Homework #1

ECE 461/661: Introduction to Machine Learning for Engineers
Prof. Carlee Joe-Wong and Prof. Gauri Joshi

Due: Friday, Jan 31, 2025
8:59 pm PT/11:59 pm ET/Saturday, Feb 1, 2025 at 6:59 am CAT

Please remember to show your work for all problems and to write down the names of any students that you collaborate with. The full collaboration and grading policies are available on the course website: `https://www.andrew.cmu.edu/course/18-661/`. You are strongly encouraged (but not required) to use Latex to typeset your solutions.

Your solutions should be uploaded to Gradescope (`https://www.gradescope.com/`) in PDF format by the deadline. We will not accept hardcopies. If you choose to hand-write your solutions, please make sure the uploaded copies are legible. Gradescope will ask you to identify which page(s) contain your solutions to which problems, so make sure you leave enough time to finish this before the deadline. We will give you a 30-minute grace period to upload your solutions in case of technical problems.

1 Probability *[10 points]*

1. *[5 points]* Suppose W is a Gaussian random variable with distribution $\mathcal{N}(\mu, \sigma^2)$ and U a uniform random variable over the interval $[a, b]$. Assuming that W and U are independent, what is the expected value $\mathbb{E}[Z]$ and variance $\mathrm{Var}[Z]$ of $Z = 3W + 2U$?

Solution: We are tasked with computing the expected value and variance of $Z = 3W + 2U$, where $W \sim \mathcal{N}(\mu, \sigma^2)$ and $U \sim \mathcal{U}[a, b]$, and W and U are independent.

- **Expected Value:**

  Using the linearity of expectation:

  $$\mathbb{E}[Z] = \mathbb{E}[3W + 2U] = 3\mathbb{E}[W] + 2\mathbb{E}[U].$$

  For $W \sim \mathcal{N}(\mu, \sigma^2)$, the expected value is:

  $$\mathbb{E}[W] = \mu.$$

  For $U \sim \mathcal{U}[a, b]$, the expected value is:

  $$\mathbb{E}[U] = \frac{a+b}{2}.$$

  Substituting these values:

  $$\mathbb{E}[Z] = 3\mu + 2\left(\frac{a+b}{2}\right) = 3\mu + (a+b).$$

- **Variance:**

  Using the independence of W and U, the variance is:

$$\text{Var}[Z] = \text{Var}[3W + 2U] = \text{Var}[3W] + \text{Var}[2U].$$

For $W \sim \mathcal{N}(\mu, \sigma^2)$:

$$\text{Var}[3W] = 3^2 \cdot \text{Var}[W] = 9\sigma^2.$$

For $U \sim \mathcal{U}[a, b]$, the variance is:

$$\text{Var}[U] = \frac{(b-a)^2}{12}.$$

Thus:

$$\text{Var}[2U] = 2^2 \cdot \text{Var}[U] = 4 \cdot \frac{(b-a)^2}{12} = \frac{(b-a)^2}{3}.$$

Substituting these values:

$$\text{Var}[Z] = 9\sigma^2 + \frac{(b-a)^2}{3}.$$

2. *[5 points]* Consider the following joint distribution between the random variable X, which takes values T or F, and the random variable Y, which takes values a, b, c, or d.

$P(X,Y)$		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. *[2 points]* What is the marginal distribution P_Y, that is, what are $\Pr(Y = a)$, $\Pr(Y = b)$, $\Pr(Y = c)$, and $\Pr(Y = d)$?

Solution:

$$\Pr(Y = a) = 0.1 + 0.1 = 0.2,$$
$$\Pr(Y = b) = 0.2 + 0.1 = 0.3,$$
$$\Pr(Y = c) = 0.1 + 0.2 = 0.3,$$
$$\Pr(Y = d) = 0.1 + 0.1 = 0.2.$$

b. *[3 points]* What is $\Pr(X = T | Y \in \{b, c, d\})$, i.e., the probability that X is T given that Y takes a value in $\{b, c, d\}$?

Solution: First, compute $\Pr(Y \in \{b, c, d\})$:

$$\Pr(Y \in \{b, c, d\}) = \Pr(Y = b) + \Pr(Y = c) + \Pr(Y = d) = 0.3 + 0.3 + 0.2 = 0.8.$$

Compute $\Pr(X = T, Y \in \{b, c, d\})$:

$$\Pr(X = T, Y \in \{b, c, d\})$$

$$= \Pr(X = T, Y = b) + \Pr(X = T, Y = c) + \Pr(X = T, Y = d) = 0.2 + 0.1 + 0.1 = 0.4.$$

Finally, compute the conditional probability:

$$\Pr(X = T | Y \in \{b, c, d\}) = \frac{\Pr(X = T, Y \in \{b, c, d\})}{\Pr(Y \in \{b, c, d\})} = \frac{0.4}{0.8} = 0.5.$$

2 Linear Algebra *[10 points]*

1. *[4 points]* Let $\boldsymbol{A}_k \in \mathbb{R}^{n\times n}$ for $k = 1, \ldots, K$ such that $\boldsymbol{A}_k = \boldsymbol{A}_k^\top$, i.e., each $\boldsymbol{A}_k$ is a symmetric, n-dimensional square matrix. Suppose all $\boldsymbol{A}_k$ have the exact same set of eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n$ with the corresponding eigenvalues $\alpha_{k1}, \cdots, \alpha_{kn}$ for each $\boldsymbol{A}_k$. Write down the eigenvectors and their corresponding eigenvalues for the following matrices:

   a. *[2 points]* $\boldsymbol{C} = \sum_{k=1}^{K} \boldsymbol{A}_k$

   b. *[2 points]* $\boldsymbol{D} = \boldsymbol{A}_i^{-1}\boldsymbol{A}_j\boldsymbol{A}_i$, where $i \neq j$ and $i, j \in \{1, 2, \ldots, K\}$. Here we assume $\boldsymbol{A}_i$ is invertible.

Solution:

a. Eigenvalues of $\boldsymbol{C}$ are $\sum_{k=1}^{K} \alpha_{k1}, \cdots, \sum_{k=1}^{K} \alpha_{kn}$.

b. Eigenvalues of $\boldsymbol{D}$ are $\alpha_{j1}, \cdots, \alpha_{jn}$.

$\boldsymbol{C}$ and $\boldsymbol{D}$ have the same set of eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_n$

2. *[6 points]* Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ be given, and $\mathrm{col}(\boldsymbol{A})$ be the column space of $\boldsymbol{A}$. For a given value of m, under what conditions on $\boldsymbol{b}$, $\mathrm{col}(\boldsymbol{A})$, and $\mathrm{rank}(\boldsymbol{A})$ will the equation $\boldsymbol{Ax} = \boldsymbol{b}$ have

   a. *[2 points]* no solution?

   b. *[2 points]* exactly one solution?

   c. *[2 points]* infinitely many solutions?

Solution:

a. $\boldsymbol{b} \notin \mathrm{col}(\boldsymbol{A})$

b. $\boldsymbol{b} \in \mathrm{col}(\boldsymbol{A})$ and $\mathrm{rank}(\boldsymbol{A}) = n$

c. $\boldsymbol{b} \in \mathrm{col}(\boldsymbol{A})$ and $\mathrm{rank}(\boldsymbol{A}) < n$

3 Matrix Calculus*[10 points]*

Find the first derivative of the following functions with respect to $\boldsymbol{X}$. Before you attempt the questions below, you are encouraged to review Matrix calculus Wikipedia page (https://en.wikipedia.org/wiki/Matrix_calculus). Note that in this problem, we follow the convention that the derivative of a scalar function $f(\boldsymbol{X})$ with respect to $\boldsymbol{X}$ should have the same dimension as $\boldsymbol{X}$.

a. *[4 points]* $f(\boldsymbol{X}) = tr(\boldsymbol{X}\boldsymbol{X}^T)$, where $\boldsymbol{X} \in \mathbb{R}^{n\times n}$ and tr is the trace of a square matrix.

b. *[2 points]* $f(\boldsymbol{X}) = \boldsymbol{a}^T\boldsymbol{X}\boldsymbol{b}$, where $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{a} \in \mathbb{R}^m$ and $\boldsymbol{b} \in \mathbb{R}^n$.

c. *[4 points]* $f(\boldsymbol{X}) = \|\boldsymbol{X}\boldsymbol{b}\|^2$, where $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$.

Solution:

a. $\nabla f(\mathbf{X}) = 2\boldsymbol{X}$

b. $\nabla f(\mathbf{X}) = \boldsymbol{a}\boldsymbol{b}^\top$

c. $f(\boldsymbol{X}) = \|\boldsymbol{X}\boldsymbol{b}\|^2$

$$f(\boldsymbol{X}) = (\boldsymbol{X}\boldsymbol{b})^\top(\boldsymbol{X}\boldsymbol{b}) = \mathrm{tr}((\boldsymbol{X}\boldsymbol{b})(\boldsymbol{X}\boldsymbol{b})^\top)$$

The derivative of $\mathrm{tr}((\boldsymbol{X}\boldsymbol{b})(\boldsymbol{X}\boldsymbol{b})^\top)$ is:

$$2(\boldsymbol{X}\boldsymbol{b})\boldsymbol{b}^\top$$

$$\nabla f(\boldsymbol{X}) = 2(\boldsymbol{X}\boldsymbol{b})\boldsymbol{b}^\top.$$

4 MLE-MAP *[20 points]*

Suppose y is a random variable, and assume $\lambda > 0$ is a scalar parameter. The random variable y follows::

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

where Poisson(λ) denotes the Poisson distribution with rate λ.

1. *[10 points]* **Maximum likelihood estimation:**

   a. *[2 points]* Write down the PMF (probability mass function) of $y|\lambda$.

   b. *[4 points]* Assume N independent obeservations $y_1, y_2, \ldots, y_N$ are drawn, where $y_n \sim \text{Poisson}(\lambda)$ for $n = 1, 2, \ldots, N$. Derive the joint PMF $P(y_1, \ldots, y_N|\lambda)$.

   c. *[4 points]* Write down the **negative** log-likelihood function of the joint PMF derived in the previous problem **b.** and simplify it into a form that can be minimized. (Hint: remove the terms that do not have λ.)

Solution:

a. The PMF of $y|\lambda$ is:

$$P(y|\lambda) = \frac{\lambda^y e^{-\lambda}}{y!}.$$

b. Assuming independence across N samples:

$$P(y_1, \ldots, y_N|\lambda) = \prod_{n=1}^{N} P(y_n|\lambda),$$

so:

$$P(y_1, \ldots, y_N|\lambda) = \prod_{n=1}^{N} \frac{\lambda^{y_n} e^{-\lambda}}{y_n!}.$$

c. The negative log-likelihood is:

$$-\log P(y_1,\ldots,y_N|\lambda) = -\sum_{n=1}^{N} \left(y_n \log\lambda - \lambda - \log(y_n!)\right).$$

Ignoring the constant term $\log(y_n!)$, the objective function to minimize is:

$$J(\lambda) = N\lambda - \left(\sum_{n=1}^{N} y_n\right)\log\lambda.$$

2. *[10 points]* **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. *[5 points]* Write the joint distribution $P(y_1,\ldots,y_N,\lambda)$ where each $y_n \sim \text{Poisson}(\lambda)$ is drawn independently as above.

b. *[5 points]* Write the negative logarithm of $P(y_1,\ldots,y_N,\lambda)$ and simplify it into a form that can be minimized to find the MAP estimate $\hat{\lambda}$. (Hint: remove the terms that do not have λ.)

Solution:

a. By Bayes' rule:

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

so:

$$P(y_1,\ldots,y_N,\lambda) = \left(\prod_{n=1}^{N}\frac{\lambda^{y_n}e^{-\lambda}}{y_n!}\right)\cdot\frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}.$$

b. The negative log of $P(y_1,\ldots,y_N,\lambda)$ is:

$$\begin{aligned}-\log P(y_1,\ldots,y_N,\lambda) &= -\sum_{n=1}^{N}\left(y_n\log\lambda - \lambda - \log(y_n!)\right) - \left(\alpha\log\beta - \log\Gamma(\alpha) + (\alpha-1)\log\lambda - \beta\lambda\right).\\ &= N\lambda + \beta\lambda - \left(\sum_{n=1}^{N} y_n + \alpha - 1\right)\log\lambda + \text{const},\end{aligned}$$

where const includes terms unrelated to λ.

Ignoring constants, the objective function to minimize is:

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

5 Linear Regression with Regularization *[10 points]*

Consider a dataset with N samples $(\boldsymbol{x}_i, y_i)$, where:

$$y_i = \boldsymbol{x}_i^\top \boldsymbol{w} + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2),$$

and $\boldsymbol{w} \in \mathbb{R}^d$ is the weight vector, ε_i is Gaussian noise, and $\boldsymbol{x}_i \in \mathbb{R}^d$.

To enforce smoothness in the weights $\boldsymbol{w}$, we introduce a regularizer based on the differences between adjacent weights. This results in the following optimization problem:

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2 + \mu\|\boldsymbol{D}\boldsymbol{w}\|_2^2, \tag{1}$$

where $\lambda, \mu > 0$ are regularization parameters and $\|\boldsymbol{D}\boldsymbol{w}\|_2^2 = \sum_{i=2}^{d-1}(2w_i - w_{i-1} - w_{i+1})^2$. Answer the following:

1. *[2 points]* Find $\boldsymbol{D} \in \mathbb{R}^{(d-2)\times d}$ such that $\|\boldsymbol{D}\boldsymbol{w}\|_2^2 = \sum_{i=2}^{d-1}(2w_i - w_{i-1} - w_{i+1})^2$. Write $\boldsymbol{D}$ explicitly. Note that $\|\cdot\|_2$ denotes the usual ℓ_2, or Euclidean, norm.

Solution: The matrix $\boldsymbol{D}$ is given by:

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

2. *[2 points]* Derive the closed-form solution for $\boldsymbol{w}^*$, the minimizer of $L(\boldsymbol{w})$.

Solution: The gradient of $L(\boldsymbol{w})$ is:

$$\nabla L(\boldsymbol{w}) = 2\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + 2\lambda\boldsymbol{w} + 2\mu\boldsymbol{D}^\top\boldsymbol{D}\boldsymbol{w}.$$

Setting $\nabla L(\boldsymbol{w}) = 0$, we get:

$$(\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I} + \mu\boldsymbol{D}^\top\boldsymbol{D})\boldsymbol{w} = \boldsymbol{X}^\top\boldsymbol{y}.$$

Thus,

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I} + \mu\boldsymbol{D}^\top\boldsymbol{D})^{-1}\boldsymbol{X}^\top\boldsymbol{y}.$$

3. *[4 points]* Denote the minimizer of problem (1) by $\boldsymbol{w}^*$. Consider the same problem without the "smooth" regularization:

$$\min_{\boldsymbol{w}} L_{\text{ridge}}(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2.$$

Let the minimizer of L_{ridge} be $\boldsymbol{w}^*_{\text{ridge}}$. Show that the "smooth" regularization is effective by proving $\|\boldsymbol{D}\boldsymbol{w}^*\|_2^2 \leq \|\boldsymbol{D}\boldsymbol{w}^*_{\text{ridge}}\|_2^2$. (Hint: You can solve these problems explicitly, but you do not have to. Consider the relationship between a) $L(\boldsymbol{w}^*)$ and $L(\boldsymbol{w}^*_{\text{ridge}})$ and b) $L_{\text{ridge}}(\boldsymbol{w}^*_{\text{ridge}})$ and $L_{\text{ridge}}(\boldsymbol{w}^*)$.)

Solution: We have $L_{\text{ridge}}(\boldsymbol{w}^*_{\text{ridge}}) \le L_{\text{ridge}}(\boldsymbol{w}^*)$ according to $\boldsymbol{w}^*_{\text{ridge}}$'s definition, or

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}^*_{\text{ridge}}\|_2^2 + \lambda\|\boldsymbol{w}^*_{\text{ridge}}\|_2^2 \le \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}^*\|_2^2 + \lambda\|\boldsymbol{w}^*\|_2^2.$$

Similarly, we have $L(\boldsymbol{w}^*) \le L(\boldsymbol{w}^*_{\text{ridge}})$, which states

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}^*_{\text{ridge}}\|_2^2 + \lambda\|\boldsymbol{w}^*_{\text{ridge}}\|_2^2 + \|\boldsymbol{D}\boldsymbol{w}^*_{\text{ridge}}\|_2^2 \ge \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}^*\|_2^2 + \lambda\|\boldsymbol{w}^*\|_2^2 + \|\boldsymbol{D}\boldsymbol{w}^*\|_2^2.$$

Subtracting the first inequality from the second finishes the proof.

4. *[2 points]* Briefly (in 1-3 sentences) explain how smooth regularization affects the bias and variance of the learned model $\boldsymbol{w}$.

Solution: Smooth regularization increases the bias because it enforces additional constraints on $\boldsymbol{w}$, reducing flexibility. However, it decreases the variance by preventing the model from fitting noise in the data. This trade-off can improve generalization performance.

6 Online Update *[10 points]*

In the standard linear regression model, we consider a model for which the observed response variable y is the prediction $\boldsymbol{x}^\top\boldsymbol{w}$ perturbed by noise, namely

$$y = \boldsymbol{w}^\top\boldsymbol{x} + \varepsilon$$

where ε is a Gaussian random variable with mean 0 and variance σ^2 and $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^d$. Suppose our training dataset contains N observations of d-dimensional features. We denote the corresponding feature matrix by $\boldsymbol{X} \in \mathbb{R}^{N\times d}$, and the labels associated with each observation are denoted by $\boldsymbol{y} \in \mathbb{R}^N$. We have shown in the class that the MLE (maximum likelihood estimate) of $\boldsymbol{w}$ is given by

$$\widehat{\boldsymbol{w}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}. \tag{2}$$

Suppose that we get a new observation $(\tilde{\boldsymbol{x}}, \tilde{y})$ and would like to compute the updated MLE $\widehat{\boldsymbol{w}}_{\text{new}}$ after adding this observation to our dataset. Now $\boldsymbol{X}_{\text{new}} = \begin{bmatrix}\boldsymbol{X}\\ \tilde{\boldsymbol{x}}^\top\end{bmatrix} \in \mathbb{R}^{(N+1)\times d}$ and $\boldsymbol{y}_{\text{new}} = \begin{bmatrix}\boldsymbol{y}\\ \tilde{y}\end{bmatrix} \in \mathbb{R}^{N+1}$. If we simply apply $\widehat{\boldsymbol{w}}_{\text{new}} = (\boldsymbol{X}_{\text{new}}^\top\boldsymbol{X}_{\text{new}})^{-1}\boldsymbol{X}_{\text{new}}^\top\boldsymbol{y}_{\text{new}}$, we will need to solve the normal equations for $\widehat{\boldsymbol{w}}_{\text{new}}$ again, which is inefficient. In this problem, we will derive a better algorithm for updating the parameters. The main idea is that the inverse of $\boldsymbol{X}_{\text{new}}^\top\boldsymbol{X}_{\text{new}}$, which is a low rank correction of $\boldsymbol{X}^\top\boldsymbol{X}$, can be computed by doing a low rank correction to the inverse of $\boldsymbol{X}^\top\boldsymbol{X}$.

Note that $\boldsymbol{X}_{\text{new}}^\top\boldsymbol{X}_{\text{new}} = \boldsymbol{X}^\top\boldsymbol{X} + \tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^\top$. Using the Sherman–Morrison–Woodbury formula, we can show that

$$(\boldsymbol{X}_{\text{new}}^\top\boldsymbol{X}_{\text{new}})^{-1} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1} - \frac{(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^\top(\boldsymbol{X}^\top\boldsymbol{X})^{-1}}{1 + \tilde{\boldsymbol{x}}^\top(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}. \tag{3}$$

Since $\boldsymbol{X}_{\text{new}}^\top\boldsymbol{y}_{\text{new}} = \boldsymbol{X}^\top\boldsymbol{y} + \tilde{\boldsymbol{x}}\tilde{y}$, we have

$$\begin{aligned}\widehat{\boldsymbol{w}}_{\text{new}} &= (\boldsymbol{X}_{\text{new}}^\top\boldsymbol{X}_{\text{new}})^{-1}\boldsymbol{X}_{\text{new}}^\top\boldsymbol{y}_{\text{new}}\\ &= \left[(\boldsymbol{X}^\top\boldsymbol{X})^{-1} - \frac{(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^\top(\boldsymbol{X}^\top\boldsymbol{X})^{-1}}{1 + \tilde{\boldsymbol{x}}^\top(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}\right](\boldsymbol{X}^\top\boldsymbol{y} + \tilde{\boldsymbol{x}}\tilde{y}). \end{aligned}\tag{4}$$

Answer the following:

1. *[3 points]* Derive the formula for $\widehat{\boldsymbol{w}}_{\text{new}}$ using the Sherman–Morrison formula. Your formula should express $\widehat{\boldsymbol{w}}_{\text{new}}$ explicitly in terms of $\widehat{\boldsymbol{w}}$.

Solution: Using the Sherman–Morrison formula, we can compute $(\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}})^{-1}$ as:

$$(\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}})^{-1} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} - \frac{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}}{1+\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}.$$

Substituting $(\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}})^{-1}$ and $\boldsymbol{X}_{\text{new}}^\top \boldsymbol{y}_{\text{new}}$ into the formula for $\widehat{\boldsymbol{w}}_{\text{new}}$:

$$\widehat{\boldsymbol{w}}_{\text{new}} = (\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}})^{-1}\boldsymbol{X}_{\text{new}}^\top \boldsymbol{y}_{\text{new}}.$$

Since $\boldsymbol{X}_{\text{new}}^\top \boldsymbol{y}_{\text{new}} = \boldsymbol{X}^\top \boldsymbol{y} + \tilde{\boldsymbol{x}}\tilde{y}$, we have:

$$\widehat{\boldsymbol{w}}_{\text{new}} = \left[(\boldsymbol{X}^\top \boldsymbol{X})^{-1} - \frac{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}}{1+\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}\right]\left(\boldsymbol{X}^\top \boldsymbol{y} + \tilde{\boldsymbol{x}}\tilde{y}\right).$$

Separate terms and simplify to express $\widehat{\boldsymbol{w}}_{\text{new}}$ in terms of $\widehat{\boldsymbol{w}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$:

$$\widehat{\boldsymbol{w}}_{\text{new}} = \widehat{\boldsymbol{w}} + \frac{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}{1+\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}\left(\tilde{y} - \tilde{\boldsymbol{x}}^\top \widehat{\boldsymbol{w}}\right).$$

Thus, the updated formula for $\widehat{\boldsymbol{w}}_{\text{new}}$ is:

$$\widehat{\boldsymbol{w}}_{\text{new}} = \widehat{\boldsymbol{w}} + \frac{\boldsymbol{K}\tilde{\boldsymbol{x}}}{1+\tilde{\boldsymbol{x}}^\top \boldsymbol{K}\tilde{\boldsymbol{x}}}\left(\tilde{y} - \tilde{\boldsymbol{x}}^\top \widehat{\boldsymbol{w}}\right),$$

where $\boldsymbol{K} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}$.

2. *[3 points]* We initialize $\boldsymbol{K} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ and $\boldsymbol{\alpha} = \widehat{\boldsymbol{w}}$. Use (3) and (4) to show that we can get $\boldsymbol{K} = (\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}})^{-1}$ and $\boldsymbol{\alpha} = \widehat{\boldsymbol{w}}_{\text{new}}$ after running the following algorithm.

Algorithm 1 Online update

```
1: Receive observation $(\tilde{\boldsymbol{x}}, \tilde{y})$.
2: Set $\boldsymbol{\xi} \leftarrow \boldsymbol{K}\tilde{\boldsymbol{x}}/(1+\tilde{\boldsymbol{x}}^\top \boldsymbol{K}\tilde{\boldsymbol{x}})$.
3: Set $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \boldsymbol{\xi}(\tilde{y} - \tilde{\boldsymbol{x}}^\top \boldsymbol{\alpha})$.
4: Set $\boldsymbol{K} \leftarrow \boldsymbol{K} - \boldsymbol{\xi}\tilde{\boldsymbol{x}}^\top \boldsymbol{K}$.
```

Solution: We have

$$\boldsymbol{\xi} = \frac{\boldsymbol{K}\tilde{\boldsymbol{x}}}{1+\tilde{\boldsymbol{x}}^\top \boldsymbol{K}\tilde{\boldsymbol{x}}} = \frac{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}{1+\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}.$$

After running the instructions, we have

$$\boldsymbol{\alpha} = \widehat{\boldsymbol{w}} + \frac{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}{1+\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}}(y - \tilde{\boldsymbol{x}}^\top \widehat{\boldsymbol{w}}) = \widehat{\boldsymbol{w}}_{\text{new}},$$

$$\boldsymbol{K} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} - \frac{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}}{1+\tilde{\boldsymbol{x}}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\tilde{\boldsymbol{x}}} = (\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}})^{-1}.$$

3. *[4 points]* Assume that n new data points arrive in a sequential order and we would like to update $\widehat{\boldsymbol{w}}$ every time a new data point arrives. Consider the following two methods:

   a. First, initialize $\widehat{\boldsymbol{w}}$ using the original N data points. Then run Algorithm 1 once for each new data point.

   b. Directly compute $\widehat{\boldsymbol{w}}_{\text{new}} = (\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}})^{-1} \boldsymbol{X}_{\text{new}}^\top \boldsymbol{y}_{\text{new}}$ every time a new data point arrives.

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

Solution:

a)

$$\begin{array}{rc} \text{Initialization}: & O(Nd^2) + O(d^3). \\ \text{Computing } \boldsymbol{\xi}: & O(d^2). \\ \text{Computing } \boldsymbol{\alpha}: & O(d). \\ \text{Computing } \boldsymbol{K}: & O(d^2). \end{array}$$

Be sure to avoid matrix multiplication when computing $\boldsymbol{K} \leftarrow \boldsymbol{K} - \boldsymbol{\xi}(\tilde{\boldsymbol{x}}^\top \boldsymbol{K})$. So the computational complexity is $O(Nd^2) + O(d^3) + O(nd^2)$,

b) The naive approach needs $O(Nd^2) + O(nd^3)$ (Note that we can get $\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}} = \boldsymbol{X}^\top \boldsymbol{X} + \tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^\top$, so we only need $O(d^2)$ to get $\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}}$ in every iteration).

The method in part 2 is more efficient.

Note: If we compute $\boldsymbol{X}_{\text{new}}^\top \boldsymbol{X}_{\text{new}}$ from scratch every time, the complexity of the naive approach becomes $O(nNd^2 + n^2d^2 + nd^3)$.

7 Python: Linear Regression with Regularization and Online Updates *[30 points]*

Follow the instructions below to solve a regression problem using ridge regression with varying regularization parameters (λ) and analyze the bias-variance trade-off. Use only the '**numpy**' library for computations and '**matplotlib**' for plotting. When uploading to Gradescope, you will need to produce a PDF version of your solutions and code. One way to do this is to use a notebook (`https://jupyter.org`).

7.1 Problem: Regularized Linear Regression

In many real-world applications, predictive models often encounter challenges such as multicollinearity among features, small sample sizes, or noisy data. These challenges can lead to overfitting, where the model captures noise instead of meaningful patterns. To address this, regularized linear regression, such as ridge regression, introduces a penalty term to the loss function, which helps constrain the model complexity and improve generalization.

Consider the following scenarios where regularized linear regression plays a critical role:

- **Financial Forecasting:** Predicting stock prices or company revenues often involves highly correlated features, such as economic indicators or market trends. Ridge regression can reduce the impact of multicollinearity, ensuring stable and interpretable predictions.

- **Healthcare Analytics:** In clinical studies with limited patient data, predictive models for treatment outcomes may suffer from high variance due to noise. Regularization helps to avoid overfitting by penalizing extreme weight values.
- **Marketing Campaign Analysis:** Estimating the impact of various advertisement strategies on sales can involve sparse and noisy data. Regularization ensures that the model remains robust despite inconsistencies in the data.

In this problem, you will explore ridge regression on synthetic data to understand its behavior and implications for real-world applications. Specifically, you will:

- **Analyze the behavior of learned coefficients:** Observe how the learned coefficients vary as the regularization parameter λ changes, helping to illustrate how ridge regression stabilizes the model.
- **Assess model performance:** Measure the model's performance using Root Mean Squared Error (RMSE) on validation data to identify the optimal regularization parameter λ^*.
- **Investigate the bias-variance trade-off:** Quantify and visualize how regularization influences bias and variance, providing insights into the trade-offs inherent in predictive modeling.

7.2 Synthetic Data Generation

Generate a synthetic dataset for training, validation, and testing. The dataset consists of $N = 500$ data points and $d = 12$ features. Each data point is generated as follows:

$$y = \boldsymbol{x}^\top \boldsymbol{w} + \varepsilon,$$

where $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, $\boldsymbol{w}$ is a vector of length d with linearly spaced values between 1 and 5, and $\varepsilon \sim \mathcal{N}(0, 0.5^2)$.

Split the data into:

- Training set (70%): For training the model.
- Validation set (15%): For selecting the optimal λ.
- Test set (15%): For final evaluation.

7.3 Tasks

a. **Plot Coefficients vs. λ:**

- Train ridge regression models for $\lambda \in \{a \cdot 10^b : a \in \{1, 2, \ldots, 9\}, b \in \{-5, -4, \ldots, 2\}\}$.
- Plot the learned coefficients (w_i) for all 12 features and the bias term against λ (use a log scale on the vertical axis).

b. **Validation RMSE vs. λ:**

- Calculate the RMSE (root mean squared error) on the validation set for each λ.
- Plot the validation RMSE against λ (use a logarithmic scale on the vertical axis) and identify λ^*, the value that minimizes the RMSE on the validation dataset.

c. **Predicted vs. True Values:**

- Use λ^* to train the model on the combined training and validation sets.
- Plot the predicted values against the true values for the test set. Your result should be a scatter plot with each point representing a (predicted value, true value) pair for one data point in the test set.

d. **Bias-Variance Trade-off:**

   - Generate $L = 20$ independent training datasets of size $N_{\text{sub}} = 50$ by sampling with replacement from the training data.
   - Train models for each dataset and calculate the bias and variance for each λ.
   - Plot the variance against λ (use a logarithmic scale for the vertical axis).

7.4 Hints

- Use gradient descent to solve ridge regression:

$$\boldsymbol{w}^* = \text{argmin}_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2.$$

- Initialize $\boldsymbol{w} = \mathbf{0}$, and use a learning rate of 0.01 with a stopping criterion of 10^{-6}.
- For sampling with replacement, use `numpy.random.choice`.

Here is the starter code:

```
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 pass

def ridge_gradient(w, X, y, lam):
    """Calculate the gradient of the ridge regression loss."""
    residuals = y - X @ w
    grad = pass
    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

# Empty sections for students to complete
def plot_coefficients_vs_lambda():
    pass

def plot_rmse_vs_lambda():
    pass

def plot_predicted_vs_true():
    pass

def plot_bias_variance_tradeoff():
    pass
```

7.5 Deliverables

- Python code for all tasks, including plots.
- Saved figures:
  - coefficients_vs_lambda.png
  - rmse_vs_lambda.png
  - predicted_vs_true.png
  - bias_variance_tradeoff.png
- Analysis discussing:
  - How coefficients behave as λ increases.
  - The trade-off between RMSE and λ.
  - Observations from the bias-variance trade-off plot.

7.6 Evaluation Criteria

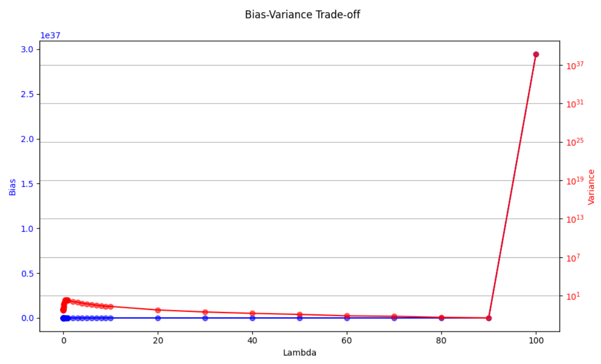
- *[10 points]* Correct implementation of ridge regression and bias-variance analysis.
- *[4 points]* Visualization of coefficients with respect to λ.
- *[6 points]* Validation RMSE plot and identification of λ^*.
- *[4 points]* Scatter plot for predictions versus true values.
- *[6 points]* Plot and meaningful analysis of bias-variance trade-off with respect to λ.
- *Make sure that all figures, plots, and diagrams referenced in your work are embedded directly in the PDF file you submit.

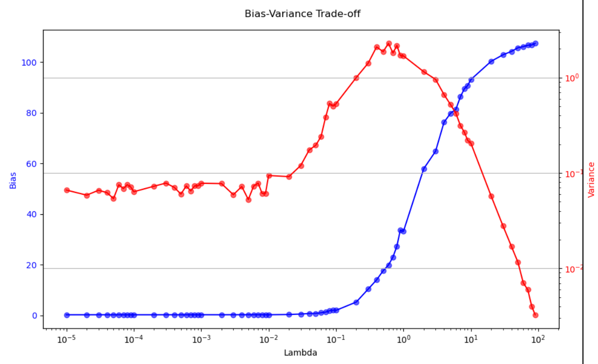
Solution:

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

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

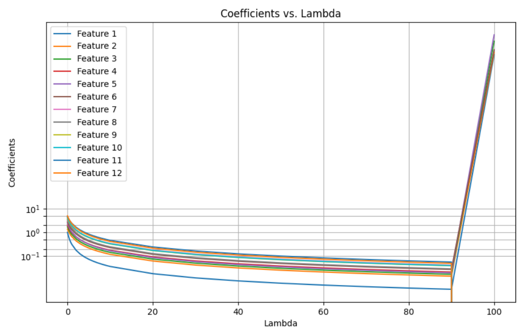
```
# Generate feature matrix and true weights
X = np.random.normal(0, 1, (N, d))
true_weights = np.linspace(1, 5, d)  # Linearly spaced true weights
epsilon = np.random.normal(0, 0.5, N)  # Noise
y = X @ true_weights + epsilon  # Generate target values

# 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:]

def ridge_loss(w, X, y, lam):
    residuals = y - X @ w
    return np.sum(residuals**2) / len(y) + lam * np.sum(w**2)

def ridge_gradient(w, X, y, lam):
    residuals = y - X @ w
    grad = -2 * X.T @ residuals / len(y) + 2 * lam * w
    return grad

def gradient_descent(loss_fn, grad_fn, w_init, X, y, lam, lr=0.01, tol=1e-6, max_iters=1000):
    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

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

# Empty sections for students to complete
def plot_coefficients_vs_lambda(lambdas, coefficients, d):
    plt.figure(figsize=(10, 6))
    for i in range(d):
        plt.plot(lambdas, coefficients[:, i], label=f"Feature {i+1}")
    plt.xscale("log")
    plt.yscale("log")
    plt.yticks([0.1, 0.2, 0.5, 1, 2, 5, 10])
    plt.xlabel("Lambda")
    plt.ylabel("Coefficients")
    plt.title("Coefficients vs. Lambda")
    plt.legend()
    plt.grid()
    plt.savefig("coefficients_vs_lambda.png")
    plt.show()

def plot_rmse_vs_lambda(lambdas, rmse_vals):
    plt.figure(figsize=(10, 6))
    plt.plot(lambdas, rmse_vals, marker="o")
    plt.xscale("log")
    plt.yscale("log")
    plt.xlabel("Lambda")
    plt.ylabel("Validation RMSE")
    plt.title("Validation RMSE vs. Lambda")
    plt.grid()
    plt.savefig("rmse_vs_lambda.png")
    plt.show()

def plot_predicted_vs_true(y_test, y_test_pred):

    plt.figure(figsize=(8, 8))
    plt.scatter(y_test, y_test_pred, alpha=0.6)
    plt.plot(y_test,y_test)
    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(lambdas, biases, variances):
    # Filter based on reasonable thresholds
    filtered_indices = [i for i in range(len(lambdas)) if biases[i] < 1e3 and variances[i] < 1e3]
    filtered_lambdas = [lambdas[i] for i in filtered_indices]
    filtered_biases = [biases[i] for i in filtered_indices]
    filtered_variances = [variances[i] for i in filtered_indices]

    fig, ax1 = plt.subplots(figsize=(10, 6))

    ax1.set_xscale("log")
    ax1.plot(filtered_lambdas, filtered_biases, label="Bias", color="blue")
    ax1.scatter(filtered_lambdas, filtered_biases, alpha=0.6, color="blue")
    ax1.set_xlabel("Lambda")
    ax1.set_ylabel("Bias", color="blue")
    ax1.tick_params(axis="y", labelcolor="blue")

    ax2 = ax1.twinx()
    ax2.plot(filtered_lambdas, filtered_variances, label="Variance", color="red")
    ax2.scatter(filtered_lambdas, filtered_variances, alpha=0.6, color="red")
    ax2.set_ylabel("Variance", color="red")
    ax2.tick_params(axis="y", labelcolor="red")
    ax2.set_yscale('log')

    fig.suptitle("Bias-Variance Trade-off")
    fig.tight_layout()
    plt.grid()
    plt.savefig("bias_variance_tradeoff.png")
    plt.show()

# Define lambdas for ridge regression
lambdas = [a * 10**b for b in range(-5, 3) for a in range(1, 10)]

coefficients = []
rmse_vals = []
for lam in lambdas:
    w_init = np.zeros(d)
    w = gradient_descent(ridge_loss, ridge_gradient, w_init, X_train, y_train, lam)
    y_pred = X_val @ w
    rmse = np.sqrt(np.mean((y_val - y_pred)**2))

    coefficients.append(w)
    rmse_vals.append(rmse)

coefficients = np.array(coefficients)
```

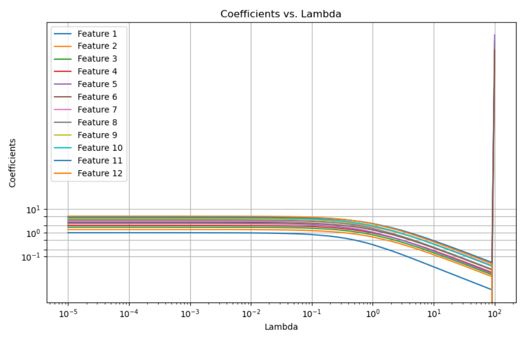
```
lambda_star = lambdas[np.argmin(rmse_vals[~np.isnan(rmse_vals).any()])]
print("Best lambda is:", lambda_star)

w_init = np.zeros(d)
w_star = gradient_descent(ridge_loss, ridge_gradient, w_init, np.vstack([X_train, X_val]),
                           np.hstack([y_train, y_val]), lambda_star)
y_test_pred = X_test @ w_star
biases, variances = calculate_bias_variance(X_train, y_train, X_val, y_val, lambdas)

plot_coefficients_vs_lambda(lambdas, coefficients, d)
plot_rmse_vs_lambda(lambdas, rmse_vals)
plot_predicted_vs_true(y_test, y_test_pred)
plot_bias_variance_tradeoff(lambdas, biases, variances)
```



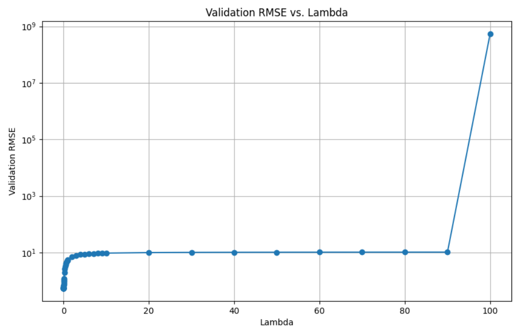


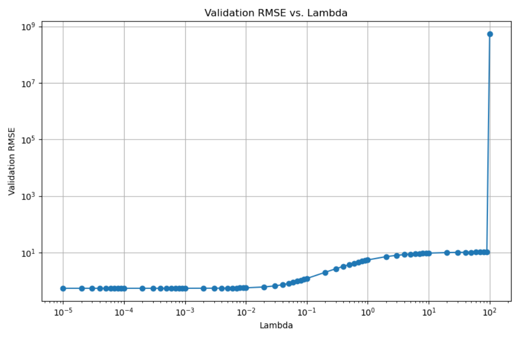
or



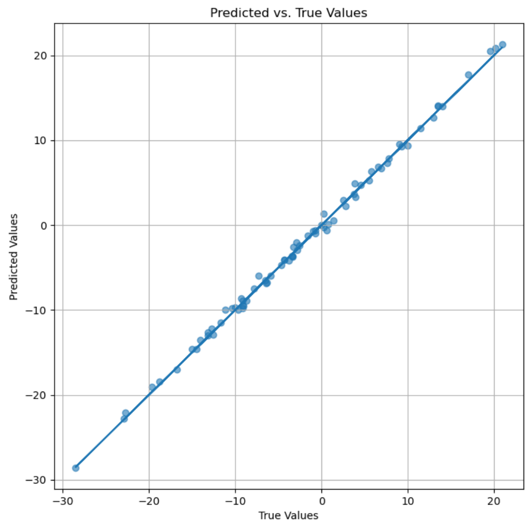


or





or



- **Coefficient Shrinkage**:
  - The plot of coefficients ($\boldsymbol{w}$) versus the regularization parameter (λ) shows that as λ increases, the coefficients shrink toward zero.
  - This behavior reflects the effect of the ridge regression penalty, which discourages large coefficient magnitudes to prevent overfitting.
  - However, with a large λ, gradient descent can lead to coefficient explosion.
- **Validation RMSE**:
  - The validation RMSE curve shows a clear minimum at an optimal regularization parameter (λ^*).
- **Bias-Variance Trade-off**:
  - For small λ values, the model is highly flexible, which makes it prone to overfitting. This results in high variance and low bias because the model closely fits the training data, including noise, which compromises its ability to generalize to unseen data.
  - In the moderate λ range, the trade-off between bias and variance becomes more apparent. As λ increases, variance decreases significantly, while bias grows slightly. This range is the most meaningful for analysis, as the model achieves a balance between underfitting and overfitting, ensuring better generalization to unseen data.
  - For large λ values, the model becomes overly constrained, leading to significant underfitting. Bias increases substantially as the model loses its ability to capture meaningful patterns in the data.
  - Unexpectedly, variance may also increase due to numerical instability or the extreme effects of over-regularization. In this range, the model's predictions often collapse into overly simplistic outputs, resulting in poor performance. This highlights the importance of selecting an optimal λ to maintain an effective balance between bias and variance.

- **Predicted vs True Values**:
  - For the optimal λ^*, the scatter plot of predicted values ($\hat{y}$) versus true values (y) shows that the predictions align closely with the true labels.
  - This alignment indicates that the model effectively balances bias and variance at λ^*, achieving good predictive performance.