Mbaruka

1) Probability

i) W is gaussian with $N(\mu, \sigma^2)$ & U is uniform random $[a, b]$. Assum W & U are independent, what is $E[z]$ & $Var[z]$ of $Z = 3W + 2U$?

Note:
$E[W] = \mu$ | $E[U] = \frac{a+b}{2}$
$Var(W) = \sigma^2$ | $Var(U) = \frac{(b-a)^2}{12}$

$$E[z] = E[3W + 2U]$$
$$= 3E[W] + 2E[U]$$
$$= 3\mu + 2\left(\frac{a+b}{2}\right)$$
$$= 3\mu + (a+b)$$

$$Var[z] = Var[3W + 2u]$$
$$= 3^2 Var[W] + 2^2 Var(u)$$
$$= 9\sigma^2 + 4\left(\frac{(b-a)^2}{3}\right)$$
$$= 9\sigma^2 + \frac{2(b-a)^2}{3}$$
$$= 9\sigma^2 + \frac{(b-a)^2}{3}$$

ii)

P(X, Y)		Y			
		a	b	c	d
X	T	0.1	0.2	0.1	0.1
	F	0.1	0.1	0.2	0.1

a) Marginal distribs P_r, $P_r(Y=a)$, $P_r(Y=b)$, $P_r(Y=c)$, $P_r(Y=d)$

$P_r(Y=a) = 0.1 + 0.1 = 0.2$
$P_r(Y=b) = 0.2 + 0.1 = 0.3$
$P_r(Y=c) = 0.1 + 0.2 = 0.3$
$P_r(Y=d) = 0.1 + 0.1 = 0.2$

b) $P_r(X=T \mid Y \in \{b, c, d\})$: Prob that X is T given that Y takes $\{b, c, d\}$

$$P_r(X=T \mid Y \in \{b,c,d\}) = \frac{P_r(X=T, Y \in \{b,c,d\})}{P_r(Y \in \{b,c,d\})}$$
$$= \frac{P_r(X=T, Y=b) + P_r(X=T, Y=c) + P_r(X=T, Y=d)}{P_r(Y=b) + P_r(Y=c) + P_r(Y=d)}$$
$$= \frac{0.2 + 0.1 + 0.1}{0.3 + 0.3 + 0.2}$$
$$= \frac{0.4}{0.8}$$
$$= 0.5$$

2) Linear Algebra

1) $A_k \in \mathbb{R}^{n \times n}$ for $k = 1, \ldots k$ such that $A_k = A_k^T$ ∴ A_k is symmetric. If all A_k have same set of eigenvectors $u_1, u_2 \ldots u_n$ with corresponding eigenvalues $\alpha_{k1} \ldots \alpha_{kn}$ for each A_k. Write Eigenvectors and their corresponding eigenvalues for:

a) $C = \sum_{k=1}^{k} A_k$

$Av = \lambda v$

∴ $A_k u_i = \alpha_{ki} u_i$

$C u_i = \left(\sum_{k=1}^{k} A_k\right) u_i = \sum_{k=1}^{k} (A_k u_i)$

Sub $A_k u_i = \alpha_{ki} u_i$

$C u_i = \sum_{ki}^{k} (\alpha_{ki} u_i)$

$= \left(\sum_{ki}^{k} \{\alpha_{ki}\}\right) u_i$

$C u_i = \lambda u_i$

$\lambda = \sum_{ki}^{k} \alpha_{ki} \Rightarrow$ eigenvalues

$u_i \Rightarrow u_i \Rightarrow$ eigenvectors

∴ for $C = \sum_{k=1}^{k} A_k$

eigenvalues $\Rightarrow \sum_{ki}^{k} \alpha_{ki} \ldots \sum_{k=1}^{k} \alpha_{kn}$

eigenvectors $\Rightarrow u_i \ldots u_n$.

b) $D = A_i^{-1} A_j A_i$ where $i \neq j$ and $i, j \in \{1, 2 \ldots k\}$: Assum A_i is invertible

We assume A_i & A_j share same eigenvector u_i

∴ $A_i u_m = \alpha_{im} u_m$; $A_j u_m = \alpha_{jm} u_m$

for $A_i^{-1} u_m = \frac{1}{\alpha_{im}} u_m$

Matrix $D = A_i^{-1} A_j A_i$

Sub $D u_m$ $\Rightarrow A_i^{-1} A_j \alpha_{im} u_m$

$\Rightarrow \alpha_{im} A_i^{-1} (A_j u_m)$

$\Rightarrow \alpha_{im} A_i^{-1} \alpha_{jm} u_m$

$\Rightarrow \alpha_{im} \frac{u_m}{\alpha_{im}} \alpha_{jm}$

$D u_m = \alpha_{jm} u_m$

Form: $D u_m = \lambda u_m$

∴ $\lambda = \alpha_{jm} \Rightarrow$ eigenvalue

$u_m \Rightarrow$ eigenvector.

∴ For D

Eigen values

$\Rightarrow \alpha_{jm}: \alpha_{j1}, \alpha_{j2} \ldots \alpha_{jn}$

Eigen vectors

$\Rightarrow u_m: u_1 \ldots u_n$.

2) $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $Col(A)$ - column space of A. For value of m, under what conditions on b, $Col(A)$ and $Rank(A)$ will equation $Ax = b$ have.

a) no solution?

$Ax = b$ is solvable if & only if $b \in Col(A)$

$\therefore$ for no solution

$$b \notin col(A)$$

b) Exactly one solution

$\Rightarrow Ax = b$ to have one solution

* b must lie in $Col(A)$
* A must have full rank ($rank(A) = n$) → Columns of A be linearly independent.

$\therefore b \in Col(A)$ and $rank(A) = n$

c) Infinitely many Solution

$\Rightarrow Ax = b$ to have infinite solutions

* b must lie in $col(A)$
* A must have full rank ($rank(A) < n$) → Colun of A be linearly dependent.

$\therefore b \in Col(A)$ and $rank(A) < n$

3) Matrix Calculus

Find derivative with respect to X. Follow convention that the derivative of a scalar function $f(x)$ with respect to X should have same dimension as X

a) $f(x) = tr(XX^T)$, where $X \in \mathbb{R}^{n \times n}$ & tr is trace of square matrix

$$tr(A) = \sum_{i=1}^{n} A_{ii}$$

$$\therefore tr(XX^T) = \sum_{i=1}^{n} \sum_{k=1}^{n} x_{ik} x_{jk} = \sum_{i=1}^{n} \sum_{k=1} x_{ik} x_{ik}$$

$$\frac{\partial}{\partial X} tr(XX^T) = 2X$$

$$\therefore f(x) = \sum_{i=1}^{n} \sum_{k=1}^{n} x_{ik}^2$$

$$\therefore \frac{\partial}{\partial x} f(x) = 2x$$

$\mathbb{Z}$

b) $f(x) = a^T x b$ where $X \in \mathbb{R}^{n \times n}$ and $a \in \mathbb{R}^m$ & $b \in \mathbb{R}^n$

$$f(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i x_{ij} b_j$$

$$\frac{\partial}{\partial x} f(x) = \frac{\partial}{\partial x_{ij}} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_i x_{ij} b_j \right)$$

$$= a_i b_j \Rightarrow a_i b_j^T \Rightarrow ab^T$$

$$\frac{\partial}{\partial x} f(x) = \frac{\partial}{\partial x} (a^T X b) = ab^T$$

$$= ab^T$$

$\mathbb{Z}$

c) $f(x) = \|Xb\|_2^2$, where $X \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$

For norms: $\|v\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$ | $\|v\|_2^2 = \sum_{i=1}^{n} v_i^2 = v^T v$

But $v = Xb$

Substitute:

$$\|Xb\|_2^2 = (Xb)^T (Xb) = b^T X^T X b$$

Since $b^T X^T X b$ is scalar we can perform trace

$$f(x) = tr(b^T X^T X b) = tr(X b b^T X^T)$$

$$\frac{\partial}{\partial X} tr(X b b^T X^T) = 2 X b b^T$$

4) MLE - MAP

$x \in \mathbb{R}$ is fixed and given. Assume $\lambda > 0$ is scalar parameter, and:

$$y \sim \text{Poisson}(\lambda)$$

Where Poisson(λ) denotes the poisson distribution with rate (λ):

1) Maximum Likelihood Estimate:

a) PMF of $y|\lambda$

$$P(y|\lambda) = \frac{\lambda^y e^{-\lambda}}{y!}; \quad y = 0, 1, 2.., \quad \lambda > 0$$

b) Assum N independent observations $y_1, y_2 \ldots y_N$ are drawn, where $y_n \sim \text{Poisson}(\lambda)$ for $n = 1, 2, \ldots N$. Derive joint PMF $P(y_1 \ldots y_N|\lambda)$.

$$P(y_1, y_2 \ldots y_N|\lambda) = P(y_1|\lambda) P(y_2|\lambda) \ldots P(y_N|\lambda)$$

$$= \left(\frac{\lambda^{y_1} e^{-\lambda}}{y_1!}\right)\left(\frac{\lambda^{y_2} e^{-\lambda}}{y_2!}\right) \ldots \left(\frac{\lambda^{y_N} e^{-\lambda}}{y_N!}\right)$$

$$= \frac{\lambda^{(y_1+y_2 \ldots y_N)} e^{(-\lambda - \lambda \ldots - \lambda_n)}}{y_1!\, y_2! \ldots y_N!}$$

$$= \frac{\lambda^{\sum_{n=1}^N y_n} e^{-N\lambda}}{\prod_{n=1}^n y_n!}$$

c) Write the -ve log-likelihood function of joint PMF derived in the above and simplify it into the form that can be minimize (remove term that don't have λ)

$$l(\lambda) = \log\left(P(y_1, y_2 \ldots y_n|\lambda)\right)$$

$$= \log\left[\frac{\lambda^{\sum_{n=1}^N y_n} e^{-N\lambda}}{\prod_{n=1}^N y_n!}\right]$$

$$= \log\left[\lambda^{\sum_{n=1}^N y_n} e^{-N\lambda}\right] - \log\left[\prod_{n=1}^N y_n!\right]$$

Using $\log(a \cdot b) = \log a + \log b$

$$= \log\left(\lambda^{\sum_{n=1}^N y_n}\right) + \log\left(e^{-N\lambda}\right) - \log\left[\prod_{n=1}^N y_n!\right]$$

$\therefore \log\left(\lambda^{\sum_{n=1}^{N} y_n}\right) = \sum_{n=1}^{N} y_n \log(\lambda)$ $\Rightarrow$ from $\log(a^b) = b\log a$

$\therefore \log\left(e^{-N\lambda}\right) = -N\lambda$ from $\log(e^x) = x$

$\therefore \log\left(\prod_{n=1}^{N} y_n!\right) = \sum_{n=1}^{N} \log(y_n!)$

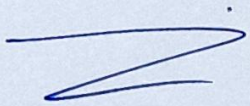
factoring in:

$$= \sum_{n=1}^{N} y_n \log(\lambda) - N\lambda - \sum_{n=1}^{n} \log(y_n!)$$

$$-\mathcal{L}(\lambda) = -\sum_{n=1}^{N} y_n \log(\lambda) + N\lambda + \sum_{n=1}^{N} \log(y_n!)$$

ignore parts that don't have λ to minimize.

$$\therefore -\mathcal{L}(x) = -\sum_{n=1}^{N} y_n \log(\lambda) + N\lambda$$



2) Maximum-a-Posteriori (MAP) Estimation: Suppose $\lambda \sim \text{Gamma}(\alpha, \beta)$ with PMF:

$$P(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda}, \quad \lambda > 0$$

a) Joint distribution $P(y_1, \dots y_n, \lambda)$ where each $y_n \sim \text{Poisson}(\lambda)$ is drawn independently as above.

* Incorporate prior distribution $P(\lambda)$ and the likelihood $P(y_1 \dots y_n | \lambda)$

$$P(y_1, \dots y_N, \lambda) = P(y_1, \dots y_N | \lambda) \cdot P(\lambda)$$

$$= \left(\frac{\lambda^{\sum_{n=1}^N y_n} e^{-N\lambda}}{\prod_{n=1}^N y_n!} \right) \left(\frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda} \right)$$

$$= \left(\frac{\lambda^{\sum_{n=1}^N y_n + \alpha - 1} e^{-N\lambda - \beta\lambda}}{\prod_{n=1}^N y_n!} \right) \cdot \frac{\beta^\alpha}{\Gamma(\alpha)}$$

b) -ve log of $P(y_1 \dots y_N, \lambda)$ and simplify into a form that can be minimize to find MAP estimate λ. (remove terms with no λ)

We ignore $\frac{\beta^\alpha}{\Gamma(\alpha)}$ and $\prod_{n=1}^N y_n!$

$$\therefore -\log P(y_1 \dots y_N) = -\log \left[\lambda^{\sum_{n=1}^N y_n + \alpha - 1} e^{-(N+\beta)} \right]$$

$$= -\left(\sum_{n=1}^N y_n + \alpha - 1 \right) \log(\lambda) + (N+\beta)\lambda$$

Minimize to find λ: Take derivative -ve log with respect to $\lambda = 0$

$$\frac{d}{d\lambda} \left(-\sum_{n=1}^N y_n + \alpha - 1 \right) \log(\lambda) = -\frac{\sum_{n=1}^N y_n + \alpha - 1}{\lambda}$$

$$\frac{d}{d\lambda} (N+\beta)\lambda = N + \beta$$

$$\therefore -\frac{\sum_{n=1}^N y_n + \alpha - 1}{\lambda} + (N+\beta) = 0$$

$$\therefore \lambda = \frac{\sum_{n=1}^N y_n + \alpha - 1}{N+\beta}$$

2

5. Linear Regression with Regularization

Dataset N samples (x_i, y_i) where:

$$y_i = x_i^T w + \varepsilon_i, \quad \varepsilon_i \sim N(0, \sigma^2)$$

and $w \in \mathbb{R}^d$ is weight vector, ε_i is gaussian noise and $x_i \in \mathbb{R}^d$

To enforce smoothness in the weights w, we introduce a regularizer based on differences between adjacent weights. This results in the following optimization Problem

$$\min_w L(w) = \|y - Xw\|_2^2 + \lambda \|w\|_2^2 + \mu \|Dw\|_2^2$$

Where $\lambda, \mu > 0$ are regularization parameters and $\|Dw\|_2^2 = \sum_{i=2}^{d-1} (2w_i - w_{i-1} - w_{i+1})^2$.

① Find $D \in \mathbb{R}^{(d-2) \times d}$ such that $\|Dw\|_2^2 = \sum_{i=2}^{d-1} (2w_i - w_{i-1} - w_{i+1})^2$. Write D explicitly. Note that $\|\cdot\|_2$ denotes the usual L_2 or Euclidean norm.

$$\|Dw\|_2^2 = \sum_{i=2}^{d-1} (2w_i - w_{i-1} - w_{i+1})^2$$

$$\sum_{i=1}^{d-2} (Dw)_i \Rightarrow 2w_i - w_{i-1} - w_{i+1}$$

for $i=2 \Rightarrow (Dw)_1 = 2w_2 - w_1 - w_3 \Rightarrow [-1\ 2\ -1\ \cdots\ 0]$

for $i=3 \Rightarrow (Dw)_2 = 2w_3 - w_2 - w_4 \Rightarrow [0\ -1\ 2\ -1\ \cdots\ 0]$

for $i=4 \Rightarrow (Dw)_3 = 2w_4 - w_3 - w_5 = [0\ 0\ -1\ 2\ -1\ \cdots\ 0]$

$$\therefore D = \begin{bmatrix} -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & -1 & 2 & -1 \end{bmatrix}$$

② Derive the closed-form solution do w^*, the minimize of $L(w)$

$$L(w) = \|y - Xw\|_2^2 + \lambda \|w\|_2^2 + \mu \|Dw\|_2^2 \Rightarrow y^T y - 2y^T Xw + w^T (X^T X + \lambda I + \mu D^T) w$$

Find gradient

Expand

$$\|y - Xw\|_2^2 = (y - Xw)^T (y - Xw)$$
$$= y^T y - 2y^T Xw + w^T X^T Xw$$
$$\lambda \|w\|_2^2 = \lambda w^T w$$
$$\mu \|Dw\|_2^2 = \mu (Dw)^T Dw$$
$$= \mu w^T D^T D w$$

Combine

$$\Rightarrow y^T y - 2y^T Xw + (w^T X^T Xw + \lambda w^T w + \mu w^T D^T D w)$$

Find gradient.

$\Rightarrow y^Ty - 2y^TXw + w^T(X^TX + \lambda I + uD^TD)w$

$y^Ty \Rightarrow$ Independent of $w \Rightarrow$ constant $\therefore \nabla_w(y^Ty) = 0$

$\nabla_w(-2y^TXw) =$ using $\nabla_w(b^TAw) = A^Tb$

$= -2X^Ty$

$\nabla_w(w^TX^TXw)$ using $\nabla_w(w^TAw) = 2Aw$

$= 2X^TXw$

$\nabla_w(\lambda w^Tw)$ using $\nabla_w(w^Tw) = 2w$

$= \lambda 2w$

$\nabla_w(\mu w^TD^TDw)$ using $\nabla(w^TAw) = 2Aw$

$= \mu 2D^TDw$

Combine

$\nabla_w L(w) = -2X^Ty + 2X^TXw + \lambda 2w + \mu 2D^TDw$

$= 2(X^TXw - X^Ty + \lambda w + \mu D^TDw)$

Equate to 0:

$X^TXw - X^Ty + \lambda w + \mu D^TDw = 0$

$X^TXw + \lambda w + \mu D^TDw = X^Ty$

$\therefore w^* = (X^TX + \lambda I + \mu D^TD)^{-1}X^Ty$

Z

3- Consider problem 1 without the smooth regularization

$$\min_w L_{ridge}(w) = \|y - Xw\|_2^2 + \lambda \|w\|_2^2$$

The minimizer of L_{ridge} be w^*_{ridge}. Show that the "smooth" regularizer is effective by proving $\|Dw^*\|_2^2 \leq \|Dw^*_{ridge}\|_2^2$.

$$\|Dw^*\|_2^2 \leq \|Dw^*_{ridge}\|_2^2$$

$$w^* = L(w) = \|y - Xw\|_2^2 + \lambda\|w\|_2^2 + \mu\|Dw\|_2^2$$

$$w^*_{ridge} = L_{ridge}(w) = \|y - Xw\|_2^2 + \lambda\|w\|_2^2$$

Loss values:

$$L(w^*) = \|y - Xw^*\|_2^2 + \lambda\|w^*\|_2^2 + \mu\|Dw^*\|_2^2$$

$$L_{ridge}(w^*_{ridge}) = \|y - Xw^*_{ridge}\|_2^2 + \lambda\|w^*_{ridge}\|_2^2$$

Compare

$$L(w^*) \leq L(w^*_{ridge})$$

$$\therefore \|y - Xw^*\|_2^2 + \lambda\|w^*\|_2^2 + \mu\|Dw^*\|_2^2 \leq \|y - Xw^*_{ridge}\|_2^2 + \lambda\|w^*_{ridge}\|_2^2 + \mu\|Dw^*_{ridge}\|_2^2$$

at w^*_{ridge} $\Rightarrow$ the smoothness term $\|Dw^*_{ridge}\|_2^2$ is not minized since the ridge loss $L_{ridge}(w)$ doesn't penalize smoothness. ✓

$$\therefore \mu\|Dw^*\|_2^2 \leq \mu\|Dw^*_{ridge}\|_2^2$$

since $\mu > 0$;

$$\therefore \|Dw^*\|_2^2 \leq \|Dw^*_{ridge}\|_2^2$$

(20) How smooth regularization affects the bias and variance of the learned model w.

Smooth regularization increases the bias because it forces the weights to vary less and leading to potential underfitting. whereas it reduces the variance and this makes it more robust to noise.

6) Online Update

In linear regression we observe variable y;

$$y = w^T x + \varepsilon$$

where $\varepsilon \sim N(0, \sigma^2)$ and $x, w \in \mathbb{R}^d$. In MLE of w is

$$\hat{w} = (X^T X)^{-1} X^T y$$

If we get new observation $(\tilde{x}, \tilde{y})$ and want to compute the updated MLE $\hat{w}_{new}$ after adding observation. $X_{new} = \begin{bmatrix} X \\ \tilde{x}^T \end{bmatrix} \in \mathbb{R}^{N+1 \times d}$ and $y_{new} = \begin{pmatrix} y \\ \tilde{y} \end{pmatrix} \in \mathbb{R}^{N+1}$

If we simply apply $\hat{w}_{new} = (X_{new}^T X_{new})^{-1} X_{new}^T y_{new}$ will need to solve $\hat{w}_{new}$. This can be solved $(X_{new}^T X_{new})$ by low rank correction of $(X^T X)$.

Note $X_{new}^T X_{new} = X^T X + \tilde{x}\tilde{x}^T$ Using Sherman-Morrison-Woodbury formula we can show that:

$$(X_{new}^T X_{new})^{-1} = (X^T X)^{-1} - \frac{(X^T X)^{-1} \tilde{x}\tilde{x}^T (X^T X)^{-1}}{1 + \tilde{x}^T (X^T X)^{-1} \tilde{x}} \quad \ldots (3)$$

Since $X_{new}^T y_{new} = X^T y + \tilde{x}\tilde{y}$, we have

$$\hat{w}_{new} = (X_{new}^T X_{new})^{-1} X_{new}^T y_{new}$$

$$= \left[(X^T X^{-1}) - \frac{(X^T X)^{-1} \tilde{x}\tilde{x}^T (X^T X)^{-1}}{1 + \tilde{x}^T (X^T X)^{-1} \tilde{x}} \right] (X^T y + \tilde{x}\tilde{y}) \quad \ldots (4)$$

1) Derive $\hat{w}_{new}$ using Sherman-Morrison formula. Your formula should express $\hat{w}_{new}$ explicitly in terms of $\hat{w}$

$$\hat{w}_{new} = (X^T X)^{-1} X^T y + (X^T X)^{-1} \tilde{x}\tilde{y} - \frac{(X^T X)^{-1} \tilde{x}\tilde{x}^T (X^T X)^{-1} (X^T y + \tilde{x}\tilde{y})}{1 + \tilde{x}^T (X^T X)^{-1} \tilde{x}}$$

Since: $\hat{w} = (X^T X)^{-1} X^T y$

$$= \hat{w} + (X^T X)^{-1} \tilde{x}\tilde{y} - \frac{(X^T X)^{-1} \tilde{x}\tilde{x}^T \hat{w} + (X^T X)^{-1} \tilde{x}\tilde{x}^T (X^T X)^{-1} \tilde{x}\tilde{y}}{1 + \tilde{x}^T (X^T X)^{-1} \tilde{x}}$$

Factoring out $\tilde{x}^T \hat{w}$

$$\hat{w}_{new} = \hat{w} + \frac{(X^T X)^{-1} \tilde{x}}{1 + \tilde{x}^T (X^T X)^{-1} \tilde{x}} (\tilde{y} - \tilde{x}^T \hat{w})$$

② Initialize $K = (X^T X)^{-1}$ and $\alpha = \hat{w}$ Use 3 & 4 to show that we can get $K = (X_{new}^T X_{new})^{-1}$ and $\alpha = \hat{w}_{new}$ after running the follow.

Algorithm 1: Online update

1. Receive observation $(\tilde{x}, \tilde{y})$
2. Set $\varepsilon \leftarrow K\tilde{x} / (1 + \tilde{x}^T K \tilde{x})$
3. Set $\alpha \leftarrow \alpha + \varepsilon (\tilde{y} - \tilde{x}^T \alpha)$
4. set $K \leftarrow K - \varepsilon \tilde{x}^T K$

$$\hat{w}_{new} = \hat{w} + \frac{(X^T X)^{-1} \tilde{x}}{1 + \tilde{x}^T (X^T X)^{-1} \tilde{x}} (y - \tilde{x}^T w) \quad \Big| \quad \varepsilon = \frac{K \tilde{x}}{1 + \tilde{x}^T K \tilde{x}}$$

Since $\alpha = \hat{w}$ and $\alpha = \hat{w}_{new}$

$$\alpha = \alpha + \varepsilon (y - \tilde{x}^T w)$$

$K_{new} = X_{new}^T X_{new}$ from ③

$$K_{new} = K - \frac{K \tilde{x} \tilde{x}^T K}{1 + \tilde{x}^T K x} = K - \left(\frac{K \tilde{x}}{1 + \tilde{x}^T K x} \right) \tilde{x}^T K$$

$$:= K - \varepsilon \tilde{x}^T K$$

③ Assume that n new data points arrive in sequential order and we would like to update $\hat{w}$ every time a new data point arrives Consider the following two methods.

Compare the total complexity computation (in terms of no. of multiplications) of methods a and b by showing the computational complexity of initialization complexity as n updates. Answers should be in terms of N, d, and n. Which method is efficient

a) First initialize $\hat{w}$ using the original N data points Then algorithm 1 once each of the new data point.

i) Initialize

$\hat{w} = (X^TX)^{-1}X^Ty$

$[X^TX] = d[\ ^N\ ]N[\ ^d\ ] = d[\ ^d\ ]$

$= \Theta(Nd^2)$

$K = (X^TX)^{-1} = \Theta(d^3)$

$X^Ty = d[\ ^N\ ][\ ^1\ ] = [\ ^d\ ]$

$= \Theta(Nd)$

$\therefore (X^TX)^{-1}X^Ty = d[\ ^d\ ][\ ^d\ ] \Rightarrow [d]$

$= \Theta(d^2)$

$\Rightarrow Nd^2 + d^3 + d^2$

$\Theta(Nd^2 + d^3)$

Update

ii) $\varepsilon = \frac{Kx}{1 + x^TKx}$

$Kx = d[\ ^d\ ][d] = \Theta(d^2)$

$= [\ ^{d^2}\ ]$

$= O(d^2)$

$x^TKx = [\ ^d\ ][\ ^d\ ]$

$= O(d)$

$1 + x^TKx = O(1)$

$= O(d^2 + d + 1) = O(d^2)$

Updating α

$\alpha + \varepsilon(y - x^T\alpha)$

$x^T\alpha = [\ ^d\ ][\ ^d\ ]$

$= O(d^2)$

Updating K

$K = \varepsilon x^TK$

$x^TK = [\ ^d\ ]d[\ ^d\ ]$

$= O(d^2)$

Total single update

$= O(d + d^2 + d^2)$

$= O(d^2)$

$\Rightarrow$ Total cost for update

$O(Nd^2 + d^3 + nd^2)$

Method b:

$$\vec{w}_{new} = (X_{new}^T X_{new})^{-1} X_{new}^T y_{new}$$

For i^{th} update, cost $O((N+i)d^2)$

for n values:

$$\sum_{i=1}^{n} O((N+i)d^2) = O\left(nd^2 + \frac{n(n+1)}{2} d^2\right) = O(nd^2 + n^2d^2)$$

$(X_{new}^T X_{new})^{-1}$

↳ for invers of a $d \times d$ $\Rightarrow O(d^3)$

↳ for n updates $\Rightarrow O(nd^3)$

$(X_{new}^T y_{new})$

↳ for each update $O((N+i)d)$

↳ for n updates.

$$\sum_{i=1}^{n} O((N+i)d) = O\left(nd + \frac{n(n+1)}{2} d\right) = O(nd + n^2d)$$

↳ $\vec{w}_{new} = O(d^2)$ per update

↳ for n updates $= O(nd^2)$

Total cost:

$$O(nd^2 + n^2d^2 + nd^3 + nd + n^2d)$$

$$= O(n^2d^2 + nd^3)$$

Comparison

↳ method a is efficient since its complexity scales linearly for larger values of n, for update $O(nd^2)$ while for method b it scales quadratically with n: $O(n^2d^2)$.

Import necessary libraries

```
import numpy as np
import matplotlib.pyplot as plt
```

==========

Step 1: Generate Synthetic Data

==========

```
np.random.seed(42)
N = 500  # Total data points
d = 12   # Number of features
train_ratio = 0.7
val_ratio = 0.15

# Generate feature matrix and true weights
X = np.random.normal(0, 1, (N, d))
true_weights = np.linspace(1, 5, d)  # Linearly spaced true weights
epsilon = np.random.normal(0, 0.5, N)  # Noise
y = X @ true_weights + epsilon  # Generate target values

# Split data into train, validation, and test sets
train_size = int(N * train_ratio)
val_size = int(N * val_ratio)
test_size = N - train_size - val_size

X_train, X_val, X_test = X[:train_size],
X[train_size:train_size+val_size], X[train_size+val_size:]
y_train, y_val, y_test = y[:train_size],
y[train_size:train_size+val_size], y[train_size+val_size:]
```

==========

Step 2: Ridge Regression Functions

==========

```
def ridge_loss(w, X, y, lam):
    """Calculate the ridge regression loss."""
    residuals = y - X @ w
    return np.sum(residuals**2) + lam * np.sum(w**2)

def ridge_gradient(w, X, y, lam):
    """Calculate the gradient of the ridge regression loss."""
    residuals = y - X @ w
    grad = - 2 * X.T @ residuals + 2* lam * w
    grad = grad/len(y)
```

```
    return grad

def gradient_descent(loss_fn, grad_fn, w_init, X, y, lam, lr=0.01,
tol=1e-6, max_iters=1000):
    """Perform gradient descent to minimize the ridge regression
loss."""
    w = w_init
    for i in range(max_iters):
        grad = grad_fn(w, X, y, lam)
        w_new = w - lr * grad
        if np.linalg.norm(w_new - w, ord=2) < tol:
            break
        w = w_new
    return w
```

==========

Step 3: Variance and Bias Calculation

==========

```
def calculate_bias_variance(X_train, y_train, X_val, y_val, lambdas,
num_datasets=20,
                            sub_sample_size=50):
    """
    Calculate the bias and variance for ridge regression models
trained on multiple datasets.
    """
    biases, variances = [], []
    for lam in lambdas:
        predictions = []
        for _ in range(num_datasets):
            # Sample with replacement
            indices = np.random.choice(len(X_train),
size=sub_sample_size, replace=True)
            X_sample, y_sample = X_train[indices], y_train[indices]

            # Train ridge regression
            w_init = np.zeros(d)
            w = gradient_descent(ridge_loss, ridge_gradient, w_init,
X_sample, y_sample, lam)

            # Predict on validation data
            predictions.append(X_val @ w)

        # Average predictions
        predictions = np.array(predictions)
        mean_prediction = np.mean(predictions, axis=0)
        bias = np.mean((mean_prediction - y_val)**2)
```

```
        variance = np.mean(np.var(predictions, axis=0))

        biases.append(bias)
        variances.append(variance)

    return biases, variances

lambdas = [a * 10**b for a in range(1, 10) for b in range(-5, 3)]  # For λ values
lambdas.sort()
```

==========

Step 4: Plotting Functions

==========

```
# Empty sections for students to complete
def plot_coefficients_vs_lambda():
    """Plot the learned coefficients for different lambda values."""
    coeffs = []
    for lam in lambdas:
        w_init = np.zeros(d)
        w_opt = gradient_descent(ridge_loss, ridge_gradient, w_init, X_train, y_train, lam)
        coeffs.append(w_opt)

    coeffs = np.array(coeffs)  # Convert to numpy array for plotting
    plt.figure(figsize=(8, 5))
    plt.plot(lambdas, coeffs)
    plt.xscale("log")  # Log scale for lambda
    plt.xlabel("Lambda (log scale)")
    plt.ylabel("Coefficient Values")
    plt.title("Coefficient Values vs. Lambda")
    plt.grid()
    plt.savefig("coefficients_vs_lambda.png")
    plt.show()


def plot_rmse_vs_lambda():
    """Plot RMSE on validation data vs lambda."""
    rmse_values = []
    for lam in lambdas:
        w_init = np.zeros(d)
        w_opt = gradient_descent(ridge_loss, ridge_gradient, w_init, X_train, y_train, lam)
        y_pred = X_val @ w_opt
        rmse = np.sqrt(np.mean((y_val - y_pred) ** 2))
        rmse_values.append(rmse)
```

```
    optimal_lambda = lambdas[np.argmin(rmse_values)]  # Find lambda
that minimizes RMSE


    plt.figure(figsize=(8, 5))
    plt.plot(lambdas, rmse_values)
    plt.plot(optimal_lambda, min(rmse_values), 'ro')  # Highlight
optimal lambda
    plt.xscale("log")
    plt.xlabel("Lambda (log scale)")
    plt.ylabel("Validation RMSE")
    plt.title(f"Validation RMSE vs. Lambda (Best λ =
{optimal_lambda:.2e})")
    plt.grid()
    plt.savefig("rmse_vs_lambda.png")
    plt.show()

    return optimal_lambda  # Return best lambda

# def plot_predicted_vs_true():
def plot_predicted_vs_true(lambda_opt):
    """Plot predicted vs true values using optimal lambda."""
    w_init = np.zeros(d)
    w_opt = gradient_descent(ridge_loss, ridge_gradient, w_init,
np.vstack((X_train, X_val)), np.hstack((y_train, y_val)), lambda_opt)

    y_pred = X_test @ w_opt  # Predict test values

    plt.figure(figsize=(6, 6))
    plt.scatter(y_test, y_pred, alpha=0.7)
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
'r', linestyle="--")  # 45-degree reference line
    plt.xlabel("True Values")
    plt.ylabel("Predicted Values")
    plt.title("Predicted vs True Values")
    plt.grid()
    plt.savefig("predicted_vs_true.png")
    plt.show()



# def plot_bias_variance_tradeoff():
def plot_bias_variance_tradeoff():
    """Plot bias and variance vs lambda."""
    biases, variances = calculate_bias_variance(X_train, y_train,
X_val, y_val, lambdas)

    plt.figure(figsize=(8, 5))
    plt.plot(lambdas, biases, label="Bias")
    plt.plot(lambdas, variances, label="Variance")
```

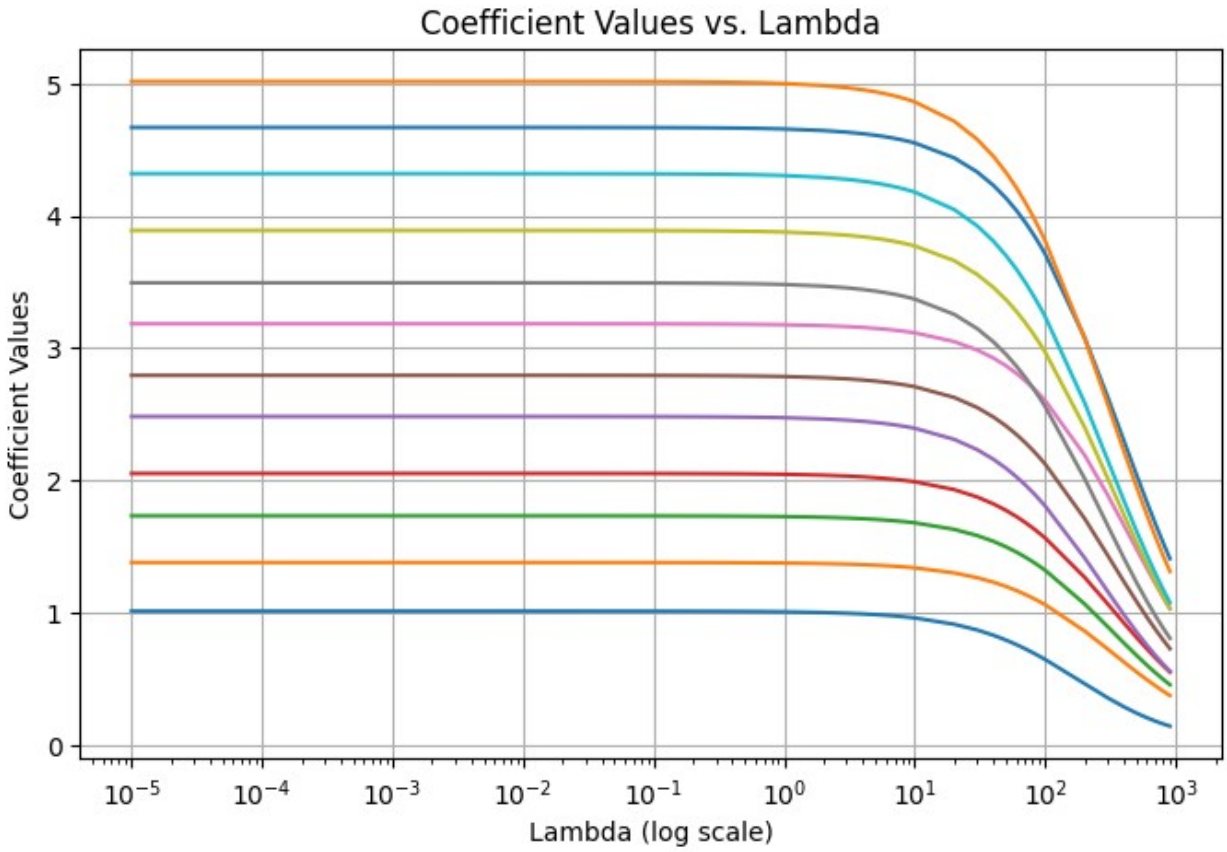
```
    plt.xscale("log")
    plt.xlabel("Lambda (log scale)")
    plt.ylabel("Variance")
    plt.title("Variance Tradeoff")
    plt.legend()
    plt.grid()
    plt.savefig("bias_variance_tradeoff.png")
    plt.show()
```

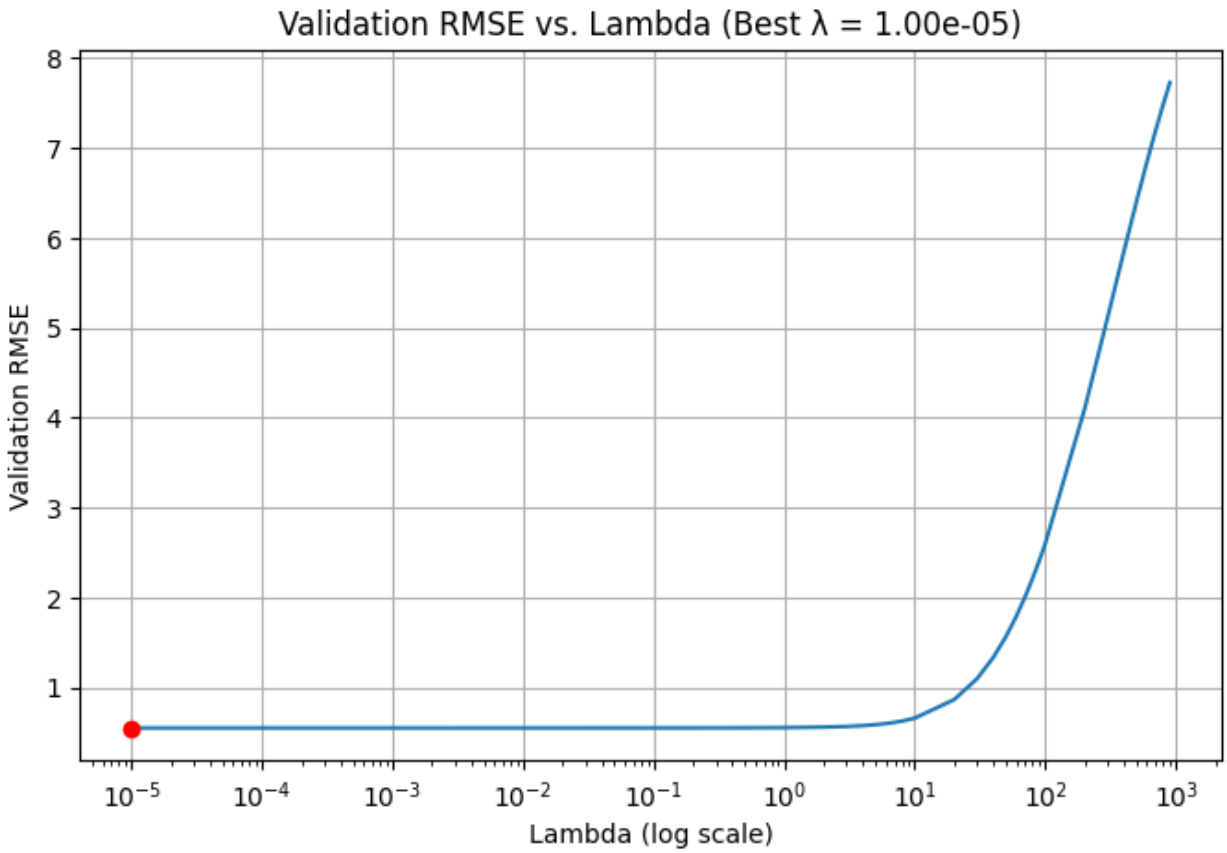
==========

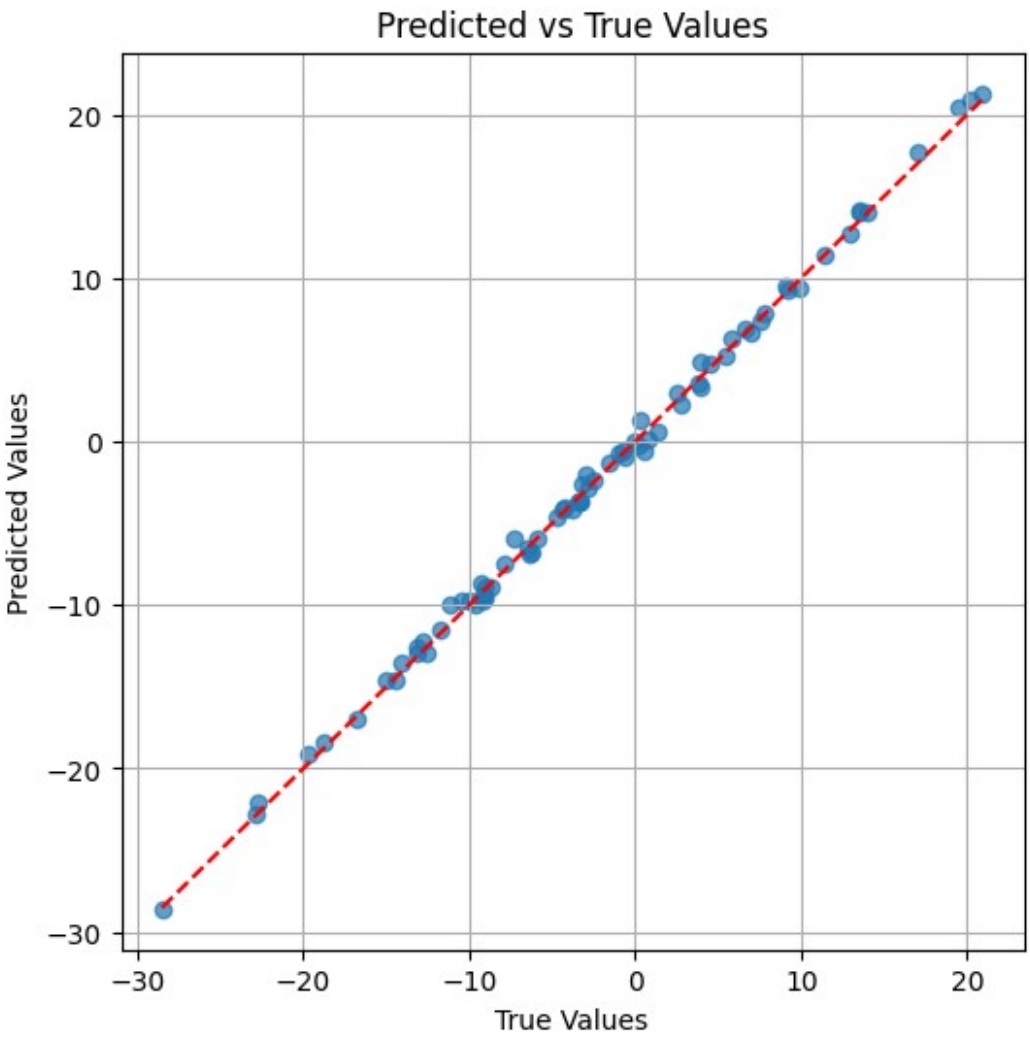
Step 5: Main Execution

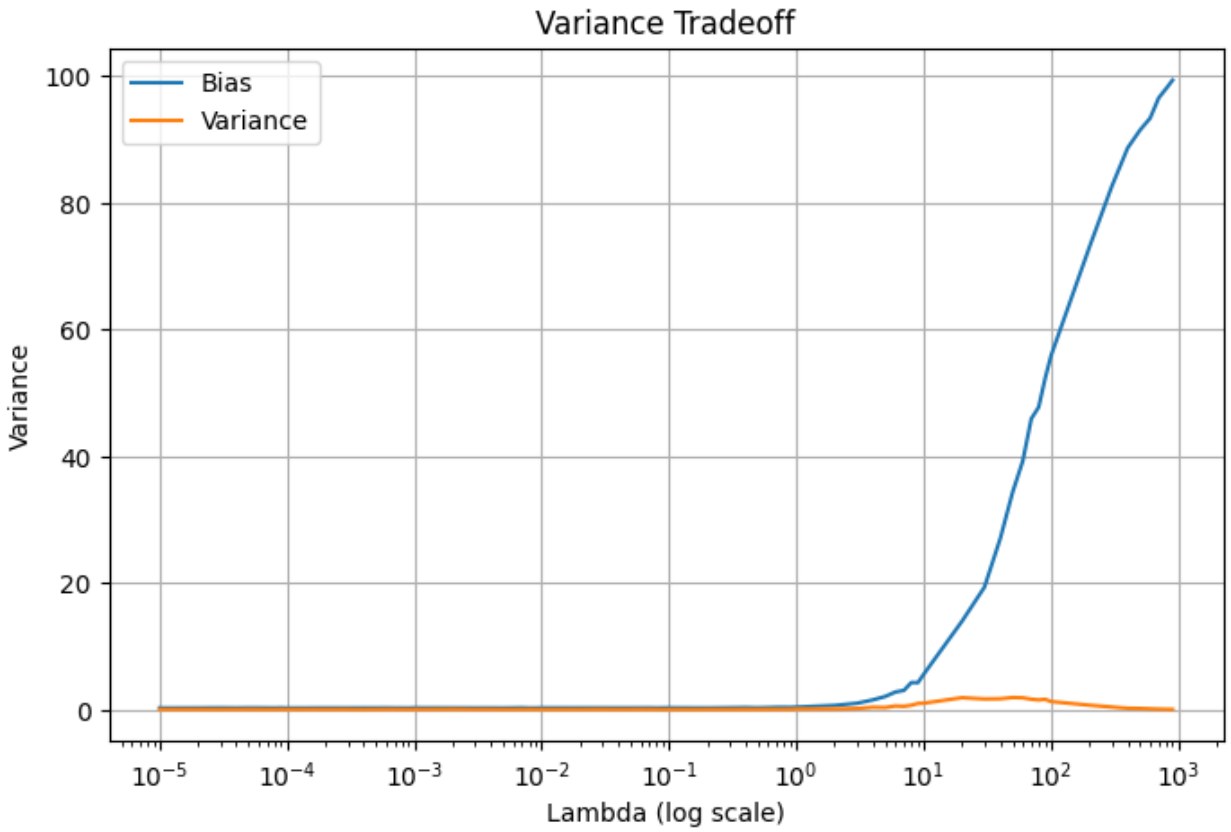
==========

```
plot_coefficients_vs_lambda()
best_lambda = plot_rmse_vs_lambda()
plot_predicted_vs_true(best_lambda)
plot_bias_variance_tradeoff()
```




Validation RMSE vs. Lambda (Best λ = 1.00e-05)
Validation RMSE
Lambda (log scale)
8
7
6
5
4
3
2
1
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}


Predicted vs True Values
Predicted Values
True Values
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20

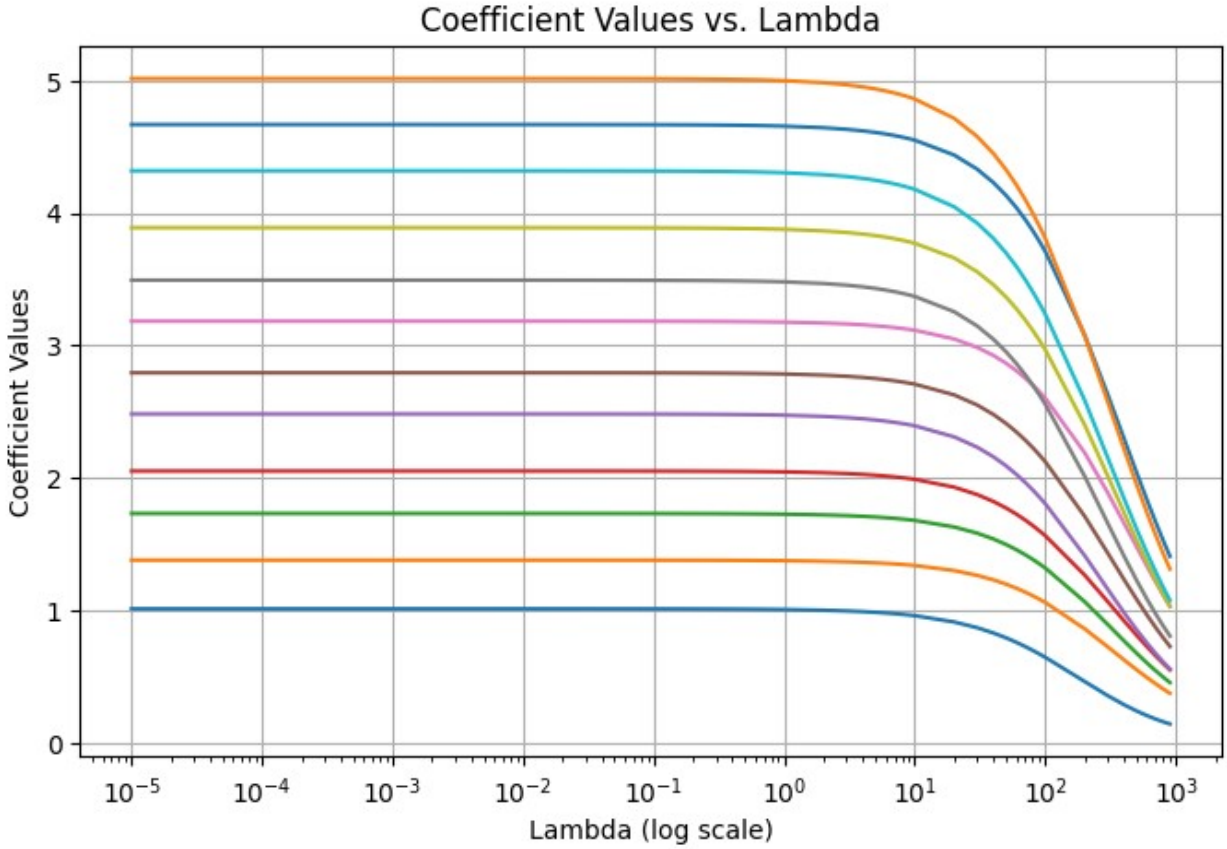

Variance Tradeoff
Bias
Variance
100
80
60
40
20
0
Variance
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Lambda (log scale)

==========

Analysis

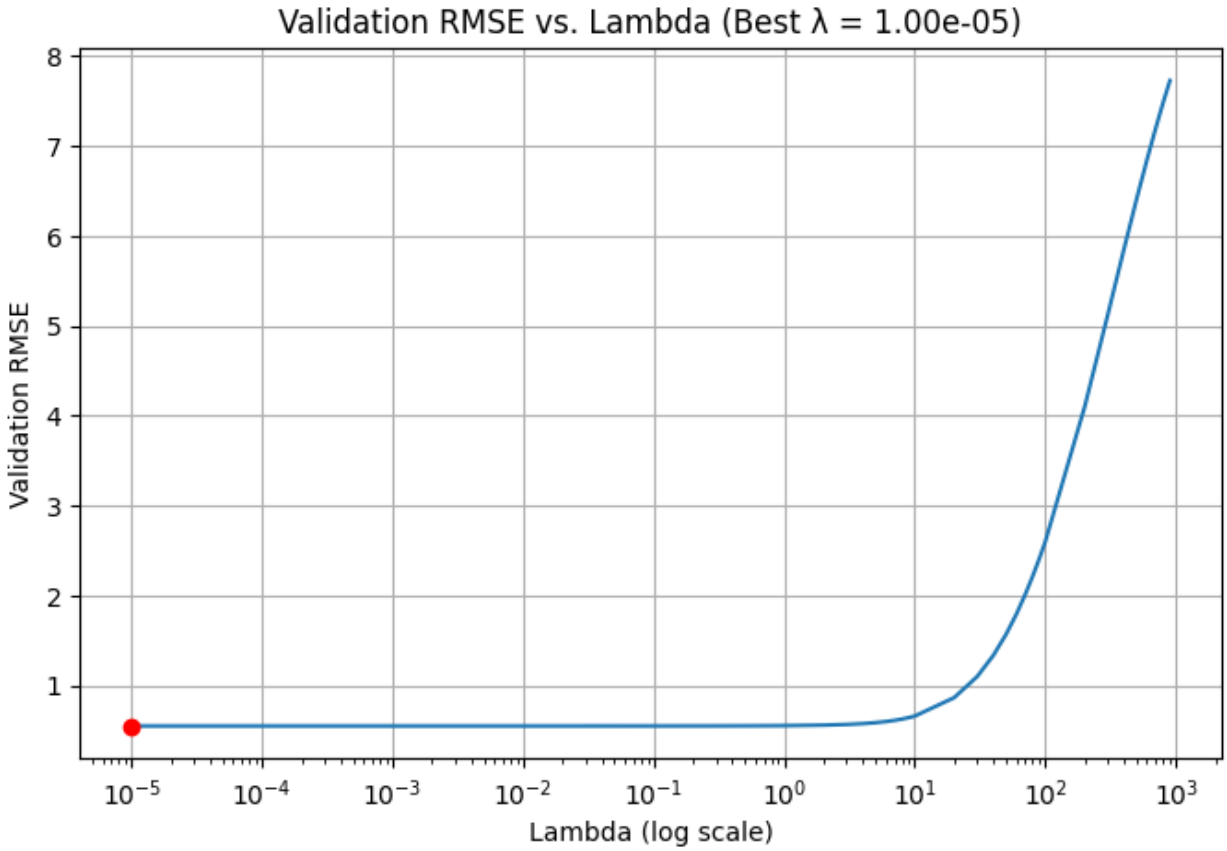
==========

1. **How coefficients behave as λ increases.**



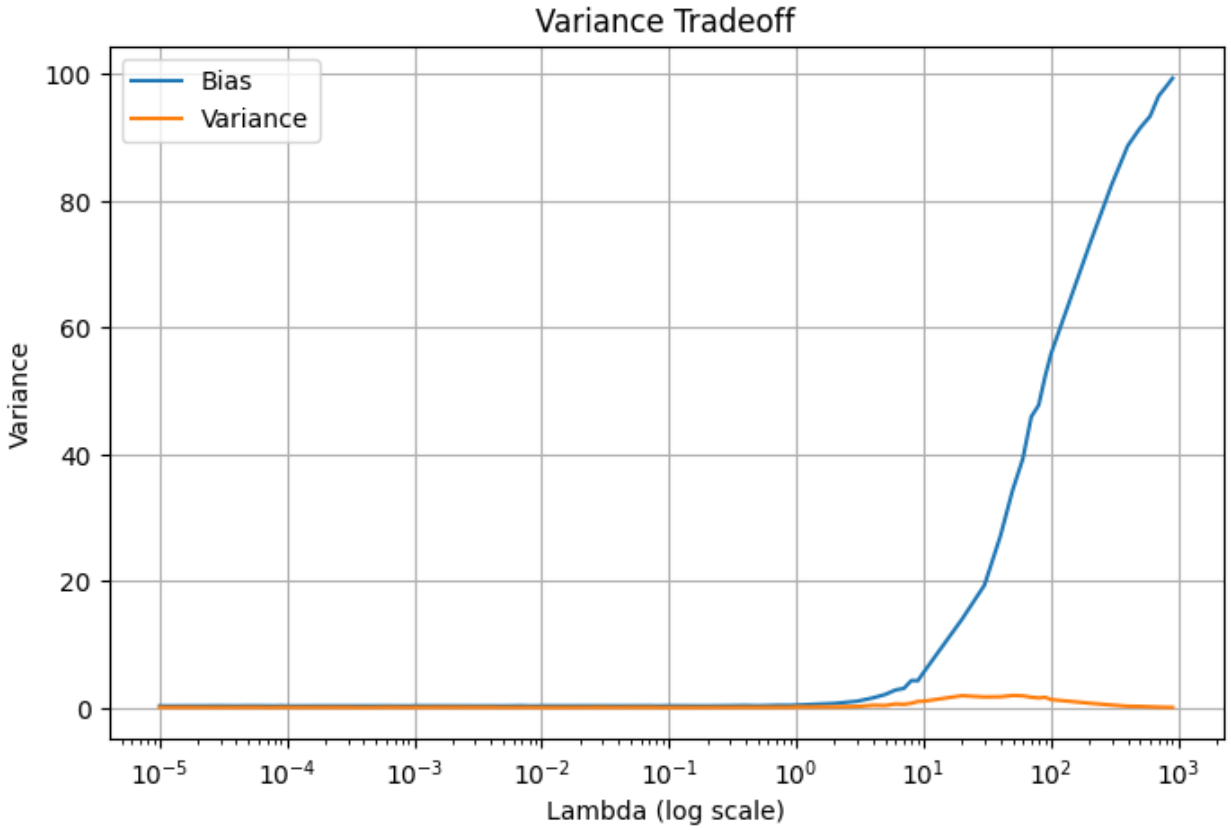
- The small values of λ the coefficients take on their unrestricted values, this indicates that the model behaves like ordinary least squares (OLS), while as λ increases, the coefficients tends to go towards zero. This is an effect of the ridge regression penalty that discourages large coeeficint values to prevent overfitting.
- For the high values of λ, all coefficients approach zero, this indicates that the model is being constrainde which leads to underfitting.

2. **The trade-off between RMSE and λ.**



- At very small λ, RMSE is low, ths indicates that the model fits the data well. As it increases, RMSE remains low for a while, this shows the region where the model generalizes well.However, as λ increases, RMSE starts increasing sharply, indicating underfitting since the model becomes too simple and fails to capture the data complexity.
- The red dot marks indicates an optimal λ, that achieves the lowest RMSE on the validation set. This represents the best balance between bias and variance.

3. **Observations from the bias-variance trade-off plot.**



- For very small values of λ, variance is high, as bias remains low. This could indicate to a very flexible model that might lead to overfitting the training data. However, as λ increases, variance decreases because the model becomes more constrained, reducing sensitivity to noise.
- Bias also starts increasing at higher λ values as the model becomes too simple to capture the underlying pattern.
- The most ideal λ is in the region where variance is controlled without significantly increasing bias.

```
# Please complete this field.
```