial \Omega(w)}{\partial w} = 2\Sigma y_n$$

$$2Nw + \alpha k \frac{\partial \Omega(w)}{\partial w} = 2\Sigma y_n$$

$$w = \frac{2\Sigma y_n - \alpha k \frac{\partial \Omega(w)}{\partial w}}{2N}$$

w = estimated head probability :

$$\frac{2\Sigma y_n - \alpha k \frac{\partial \Omega(w)}{\partial w}}{2N} = \frac{\sum_{n=1}^{N} y_n + \alpha}{N + \alpha k}$$

$$2\Sigma y_n - \alpha k \frac{\partial \Omega(w)}{\partial w} = \frac{2N\Sigma y_n + 2N\alpha}{N + \alpha k}$$

$$\alpha k \frac{\partial \Omega(w)}{\partial w} = \frac{\cancel{2N\Sigma y_n} + 2\alpha k \Sigma y_n - \cancel{2N\Sigma y_n} - 2N\alpha}{N + \alpha k}$$

$$\alpha k \frac{\partial \Omega(w)}{\partial w} = \frac{2\alpha k \Sigma y_n - 2N\alpha}{N + \alpha k} = \frac{2\alpha k \left(\Sigma y_n - \frac{N}{k}\right)}{N + \alpha k}$$

$$\frac{\partial \Omega(w)}{\partial w} = \frac{2(\Sigma y_n - \frac{N}{k})}{N + \alpha k}$$

$$= \frac{2\Sigma y_n - \frac{2N}{k} + 2\alpha - 2\alpha}{N + \alpha k}$$

$$= \frac{2(\Sigma y_n + \alpha) - \frac{2}{k}(N + \alpha k)}{N + \alpha k}$$

$N + \alpha k$ → w

$$= 2w - \frac{2}{k}$$

$\Rightarrow$ The final result is the same as the gradient of [d].

#

Q6

$$g = \frac{1}{2}(w-w^*)^T H (w-w^*)$$

$$= \frac{1}{2}\left[w^T H - (w^*)^T H\right](w-w^*)$$

$$= \frac{1}{2}\left[\underline{w^T H w} - \underline{w^T H w^*} - \underline{(w^*)^T H w} + (w^*)^T H w^*\right]$$

$$\frac{\partial g}{\partial w} = \nabla g$$

$$= \frac{1}{2}\left[\underline{Hw + H^T w} - \underline{Hw^*} - \underline{H^T w^*}\right]$$

$$= \frac{1}{2}\left[2Hw - 2Hw^*\right]$$

$$= Hw - Hw^*$$

$$\frac{\partial \tilde{E}_{aug}}{\partial w} = \nabla \tilde{E}_{aug}$$

$$= \nabla \tilde{E}_{in}(w) + \frac{2\lambda}{N} w$$

$$= \nabla g + \frac{2\lambda}{N} w$$

$$= Hw - Hw^* + \frac{2\lambda}{N} w$$

$\nabla \tilde{E}_{aug} = 0$:

$$Hw - Hw^* + \frac{2\lambda}{N} w = 0$$

$$\left(H + \frac{2\lambda}{N} I\right) w = Hw^*$$

$$w = \left(H + \frac{2\lambda}{N} I\right)^{-1} Hw^*$$

#

Q7

Binary $\longrightarrow$ A_{minority}

$D \rightarrow N$ positive, N negative

$$E_{\text{loocv}}(A_{\text{minority}}) = \frac{1}{2N}\sum_{n=1}^{2N} e_n$$

$$= \frac{1}{2N}\sum_{n=1}^{2N} err(g_n^-(x_n), y_n)$$

If $y_n = +1$

$+1 \rightarrow$ less

$g_n^-(x_n) = +1$, equal to y_n

$err(g_n^-(x_n), y_n) = 0$

if $y_n = -1$

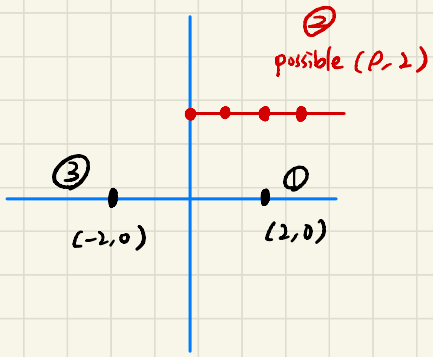
$-1 \rightarrow$ less

$g_n^-(x_n) = -1$, equal to y_n

$err(g_n^-(x_n), y_n) = 0$

$$\Rightarrow \frac{1}{2N}\sum_{n=1}^{2N} err(g_n^-(x_n), y_n) = \frac{1}{2N}\sum_{n=1}^{2N} 0 = 0 \quad \#$$

Q8



$(x_1, y_1) = (2, 0)$

$(x_2, y_2) = (\rho, 2),\ \rho > 0$

$(x_3, y_3) = (-2, 0)$

For linear:

$$\bar{g_1} = \frac{4}{\rho+2} + \frac{2}{\rho+2}x$$

$$err(\bar{g_1}(x_1), y_1)$$

$$= err\left(\frac{4}{\rho+2} + \frac{4}{\rho+2}, 0\right)$$

$$= \underline{\left(\frac{8}{\rho+2} - 0\right)^2}$$

$$\bar{g_2} = 0$$

$$err(\bar{g_2}(x_2), y_2)$$

$$= err(0, 2)$$

$$= \underline{(0-2)^2}$$

$$\bar{g_3} = \frac{-4}{\rho-2} + \frac{2}{\rho-2}x$$

$$err(\bar{g_3}(x_3), y_3) = err\left(\frac{-4}{\rho-2} + \frac{-4}{\rho-2}, 0\right) = \underline{\left(\frac{-8}{\rho-2} - 0\right)^2}$$

For constant :

$g_1^- = 1$

$err(g_1^-(x_1), y_1)$

$= err(1, 0)$

$= \underline{(1-0)^2}$

$g_2^- = 0$

$err(g_2^-(x_2), y_2)$

$= err(0, 2)$

$= \underline{(0-2)^2}$

$g_3^- = 1$

$err(g_3^-(x_3), y_3)$

$= (1, 0)$

$= \underline{(1-0)^2}$

For problem: Make $E_{Loocv}(A_{linear}) = E_{Loocv}(A_{constant})$

$$\frac{1}{3}\left[\left(\frac{8}{\rho+2}\right)^2 + 4 + \left(\frac{-8}{\rho-2}\right)^2\right] = \frac{1}{3}(1+4+1)$$

$$\left(\frac{8}{\rho+2}\right)^2 + \left(\frac{-8}{\rho-2}\right)^2 + 4 = 6$$

$$\underbrace{\left(\frac{8}{\rho+2}\right)^2 + \left(\frac{-8}{\rho-2}\right)^2}_{z} = 2$$

Evaluate each option:

[a] $z = \left(\frac{8}{8.5}\right)^2 + \left(\frac{-8}{4.5}\right)^2 \approx 4.05$

[b] $z = \left(\frac{8}{9.5}\right)^2 + \left(\frac{-8}{5.5}\right)^2 \approx 2.82$

[c] $z = \left(\frac{8}{10.5}\right)^2 + \left(\frac{-8}{6.5}\right)^2 \approx 2.1$

[d] $z = \left(\frac{8}{11.5}\right)^2 + \left(\frac{-8}{7.5}\right)^2 \approx 1.62$

[e] $z = \left(\frac{8}{12.5}\right)^2 + \left(\frac{-8}{8.5}\right)^2 \approx 1.3$

$\Rightarrow$ [c] is the closest. #

Q9

$$\mathbb{E}\left(\frac{1}{K}\sum_{n=N-K+1}^{N}(y_n-\bar{y})^2\right)$$

$$=\frac{1}{K}\sum_{n=N-K+1}^{N}\mathbb{E}(y_n^2-2y_n\bar{y}+\bar{y}^2)$$

$$=\frac{1}{K}\sum_{n=N-K+1}^{N}\left[\mathbb{E}(y_n^2)-2\underline{\mathbb{E}(y_n\bar{y})}+\mathbb{E}(\bar{y}^2)\right]$$

$\underline{\mathbb{E}(y_n\bar{y})} = \mathbb{E}(y_n)\mathbb{E}(\bar{y})$ ← Independence between y_n and $\bar{y}$

$= \mathbb{E}(y_n) \times 0 = 0$

$$\because \text{Var}(\bar{y}) = \mathbb{E}(\bar{y}^2) - [\mathbb{E}(\bar{y})]^2$$

$$= \mathbb{E}(\bar{y}^2) - 0^2$$

$$= \mathbb{E}(\bar{y}^2)$$

$\therefore \mathbb{E}(\bar{y}^2)$ above is equal to $\text{Var}(\bar{y})$

$$\text{Var}(\bar{y}) = \text{Var}\left(\frac{1}{N-K}\underline{\sum_{n=1}^{N-K} y_n}\right)$$

y_n is i.i.d.

$$= \frac{1}{(N-K)^2}\underline{\sum_{n=1}^{N-K}\text{Var}(y_n)}$$

$$= \frac{1}{(N-K)^2} \times (N-K)\sigma^2 = \frac{1}{(N-K)}\sigma^2$$

$$= \frac{1}{K} \sum_{n=N-K+1}^{N} \left(\sigma^2 + \frac{1}{(N-K)} \sigma^2 \right)$$

$$= \frac{1}{\cancel{K}} \times \cancel{K} \left(\sigma^2 + \frac{1}{(N-K)} \sigma^2 \right)$$

$$= \sigma^2 + \frac{1}{(N-K)} \sigma^2$$

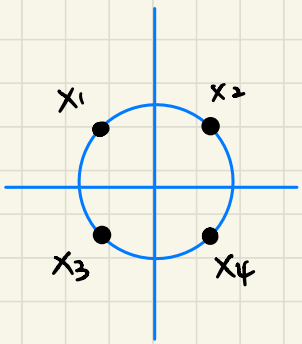
#

Q10

$E_{in}^{0/1}(w)$ minimized by 2D perceptron model.

Measure the power of the model : $\mathbb{E}_{y_1, y_2, y_3, y_4}\left(\min_{w \in R^{2+1}} E_{in}(w)\right)$

Given an example :

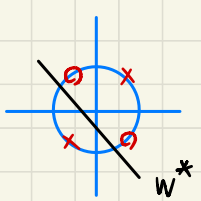


Let $Y = [y_1, y_2, y_3, y_4]$

Total number of combination of Y : 16.

Only 2 combinations cannot be shattered $\Rightarrow E_{in}^{0/1}(w^*) \neq 0$

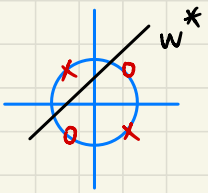
$Y = (1, -1, -1, 1)$



$\Rightarrow E_{in}^{0/1}(w^*) = \frac{1}{4}$

(probability of occurrence : $\frac{1}{16}$)

$$y = (-1, 1, 1, -1)$$



$$\Rightarrow E_{in}^{y_i}(w^*) = \frac{1}{4}$$

(probability of occurrence : $\frac{1}{16}$)

Therefore, we can compute

$$\mathbb{E}_{y_1, y_2, y_3, y_4}\left(\min_{w \in R^{2+1}} E_{in}(w)\right)$$

$$= \frac{1}{16} \times \frac{1}{4} + \frac{1}{16} \times \frac{1}{4}$$

$$= \boxed{\frac{1}{32}} \#$$

Q11

Test distribution: $P(y=+1) = p$

$E_{out}(g) = p \times \varepsilon_+ + (1-p) \times \varepsilon_-$

$E_{out}(g^-) = p$

Calculation:

$$p\varepsilon_+ + (1-p)\varepsilon_- = p$$

$$p\varepsilon_+ + \varepsilon_- - p\varepsilon_- = p$$

$$p - p\varepsilon_+ + p\varepsilon_- = \varepsilon_-$$

$$p(1 - \varepsilon_+ + \varepsilon_-) = \varepsilon_-$$

$$p = \frac{\varepsilon_-}{1 - \varepsilon_+ + \varepsilon_-}$$

#

Q12 ~ Q16

Figure out relationship between C and λ.

$$W_{\lambda_1} = \underset{W}{\operatorname{argmin}} \frac{\lambda}{N}\|w\|^2 + \frac{1}{N}\sum err \quad - ①$$

$$W_{\lambda_2} = \underset{W}{\operatorname{argmin}} \frac{1}{\Sigma}\|w\|^2 + C\sum err \quad - ②$$

① : $\frac{2\lambda}{N} W + \frac{1}{N}\sum \nabla err = 0$

$$W = -\frac{1}{2\lambda}\sum \nabla err$$

② : $W + C\sum \nabla err = 0$

$$W = -C\sum \nabla err$$

should be the same.

$$\Rightarrow C = \frac{1}{2\lambda} \quad \#$$

Q12- code

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 10 11:08:18 2021

@author: Eric
"""

import numpy as np
import itertools
from liblinear.liblinearutil import *


def polynomialTransform(x, orderQ):
    dim = x.shape[1]
    # add x0 = 1
    z = np.insert(x, 0, 1, axis=1)
    # add higher order
    for num in range(2, orderQ+1):
        combSet = np.array(list(itertools.combinations_with_replacement(np.arange(dim), num)))
        for comb in combSet:
            # multiplyTerm = x[:, comb[0]] * x[:, comb[1]]
            multiplyTerm = np.prod(x[:, comb], axis=1)
            z = np.concatenate((z, multiplyTerm[:, None]), axis=1)
    return z


# parameters
trainDataPath = "hw4_train.txt"
testDataPath = "hw4_test.txt"
Q = 3
log10LambdaSet = [-4, -2, 0, 2, 4]


# read data
trainData = np.genfromtxt(trainDataPath)
testData = np.genfromtxt(testDataPath)


# full order polynomial transform
trainData = np.concatenate((polynomialTransform(trainData[:, :-1], orderQ=Q), trainData[:, -1][:,
None]), axis=1)
testData = np.concatenate((polynomialTransform(testData[:, :-1], orderQ=Q), testData[:, -1][:, None]),
axis=1)


# model training and testing
prob = problem(trainData[:, -1], trainData[:, :-1])
for log10Lambda in log10LambdaSet:
    actualLambda = 10**(log10Lambda)
    C = 1 / (2*actualLambda)
    param = parameter('-s 0 -c {} -e 0.000001'.format(C))
    model = train(prob, param)
    print("log10Lambda:", log10Lambda)
    p_label, p_acc, p_val = predict(testData[:, -1], testData[:, :-1], model)
    Eout = 100-p_acc[0]
    print("Eout: {}%".format(Eout))
    print()
```

```
In [1]: runfile('C:/Users/Eric/Desktop/ml21fall/hw4/Q12.py', wdir='C:/Users/Eric/Desktop/ml21fall/hw4')
log10Lambda: -4
Accuracy = 87.75% (702/800) (classification)
Eout: 12.25%

log10Lambda: -2
Accuracy = 87.75% (702/800) (classification)
Eout: 12.25%

log10Lambda: 0
Accuracy = 92% (736/800) (classification)
Eout: 8.0%

log10Lambda: 2
Accuracy = 92.75% (742/800) (classification)
Eout: 7.25%

log10Lambda: 4
Accuracy = 83.5% (668/800) (classification)
Eout: 16.5%
```

Q12 - result

Q13- code

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 10 18:24:07 2021

@author: Eric
"""

import numpy as np
import itertools
from liblinear.liblinearutil import *


def polynomialTransform(x, orderQ):
    dim = x.shape[1]
    # add x0 = 1
    z = np.insert(x, 0, 1, axis=1)
    # add higher order
    for num in range(2, orderQ+1):
        combSet = np.array(list(itertools.combinations_with_replacement(np.arange(dim), num)))
        for comb in combSet:
            # multiplyTerm = x[:, comb[0]] * x[:, comb[1]]
            multiplyTerm = np.prod(x[:, comb], axis=1)
            z = np.concatenate((z, multiplyTerm[:, None]), axis=1)
    return z


# parameters
trainDataPath = "hw4_train.txt"
testDataPath = "hw4_test.txt"
Q = 3
log10LambdaSet = [-4, -2, 0, 2, 4]


# read data
trainData = np.genfromtxt(trainDataPath)
testData = np.genfromtxt(testDataPath)


# full order polynomial transform
trainData = np.concatenate((polynomialTransform(trainData[:, :-1], orderQ=Q), trainData[:, -1][:,
None]), axis=1)
testData = np.concatenate((polynomialTransform(testData[:, :-1], orderQ=Q), testData[:, -1][:, None]),
axis=1)


# model training and testing
prob = problem(trainData[:, -1], trainData[:, :-1])
for log10Lambda in log10LambdaSet:
    actualLambda = 10**(log10Lambda)
    C = 1 / (2*actualLambda)
    param = parameter('-s 0 -c {} -e 0.000001'.format(C))
    model = train(prob, param)
    print("log10Lambda:", log10Lambda)
    p_label, p_acc, p_val = predict(trainData[:, -1], trainData[:, :-1], model)
    Ein = 100-p_acc[0]
    print("Ein: {}%".format(Ein))
    print()
```

```
In [2]: runfile('C:/Users/Eric/Desktop/ml21fall/hw4/Q13.py', wdir='C:/Users/Eric/Desktop/ml21fall/hw4')
log10Lambda: -4
Accuracy = 100% (200/200) (classification)
Ein: 0.0%

log10Lambda: -2
Accuracy = 100% (200/200) (classification)
Ein: 0.0%

log10Lambda: 0
Accuracy = 97.5% (195/200) (classification)
Ein: 2.5%

log10Lambda: 2
Accuracy = 95% (190/200) (classification)
Ein: 5.0%

log10Lambda: 4
Accuracy = 82% (164/200) (classification)
Ein: 18.0%
```

Q13- result

Q14 - code

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 10 18:30:24 2021

@author: Eric
"""

import numpy as np
import itertools
from liblinear.liblinearutil import *


def polynomialTransform(x, orderQ):
    dim = x.shape[1]
    # add x0 = 1
    z = np.insert(x, 0, 1, axis=1)
    # add higher order
    for num in range(2, orderQ+1):
        combSet = np.array(list(itertools.combinations_with_replacement(np.arange(dim), num)))
        for comb in combSet:
            # multiplyTerm = x[:, comb[0]] * x[:, comb[1]]
            multiplyTerm = np.prod(x[:, comb], axis=1)
            z = np.concatenate((z, multiplyTerm[:, None]), axis=1)
    return z


# parameters
trainDataPath = "hw4_train.txt"
testDataPath = "hw4_test.txt"
Q = 3
log10LambdaSet = [-4, -2, 0, 2, 4]


# read data
trainData = np.genfromtxt(trainDataPath)
testData = np.genfromtxt(testDataPath)


# full order polynomial transform and split data
trainData = np.concatenate((polynomialTransform(trainData[:, :-1], orderQ=Q), trainData[:, -1][:,
None]), axis=1)
Dtrain = trainData[:120, :]
Dval = trainData[120:, :]
testData = np.concatenate((polynomialTransform(testData[:, :-1], orderQ=Q), testData[:, -1][:, None]),
axis=1)


# model training and evaluating Eval
prob = problem(Dtrain[:, -1], Dtrain[:, :-1])
modelCandidates = []
EvalSet = []
for log10Lambda in log10LambdaSet:
    actualLambda = 10**(log10Lambda)
    C = 1 / (2*actualLambda)
    param = parameter('-s 0 -c {} -e 0.000001'.format(C))
    model = train(prob, param)
    print("log10Lambda:", log10Lambda)
    p_label, p_acc, p_val = predict(Dval[:, -1], Dval[:, :-1], model)
    Eval = 100-p_acc[0]
    EvalSet.append(Eval)
    print("Eval: {}%".format(Eval))
    print()
    modelCandidates.append(model)


# select the best lambda with respect to best Eval and test for its Eout
bestEval = min(EvalSet)
bestEvalIdx = max(np.where(np.array(EvalSet)==bestEval)[0])
print("Final information for best lambda with respect to best Eval:")
print("Best log10Lambda ->", log10LambdaSet[bestEvalIdx])
p_label, p_acc, p_val = predict(testData[:, -1], testData[:, :-1], modelCandidates[bestEvalIdx])
Eout = 100-p_acc[0]
print("Eout: {}%".format(Eout))
print()
```

```
In [3]: runfile('C:/Users/Eric/Desktop/ml21fall/hw4/Q14.py', wdir='C:/Users/Eric/Desktop/ml21fall/hw4')
log10Lambda: -4
Accuracy = 77.5% (62/80) (classification)
Eval: 22.5%

log10Lambda: -2
Accuracy = 80% (64/80) (classification)
Eval: 20.0%

log10Lambda: 0
Accuracy = 88.75% (71/80) (classification)
Eval: 11.25%

log10Lambda: 2
Accuracy = 93.75% (75/80) (classification)
Eval: 6.25%

log10Lambda: 4
Accuracy = 85% (68/80) (classification)
Eval: 15.0%

Final information for best lambda with respect to best Eval:
Best log10Lambda -> 2
Accuracy = 92.5% (740/800) (classification)
Eout: 7.5%
```

Q14 - result

Q15 - code

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 10 22:48:49 2021

@author: Eric
"""

import numpy as np
import itertools
from liblinear.liblinearutil import *


def polynomialTransform(x, orderQ):
    dim = x.shape[1]
    # add x0 = 1
    z = np.insert(x, 0, 1, axis=1)
    # add higher order
    for num in range(2, orderQ+1):
        combSet = np.array(list(itertools.combinations_with_replacement(np.arange(dim), num)))
        for comb in combSet:
            # multiplyTerm = x[:, comb[0]] * x[:, comb[1]]
            multiplyTerm = np.prod(x[:, comb], axis=1)
            z = np.concatenate((z, multiplyTerm[:, None]), axis=1)
    return z


# parameters
trainDataPath = "hw4_train.txt"
testDataPath = "hw4_test.txt"
Q = 3
log10LambdaSet = [-4, -2, 0, 2, 4]


# read data
trainData = np.genfromtxt(trainDataPath)
testData = np.genfromtxt(testDataPath)


# full order polynomial transform and split data
trainData = np.concatenate((polynomialTransform(trainData[:, :-1], orderQ=Q), trainData[:, -1][:,
None]), axis=1)
Dtrain = trainData[:120, :]
Dval = trainData[120:, :]
testData = np.concatenate((polynomialTransform(testData[:, :-1], orderQ=Q), testData[:, -1][:, None]),
axis=1)


# model training and evaluating Eval
prob = problem(Dtrain[:, -1], Dtrain[:, :-1])
modelCandidates = []
EvalSet = []
for log10Lambda in log10LambdaSet:
    actualLambda = 10**(log10Lambda)
    C = 1 / (2*actualLambda)
    param = parameter('-s 0 -c {} -e 0.000001'.format(C))
    model = train(prob, param)
    print("log10Lambda:", log10Lambda)
    p_label, p_acc, p_val = predict(Dval[:, -1], Dval[:, :-1], model)
    Eval = 100-p_acc[0]
    EvalSet.append(Eval)
    print("Eval: {}%".format(Eval))
    print()
    modelCandidates.append(model)


# select the best lambda with respect to best Eval, re-train with full training set,
# and test for its Eout
bestEval = min(EvalSet)
bestEvalIdx = max(np.where(np.array(EvalSet)==bestEval)[0])
# re-train
prob = problem(trainData[:, -1], trainData[:, :-1])
actualLambda = 10**(log10LambdaSet[bestEvalIdx])
C = 1 / (2*actualLambda)
param = parameter('-s 0 -c {} -e 0.000001'.format(C))
model = train(prob, param)
# test
print("Final information for best lambda with respect to best Eval (after re-training with full
training set):")
print("Best log10Lambda ->", log10LambdaSet[bestEvalIdx])
p_label, p_acc, p_val = predict(testData[:, -1], testData[:, :-1], model)
Eout = 100-p_acc[0]
print("Eout: {}%".format(Eout))
print()
```

```
In [4]: runfile('C:/Users/Eric/Desktop/ml21fall/hw4/Q15.py', wdir='C:/Users/Eric/Desktop/ml21fall/hw4')
log10Lambda: -4
Accuracy = 77.5% (62/80) (classification)
Eval: 22.5%

log10Lambda: -2
Accuracy = 80% (64/80) (classification)
Eval: 20.0%

log10Lambda: 0
Accuracy = 88.75% (71/80) (classification)
Eval: 11.25%

log10Lambda: 2
Accuracy = 93.75% (75/80) (classification)
Eval: 6.25%

log10Lambda: 4
Accuracy = 85% (68/80) (classification)
Eval: 15.0%

Final information for best lambda with respect to best Eval (after re-training with full training set):
Best log10Lambda -> 2
Accuracy = 92.75% (742/800) (classification)
Eout: 7.25%
```

Q15 - result

Q16-code

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 10 23:10:09 2021

@author: Eric
"""

import numpy as np
import itertools
from liblinear.liblinearutil import *


def polynomialTransform(x, orderQ):
    dim = x.shape[1]
    # add x0 = 1
    z = np.insert(x, 0, 1, axis=1)
    # add higher order
    for num in range(2, orderQ+1):
        combSet = np.array(list(itertools.combinations_with_replacement(np.arange(dim), num)))
        for comb in combSet:
            # multiplyTerm = x[:, comb[0]] * x[:, comb[1]]
            multiplyTerm = np.prod(x[:, comb], axis=1)
            z = np.concatenate((z, multiplyTerm[:, None]), axis=1)
    return z


# parameters
trainDataPath = "hw4_train.txt"
testDataPath = "hw4_test.txt"
Q = 3
log10LambdaSet = [-4, -2, 0, 2, 4]
Vfold = 5


# read data
trainData = np.genfromtxt(trainDataPath)
testData = np.genfromtxt(testDataPath)


# full order polynomial transform and split data
trainData = np.concatenate((polynomialTransform(trainData[:, :-1], orderQ=Q), trainData[:, -1][:,
None]), axis=1)
testData = np.concatenate((polynomialTransform(testData[:, :-1], orderQ=Q), testData[:, -1][:, None]),
axis=1)


# model training and evaluating Ecv
trainData = trainData.reshape(Vfold, trainData.shape[0]//Vfold, -1)
modelCandidates = []
DvalIdxSet = [*range(Vfold)]
EcvSet = []
for log10Lambda in log10LambdaSet:
    print("\nlog10Lambda:", log10Lambda)
    # do cross validation
    EvalSet = []
    for DvalIdx in DvalIdxSet:
        # split data
        Dtrain = trainData[np.delete(DvalIdxSet, DvalIdx), :, :].reshape(-1, trainData.shape[-1])
        Dval = trainData[DvalIdx, :, :].reshape(-1, trainData.shape[-1])
        # train model
        actualLambda = 10**(log10Lambda)
        C = 1 / (2*actualLambda)
        prob = problem(Dtrain[:, -1], Dtrain[:, :-1])
        param = parameter('-s 0 -c {} -e 0.000001'.format(C))
        model = train(prob, param)
        # evaluate Eval
        p_label, p_acc, p_val = predict(Dval[:, -1], Dval[:, :-1], model)
        Eval = 100-p_acc[0]
        EvalSet.append(Eval)
    EcvSet.append(np.mean(EvalSet)/100)
print("\nEcvSet:", EcvSet)
```

```
In [5]: runfile('C:/Users/Eric/Desktop/ml21fall/hw4/Q16.py', wdir='C:/Users/Eric/Desktop/ml21fall/hw4')

log10Lambda: -4
Accuracy = 82.5% (33/40) (classification)
Accuracy = 82.5% (33/40) (classification)
Accuracy = 70% (28/40) (classification)
Accuracy = 82.5% (33/40) (classification)
Accuracy = 87.5% (35/40) (classification)

log10Lambda: -2
Accuracy = 82.5% (33/40) (classification)
Accuracy = 82.5% (33/40) (classification)
Accuracy = 75% (30/40) (classification)
Accuracy = 85% (34/40) (classification)
Accuracy = 90% (36/40) (classification)

log10Lambda: 0
Accuracy = 85% (34/40) (classification)
Accuracy = 87.5% (35/40) (classification)
Accuracy = 85% (34/40) (classification)
Accuracy = 92.5% (37/40) (classification)
Accuracy = 92.5% (37/40) (classification)

log10Lambda: 2
Accuracy = 90% (36/40) (classification)
Accuracy = 90% (36/40) (classification)
Accuracy = 87.5% (35/40) (classification)
Accuracy = 97.5% (39/40) (classification)
Accuracy = 95% (38/40) (classification)

log10Lambda: 4
Accuracy = 72.5% (29/40) (classification)
Accuracy = 80% (32/40) (classification)
Accuracy = 75% (30/40) (classification)
Accuracy = 77.5% (31/40) (classification)
Accuracy = 90% (36/40) (classification)

EcvSet: [0.19, 0.17, 0.115, 0.08, 0.21]
```

Q16- result