Regularization

1. ANS [c]

Consider the augmented error

$$E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

with some $\lambda > 0$. Optimize the error by gradient descent with η as the learning rate. That is,

$$\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) - \eta\nabla E_{\text{aug}}(\mathbf{w}(t))$$

The update rule above is equivalent to multiplying $\mathbf{w}(t)$ by a shrinkage factor s before updating with the negative gradient of E_{in}

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

What is s? Choose the correct answer; explain your answer.

[a] $1 - \frac{\eta\lambda}{N}$
[b] $1 - \frac{2\lambda}{N}$
[c] $1 - \frac{2\eta\lambda}{N}$
[d] $1 - \frac{2\eta}{N}$
[e] none of the other choices

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

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

$\Rightarrow w(t+1) \leftarrow w(t) - \eta \nabla \left(E_{in}(w(t)) + \frac{\lambda}{N} w_{(t)}^T w(t) \right)$

equivalent to

$\Rightarrow w(t+1) \leftarrow s \cdot w(t) - \eta \nabla E_{in}(w(t))$, 求 s

$\Rightarrow \nabla \left(E_{in}(w(t)) + \frac{\lambda}{N} w_{(t)}^T w(t) \right)$

$= \frac{\partial E_{in}(w(t))}{\partial w(t)} + \frac{\partial \frac{\lambda}{N} w_{(t)}^T w(t)}{\partial w(t)} = \nabla E_{in}(w(t)) + \frac{2\lambda}{N} w(t)$

$\Rightarrow w(t+1) \leftarrow w(t) - \eta \left(\nabla E_{in}(w(t)) + \frac{2\lambda}{N} w(t) \right)$

$w(t+1) \leftarrow w(t) - \eta \nabla E_{in}(w(t)) - \frac{2\eta\lambda}{N} w(t)$

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

$S = 1 - \frac{2\eta\lambda}{N}$ # 選 C

2. ANS [b]

Consider N "labels" $\{y_n\}_{n=1}^N$ with each $y_n \in \mathbb{R}$. Then, solve the following one-variable regularized problem:

$$\min_{w\in\mathbb{R}} \frac{1}{N}\sum_{n=1}^{N}(w-y_n)^2 + \frac{\lambda}{N}w^2.$$

If the optimal solution to the problem above is w^*, it can be shown that w^* is also the optimal solution of

$$\min_{w\in\mathbb{R}} \frac{1}{N}\sum_{n=1}^{N}(w-y_n)^2 \text{ subject to } w^2 \le C$$

with $C = (w^*)^2$. This allows us to express the relationship between C in the constrained optimization problem and λ in the augmented optimization problem for any $\lambda > 0$. What is the relationship? Choose the correct answer; explain your answer.

[a] $C = \left(\dfrac{\sum_{n=1}^{N} y_n}{N+N\lambda}\right)^2$

[b] $C = \left(\dfrac{\sum_{n=1}^{N} y_n}{N+\lambda}\right)^2$

[c] $C = \left(\dfrac{\sum_{n=1}^{N} y_n^2}{\sum_{n=1}^{N} y_n + \lambda}\right)^2$

[d] $C = \left(\dfrac{\sum_{n=1}^{N} y_n^2}{N+\lambda}\right)$

[e] none of the other choices

(*Note: All the choices hint you that a smaller λ corresponds to a bigger C.*)

$$\min_{w\in\mathbb{R}} \frac{1}{N}\sum_{n=1}^{N}(w-y_n)^2 + \frac{\lambda}{N}w^2$$

[對 w 微分，並代入 w^* 讓微分後等於 0]

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

$$\sum_{n=1}^{N}(w^*-y_n) + \lambda w^* = 0$$

$$Nw^* - \sum_{n=1}^{N} y_n + \lambda w^* = 0$$

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

[由題目中提到 $C = w^*$]

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

3. ANS [d]

Scaling can affect regularization. Consider a data set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$. Define $\Phi(\mathbf{x}) = \mathrm{V}\mathbf{x}$ where V is a diagonal matrix with the i-th diagonal component storing a *positive* value to scale the i-th feature. Now, conduct L2-regularized linear regression with the transformed data $\{(\Phi(\mathbf{x}_n), y_n)\}_{n=1}^N$.

$$\min_{\tilde{\mathbf{w}} \in \mathbb{R}^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{\mathbf{w}}^T \Phi(\mathbf{x}_n) - y_n)^2 + \frac{\lambda}{N} (\tilde{\mathbf{w}}^T \tilde{\mathbf{w}})$$

The problem is equivalent to the following regularized linear regression problem on the original data with a different regularizer.

$$\min_{\mathbf{w} \in \mathbb{R}^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - y_n)^2 + \frac{\lambda}{N} (\mathbf{w}^T \mathrm{U} \mathbf{w})$$

What is U? Choose the correct answer; explain your answer.

[a] V
[b] V^2
[c] V^{-1}
[d] $(\mathrm{V}^{-1})^2$
[e] none of the other choices

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

equivalent to

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

[題目中提到 $\Phi(x) = Vx$
且投影片提到 $h(x) = \tilde{h}(\Phi(x))$]

$$\Rightarrow \tilde{w}^T \Phi(x) = w^T x$$

$$\tilde{w}^T = w^T X (VX)^{-1}$$

$$= w^T X X^{-1} V^{-1} = w^T V^{-1}$$

$$\Rightarrow \tilde{w}^T \tilde{w} = w^T U w$$

$$\tilde{w}^T \tilde{w} = (w^T V^{-1})(w^T V^{-1})^T$$

$$= w^T V^{-1} V^{-1T} w$$

[V 為對角矩陣 $\therefore V^T = V$, $V^{-1T} = V^{-1}$]

$$= w^T V^{-1} V^{-1} w$$

$$= w^T (V^{-1})^2 w$$

$U = (V^{-1})^2$ # 選 d

4. ANS [a]

Noise might seem to be an absolute evil at first glance. Can we add noise to combat overfitting, much like how we get vaccinated? Consider a *noisy transform* $\Phi(\mathrm{x}) = \mathrm{x} + \epsilon$ that adds a noise vector $\epsilon = (\epsilon_0, \cdots, \epsilon_d)$ to x. Assume that the noise vector is generated i.i.d. from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2 I)$. Consider the following optimization problem that minimizes the expected E_{in} of the transformed data:

$$\min_{\mathbf{w}\in\mathbb{R}^{d+1}} \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\Phi(\mathrm{x}_n) - y_n)^2\right]$$

The problem is actually equivalent to L2-regularized linear regression with some λ.

$$\min_{\mathbf{w}\in\mathbb{R}^{d+1}} \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathrm{x}_n - y_n)^2 + \frac{\lambda}{N}||\mathrm{w}||_2^2$$

What is λ? Choose the correct answer; explain your answer.

[a] $N\sigma^2$
[b] $2N\sigma^2$
[c] $\frac{N}{2}\sigma^2$
[d] $\frac{N}{\sigma^2}$
[e] none of the other choices

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

$\sum_{n=1}^{N}\Phi(x_n) \Rightarrow \sum_{n=1}^{N}(x_n+\epsilon_n)$ 其中的 $x_n + \epsilon_n$ 為 $\begin{bmatrix}x_0\\ \vdots \\ x_d\end{bmatrix}_{d+1} \begin{bmatrix}\epsilon_0\\ \vdots \\ \epsilon_d\end{bmatrix}_{d+1}$

假設 $\sum_{n=1}^{N}x_n + \sum_{n=1}^{N}\epsilon_n$ 為 $X + \varepsilon$

$X \sim \underbrace{\begin{bmatrix}-x_0^T-\\ \vdots \\ -x_N^T-\end{bmatrix}}_{d+1}$ $\varepsilon \sim \underbrace{\begin{bmatrix}-\epsilon_0^T-\\ \vdots \\ -\epsilon_N^T-\end{bmatrix}}_{d+1}$

$\min_{w\in\mathbb{R}^{d+1}} \mathbb{E}\left[\frac{1}{N}\|(X+\varepsilon)w - y\|_2^2\right]$

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

$= \min_{w\in\mathbb{R}^{d+1}} \mathbb{E}\left[\frac{1}{N}\left(w^TX^TXw + w^TX^T\varepsilon w + w^T\varepsilon^TXw + w^T\varepsilon^T\varepsilon w - 2Xw^Ty - 2\varepsilon w^Ty + y^Ty\right)\right]$

[藍色部分等於 $\min_{w\in\mathbb{R}^{d+1}} \frac{1}{N}\sum_{n=1}^{N}(w^Tx_n - y_n)^2 + \frac{\lambda}{N}\|w\|_2^2$]

$= \min_{w\in\mathbb{R}^{d+1}} \mathbb{E}\left[\frac{1}{N}\|Xw-y\|_2^2\right] + \frac{1}{N}\left[\mathbb{E}(w^TX^T\varepsilon w + w^T\varepsilon^TXw + w^T\varepsilon^T\varepsilon w - 2\varepsilon w^Ty)\right]$

[所以換下來比較綠色部分]

$\Rightarrow \frac{1}{N}\left[\mathbb{E}(w^TX^T\varepsilon w) + \mathbb{E}(w^T\varepsilon^TXw) + \mathbb{E}(w^T\varepsilon^T\varepsilon w) - 2\mathbb{E}(\varepsilon w^Ty)\right]$

由題目可知 ϵ 來自 $N(0,\sigma^2 I)$

所以 $E(\epsilon_n)=0$ ⇨ $E(\epsilon)=\begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}\Big]N$

根據多維高斯分佈，Σ 為 covariance matrix

$$\Sigma \Rightarrow \begin{bmatrix} \frac{\sum(x_1-\mu_1)^2}{n} & \frac{\sum(x_1-\mu_1)(x_2-\mu_2)}{n} & \cdots & \frac{\sum(x_1-\mu_1)(x_n-\mu_n)}{n} \\ \frac{\sum(x_2-\mu_2)(x_1-\mu_1)}{n} & \frac{\sum(x_2-\mu_2)^2}{n} & \cdots & \frac{\sum(x_2-\mu_2)(x_n-\mu_n)}{n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\sum(x_n-\mu_n)(x_1-\mu_1)}{n} & \frac{\sum(x_n-\mu_n)(x_2-\mu_2)}{n} & \cdots & \frac{\sum(x_n-\mu_n)^2}{n} \end{bmatrix}$$

由於此題為 i.i.d. 且 $\mu=0$，所以 $\Sigma=\sigma^2 I$

$$\Sigma \Rightarrow \begin{bmatrix} \frac{\sum\epsilon_0^2}{n} & 0 & \cdots & 0 \\ 0 & \frac{\sum\epsilon_1^2}{n} & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \frac{\sum\epsilon_d^2}{n} \end{bmatrix}_{d+1\times d+1}$$

$$\epsilon_n\cdot\epsilon_n^T=\begin{bmatrix}\epsilon_0\\ \vdots\\ \epsilon_d\end{bmatrix}\begin{bmatrix}\epsilon_0 & \cdots & \epsilon_d\end{bmatrix}=\begin{bmatrix}\epsilon_0\epsilon_0^T & \epsilon_0\epsilon_1^T & \cdots & \epsilon_0\epsilon_d^T \\ \epsilon_1\epsilon_0^T & \epsilon_1\epsilon_1^T & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \epsilon_d\epsilon_0^T & \cdots & \cdots & \epsilon_d\epsilon_d^T\end{bmatrix}_{d+1\times d+1}$$

$$E(\epsilon_n\cdot\epsilon_n^T)=\begin{bmatrix}E(\epsilon_0\epsilon_0^T) & E(\epsilon_0\epsilon_1^T) & \cdots & E(\epsilon_0\epsilon_d^T) \\ E(\epsilon_1\epsilon_0^T) & E(\epsilon_1\epsilon_1^T) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ E(\epsilon_d\epsilon_0^T) & \cdots & \cdots & E(\epsilon_d\epsilon_d^T)\end{bmatrix}_{d+1\times d+1}$$

藍色部份都是 0，以 $E(\epsilon_0\epsilon_1)$ 說明

$E(\epsilon_0\epsilon_1)$ ∵ i.i.d. ∴ $E(\epsilon_0\epsilon_1)=E(\epsilon_0)E(\epsilon_1)$

$=0\times 0=0$

紅色部份不為 0，∵ 自己與自己不獨立

且 $Var(\epsilon_n)=E(\epsilon_n\cdot\epsilon_n^T)-\underbrace{(E(\epsilon_n))^2}_{0}$

$=E(\epsilon_n\cdot\epsilon_n^T)=\sigma^2 I$

$$\underset{d+1}{\varepsilon^T\varepsilon}=\underbrace{\begin{bmatrix}| & | & & | \\ \epsilon_1 & \epsilon_2 & \cdots & \epsilon_N \\ | & | & & |\end{bmatrix}}_{N}\ N\ \underbrace{\begin{bmatrix}— & \epsilon_1 & — \\ — & \epsilon_2 & — \\ & \vdots & \\ — & \epsilon_N & —\end{bmatrix}}_{d+1}$$

根據矩陣計算

$$\varepsilon^T\varepsilon=\begin{bmatrix}| \\ \epsilon_1 \\ |\end{bmatrix}\begin{bmatrix}— & \epsilon_1 & —\end{bmatrix}+\cdots+\begin{bmatrix}| \\ \epsilon_N \\ |\end{bmatrix}\begin{bmatrix}— & \epsilon_N & —\end{bmatrix}$$

$$=\sum_{n=1}^{N}\epsilon_n\cdot\epsilon_n^T$$

⇨ $E(\varepsilon^T\varepsilon)=E\left(\sum_{n=1}^{N}\epsilon_n\epsilon_n^T\right)=\sum_{n=1}^{N}E(\epsilon_n\epsilon_n^T)$

$=N\sigma^2 I$

以上述可解

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

0 by matrix　　0 by matrix　　0 by matrix

$$= \frac{1}{N}\left[\mathbb{E}(w^T \varepsilon^T \varepsilon w)\right] = \frac{1}{N} w^T \mathbb{E}(\varepsilon^T \varepsilon) w$$

$$= \frac{1}{N} w^T (N\sigma^2 I) w$$

$$= \frac{1}{N} N\sigma^2 w^T w$$

$$\frac{1}{N} N\sigma^2 w^T w = \frac{\lambda}{N} \|w\|_2^2 = \frac{\lambda}{N} w^T w$$

得 $\lambda = N\sigma^2$ # 證Q

5. ANS [d]

Additive smoothing (`https://en.wikipedia.org/wiki/Additive_smoothing`) is a simple yet useful technique in estimating discrete probabilities. Consider the technique for estimating the head probability of a coin. Let $y_1, y_2, \ldots, y_N$ denotes the flip results from a coin, with $y_n = 1$ represents a head and $y_n = 0$ represents a tail. Additive smoothing adds (αk) "virtual flips", with α of them being head and the other $(k-1)\alpha$ being tail. Then, the head probability is estimated by

$$\frac{(\sum_{n=1}^{N} y_n) + \alpha}{N + \alpha k}$$

The estimate can be viewed as the optimal solution of

$$\min_{y \in \mathbb{R}} \frac{1}{N} \sum_{n=1}^{N} (y - y_n)^2 + \frac{\alpha k}{N} \Omega(y),$$

where $\Omega(y)$ is a "regularizer" to this estimation problem. What is $\Omega(y)$? Choose the correct answer; explain your answer.

[a] $(y+k)^2$
[b] $(y+\frac{1}{k})^2$
[c] $(y-k)^2$
[d] $(y-\frac{1}{k})^2$
[e] none of the other choices

目標要找出 $\Omega(y)$

所以透過微分 = 0 去找

$$\min_{y \in \mathbb{R}} \frac{1}{N} \sum_{n=1}^{N} (y-y_n)^2 + \frac{\alpha k}{N} \Omega(y)$$

對 y 微分

$$\frac{d \frac{1}{N} \sum_{n=1}^{N} (y^T y - 2y^T y_n + y_n^2)}{dy} + \frac{d \frac{\alpha k}{N} \Omega(y)}{dy} = 0$$

$$\frac{1}{N} \sum_{n=1}^{N} (2y - 2y_n) + \frac{\alpha k}{N} \Omega(y)' = 0$$

$$\left[\ y=\frac{\sum_{n=1}^{N} y_n+\alpha}{N+\alpha k}\ ,\ \sum_{n=1}^{N} y_n=y(N+\alpha k)-\alpha\ \right]$$

$$\frac{2}{N}\sum_{n=1}^{N}(y-y_n)+\frac{\alpha k}{N}\Omega(y)'=0$$

$$\Omega(y)'=\frac{-2\sum_{n=1}^{N}(y-y_n)}{\alpha k}=\frac{-2\left(Ny-(y(N+\alpha k)-\alpha)\right)}{\alpha k}$$

$$=\frac{-2(Ny-Ny-\alpha ky+\alpha)}{\alpha k}$$

$$=\frac{2\alpha ky-2\alpha}{\alpha k}=\frac{2ky-2}{k}=2y-\frac{2}{k}$$

對 y 積分

$$\Omega(y)=\frac{2}{2}y^2-\frac{2}{k}y+C=y^2-\frac{2}{k}y+C\ ,\ C\text{ 常數}$$

$$\Rightarrow\left(y-\frac{1}{k}\right)^2$$ ※ 題d

6. ANS [b]

When performing L2-regularization on any twice-differentiable $E_{\text{in}}(\mathbf{w})$ for some linear model of interest, the optimization problem can be written as:

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^{d+1}} \quad & E_{\text{aug}}(\mathbf{w}) \\ \text{subject to} \quad & E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}||\mathbf{w}||_2^2 \end{aligned}$$

Suppose $\mathbf{w}^*$ is the minimizer of $E_{\text{in}}(\mathbf{w})$. That is, $\nabla E_{\text{in}}(\mathbf{w}^*) = \mathbf{0}$. Take the second-order Taylor's expansion of $E_{\text{in}}(\mathbf{w})$ around $\mathbf{w}^*$, we can approximate $E_{\text{in}}(\mathbf{w})$ by

$$\tilde{E}_{\text{in}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}^*) + \underbrace{(\mathbf{w}-\mathbf{w}^*)^T \nabla E_{\text{in}}(\mathbf{w}^*)}_{0} + \frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^T \ \mathrm{H} \ (\mathbf{w}-\mathbf{w}^*)$$

where $\mathrm{H} \in \mathbb{R}^{(d+1)\times(d+1)}$ is some Hessian matrix. Then, $E_{\text{aug}}(\mathbf{w})$ can be approximated by

$$\tilde{E}_{\text{aug}}(\mathbf{w}) = \tilde{E}_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\|\mathbf{w}\|^2.$$

Which of the following is the minimizer of $\tilde{E}_{\text{aug}}(\mathbf{w})$? Choose the correct answer; explain your answer.

[a] $(\mathrm{H}+\lambda\mathrm{I})^{-1}\mathrm{H}\mathbf{w}^*$
[b] $(\mathrm{H}+\frac{2\lambda}{N}\mathrm{I})^{-1}\mathrm{H}\mathbf{w}^*$
[c] $(\mathrm{H}^T\mathrm{H}+\lambda\mathrm{I})^{-1}\mathrm{H}^T\mathrm{H}\mathbf{w}^*$
[d] $(\mathrm{H}+\frac{\lambda}{2N}\mathrm{I})^{-1}\mathrm{H}\mathbf{w}^*$
[e] none of the other choices.

(*Note: The result demonstrates the difference between the minimizers of* E_{in} *and* $\tilde{E}_{\text{aug}}$)

$$\tilde{E}_{in}(w) = E_{in}(w^*) + 0 + \frac{1}{2}(w-w^*)^T H (w-w^*)$$

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

$$= E_{in}(w^*) + \frac{1}{2}(w-w^*)^T H (w-w^*) + \frac{\lambda}{N} w^T w$$

常數

對 w 微分

$$\nabla \tilde{E}_{aug}(w) = \frac{\partial \frac{1}{2}(w-w^*)^T H (w-w^*)}{\partial (w-w^*)} \cdot \frac{\partial (w-w^*)}{\partial w} + \frac{\partial \frac{\lambda}{N} w^T w}{\partial w}$$

$$= \frac{2}{2} H (w-w^*) + \frac{2\lambda}{N} w = 0$$

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

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

選 b

Validation

7. ANS [a]

Consider a binary classification algorithm $\mathcal{A}_{\text{minority}}$, which returns a constant classifier that always predicts the minority class (i.e., the class with fewer instances in the data set that it sees). As you can imagine, the returned classifier is the worst-E_{in} one among all constant classifiers. For a binary classification data set with N positive examples and N negative examples, what is $E_{\text{loocv}}(\mathcal{A}_{\text{minority}})$? Choose the correct answer; explain your answer.

[a] 0
[b] $1/N$
[c] $1/2$
[d] $(N-1)/N$
[e] 1

$$E_{\text{loocv}}(\mathcal{H}\cdot\mathcal{A}) = \frac{1}{N}\sum_{n=1}^{N} e_n = \frac{1}{N}\sum_{n=1}^{N} \text{err}\left(g_n^-(x_n), y_n\right)$$

當取的點	positive data	negative data	$g_n^-(x_n)$
positive	$N-1$	N	$\mathcal{A}_{\min}$ → positive
negative	N	$N-1$	$\mathcal{A}_{\min}$ → negative

↳ 所以不管怎麼取都是對的

$$\text{err}\left(g_n^-(x_n), y_n\right)$$

$$\begin{cases} \text{err}(\text{pos}, \text{pos}) = 0 \\ \text{err}(\text{neg}, \text{neg}) = 0 \end{cases} \Rightarrow E_{\text{loocv}}(\mathcal{H}\cdot\mathcal{A}) = 0 \quad \#$$

選 a

8. ANS [c]

You are given three data points: $(x_1, y_1) = (2, 0), (x_2, y_2) = (\rho, 2), (x_3, y_3) = (-2, 0)$ with $\rho \geq 0$, and a choice between two models: constant (all hypotheses are of the form $h(x) = w_0$) and linear (all hypotheses are of the form $h(x) = w_0 + w_1 x$). For which value of ρ would the two models be tied using leave-one-out cross-validation with the squared error measure? Choose the closest answer; explain your answer.

[a] 6.5
[b] 7.5
[c] 8.5
[d] 9.5
[e] 10.5

$\mathcal{D}$: $(x_1, y_1) = (2, 0)$
$(x_2, y_2) = (\rho, 2)$, $\rho \geq 0$
$(x_3, y_3) = (-2, 0)$

Models: constant $\Rightarrow h(x) = w_0$
linear $\Rightarrow h(x) = w_0 + w_1 x$

leave-one-out cross-validation with squared error measure

$$E_{loocv}(\mathcal{H}, \mathcal{A}) = \frac{1}{N}\sum_{n=1}^{N} e_n = \frac{1}{N}\sum_{n=1}^{N} err(g_n^-(x_n), y_n)$$
$$= \frac{1}{N}\sum_{n=1}^{N} (g_n^-(x_n) - y_n)^2$$

constant

取 x_1 時, $w_0 = \frac{2+0}{2} = 1$
取 x_2 時, $w_0 = 0$
取 x_3 時, $w_0 = \frac{2+0}{2} = 1$

$$\Rightarrow E_{loocv}(\mathcal{H}, \mathcal{A}_{constant}) = \frac{1}{3}\left[(1-0)^2 + (0-2)^2 + (1-0)^2\right]$$
$$= 2$$

linear

取 x_1 時, $w_0 + w_1 x = y$

$$\begin{cases} w_0 + \rho w_1 = 2 \\ w_0 - 2w_1 = 0 \end{cases} \Rightarrow w_1 = \frac{2}{\rho+2},\ w_0 = \frac{4}{\rho+2}$$
$$\frac{4}{\rho+2} + \frac{2}{\rho+2}x = y$$

取 x_2 時, $\begin{cases} w_0 + 2w_1 = 0 \\ w_0 - 2w_1 = 0 \end{cases} \Rightarrow 4w_1 = 0,\ w_1 = 0 = w_0$

取 x_3 時, $\begin{cases} w_0 + 2w_1 = 0 \\ w_0 + \rho w_1 = 2 \end{cases} \Rightarrow w_1 = \frac{2}{\rho-2},\ w_0 = \frac{-4}{\rho-2}$
$$\frac{-4}{\rho-2} + \frac{2}{\rho+2}x = y$$

$$E_{loocv}(f_1, A_{linear})$$

$$= \frac{1}{3}\left[\left(\frac{4}{\rho+2} + \frac{4}{\rho+2} - 0\right)^2 + (0-2)^2 + \left(\frac{-4}{\rho-2} - \frac{4}{\rho-2}\right)^2\right]$$

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

$$2(\rho+2)^2(\rho-2)^2 = 64(\rho+2)^2 + 64(\rho-2)^2$$

$$= 64(\rho^2+4\rho+4+\rho^2-4\rho+4)$$

$$= 64(2\rho^2+8)$$

$$2(\rho^2-4)^2 = 64(2\rho^2+8)$$

$$\div 32$$

$$\rho^4 - 8\rho^2 + 16 = 64\rho^2 + 256$$

$$\rho^4 - 72\rho^2 - 240 = 0$$

令 $u = \rho^2$, $u^2 = \rho^4$

$$u^2 - 72u - 240 = 0$$

$$u = 4(9+4\sqrt{6}) \text{ or } 4(9-4\sqrt{6})$$

$$\rho = 2\sqrt{9+4\sqrt{6}} \text{ or } -2\sqrt{9+4\sqrt{6}}$$

$\rho \geq 0$

$$\rho = 2\sqrt{9+4\sqrt{6}} \fallingdotseq 8.6913$$

接近 8.5 ∴ 選 C

9. ANS [b]

For N labels $y_1, y_2, \ldots, y_N$ generated i.i.d. from some distribution of mean 0 and variance σ^2. Partition the first $N-K$ labels to be the training data, and the last K labels to be the validation data. If we "estimate" the mean by the "target" of 0, the expected validation error

$$\mathbb{E}\left(\frac{1}{K}\sum_{n=N-K+1}^{N}(y_n-0)^2\right),$$

where the expectation is taken on the process of generating N labels, is simply σ^2 by the independence of the y_n's. Now, assume that we take the average on the first $N-K$ examples to estimate the mean instead. That is,

$$\bar{y}=\frac{1}{N-K}\sum_{n=1}^{N-K}y_n$$

What is the expected validation error

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

Choose the correct answer; explain your answer.

[a] σ^2

[b] $\sigma^2+\frac{1}{N-K}\sigma^2$

[c] $\frac{1}{K(N-K)}\sigma^2$

[d] $\sigma^2+\frac{1}{K(N-K)}\sigma^2$

[e] none of the other choices

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

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

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

[由題目可知 $\mathbb{E}\left(\frac{1}{K}\sum_{n=N-K+1}^{N}y_n^2\right)=\sigma^2$]

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

由於 $y_1, y_2, \cdots, y_N$ 為 i.i.d. $\sim N(0, \sigma^2)$

根據文的標準偏差公式

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} \Rightarrow \sigma_{\bar{x}}^2 = \frac{\sigma^2}{n}$$

可知 $\frac{1}{K}\sum_{n=N-K+1}^{N} y_n$ 為 i.i.d. $\sim N(0, \frac{\sigma^2}{K})$

且 $\bar{y} = \frac{1}{N-K}\sum_{n=1}^{N-K} y_n$ 為 i.i.d. $\sim N(0, \frac{\sigma^2}{N-K})$

$$= \sigma^2 - 2\,E(K\cdot\bar{y})\cdot E\left(\frac{1}{K}\sum_{n=N-K+1}^{N} y_n\right) + E(\bar{y}^2) \quad (\because \text{i.i.d.})$$

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

$$= \sigma^2 - 2K \times 0 \times 0 + \frac{\sigma^2}{N-K}$$

$$= \sigma^2 + \frac{\sigma^2}{N-K}$$ 選 b

Learning Principles

10. ANS [a]

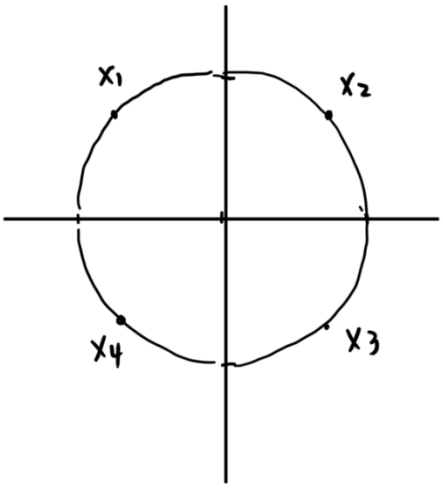
In Lecture 9, we talked about the probability to fit data perfectly when the labels are random. For instance, page 5 of Lecture 9 shows that the probability of fitting the data perfectly with positive rays is $\frac{N+1}{2^N}$. Consider 4 different points on the unit circle in $\mathbb{R}^2$ as input vectors $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_4$, and a 2D perceptron model that minimizes $E_{\text{in}}(\mathbf{w})$ (0/1 error) to the lowest possible value. One way to measure the power of the model is to consider four random labels y_1, y_2, y_3, y_4, each in ± 1 and generated by i.i.d. fair coin flips, and then compute

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

For a perfect fitting, $\min E_{\text{in}}(\mathbf{w})$ will be 0; for a less perfect fitting (when the data is not linearly separable), $\min E_{\text{in}}(\mathbf{w})$ will be some non-zero value. The expectation above averages over all 16 possible combinations of y_1, y_2, y_3, y_4. What is the value of the expectation? Choose the correct answer; explain your answer.

[a] 1/32
[b] 2/32
[c] 3/32
[d] 5/32
[e] 8/32

(*Note: It can be shown that* 1 *minus twice the expected value above is the same as the so-called empirical Rademacher complexity of 2D perceptrons. Rademacher complexity, similar to the VC dimension, is another tool to measure the complexity of a hypothesis set. If a hypothesis set shatters some data points, zero* E_{in} *can always be achieved and thus Rademacher complexity is* 1*; if a hypothesis set cannot shatter some data points, Rademacher complexity provides a soft measure of how "perfect" the hypothesis set is.*)



x_1, x_2, x_3, x_4 落在 unit circle 上4点

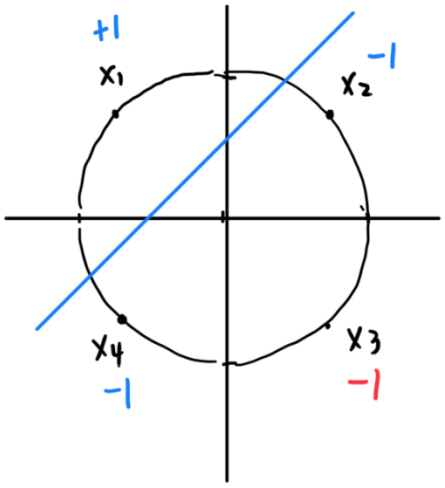
此時無法 shatter

無法得到下面2種情形

x_1	x_2	x_3	x_4
+1	-1	+1	-1
-1	+1	-1	-1

且每種情形中都至少會有一個點被分類錯誤

EXAMPLE



	x_1	x_2	x_3	x_4
LABEL	+1	-1	+1	-1
PREDIT	+1	-1	-1	-1

PAIX $E_{in} = \frac{1+0\times3}{4} = \frac{1}{4}$

共有 2^4 種情形

$$\text{正}_{y_1,y_2,y_3,y_4}\left(\min_{w\in R^{21}} E_{in}(w)\right) = \frac{1}{2^4}\times\left(\frac{1}{4}+\frac{1}{4}\right)$$

$$= \frac{1}{2^5} = \frac{1}{32} \text{ ** 選 a}$$

11. ANS [b]

Consider a binary classifier g such that

$$\begin{aligned} P(g(\mathrm{x}) = -1 | y = +1) &= \epsilon_+ \\ P(g(\mathrm{x}) = +1 | y = -1) &= \epsilon_-. \end{aligned}$$

When deploying the classifier to a test distribution of $P(y = +1) = P(y = -1) = 1/2$, we get $E_{\text{out}}(g) = \frac{1}{2}\epsilon_+ + \frac{1}{2}\epsilon_-$. Now, if we deploy the classifier to another test distribution $P(y = +1) = p$ instead of $1/2$, the $E_{\text{out}}(g)$ under this test distribution will then change to a different value. Note that under this test distribution, a constant classifier g_- that always predicts -1 will suffer from $E_{\text{out}}(g_-) = p$ as it errors on all the positive examples. At what p, if its value is between $[0, 1]$, will our binary classifier g be as good as (or as bad as) the constant classifier g_- in terms of E_{out}? Choose the correct answer; explain your answer.

[a] $p = \frac{\epsilon_+}{\epsilon_- - \epsilon_+ + 1}$

[b] $p = \frac{\epsilon_-}{\epsilon_- - \epsilon_+ + 1}$

[c] $p = \frac{1-\epsilon_+}{\epsilon_- - \epsilon_+ + 1}$

[d] $p = \frac{1-\epsilon_-}{\epsilon_- - \epsilon_+ + 1}$

[e] none of the other choices

$g_- \Rightarrow -1$

$E_{out}(g_-) = P = P\epsilon_+ + (1-P)\epsilon_-$

$P = P\epsilon_+ + \epsilon_- - P\epsilon_-$

$P = P(\epsilon_+ - \epsilon_-) + \epsilon_-$

$P(1 - \epsilon_+ + \epsilon_-) = \epsilon_-$

$P = \frac{\epsilon_-}{1 - \epsilon_+ + \epsilon_-}$ ※選 b

Experiments with Regularized Logistic Regression

Consider L2-regularized logistic regression with third-order polynomial transformation.

$$\mathbf{w}_\lambda = \operatorname*{argmin}_{\mathbf{w}} \frac{\lambda}{N}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ln(1+\exp(-y_n \mathbf{w}^T \mathbf{\Phi}_3(\mathbf{x}_n))),$$

Here $\mathbf{\Phi}_3$ is the third-order polynomial transformation introduced in Lecture 6 (with $Q = 3$), defined as

$$\mathbf{\Phi}_3(\mathbf{x}) = (1, x_1, x_2, \ldots, x_d, x_1^2, x_1x_2, \ldots, x_1x_d, x_2^2, x_2x_3, \ldots, x_2x_d, \ldots, x_d^2, x_1^3, x_1^2x_2, \ldots, x_d^3)$$

Next, we will take the following file as our training data set $\mathcal{D}$:

`http://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_train.dat`

and the following file as our test data set for evaluating E_{out}:

`http://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_test.dat`

We call the algorithm for solving the problem above as $\mathcal{A}_\lambda$. The problem guides you to use LIBLINEAR (`https://www.csie.ntu.edu.tw/~cjlin/liblinear/`), a machine learning packaged developed in our university, to solve this problem. In addition to using the default options, what you need to do when running LIBLINEAR are

- set option `-s 0`, which corresponds to solving regularized logistic regression
- set option `-c C`, with a parameter value of `C` calculated from the λ that you want to use; read `README` of the software package to figure out how `C` and your λ should relate to each other
- set option `-e 0.000001`, which corresponds to getting a solution that is really really close to the optimal solution

LIBLINEAR can be called from the command line or from major programming languages like python. If you run LIBLINEAR in the command line, please include screenshots of the commands/results; if you run LIBLINEAR from any programming language, please include screenshots of your code.

We will consider the data set as a *binary classification problem* and take the "regression for classification" approach with regularized logistic regression ~~(see Page 6 of Lecture 10)~~. So please evaluate all errors below with the 0/1 error.

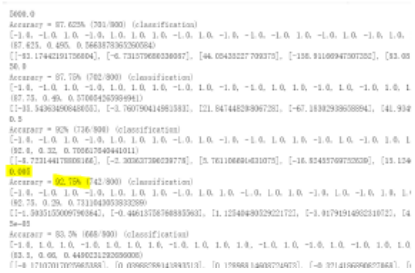
12. ANS [d]

Select the best λ^* *in a cheating manner* as

$$\operatorname*{argmin}_{\log_{10}\lambda\in\{-4,-2,0,2,4\}} E_{\text{out}}(\mathbf{w}_\lambda).$$

Break the tie, if any, by selecting the largest λ. What is $\log_{10}(\lambda^*)$? Choose the closest answer; provide your command/code.

[a] -4
[b] -2
[c] 0
[d] 2
[e] 4



colab：https://colab.research.google.com/drive/1vyjBAJy9_L-ExNjQufvVho9Kj9Bi3wEf?usp=sharing

根據 liblinear 的文件提到

A.3 L2-regularized Logistic Regression

L2-regularized LR solves the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\log(1+e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i}). \tag{5}$$

Its dual form is:

$$\begin{aligned}\min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_{i:\alpha_i>0}\alpha_i\log\alpha_i + \sum_{i:\alpha_i<C}(C-\alpha_i)\log(C-\alpha_i) - \sum_{i=1}^{l}C\log C \\ \text{subject to} \quad & 0\le\alpha_i\le C,\ i=1,\dots,l.\end{aligned} \tag{6}$$

對應到此題題目

$$\mathrm{w}_\lambda = \operatorname*{argmin}_{\mathbf{w}} \frac{\lambda}{N}\|\mathrm{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N}\ln(1+\exp(-y_n\mathrm{w}^T\boldsymbol{\Phi}_3(\mathrm{x}_n))),$$

可知$C = \frac{1}{N}$ ，$\frac{\lambda}{N} = \frac{1}{2}$

所以$C = \frac{1}{2\lambda}$

根據題目先計算出 lambda：

$$\operatorname*{argmin}_{\log_{10}\lambda\in\{-4,-2,0,2,4\}} E_{\text{out}}(\mathrm{w}_\lambda).$$

$log_{10}\lambda = -4$ ，$\lambda = 0.0001$

$log_{10}\lambda = -2$ ，$\lambda = 0.01$

$log_{10}\lambda = 0$ ，$\lambda = 1$

$log_{10}\lambda = 2$ ，$\lambda = 100$

$log_{10}\lambda = 4$ ，$\lambda = 10000$

程式大綱：

1. pip install -U liblinear-official 與 import 套件
2. Get Data
3. Third-order Polynomial Transformation
4. C = 1/(2*lambda)
5. LIBLINEAR train and predict

```
pip install -U liblinear-official
```

```
Collecting liblinear-official
  Downloading liblinear-official-2.43.0.tar.gz (47 kB)
     |████████████████████████████████| 47 kB 1.8 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from li
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: liblinear-official
  Building wheel for liblinear-official (setup.py) ... done
  Created wheel for liblinear-official: filename=liblinear_official-2.43.0-cp37-cp37m-l
  Stored in directory: /root/.cache/pip/wheels/83/6c/0d/06763f665bbc6f69fdeb3ff868df47b
Successfully built liblinear-official
Installing collected packages: liblinear-official
Successfully installed liblinear-official-2.43.0
```

```
import urllib.request
import numpy as np
import math
from sklearn.preprocessing import PolynomialFeatures
from liblinear.liblinearutil import *
```

```
def getData(url):
    #get data
    contents = urllib.request.urlopen(url).read()
    text = str(contents,'utf-8')
    # data preprocessing
    sptext = text.split('\n')

    X = []
    Y = []

    for i in range(len(sptext)-1):
        data = sptext[i].split('  ')
        # print(data)
        label = data[-1]
        # print(type(data))
        data.pop()
        # print(len(data))
        X += [data]
        Y += [label]

    X = np.asarray(X, dtype = float)
    Y = np.asarray(Y, dtype = float)
    # print(X)
    # print(Y)
    return X, Y
```

```python
if __name__ == '__main__':
    # Get Data
    train_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_train.dat"
    test_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_test.dat"
    train_X ,train_Y = getData(train_data)
    test_X, test_Y = getData(test_data)
    # print(train_X)

    # Third-order Polynomial Transformation
    poly = PolynomialFeatures(3)
    ptrain_X = poly.fit_transform(train_X)
    ptest_X = poly.fit_transform(test_X)


    # C = 1/(2*lambda)
    lamb = [0.0001, 0.01, 1, 100, 10000]
    C = []
    for l in lamb:
        C.append(str(1/(2*l)))
    # print(C)
    # LIBLINEAR train and predict
    for c in C:
        print(c)
        prob = problem(train_Y, ptrain_X)
        param = parameter('-s 0 -c {} -e 0.000001'.format(c))
        m = train(prob, param)
        p_label, p_acc, p_val = predict(test_Y, ptest_X, m)
        print(p_label)
        print(p_acc)
        print(p_val)
```

利用 `PolynomialFeatures` 來產生 third-order polynomial transformation。

將先算好的 lambda 值給 lamb list，利用 for 迴圈算出 C。

用 for 迴圈去跑每一個 C 當作-c 的參數值進行訓練，訓練好的 model 為 m。

此題要求選最好的 Eout 所以是利用 test data 進行 predict，程式執行結果如下圖。

```
5000.0
Accuracy = 87.625% (701/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.
(87.625, 0.495, 0.5663878365260584)
[[-83.17442191756804], [-6.731579680336087], [44.05435227709375], [-15
50.0
Accuracy = 87.75% (702/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.(
(87.75, 0.49, 0.570054265984941)
[[-35.54363490848055], [-3.760790414981583], [21.84744820806728], [-67
0.5
Accuracy = 92% (736/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.(
(92.0, 0.32, 0.705617640441011)
[[-8.723144178808166], [-2.303637390239778], [5.761106691631075], [-16
0.005
Accuracy = 92.75% (742/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0,
(92.75, 0.29, 0.7311043053833289)
[[-1.5035155009790364], [-0.4461375876088563], [1.1254048052922172],
5e-05
Accuracy = 83.5% (668/800) (classification)
[-1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0,
(83.5, 0.66, 0.4490231292686008)
[[-0.17107017025985388], [0.03988289141893513], [0.12898814608724973],
```

可見當 C=0.005 也就是$\lambda = 100$，$log_{10}\lambda = 2$的時候，accuracy=92.75%最高，所以答案選 d。

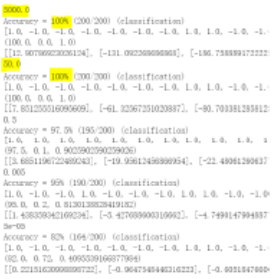
13. ANS [b]

Select the best λ^* as

$$\underset{\log_{10}\lambda\in\{-4,-2,0,2,4\}}{\operatorname{argmin}} E_{\text{in}}(\mathbf{w}_\lambda).$$

Break the tie, if any, by selecting the largest λ. What is $\log_{10}(\lambda^*)$? Choose the closest answer; provide your command/code.

[a] -4
[b] -2
[c] 0
[d] 2
[e] 4



colab：
https://colab.research.google.com/drive/1KyxprjoacnPvHfdZZRwQpcmJQk3r3dzg?usp=sharing

程式大綱：

1. pip install -U liblinear-official 與 import 套件
2. Get Data
3. Third-order Polynomial Transformation
4. C = 1/(2*lambda)
5. LIBLINEAR train and predict

(此題與上一題相同，只差在是選 Ein 所以利用 train data 進行 predict)

```
pip  install  -U  liblinear-official
```

```
Collecting liblinear-official
  Downloading liblinear-official-2.43.0.tar.gz (47 kB)
     |████████████████████████████████████████| 47 kB 1.8 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from li
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: liblinear-official
  Building wheel for liblinear-official (setup.py) ... done
  Created wheel for liblinear-official: filename=liblinear_official-2.43.0-cp37-cp37m-l
  Stored in directory: /root/.cache/pip/wheels/83/6c/0d/06763f665bbc6f69fdeb3ff868df47b
Successfully built liblinear-official
Installing collected packages: liblinear-official
Successfully installed liblinear-official-2.43.0
```

```
import urllib.request
import numpy as np
import math
from sklearn.preprocessing import PolynomialFeatures
from liblinear.liblinearutil import *
```

```
def getData(url):
    #get data
    contents = urllib.request.urlopen(url).read()
    text = str(contents,'utf-8')
    # data preprocessing
    sptext = text.split('\n')

    X = []
    Y = []

    for i in range(len(sptext)-1):
        data = sptext[i].split(' ')
        # print(data)
        label = data[-1]
        # print(type(data))
        data.pop()
        # print(len(data))
        X += [data]
        Y += [label]

    X = np.asarray(X, dtype = float)
    Y = np.asarray(Y, dtype = float)
    # print(X)
    # print(Y)
    return X, Y
```

```
if __name__ == '__main__':
    # Get Data
    train_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_train.dat"
    test_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_test.dat"
    train_X ,train_Y = getData(train_data)
    test_X, test_Y = getData(test_data)
    # print(train_X)

    # Third-order Polynomial Transformation
    poly = PolynomialFeatures(3)
    ptrain_X = poly.fit_transform(train_X)
    ptest_X = poly.fit_transform(test_X)

    # C = 1/(2*lambda)
    lamb = [0.0001, 0.01, 1, 100, 10000]
    C = []
    for l in lamb:
        C.append(str(1/(2*l)))
    # print(C)
    # LIBLINEAR train and predict
    for c in C:
        print(c)
        prob = problem(train_Y, ptrain_X)
        param = parameter('-s 0 -c {} -e 0.000001'.format(c))
        m = train(prob, param)
        p_label, p_acc, p_val = predict(train_Y, ptrain_X, m)
        print(p_label)
        print(p_acc)
        print(p_val)
```

利用 `PolynomialFeatures` 來產生 third-order polynomial transformation。

將先算好的 lambda 值給 lamb list，利用 for 迴圈算出 C。

用 for 迴圈去跑每一個 C 當作-c 的參數值進行訓練，訓練好的 model 為 m。

此題要求選最好的 Ein 所以是利用 train data 進行 predict，程式執行結果如下圖。

```
5000.0
Accuracy = 100% (200/200) (classification)
[1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -
(100.0, 0.0, 1.0)
[[12.90786923026124], [-131.092268686868], [-186.758889172
50.0
Accuracy = 100% (200/200) (classification)
[1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -
(100.0, 0.0, 1.0)
[[7.851255516095609], [-61.32567251020887], [-80.703381285
0.5
Accuracy = 97.5% (195/200) (classification)
[1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, -1.0,
(97.5, 0.1, 0.9025902590259026)
[[3.6851196722489243], [-19.95612456866954], [-22.480612606
0.005
Accuracy = 95% (190/200) (classification)
[1.0, -1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -
(95.0, 0.2, 0.8130138828419182)
[[1.438359342169234], [-5.427688600316662], [-4.74901479048
5e-05
Accuracy = 82% (164/200) (classification)
[1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -
(82.0, 0.72, 0.4095539166877984)
[[0.22151630998898722], [-0.9647548446316223], [-0.6051847
```

可見當 C=5000 也就是$\lambda = 0.0001$，$log_{10}\lambda = -4$的時候，accuracy=100%，且當 C=50 也就是$\lambda = 0.01$，$log_{10}\lambda = -2$的時候，accuracy=100%，題目中說選最大的λ，所以答案選 b。

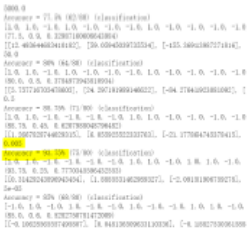
14. ANS [c]

Now split the given training examples in $\mathcal{D}$ to two sets: the first 120 examples as $\mathcal{D}_{\text{train}}$ and 80 as $\mathcal{D}_{\text{val}}$. (*Ideally, you should randomly do the* 120/80 *split. Because the given examples are already randomly permuted, however, we would use a fixed split for the purpose of this problem*). Run $\mathcal{A}_\lambda$ on *only* $\mathcal{D}_{\text{train}}$ to get w_λ^- (the weight vector within the g^- returned), and validate w_λ^- with $\mathcal{D}_{\text{val}}$ to get $E_{\text{val}}(\mathrm{w}_\lambda^-)$. Select the best λ^* as

$$\underset{\log_{10}\lambda\in\{-4,-2,0,2,4\}}{\operatorname{argmin}} E_{\text{val}}(\mathrm{w}_\lambda^-).$$

Break the tie, if any, by selecting the largest λ. Then, estimate $E_{\text{out}}(\mathrm{w}_{\lambda^*}^-)$ with the test set. What is the value of $E_{\text{out}}(\mathrm{w}_{\lambda^*}^-)$? Choose the closest answer; provide your command/code.

[a] 0.081
[b] 0.078
[c] 0.075
[d] 0.072
[e] 0.069



```
Accuracy = 92.5% (740/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.
(92.5, 0.3, 0.722637123551138)
[[-1.3211029176706193], [-0.461261919957713], [
Eout: 0.075
```

colab：
https://colab.research.google.com/drive/1n2usDlcDbVK9L4RYSoZw2PKfGCgb9tAl?usp=sharing

程式大綱：

1. pip install -U liblinear-official 與 import 套件
2. Get Data 並且要切 120 當 training data，80 當 validation data。
3. Third-order Polynomial Transformation
4. C = 1/(2*lambda)
5. LIBLINEAR train and predict (此題是選 Eval 所以利用 validation data 進行 predict)
6. Use the best lambda to estimate Eout (Eout 所以用 test data predict)
7. Calculate Eout

```
pip install -U liblinear-official
```

```
Collecting liblinear-official
  Downloading liblinear-official-2.43.0.tar.gz (47 kB)
     |████████████████████████████████| 47 kB 1.8 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from li
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: liblinear-official
  Building wheel for liblinear-official (setup.py) ... done
  Created wheel for liblinear-official: filename=liblinear_official-2.43.0-cp37-cp37m-l
  Stored in directory: /root/.cache/pip/wheels/83/6c/0d/06763f665bbc6f69fdeb3ff868df47b
Successfully built liblinear-official
Installing collected packages: liblinear-official
Successfully installed liblinear-official-2.43.0
```

```
import urllib.request
import numpy as np
import math
from sklearn.preprocessing import PolynomialFeatures
from liblinear.liblinearutil import *
```

```
def getData(url):
    #get data
    contents = urllib.request.urlopen(url).read()
    text = str(contents,'utf-8')
    # data preprocessing
    sptext = text.split('\n')

    X = []
    Y = []

    for i in range(len(sptext)-1):
        data = sptext[i].split('  ')
        # print(data)
        label = data[-1]
        # print(type(data))
        data.pop()
        # print(len(data))
        X += [data]
        Y += [label]

    X = np.asarray(X, dtype = float)
    Y = np.asarray(Y, dtype = float)
    # print(X)
    # print(Y)
    return X, Y
```

```python
if __name__ == '__main__':
    # Get Data
    train_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_train.dat"
    test_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_test.dat"
    train_X ,train_Y = getData(train_data)
    val_X, val_Y = train_X[120:], train_Y[120:]
    train_X, train_Y = train_X[:120], train_Y[:120]
    test_X, test_Y = getData(test_data)
    # print(train_X)

    # Third-order Polynomial Transformation
    poly = PolynomialFeatures(3)
    ptrain_X = poly.fit_transform(train_X)
    pval_X = poly.fit_transform(val_X)
    ptest_X = poly.fit_transform(test_X)



    # C = 1/(2*lambda)
    lamb = [0.0001, 0.01, 1, 100, 10000]
    C = []
    for l in lamb:
        C.append(str(1/(2*l)))
    # print(C)
    # LIBLINEAR train and predict
    for c in C:
        print(c)
        prob = problem(train_Y, ptrain_X)
        param = parameter('-s 0 -c {} -e 0.000001'.format(c))
        m = train(prob, param)
        p_label, p_acc, p_val = predict(val_Y, pval_X, m)
        print(p_label)
        print(p_acc)
        print(p_val)
```

切 120 給 training data，8 給 validation data。

利用 `PolynomialFeatures` 來產生 third-order polynomial transformation。

將先算好的 lambda 值給 lamb list，利用 for 迴圈算出 C。

用 for 迴圈去跑每一個 C 當作-c 的參數值進行訓練，訓練好的 model 為 m。

此題要求選最好的 Eval 所以利用 validation data 進行 predict，程式執行結果如下圖。

```
5000.0
Accuracy = 77.5% (62/80) (classification)
[1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, -1.0, -
(77.5, 0.9, 0.32987160608643984)
[[12.493644683410182], [59.05945039735534], [-155.36
50.0
Accuracy = 80% (64/80) (classification)
[1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, -1.0, -
(80.0, 0.8, 0.3784973943810934)
[[5.757716705478605], [24.297191989146622], [-84.276
0.5
Accuracy = 88.75% (71/80) (classification)
[1.0, 1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -
(88.75, 0.45, 0.6207988048796482)
[[1.2667828744029315], [6.055925522333763], [-21.177
0.005
Accuracy = 93.75% (75/80) (classification)
[1.0, 1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1
(93.75, 0.25, 0.7770345596432553)
[[0.31429243896943454], [1.8885531462989327], [-2.09
5e-05
Accuracy = 85% (68/80) (classification)
[-1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, -1.0,
(85.0, 0.6, 0.5282750781472009)
[[-0.10628565557495807], [0.048136509633110336], [-0
```

可見當 C=0.005 也就是$\lambda = 100$，$log_{10}\lambda = 2$的時候，accuracy=93.75%最好。

```
# Use the best lambda to estimate Eout
c = '0.005'
prob = problem(train_Y, ptrain_X)
param = parameter('-s 0 -c {} -e 0.000001'.format(c))
m = train(prob, param) # m is a ctype pointer to a model
p_label, p_acc, p_val = predict(test_Y, ptest_X, m)
print(p_label)
print(p_acc)
print(p_val)

# Calculate Eout
EoutN = 0
for i in range(len(p_label)):
    if p_label[i] != test_Y[i]:
        EoutN += 1
# print("EoutN: ", EoutN)
Eout = EoutN/len(p_label)
print("Eout: ", Eout)
```

```
Accuracy = 92.5% (740/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0,
(92.5, 0.3, 0.722637123551138)
[[-1.3211029176706193], [-0.461261919957713], [0.937820428177678 3],
Eout: 0.075
```

最後利用$\lambda = 100$，C=0.005 與 test data 來估計 Eout 並計算出 Eout 值。

結果為 0.075 選 c。

15. ANS [d]

For the λ_* selected in the previous problem, compute w_{λ_*} by running $\mathcal{A}_{\lambda_*}$ with the full training set $\mathcal{D}$. Then, estimate $E_{\text{out}}(\mathrm{w}_{\lambda_*})$ with the test set. What is the value of $E_{\text{out}}(\mathrm{w}_{\lambda_*})$? Choose the closest answer; provide your command/code.

[a] 0.081
[b] 0.078
[c] 0.075
[d] 0.072
[e] 0.069

```
Accuracy = 92.75% (742/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1
(92.75, 0.29, 0.7311043053833289)
[[-1.5035155009790364], [-0.44613758760885563], [1.12540480529
Eout: 0.0725
```

colab：https://colab.research.google.com/drive/1ouzByxe0-xX59rJoIHwdthfpIgTSGNkG?usp=sharing

程式大綱：

1. pip install -U liblinear-official 與 import 套件
2. Get Data
3. Third-order Polynomial Transformation
4. Use the best lambda to estimate Eout (Eout 所以用 test data predict)

   C = 0.005，LIBLINEAR train and predict

   (此題延續上一題選好的最好的 lambda 並用整個 training data 來訓練)
5. Calculate Eout

```
pip install -U liblinear-official
```

```
Collecting liblinear-official
  Downloading liblinear-official-2.43.0.tar.gz (47 kB)
     |████████████████████████████████| 47 kB 1.8 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from li
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: liblinear-official
  Building wheel for liblinear-official (setup.py) ... done
  Created wheel for liblinear-official: filename=liblinear_official-2.43.0-cp37-cp37m-l
  Stored in directory: /root/.cache/pip/wheels/83/6c/0d/06763f665bbc6f69fdeb3ff868df47b
Successfully built liblinear-official
Installing collected packages: liblinear-official
Successfully installed liblinear-official-2.43.0
```

```
import urllib.request
import numpy as np
import math
from sklearn.preprocessing import PolynomialFeatures
from liblinear.liblinearutil import *
```

```
def getData(url):
    #get data
    contents = urllib.request.urlopen(url).read()
    text = str(contents,'utf-8')
    # data preprocessing
    sptext = text.split('\n')

    X = []
    Y = []

    for i in range(len(sptext)-1):
        data = sptext[i].split('  ')
        # print(data)
        label = data[-1]
        # print(type(data))
        data.pop()
        # print(len(data))
        X += [data]
        Y += [label]

    X = np.asarray(X, dtype = float)
    Y = np.asarray(Y, dtype = float)
    # print(X)
    # print(Y)
    return X, Y
```

```
if __name__ == '__main__':
    # Get Data
    train_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_train.dat"
    test_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_test.dat"
    train_X ,train_Y = getData(train_data)
    test_X, test_Y = getData(test_data)


    # Third-order Polynomial Transformation
    poly = PolynomialFeatures(3)
    ptrain_X = poly.fit_transform(train_X)
    ptest_X = poly.fit_transform(test_X)

    # Use the best lambda to estimate Eout
    c = '0.005'
    prob = problem(train_Y, ptrain_X)
    param = parameter('-s 0 -c {} -e 0.000001'.format(c))
    m = train(prob, param)
    p_label, p_acc, p_val = predict(test_Y, ptest_X, m)
    print(p_label)
    print(p_acc)
    print(p_val)

    # Calculate Eout
    EoutN = 0
    for i in range(len(p_label)):
        if p_label[i] != test_Y[i]:
            EoutN += 1
    # print("EoutN: ", EoutN)
    Eout = EoutN/len(p_label)
    print("Eout: ", Eout)
```

利用 `PolynomialFeatures` 來產生 third-order polynomial transformation。

前一題選出最好的$\lambda = 100$，所以 C=0.005 當作-c 的參數值並利用整個 training data 進行訓練，訓練好的 model 為 m。

最後利用 test data 來 predict 並計算出 Eout 值。

結果如下圖為 0.0725 選 d。

```
Accuracy = 92.75% (742/800) (classification)
[-1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0, -1.0, 1.0
(92.75, 0.29, 0.7311043053833289)
[[-1.5035155009790364], [-0.4461375876088556 3],
Eout:  0.0725
```

16. ANS [b]

Now split the given training examples in $\mathcal{D}$ to five folds, the first 40 being fold 1, the next 40 being fold 2, and so on. Again, we take a fixed split because the given examples are already randomly permuted. Select the best λ^* as

$$\underset{\log_{10}\lambda\in\{-4,-2,0,2,4\}}{\operatorname{argmin}} E_{\text{cv}}(\mathcal{A}_\lambda).$$

Break the tie, if any, by selecting the largest λ. What is the value of $E_{\text{cv}}(\mathcal{A}_{\lambda^*})$ Choose the closest answer; provide your command/code.

[a] 0.07
[b] 0.08
[c] 0.09
[d] 0.10
[e] 0.11

colab：
https://colab.research.google.com/drive/1ttaMSDqvtmanWLS1bHRxHhmlNuZZ-c8v?usp=sharing

程式大綱：

1. pip install -U liblinear-official 與 import 套件
2. Get Data
3. Third-order Polynomial Transformation
4. 5-fold Cross Validation
5. C = 1/(2*lambda)
6. LIBLINEAR train and predict
7. Calculate Ecv

```
pip install -U liblinear-official
```

```
Collecting liblinear-official
  Downloading liblinear-official-2.43.0.tar.gz (47 kB)
     |████████████████████████████████| 47 kB 1.8 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from li
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: liblinear-official
  Building wheel for liblinear-official (setup.py) ... done
  Created wheel for liblinear-official: filename=liblinear_official-2.43.0-cp37-cp37m-l
  Stored in directory: /root/.cache/pip/wheels/83/6c/0d/06763f665bbc6f69fdeb3ff868df47b
Successfully built liblinear-official
Installing collected packages: liblinear-official
Successfully installed liblinear-official-2.43.0
```

```
import urllib.request
import numpy as np
import math
from sklearn.preprocessing import PolynomialFeatures
from liblinear.liblinearutil import *
```

```
def getData(url):
    #get data
    contents = urllib.request.urlopen(url).read()
    text = str(contents,'utf-8')
    # data preprocessing
    sptext = text.split('\n')

    X = []
    Y = []

    for i in range(len(sptext)-1):
        data = sptext[i].split(' ')
        # print(data)
        label = data[-1]
        # print(type(data))
        data.pop()
        # print(len(data))
        X += [data]
        Y += [label]

    X = np.asarray(X, dtype = float)
    Y = np.asarray(Y, dtype = float)
    # print(X)
    # print(Y)
    return X, Y
```

```python
if __name__ == '__main__':
    # Get Data
    train_data = "https://www.csie.ntu.edu.tw/~htlin/course/ml21fall/hw4/hw4_train.dat"
    train_X ,train_Y = getData(train_data)

    # Third-order Polynomial Transformation
    poly = PolynomialFeatures(3)
    train_X = poly.fit_transform(train_X)

    # 5-fold Cross Validation
    cv_train_X = []
    cv_train_Y = []
    cv_val_X = []
    cv_val_Y = []
    for i in range(0, len(train_X), 40):
        if i == 0:
            cv_train_X.append(train_X[i+40:])
            cv_train_Y.append(train_Y[i+40:])
        else:
            cv_train_X.append(np.concatenate((train_X[:i], train_X[i+40:])))
            cv_train_Y.append(np.concatenate((train_Y[:i], train_Y[i+40:])))

    for i in range(0, len(train_X), 40):
        cv_val_X.append(train_X[i:i+40])
        cv_val_Y.append(train_Y[i:i+40])

    # C = 1/(2*lambda)
    lamb = [0.0001, 0.01, 1, 100, 10000]
    C = []
    for l in lamb:
        C.append(str(1/(2*l)))

    # LIBLINEAR train and predict
    for c in C:
        print("c: ",c)
        AccvalN = 0
        for i in range(5):
            prob = problem(cv_train_Y[i], cv_train_X[i])
            param = parameter('-s 0 -c {} -e 0.000001'.format(c))
            m = train(prob, param)
            p_label, p_acc, p_val = predict(cv_val_Y[i], cv_val_X[i], m)
            # print(p_label)
            # print(p_acc)
            # print(p_val)
            AccvalN += p_acc[0]
        Ecv = (100 - (AccvalN/5))/ 100
        print("Ecv: ",Ecv)
```

利用 `PolynomialFeatures` 來產生 third-order polynomial transformation。

此題要求做 5-fold Cross Validation，將 training data 切成五等分，每等分為 40 筆，每份都輪流當作 validation data，其餘當作該次的 training data。

將先算好的 lambda 值給 lamb list，利用 for 迴圈算出 C。

用大 for 迴圈去跑每一個 C 當作-c 的參數值進行訓練，裡面都再用小 for 迴圈去跑五次 cross validation，訓練好的 model 為 m，利用 validation data 進行 predict，算出

五次的 accuracy，累加進去 AccvalN。跳出小 for 迴圈算出該 lambda 的 Ecv。

根據 Ecv 的公式

$$E_{\text{cv}}(\mathcal{H}, \mathcal{A}) = \frac{1}{V} \sum_{v=1}^{V} E_{\text{val}}^{(v)}(g_v^-)$$

剛剛 AccvalN 存的為五次 accuracy 的累加，單次的 100-accuracy=Eval%，所以(500-AccvalN)/5 = 100-(AccvalN/5) = Ecv%，Ecv%/100 = Ecv。

結果如下圖為最低的 Ecv = 0.08 選 b。

```
c:   5000.0
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)
Ecv:   0.19
c:   50.0
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)
Ecv:   0.17
c:   0.5
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)
Ecv:   0.115
c:   0.005
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)
Ecv:   0.08
c:   5e-05
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)
Ecv:   0.21
```