Problem 1 : Coding

The following questions should be completed before you start the programming component of this assignment. Assume the `dtypes` of all `ndarrays` are `np.float64`. Vectors are 1D `ndarrays`.

Note: These questions require understanding of NumPy broadcasting. Please refer to the NumPy documentation for broadcasting concepts and rules: `https://numpy.org/doc/stable/user/basics.broadcasting.html`

(1) **Select all that apply:** Given a data matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ and a weight vector $\mathbf{v} \in \mathbb{R}^{M}$. The following code snippet computes a result vector $\mathbf{u} \in \mathbb{R}^{N}$ where each component u_i represents the weighted sum of the i-th row of $\mathbf{X}$ with weights given by $\mathbf{v}$:

```
# X and v are numpy ndarrays
# X has shape (N, M), v has shape (M,)
u = np.zeros(N)
for row_idx in range(N):
    for col_idx in range(M):
        u[row_idx] += X[row_idx, col_idx] * v[col_idx]
```

Select all equivalent NumPy operations that compute the same output:

- ■ `u = X @ v`
- □ `u = v @ X`
- ■ `u = np.matmul(X, v)`
- □ `u = np.matmul(v, X)`
- □ `u = X * v`
- □ `u = v * X`
- ■ `u = np.dot(X, v)`
- □ `u = np.dot(v, X)`
- □ None of the above.

(2) Suppose we have a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ and a coefficient vector $\mathbf{w} \in \mathbb{R}^{N}$. Define a matrix $\mathbf{\Omega}$ computed as follows: $\mathbf{\Omega} = \sum_{i=0}^{N-1} w_i \left(\mathbf{x}_i - \overline{\mathbf{x}_i}\right) \left(\mathbf{x}_i - \overline{\mathbf{x}_i}\right)^T$, where $\mathbf{x}_i \in \mathbb{R}^{M}$ represents the i-th *row* of $\mathbf{X}$ (expressed as a *column* vector), $\overline{\mathbf{x}_i} \in \mathbb{R}$ denotes the average value of all elements in $\mathbf{x}_i$, and $w_i \in \mathbb{R}$ is the i-th component of $\mathbf{w}$ (for $i \in \{0, 1, \cdots, N-1\}$). In the following questions, let `X` and `w` represent $\mathbf{X}$ and $\mathbf{w}$ respectively, where `X.shape == (N, M)` and `w.shape == (N,)`.

(i) **Select one:** Identify the correct line(s) of Python code that generates a matrix where each row i equals $\left(\mathbf{x}_i - \overline{\mathbf{x}_i}\right)^T$ (i.e., the i-th row of $\mathbf{X}$ after subtracting its row-wise mean).

- □ `(X - np.mean(X, axis=0)).T`
- ■ `X - np.mean(X, axis=1, keepdims=True)`
- □ `X - np.mean(X, axis=0, keepdims=True)`
- □ `X - np.expand_dims(np.mean(X, axis=1), 1)`
- □ None of the above.

(ii) **Select one:** Let `M` store the output from part (i). Which line(s) of code correctly evaluate $\boldsymbol{\Omega}$ using `M`?

- ■ `np.matmul(w * M.T, M)`
- ☐ `np.matmul(w * M, M.T)`
- ☐ `np.dot(w * M, M.T)`
- ☐ `w * np.dot(M.T, M)`
- ☐ None of the above.

Problem 2 : Linear Regression

1. (1) You are given the following training data for a regression task:

x	7.0	3.0	5.0	2.0	6.0
y	2.0	1.0	2.0	0.0	3.0

Table 1: Regression Dataset

Denote $\boldsymbol{x}$ as the feature vector and $\boldsymbol{y}$ as the target vector. We apply gradient descent optimization to fit a linear model with cost function $J(w, b) = \frac{1}{N}\sum_{i=1}^{N}(wx_i + b - y_i)^2$, where N represents the sample count, w denotes the slope parameter, and b denotes the bias term.

Note: While demonstrating your calculation steps is optional, we encourage you to show your work for potential partial credit if your final answer contains errors.

(a) Suppose we set the initial slope parameter to 3.0 and the initial bias term to 0.0. Compute the derivative of the cost function with respect to the slope w using the complete dataset during the initial gradient descent iteration.
Round to 4 decimal places after the decimal point.

Gradient:

129.6000

Work

$\nabla_w J = \frac{2}{N}\sum_{i=1}^{N}(wx_i + b - y_i)x_i$
Let $N = 5$, $w = 3.0$, $b = 0.0$, and input the data points:
$\nabla_w J = \frac{2}{5}((3 * 7.0 + 0.0 - 2.0) * 7.0 + (3 * 3.0 + 0.0 - 1.0) * 3.0 +$
$(3 * 5.0 + 0.0 - 2.0) * 5.0 + (3 * 2.0 + 0.0 - 0.0) * 2.0 + (3 * 6.0 + 0.0 - 3.0) * 6.0)$
$= 129.6000.$

(b) Calculate the derivative of the cost function with respect to the bias parameter b, evaluated across the entire training set for the first iteration.

Gradient:

24.4000

Work

$\nabla_b J = \frac{2}{N} \sum_{i=1}^{N} (wx_i + b - y_i)$
Let $N = 5$, $w = 3.0$, $b = 0.0$, and input the data points:
$\nabla_b J = \frac{2}{5} ((3 * 7.0 + 0.0 - 2.0) + (3 * 3.0 + 0.0 - 1.0) +$
$(3 * 5.0 + 0.0 - 2.0) + (3 * 2.0 + 0.0 - 0.0) + (3 * 6.0 + 0.0 - 3.0))$
$= 24.4000.$

(c) Using a step size of 0.01, execute one iteration of the gradient descent algorithm. Report the updated values of both the slope and bias parameters following this update step.
Round to 4 decimal places after the decimal point.

Weight:

3.0 - 0.01 * 129.6000 = 1.7040

Intercept:

0.0 - 0.01 * 24.4000 = -0.2440

(2) Let $\mathcal{D}_1 = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ be a training dataset. The optimal linear model that minimizes squared error loss on $\mathcal{D}_1$ has the form $y = w_1 x + b_1$.

(a) **Select one:** Consider a transformed dataset $\mathcal{D}_2 = \{(x^{(1)} + \alpha,\, y^{(1)} + \beta), \ldots, (x^{(N)} + \alpha,\, y^{(N)} + \beta)\}$, where $\alpha > 0, \beta > 0$ are constant shifts and $w_1\alpha \neq \beta$. Let $y = w_2 x + b_2$ denote the optimal linear model minimizing squared error on $\mathcal{D}_2$. Identify which relationship between w_1, w_2, b_1, b_2 must hold universally for all valid choices of α, β satisfying the given constraints.

- ☐ $w_1 = w_2, b_1 = b_2$
- ☐ $w_1 \neq w_2, b_1 = b_2$
- ■ $w_1 = w_2, b_1 \neq b_2$
- ☐ $w_1 \neq w_2, b_1 \neq b_2$

(b) Suppose a colleague attempts to process $\mathcal{D}_1$ but inadvertently creates multiple copies of certain data points. Explain why the optimal regression coefficients obtained from this corrupted dataset *may* deviate from the original parameters w_1 and b_1.

Your answer:

In standard linear regression, each data point contributes equally to the total loss. When a point is duplicated multiple times, its contribution to the total loss becomes amplified. During gradient descent, the optimization process overemphasizes these duplicated points, prioritizing their fit, which can cause the optimal regression coefficients to deviate from the original parameters.

(3) Consider training a linear model on dataset $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$ with $\boldsymbol{x} \in \mathbb{R}^k$. We employ a robust loss function known as Cauchy loss, defined as:

$$\ell(\hat{y}, y) = \frac{c^2}{2} \log\left(1 + \left(\frac{y - \hat{y}}{c}\right)^2\right)$$

where c is a scaling parameter. For this problem, set $c = 1$, use natural logarithm (base **e**), and assume zero bias (no intercept). Under these settings, the Cauchy loss for a single observation $\boldsymbol{x}^{(i)}$ with model parameters $\boldsymbol{\theta}$ becomes:

$$J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2} \log\left(1 + \left(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)}\right)^2\right)$$

Our optimization objective is to minimize the mean Cauchy loss computed over all samples in $\mathcal{D}$. Remember that bias equals 0 throughout.

(a) Derive the partial derivative of $J^{(i)}(\boldsymbol{\theta})$ with respect to an arbitrary parameter θ_j (the j^{th} component). Recall that no intercept is included.

Your answer:

$\frac{\partial J^{(i)}}{\partial \theta_j} = -\frac{(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)}) x_j^{(i)}}{1 + (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2}$

(b) Express the full gradient vector of $J^{(i)}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. (Hint: Leverage your answer from part (a))

Your answer:

$\nabla_{\boldsymbol{\theta}} J^{(i)} = -\frac{(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)}) \boldsymbol{x}^{(i)}}{1 + (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2}$

(c) Based on your result from (b), analyze the behavior of $\nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$ when the prediction error for sample $(\boldsymbol{x}^{(i)}, y^{(i)})$ becomes very large. Provide an explanation for this phenomenon.

Your Answer

When the prediction error $(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})$ becomes very large, the term $(y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2$ dominates the denominator $1 + (y^{(i)} - \boldsymbol{\theta}^T \boldsymbol{x}^{(i)})^2$. As a result, the gradient $\nabla_{\boldsymbol{\theta}} J^{(i)}$ approaches zero.

This behavior indicates that the Cauchy loss function is robust to outliers, as it reduces the influence of samples with large prediction errors on the parameter updates during optimization.

2. (1) Consider fitting a linear predictor to the training set

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)} \right), \left(\mathbf{x}^{(2)}, y^{(2)} \right), \cdots, \left(\mathbf{x}^{(N)}, y^{(N)} \right) \right\}$$

where $\mathbf{x}^{(i)} \in \mathbb{R}^M$, using the ordinary least squares (OLS) criterion:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - \sum_{j=1}^{M} w_j x_j^{(i)} \right)^2 .$$

(i) **Select one:** To determine each weight w_k (where $1 \leq k \leq M$), we find the stationary point by setting $\frac{\partial J(\mathbf{w})}{\partial w_k} = 0$. Identify the correct closed-form expression for w_k as a function of the training samples $(\mathbf{x}^{(1)}, y^{(1)})$, $\cdots$, $(\mathbf{x}^{(N)}, y^{(N)})$ and the remaining weights $w_1, \cdots, w_{k-1}, w_{k+1}, \cdots, w_M$:

- ■ $w_k = \frac{\sum_{i=1}^{N} x_k^{(i)} (y^{(i)} - \sum_{j=1, j\neq k}^{M} w_j x_j^{(i)})}{\sum_{i=1}^{N} (x_k^{(i)})^2}$
- □ $w_k = \frac{\sum_{i=1}^{N} x_k^{(i)} (y^{(i)} - \sum_{j=1, j\neq k}^{M} w_j x_j^{(i)})}{\sum_{i=1}^{N} (y^{(i)})^2}$
- □ $w_k = \frac{\sum_{i=1}^{N} x_k^{(i)} (y^{(i)} - \sum_{j=1, j\neq k}^{M} w_j x_j^{(i)})}{\sum_{i=1}^{N} (x_k^{(i)} y^{(i)})^2}$
- □ $w_k = \sum_{i=1}^{N} x_k^{(i)} (y^{(i)} - \sum_{j=1}^{M} w_j x_j^{(i)})$

(ii) **Select one:** Determine the dimensionality of the parameter space (number of weights w_k to estimate) and the number of constraint equations available from the optimality conditions:

- ■ M coefficients, M equations
- □ M coefficients, N equations
- □ N coefficients, M equations
- □ N coefficients, N equations

(2) **Select all that apply:** Function convexity is crucial for understanding optimization behavior in machine learning. Evaluate the following statements regarding convexity and identify all correct assertions:

- □ Strictly convex functions may possess infinitely many global minimizers, whereas convex functions have at most one global minimizer.
- □ The ℓ_1 loss (Mean Absolute Error), expressed as $L(y, \hat{y}) = \sum_{i=1}^{N} |y_i - \hat{y}_i|$, exhibits non-convex geometry.
- ■ When the objective function is convex, optimization algorithms need not worry about being trapped in suboptimal local minima.
- □ The squared ℓ_2 penalty term $\lambda ||\theta||_2^2$ is convex but lacks strict convexity.

Problem 3 : k-Nearest Neighbors

(1) You are implementing a binary k-NN classification model that predicts the label of a query sample based on plurality voting among its k closest neighbors from the labeled training data, where proximity is measured using Euclidean distance. When the vote yields equal counts for both classes, resolve the tie by randomly selecting one label with equal probability.
NOTE: For instance, if the 6 closest neighbors have labels {+, +, +, -, -, -}, meaning each class appears exactly 3 times, then randomly assign either + or - to the query sample.

(a) Train your model using the dataset depicted in Figure 1 with $k = 6$. Calculate the misclassification rate when applying this model to its own training data.
Express your result as either a fraction or a decimal rounded to 4 decimal places.

Your answer:

$\frac{2}{7} \approx 0.2857$

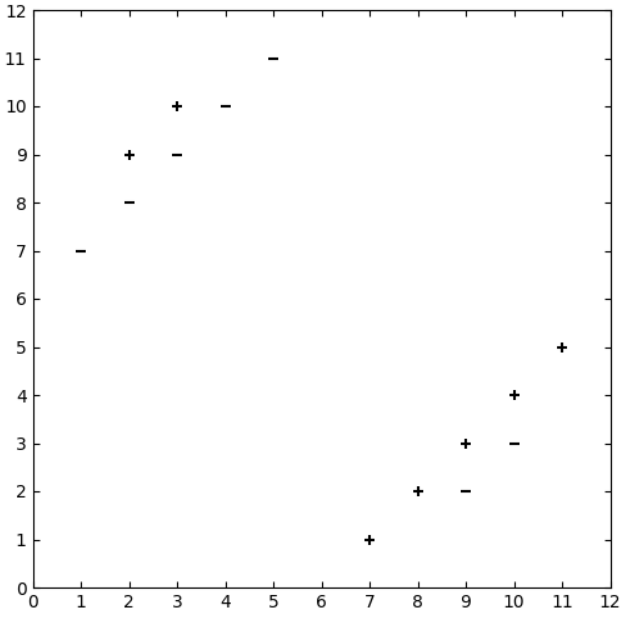


Figure 1: k-NN Dataset with Test Point

(b) **Select all that apply:** Consider an unseen query sample $\mathbf{x}^{\text{new}} = [3, 11]^T$ (with horizontal coordinate 3 and vertical coordinate 11) that needs prediction, as illustrated in Figure 2.

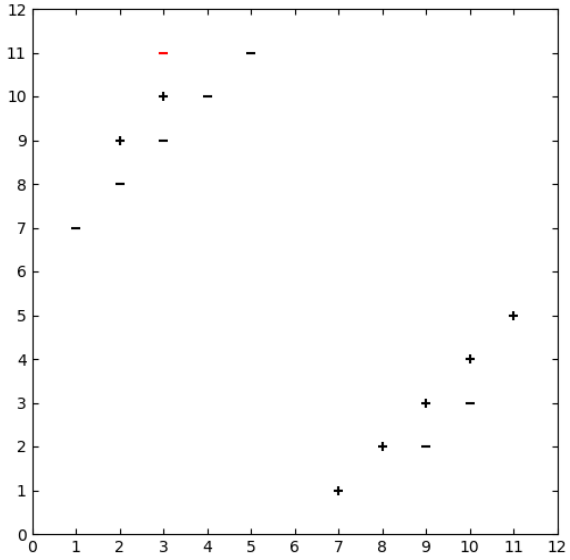


Figure 2: k-NN Dataset with Test Point

Identify which neighborhood size(s) k guarantee correct prediction for this query sample under the k-NN method:

- ☐ $k = 1$
- ■ $k = 5$
- ■ $k = 9$
- ■ $k = 12$
- ☐ None of the above

(2) **Select one:** Suppose you partition a large labeled dataset randomly into training and test subsets, with the goal of predicting labels for the test subset using k-NN.

(a) To achieve optimal test performance, the best strategy is to select k by minimizing the error rate observed on the training data.

- ☐ True
- ■ False

(b) **Select one:** An alternative approach divides your available training data into two parts: one for model fitting and another for validation. You then select hyperparameters that minimize validation error rather than training error. Does this validation-based selection strategy offer advantages over training-error-based selection?

- ■ Yes, optimizing for validation performance is superior because minimizing training error can result in overfitting and poor generalization.
- ☐ Yes, validation-based optimization ensures better test performance due to the guarantees provided by cross-validation.
- ☐ No, training error minimization is preferable because validation error reduction does not improve generalization and may cause overfitting.
- ☐ No, achieving the lowest possible training error is essential to guarantee optimal test set performance.

(c) **Select one:** A colleague proposes using the test data (rather than creating a separate validation partition from the training data) for hyperparameter tuning. Would you support this proposal? Provide justification in at most 3 sentences.

- ☐ Yes
- ■ No

Your answer:

No, because using the test set for hyperparameter tuning leaks information about the test distribution into the model selection process, leading to an overestimation of final performance and undermining the fairness of the test set as truly "unseen data".

(3) **Select all that apply:** You have a binary k-NN model with $k = 4$ distinguishing between two classes: "triangle" and "square". For query point $\mathbf{x} = (1, 1)$, exactly two of its four closest neighbors carry label "triangle" while the other two are labeled "square", as displayed below.

When multiple training points have identical distances to the query, prioritize those with smaller horizontal coordinates, then by smaller vertical coordinates.

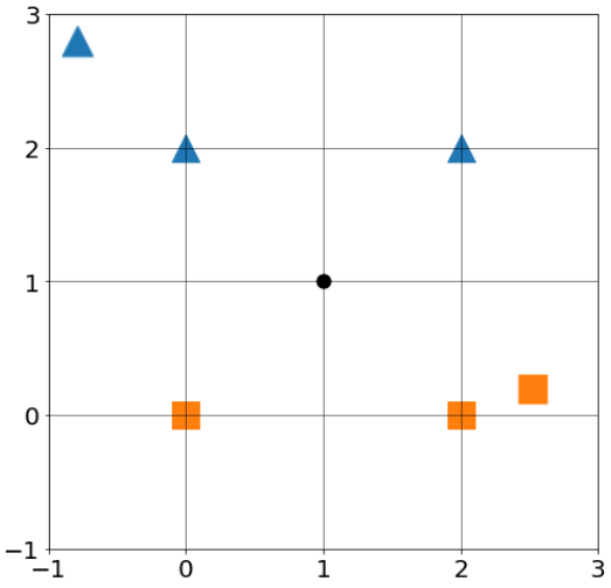


Figure 3: k-NN Tie-Breaking

Which modification(s) ensure that the voting procedure produces a unique class prediction for x without ties?

- ☐ Use $k = 2$ instead
- ■ Flip a coin to randomly assign a label to $\mathbf{x}$ (from the labels of its 4 closest points)
- ■ Use $k = 3$ instead
- ■ Use $k = 5$ instead
- ☐ None of the above.

(4) **Select all that apply:** Evaluate the following claims regarding k-NN methodology and select all accurate statements:

- ■ Increasing k typically results in more regular and less complex decision boundaries.
- ■ Raising the neighborhood size k helps mitigate the influence of noisy observations and anomalous data points.
- ☐ Setting k excessively high can lead to overfitting on the training data.
- ■ Cross-validation provides a principled approach for determining an appropriate value of k.
- ☐ The k value that achieves minimum validation error should always be avoided.
- ☐ None of the above.

(5) The table below shows observations linking academic achievement to post-graduation income. Two predictor variables (secondary school GPA and college GPA) and one response variable (annual salary in thousands of dollars) are recorded.

Student ID	High School GPA	University GPA	Salary
1	2.6	3.4	50
2	3.4	3.4	85
3	3.8	4.0	130
4	3.2	2.5	150
5	3.4	3.1	2500
6	3.2	3.1	70
7	3.8	3.7	unknown

Table 2: Student GPA-Salary Data

(a) Identify which student from Students 1-6 has the smallest Euclidean distance to Student 7. Report only the Student ID.

Your answer:

3

(b) Your objective is to estimate Student 7's post-graduation income using k-NN regression, where the predicted salary equals the mean salary of the k closest training samples. With $k = 3$, compute your salary prediction for Student 7 in the same units as the table (thousands of dollars annually).
Round your answer to the nearest integer.

Your answer:

$\frac{130+85+2500}{3} = 905$

(c) **Select all that apply:** Assume the 6 students above represent merely a sample from your complete training corpus of 10,000 students. You fit a k-NN regression model with Euclidean distance metric, evaluating performance via mean squared error (MSE) of salary predictions on the training data. Analyze which of the following feature transformations **could** alter the training MSE:

- ■ Normalizing only "High School GPA" as a percentage of 4.0
- ■ Normalizing only "University GPA" as a percentage of 4.0
- ☐ Applying identical normalization (same scaling factor) to both High School GPA and University GPA
- ☐ None of the above.

(6) An archaeologist unearths a 242KB 8-inch floppy disk buried near Wean Hall containing several hundred monochrome 3×3 pixel images. Your task is to categorize each image as either a photograph ($y = +$) or artistic rendering ($y = -$) for historical analysis purposes.

You construct a k-NN classification system using web-sourced training images (downsampled to matching 3×3 monochrome format). Each image encodes as a 3×3 binary matrix $\mathbf{x}$, with dissimilarity quantified via Hamming distance:

$$d(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{3} \sum_{j=1}^{3} \mathbf{1}(\mathbf{u}_{i,j} \neq \mathbf{v}_{i,j}) = \text{count of differing pixels between } \mathbf{u} \text{ and } \mathbf{v}$$

When selecting the k closest neighbors, if multiple candidates share the same distance, expand the neighborhood to include all tied candidates in the voting. Should the final vote produce a tie, output $\hat{y} =?$. Test your implementation using the samples provided below.

sample	y	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$
1	+	0	0	1	0	1	1	0	0	0
2	+	1	0	1	0	1	0	1	0	0
3	+	0	1	0	1	1	1	1	0	1
4	−	0	0	1	0	0	0	0	0	0
5	−	1	0	1	1	1	0	0	1	1

Table 3: Training Data (Problem 6)

sample	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$
6	1	0	1	0	0	0	1	0	1
7	0	0	1	0	0	1	1	1	0

Table 4: Test Data (Problem 6)

(a) Compute the Hamming distance between samples $\mathbf{x}^{(2)}$ and $\mathbf{x}^{(6)}$.

Your answer:

2

(b) **Select one:** Using $k = 3$ neighbors, what class label would your k-NN model assign to query sample $\mathbf{x}^{(7)}$?

- ■ $\hat{y} = +$
- □ $\hat{y} = -$
- □ $\hat{y} =?$

(c) **Select one:** With neighborhood size $k = 5$, determine the predicted label for query sample $\mathbf{x}^{(7)}$:

- ■ $\hat{y} = +$

☐ $\hat{y} = -$

☐ $\hat{y} =?$

(d) **Short answer:** A colleague suggests replacing Hamming distance with Euclidean distance for potentially improved accuracy. Would this substitution likely reduce test error? Explain your reasoning.

Your answer:

No, because the pixel values are binary (0 or 1), the Hamming distance and Euclidean distance produce identical neighbor rankings, so the final k-NN prediction is unaffected. Moreover, Euclidean distance requires additional square root computations, making it slightly less efficient than Hamming distance.

(7) **Numeric answer** You possess a substantial labeled dataset for training a k-NN classification model. To optimize hyperparameters, you plan to conduct an exhaustive grid search over two dimensions: neighborhood size k and distance function. Performance evaluation for each configuration will use cross-validation.

Your grid spans 5 candidate values for k (specifically: 3, 5, 7, 9, and 11) and 2 distance metrics (Euclidean and Hamming). You elect to use 10-fold cross-validation. Calculate the total number of distinct models that must be trained throughout this hyperparameter optimization procedure.

Your answer:

$5 \times 2 \times 10 = 100$

Problem 4 : Regression Tree

(1) You are working with the dataset shown below, where x_1 serves as an input variable for predicting the target y. Throughout this problem, each terminal node (leaf) outputs the mean y value of all training samples assigned to that node, and tree construction employs the squared error (SSE) partitioning criterion.

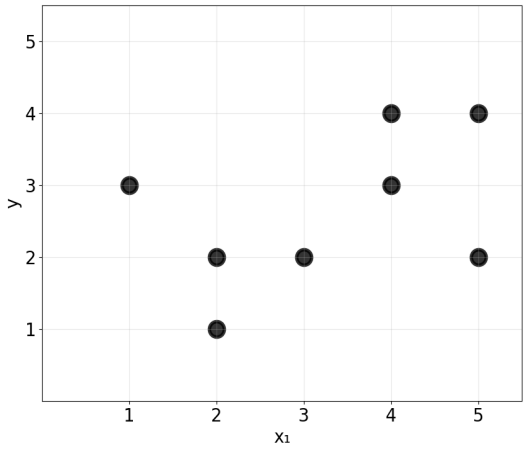


Figure 4: Regression Tree Data

(a) Suppose you construct a regression tree with a single partition point at $x_1 = 4.5$ as illustrated. Determine the model's prediction for a new observation having $x_1 = 2$. Report your result with one decimal place.

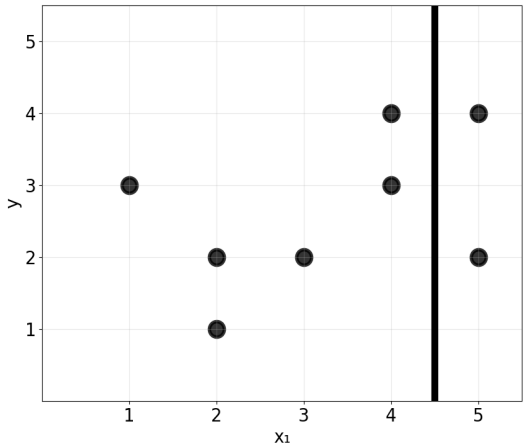


Figure 5: Sample Regression Tree

Your Answer

$y = \frac{3+3+2+2+1}{5} = 2.2$

(b) Identify which diagram below corresponds to the optimal single-split regression tree achievable for this dataset.

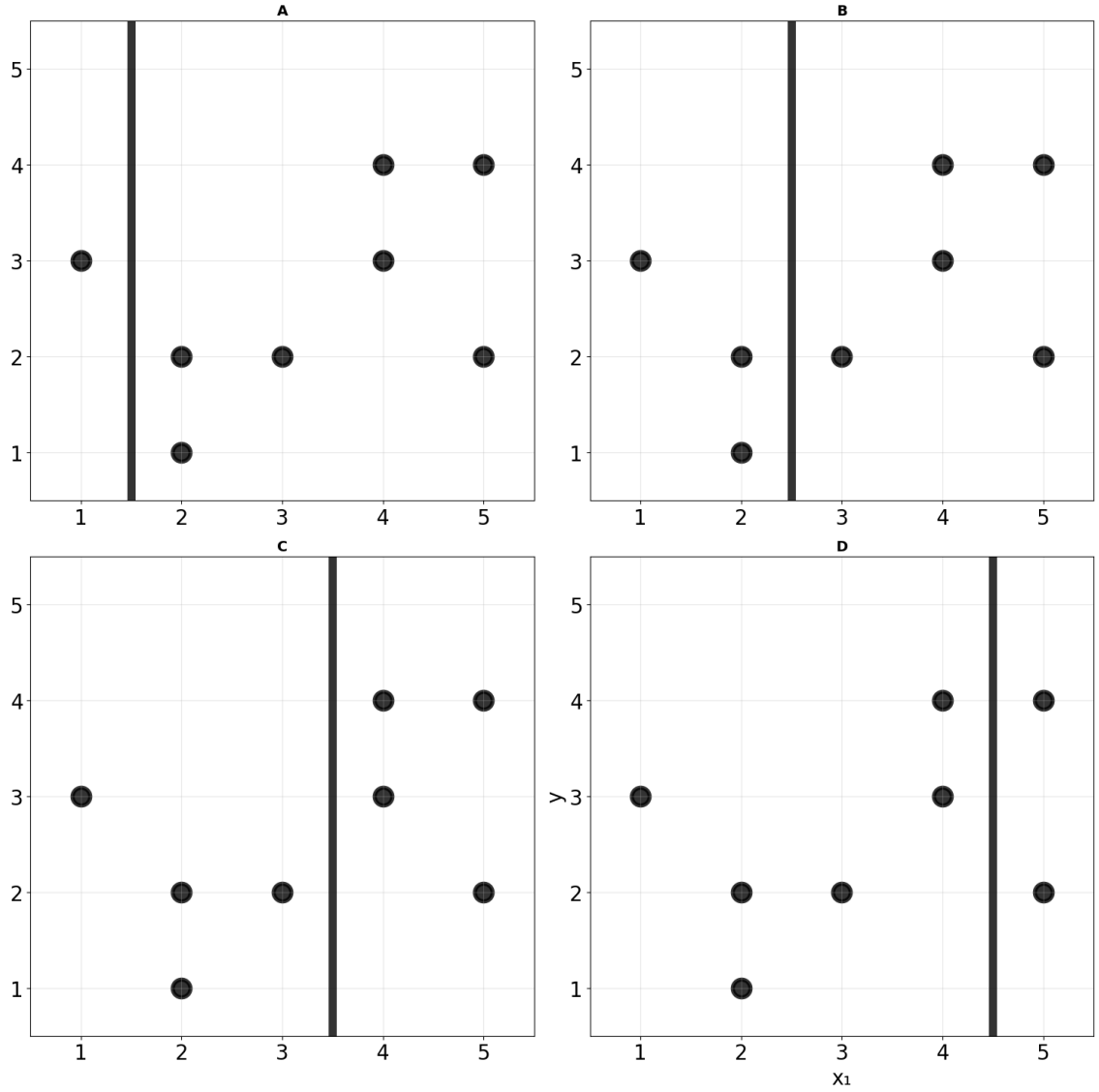


Figure 6: Answer Choices

☐ A
☐ B
■ C
☐ D

(c) Consider extending the tree from the previous question by adding one additional **level** using the same partitioning criterion. Select the resulting tree structure:

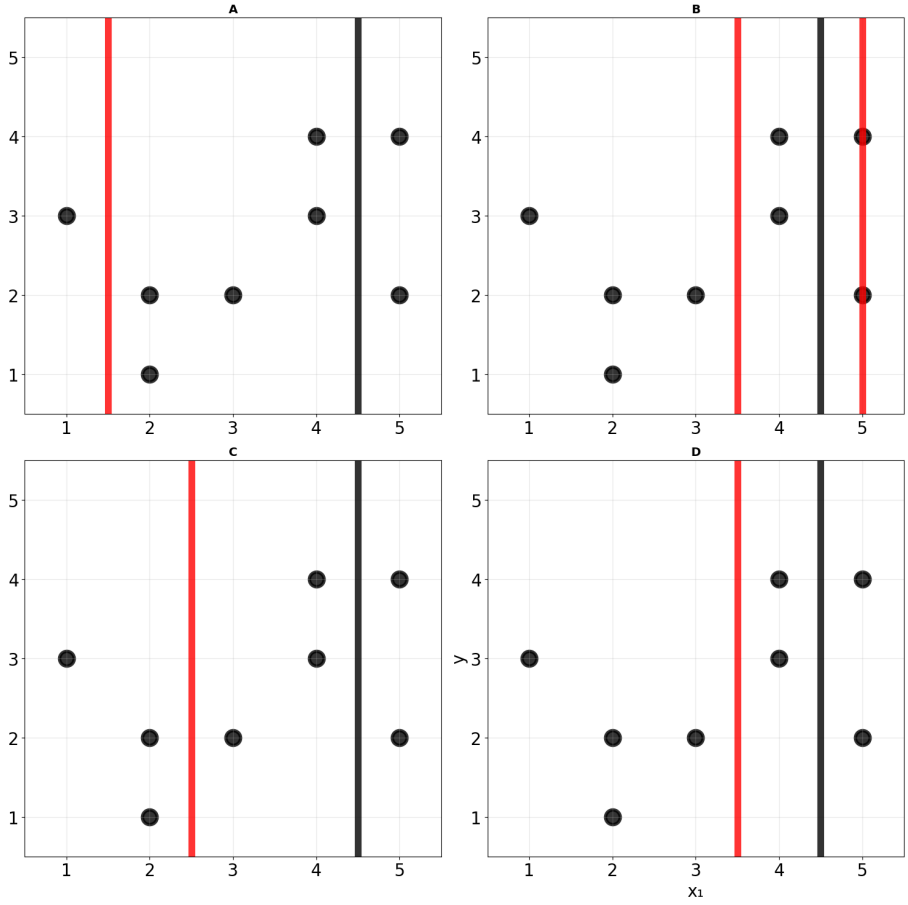


Figure 7: Answer Choices

- ☐ A
- ☐ B
- ☐ C
- ■ D

(2) Given an arbitrary dataset, suppose you construct a depth-2 regression tree by selecting optimal partitions at each decision node sequentially. Does this greedy construction procedure necessarily yield the depth-2 tree with minimum achievable training error? (**Hint:** Reflect on the algorithmic nature of regression tree learning)

Your answer:

No, because the greedy algorithm makes locally optimal choices at each node without considering the global structure of the tree. This can lead to suboptimal splits that do not minimize overall training error for the entire depth-2 tree.

Problem 5: Decision Tree

We examine decision tree construction principles using the training data below. Dataset D comprises 8 observations, each characterized by three features (A, B, C) and a target variable Y.

A	B	C	Y
1	2	0	1
0	1	0	0
0	0	1	0
0	2	0	1
1	1	0	1
1	0	1	0
1	2	1	0
1	1	0	1

Apply the dataset above to respond to the subsequent questions.
Important specifications:

- *Perform all intermediate computations without rounding!* Report your final rounded answers in the designated answer boxes.
- Unless stated otherwise, provide numerical results with 4-digit precision (e.g. 0.1234).
- Throughout this problem, we adopt the convention that terminal nodes (leaves) are excluded from node counts, and therefore do not contribute to depth or split tallies. (For instance, a tree with a single decision node has depth 1 and contains 1 split.)
- The dataset includes repeated observations; treat each row as a distinct sample without removing duplicates.

Note: Demonstrating your computational steps is optional but encouraged to facilitate partial credit. Only the numerical value in the left box receives grading.

(1) Compute the Shannon entropy $H(Y)$ for the target variable Y, expressed in bits. For this and all following questions, reporting values in *bits* requires using log base 2 in your computations.[1] (Report a single value rounded to four decimal places, e.g. 0.1234)

[1] Alternative logarithm bases yield different units: base e produces *nats*, while base 10 yields *bats*.

$H(Y)$	Work
1 bit	$P(Y=0) = 4/8 = 0.5$; $P(Y=1) = 4/8 = 0.5$; $H(Y) = -\sum P(Y)\log_2 P(Y) = -[0.5\log_2 0.5 + 0.5\log_2 0.5] = 1$ bit.

(2) Calculate the mutual information $I(Y;A)$ between target Y and feature A, measured in bits. (Provide a single number rounded to four decimal places, e.g. 0.1234)

$I(Y;A)$	Work
0.0488 bits	$P(Y=0\|A=1) = \frac{2}{5}$; $P(Y=1\|A=1) = \frac{3}{5} = 0.6$; $H(Y\|A=1) = -\sum P(Y\|A)\log_2 P(Y\|A) = -[\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5}]$; $P(Y=0\|A=0) = \frac{2}{3}$; $P(Y=1\|A=0) = \frac{1}{3}$; $H(Y\|A=0) = -\sum P(Y\|A)\log_2 P(Y\|A) = -[\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}]$; $P(A=1) = \frac{5}{8}$; $P(A=0) = \frac{3}{8}$; $H(Y\|A) = P(A=1)H(Y\|A=1) + P(A=0)H(Y\|A=0) = \frac{5}{8}H(Y\|A=1) + \frac{3}{8}H(Y\|A=0)$; $I(Y;A) = H(Y) - H(Y\|A) \approx 0.0488$ bits.

(3) Determine the mutual information $I(Y;B)$ between target Y and feature B, in bits. (Provide a single number rounded to four decimal places, e.g. 0.1234)

$I(Y;B)$	Work
0.3113 bits	$P(Y=0\|B=0) = 1$; $H(Y\|B=0) = -\sum P(Y\|B)\log_2 P(Y\|B) = -[1\log_2 1] = 0$; $P(Y=0\|B=1) = \frac{1}{3}$; $P(Y=1\|B=1) = \frac{2}{3}$; $H(Y\|B=1) = -\sum P(Y\|B)\log_2 P(Y\|B) = -[\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}]$; $P(Y=0\|B=2) = \frac{1}{3}$; $P(Y=1\|B=2) = \frac{2}{3}$; $H(Y\|B=2) = -\sum P(Y\|B)\log_2 P(Y\|B) = -[\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}]$; $P(B=0) = \frac{2}{8} = \frac{1}{4}$; $P(B=1) = \frac{3}{8}$; $P(B=2) = \frac{3}{8}$; $H(Y\|B) = P(B=0)H(Y\|B=0) + P(B=1)H(Y\|B=1) + P(B=2)H(Y\|B=2) = \frac{1}{4}\cdot 0 + \frac{3}{8}H(Y\|B=1) + \frac{3}{8}H(Y\|B=2)$; $I(Y;B) = H(Y) - H(Y\|B) \approx 0.3113$ bits.

(4) Evaluate the mutual information $I(Y;C)$ between target Y and feature C, measured in bits. (Report a single number rounded to four decimal places, e.g. 0.1234)

$I(Y;C)$	Work
0.5488 bits	$P(Y=0\|C=0)=\frac{1}{5}$; $P(Y=1\|C=0)=\frac{4}{5}$; $H(Y\|C=0)=-\sum P(Y\|C)\log_2 P(Y\|C)=-[\frac{1}{5}\log_2\frac{1}{5}+\frac{4}{5}\log_2\frac{4}{5}]$; $P(Y=0\|C=1)=1$; $H(Y\|C=1)=-\sum P(Y\|C)\log_2 P(Y\|C)=-[1\log_2 1]=0$; $P(C=0)=\frac{5}{8}$; $P(C=1)=\frac{3}{8}$; $H(Y\|C)=P(C=0)H(Y\|C=0)+P(C=1)H(Y\|C=1)=\frac{5}{8}H(Y\|C=0)+\frac{3}{8}\cdot 0$; $I(Y;C)=H(Y)-H(Y\|C)\approx 0.5488$ bits.

(5) Select one: Using the dataset provided, identify which feature (A, B, or C) would be selected for the root split by a decision tree learner employing mutual information as its selection criterion.

- ☐ A
- ☐ B
- ■ C

(6) Select one: Following the initial partition, which feature would the tree learning algorithm choose for the subsequent split, assuming mutual information guides feature selection? (*Hint:* Observe that this question presumes *exactly one* feature will be selected second.)

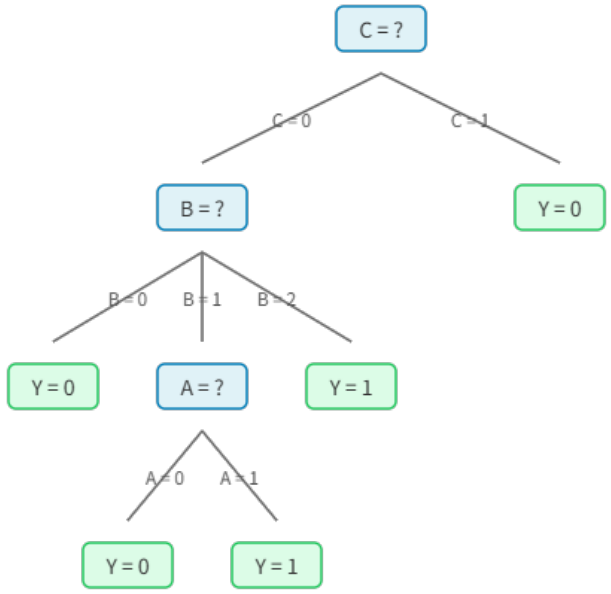
- ☐ A
- ■ B
- ☐ C

(7) Suppose the learning procedure continues recursively until achieving perfect classification accuracy on the training data. Report the resulting tree's depth.

Depth
2

(8) Construct and illustrate the complete decision tree learned from the dataset above, utilizing mutual information for split selection and growing until zero training error is achieved. Mark each internal (non-terminal) node with its splitting feature (e.g. B), annotate edges with the corresponding feature values (e.g. 1 or 0), and label terminal nodes with their predicted class (e.g. $Y=0$). You may insert an image file using the provided commented LaTeX code, substituting your filename for *DecTree.png*. Hand-drawn diagrams are acceptable. For programmatic visualization, you may optionally use libraries such as `matplotlib` or `graphviz`.

Decision Tree



(9) Apply your learned decision tree to classify the test instances listed below. The table displays the feature values for reference. Record the predictions for all samples sequentially, delimited by commas. For instance, if your predictions are 1,0,1,0, format as: 1,0,1,0

A	B	C
1	1	0
1	2	0
0	1	0
0	2	0
0	1	1
1	1	1

Predicted Labels

1,1,0,1,0,0

Problem 6 : Logistic Regression

(1) Complete the following analytical exercises prior to beginning the implementation portion of this assignment.

You are provided with a training set containing 4 observations. For each sample $i \in \{1, 2, 3, 4\}$, the notation $x_k^{(i)}$ represents the value of the k-th feature ($k \in \{1, 2, 3\}$) in feature vector $\mathbf{x}^{(i)}$, while $y^{(i)}$ denotes the corresponding binary class label.

i	x_1	x_2	x_3	y
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0

A binary classification model using logistic regression has been fitted to this data, yielding the learned weight vector:

$$\boldsymbol{\theta} = \begin{bmatrix} 1.5 & 2 & 1 \end{bmatrix}^T.$$

Note: This model excludes the **bias term** (operates without an intercept).

Employ the dataset above to address the subsequent questions. Report all numerical results as single values rounded to four decimal places (format: 0.1234). Presenting your derivations is optional but encouraged for partial credit consideration.

(a) Evaluate $J(\boldsymbol{\theta})$, defined as $\frac{1}{N}$ multiplied by the negative log-likelihood computed across the provided data using parameter vector $\boldsymbol{\theta}$. (Apply natural logarithm with base e throughout).

$J(\boldsymbol{\theta})$

0.7975

Work

$J(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log(\sigma(\boldsymbol{\theta}^T x^{(i)})) + (1-y^{(i)})\log(1-\sigma(\boldsymbol{\theta}^T x^{(i)}))\right]$
$= \frac{1}{N}\sum_{i=1}^{N} -y^{(i)}(\boldsymbol{\theta}^T \mathbf{x}^{(i)}) + \log(1+\exp(\boldsymbol{\theta}^T \mathbf{x}^{(i)}))$
$= \frac{1}{4}\left[(-0+\log(1+e^1)) + (-2+\log(1+e^2)) + (-3+\log(1+e^3)) + (-0+\log(1+e^{1.5}))\right]$
≈ 0.7975

(b) Compute the partial derivatives $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j}$ for each weight component θ_j where $j \in \{1, 2, 3\}$.

$\partial J(\boldsymbol{\theta})/\partial \theta_1$	$\partial J(\boldsymbol{\theta})/\partial \theta_2$	$\partial J(\boldsymbol{\theta})/\partial \theta_3$
0.2044	-0.0417	0.1709

Work

$\frac{\partial J}{\partial \theta_j} = \frac{1}{N}\sum_{i=1}^{N}(\sigma(\boldsymbol{\theta}^T x^{(i)}) - y^{(i)})x_j^{(i)}$
where $\sigma(z) = \frac{1}{1+\exp(-z)}$
For θ_1:
$\frac{\partial J}{\partial \theta_1} = \frac{1}{4}[(\sigma(1.5\times 0+2\times 0+1\times 1)-0)\times 0+(\sigma(1.5\times 0+2\times 1+1\times 0)-1)\times 0+(\sigma(1.5\times 0+2\times 1+1\times 1)-1)\times 0+(\sigma(1.5\times 1+2\times 0+1\times 0)-0)\times 1]$
$= \frac{1}{4}[0+0+0+\sigma(1.5)] \approx 0.2044$
For θ_2:
$\frac{\partial J}{\partial \theta_2} = \frac{1}{4}[(\sigma(1.5\times 0+2\times 0+1\times 1)-0)\times 0+(\sigma(1.5\times 0+2\times 1+1\times 0)-1)\times 1+(\sigma(1.5\times 0+2\times 1+1\times 1)-1)\times 1+(\sigma(1.5\times 1+2\times 0+1\times 0)-0)\times 0]$
$= \frac{1}{4}[0+(\sigma(2)-1)+(\sigma(3)-1)+0] \approx -0.0417$
For θ_3:
$\frac{\partial J}{\partial \theta_3} = \frac{1}{4}[(\sigma(1.5\times 0+2\times 0+1\times 1)-0)\times 1+(\sigma(1.5\times 0+2\times 1+1\times 0)-1)\times 0+(\sigma(1.5\times 0+2\times 1+1\times 1)-1)\times 1+(\sigma(1.5\times 1+2\times 0+1\times 0)-0)\times 0]$
$= \frac{1}{4}[(\sigma(1)-0)\times 1+(\sigma(3)-1)+0] \approx 0.1709$

(c) Apply the gradient descent update rule $\theta_j \leftarrow \theta_j - \eta \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j}$ to revise each parameter, then report the resulting numerical values of the updated weight vector $\boldsymbol{\theta}$. Set step size $\eta = 1$.

θ_1	θ_2	θ_3
1.2956	2.0417	0.8291

Work

$\theta_1 \leftarrow 1.5 - 1 \times 0.2044 \approx 1.2956$
$\theta_2 \leftarrow 2 - 1 \times (-0.0417) \approx 2.0417$
$\theta_3 \leftarrow 1 - 1 \times 0.1709 \approx 0.8291$

(2) (a) **Select all that apply:** Identify all valid statements regarding logistic regression:

- ■ The binary logistic regression framework accommodates both real-valued and discrete binary input features.
- ■ Without feature transformations, binary logistic regression produces a linear separating hyperplane in the feature space.
- ☐ The logistic sigmoid activation function exhibits convex geometry.
- ☐ None of the above.

(b) **Select all that apply:** Recognizing that the negative log-likelihood objective is convex, select all accurate claims about optimizing logistic regression via mini-batch stochastic gradient descent:

- ☐ The optimization procedure may become trapped at suboptimal local minima when applied to small-scale datasets.
- ☐ Convergence to the global optimum is guaranteed irrespective of the chosen step size.
- ☐ The optimization procedure may become trapped at suboptimal local minima when applied to large-scale datasets.
- ☐ Convergence to the global optimum is guaranteed irrespective of the chosen batch size.
- ■ None of the above.

(c) **Select one:** For binary logistic regression, the *mean* negative log-likelihood $J(\boldsymbol{\theta})$ admits

the representation

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[-y^{(i)}\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right) + \log\left(1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})\right)\right]$$

where $\mathbf{x}^{(i)} \in \mathbb{R}^{M+1}$ denotes the feature vector for sample i, $y^{(i)} \in \{0,1\}$ represents the binary label for sample i, and $\boldsymbol{\theta} \in \mathbb{R}^{M+1}$ is the parameter vector. To implement ridge-regularized logistic regression (incorporating ℓ_2 penalty), we augment the objective as follows:

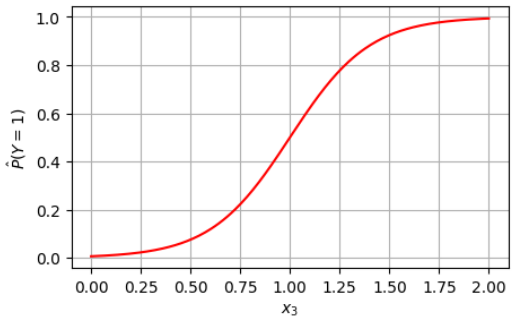
$$f(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda\frac{1}{2}\sum_{j=0}^{M}\theta_j^2$$

where λ controls regularization strength and θ_j denotes the j-th component of $\boldsymbol{\theta}$. When performing gradient-based optimization with step size η to update parameter θ_k, identify the correct update formula:

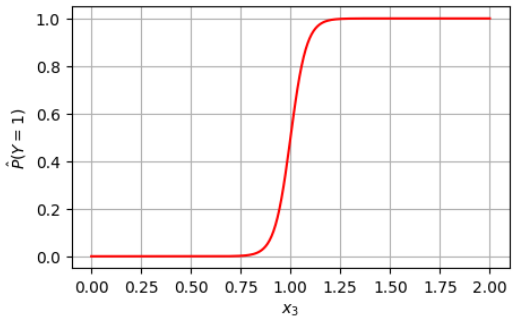
- ☐ $\theta_k \leftarrow \theta_k + \eta\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(y^{(i)} - \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] + \lambda\theta_k$
- ☐ $\theta_k \leftarrow \theta_k + \eta\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(-y^{(i)} + \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] - \lambda\theta_k$
- ■ $\theta_k \leftarrow \theta_k - \eta\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(-y^{(i)} + \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] + \lambda\theta_k$
- ☐ $\theta_k \leftarrow \theta_k - \eta\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(-y^{(i)} - \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] + \lambda\theta_k$

(d) Consider fitting a logistic regression classifier to training data $\mathcal{D}_{train}$ comprising M input features and binary outputs $\mathbf{y} \in \{0,1\}$. The third predictor, $\mathbf{x}_3$, perfectly partitions the training samples. Specifically, all instances satisfying $\mathbf{x}_3 \geq 1$ belong to class $y = 1$, while the feature $\mathbf{x}_3$ takes values uniformly distributed over the interval [0, 2].

(i) **Select one:** The plots below illustrate how the predicted probability $\hat{P}(y = 1)$ varies as a function of $\mathbf{x}_3$. After extensive training via SGD (numerous epochs, >> 100), which plot most accurately characterizes your model's behavior? **Justify your selection**.



(a)



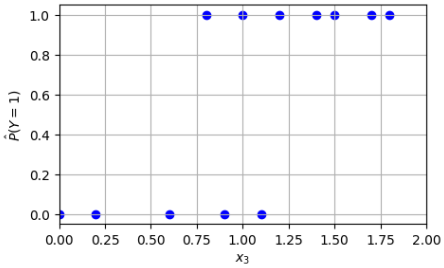
(b)

Hint: Imagine discovering a weight configuration that achieves complete linear separation of the data. Consider the implied probabilities for each sample and

whether further weight adjustments could enhance the likelihood. What trajectory would the coefficient for $\mathbf{x}_j$ follow?

Choice (b) is correct.
Since feature $\mathbf{x}_3$ perfectly separates the classes, logistic regression will attempt to push the decision boundary towards infinity to maximize the likelihood. As a result, the predicted probabilities will approach 0 for $\mathbf{x}_3 < 1$ and 1 for $\mathbf{x}_3 \geq 1$, leading to a step function-like behavior as shown in plot (b). Plot (a) would imply that the model is not fully utilizing the perfect separation provided by $\mathbf{x}_3$.

(ii) **Select one:** Following model training, you evaluate performance on a held-out validation set $\mathcal{D}_{validation}$, producing the visualization shown below.



A colleague named Neural reviews your results and recommends incorporating L2 regularization into your training objective.
Select one: What effect would regularization have on the learned parameter values?

- ☐ ℓ_2 regularization constrains parameter growth by normalizing the weight vector to unit norm.
- ☐ ℓ_2 regularization prevents parameter divergence by setting certain weights exactly to zero, effectively performing feature elimination.
- ■ ℓ_2 regularization controls parameter magnitude by shrinking weight values toward (but not necessarily to) zero, diminishing feature influence without complete removal.
- ☐ None of the above.

Problem 7 : PCA

(1) Consider a binary classification scenario with observations residing in two-dimensional space, partitioned into two classes:

$$c_1 = \left\{ \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix} \right\}, \quad c_2 = \left\{ \begin{bmatrix} 4 \\ 3 \end{bmatrix}, \begin{bmatrix} 5 \\ 3 \end{bmatrix}, \begin{bmatrix} 6 \\ 4 \end{bmatrix} \right\}$$

For this problem, although you are allowed to use Python for aiding computations, you are required to show all intermediate steps. You may use `numpy` for matrix operations (e.g., `numpy.linalg.eig`, `numpy.cov`) and `matplotlib.pyplot` for visualization.

Apply global PCA to identify the optimal projection direction for this dataset.

(a) Derive the linear equation characterizing this principal direction through the data centroid, formatted as $\mathbf{w}^T\mathbf{x} + w_0 = 0$, where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

with w_0 representing the offset parameter. Visualize this hyperplane $\mathbf{w}^T\mathbf{x} + w_0 = 0$ together with all data points and the direction vector $\mathbf{w}$ originating from the centroid.

(b) Transform all observations onto the principal component axis, then reconstruct them back to the original space. Visualize the resulting reconstructed coordinates.

(c) Calculate the aggregate mean squared error (MSE) quantifying reconstruction loss across all samples (measuring discrepancy between original and reconstructed coordinates).

(d) Compute the Fisher discriminant ratio for this projection, formulated as

$$FR = \frac{(m_1 - m_2)^2}{\sigma_1^2 + \sigma_2^2},$$

where m_i represents the centroid of class i samples after projection, and σ_i^2 denotes the corresponding variance. Evaluate FR using the one-dimensional projected coordinates (not the two-dimensional reconstructions).

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

(a)

[2]:
```
c1 = np.array([[2, 1],
               [2, 2],
               [2, 3]], dtype=float)
c2 = np.array([[4, 3],
               [5, 3],
               [6, 4]], dtype=float)
X = np.vstack((c1, c2))
X
```

[2]:
```
array([[2., 1.],
       [2., 2.],
       [2., 3.],
       [4., 3.],
       [5., 3.],
       [6., 4.]])
```

Compute the covariance matrix:

[3]:
```
cov_matrix = np.cov(X.T)
cov_matrix
```

[3]:
```
array([[3.1       , 1.4       ],
       [1.4       , 1.06666667]])
```

Compute eigenvalues and eigenvectors:

[4]:
```
eigvals, eigvecs = np.linalg.eig(cov_matrix)
eigvals, eigvecs
```

[4]:
```
(array([3.81353884, 0.35312782]),
 array([[ 0.89095421, -0.45409317],
        [ 0.45409317,  0.89095421]]))
```

Select the eigenvector corresponding to the largest eigenvalue (principal direction):

[5]:
```
max_index = np.argmax(eigvals)
principal_direction = eigvecs[:, max_index]
principal_direction = principal_direction / np.linalg.norm(principal_direction)
max_index, principal_direction
```

[5]:
```
(0, array([0.89095421, 0.45409317]))
```

Compute the line equation through the centroid:

[6]:
```
# Equation form: w^T x + w0 = 0, where w0 = -w^T * centroid
centroid = np.mean(X, axis=0)
w0 = -np.dot(principal_direction, centroid)
w0
```

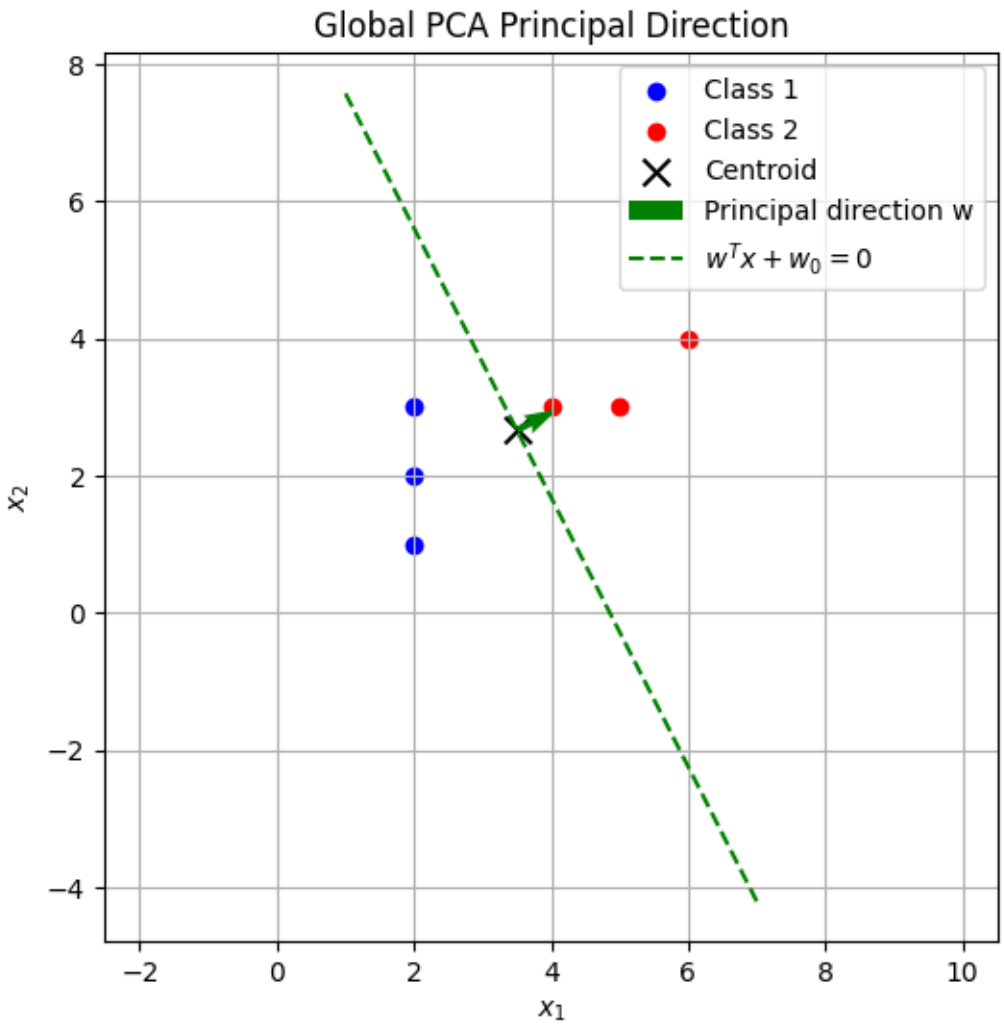
[6]:
```
-4.329254830199028
```

[7]:
```
print(f"\nLinear equation: {principal_direction[0]:.4f} * x1 +␣
↪{principal_direction[1]:.4f} * x2 + ({w0:.4f}) = 0")
```

```
Linear equation: 0.8910 * x1 + 0.4541 * x2 + (-4.3293) = 0
```

Visualization:

[8]:
```
plt.figure(figsize=(6, 6))
# Plot data points of both classes
plt.scatter(c1[:, 0], c1[:, 1], color='blue', label='Class 1')
plt.scatter(c2[:, 0], c2[:, 1], color='red', label='Class 2')
# Plot centroid
plt.scatter(*centroid, color='black', marker='x', s=100, label='Centroid')
# Plot the principal direction vector
plt.quiver(centroid[0], centroid[1], principal_direction[0],␣
↪principal_direction[1],
          angles='xy', scale_units='xy', scale=1.5, color='green',␣
↪label='Principal direction w')
# Plot the line passing through the centroid
x_vals = np.linspace(1, 7, 100)
y_vals = -(principal_direction[0] * x_vals + w0) / principal_direction[1]
plt.plot(x_vals, y_vals, 'g--', label=r'$w^T x + w_0 = 0$')
# Plot settings
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.axis('equal')
plt.legend()
plt.grid(True)
plt.title("Global PCA Principal Direction")
plt.show()
```



(b)

Project centered data onto principal component to obtain scalar projections z

[9]:
```
# equation: z_i = w^T (x_i - centroid)
X_centered = X - centroid  # shape (n, 2)
z = X_centered.dot(principal_direction)  # shape (n,)
z
```

[9]:
```
array([-2.09325325, -1.63916009, -1.18506692,  0.59684149,  1.4877957 ,
        2.83284307])
```

Reconstruct back to original space:

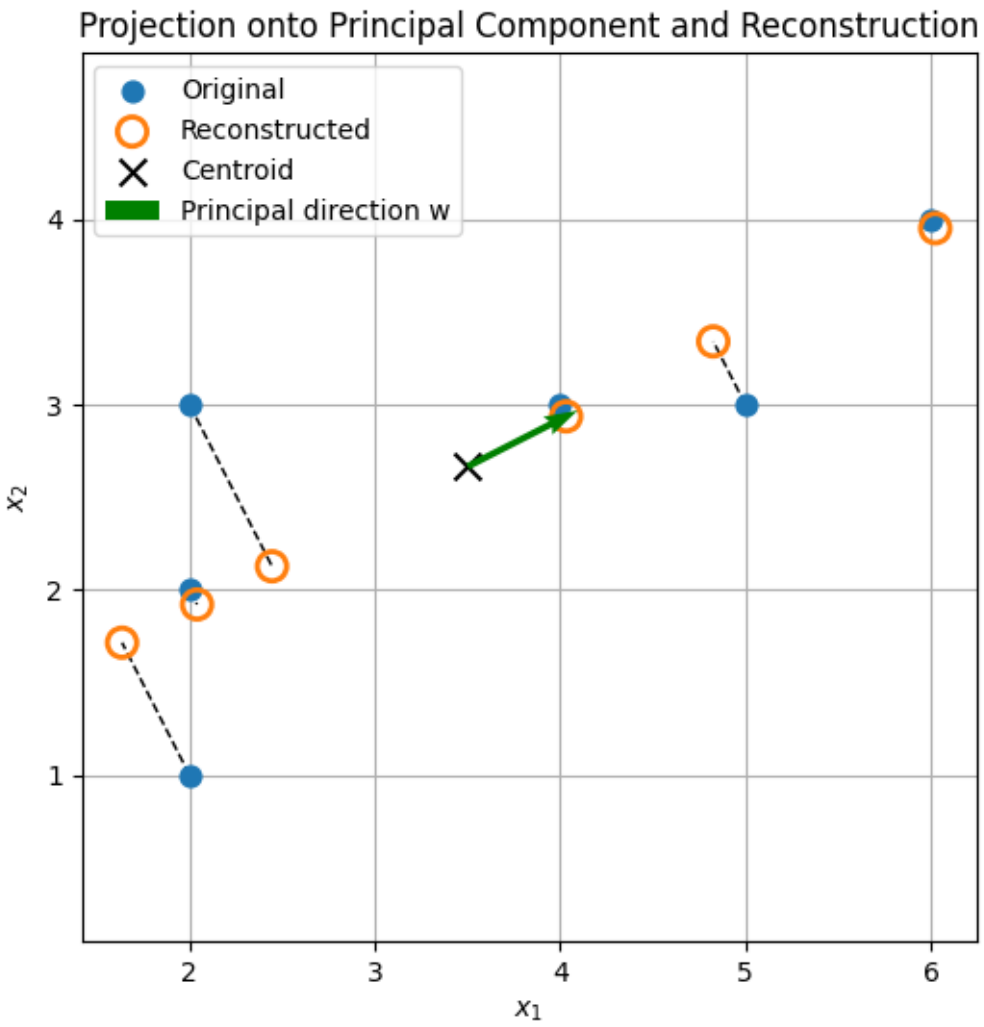
[10]:
```
# equation: x_hat_i = centroid + z_i * w
X_recon = np.outer(z, principal_direction) + centroid
```

```
X_recon
```

[10]:
```
array([[1.63500721, 1.71613467],
       [2.03958343, 1.92233527],
       [2.44415965, 2.12853588],
       [4.03175844, 2.93768831],
       [4.82555783, 3.34226453],
       [6.02393344, 3.95304135]])
```

Visualization: original vs reconstructed, connect with lines

[11]:
```
plt.figure(figsize=(6,6))

# original points (filled)
plt.scatter(X[:,0], X[:,1], marker='o', s=60, label='Original', zorder=3)

# reconstructed points (hollow)
plt.scatter(X_recon[:,0], X_recon[:,1], facecolors='none', edgecolors='tab:
 ↪orange',
            marker='o', s=120, linewidths=2, label='Reconstructed', zorder=4)

# connect each original point to its reconstruction
for i in range(X.shape[0]):
    plt.plot([X[i,0], X_recon[i,0]], [X[i,1], X_recon[i,1]], 'k--', linewidth=1)

# plot centroid and principal direction
plt.scatter(*centroid, color='black', marker='x', s=100, label='Centroid',␣
 ↪zorder=5)
plt.quiver(centroid[0], centroid[1], principal_direction[0],␣
 ↪principal_direction[1],
           angles='xy', scale_units='xy', scale=1.5, color='green',␣
 ↪label='Principal direction w', zorder=6)

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.axis('equal')
plt.grid(True)
plt.legend()
plt.title("Projection onto Principal Component and Reconstruction")
plt.show()
```



(c)

Compute reconstruction errors and MSE:

[12]:
```
# equation: error_i = x_i - x_hat_i
errors = X - X_recon
squared_errors = np.sum(errors**2, axis=1)
mse_total = np.mean(squared_errors)
print("\nPer-point squared reconstruction errors :")
for i, se in enumerate(squared_errors):
    print(f"Point {i+1}: {se:.6f}")
print(f"\nMean Squared Error (MSE): {mse_total:.6f}")
```

```
Per-point squared reconstruction errors :
Point 1: 0.646069
```

```
Point 2: 0.007599
Point 3: 0.956728
Point 4: 0.004891
Point 5: 0.147575
Point 6: 0.002778

Mean Squared Error (MSE): 0.294273
```

(d)

Split projections by class

[13]:
```
z1 = z[:3]
z2 = z[3:]
z1, z2
```

[13]:
```
(array([-2.09325325, -1.63916009, -1.18506692]),
 array([0.59684149, 1.4877957 , 2.83284307]))
```

Compute per-class mean and variance:

[14]:
```
m1 = np.mean(z1)
m2 = np.mean(z2)
sigma1_sq = np.var(z1, ddof=0)
sigma2_sq = np.var(z2, ddof=0)
m1, sigma1_sq, m2, sigma2_sq
```

[14]:
```
(-1.6391600857157167,
 0.13746706965069366,
 1.6391600857157167,
 0.8447394314242658)
```

Calculate Fisher discriminant ratio:

FR = (m1 - m2)^2 / (sigma1^2 + sigma2^2)

[15]:
```
FR = (m1 - m2)**2 / (sigma1_sq + sigma2_sq)
print(f"\nFisher's Ratio (FR): {FR:.4f}")
```

```
Fisher's Ratio (FR): 10.9421
```

(2) An alternative perspective on PCA views it as identifying the low-dimensional subspace that minimizes reconstruction error from dimensionality reduction. Remarkably, the variance-maximization and error-minimization objectives yield identical PCA solutions.

(a) Suppose all observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ have been mean-centered. State the constrained optimization task for determining a direction vector $\mathbf{w}$ that defines a linear subspace minimizing the total squared reconstruction loss

$$\sum_{i=1}^{n} d_i^2$$

where d_i quantifies the reconstruction discrepancy for $\mathbf{x}_i$, measured as the Euclidean distance from $\mathbf{x}_i$ to its orthogonal projection onto the subspace. Impose unit length constraint $\|\mathbf{w}\| = 1$ since only direction matters. Express your objective solely using $\mathbf{w}$ and $\mathbf{x}_i$, eliminating d_i.

The dataset $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ has been mean-centered, i.e. $\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$.
We seek a unit-length direction vector $\mathbf{w}$ (i.e. $\|\mathbf{w}\|_2 = 1$) that defines a one-dimensional linear subspace onto which we orthogonally project each $\mathbf{x}_i$.
The orthogonal projection of $\mathbf{x}_i$ onto $\text{span}\{\mathbf{w}\}$ is $(\mathbf{w}^\top \mathbf{x}_i)\mathbf{w}$.
The squared reconstruction error for $\mathbf{x}_i$ is therefore

$$d_i^2 \;=\; \left\|\mathbf{x}_i - (\mathbf{w}^\top \mathbf{x}_i)\mathbf{w}\right\|_2^2.$$

The total squared reconstruction loss over all samples is $\sum_{i=1}^n d_i^2$.
Thus the constrained optimization problem (with the unit-norm constraint) is

$$\boxed{\min_{\mathbf{w}} \sum_{i=1}^{n} \left\|\mathbf{x}_i - (\mathbf{w}^\top \mathbf{x}_i)\mathbf{w}\right\|_2^2 \quad \text{s.t.} \quad \|\mathbf{w}\|_2 = 1}.$$

(b) Using your formulation from (a), demonstrate that this minimization problem coincides with standard PCA when recognizing:

$$\Sigma = \sum_i \mathbf{x}_i \mathbf{x}_i^T$$

We'll prove that the above minimization is equivalent to maximizing $\mathbf{w}^\top \Sigma \mathbf{w}$ where $\Sigma = \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top$.
Start by expanding each term:

$$\begin{aligned}\left\|\mathbf{x}_i - (\mathbf{w}^\top \mathbf{x}_i)\mathbf{w}\right\|_2^2 &= (\mathbf{x}_i - (\mathbf{w}^\top \mathbf{x}_i)\mathbf{w})^\top (\mathbf{x}_i - (\mathbf{w}^\top \mathbf{x}_i)\mathbf{w}) \\ &= \left[\mathbf{x}_i^\top - w^\top (\mathbf{x}_i^\top \mathbf{w})\right]\left[\mathbf{x}_i - (\mathbf{w}^\top \mathbf{x}_i)\mathbf{w}\right] \\ &= \mathbf{x}_i^\top \mathbf{x}_i - 2(\mathbf{w}^\top \mathbf{x}_i)(\mathbf{w}^\top \mathbf{x}_i) + (\mathbf{w}^\top \mathbf{x}_i)^2 (\mathbf{w}^\top \mathbf{w}) \\ &= \|\mathbf{x}_i\|_2^2 - (\mathbf{w}^\top \mathbf{x}_i)^2,\end{aligned}$$

where in the second line we used $\mathbf{w}^\top \mathbf{w} = \|\mathbf{w}\|_2^2 = 1$.
Summing over i yields

$$\sum_{i=1}^n d_i^2 \;=\; \sum_{i=1}^n \|\mathbf{x}_i\|_2^2 \;-\; \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i)^2.$$

The first term $\sum_i \|\mathbf{x}_i\|_2^2$ does not depend on $\mathbf{w}$, so minimizing the left-hand side over unit $\mathbf{w}$ is equivalent to maximizing the second term:

$$\max_{\|\mathbf{w}\|=1} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i)^2.$$

Rewrite the objective in matrix form:

$$\sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i)^2 = \sum_{i=1}^n \mathbf{w}^\top (\mathbf{x}_i \mathbf{x}_i^\top)\mathbf{w} = \mathbf{w}^\top \left(\sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top\right) \mathbf{w} = \mathbf{w}^\top \Sigma \mathbf{w}.$$

Therefore the constrained minimization of total reconstruction error is equivalent to the Rayleigh quotient maximization

$$\max_{\|\mathbf{w}\|_2=1} \mathbf{w}^\top \Sigma \mathbf{w}.$$

By classical results, the maximizer is the eigenvector of Σ associated with the largest eigenvalue. This eigenvector is precisely the first principal component obtained by standard PCA (using Σ as the covariance matrix). Hence the reconstruction-error minimization formulation yields the same principal direction as the variance-maximization PCA formulation.

(3) Acquire the MNIST handwritten digit dataset from `http://yann.lecun.com/exdb/mnist/`. Various Python utilities exist online for parsing the native format, or retrieve pre-processed Python-compatible data from `http://cs.nyu.edu/roweis/data.html`. You may use libraries such as `keras.datasets.mnist`, `torchvision.datasets.MNIST`, or `sklearn.datasets.fetch_openml` for convenient data loading.

Develop your own PCA implementation without relying on pre-built library functions. You may utilize Python's built-in routines such as `numpy.linalg.svd` for SVD and `numpy.linalg.eig` (or `numpy.linalg.eigh` for symmetric matrices) for eigendecomposition. Include source code printout with your submission.

Memory constraints: If computational resources prove insufficient, subsample to the first 1000 images per digit class. Should memory issues persist, further reduce to 100 images per class.

(a) Visualize the average digit-1 template and display the leading 5 principal components

extracted from the complete dataset under two computational approaches: employing the Gram matrix technique versus the standard covariance approach. Ensure mean-centering precedes PCA execution. Record computational time for principal component extraction in both scenarios. You may use `matplotlib.pyplot.imshow` for visualization and Python's `time` module (e.g., `time.time()`) for timing measurements.

(b) Analyze the computational efficiency of each approach. Justify whether the Gram matrix strategy offers advantages for this specific dataset.

(c) Select an arbitrary image (from any digit class) and transform it to the principal component basis (the global eigenspace from part (a)), then reconstruct using the leading n components (cumulative up to n, not just component n) for both computational methods (Gram versus standard), with $n \in \{1, 2, 5, 10, 20\}$. Apply mean-centering before projection and restore the mean post-reconstruction.

For each of the 5 reconstructions, quantify fidelity via mean squared error. The MSE metric for vectors $\mathbf{a}$ and $\mathbf{b}$ is

$$\text{MSE}(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_2^2$$

Present all reconstructions alongside the original in a composite figure, annotating each panel with its corresponding MSE. You may use `matplotlib.pyplot.subplot` to create composite figures and `numpy.linalg.norm` for computing Euclidean norms.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import time
from torchvision import datasets
from torchvision.transforms import ToTensor
```

(a)

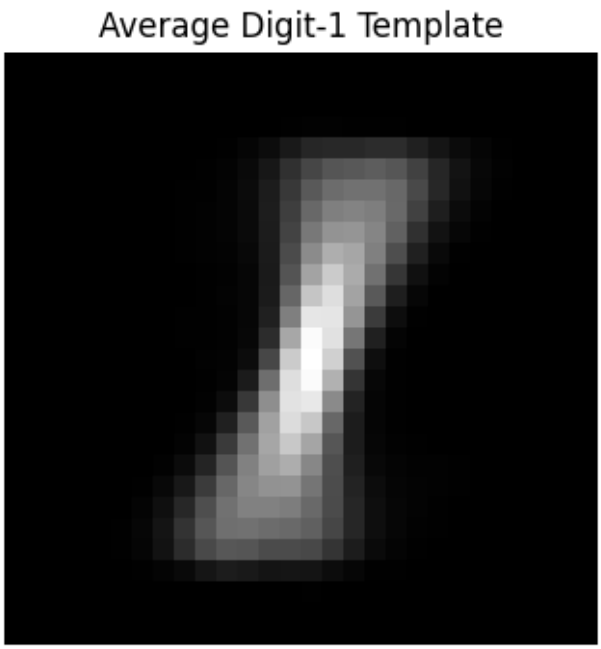
Data loading and preprocessing:

[2]:
```
# Download MNIST dataset
mnist_train = datasets.MNIST(root='./data', train=True, download=True,␣
 ↪transform=ToTensor())
# Convert to a numpy array and flatten
x_train_flat = np.array([img.numpy().reshape(-1) for img, label in␣
 ↪mnist_train], dtype=np.float64)
y_train = np.array([label for img, label in mnist_train])
# Select the first 1000 images in each class
x_train_flat = np.vstack([x_train_flat[y_train == i][:1000] for i in range(10)])
y_train = np.hstack([y_train[y_train == i][:1000] for i in range(10)])

x_train_flat.shape, y_train.shape
```

[2]:
```
((10000, 784), (10000,))
```

Visualize the average digit-1 template:

[3]:
```
digit1_images = x_train_flat[y_train == 1]
digit1_mean = np.mean(digit1_images, axis=0)

plt.figure(figsize=(4, 4))
plt.imshow(digit1_mean.reshape(28, 28), cmap='gray')
plt.title("Average Digit-1 Template")
plt.axis('off')
plt.show()
```



Calculate PCA using two methods:

The Gram matrix technique and the standard covariance approach.

[4]:
```
# The Gram matrix: G = XX^T
def pca_gram(X, n_components):
    start_time = time.time()
    mean_X = np.mean(X, axis=0)
    X = X - mean_X
    G = X @ X.T
    eigvals, eigvecs = np.linalg.eigh(G)
    idx = np.argsort(eigvals)[::-1]
    eigvals = eigvals[idx][:n_components]
    eigvecs = eigvecs[:, idx][:, :n_components]
    eigvecs = X.T @ eigvecs
    eigvecs = eigvecs / np.linalg.norm(eigvecs, axis=0)
    end_time = time.time()
    return eigvecs, eigvals, mean_X, end_time - start_time
```

[5]:
```
# Standard covariance approach: C = (1/n)X^TX
def pca_std(X, n_components):
    start_time = time.time()
    mean_X = np.mean(X, axis=0)
    X = X - mean_X
    C = (X.T @ X) / X.shape[0]
    eigvals, eigvecs = np.linalg.eigh(C)
```

```
    idx = np.argsort(eigvals)[::-1]
    eigvals = eigvals[idx][:n_components]
    eigvecs = eigvecs[:, idx][:, :n_components]
    end_time = time.time()
    return eigvecs, eigvals, mean_X, end_time - start_time
```

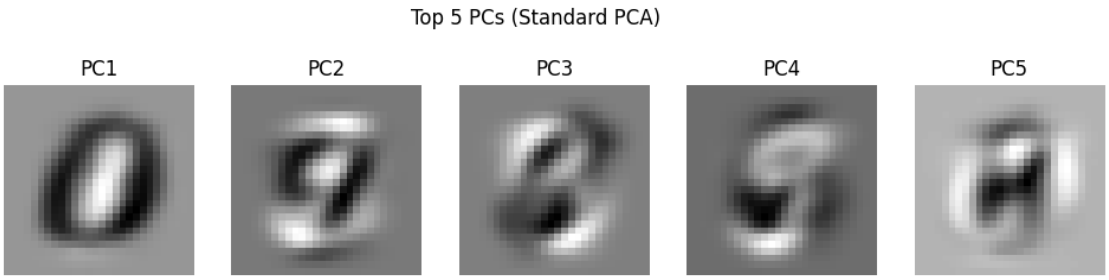
Record computational time:

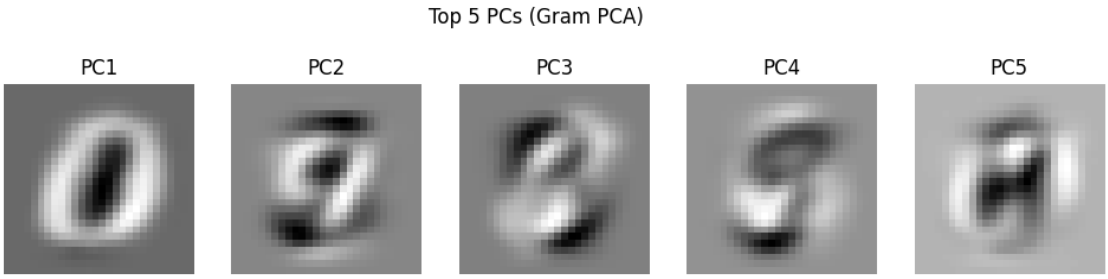
[6]:
```
n_components = 5
eigvecs_std, eigvals_std, mean_X_std, time_std = pca_std(x_train_flat,␣
 ↪n_components)
eigvecs_gram, eigvals_gram, mean_X_gram, time_gram = pca_gram(x_train_flat,␣
 ↪n_components)

print(f"Standard PCA time: {time_std:.4f}s")
print(f"Gram PCA time: {time_gram:.4f}s")
```

```
Standard PCA time: 0.3533s
Gram PCA time: 140.7519s
```

Visualization:

[7]:
```
def plot_pcs(pcs, title):
    plt.figure(figsize=(12, 3))
    for i in range(pcs.shape[1]):
        plt.subplot(1, pcs.shape[1], i+1)
        plt.imshow(pcs[:, i].reshape(28, 28), cmap='gray')
        plt.axis('off')
        plt.title(f'PC{i+1}')
    plt.suptitle(title)
    plt.show()

plot_pcs(eigvecs_std, "Top 5 PCs (Standard PCA)")
plot_pcs(eigvecs_gram, "Top 5 PCs (Gram PCA)")
```





(b)

In the MNIST dataset, the number of samples (1000) is much larger than the feature dimension of each image (784). Since the Gram matrix method is only more efficient when the number of samples is smaller than the number of features, using the Gram matrix approach (based on $XX^\top$) does not provide any computational advantage and may actually slow down the process due to the larger matrix size.

(c)

Image reconstruct and MSE

[8]:
```
def reconstruct_image(img, eigvecs, mean_X, n_list=[1, 2, 5, 10, 20]):
    img_centered = img - mean_X
    results = []
    for n in n_list:
        pcs = eigvecs[:, :n] # choose top-n PCs
        coeffs = pcs.T @ img_centered # projection coefficients
        recon = pcs @ coeffs + mean_X # reconstruction
        mse = np.linalg.norm(img - recon)**2
        results.append((recon, mse))
    return results

img_idx = 0
img_original = x_train_flat[img_idx]

recon_std = reconstruct_image(img_original, eigvecs_std, mean_X_std)
recon_gram = reconstruct_image(img_original, eigvecs_gram, mean_X_gram)
```

Visualization:

[9]:
```
def plot_reconstructions(original, recon_list, method_name,␣
 ↪n_list=[1,2,5,10,20]):
    plt.figure(figsize=(15, 3))
    plt.subplot(1, len(n_list)+1, 1)
    plt.imshow(original.reshape(28,28), cmap='gray')
    plt.title("Original")
    plt.axis('off')
```

```
    for i, (recon, mse) in enumerate(recon_list):
        plt.subplot(1, len(n_list)+1, i+2)
        plt.imshow(recon.reshape(28,28), cmap='gray')
        plt.title(f"n={n_list[i]}\nMSE={mse:.1f}")
        plt.axis('off')
    plt.suptitle(f"Reconstructions using {method_name}")
    plt.show()

plot_reconstructions(img_original, recon_std, "Standard PCA")
plot_reconstructions(img_original, recon_gram, "Gram PCA")
```