1. Multiple Choice (5 × 4 = 20 P)

Answer the following multiple choice questions. There is only one good answer per question. To mark an answer, put an × in the ☐ next to it. For each question, no or false answer is zero points, correct answer is full points.

(a) Which of the following is **True** about Random Forest and Boosting?

- ☐ Random Forest reduces bias by sequentially fitting residuals.
- ☐ Boosting combines weak learners through a weighted sum.
- ☐ Boosting trains all base learners independently using bootstrap samples.
- ☐ Random Forest does not use bootstrap sampling.

(b) Which of the following is **True** about kernel functions?

- ☐ Every symmetric and continuous function is a valid kernel.
- ☐ A function is a valid kernel if and only if the corresponding Gram matrix is positive semidefinite for all possible datasets.
- ☐ Kernel methods require the explicit construction of the feature map $\phi(x)$.
- ☐ A valid kernel must satisfy the triangle inequality.

(c) Which of the following is **True** about the VC dimension?

- ☐ The VC dimension is equal to the number of parameters of a model.
- ☐ The VC dimension is the maximum number of points that can be shattered by a hypothesis class.
- ☐ A higher VC dimension always guarantees better generalization performance.
- ☐ The VC dimension is defined only for linear classifiers.

(d) Which of the following is **True** about decision trees?

- ☐ They partition the feature space using axis-aligned splits at each node.
- ☐ They always find the globally optimal tree structure.
- ☐ They are immune to overfitting, regardless of tree depth.
- ☐ They require the input features to be continuous.

(e) Which of the following is **True** about the backpropagation algorithm?

- ☐ It computes gradients of the loss by applying the chain rule from output to input layers.
- ☐ It requires the network's activation functions to be linear.
- ☐ It can only be applied to fully connected feed-forward networks.
- ☐ It computes each parameter's gradient independently by perturbing one parameter at a time.

2. Maximum Likelihood and Bayes (5 + 5 + 3 + 7 = 20 P)

The geometric distribution is given by

$$P(x \mid \theta) = \theta \cdot (1-\theta)^{x-1}$$

where $x \in \mathbb{Z}^+$ (positive integers) and $0 < \theta < 1$ is a parameter. Let $\mathcal{D} = \{x_1, \ldots, x_N\}$ be a dataset of independent draws from that distribution. We would like to learn the parameter θ from data.

(a) *Write* the likelihood function $P(\mathcal{D} \mid \theta)$ and *simplify* it to a closed-form expression in θ, N, and $\sum_i x_i$.

[redacted]

(b) *Compute* the maximum likelihood solution for $\hat{\theta}$ given the dataset $\mathcal{D} = \{1, 2, 4\}$ and the probability $P(x_4 = 1 \mid \hat{\theta})$.

[redacted]

Now, we adopt the Bayesian viewpoint and assume that the parameter θ has prior probability

$$p(\theta) = \begin{cases} 2(1-\theta) & \text{if } 0 < \theta < 1 \\ 0 & \text{otherwise} \end{cases}$$

(c) *Show* that the posterior distribution can be written as

$$p(\theta \mid \mathcal{D}) = \frac{\theta^a (1-\theta)^b}{\int_0^1 \theta^a (1-\theta)^b \, d\theta}$$

and *determine* a and b.

[redacted]

(d) *Evaluate* the Bayesian predictive probability

$$P(x_4 = 1 \mid \mathcal{D}) = \int_0^1 P(x_4 = 1 \mid \theta) \, p(\theta \mid \mathcal{D}) \, d\theta.$$

Hint: Express this as a ratio of two integrals and use the identity $\int_0^1 \theta^a (1-\theta)^b \, d\theta = \frac{a!\, b!}{(a+b+1)!}$ *for non-negative integers* a *and* b.

3. Principal Component Analysis (6 + 6 + 8 = 20 P)

Let $C \in \mathbb{R}^{d \times d}$ be a symmetric covariance matrix, i.e. $C = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$ with $\boldsymbol{x} \in \mathbb{R}^d$ and $\mathbb{E}[\boldsymbol{x}] = 0$.

PCA seeks a unit vector $\boldsymbol{u} \in \mathbb{R}^d$ that maximizes the variance of the projected data, which can be written as

$$\max_{\boldsymbol{u}} \boldsymbol{u}^\top C \boldsymbol{u}$$
$$\text{s.t. } \|\boldsymbol{u}\|_2 = 1$$

(a) Using the method of Lagrange multipliers, *derive* the necessary optimality condition for this problem.

(b) Show that the maximum of $\boldsymbol{u}^\top C \boldsymbol{u}$ under $\|\boldsymbol{u}\|_2 = 1$ is attained by the eigenvector corresponding to the largest eigenvalue of C.

(c) Let $(\boldsymbol{u}_1, \dots, \boldsymbol{u}_d)$ be an orthonormal eigenbasis of C with eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d \geq 0$. Assume that the data is centered, i.e. $\mathbb{E}[\boldsymbol{x}] = 0$. Define the rank-1 PCA reconstruction

$$\hat{\boldsymbol{x}} = (\boldsymbol{u}_1^\top \boldsymbol{x})\, \boldsymbol{u}_1.$$

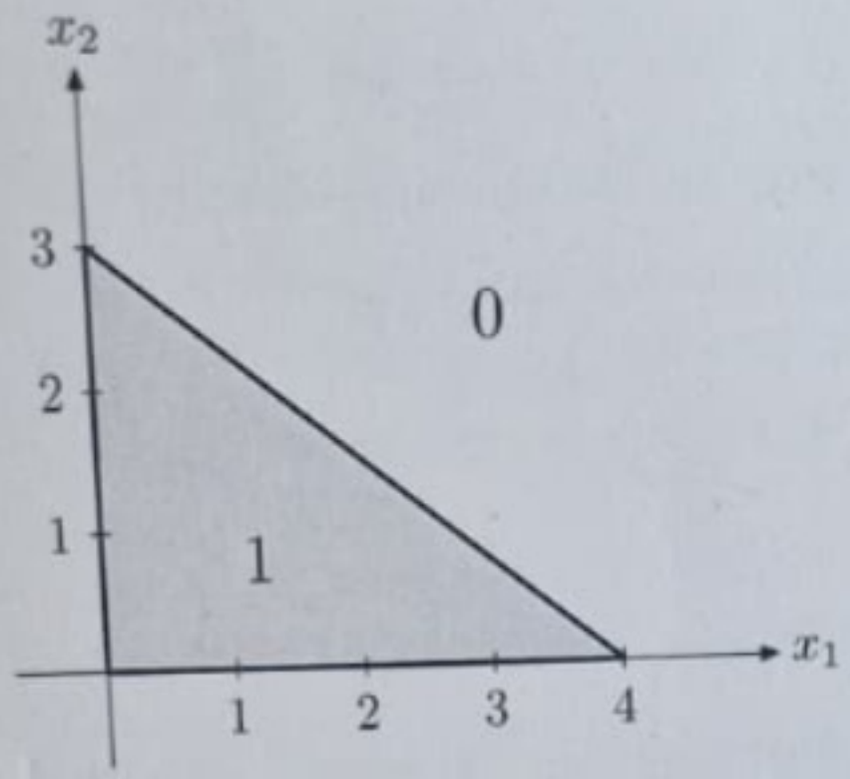
Show that the expected squared reconstruction error satisfies

$$\mathbb{E}[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2] = \sum_{i=2}^{d} \lambda_i.$$

Hint: Expand $\boldsymbol{x}$ in the eigenbasis.

4. Neural Networks (15 + 5 = 20 P)

The diagram below shows the decision boundary implemented by the function $f(\boldsymbol{x})$ for classifying data points $\boldsymbol{x} \in \mathbb{R}^2$.



We consider the neurons

$$a_j = \mathbf{1}_{\{\sum_i w_{ij} a_i + b_j \geq 0\}}$$

where a_i are the inputs of the neuron and w_{ij} and b_j are the real-valued parameters of the neuron.

(a) Construct a neural network composed exclusively of the elementary neurons above, that implements the function $f(\boldsymbol{x})$. *Annotate* your drawing with the relevant variable names, and the values of the neuron's parameters.

[redacted]

(b) Write the values of all neuron activations for the datapoint $\boldsymbol{x} = (3, 1)$ and *state* the classification $f(\boldsymbol{x})$.

5. Kernel Ridge Regression (10 + 10 = 20 P)

Given a training set `X` of size $N \times d$ and targets `y` of size N, we want to implement kernel ridge regression with the polynomial kernel

$$k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z} + c)^p .$$

Your implementations must be *efficient*, i.e. use vectorized NumPy operations on full matrices.

(a) *Implement* a function that computes the polynomial kernel matrix. If `Z` is `None`, return the $N \times N$ Gram matrix $K_{ij} = k(\mathtt{X}_i, \mathtt{X}_j)$. Otherwise, return the $N \times M$ matrix $K_{ij} = k(\mathtt{X}_i, \mathtt{Z}_j)$.

```
import numpy [redacted]

def kernel_matrix(X, Z=None, c=1.0, p=3):
    # X: N x d data matrix
    # Z: M x d data matrix, or None
    # c: offset parameter (non-negative scalar)
    # p: degree of the polynomial (positive integer)
[redacted]
    return K
```

(b) *Implement* the training function. Given training data X, targets $\boldsymbol{y}$, and regularization parameter $\lambda > 0$, return the coefficient vector $\alpha = (K + \lambda I)^{-1}\boldsymbol{y}$ where K is the Gram matrix of X. Your implementation should be *numerically stable*.

```
def fit(X, y, lam, c=1.0, p=3):
    # X:   N x d data matrix
    # y:   N target vector
    # lam: regularization parameter (positive scalar)
    # c:   offset parameter (non-negative scalar)
    # p:   degree of the polynomial (positive integer)
[redacted]
    return alpha
```