Version A

1. **[1pt]** *Give one reason why an algorithm implemented in terms of matrix and vector operations can be faster than the same algorithm implemented in terms of* `for`*-loops.*

   Possible answers include:

   - The operations can be implemented in a low-level language like C or Fortran, thereby eliminating the Python interpreter overhead of `for`-loops.
   - The linear algebra libraries are carefully engineered (e.g. for cache efficiency).
   - Matrix operations can be parallelized on a GPU architecture.

   Mean: 0.84/1

2. **[2pts]** *Briefly explain two advantages of decision trees over K-nearest-neighbors.*

   Possible answers include:

   - good when there are lots of attributes, but only a few are important
   - good with discrete attributes
   - easily deal with missing values (just treat as another value)
   - robust to scale of inputs
   - fast at test time
   - more interpretable

   Mean: 1.62/2

3. **[1pt]** *TRUE or FALSE: AdaBoost will eventually choose a weak classifier that achieves a weighted error rate of 0. Briefly justify your answer.*

   FALSE. If the weak classifier achieves a weighted error rate of 0, that means it classifies all the training examples correctly. This only happens if we're very lucky, since then it means this weak classifier would be chosen in the first iteration, and the algorithm would immediately achieve perfect training accuracy.

   Mean: 0.40/1

4. **[1pt]** *In class, we considered using the squared Euclidean norm of the weights, $\|\mathbf{w}\|^2$, as a regularizer. Suppose we instead used the Euclidean norm $\|\mathbf{w}\|$ as a regularizer*

(i.e. without squaring it). Briefly explain one way in which this would lead to different behavior.

Possible answers include:

- it's more willing to allow the weights to get very large
- it tries harder to set $\mathbf{w}$ to exactly 0

Mean: 0.46/1

5. **[2pts]** *Recall that hyperparameters are often tuned using a validation set.*

   (a) **[1pt]** *Give an example of a hyperparameter which it is OK to tune on the training set (rather than a validation set). Briefly explain your answer.*

   Hyperparameters relating to optimization can be tuned on the training set, since they determine the rate of convergence rather than the amount of overfitting. Possible answers include the learning rate, momentum decay factor, or batch size.

   Marking: Half credit for saying that no hyperparameters can be tuned on the training set, and giving a reasonable justification. No credit for giving an incorrect example (e.g. a regularization hyperparameter).

   (b) **[1pt]** *Give an example of a hyperparameter which should be tuned on a validation set, rather than the training set. Briefly explain your answer.*

   Hyperparameters that control overfitting should be tuned on the validation set, since tuning them on the training set would always choose the most complex model, the weakest regularizer, etc. Examples include K in KNN, the number of steps of boosting, the depth of a decision tree, the L_2 regularization weight, the slack penalty for a soft-margin SVM, the number of units or layers in a neural net, etc.

   Marking: Most answers were either right or wrong for this one.

   Mean: 1.39/2

6. **[2pts]** *In this question, you will write NumPy code to implement a 1-nearest-neighbour classifier. Assume you are given an $N \times D$ data matrix `X`, where each row corresponds to one of the input vectors, and an integer array `y` with the corresponding labels. (You may assume the labels are integers from 1 to K.) You are given a query vector `x_query`. Your job is to return the predicted class (as an integer). Do not use a `for`-loop.*

   If you don't remember the API for a NumPy operation, then for partial credit, explain what you are trying to do.

```
dist = np.sum((X - x_query)**2, axis=1)
return y[np.argmin(dist)]
```

Marking: 1.5 points for the distance computation, 0.5 for the rest. We were very generous about the APIs, e.g. we gave full credit for things like `dist.smallest()` as long as it was obvious what you were trying to do.

Mean: 1.11/2

7. [**2pts**] *Consider the classification problem with the following dataset:*

x_1	x_2	x_3	t
0	0	0	1
0	1	0	0
0	1	1	1
1	1	1	0

Your job is to find a linear classifier with weights w_1, w_2, w_3, and b which correctly classifies all of these training examples. None of the examples should lie on the decision boundary.

(a) [**1pt**] *Give the set of linear inequalities the weights and bias must satisfy.*

$$\begin{aligned} b &> 0 \\ w_2 + b &< 0 \\ w_2 + w_3 + b &> 0 \\ w_1 + w_2 + w_3 + b &< 0 \end{aligned}$$

(b) [**1pt**] *Give a setting of the weights and bias that correctly classifies all the training examples. You don't need to show your work, but it might help you get partial credit.*

Marking: Half a point off for the inequalities not being strict (we asked for the examples not to lie on the decision boundary).

Many answers are possible. Here's one:

$$\begin{aligned} b &= 1 \\ w_1 &= -2 \\ w_2 &= -2 \\ w_3 &= 2 \end{aligned}$$

Marking: You should have received full credit if your answer is consistent with part (a). Half credit can be given if the work is shown.

Mean: 1.41/2

8. **[2pts]** *Recall that in bagging, we compute an average of the predictions* $y_{\text{avg}} = \frac{1}{m}\sum_{i=1}^{m} y_i$*, where* y_i *denotes the prediction made by the model trained on the* i*th bootstrapped dataset. Recall that these predictions are not fully independent, i.e. they are correlated because their training sets come from the same underlying dataset. Suppose* $\text{Var}[y_i] = \sigma^2$ *and the correlation between* y_i *and* y_j *is* ρ *for* $i \neq j$*. Give the formula for the variance* $\text{Var}[y_{\text{avg}}]$*. (If you don't remember the formula from lecture, you should be able to figure it out from the properties of covariance. In either case, you should justify where the formula comes from.)*

$$\begin{aligned}
\text{Var}(y_{\text{avg}}) &= \text{Var}\left(\frac{1}{m}\sum_{i=1}^{m} y_i\right) \\
&= \frac{1}{m^2}\text{Var}\left(\sum_{i=1}^{m} y_i\right) \\
&= \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\text{Cov}(y_i, y_j) \\
&= \frac{1}{m^2}\left[m\sigma^2 + m(m-1)\rho\sigma^2\right] \\
&= \frac{1}{m}\sigma^2 + \frac{m-1}{m}\rho\sigma^2
\end{aligned}$$

Marking: Half a point for each of the following:

(a) making the substitution given for y_{avg}

(b) moving $1/m$ outside the variance to $1/m^2$

(c) applying a correct covariance formula

(d) arriving at the final correct expression

In the case of a correct answer with an incorrect or incomplete derivation, 1 point was given for the answer, plus half a point for each of the above.

Mean: 0.89/2

9. **[2pts]** *Consider a regression problem where the input is a scalar x. Suppose we know that the dataset is generated by the following process. First, the target t is chosen from $\{0, 1\}$ with equal probability. If $t = 0$, then x is sampled from a uniform distribution over the interval $[1, 2]$. If $t = 1$, then x is sampled from a uniform distribution over the interval $[0, 2]$. Give a function $f(x)$, defined for $x \in [0, 2]$, such that $y_* = f(x)$ is the Bayes optimal predictor for t given x. (Note that even though t is binary valued, this is a regression problem, with squared error loss.)*

   Our job is to compute $f(x) = \mathbb{E}[t|x]$, the formula for the Bayes optimal predictor.

   Note that $p(x|t = 1) = 1/2$ on $[0, 2]$ and 0 elsewhere, and $p(x|t = 0) = 1$ on $[1, 2]$ and 0 elsewhere. By Bayes' Rule, for $x \in [0, 1]$,

$$\begin{aligned} p(t = 1|x) &= \frac{p(t = 1)p(x|t = 1)}{p(t = 0)p(x|t = 0) + p(t = 1)p(x|t = 1)} \\ &= \frac{\frac{1}{2} \cdot \frac{1}{2}}{\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot \frac{1}{2}} \\ &= 1 \end{aligned}$$

   And for $x \in [1, 2]$,

$$\begin{aligned} p(t = 1|x) &= \frac{p(t = 1)p(x|t = 1)}{p(t = 0)p(x|t = 0) + p(t = 1)p(x|t = 1)} \\ &= \frac{\frac{1}{2} \cdot \frac{1}{2}}{\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2}} \\ &= \frac{1}{3} \end{aligned}$$

   Hence,

$$f(x) = \begin{cases} 1 & \text{if } 0 \le x \le 1 \\ \frac{1}{3} & \text{if } 1 < x \le 2 \end{cases}.$$

Marking: One point for writing down $y_* = \mathbb{E}[t|x]$ or showing understanding that this is the right thing to compute. Half a point for the correct expectation in each of the two intervals.

Mean: 0.45/2

Version B

1. *As discussed in lecture, when applying K-nearest-neighbors, it is common to normalize each input dimension to unit variance.*

   (a) **[1pt]** *Why might it be advantageous to do this?*

   If one feature has higher variance than the others, it will be treated as more important. This is problematic if the features have arbitrary scales, since the importance of a feature would depend on the unit used. Normalizing to unit variance fixes this problem.

   (b) **[1pt]** *When might this normalization step not be a good idea? (Hint: You may want to consider the task of classifying images of handwritten digits, where the digit is centered within the image.)*

   The fact that a feature has higher variance might indicate it is actually more important. For instance, in MNIST classification, pixels near the boundary are almost always off, and hence not very informative for the decision. Scaling them up to unit variance would just add noise to the classifications.

   Mean: 0.96/2

2. **[1pt]** *In random forests, what is the motivation for randomizing the set of attributes considered for each split?*

   The aim is to reduce the correlation between the predictions of different classifiers in the ensemble, thereby reducing the variance of the average prediction.

   Mean: 0.65/1

3. **[1pt]** *Suppose you want to evaluate the test error rate of a 1-nearest-neighbors classifier. Assume you implement the algorithm the naïve way, i.e. by explicitly computing all the distances and taking the min, rather than by using a fancy data structure. What is the running time of evaluating the test error? Give your answer in big-O notation, in terms of the number of training examples N_{train}, the number of test examples N_{test}, and the input dimension D. Briefly explain your answer.*

   For each test example, we need to compute the distances to each of the N_{train} training examples. Each distance computation is $\mathcal{O}(D)$. This needs to be done for all N_{test} examples, so the overall cost is $\mathcal{O}(N_{\text{train}} N_{\text{test}} D)$.

   Mean: 0.78/2

4. (a) **[1pt]** *Give one advantage of K-nearest-neighbors over linear regression.*
   Possible answers include:
   - able to fit nonlinear functions
   - no cost at training time
   - able to interpret by identifying which training examples led to the prediction
   - able to approximate any function arbitrarily well given enough data

   (b) **[1pt]** *Give one advantage of linear regression over K-nearest-neighbors.*
   Possible answers include:
   - faster at test time
   - don't need to store the whole training set to make predictions
   - able to interpret by inspecting the weights for different features
   - less likely to overfit (compared to 1-NN)

   Mean: 1.65/2

5. **[1pt]** *Suppose linear regression (with squared error loss) is used as a classification algorithm. TRUE or FALSE: if it correctly classifies every training example, then its cost is zero. (By "cost", we mean the function minimized during training.) Briefly justify your answer.*

   FALSE. In order to have a loss of zero, it must predict $z^{(i)} \in \{-1, 1\}$ for every training example. The fact that every example is correct only implies that $z^{(i)}$ is on the correct side of the threshold.

   Mean: 0.55/1

6. **[2pts]** *Let Z be a random variable and t be a real number. Show that*

$$\mathbb{E}[(Z-t)^2] = (\mathbb{E}[Z]-t)^2 + \mathrm{Var}[Z].$$

   (This is a simplified verison of the bias-variance decomposition.)

$$\begin{aligned}\mathbb{E}[(Z-t)^2] &= \mathbb{E}[Z^2 - 2Zt + t^2] \\ &= \mathbb{E}[Z^2] - 2\mathbb{E}[Z]t + t^2 \\ &= \mathrm{Var}(Z) + \mathbb{E}[Z]^2 - 2\mathbb{E}[Z]t + t^2 \\ &= \mathrm{Var}(Z) + (\mathbb{E}[Z]-t)^2.\end{aligned}$$

   Mean: 1.97/2

7. **[2pts]** *Suppose binary-valued random variables X and Y have the following joint distribution:*

	$Y = 0$	$Y = 1$
$X = 0$	$1/8$	$3/8$
$X = 1$	$2/8$	$2/8$

Determine the information gain $IG(Y|X)$. You may write your answer as a sum of logarithms.

$$\begin{aligned}
H(Y) &= -\tfrac{3}{8}\log\tfrac{3}{8} - \tfrac{5}{8}\log\tfrac{5}{8} \\
H(Y|X) &= \tfrac{1}{2}\left(-\tfrac{1}{4}\log\tfrac{1}{4} - \tfrac{3}{4}\log\tfrac{3}{4}\right) + \tfrac{1}{2}\left(-\tfrac{1}{2}\log\tfrac{1}{2} - \tfrac{1}{2}\log\tfrac{1}{2}\right) \\
&= \tfrac{1}{2}\left(\tfrac{1}{2} - \tfrac{3}{4}\log\tfrac{3}{4}\right) + \tfrac{1}{2} \\
&= \tfrac{3}{4} - \tfrac{3}{8}\log\tfrac{3}{4} \\
IG(Y|X) &= H(Y) - H(Y|X) \\
&= -\tfrac{3}{8}\log\tfrac{3}{8} - \tfrac{5}{8}\log\tfrac{5}{8} - \tfrac{3}{4} + \tfrac{3}{8}\log\tfrac{3}{4}
\end{aligned}$$

Marking: Half a point off if the sign of the solution is wrong. One point assigned to each of the two parts of the information gain formula. Full credit if the answer is correct; half credit if it's wrong but reasonable work is shown. We didn't expect much in terms of simplifying the expressions.

Mean: 1.31/2

8. **[2pts]** *Recall that combining the logistic activation function with squared error loss suffers from saturation, whereby the gradient signal is very small when the prediction for a training example is very wrong. Logistic regression (i.e. logistic activation function with cross-entropy loss) doesn't have this problem. Recall that the logistic function is defined as $\sigma(z) = 1/(1+e^{-z})$. Now suppose we modify the activation function to squash the prediction y to be in the interval $[0.1, 0.9]$, and then apply cross-entropy loss. I.e.,*

$$\begin{aligned}
z &= \mathbf{w}^\top \mathbf{x} + b \\
y &= 0.8\sigma(z) + 0.1 \\
\mathcal{L}(y, t) &= -t\log y - (1-t)\log(1-y),
\end{aligned}$$

where σ is the logistic activation function. Does this model have a problem with saturation? You don't need to give a formal proof, but you should informally justify your *answer. Hint: it is possible to answer this question without calculating derivatives. Think qualitatively.*

Yes, it does suffer from saturation. For a positive training example ($t = 1$), as $z \to -\infty$, $y \to 0.1$, so the loss approaches $-\log 0.1$. Since the loss asymptotes to a particular value, this means $\mathrm{d}\mathcal{L}/\mathrm{d}z \to 0$.

Mean: 0.65/2

9. **[2pts]** *Recall that the soft-margin SVM can be viewed as minimizing the hinge loss with an L_2 regularization term. I.e.,*

$$\begin{aligned} z &= \mathbf{w}^\top \mathbf{x} + b \\ \mathcal{L}(z, t) &= \max(0, 1 - tz) \\ \mathcal{J}(\mathbf{w}, b) &= \tfrac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z^{(i)}, t^{(i)}). \end{aligned}$$

Here, $t \in \{-1, +1\}$. Complete the formulas for the gradient calculations. You don't need to show your work.

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial \mathbf{w}} &= \lambda \mathbf{w} + \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{w}} \\ \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} &= \begin{cases} -t & \text{if } 1 - tz > 0 \\ 0 & \text{otherwise} \end{cases} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\mathbf{x} \end{aligned}$$

Marking: The three parts were worth 0.5, 1, and 0.5, respectively.

Mean: 1.19/2