Bottom right: the laplacian is never invertible; **1** is always in the nullspace.

**(v)** [3 pts] For binary classification, which of the following statements are true of AdaBoost?

- ● It can be applied to neural networks
- ○ It uses the majority vote of learners to predict the class of a data point
- ● The metalearner provides not just a classification, but also an estimate of the posterior probability
- ● The paper on AdaBoost won a Gödel Prize

'PSQBSUX

Top left: When we use a single decision tree, we usually make it very deep. If there are no two coincident points in different classes and we always refine until the leaves are pure, the bias is zero. It's hard to get lower than zero bias.

Top right: Decision trees don't use a loss function. They do choose splits locally with a cost function based on entropy, but you don't use w_i to scale the entropy. Rather, you redefine p_C from "the proportion of points in S that are in class C" to "the proportion of weight in S that is in class C" before you compute the entropy.

**(w)** [3 pts] For binary classification, which of the following statements are true of AdaBoost with decision trees?

- ● It usually has lower bias than a single decision tree
- ● It is popular because it usually works well even before any hyperparameter tuning
- ○ To use the weight $w_i$ of a sample point $X_i$ when training a decision tree $G$, we scale the loss function $L(G(X_i), y_i)$ by $w_i$
- ○ It can train multiple decision trees in parallel

**(x)** [3 pts] Which of the following are reasons one might choose latent factor analysis (LFA) over $k$-means clustering to group together $n$ data points in $\mathbb{R}^d$?

- ● LFA is not sensitive to how you initialize it, whereas Lloyd's algorithm is
- ● LFA allows us to consider points as belonging to multiple "overlapping" clusters, whereas in $k$-means, each point belongs to only one cluster
- ○ In market research, LFA can distinguish different consumer types, whereas $k$-means cannot
- ● $k$-means requires you to guess $k$ in advance, whereas LFA makes it easier to infer the right number of clusters after the computation

The first choice is true due to the curse of dimensionality. The second one is false: LFA is more expensive than k-means. For $I$ iterations and $k$ clusters, k-means runs in $O(nkID)$ time, whereas SVD takes $O(\min(dn^2, d^2n))$ time. The third one is true. We can measure how much a user vector belongs to a particular cluster by taking its inner product with the corresponding singular vector. The fourth one is false because of the above application.

**(y)** [3 pts] Which of the following are true for $k$-nearest neighbor classification?

- ● It is more likely to overfit with $k = 1$ (1-NN) than with $k = 1{,}000$ (1,000-NN)
- ● In very high dimensions, exhaustively checking every training point is often faster than any widely used competing exact $k$-NN query algorithm
- ● If you have enough training points drawn from the same distribution as the test points, $k$-NN can achieve accuracy almost as good as the Bayes decision rule
- ○ The optimal running time to classify a point with $k$-NN grows linearly with $k$

Top left: correct; smaller $k$'s overfit more.

Bottom left: empirical fact.

Top right: true; Fix & Hodges, 1951.

Bottom right: false, it's poly.

**(z)** [3 pts] Suppose we use the $k$-d tree construction and query algorithms described in class to find the *approximate* nearest neighbor of a query point among $n$ sample points. Select the true statements.

- ● It is possible to guarantee that the tree has $O(\log n)$ depth by our choice of splitting rule at each treenode
- ○ Sometimes we permit the $k$-d tree to be unbalanced so we can choose splits with better information gain
- ○ Querying the $k$-d tree is faster than querying a Voronoi diagram for sample points in $\mathbb{R}^2$
- ● Sometimes the query algorithm declines to search inside a box that's closer to the query point than the nearest neighbor it's found so far

Bottom Left: there is no entropy or sense of info gain in the k-d algorithm

Bottom Right: if your approximate nearest neighbor is good enough than you won't search for a better neighbor

It is hard to beat O(log n) query time. Too bad Voronoi diagrams are only guaranteed to be that fast in 2D, and they can only be used for 1-NN. Nevertheless, if you need to do 1-NN on a large 2D dataset, and you will be doing a lot of queries once you've built the search structure, using a Voronoi diagram is actually really fast and effective. Just don't try to write the code yourself. Get a library written by an expert.

# Q2. [17 pts] Getting Down(hill) with the Funk Function

The Netflix Prize was an open competition for the best *collaborative filtering* algorithm to predict user ratings for films. Competitors were given an $n \times d$ *ratings matrix* $R$; entry $R_{jk}$ is user $j$'s rating of movie $k$. Because users only watch a small fraction of the movies, most entries in $R$ are unobserved, hence filled with a default value of zero. Latent factor analysis attempts to predict missing ratings by replacing $R$ with a low-rank approximation, which is a truncated singular value decomposition (SVD).

**(a)** [4 pts] Given the SVD $R = UDV^\top$, write a formula for the rank-$r$ truncated SVD $R'$ for comparison; make sure you explain your notation. Then write the standard restrictions (imposed by the definition of SVD) on $U$, $D$, and $V$.

The rank-$r$ truncated SVD of $R$ is $R' = \sum_{i=1}^{r} \delta_i u_i v_i^\top$, where $u_i$ is column $i$ of $U$ and $v_i$ is column $i$ of $V$. The standard restrictions are $U^\top U = I$, $V^\top V = I$, and $D$ is a diagonal matrix with nonnegative components.

LFA leaves plenty of room for improvement. Simon Funk (a pseudonym, but a real person), who at one point was ranked third in the competition, developed a method called "Funk SVD." Recall that the rank-$r$ truncated SVD $R'$ minimizes the Frobenius norm $\|R - R'\|_F$, subject to the constraint that $R'$ has rank $r$. Mr. Funk modified this approach to learn two matrices $A \in \mathbb{R}^{n \times r}$ and $B \in \mathbb{R}^{r \times d}$ such that $AB \approx R$. The rank of $AB$ cannot exceed $r$. Let $a_j$ be the $j$th row of $A$, let $b_k$ be the $k$th column of $B$, and observe that $(AB)_{jk} = a_j \cdot b_k$. Mr. Funk solves the problem of finding matrices $A$ and $B$ that minimize the objective function

$$L(A, B) = \sum_{j,k:\, R_{jk} \neq 0} (R_{jk} - a_j \cdot b_k)^2.$$

The key difference between this objective function and the one optimized by the truncated SVD is that the summation is over **only nonzero** components of $R$. Instead of computing an SVD, Mr. Funk minimizes this objective with gradient descent.

**(b)** [2 pts] Explain why the optimal solution is not unique; that is, there is more than one pair of optimal matrices $(A, B)$.

If $AB = R$, then $(2A)(\frac{1}{2}B) = R$ too.

**(c)** [5 pts] Although Mr. Funk uses stochastic gradient descent, we will derive a batch gradient descent algorithm. It turns out to be easiest to write the update rule for $A$ one row at a time. State the gradient descent rule for updating row $a_j$ during the minimization of Mr. Funk's objective function $L(A, B)$. Use some step size $\epsilon > 0$. (Be careful that you sum only the correct terms!) (Note: there is a symmetric rule for updating $b_k$; the algorithm must update both $A$ and $B$.)

$$\nabla_{a_j} L(A, B) = -2 \sum_{k:R_{jk} \neq 0} (R_{jk} - a_j \cdot b_k) b_k.$$

Hence the gradient descent update is

$$a_j \leftarrow a_j + 2\epsilon \sum_{k:R_{jk} \neq 0} (R_{jk} - a_j \cdot b_k) b_k.$$

(You may omit the factor of 2.)

**(d)** [3 pts] What will happen if you initialize Funk SVD by setting $A \leftarrow 0$ and $B \leftarrow 0$? Suggest a better initialization.

If the matrices are initialized to zero, the gradient descent rule cannot make them nonzero.

There are many better initializations. You could use $A \leftarrow UD$ and $B \leftarrow V^\top$, or better yet, $A \leftarrow UD^{1/2}$ and $B \leftarrow D^{1/2}V^\top$. A random initialization will generally work okay.

Note that a choice in which all the components of $a_j$ are the same and all the components of $b_k$ are the same will not work.

**(e)** [3 pts] Consider the special case where $r = 1$ and the matrix $R$ has no zero entries. In this case, what is the relationship between an optimal solution $A, B$ and the rank-one truncated singular value decomposition?

$AB = \delta_1 u_1 v_1^\top$.

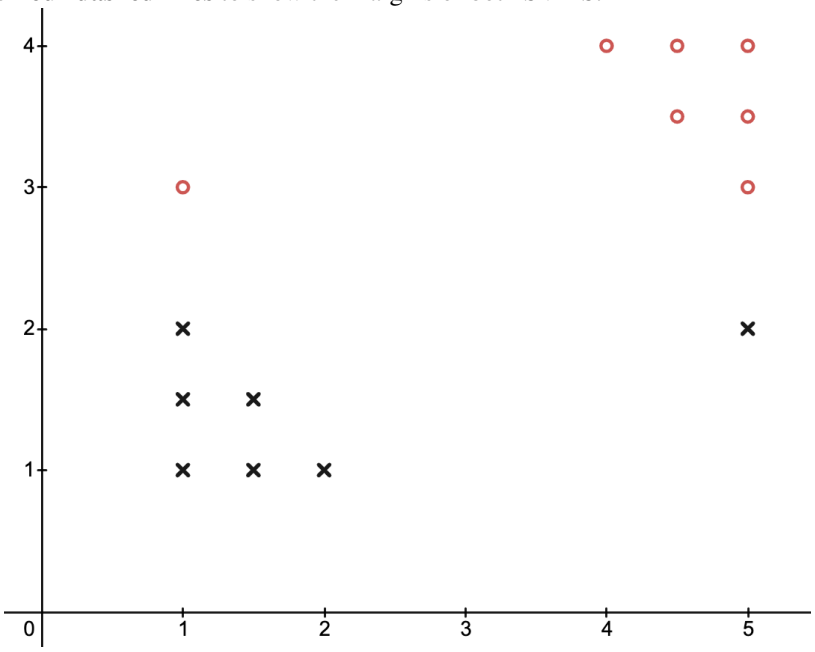
(Because the rank-1 truncated SVD is $\delta_1 u_1 v_1^\top$, and it minimizes the Frobenius norm reconstruction error. This is equivalent to Mr. Funk's objective when $R$ has no zeros.)

# Q3. [10 pts] Decision Boundaries

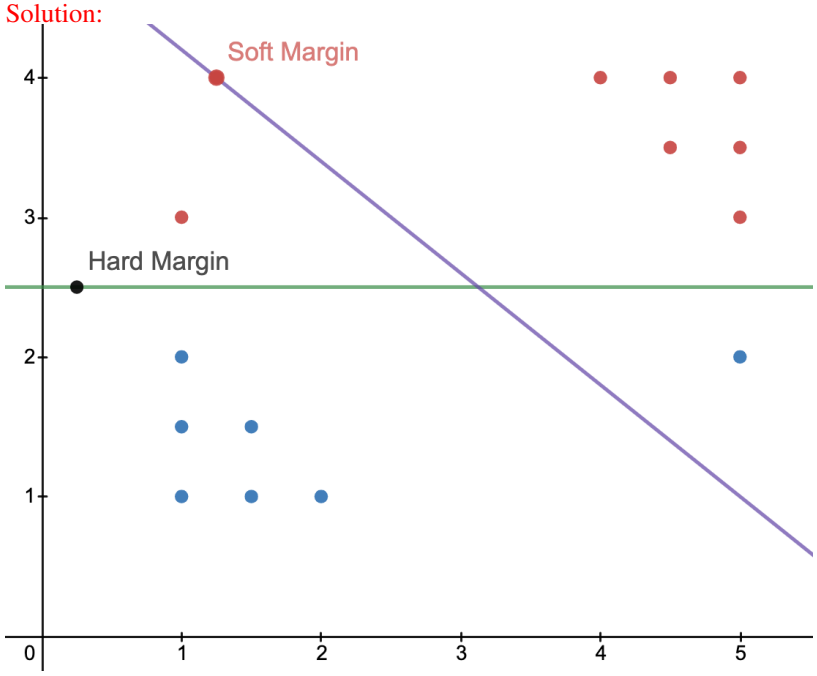
In the question, you will draw the decision boundaries that classifiers would learn.

**(a)** [6 pts] Given the sample points below, draw and label **two lines**: the decision boundary learned by a hard-margin SVM and the decision boundary learned by a soft-margin SVM. We are not specifying the hyperparameter $C$, but don't make $C$ too extreme. (We are looking for a qualitative difference between hard- and soft-margin SVMs.) **Label the two lines clearly.**

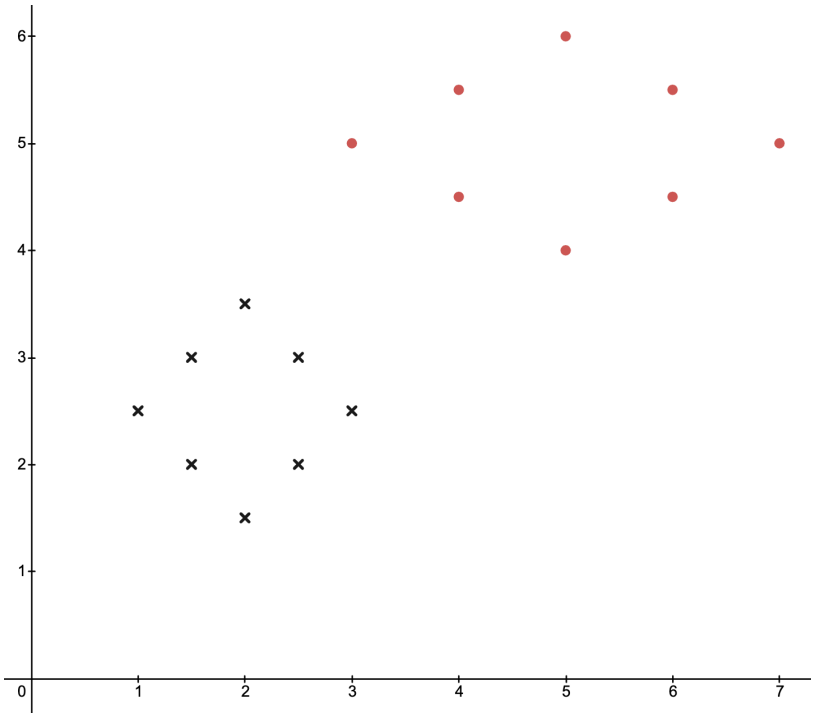
Also draw and label **four dashed lines** to show the margins of both SVMS.



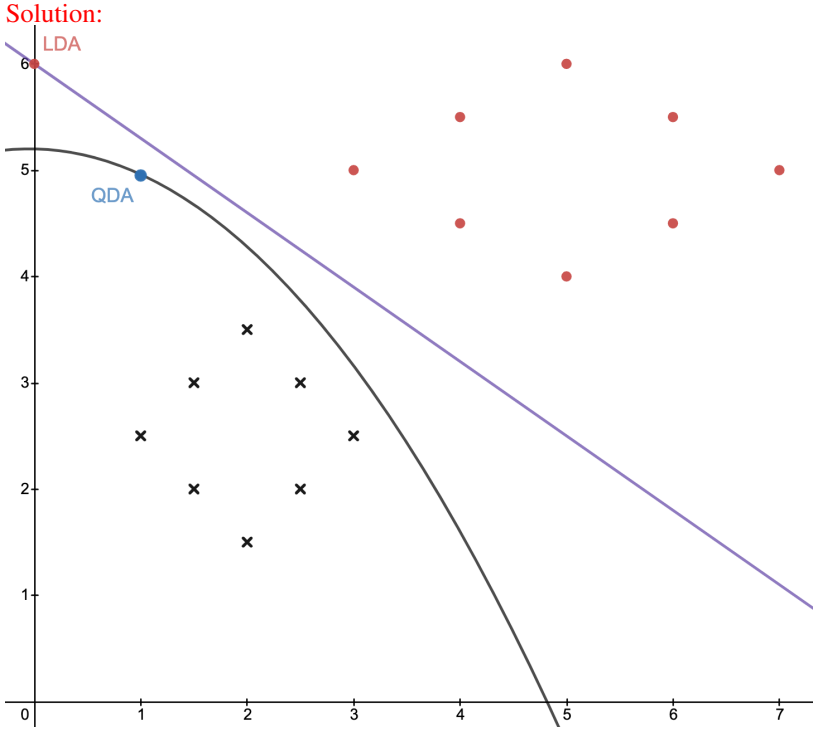

Solution:




**(b)** [4 pts] Given the sample points below, draw and label **two curves**: the decision boundary learned by LDA and the decision boundary learned by QDA. Label the two curves clearly.




Solution:



# Q4. [16 pts] Kernel Principal Components Analysis

Let $X$ be an $n \times d$ design matrix. Suppose that $X$ has been centered, so the sample points in $X$ have mean zero. In this problem we consider kernel PCA and show that it equates to solving a *generalized Rayleigh quotient problem*.

**(a)** [1 pt] Fill in the blank: every principal component direction for $X$ is an eigenvector of _____. $X^\top X$

**(b)** [1 pt] Fill in the blank: an optimization problem can be kernelized only if its solution $w$ is always a linear combination of the sample points. In other words, we can write it in the form $w =$ _____.
$X^\top a$ (for some vector $a$).

**(c)** [4 pts] Show that every principal component direction $w$ with a nonzero eigenvalue is a linear combination of the sample points (even when $n < d$).

As $w$ is an eigenvector of $X^\top X$, there is a $\lambda \in \mathbb{R}$ such that $X^\top X w = \lambda w$. Setting $a = \frac{1}{\lambda} X w$, we have $X^\top a = w$.

**(d)** [4 pts] Let $\Phi(z)$ be a feature map that takes a point $z \in \mathbb{R}^d$ and maps it to a point $\Phi(z) \in \mathbb{R}^D$, where $D$ might be extremely large or even infinite. But suppose that we can compute the kernel function $k(x, z) = \Phi(x) \cdot \Phi(z)$ much more quickly than we can compute $\Phi(x)$ directly. Let $\Phi(X)$ be the $n \times D$ matrix in which each sample point is replaced by a featurized point. By our usual convention, row $i$ of $X$ is $X_i^\top$, and row $i$ of $\Phi(X)$ is $\Phi(X_i)^\top$.

Remind us: what is the kernel matrix $K$? Answer this two ways: explain the relationship between $K$ and the kernel function $k(\cdot, \cdot)$; then write the relationship between $K$ and $\Phi(X)$. Lastly, show that these two definitions are equivalent.

$K$ is the $n \times n$ matrix with components $K_{ij} = k(X_i, X_j)$. Also, $K = \Phi(X)\Phi(X)^\top$.

These two characterizations are equivalent because $K_{ij} = k(X_i, X_j) = \Phi(X_i) \cdot \Phi(X_j)$ is the inner product of row $i$ of $\Phi(X)$ and column $j$ of $\Phi(X)^\top$, which implies that $K = \Phi(X)\Phi(X)^\top$.

**(e)** [2 pts] Fill in the space: the first principle component direction of the *featurized* design matrix $\Phi(X)$ is any nonzero vector $w \in \mathbb{R}^D$ that maximizes the *Rayleigh quotient*, which is .

$$\frac{w^\top \Phi(X)^\top \Phi(X) w}{w^\top w}.$$

**(f)** [4 pts] Show that the problem of maximizing this Rayleigh quotient is equivalent to maximizing

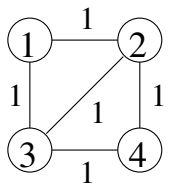
$$\frac{a^\top B a}{a^\top C a}$$

for some positive semidefinite matrices $B, C \in \mathbb{R}^{n \times n}$, where $a \in \mathbb{R}^n$ is a vector of dual weights. This expression is called a *generalized Rayleigh quotient*. What are the matrices $B$ and $C$? For full points, express them in a form that does not require any direct computation of the feature vectors $\Phi$, which could be extremely long.

$\frac{w^\top \Phi(X)^\top \Phi(X) w}{w^\top w} = \frac{a^\top \Phi(X)\Phi(X)^\top \Phi(X)\Phi(X)^\top a}{a^\top \Phi(X)\Phi(X)^\top a} = \frac{a^\top K^2 a}{a^\top K a}$. Hence $B = K^2$ and $C = K$.

# Q5. [12 pts] Spectral Graph Clustering

Let's apply spectral graph clustering to this graph.




**(a)** [4 pts] Write the Laplacian matrix $L$ for this graph. All the edges have weight 1.

$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

**(b)** [2 pts] Consider the minimum bisection problem, where we find an indicator vector $y$ that minimizes $y^\top Ly$, subject to the balance constraint $1^\top y = 0$ and the strict binary constraint $\forall i, y_i = 1$ or $y_i = -1$. Write an indicator vector $y$ that represents a minimum bisection of this graph.

Any one of $\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$ or $\begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$ or $\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$ or $\begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$ will do.

**(c)** [4 pts] Suppose we relax (discard) the binary constraint and replace it with the weaker constraint $y^\top y =$ constant, permitting $y$ to have real-valued components. (We keep the balance constraint.) What indicator vector is a solution to the relaxed optimization problem? What is its eigenvalue?

Hint: Look at the symmetries of the graph. Given that the continuous values of the $y_i$'s permit some of the vertices to be at or near zero, what symmetry do you think would minimize the continuous-valued cut? Guess and then check whether it's an eigenvector.

$\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$ or $\begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ or any nonzero multiple of these. The eigenvalue is 2.

**(d)** [2 pts] If we apply the sweep cut to find a cut with good sparsity, what two clusters do we get? Is it a bisection?

The sweep cut either puts vertex 1 in a subgraph by itself, or vertex 4 in a subgraph by itself. It does not choose a bisection.

# Q6. [17 pts] Learning Mixtures of Gaussians with k-Means

Let $X_1, \ldots, X_n \in \mathbb{R}^d$ be independent, identically distributed points sampled from a mixture of two normal (Gaussian) distributions. They are drawn independently from the probability distribution function (PDF)

$$p(x) = \theta\, N_1(x) + (1-\theta)\, N_2(x), \quad \text{where } N_1(x) = \frac{1}{(\sqrt{2\pi})^d} e^{-\|x-\mu_1\|^2/2} \text{ and } N_2(x) = \frac{1}{(\sqrt{2\pi})^d} e^{-\|x-\mu_2\|^2/2}$$

are the PDFs for the isotropic multivariate normal distributions $\mathcal{N}(\mu_1, 1)$ and $\mathcal{N}(\mu_2, 1)$, respectively. The parameter $\theta \in (0, 1)$ is called the *mixture proportion*. In essence, we flip a biased coin to decide whether to draw a point from the first Gaussian (with probability $\theta$) or the second (with probability $1 - \theta$).

Each data point is generated as follows. First draw a random $Z_i$, which has value 1 with probability $\theta$, and has value 2 with probability $1 - \theta$. Then, draw $X_i \sim \mathcal{N}(\mu_{Z_i}, 1)$. Our learning algorithm gets $X_i$ as an input, but does not know $Z_i$.

Our goal is to find the maximum likelihood estimates of the three unknown distribution parameters $\theta \in (0, 1)$, $\mu_1 \in \mathbb{R}^d$, and $\mu_2 \in \mathbb{R}^d$ from the sample points $X_1, \ldots, X_n$. Unlike MLE for one Gaussian, it is **not** possible to give explicit analytic formulas for these estimates. Instead, we develop a variant of $k$-means clustering which (often) converges to the correct maximum likelihood estimates of $\theta$, $\mu_1$, and $\mu_2$. This variant doesn't assign each point entirely to one cluster; rather, each point is assigned an estimated posterior probability of coming from normal distribution 1.

**(a)** [4 pts] Let $\tau_i = P(Z_i = 1 | X_i)$. That is, $\tau_i$ is the posterior probability that point $X_i$ has $Z_i = 1$. Use Bayes' Theorem to express $\tau_i$ in terms of $X_i$, $\theta$, $\mu_1$, $\mu_2$, and the Gaussian PDFs $N_1(x)$ and $N_2(x)$. To help you with part (c), also write down a similar formula for $1 - \tau_i$, which is the posterior probability that $Z_i = 2$.

Bayes' Theorem implies that

$$\tau_i = \frac{\theta N_1(X_i)}{\theta N_1(X_i) + (1-\theta) N_2(X_i)}, \quad 1 - \tau_i = \frac{(1-\theta) N_2(X_i)}{\theta N_1(X_i) + (1-\theta) N_2(X_i)}.$$

**(b)** [3 pts] Write down the log-likelihood function, $\ell(\theta, \mu_1, \mu_2; X_1, \ldots, X_n) = \ln p(X_1, \ldots, X_n)$, as a summation. Note: it doesn't simplify much.

Because the samples are iid, $p(X_1, \ldots, X_n) = \prod_{i=1}^n p(X_i)$, so

$$\ell(\theta, \mu_1, \mu_2; X_1, \ldots, X_n) = \sum_{i=1}^n \ln(\theta N_1(X_i) + (1-\theta) N_2(X_i)).$$

**(c)** [3 pts] Express $\frac{\partial \ell}{\partial \theta}$ in terms of $\theta$ and $\tau_i, i \in \{1, \ldots, n\}$ and simplify as much as possible. There should be no normal PDFs explicitly in your solution, though the $\tau_i$'s may implicitly use them. Hint: Recall that $(\ln f(x))' = \frac{f'(x)}{f(x)}$.

$$\frac{\partial \ell}{\partial \theta} = \sum_{i=1}^n \left( \frac{\tau_i}{\theta} - \frac{1-\tau_i}{1-\theta} \right) = \frac{1}{\theta - \theta^2} \sum_{i=1}^n (\tau_i - \theta) = \frac{(\sum \tau_i) - \theta n}{\theta - \theta^2}.$$

(Any of these expressions is simple enough to receive full marks.)

**(d)** [4 pts] Express $\nabla_{\mu_1}\ell$ in terms of $\mu_1$ and $\tau_i, X_i, i \in \{1,\dots,n\}$. Do the same for $\nabla_{\mu_2}\ell$ (but in terms of $\mu_2$ rather than $\mu_1$). Again, there should be no normal PDFs explicitly in your solution, though the $\tau_i$'s may implicitly use them.
Hint: It will help (and get you part marks) to first write $\nabla_{\mu_1} N_1(x)$ as a function of $N_1(x)$, $x$, and $\mu_1$.

$$\begin{aligned}\nabla_{\mu_1}\ell &= \sum_{i=1}^{n} \frac{\theta \nabla_{\mu_1} N_1(X_i)}{\theta N_1(X_i) + (1-\theta)N_2(X_i)} \\ &= \sum_{i=1}^{n} \frac{\theta N_1(X_i)}{\theta N_1(X_i) + (1-\theta)N_2(X_i)} (X_i - \mu_1) \\ &= \sum_{i=1}^{n} \tau_i (X_i - \mu_1).\end{aligned}$$

Similarly,

$$\nabla_{\mu_2}\ell = \sum_{i=1}^{n} (1-\tau_i)(X_i - \mu_2).$$

**(e)** [3 pts] We conclude: if we know $\mu_1$, $\mu_2$, and $\theta$, we can compute the posteriors $\tau_i$. On the other hand, if we know the $\tau_i$'s, we can estimate $\mu_1$, $\mu_2$, and $\theta$ by using the derivatives in parts (c) and (d) to find the maximum likelihood estimates. This leads to the following $k$-means-like algorithm.

- Initialize $\tau_1, \tau_2, \dots, \tau_n$ to arbitrary values in the range $[0, 1]$.
- Repeat the following two steps.
  1. Update the Gaussian cluster parameters: for fixed values of $\tau_1, \tau_2, \dots, \tau_n$, update $\mu_1$, $\mu_2$, and $\theta$.
  2. Update the posterior probabilities: for fixed values of $\mu_1$, $\mu_2$ and $\theta$, update $\tau_1, \tau_2, \dots, \tau_n$.

In part (a), you wrote the update rule for step 2. Using your results from parts (c) and (d), write down the explicit update formulas for step 1.

$$\mu_1 \leftarrow \frac{\sum_{i=1}^{n} \tau_i X_i}{\sum_{i=1}^{n} \tau_i}, \quad \mu_2 \leftarrow \frac{\sum_{i=1}^{n} (1-\tau_i) X_i}{\sum_{i=1}^{n} (1-\tau_i)}, \quad \theta \leftarrow \frac{1}{n} \sum_{i=1}^{n} \tau_i.$$