Q1. [10 marks total] This question is about general concepts in machine learning.

a) [2 marks] Why do we apply machine learning to problems like "spam detection" and "face detection" whereas for problems like "prime number detection" and "singular matrix detection" we do not?

Because we *know how to write programs* that solve problems like testing primality or singularity. For problems like spam detection and face detection we *don't know how to write programs* that solve these problems very well, and yet we have many examples to learn from.

b) [2 marks] What characterizes a learning problem as being *supervised*, and what is the goal?

In a supervised learning problem, the data is given as (input, output) pairs.
The goal is to learn a program that predicts outputs for new inputs.

c) [2 marks] What characterizes a learning problem as being *unsupervised*, and what is the goal?

In an unsupervised learning problem, the data is given as just examples (just inputs).
The goal is to find any "structure" in the data that might be useful for some later task.

d) [1 mark] Is *classification* a supervised learning problem, or is it an unsupervised learning problem? Why?

Classification is a supervised learning problem because for each training input we are given a class as the target output.

e) [2 marks] Suppose we have a pattern classification problem and we transform our features $\boldsymbol{x} \in \mathbb{R}^D$ by $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ for some matrix $\boldsymbol{A} \in \mathbb{R}^{M \times D}$ where $M > D$. According to Cover's theorem, have we made our data more likely to be linearly separable? Explain.

No we have not changed the linear separability of the data, even though the data is now in a higher dimension. Cover's theorem only applies if we cast in a high-dimensional space *non-linearly*, but $\boldsymbol{\phi}(\boldsymbol{x})$ is a linear function of $\boldsymbol{x}$, so Cover's theorem does not apply.

f) [1 mark] What is the defining feature of a *parametric* model?

A parametric model has a fixed number of parameters that we can tune, irrespective of the amount of training data available.

Q2. [10 marks total] This question is about *basic linear regression*.

a) [1 mark] What is the mathematical expression for how *basic linear model* $\hat{y}(\boldsymbol{x}, \boldsymbol{w})$ compute a prediction?

$$\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = \qquad \boldsymbol{w}^T \boldsymbol{x}$$

b) [1 mark] We often assume that at least one feature is a constant, as in $\boldsymbol{x} = \begin{bmatrix} 1 & x_1 & \cdots & x_D \end{bmatrix}$. If we did not include this constant, how would that impact the kind of predictions our linear model could make?

If we did not include the constant, we the hyperplane that we fit to the data would be forced to always pass through the origin. This can be seen from the fact that when $\boldsymbol{x} = \boldsymbol{0}$ we would necessarily have $\boldsymbol{w}^T \boldsymbol{x} = 0$ no matter what $\boldsymbol{w}$ was.

c) [3 marks] Write the loss function $\ell_{\text{LS}}(\boldsymbol{w})$ for *basic linear least squares regression*. Explain each symbol. If you don't know the answer to (a) just use $\hat{y}$ in its place.

The least squares loss is

$$\ell_{\text{LS}}(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 \quad \text{or} \quad \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}(\boldsymbol{x}_i, \boldsymbol{w}))^2$$

$\boldsymbol{w}$ is the parameter vector (weight vector)
$\boldsymbol{x}_i$ is the i^{th} feature vector.
y_i is the i^{th} training target.
N is the number of training samples.

d) [2 marks] Given a training set in matrix form $\boldsymbol{X} \in \mathbb{R}^{N\times(D+1)}, \boldsymbol{y} \in \mathbb{R}^N$ the gradient of the *basic least squares regression* loss can be written as

$$\nabla \ell_{\text{LS}}(\boldsymbol{w}) = (\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y}$$

Write the complete Python function definition for `linear_least_squares_grad(X, y, w)` , a function that returns the gradient at the current parameter settings $\boldsymbol{w} \in \mathbb{R}^{D+1}$. (You do not need to write a docstring.)

```
def linear_least_squares_grad(X, y, w):
    return (X.T @ X) @ w - X.T @ y
```

e) [2 marks] Derive an expression for the optimal parameter setting $\boldsymbol{w}^*$ for the basic linear least squares regression problem. Show your steps.

Since linear least squares regression is a convex in $\boldsymbol{w}$ we can set the gradient to zero and solve for $\boldsymbol{w}^*$ directly.

$$\begin{aligned}
& 0 = (\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y} \\
\Rightarrow \quad & (\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y} \\
\Rightarrow \quad & \boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}
\end{aligned}$$

f) [1 mark] Repeat (e) for the *regularized linear least squares regression* where the loss $\ell_{\text{RLS}}(\boldsymbol{w}) = \ell_{\text{LS}}(\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$. Show your steps.

We have

$$\begin{aligned}
\nabla \ell_{\text{RLS}}(\boldsymbol{w}) &= \nabla \ell_{\text{LS}}(\boldsymbol{w}) + \nabla\left[\frac{\lambda}{2}\|\boldsymbol{w}\|^2\right] \\
&= \nabla \ell_{\text{LS}}(\boldsymbol{w}) + \lambda\boldsymbol{w}
\end{aligned}$$

Setting the left-hand side to zero gives:

$$\begin{aligned}
& 0 = (\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y} + \lambda\boldsymbol{w} \\
\Rightarrow \quad & (\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{w} + \lambda\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y} \\
\Rightarrow \quad & (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y} \\
\Rightarrow \quad & \boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}
\end{aligned}$$

Q3. [10 marks total] This question is about *logistic regression*.

a) [3 marks] Write the mathematical expression for how a *logistic regression model* $\hat{y}(\boldsymbol{x}, \boldsymbol{w})$ computes a prediction. Use only symbols e, $\boldsymbol{w}$, $\boldsymbol{\phi}$, $\boldsymbol{x}$ but not σ in your answer.

$$\hat{y}(\boldsymbol{x}, \boldsymbol{w}) = \qquad \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})}}$$

b) [1 mark] In logistic regression, what does the quantity $\hat{y}(\boldsymbol{x})$ represent in terms of probability?

It represents $p(y = 1 \mid \boldsymbol{x})$, the probability that input $\boldsymbol{x}$ belongs to class $y = 1$.

c) [1 mark] Is the decision boundary of logistic regression linear or non-linear within the feature space $\boldsymbol{\phi}$? Explain.

Linear. The standard decision boundary is all points $\boldsymbol{\phi}$ where $\sigma(\boldsymbol{w}^T \boldsymbol{\phi}) = 0.5$. This is the same set of points as those satisfying $\boldsymbol{w}^T \boldsymbol{\phi} = 0$. This is linear in $\boldsymbol{\phi}$ so the decision boundary will be linear in feature space. Even if we chose a different decision threshold than 0.5, say γ, the decision boundary would be all points $\boldsymbol{w}^T \boldsymbol{\phi} = \sigma^{-1}(\gamma)$ which is still linear in $\boldsymbol{\phi}$ since $\sigma^{-1}(\gamma)$ is constant.

d) [2 marks] Suppose you used feature transformation $\boldsymbol{\phi}(x) = \begin{bmatrix} 1 & 2^x & 3^x & \cdots & d^x \end{bmatrix}$ for some d. In what way(s) might a large d pose a challenge for logistic regression in practice?

First, it would result in a large feature vector and might over-fit to the data.
Second, the scale of the final features is likely to be be *much* larger than the earlier features, causing the gradient-based training algorithm to rely mainly on high-order terms. Feature normalization could help.

e) [2 marks] Write the formula for the *K*-class softmax function $\operatorname{softmax}(\boldsymbol{a})$ where $\boldsymbol{a} = \begin{bmatrix} a_1 & \cdots & a_K \end{bmatrix}$. If you only remember the 2-class case, write that for partial marks.

$$\operatorname{softmax}(\boldsymbol{a}) = \frac{\exp(\boldsymbol{a})}{\sum_{k=1}^{K} \exp(a_k)} \qquad \text{or} \quad \operatorname{softmax}(\boldsymbol{a})_k = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)} \quad \text{is OK too}$$

f) [1 mark**] You are given 1-dimensional data set $\mathcal{D} = \{(x_1, 0), (x_2, 1)\}$ and feature transformation $\boldsymbol{\phi}(x) = \begin{bmatrix} 1 & e^x \end{bmatrix}$. In terms of x_1 and x_2, where would the logistic regression decision boundary be in the original feature space x?

The new feature space is also 1-dimensional. Let $\phi_1 = e^{x_1}$ and $\phi_2 = e^{x_2}$ be the transformed features. The decision boundary in feature space would be at $\phi' = \frac{1}{2}(\phi_1 + \phi_2)$, *i.e.* the midpoint in feature space. The decision boundary in the original feature space would then be some x' such that $e^{x'} = \frac{1}{2}(e^{x_1} + e^{x_2})$. The decision boundary is at $x' = \ln\left(\frac{1}{2}(e^{x_1} + e^{x_2})\right)$.

Q4. [8 marks total] This question is about the *K-means* clustering algorithm.

a) [3 marks] What is the optimization problem that the *K*-means clustering algorithm tries to solve? Express your answer mathematically, then explain what each symbol means.

The *K*-means algorithm is minimizing the following problem:

$$\begin{aligned} \min_{\boldsymbol{r}, \boldsymbol{\mu}} \sum_{i=1}^{N} \sum_{k=1}^{K} r_{ik} \left\| \boldsymbol{x}_i - \boldsymbol{\mu}_k \right\|^2 \\ \text{s.t.} \sum_{k=1}^{K} r_{ik} = 1 \text{ for all } i = 1, \ldots, N \\ r_{ik} \in \{0, 1\} \text{ for all } i = 1, \ldots, N \text{ and } k = 1, \ldots, K \end{aligned}$$

where N is the number of data points, $\boldsymbol{x}_i$ is the i^{th} data point, K is the number of clusters, $\boldsymbol{\mu}_k$ is the mean for cluster k, and r_{ik} indicates whether data point i is assigned to cluster k or not.

b) [1 mark] Is *K*-means a *supervised* or *unsupervised* learning algorithm?

Unsupervised

c) [2 marks] Write the formula for updating mean $\boldsymbol{\mu}_k$ in the *K*-means clustering algorithm.

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_{ik} \boldsymbol{x}_i}{\sum_{i=1}^{N} r_{ik}}$$

d) [2 marks] *K*-means obviously assumes that there are *K* clusters in the data. Give exactly two other assumptions that *K*-means makes about clusters in the data.

It assumes that the clusters are *isotropic*, or ball shaped.
It assumes that the clusters have equal weights, so that the data should be evenly divided among the clusters.

Q5. [6 marks total] This question is about the *normal distribution* and *Gaussian mixture models* (GMMs).

a) [3 marks] Write the probability density $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ for a set of data point $\boldsymbol{x}_i$ under a Gaussian mixture model with K components, where $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_K\}$, $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$ and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K\}$. (It is OK to use $\mathcal{N}$ to represent the multivariate normal distribution, if used correctly.)

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

b) [1 mark] Why must we require $\pi_i \geq 0$ and $\pi_1 + \cdots + \pi_K = 1$ for a Gaussian mixture model? What what's wrong with allowing π_k to take an arbitrary value? Explain.

If we did not make this assumption, then the GMM density formula would not be a valid density function because it would not be normalized (would not sum to 1 over all possible $\boldsymbol{x}$).

(I'm guessing some students may just write "Because π has to be a probability distribution" or something, but this is not a direct explanation.)

c) [2 marks] Describe the procedure for sampling a new data point $\boldsymbol{x}$ from a Gaussian mixture model with K components. Be clear about how the model parameters are used in this procedure. (Note: this question is *not* about the EM algorithm or the formulation we used to derive the EM algorithm, so keep it simple!)

1. Sample an integer $k \in \{1, \ldots, K\}$ where integer k has probability π_k of being selected.
2. Sample a data point $\boldsymbol{x}$ from normal distribution $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Q6. [5 marks total] This question is about the *expectation maximization* (EM) algorithm.

a) [2 marks] The EM algorithm computes $N \times K$ "responsibilities" γ_{ik}. In words, what probability does a particular γ_{ik} represent? Be precise, and avoid using the word "responsibility" in your answer.

γ_{ik} represents $p(z_{ik} = 1 \mid \boldsymbol{x}_i)$, "the probability that mixture component k generated data point $\boldsymbol{x}_i$."

b) [1 mark] Is the EM algorithm an example of gradient ascent, coordinate ascent, or direct solution?

Coordinate ascent. [Writing "descent" is also acceptable since ascent/descent isn't the point.]

c) [2 marks] Let $p(x_1, \ldots, x_N \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma})$ be the likelihood for a univariate GMM where $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_K\}$, $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_K\}$ and $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_K\}$. Given the formula below

$$\frac{\partial}{\partial u_k} \left[\, \ln p(x_1, \ldots, x_N \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}) \,\right] = \sum_{i=1}^{N} \gamma_{ik} \frac{1}{\sigma_k^2} (x_i - \mu_k)$$

derive the formula by which the EM algorithm updates the μ_k parameters. Show your steps.

This formula is the partial derivative of the log likelihood with respect to μ_k.
First we set this partial derivative to zero:

$$0 = \sum_{i=1}^{N} \gamma_{ik} \frac{1}{\sigma_k^2} (x_i - \mu_k)$$

Then multiply through by σ_k^2 and rearrange to get

$$\sum_{i=1}^{N} \gamma_{ik} \mu_k = \sum_{i=1}^{N} \gamma_{ik} x_i$$

Since μ_k is independent of i we then have

$$\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{\sum_{i=1}^{N} \gamma_{ik}}$$

Q7. [4 marks total] This question is about *kernel density estimation* from a training set $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$.

a) [2 marks] Write the formula that a *kernel density estimate* (or *Parzen estimate*) uses to compute $p(\boldsymbol{x})$ for a new point $\boldsymbol{x}$.

The kernel density estimate for $p(\boldsymbol{x})$ given training set $\mathcal{D}$ is

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} k(\boldsymbol{x} - \boldsymbol{x}_i)$$

where $k(\cdot)$ is the kernel.

b) [1 mark] Why might a "tophat" kernel allow us to compute $p(\boldsymbol{x})$ faster than a Gaussian kernel?

Because a tophat kernel has limited support and so only a small fraction of the $\boldsymbol{x}_i$ need to contribute to $p(\boldsymbol{x})$. We can use data structures for efficient D-dimensional search that allow us to ignore most of the $\boldsymbol{x}_i$ when evaluating $p(\boldsymbol{x})$.

c) [1 mark] For kernel density estimation, the basic 1-dimensional Gaussian kernel is

$$k(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$$

Show that when we construct a new kernel $k_h(x)$ by "adding bandwidth h" we get $k_h(x) = \mathcal{N}(x \mid 0, h^2)$.

In one dimension, we add bandwidth using the formula $k_h(x) = \frac{1}{h}k(\frac{x}{h})$. This gives

$$k_h(x) = \frac{1}{\sqrt{2\pi}h} \exp\left(-\frac{1}{2}\frac{x^2}{h^2}\right)$$

which has exactly the form of $\mathcal{N}(x \mid 0, h^2)$.

Q8. [12 marks total] This question is about *support vector machines* (SVMs).

a) [2 marks] What is the learning principle used by SVMs? What is an SVM trying to do?

SVMs try to *maximize the margin* between the decision boundary and the nearest training points of each class.

b) [1 mark] For a hard-margin SVM, which training points become the "support vectors" after training is complete?

The support vectors are the training points for which no other training point is strictly 'closer' to the margin in feature space.

c) [5 marks] Given a training set $\mathcal{D} = \{(\boldsymbol{x}_i, t_i)\}_{i=1}^N$ where $t_i \in \{-1, +1\}$, write the primal formulation of the soft-margin *linear* SVM learning problem on this data. Give a name to each new symbol you use (does't need to be too precise). If you cannot remember the soft-margin formulation, write hard-margin for up to 4 marks.

The soft-margin linear SVM learning problem has the form

$$\begin{aligned} \min_{\boldsymbol{w}, b} \ & \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \ & 1 - \xi_i \leq \boldsymbol{w}^T \boldsymbol{x}_i + b \quad \text{for } i = 1, \ldots, N \\ & \xi_i \geq 0 \end{aligned}$$

where $\boldsymbol{w}$ are the weights, b is the bias, ξ_i are the slack variables that penalize violation of the margin constraint, and C is the regularization coefficient (strength of slack penalty).

d) [2 marks] Suppose we want to perform multi-class classification with K classes. Describe the "one versus rest" (OvR) strategy, and state how many SVMs we would have to train.

The OvR strategy asks us to train K separate SVMs, where the k^{th} SVM is trained to classify data from class k against all data from the $K - 1$ other classes.

e) [2 marks**] The dual formulation of hard-margin SVM is defined in terms of dual variables $\boldsymbol{\alpha} \in \mathbb{R}^N_{\geq 0}$. Write the formula for the SVM decision function $y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b$ in terms of the α_i, training labels t_i, training samples $\boldsymbol{x}_i$, kernel function $k(\cdot, \cdot)$, and the bias term b which you can assume is known. Show your steps.

For any choice of $\boldsymbol{\alpha}$ the optimal corresponding $\boldsymbol{w}$ is

$$\boldsymbol{w}^* = \sum_{i \in \mathcal{S}} \alpha_i t_i \boldsymbol{\phi}(\boldsymbol{x}_i)$$

where $\mathcal{S}$ is the set of support vectors (since $\alpha_i = 0$ for all $i \notin \mathcal{S}$). Substituting, we then have

$$\begin{aligned} y(\boldsymbol{x}) &= \left(\sum_{i \in \mathcal{S}} \alpha_i t_i \boldsymbol{\phi}(\boldsymbol{x}_i) \right)^T \boldsymbol{\phi}(\boldsymbol{x}) + b \\ &= b + \sum_{i \in \mathcal{S}} \alpha_i t_i \boldsymbol{\phi}(\boldsymbol{x}_i)^T \boldsymbol{\phi}(\boldsymbol{x}) \\ &= b + \sum_{i \in \mathcal{S}} \alpha_i t_i k(\boldsymbol{x}_i, \boldsymbol{x}) \end{aligned}$$

which is our final expression for the decision function.

Q9. [10 marks total] This question is about *decision trees*, *random forests*, and *boosting*.

a) [3 marks] When building a decision tree for a classification problem with class labels $y \in \{1, \ldots, K\}$, what are the two measures of "impurity" we discussed in class? Two marks for naming them both, one mark for giving a correct formula for at least one.

Entropy impurity:

$$I_{\text{entropy}} = -\sum_{k=1}^{K} p(y=k) \log_2 p(y=k)$$

Gini impurity:

$$I_{\text{gini}} = \sum_{k \neq k} p(y=k) p(y=k') = \frac{1}{2}\left(1 - \sum_{k=1}^{K} p(y=k)^2\right)$$

b) [2 marks] Consider the training set below. (It may help to plot it.)

$$\boldsymbol{X} = \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}, \quad \boldsymbol{t} = \begin{bmatrix} -1 \\ +1 \\ -1 \\ +1 \end{bmatrix}$$

Can a decision tree classifier achieve zero training error on this data set? If yes, draw a tree that achieves zero training error, including split thresholds and leaf class labels. If not, explain what limitation prevents it.

If student answers "Yes" they must give a balanced tree of height 2 (3 internal nodes, 4 leaf nodes). Here is a pre-order traversal of one such tree where we go "left" if the test is true:

1: $[x_2 \leq 0]$ 2: $[x_1 \leq 0]$ 3: $[\text{negative}]$ 4: $[\text{positive}]$ 5: $[x_1 \leq 0]$ 6: $[\text{positive}]$ 7: $[\text{negative}]$

If the student answers "No" they must justify by stating that the impurity (entropy or gini) of any possible split is identical to the impurity of the entire data set; therefore, even though a tree exists, a greedy algorithm may not find it because it would not make the initial split required. (That being said, scikit-learn tries to make at least one split regardless, and therefore succeeds in finding a tree.)

c) [1 mark] Compared to training a single decision tree on all the training data, does bagging tend to reduce the *training error*, the *testing error*, or *both*?

Bagging tends to improve the testing error, but not the training error. The training error of the bagged ensemble will be, on average, worse compared to an individual decision tree.

d) [2 marks] AdaBoost and gradient boosting both search for a combination of weights $\{\alpha_1, \ldots, \alpha_R\}$ and weak models $\{y_1, \ldots, y_R\}$ that combine to give a strong model $y(\boldsymbol{x})$. In class, we reviewed AdaBoost for 2-class classification, and gradient boosting for regression. What is the formula for the strong predictor $y(\boldsymbol{x})$ when we are boosting 2-class classifiers?

$$y(\boldsymbol{x}) = \text{sign}\left(\sum_{r=1}^{R} \alpha_r y_r(\boldsymbol{x})\right)$$

e) [2 marks**] Suppose we use a decision stump classifier (a tree with only one split) as a weak learner. Can boosting achieve zero training error on the data set from (b)? If yes, list the decision stump thresholds and weights that achieve zero training error. If not, explain what limitation prevents it.

> No, boosted decision stumps cannot. *Proof sketch:* Consider the two positive examples, call them $\mathbf{x}_2$ and $\mathbf{x}_4$. For this data set, any decision stump $\alpha_r y_r(\mathbf{x})$ must either split $\mathbf{x}_2$ and $\mathbf{x}_4$ or not split any points at all. Any stump that splits $\mathbf{x}_2$ and $\mathbf{x}_4$ must contribute $+\alpha_r$ to the decision function one point (e.g., $\mathbf{x_2}$) and $-\alpha_r$ to the decision funciton at other point (e.g., $\mathbf{x_4}$). Therefore, assuming all stumps split the data, the decision function $f(\mathbf{x}) = \sum_r \alpha_r y_r(\mathbf{x})$ must symmetric so that $f(\mathbf{x}_2) = -f(\mathbf{x_4})$. Adding stumps that do not split the data simply biases the decision function at all training points equally, and does not provide any additional discriminative power, so these can be ignored.
> **[Student answers do not need to be so detailed; half-reasonable argument gets full marks!!]**

Q10. [4 marks total] This question is about the assigned reading: the 2001 paper by Leo Breiman.

a) [2 marks] How are what Breiman calls "data models" typically validated? How are what he calls "algorithmic models" typically validated?

> *Data models* are typically validated by measuring "goodness-of-fit" and analyzing residuals. A model is either accepted or rejected based on that.
>
> *Algorithmic models* are typically validated by measuring their predictive accuracy, ideally on a held-out test set.

b) [1 mark] What learning algorithm does Breiman rate as "A+ for prediction" but "F for interpretability"?

> Random forests

c) [1 mark] Describe the main symptom of what Breiman calls 'model instability'.

> Model instability means that there are many possible models that have similar training or test error. Slight perturbation of the data or the training algorithm will cause a large change in the final model.

Q11. [6 marks total] This question is about programming machine learning concepts with Numpy. You can assume that all the required imports (numpy, sklearn, etc) have been run.

a) [2 marks] You are given the following incomplete function:

```
def linear_regression_by_gradient_descent(X, y, w_init, learn_rate=0.05, num_steps=500):
    """
    Fits a linear model by gradient descent.
    Given X, y, and w_init with shapes (N,D), (N,), and (D,) respectively,
    returns a new parameter vector w that minimizes the squared error to the targets
    by running num_steps of gradient descent.
    """
```

Complete the function in the space below. Call the `linear_least_squares_grad` function from Q2 (d).

```
        w = w_init
        for i in range(num_steps):
            grad = linear_least_squares_grad(X, y, w)
            w = w - learn_rate*grad
        return w
```

b) [4 marks] You are given code to load `data.csv` , a binary classification data set. Write code to normalize the features, train a *DecisionTreeClassifier* with maximum depth of 3, and then print the training accuracy. (Do not worry about minor syntax issues, but your code should be recognizable as Python.)

```
data = np.loadtxt('data.csv', delimiter=',', skiprows=1)
X = data[:,:2]
y = data[:,2].astype(np.int32)*2-1

# Your code below here. Aim for 4-7 lines.
```

```
        X_scaled = sklearn.preprocessing.StandardScaler().fit_transform(X)

        tree = sklearn.tree.DecisionTreeClassifier(max_depth=3)
        tree.fit(X_scaled, y)

        accuracy = tree.score(X_scaled, y)
        # or     = sklearn.metrics.accuracy_score(y, tree.predict(X_scaled))

        print(accuracy)
```