1. (20 points) If some algorithm always takes a total CPU time of aN^3 for training a binary classifier on a size-N binary classification data set. Consider a size-N K-class classification data set where each class is of size N/K. What is the total CPU time needed for training a K-class classifier via one-versus-one decomposition on the data set (ignoring the minor time needed for re-labeling the data set for the sub-problems)? List your derivation steps.

(Note: This result tells you that one-versus-one may actually be computationally "cheap" because each sub-problem has fewer data.)

$$C_2^k = \frac{k(k-1)}{2}$$

$$a\left(\frac{N}{k}\times 2\right)^3 \frac{k(k-1)}{2} = 4a\frac{N^3(k-1)}{k^2}$$

#

2. (20 points) Consider the following matrix, which is called the Vandermonde matrix.

$$\mathrm{V} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{N-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^{N-1} \end{bmatrix}$$

An N by N Vandermonde matrix has a determinant of

$$\det(\mathrm{V}) = \prod_{1 \le n < m \le N} (x_m - x_n)$$

and is thus invertible if all $\{x_n\}_{n=1}^N$ are different.

Consider some one-dimensional data $\{(x_n, y_n)\}_{n=1}^N$ where $x_n \in \mathbb{R}$ and $y_n \in \mathbb{R}$. Assume that all $\{x_n\}_{n=1}^N$ are different. Obtain a hypothesis $g(x) = \tilde{\mathbf{w}}^T \mathbf{\Phi}_Q(x)$ by applying a Q-dimensional polynomial transform $\mathbf{z}_n = \mathbf{\Phi}_Q(x_n)$, and running linear regression on $\{(\mathbf{z}_n, y_n)\}_{n=1}^N$ to get some $\tilde{\mathbf{w}}$. Use the property of the Vandermonde matrix above to prove that there exists some Q such that $E_{\text{in}}(g) = 0$ when E_{in} is measured by the squared error.

$$E_{in} = \sum_{n=1}^{N} (y_n - \tilde{w}^T \Phi_Q(x_n))^2 = 0$$

$$\Rightarrow \quad y_n = w^T \Phi_Q(x_n) = \begin{bmatrix} w_0, w_1 \dots \end{bmatrix} \begin{bmatrix} 1 \\ x_n \\ x_n^2 \\ \vdots \end{bmatrix}$$

$$= \begin{bmatrix} 1 & x_n & x_n^2 \dots \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \end{bmatrix}$$

$$\Rightarrow \quad y = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^Q \\ 1 & x_2 & & & \\ \vdots & & \ddots & & \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \end{bmatrix}$$

if w exist, and the determinant $= \prod (x_m - x_n) \Rightarrow V$ is invertible

$w = \begin{bmatrix} 1 & x_1 & x_1^2 \cdots x_1^Q \\ 1 & x_2 & \cdots \\ \vdots & & \end{bmatrix}^{-1} y$, the inverse matrix of V must exist

so there exist $Q = N-1$, such that $y_n = w^T \Phi_Q(x_n)$, $E_{in}(g) = 0$ #

3. (20 points) Assume that a transformer (no, not chat-Generative-Pretrained-Transformer!) peeks some one-dimensional examples and decides the following transform $\mathbf{\Phi}$ "intelligently" from the data of size N. The transform maps $x \in \mathbb{R}$ to $\mathbf{z} = (z_1, z_2, \ldots, z_N) \in \mathbb{R}^N$, where

$$(\mathbf{\Phi}(x))_n = z_n = [\![x = x_n]\!] .$$

Assume that each training and testing example is generated i.i.d. from a joint distribution $p(x, y)$ where x is sampled uniformly from $[-1, 1]$ and $y = x + \epsilon$, where ϵ is independently sampled from a Gaussian distribution with mean 0 and variance 1. For simplicity, you can assume that all x_n are different in the training data set. Consider a learning algorithm that performs linear regression after the feature transform (for simplicity, please exclude $z_0 = 1$) to get a $g(x) = \tilde{\mathbf{w}}^T \mathbf{\Phi}(x)$. Consider the squared error. What is $E_{\text{in}}(g)$? What is $E_{\text{out}}(g)$? List your derivation steps.

(Note: This result tells you that "snooping" your data too much can be a bad idea.)

$(\Phi(x))_n = z_n = [\![x = x_n]\!]$

$X = [x_1, x_2 \cdots x_N]$

$(\Phi(x))_2$ ↑

$(\Phi(x))_1$ ↑

$z_{x_1} = \begin{bmatrix} [\![x_1 = x_1]\!] \\ [\![x_1 = x_2]\!] \\ \vdots \\ [\![x_1 = x_N]\!] \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} z_1 \; z_2 \; \vdots$

$Z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{bmatrix}$ $\quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

$E_{in} = \frac{1}{N} \sum_{n=1}^{N} (y_n - \tilde{w}^T (\Phi(x))_n)^2$, we can find $\tilde{w}$ that perfectly fit

$y_n = \tilde{w}^T (\Phi(x))_n : [w_1, w_2 \cdots] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{bmatrix} = [y_1, y_2 \cdots]$

$\Rightarrow E_{in} = 0$ #

X_{test}

$Z_{x_{test,1}} = \begin{bmatrix} [\![x_{test,1} = x_1]\!] \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}$

$Z_{test} = \begin{bmatrix} 0 & 0 \cdots \\ 0 & \ddots \\ \vdots & \\ & 0 \end{bmatrix}$, $E_{out} = \mathbb{E}(y - \tilde{w}^T (\Phi(x_{test}))_n)^2$ (the $\Phi(x_{test})$ term is 0)

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

$= \mathbb{E}(x^2) + \mathbb{E}(2x\epsilon) + \mathbb{E}(\epsilon^2)$

$= \frac{1}{2} \int_{-1}^{1} x^2 dx + 0 + \text{Var}(\epsilon) + (E(\epsilon))^2$

$= \frac{1}{2} \frac{1}{3} x^3 \Big|_{-1}^{1} + 1 + 0^2$

$= \frac{4}{3}$ #

4. (20 points) On page 20 of Lecture 13, we discussed about adding "virtual examples" (hints) to help combat overfitting. One way of generating virtual examples is to add a small noise to the input vector $\mathbf{x} \in \mathbb{R}^{d+1}$ (including the 0-th component x_0) For each $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$ in our training data set, assume that we generate virtual examples $(\tilde{\mathbf{x}}_1, y_1), (\tilde{\mathbf{x}}_2, y_2), \ldots, (\tilde{\mathbf{x}}_N, y_N)$ where $\tilde{\mathbf{x}}_n$ is simply $\mathbf{x}_n + \boldsymbol{\epsilon}$ and each component of the noise vector $\boldsymbol{\epsilon} \in \mathbb{R}^{d+1}$ is generated i.i.d. from a uniform distribution within $[-\delta, \delta]$. The vector $\boldsymbol{\epsilon}$ is a random vector that varies for each virtual example.

Recall that when training the linear regression model, we need to calculate $\mathrm{X}^T\mathrm{X}$ first. Define the hinted input matrix

$$\mathrm{X}_h = \begin{bmatrix} | & \cdots & | & | & \cdots & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_N & \tilde{\mathbf{x}}_1 & \cdots & \tilde{\mathbf{x}}_N \\ | & \cdots & | & | & \cdots & | \end{bmatrix}^T.$$

What is the expected value $\mathbb{E}(\mathrm{X}_h^T \mathrm{X}_h)$ as a function of X and δ, where the expectation is taken over the (uniform)-noise generating process above? Prove your result.

(Note: This result may ring a bell on how such virtual examples can act like regularizers.)

$$X_h^T \cdot X_h = \begin{bmatrix} | & & | & | & & | \\ x_1 & \cdots & x_N & \tilde{x}_1 & \cdots & \tilde{x}_N \\ | & & | & | & & | \end{bmatrix} \begin{bmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \\ — & \tilde{x}_1 & — \\ & \vdots & \\ — & \tilde{x}_N & — \end{bmatrix}$$

$$(X_h^T X_h)_{i,j} = \sum_{n=1}^{N} X_{i,n}^T \times X_{n,j} + \sum_{n=1}^{N} (X_{i,n}^T + \varepsilon_{i,n})(X_{n,j} + \varepsilon_{n,j})$$

$$= 2\sum_{n=1}^{N} X_{i,n}^T X_{n,j} + \sum_{n=1}^{N} X_{i,n}\varepsilon_{n,j} + \varepsilon_{i,n} X_{n,j}^T + \varepsilon_{i,n}\varepsilon_{n,j}$$

$$X_h^T X_h = 2X^T X + X^T \varepsilon + \varepsilon^T X + N\varepsilon^T \varepsilon$$

$$E(X_h^T X_h) = 2X^T X + E(\varepsilon) \cdot (X^T + X) \times \begin{bmatrix} 1 & 1 & \cdots \\ 1 & \ddots & \\ \vdots & & \end{bmatrix} + N \begin{bmatrix} \varepsilon_1^2 & \varepsilon_1\varepsilon_2 & \\ \varepsilon_1\varepsilon_2 & \varepsilon_2^2 & \\ \vdots & & \ddots \\ & & & \varepsilon_N^2 \end{bmatrix}$$

$$E(\varepsilon) = \int_{-\delta}^{\delta} \frac{x}{2\delta} dx = 0$$

$$E(\varepsilon^2) = \int_{-\delta}^{\delta} \frac{x^2}{2\delta} dx = \frac{\delta^3}{3}$$

$$= 2X^T X + N\frac{\delta^3}{3} \cdot I \quad \#$$

5. (20 points) Consider the augmented error

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

with some $\lambda > 0$. When minimizing E_{aug} with the fixed-learning rate gradient descent algorithm with a learning rate $\eta > 0$, the update rule is

$$\mathbf{w}_{t+1} \leftarrow \alpha(\mathbf{w}_t - \beta\nabla E_{\text{in}}(\mathbf{w}_t)).$$

What are α and β? Prove your result.

(Note: You should get some $\alpha < 1$, which means that the weight vector is decayed (decreased). This is why L2 regularizer is often also called the weight-decay regularizer.)

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

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

$$= \frac{N-2\lambda\eta}{N} W_t - \eta \nabla E_{in}$$

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

$$\alpha = \frac{N-2\lambda\eta}{N} \qquad \beta = \frac{N\eta}{N-2\lambda\eta}$$

#

6. (20 points) Consider a one-dimensional data set $\{(x_n, y_n)\}_{n=1}^N$ where each $x_n \in \mathbb{R}$ and $y_n \in \mathbb{R}$. Then, solve the following one-variable regularized linear regression problem:

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

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

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

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 $\lambda > 0$. In particular,

$$\lambda = \frac{\alpha}{\sqrt{C}} + \beta$$

What are α and β? Prove your result.

(Note: This should allow you to see how λ decreases [when $\lambda > 0$] as C increases [until some upper bound].)

$$\min \frac{1}{N}\sum_{n=1}^{N}(w\cdot x_n - y_n)^2 + \frac{\lambda}{N}w^2$$

$$= \min \frac{1}{N}\sum_{n=1}^{N}(w^2x_n^2 + y_n^2 - 2wx_ny_n) + \frac{\lambda}{N}w^2$$

$$\frac{d}{dw}\left(\frac{1}{N}\sum_{n=1}^{N}(w^2x_n^2 + y_n^2 - 2wx_ny_n) + \frac{\lambda}{N}w^2\right) = 0$$

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

$$2w\sum_{n=1}^{N}x_n^2 - 2\sum_{n=1}^{N}x_ny_n + 2\lambda w = 0$$

$$w\left(\sum_{n=1}^{N}x_n^2 + \lambda\right) - \sum_{n=1}^{N}x_ny_n = 0$$

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

$$\sqrt{C}\times\left(\sum_{n=1}^{N}x_n^2 + \lambda\right) = \sum_{n=1}^{N}x_ny_n$$

$$\lambda = \left(\frac{1}{\sqrt{C}}\sum_{n=1}^{N}x_ny_n - \sum_{n=1}^{N}x_n^2\right) = \frac{1}{\sqrt{C}}\sum_{n=1}^{N}x_ny_n - \sum_{n=1}^{N}x_n^2$$

$$\alpha = \sum_{n=1}^{N}x_ny_n \qquad \beta = -\sum_{n=1}^{N}x_n^2$$

7. (20 points) 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 L1-regularized linear regression with the transformed data $\{(\Phi(\mathbf{x}_n), y_n)\}_{n=1}^N$.

$$\min_{\tilde{\mathbf{w}} \in \mathbb{R}^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{\mathbf{w}}^T \Phi(\mathbf{x}_n) - y_n)^2 + \frac{\lambda}{N} \|\tilde{\mathbf{w}}\|_1$$

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

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

What is $\Omega(\mathbf{w})$? How do the optimal $\tilde{\mathbf{w}}$ and the optimal $\mathbf{w}$ correspond to each other? Prove your result.

(Note: The result shows you how scaling the data effectively changes the regularizer.)

$V = \begin{bmatrix} a & & \\ & b & \\ & & c \\ & & & \ddots \end{bmatrix}$ diagonal

$\min_{\tilde{w} \in R^{d+1}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{w}^T (V x_n) - y_n)^2 + \frac{\lambda}{N} \|\tilde{w}\|_1 \qquad X = \begin{bmatrix} - x_1 - \\ - x_2 - \\ \vdots \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix}$

equivelent to :

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

$(V^T = V)$

$\Rightarrow \quad \tilde{w}^T V = w^T \quad \Rightarrow \quad V\tilde{w} = w \quad \Rightarrow \quad \tilde{w} = V^{-1} w$ #

$\Omega(w) = \|\tilde{w}\|_1 = \|V^{-1} w\|_1$ #

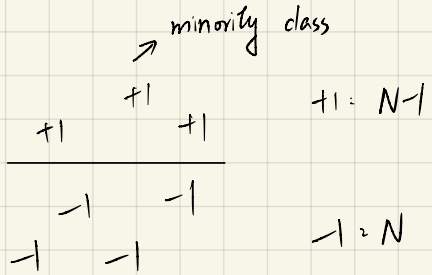
8. (20 points) Consider a binary classification algorithm $\mathcal{A}_{\text{minority}}$, which returns a constant classifier that always predicts the minority class (i.e., the class with fewer instances in the data set that it sees). As you can imagine, the returned classifier is the worst-E_{in} one among all constant classifiers. Consider the 0/1 error. For a binary classification data set with N positive examples and N negative examples, what is $E_{\text{loocv}}(\mathcal{A}_{\text{minority}})$? Prove your result.

(Note: This result may tell you that in some special situations, leave-one-out cross-validation is not always trustworthy.)

(1) if we choose +1 as test data, the number of positive data would be N-1 and it would be minority class. the err^+ would be 0

(2) if we choose -1 as test data, the number of negative data would be N-1 and it would be minority class, the err^- would be 0

$\Rightarrow$ $E_{loocv}(A_{minority}) = \frac{0}{2N} = 0$, although the E_{loocv} is 0, the Loocv may give a misleading estimation. because it evaluate the ability to handle specific instances, so it not always trustworthy.

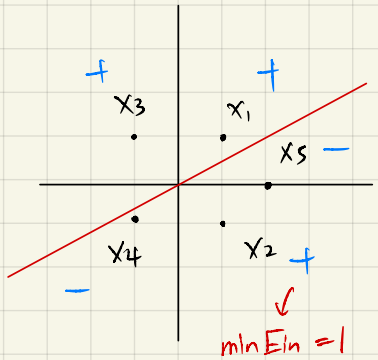


9. In Lecture 16, we talked about the probability to fit data perfectly when the labels are random. For instance, page 6 of Lecture 16 shows that the probability of fitting the data perfectly with decision stumps is $(2N)/2^N$. Consider five points in $\mathbb{R}^2$ as input vectors $\mathbf{x}_1 = (+1, +1)$, $\mathbf{x}_2 = (+1, -1)$, $\mathbf{x}_3 = (-1, +1)$, $\mathbf{x}_4 = (-1, -1)$, $\mathbf{x}_5 = (2, 0)$, and a 2D perceptron model that minimizes $E_{\text{in}}(\mathbf{w})$ to the lowest possible value. One way to measure the power of the model is to consider five random labels y_1, y_2, y_3, y_4, y_5, each in ± 1 and generated by i.i.d. fair coin flips, and then compute

$$\mathbb{E}_{y_1,y_2,y_3,y_4,y_5}\left(\min_{\mathbf{w}\in\mathbb{R}^{2+1}} E_{\text{in}}(\mathbf{w})\right)$$

in terms of the 0/1 error. For a perfect fitting, $\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w})$ will be 0; for a less perfect fitting (when the data is not linearly separable), $\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w})$ will be some non-zero value. The expectation above averages over all 32 possible combinations of y_1, y_2, y_3, y_4, y_5. What is the value of the expectation? Prove your result.

(*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	−1	1	1	1	−1	1	−1	−1	−1	1
X_2	−1	−1	1	1	1	1	1	−1	−1	−1
X_3	1	−1	−1	−1	1	−1	1	1	1	−1
X_4	1	1	−1	1	−1	−1	−1	1	−1	1
X_5	1	1	1	−1	1	−1	−1	−1	1	−1
		2⊖ 3⊕					2⊕ 3⊖			

C_2^5 - two ⊖ are adjacent = 10 - 5 = 5

C_2^5 - two ⊕ are adjacent = 10 - 5 = 5

only 10 combination can't be seperate.

there are 10 possible combination that occurs error = 1

$$\mathbb{E}_{y_1\sim y_5}(\min E_{in}(w)) = \frac{1\times 10}{32} = \frac{5}{16} \#$$

10. (20 points, *) Select the best λ^* as

$$\underset{\log_{10}\lambda\in\{-6,-4,-2,0,2\}}{\operatorname{argmin}} E_{\text{in}}(\mathbf{w}_\lambda).$$

Break the tie, if any, by selecting the largest λ. What is $\log_{10}(\lambda^*)$?

```
Accuracy = 96% (192/200) (classification)
Accuracy = 92% (184/200) (classification)
Accuracy = 91% (182/200) (classification)
Accuracy = 87.5% (175/200) (classification)
Accuracy = 80.5% (161/200) (classification)
```

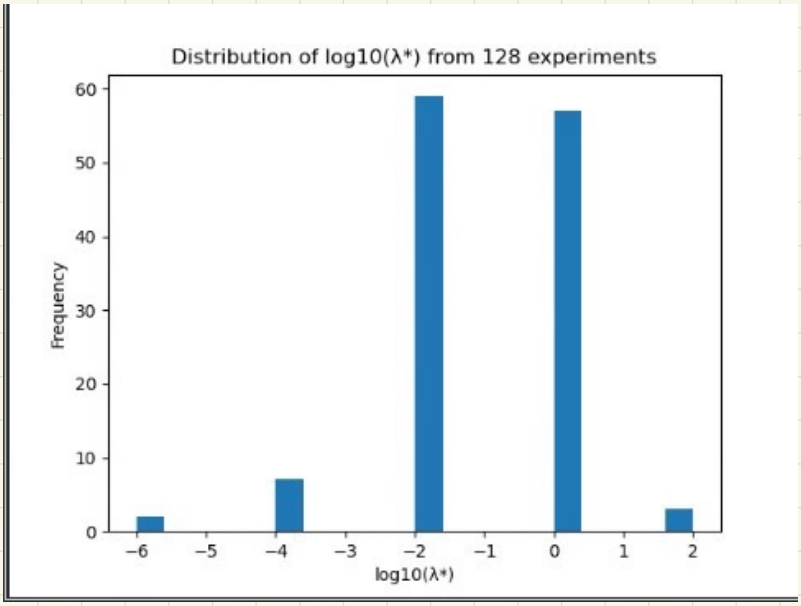
→ highest accuracy ⇒ min Ein(wλ)

$\log \lambda^* = -6$ #

11. (20 points, *) Now randomly split the given training examples in $\mathcal{D}$ to two sets: 120 examples as $\mathcal{D}_{\text{train}}$ and 80 as $\mathcal{D}_{\text{val}}$. Run $\mathcal{A}_\lambda$ on *only* $\mathcal{D}_{\text{train}}$ to get $\mathbf{w}_\lambda^-$ (the weight vector within the g^- returned), and validate $\mathbf{w}_\lambda^-$ with $\mathcal{D}_{\text{val}}$ to get $E_{\text{val}}(\mathbf{w}_\lambda^-)$. Select the best λ^* as

$$\underset{\log_{10}\lambda\in\{-6,-4,-2,0,2\}}{\operatorname{argmin}} \quad E_{\text{val}}(\mathbf{w}_\lambda^-).$$

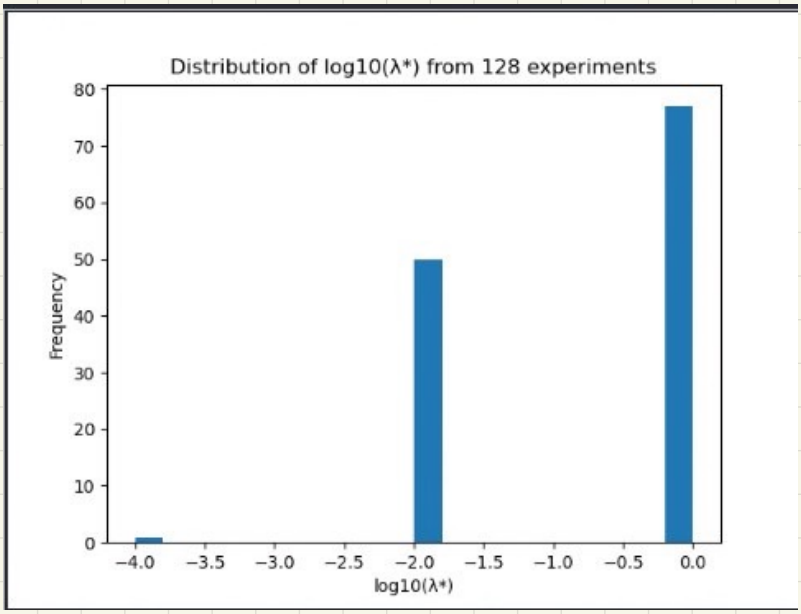
Break the tie, if any, by selecting the largest λ. Repeat the experiment above for 128 times, each with a different random split. Plot a histogram on the distribution of $\log_{10}(\lambda^*)$ selected from the 128 experiments.



12. (20 points, *) Now randomly split the given training examples in $\mathcal{D}$ to five folds, 40 being fold 1, another 40 being fold 2, and so on. Select the best λ^* as

$$\underset{\log_{10}\lambda\in\{-6,-4,-2,0,2\}}{\operatorname{argmin}} E_{\mathrm{CV}}(\mathcal{A}_\lambda).$$

Break the tie, if any, by selecting the largest λ. Repeat the experiment above for 128 times, each with a different random split. Plot a histogram on the distribution of $\log_{10}(\lambda^*)$ selected from the 128 experiments. Compare your result with the $\log_{10}(\lambda^*)$ selected for the two problems above. Describe your findings.



most of the time $\log_{10}(\lambda)=0$ has the minimum Ein, while $\log_{10}(\lambda)=-2$ gives the minimum Ein in Problem 11, and $\log_{10}(\lambda)=-6$ gives the minimum Ein in Problem 10.

both Problem 11 and Problem 12 has two peak at $\log_{10}(\lambda)=-2$ and 0 however, Problem 12 almost get 0 time at $\log_{10}(\lambda)=-6,-4,2$ but $\log_{10}(\lambda)=-6$ get the highest accuracy at Problem 10.

13. (Bonus 20 points) Dr. Regularize recently learned regularization and thought that its basic goal is to restrict the length of the weight vector $\mathbf{w}$ to be less than $\sqrt{C}$. Ze then designed a "new" regularization algorithm—simply performing linear regression first to get some $\mathbf{w}_{\text{LIN}}$, and then get $\mathbf{w}_C = \frac{\mathbf{w}_{\text{LIN}}}{\|\mathbf{w}_{\text{LIN}}\|} \cdot \sqrt{C}$. Then, $\mathbf{w}_C$ would be of length $\sqrt{C}$ only. Ze then asks chatGPT whether this is equivalent to the C-constrained regularization (and hence equivalent to λ-penalized L2 regularization) that ze learned in class, and got the following answer.

After reading the answer from chatGPT, ze still does not understand why scaling after linear regression is different from the C-constrained linear regression. Please help ze understand chatGPT's answer by proving that if $\mathbf{w}_{\text{LIN}} \neq \mathbf{0}$, then $\mathbf{w}_C$ solves the C-constrained linear regression problem

$$\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) \text{ subject to } \mathbf{w}^T\mathbf{w} \leq C.$$

if and only if $\mathrm{X}^T\mathrm{X} = \alpha \mathrm{I}$, meaning that there is *no* dependence between the features.

(Hint: The "no dependence" condition tells you that regularized regression (i.e. ridge regression) handles the dependence cases better, which corresponds to the "multicollinearity" issue mentioned by chatGPT.)

$W_{Lin} = (X^TX)^{-1}X^Ty$

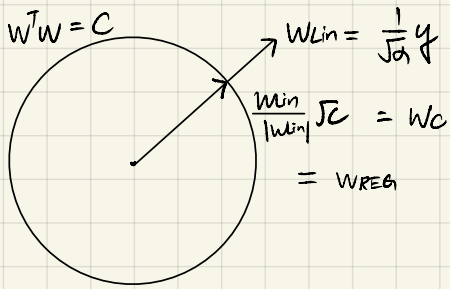
$\min_w E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}(w^Tx_n - y_n) \quad . \quad \sum_{q=0}^{Q} w_q^2 \le C$

If $X^TX = \alpha I = \begin{bmatrix} \alpha & & \\ & \alpha & \\ & & \alpha \end{bmatrix}_{N\times N}$, x_n are orthogonal

$\Rightarrow \quad X = \begin{bmatrix} \sqrt{\alpha} & 0 & \cdots \\ 0 & \sqrt{\alpha} & \\ \vdots & & \ddots \end{bmatrix}_{N\times N}$

$W_{Lin} = \frac{1}{\alpha}X^Ty = \frac{1}{\alpha}Xy = \frac{1}{\sqrt{\alpha}}y$

$W_C = \sqrt{\frac{C}{N}}y \Rightarrow W_C \parallel y$



in terms of constrained regularization

we update w until $W_{REG} \parallel -\nabla E_{in}$

$\Rightarrow \quad W_{REG}^T x_n - y_n = 0$

$\Rightarrow \quad W_{REG}^T X = y^T$

$W_{REG}^T \sqrt{\alpha} I = y^T \quad W_{REG} = \frac{1}{\sqrt{\alpha}}y$, also $w^Tw = C \Rightarrow W_{REG} = \sqrt{\frac{C}{N}}y$

When $X^TX = \alpha I$, we have $W_C = W_{REG} = \sqrt{\frac{C}{N}}y$ #