CS 189 Spring 2019 Introduction to Machine Learning Final Exam

- Please do not open the exam before you are instructed to do so.
- **Electronic devices are forbidden on your person**, including cell phones, iPods, headphones, and laptops. Turn your cell phone off and **leave all electronics at the front of the room**, or **risk getting a zero** on the exam.
- When you start, the **first thing you should do** is **check that you have all 12 pages and all 6 questions**. The second thing is to please **write your initials at the top right of every page after this one** (e.g., write "JS" if you are Jonathan Shewchuk).
- The exam is closed book, closed notes except your two cheat sheets.
- You have 3 hours.
- Mark your answers on the exam itself in the space provided. Do **not** attach any extra sheets.
- The total number of points is 150. There are 26 multiple choice questions worth 3 points each, and 5 written questions worth a total of 72 points.
- For multiple answer questions, fill in the bubbles for **ALL correct choices:** there may be more than one correct choice, but there is always at least one correct choice. **NO partial credit** on multiple answer questions: the set of all correct answers must be checked.

First name	
Last name	
SID	
First and last name of student to your left	
First and last name of student to your right	

Q1. [78 pts] Multiple Answer

Fill in the bubbles for **ALL correct choices**: there may be more than one correct choice, but there is always at least one correct choice. **NO partial credit**: the set of all correct answers must be checked.

(a) [3 pts] Which of the following algorithms can learn nonlinear decision boundaries? The decision trees use only axis-aligned splits.

[✓ A depth-five decision tree] [✓ AdaBoost with depth-one decision trees]

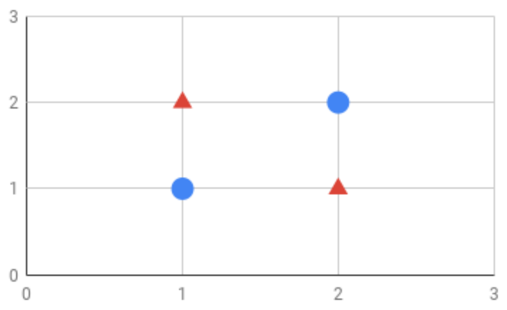
[✓ Quadratic discriminant analysis (QDA)] ○ Perceptron

The solutions are obvious other than AdaBoost with depth-one decision trees, where you can form non-linear boundaries due to the final classifier not actually being a linear combination of the linear weak learners. Do not assume added features on exam unless said otherwise. If every ML exam question carried an assumption of "you can add any extra features you want unless we explicitly say otherwise", then all of our exam answers would be wrong. As for AdaBoost, you do end up with a -1 or 1 at the end, but the decision BOUNDARY could be nonlinear.

(b) [3 pts] Which of the following classifiers are capable of achieving 100% training accuracy on the data below? The decision trees use only axis-aligned splits.

○ Logistic regression ○ AdaBoost with depth-one decision trees

[✓ A neural network with one hidden layer] [✓ AdaBoost with depth-two decision trees]



top left: Each weak learner will either classify the points from each pair in different classes, or classify every point in the same class. Since the meta classifier is a weighted sum of all of these weak classifiers, each which has a 50% training accuracy, the meta classifier cannot have 100% accuracy.

top right: A neural network with one hidden layer (with enough units) is a universal function approximator.

lower left: Logistic regression finds a linear decision boundary, which cannot separate the data.

lower right: A depth two decision tree can fully separate the data.

(c) [3 pts] Which of the following are true of support vector machines?

[✓ Increasing the hyperparameter C tends to decrease the training error] [✓ Increasing the hyperparameter C tends to decrease the margin]

○ The hard-margin SVM is a special case of the soft-margin with the hyperparameter C set to zero ○ Increasing the hyperparameter C tends to decrease the sensitivity to outliers

Top left: True, from the lecture notes.

Bottom left: False, Hard-margin SVM is where C tends towards infinity.

Top right: false, perceptron is trained using gradient descent and SVM is trained using a quadratic program.

Bottom right: True: slack becomes less expensive, so you allow data points to be farther on the wrong side of the margin and make the margin bigger. Doing this will never reduce the number of data points inside the margin.

(d) [3 pts] Let $r(x)$ be a decision rule that minimizes the risk for a three-class classifier with labels $y \in \{0, 1, 2\}$ and an asymmetric loss function. What is true about $r(\cdot)$?

○ $\forall y \in \{0, 1, 2\}, \quad \exists x : r(x) = y$

○ $\forall x$, $r(x)$ is a class y that maximizes the posterior probability $P(Y = y|X = x)$

● If we don't have access to the underlying data distribution $P(X)$ or $P(Y|X)$, we cannot exactly compute the risk of $r(\cdot)$

● If $P(X = x)$ changes but $P(Y = y|X = x)$ remains the same for all x, y, $r(X)$ still minimizes the risk

top left: it is possible that $r(X)$ is the same for all X.

top right: no, because the risk is asymmetric

lower left: by definition of risk we need to be able to compute expectations over these two distributions.

lower right: Given that $r(X)$ has no constraint, it can pick the y that minimizes risk for every $X = x$ without trade-offs. Therefore, if only the marginals change, that choice is not affected.

(e) [3 pts] Which of the following are true about two-class Gaussian discriminant analysis? Assume you have estimated the parameters $\hat{\mu}_C, \hat{\Sigma}_C, \hat{\pi}_C$ for class C and $\hat{\mu}_D, \hat{\Sigma}_D, \hat{\pi}_D$ for class D.

○ If $\hat{\mu}_C = \hat{\mu}_D$ and $\hat{\pi}_C = \hat{\pi}_D$, then the LDA and QDA classifiers are identical

● If $\hat{\Sigma}_C = \hat{\Sigma}_D$, $\hat{\pi}_C = 1/6$, and $\hat{\pi}_D = 5/6$, then the LDA and QDA classifiers are identical

○ If $\hat{\Sigma}_C = I$ (the identity matrix) and $\hat{\Sigma}_D = 5I$, then the LDA and QDA classifiers are identical

○ If the LDA and QDA classifiers are identical, then the posterior probability $P(Y = \text{C}|X = x)$ is linear in x

Top left: false, the covariance matrices might differ, making the QDA decision function nonlinear.

Bottom left: false, the QDA decision function is nonlinear. Check out the figure on p. 36 of Lecture 7 and look at the two sites furthest to the right. one of which has a much wider Gaussian than the other). That's the QDA classifier for this example (maybe the constant of 5 is not quite right, but close enough), and it has an elliptical decision boundary. For the same points, the LDA decision boundary would just be a straight line.

Top right: correct. All points in R^d are classified as class D by both classifiers. So the decision boundary is the empty set.

Bottom right: no, the posterior is a logistic function.

(f) [3 pts] Consider an $n \times d$ design matrix X with labels $y \in \mathbb{R}^n$. What is true of fitting this data with dual ridge regression with the polynomial kernel $k(X_i, X_j) = (X_i^T X_j + 1)^p = \Phi(X_i)^\top \Phi(X_j)$ and regularization parameter $\lambda > 0$?

○ If the polynomial degree is high enough, the polynomial will fit the data exactly

● The algorithm solves an $n \times n$ linear system

○ The algorithm computes $\Phi(X_i)$ and $\Phi(X_j)$ in $O(d^p)$ time

○ When n is very large, this dual algorithm is more likely to overfit than the primal algorithm with degree-p polynomial features

Top right: see definition of dual ridge regression

Lower right: both give the same solution, no matter n!

Top left: The dual method problem of ridge regression is indeed recommended only when $d > n$. But in this case we use a Kernel, so in fact we have a number of features of $d' = d^p$! Also, In ridge regression, we would never reach the point of perfectly fitting on the data due to the regularization term. Ridge regression helps us prevent from overfitting on the training data regardless of the degree of the polynomial features.

Bottom left: no need! just their dot product, which can easily be obtained with $(X_i^T X_j + 1)^p$.

(g) [3 pts] Consider the kernel perceptron algorithm on an $n \times d$ design matrix X. We choose a matrix $M \in \mathbb{R}^{D \times d}$ and define the feature map $\Phi(x) = Mx \in \mathbb{R}^D$ and the kernel $k(x, z) = \Phi(x) \cdot \Phi(z)$. Which of the following are always true?

● The kernel matrix is $XM^\top MX^\top$

○ The kernel matrix is $MX^\top XM^\top$

● If the primal perceptron algorithm terminates, then the kernel perceptron algorithm terminates

● If the kernel perceptron algorithm terminates, then the primal perceptron algorithm terminates

Bottom left: True. The dual is computational faster in this case than the primal. So if the primal finished, then the dual has as well

Top left: Yes, because K=$\Phi(X)\Phi(X)^\top$ and $\Phi(X)=XM^\top$.

Top right: No algebarically if we do the substuition

Bottom right: The dual is computational faster in this case than the primal. If the projected data is linearly separable, the original data must be linearly separable too. (But the reverse is not true.)

(h) [3 pts] Which of the following are true of decision trees? Assume splits are binary and are done so as to maximize the information gain.

○ If there are at least two classes at a given node, there exists a split such that information gain is strictly positive

● The deeper the decision tree is, the more likely it is to overfit

○ As you go down any path from the root to a leaf, the information gain at each level is non-increasing

● Random forests are less likely to overfit than decision trees

Top left: false, recall example from section. consider the first split on an XOR example. Note that there does not exist a binary, axis-aligned linear split that can give you positive information gain here (the entropy before and after the split is the same).

Bottom left: false, recall example from section. (same example) the information gain at the second split is larger than the first one.

Top right: correct.

Bottom right: correct.

(i) [3 pts] While solving a classification problem, you use a pure, binary decision tree constructed by the standard greedy procedure we outlined in class. While your training accuracy is perfect, your validation accuracy is unexpectedly low. Which of the following, in isolation, is likely to improve your validation accuracy in most real-world applications?

○ Lift your data into a quadratic feature space

○ Normalize each feature to have variance 1

○ Select a random subset of the features and use only those in your tree

● Prune the tree, using validation to decide how to prune

Top left: False, lifting to a more complex feature space will not generally stop you from overfitting.

Bottom left: False, an ensemble of standard decision trees fit to the same data-set will not learn

Top right: The small change in split criterion will not generally stop you from overfitting.

Bottom right: Correct, lowering depth defends against overfitting.

(j) [3 pts] For the sigmoid activation function and the ReLU activation function, which of the following are true in general?

● Both activation functions are monotonically non-decreasing

○ Compared to the sigmoid, the ReLU is more computationally expensive

○ Both functions have a monotonic first derivative

● The sigmoid derivative $s'(\gamma)$ is quadratic in $s(\gamma)$

Top left: True. Simply graph the activation functions
Bottom left: False. Sigmoid has non-monotonic derivative
Top right: False. ReLU is simpler as all positives have derivative 1 and all negatives have 0. While we have to calculate exponential for Sigmoid
Bottom right: True. Unlike Sigmoid, the product of gradients of ReLU function doesn't end up converging to 0 as the value is either 0 or 1

(k) [3 pts] Which of the following are true in general for backpropagation?

- [✓ It is a dynamic programming algorithm]
- ○ The weights are initially set to zero
- [✓ Some of the derivatives cannot be fully computed until the backward pass]
- ○ Its running time grows exponentially in the number of layers

Top left: False. As it is not a model, but a quick algorithm to compute derivatives in the network b True. We have a forward pass and a backward pass

Bottom right: False. Linear time complexity instead

Top right: False. The weights set randomly

Bottom left: True. In the backward pass

(l) [3 pts] Facets of neural networks that have (reasonable, though not perfect) analogs in human brains include

- ○ backpropagation
- ○ convolutional masks applied to many patches
- [✓ linear combinations of input values]
- [✓ edge detectors]

(m) [3 pts] Which of the following are true of the vanishing gradient problem for sigmoid units?

- [✓ Deeper neural networks tend to be more susceptible to vanishing gradients]
- [✓ Using ReLU units instead of sigmoid units can reduce this problem]
- ○ If a unit has the vanishing gradient problem for one training point, it has the problem for every training point
- ○ Networks with sigmoid units don't have this problem if they're trained with the cross-entropy loss function

Top left: false, as the number of layers goes up, the gradient is more likely to vanish during backpropagation. If one node yields a gradient close to zero, the gain of the nodes in the previous layers will also be very low.

Bottom left: true, ReLU is generally better since its gradient does not go to zero as the input goes to zero.

Top right: false, if gradients are vanishing, the weights have already effectively stopped changing their values.

Bottom right: true, this is the incentive for ResNets.

(n) [3 pts] Suppose our input is two-dimensional sample points, with ten non-exclusive classes those points may belong to (i.e., a point can belong to more than one class). To train a classifier, we build a fully-connected neural network (with bias terms) that has a single hidden layer of twenty units and an output layer of ten units (one for each class). Which statements apply?

- ○ For the output units, softmax activations are more appropriate than sigmoid activations
- [✓ For the hidden units, ReLU activations are more appropriate than linear activations]
- ○ This network will have 240 trainable parameters
- [✓ This network will have 270 trainable parameters]

Softmax will create a valid probability distribution across all the outputs, making it well suited to predicting the single class a point is most likely to belong to but not to predicting whether or not the point is in each class. Sigmoid will give us a valid in-class probability for each class independently, allowing us to perform multiclass predictions.

There are $2 * 20 + 20 = 60$ parameters in the first layer and $20 * 10 + 10 = 210$ in the second, giving us a total of 270 parameters.

With a linear activation on the hidden layer, this network reduces to a perceptron and cannot model non-linear decision boundaries.

Randomly initializing the weight terms breaks the symmetry of the network; its okay (and in fact standard practice) to initialize the bias terms to zero.

(o) [3 pts] Which of the following can lead to valid derivations of PCA?

- ● Fit the mean and covariance matrix of a Gaussian distribution to the sample data with maximum likelihood estimation
- ○ Find the direction w that minimizes the sum of projection distances
- ○ Find the direction w that minimizes the sample variance of the projected data
- ● Find the direction w that minimizes the sum of squares of projection distances

This is best explained by the lecture notes - in particular, lecture note 20 from the Spring 2019 iteration of the course.

(p) [3 pts] Write the SVD of an $n \times d$ design matrix X (with $n \geq d$) as $X = UDV^T$. Which of the following are true?

- ● The components of D are all nonnegative
- ● The columns of V all have unit length and are orthogonal to each other
- ○ If X is a real, symmetric matrix, the SVD is always the same as the eigendecomposition
- ● The columns of D are orthogonal to each other

Top left: True.

Bottom left: False. Consider a symmetric matrix with negative eigenvalues, like [[-1,0],[0,-2]] whose eigenvalues are -1 and -2. Singular values cannot be negative (by definition), so the singular values are 1 and 2. The SVD makes up for the signs of the singular values by making the right singular vectors have signs opposite of those of the left singular vectors.

Top right: True.

Bottom right: False, a subset of the columns of V correspond to the null space of X. The zero vector is the only vector in a Euclidean space that is orthogonal to all other vectors in that space (including itself)

(q) [3 pts] Which of the following is true about Lloyd's algorithm for k-means clustering?

- ○ It is a supervised learning algorithm
- ● If run for long enough, it will always terminate
- ● It never returns to a particular assignment of classes to sample points after changing to another one
- ○ No algorithm (Lloyd's or any other) can always find the optimal solution

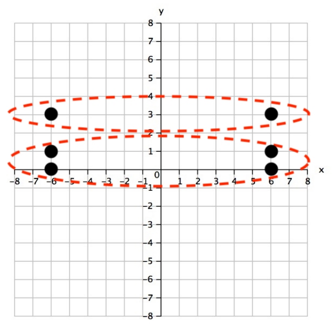
k-means is an unsupervised learning algorithm. The number of clusters k is a hyperparameter.
Bottom Right: Clustering is NP hard and only able to find local optimum not global optimum, according to lecture 21. It should say "Unless P = NP, no polynomial time algorithm can always find the optimal solution." There is a trivial brute-force algorithm to find the optimal solution.

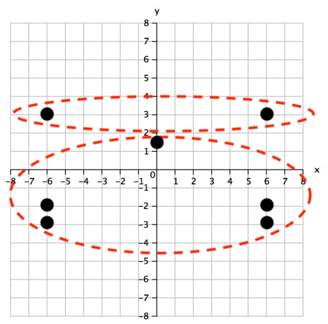
(r) [3 pts] Which of the following are advantages of using k-medoid clustering instead of k-means?

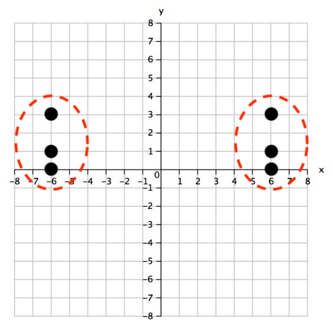
- ● k-medoids is less sensitive to outliers
- ○ Medoids are faster to compute than means
- ● Medoids make more sense than means for non-Euclidean distance metrics
- ○ The k-medoids algorithm with the Euclidean distance metric has no hyperparameters, unlike k-means

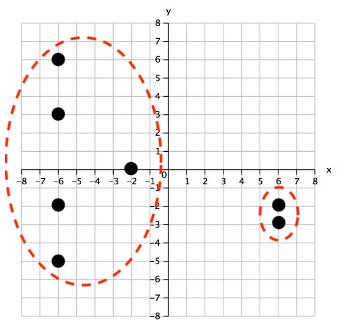
Both k means and k medoids have k as a hyperparameter. Medoids are much more expensive to compute than means (calculating all pairwise distances, rather than just summing all points and averaging).

(s) [3 pts] We wish to cluster 2-dimensional points into two clusters, so we run Lloyd's algorithm for *k*-means clustering until convergence. Which of the following clusters could it produce? (The points inside an oval belong to the same cluster).

[✓] 

○ 

[✓] 

[✓] 

In the top right image, the mean of the top cluster would be at (0,3) and the mean of the bottom cluster would be somewhere around (0,-1.7). That point that is around (0,1.5) is closer to the mean of the top cluster than it is to the mean of the bottom cluster, so Lloyd's algorithm has not converged.

The algorithm converges if none of the points are closer to the mean of a cluster they are not currently assigned to. In the top right figure, the point on the y axis is closer to the mean of the top cluster than it is to the mean of the bottom cluster it is currently assigned to, so at the next iteration of the algorithm that point would switch clusters. In the top left image, none of the points are closer to the mean of the cluster they were not assigned to. Because finding a solution to the k-means optimization problem is NP hard, we use Lloyd's algorithm, which is an iterative approach. While this algorithm is guaranteed to converge, it may end up at a local minimum that is not globally optimal.

(t) [3 pts] Which of the following are true of hierarchical clustering?

○ The number *k* of clusters is a hyperparameter

○ Complete linkage works only with the Euclidean distance metric

[✓] The greedy agglomerative clustering algorithm repeatedly fuses the two clusters that minimize the distance between clusters

[✓] During agglomerative clustering, single linkage is more sensitive to outliers than complete linkage

Top left: Part of the point of hierarchy is so you don't have to guess *k* in advance

Bottom left: Correct

Top right: Complete linkage is compatible with any distance function

Bottom right: Single linkage is very sensitive to outliers

(u) [3 pts] Which of the following are true of spectral clustering?

○ The Fiedler vector is the eigenvector associated with the second largest eigenvalue of the Laplacian matrix

[✓] The relaxed optimization problem for partitioning a graph involves minimizing the Rayleigh quotient of the Laplacian matrix and an indicator vector (subject to a constraint)

[✓] Nobody knows how to find the sparsest cut in polynomial time

○ The Laplacian matrix of a graph is invertible

Top left: The Fiedler vector corresponds to the second smallest eigenvalue.

Bottom left: The relaxed optimization problem minimizes the rayleigh quotient with constraints.

Top right: It's NP-hard.

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]

For part w:

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 jth row of A, let b_k be the kth 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?

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

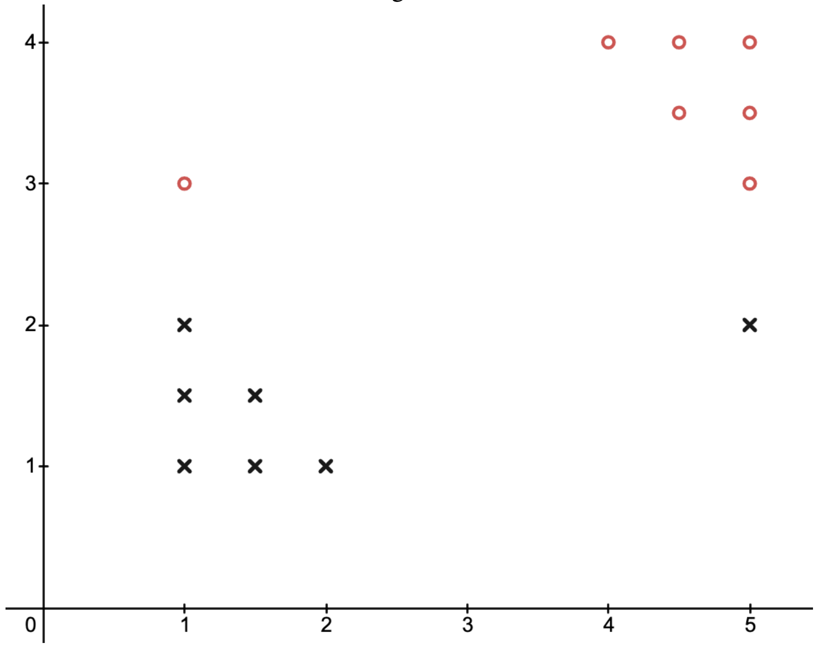
(Because the rank-1 truncated SVD is $\delta_1 u_1 v_1^\top$, and it minimizes the Frobenius norm reconstruction error. This is equivalent to Mr. Funk's objective when R has no zeros.)

Q3. [10 pts] Decision Boundaries

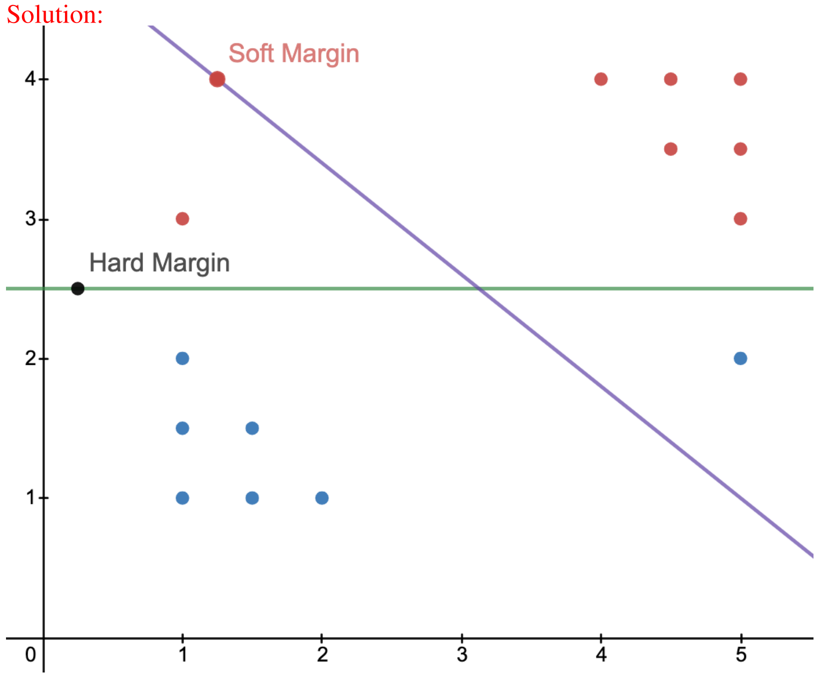
In the question, you will draw the decision boundaries that classifiers would learn.

(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.



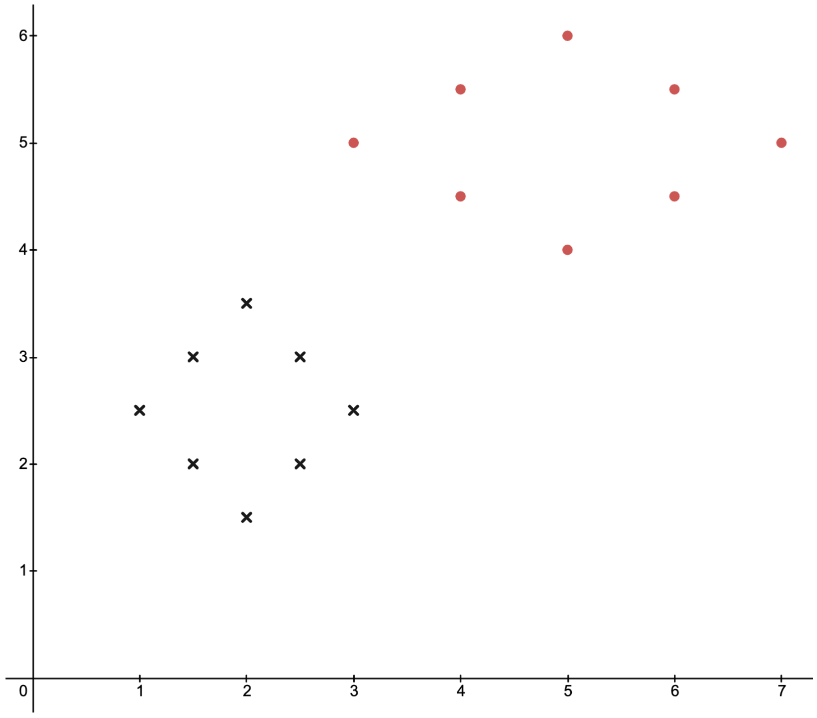
Solution:



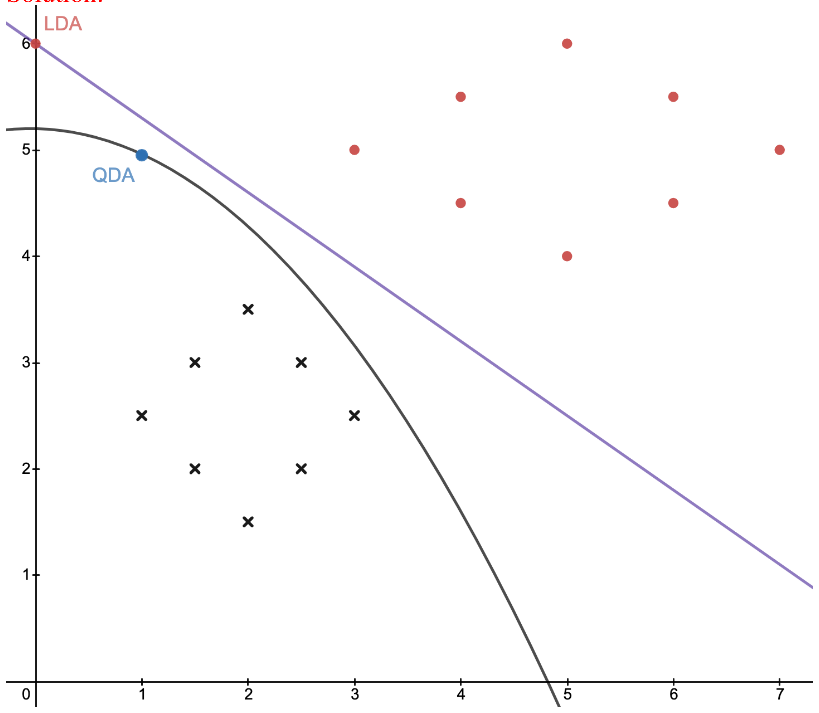
With a soft-margin SVM, we care more about maximizing the distance from the margin to each of the data points, and less about classifying every point correctly; the way to achieve such an objective for these points in 3A is by a downward-sloped line, which would achieve wide margins from most of the points as opposed to the hard margin SVM.

Regarding "the same center line", if you mean a horizontal line for the soft-margin SVM, that wouldn't receive credit since it's not maximizing the distance from the points compared to the downward-sloping line.

(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:



Draw the isocontours first. Say, three isocontours for each of the two classes, big enough to overlap each other. Then look at where the red isocontours intersect the black isocontours.

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

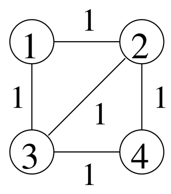
$$\frac{a^\top B a}{a^\top C a}$$

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 Φ, 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 θ) 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 θ, 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 θ, μ_1, and μ_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, τ_i is the posterior probability that point X_i has $Z_i = 1$. Use Bayes' Theorem to express τ_i in terms of X_i, θ, μ_1, μ_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 θ 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 τ_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 μ_1 and $\tau_i, X_i, i \in \{1, \dots, n\}$. Do the same for $\nabla_{\mu_2}\ell$ (but in terms of μ_2 rather than μ_1). Again, there should be no normal PDFs explicitly in your solution, though the τ_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 μ_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 μ_1, μ_2, and θ, we can compute the posteriors τ_i. On the other hand, if we know the τ_i's, we can estimate μ_1, μ_2, and θ 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 μ_1, μ_2, and θ.
  2. Update the posterior probabilities: for fixed values of μ_1, μ_2 and θ, 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.

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