# CS 189 Spring 2020 — Introduction to Machine Learning — Final Exam

- The exam is open book, open notes, and open web. However, you may not consult or communicate with other people (besides your exam proctors).
- You will submit your answers to the multiple-choice questions through Gradescope via the assignment "**Final Exam – Multiple Choice**"; please **do not** submit your multiple-choice answers on paper. By contrast, you will submit your answers to the written questions by writing them on paper by hand, scanning them, and submitting them through Gradescope via the assignment "**Final Exam – Writeup**."
- Please write your name at the top of each page of your written answers. (You may do this before the exam.)
- You have 180 minutes to complete the midterm exam (3:00–6:00 PM). (If you are in the DSP program and have an allowance of 150% or 200% time, that comes to 270 minutes or 360 minutes, respectively.)
- When the exam ends (6:00 PM), **stop writing**. You must submit your multiple-choice answers before 6:00 PM sharp. Late multiple-choice submissions will be penalized at a rate of 5 points per minute after 6:00 PM. (The multiple-choice questions are worth 60 points total.)
- From 6:00 PM, you have 15 minutes to scan the written portion of your exam and turn it into Gradescope via the assignment "Final Exam – Writeup." Most of you will use your cellphone and a third-party scanning app. If you have a physical scanner, you may use that. Late written submissions will be penalized at a rate of 5 points per minute after 6:15 PM.
- Mark your answers to multiple-choice questions directly into Gradescope. Write your answers to written questions on blank paper. **Clearly label all written questions and all subparts of each written question. Show your work in written questions.**
- Following the exam, you must use Gradescope's **page selection mechanism** to mark which questions are on which pages of your exam (as you do for the homeworks).
- The total number of points is 150. There are 16 multiple choice questions worth 4 points each, and six written questions worth 86 points total.
- 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 | |

## Q1. [64 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.

**(1)** [4 pts] Which of the following are true for the $k$**-nearest neighbor** ($k$-NN) algorithm?

[✓ A: $k$-NN can be used for both classification and regression.]

[✓ B: As $k$ increases, the bias usually increases.]

○ C: The decision boundary looks smoother with smaller values of $k$.

○ D: As $k$ increases, the variance usually increases.

**(2)** [4 pts] Let $X$ be a matrix with **singular value decomposition** $X = U\Sigma V^\top$. Which of the following are true for all $X$?

[✓ A: $\text{rank}(X) = \text{rank}(\Sigma)$.]

○ B: If all the singular values are unique, then the SVD is unique.

[✓ C: The first column of $V$ is an eigenvector of $X^\top X$.]

[✓ D: The singular values and the eigenvalues of $X^\top X$ are the same.]

A is correct because the number of non-zero singular values is equal to the rank. B is incorrect because you could change both $U \to -U, V \to -V$. C is correct because the SVD and eigendecomposition of $X^\top X$ is $V\Sigma^2 V^\top$. D is correct as $X^\top X$ is positive semidefinite, so the eigenvalues can't be negative.

**(3)** [4 pts] Lasso (with a fictitious dimension), random forests, and principal component analysis (PCA) **all** . . .

[✓ A: can be used for dimensionality reduction or feature subset selection]

○ B: compute linear transformations of the input features

○ C: are supervised learning techniques

[✓ D: are *translation invariant*: changing the origin of the coordinate system (i.e., translating all the training and test data together) does not change the predictions or the principal component directions]

Option B is incorrect because random forests don't compute linear transformations. Option C is incorrect because PCA is unsupervised.

**(4)** [4 pts] Suppose your training set for two-class classification in one dimension ($d = 1$; $x_i \in \mathbb{R}$) contains three sample points: point $x_1 = 3$ with label $y_1 = 1$, point $x_2 = 1$ with label $y_2 = 1$, and point $x_3 = -1$ with label $y_3 = -1$. What are the values of $w$ and $b$ given by a **hard-margin SVM**?

○ A: $w = 1, b = 1$

○ B: $w = 0, b = 1$

[✓ C: $w = 1, b = 0$]

○ D: $w = \infty, b = 0$

**(5)** [4 pts] Use the same training set as part (d). What is the value of $w$ and $b$ given by **logistic regression** (with no regularization)?

○ A: $w = 1, b = 1$

○ B: $w = 0, b = 1$

○ C: $w = 1, b = 0$

[✓ D: $w = \infty, b = 0$]

**(6)** [4 pts] Below are some choices you might make while training a neural network. Select all of the options that will generally make it **more difficult** for your network to achieve high accuracy on the test data.

[✓ A: Initializing the weights to all zeros]

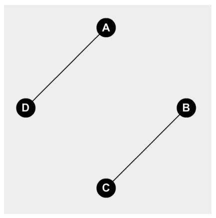
[✓ B: Normalizing the training data but leaving the test data unchanged]

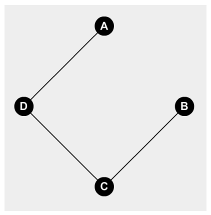
○ C: Using momentum

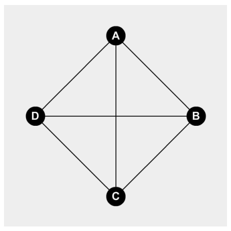
○ D: Reshuffling the training data at the beginning of each epoch

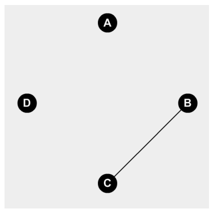
A) Initializing weights with zeros makes it impossible to learn. B) Mean and standard deviation should be computed on the training set and then used to standardize the validation and test sets, so that the distributions are matched for each set. C) This describes momentum and will generally help training. D) This is best practice.

**(7)** [4 pts] To the left of each graph below is a number. Select the choices for which the number is the multiplicity of the eigenvalue zero in the **Laplacian matrix** of the graph.

○ A: 1 

○ C: 2 

[✓ B: 1] 

○ D: 4 

The multiplicity is equal to the number of connected components in the graph.

**(8)** [4 pts] Given the spectral graph clustering optimization problem

$$\begin{aligned} &\text{Find } y \text{ that minimizes } y^\top L y \\ &\text{subject to} \quad y^\top y = n \\ &\text{and} \quad \mathbf{1}^\top y = 0, \end{aligned}$$

which of the following optimization problems produce a vector $y$ that leads to **the same sweep cut** as the optimization problem above? $M$ is a diagonal mass matrix with different masses on the diagonal.

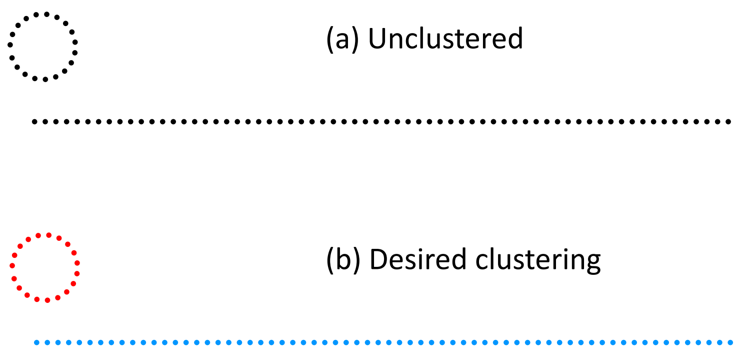
[✓ A:] Minimize $y^\top L y$ subject to $y^\top y = 1$ and $\mathbf{1}^\top y = 0$

[✓ C:] Minimize $y^\top L y / (y^\top y)$ subject to $\mathbf{1}^\top y = 0$

○ B: Minimize $y^\top L y$ subject to $\forall i,\ y_i = 1$ or $y_i = -1$ and $\mathbf{1}^\top y = 0$

○ D: Minimize $y^\top L y$ subject to $y^\top M y = 1$ and $\mathbf{1}^\top M y = 0$

**(9)** [4 pts] Which of the following methods will cluster the data in panel (a) of the figure below into the two clusters (red circle and blue horizontal line) shown in panel (b)? Every dot in the circle and the line is a data point. In all the options that involve hierarchical clustering, the algorithm is run until we obtain two clusters.



○ A: Hierarchical agglomerative clustering with Euclidean distance and complete linkage

[✓ B: Hierarchical agglomerative clustering with Euclidean distance and single linkage]

◯ C: Hierarchical agglomerative clustering with Euclidean distance and centroid linkage

◯ D: $k$-means clustering with $k = 2$

Single linkage uses the minimum distance between two clusters as a metric for merging clusters. Since the two clusters are densely packed with points and the minimum distance between the two clusters is greater than the within-cluster distances between points, single linkage doesn't link the circle to the line until the very end.

The other three methods will all join some of the points at the left end of the line with the circle, before they are joined with the right end of the line.

**(10)** [4 pts] Which of the following statement(s) about **kernels** are true?

[✓ A: The dimension of the lifted feature vectors $\Phi(\cdot)$, whose inner products the kernel function computes, can be infinite.]

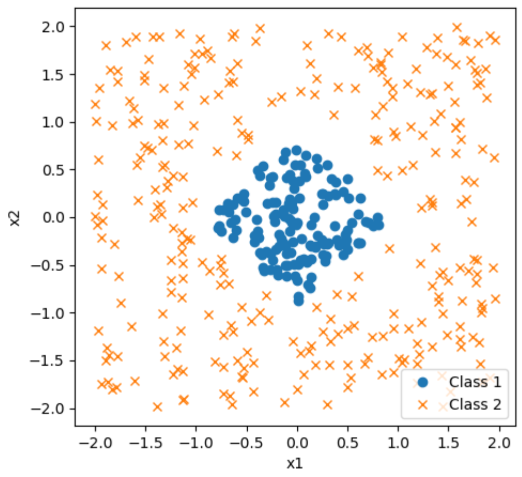
B: For any desired lifting $\Phi(x)$, we can design a kernel function $k(x,z)$ that will evaluate $\Phi(x)^\top\Phi(z)$ more quickly than explicitly computing $\Phi(x)$ and $\Phi(z)$.

[✓ C: The kernel trick, when it is applicable, speeds up a learning algorithm if the number of sample points is substantially less than the dimension of the (lifted) feature space.]

[✓ D: If the raw feature vectors $x, y$ are of dimension 2, then $k(x,y) = x_1^2 y_1^2 + x_2^2 y_2^2$ is a valid kernel.]

A is correct; consider the Gaussian kernel from lecture. B is wrong; most liftings don't lead to super-fast kernels. Just some special ones do. C is correct, straight from lecture. Though in this case, the dual algorithm is faster than the primal whether you use a fancy kernel or not. D is correct because $k(x,y)$ is inner product of $\Phi(x) = [x_1^2 \quad x_2^2]^\top$ and $\Phi(y) = [y_1^2 \quad y_2^2]^\top$.

**(11)** [4 pts] We want to use a **decision tree** to classify the training points depicted. Which of the following decision tree classifiers is capable of giving 100% accuracy on the training data with **four splits or fewer**?



A: A standard decision tree with axis-aligned splits

B: Using PCA to reduce the training data to one dimension, then applying a standard decision tree

[✓ C: A decision tree with multivariate linear splits]

[✓ D: Appending a new feature $|x_1| + |x_2|$ to each sample point $x$, then applying a standard decision tree]

A standard decision tree will need (substantially) more than four splits. PCA to 1D will make it even harder. However, four non-axis-aligned multivariate linear splits suffice to cut the diamond out of the center. Finally, adding the $L_1$ norm feature lets us perfectly classify the data with a single split that cuts off the top of the pyramid.

**(12)** [4 pts] Which of the following are true about **principal components analysis** (PCA)?

A: The principal components are eigenvectors of the centered data matrix.

[✓ B: The principal components are right singular vectors of the centered data matrix.]

[✓ C: The principal components are eigenvectors of the sample covariance matrix.]

[✓ D: The principal components are right singular vectors of the sample covariance matrix.]

The first three follow directly from definitions. The last is because the covariance matrix is symmetric, so the singular vectors are the eigenvectors.

**(13)** [4 pts] Suppose we are doing **ordinary least-squares linear regression** with a fictitious dimension. Which of the following changes can **never** make the cost function's value on the **training data** smaller?

● A: Discard the fictitious dimension (i.e., don't append a 1 to every sample point).

○ B: Append quadratic features to each sample point.

● C: Project the sample points onto a lower-dimensional subspace with PCA (without changing the labels) and perform regression on the projected points.

● D: Center the design matrix (so each feature has mean zero).

A: Correct. Discarding the fictitious dimension forces the linear regression function to be zero at the origin, which may increase the cost function but can never decrease it.

B: Incorrect. Added quadratic features often help to fit the data better.

C: Correct. Regular OLS is at least as expressive. Projecting the points may incrase the cost function but can never decrease it. Centering features doesn't matter so WLOG assume $X$ has centered features. If the full SVD is $X = U\Sigma V^\top$, then projecting onto a $k$-dimensional subspace gives $X_k = U\Sigma_k V^\top$. If $w_k$ is a solution for the PCA-projected OLS, we can take $w = Vz$ where $z$ is the first $k$ elements of $V^\top w_k$ with the rest zero, and get $X_k w_k = Xw$.

D: Correct. Since we're using a fictitious dimension, translating the points does not affect the cost of the optimal regression function (which translates with the points).

**(14)** [4 pts] Which of the following are true about **principal components analysis** (PCA)? Assume that no two eigenvectors of the sample covariance matrix have the same eigenvalue.

● A: Appending a 1 to the end of every sample point doesn't change the results of performing PCA (except that the useful principal component vectors have an extra 0 at the end, and there's one extra useless component with eigenvalue zero).

● B: If you use PCA to project $d$-dimensional points down to $j$ principal coordinates, and then you run PCA again to project those $j$-dimensional coordinates down to $k$ principal coordinates, with $d > j > k$, you always get the same result as if you had just used PCA to project the $d$-dimensional points directly down to $k$ principle coordinates.

○ C: If you perform an arbitrary rigid rotation of the sample points as a group in feature space before performing PCA, the principal component directions do not change.

● D: If you perform an arbitrary rigid rotation of the sample points as a group in feature space before performing PCA, the largest eigenvalue of the sample covariance matrix does not change.

Appending an extra dimension with the same values introduces no variance in the extra dimension, so PCA will ignore that dimension. PCA discards the eigenvector directions associated with the largest eigenvalues; as the eigenvectors are mutually orthogonal, this does not affect the variance in the surviving dimensions, so your results depend solely on how many directions you discard. Rotating the sample points rotates the principal components, but it doesn't change the variance along each of those (rotated) component directions.

**(15)** [4 pts] Consider running a single iteration of **AdaBoost** on three sample points, starting with uniform weights on the sample points. All the ground truth labels and predictions are either +1 or −1. In the table below, some values have been omitted. Which of the following statements can we say **with certainty**?

| | True Label | Classifier Prediction | Initial Weight | Updated Weight |
|---|---|---|---|---|
| $X_1$ | $-1$ | $-1$ | $1/3$ | ? |
| $X_2$ | ? | $+1$ | $1/3$ | $\sqrt{2}/3$ |
| $X_3$ | ? | ? | $1/3$ | $\sqrt{2}/6$ |

● A: $X_1$'s updated weight is $\sqrt{2}/6$

○ B: $X_3$'s classifier prediction is $-1$

● C: $X_2$ is misclassified

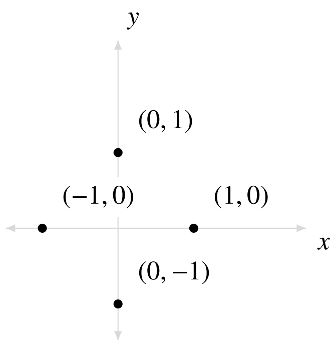
○ D: $X_3$ is misclassified

In the AdaBoost algorithm, all correctly classified points have their weights changed by the same multiplicative factor. Since we observe two different updated weights, we know one of $x_2$ or $x_3$ is correctly classified, and the other is misclassified. Since $x_1$ is correctly classified, the error rate is err $= 1/3$. As the error rate is less than $1/2$, the weights of correctly classified points will decrease and the weights of misclassified points will increase. Hence, $X_2$ is misclassified and $X_3$ is correctly classified. As $X_1$ is correctly classified, it has the same updated weight as $X_3$. But we can't tell what $X_3$'s classifier prediction is; only that it is correctly classified.

As an aside, we can confirm the multipliers used for reweighting of misclassified and correctly classified points (in that order):

$$\sqrt{\frac{\text{err}}{1-\text{err}}} = \sqrt{\frac{2/3}{1/3}} = \sqrt{2} \qquad\qquad \sqrt{\frac{1-\text{err}}{\text{err}}} = \sqrt{\frac{1/3}{2/3}} = \frac{\sqrt{2}}{2}$$

**(16)** [4 pts] Consider running the **hierarchical agglomerative clustering** algorithm on the following set of four points in $\mathbb{R}^2$, breaking ties arbitrarily. If we stop when only two clusters remain, which of the following linkage methods *ensures* the resulting clusters are balanced (each have two sample points)? Select all that apply.



[✓ A: Complete linkage]

B: Single linkage

[✓ C: Centroid linkage]

[✓ D: Average linkage]

Under each of these linkage methods, we know that two adjacent points along a side of the rhombus will first be fused together. Without loss of generality, assume these are the points $(0, 1)$ and $(1, 0)$. Treating these as a cluster, the average, maximum and centroid distances to each of the two remaining points are all larger that the distance between the two points themselves. Therefore, complete linkage, single linkage and average linkage all result in balanced clusters. On there other hand, single linkage does not, since, for example, $(1, 0)$ and $(0, -1)$ have the same distance as $(-1, 0)$ and $(0, -1)$.

# Q2. [14 pts] Principal Components Analysis

Consider the following design matrix, representing four sample points $X_i \in \mathbb{R}^2$.

$$X = \begin{bmatrix} 4 & 1 \\ 2 & 3 \\ 5 & 4 \\ 1 & 0 \end{bmatrix}.$$

We want to represent the data in only one dimension, so we turn to principal components analysis (PCA).

**(1)** [5 pts] Compute the **unit-length principal component directions** of $X$, and **state which one the PCA algorithm would choose** if you request just one principal component. Please provide an exact answer, without approximation. (You will need to use the square root symbol.) **Show your work!**

We center $X$, yielding

$$\dot{X} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 2 & 2 \\ -2 & -2 \end{bmatrix}.$$

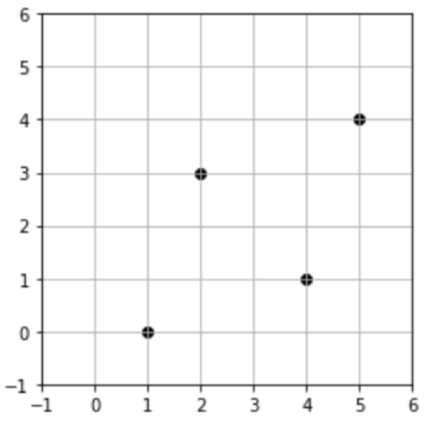
Then $\dot{X}^\top \dot{X} = \begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix}$. (Divide by 4 if you want the sample covariance matrix. But we don't care about the magnitude.)

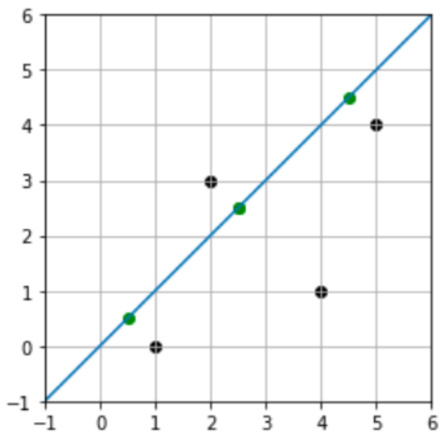
Its eigenvectors are $[1/\sqrt{2} \quad 1/\sqrt{2}]^\top$ with eigenvalue 16 and $[1/\sqrt{2} \quad -1/\sqrt{2}]^\top$ with eigenvalue 4. The former eigenvector is chosen.

(Negated versions of these vectors also get full points.)

**(2)** [5 pts] The plot below depicts the sample points from $X$. We want a one-dimensional representation of the data, so **draw the principal component direction (as a line) and the projections of all four sample points onto the principal direction**.

**Label each projected point with its principal coordinate value** (where the origin's principal coordinate is zero). Give the principal coordinate values exactly.



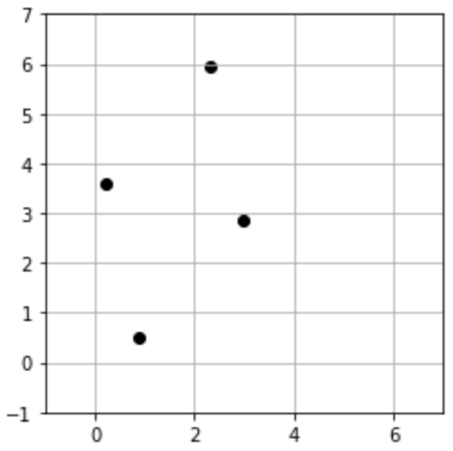


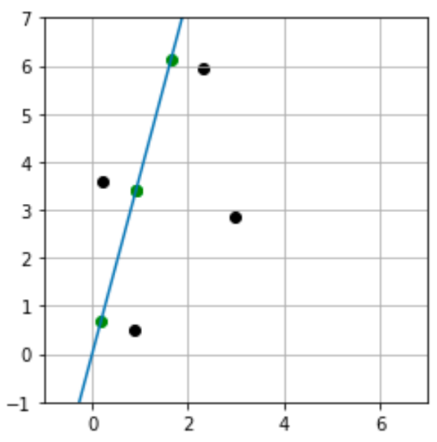
The principal coordinates are $\frac{1}{\sqrt{2}}$, $\frac{5}{\sqrt{2}}$, $\frac{5}{\sqrt{2}}$, and $\frac{9}{\sqrt{2}}$. (Alternatively, all of these could be negative, but they all have to have the same sign.)

**(3)** [4 pts] The plot below depicts the sample points from $X$ rotated 30 degrees counterclockwise about the origin.

As in part (b), **identify the principal component direction that the PCA algorithm would choose and draw it (as a line) on the plot**. Also **draw the projections of the rotated points** onto the principal direction.

**Label each projected point with the exact value of its principal coordinate.**





The line passes through the origin and is parallel to the two sample points that are farthest apart, so it's easy to draw. Rotation has not changed the principal coordinates: $\frac{1}{\sqrt{2}}$, $\frac{5}{\sqrt{2}}$, $\frac{5}{\sqrt{2}}$, and $\frac{9}{\sqrt{2}}$. (Again, these could all be negative.)

## Q3. [14 pts] A Decision Tree

In this question we investigate whether students will pass or fail CS 189 based on whether or not they studied, cheated, and slept well before the exam. You are given the following data for five students. There are three features, “Studied,” “Slept,” and “Cheated.” The column “Result” shows the label we want to predict.

| | **Studied** | **Slept** | **Cheated** | **Result** |
|---|---|---|---|---|
| **Student 1** | Yes | No | No | Passed |
| **Student 2** | Yes | No | Yes | Failed |
| **Student 3** | No | Yes | No | Failed |
| **Student 4** | Yes | Yes | Yes | Failed |
| **Student 5** | Yes | Yes | No | Passed |

**(1)** [4 pts] What is the **entropy** $H$(Result) at the root node? (There is no need to compute the exact number; you may write it as an arithmetic expression.)

$$H(\text{Result}) = -\left(\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5}\right).$$

**(2)** [5 pts] **Draw the decision tree** where every split maximizes the information gain. (An actual drawing, please; a written description does not suffice.) Do not perform a split on a pure leaf or if the split will produce an empty child; otherwise, split. **Explain (with numbers)** why you chose the splits you chose.

A tree that first splits on “Cheated” and then “Studied.”

**(3)** [2 pts] Did the tree you built implicitly perform feature subset selection? **Explain.**

Yes, because it does not use the feature “Slept.”

**(4)** [3 pts] Suppose you have a sample of $n$ students for some large $n$, with the same three features. Assuming that we use a reasonably efficient algorithm to build the tree (as discussed in class), what is the **worst-case running time to build** the decision tree? (Write your answer in the simplest asymptotic form possible.) **Why?**

We have 3 binary features, so the tree’s depth cannot exceed 3 and each sample point participates in at most four treenodes. Hence, it cannot take more than $\Theta(n)$ time to build the tree.

# Q4. [20 pts] Spectral Graph Clustering

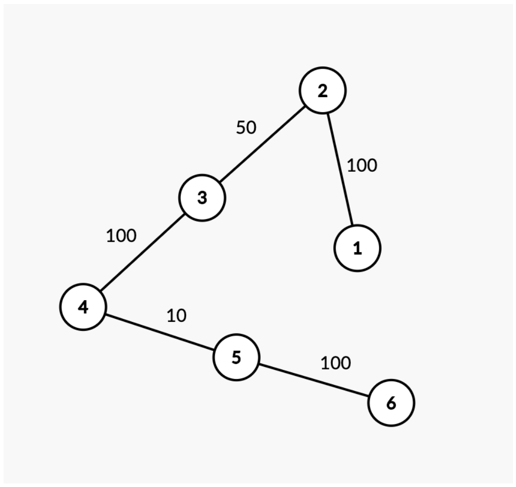


Figure 1: An undirected, weighted graph in which all vertices have mass 1. The numbers inside the vertices are their indices (not masses).

In this problem, we approximate the sparsest cut of the graph above with the spectral graph clustering algorithm. Recall the spectral graph clustering optimization objective is to

$$\begin{aligned} &\text{find } y \text{ that minimizes } y^\top L y \\ &\text{subject to} \quad y^\top y = 6 \\ &\qquad \text{and} \quad \mathbf{1}^\top y = 0. \end{aligned}$$

**(1)** [4 pts] Write out the **Laplacian matrix** $L$.

$$\begin{bmatrix} 100 & -100 & 0 & 0 & 0 & 0 \\ -100 & 150 & -50 & 0 & 0 & 0 \\ 0 & -50 & 150 & -100 & 0 & 0 \\ 0 & 0 & -100 & 110 & -10 & 0 \\ 0 & 0 & 0 & -10 & 110 & -100 \\ 0 & 0 & 0 & 0 & -100 & 100 \end{bmatrix}$$

**(2)** [3 pts] What is the rank of $L$? **Explain your answer.**

5, because the Laplacian matrix of a connected graph has one and only one eigenvector with eigenvalue zero.

**(3)** [4 pts] $L$ has the following six unit eigenvectors, listed in random order. **Write down the Fiedler vector. Then explain how you can tell which one is the Fiedler vector without doing a full eigendecomposition computation.** (There are several ways to see this; describe one.)

○ $[0.36, -0.58, 0.61, -0.4, 0.04, -0.03]$

○ $[-0.56, -0.32, 0.43, 0.62, -0.06, -0.1]$

○ $[0.3, -0.33, -0.23, 0.3, -0.6, 0.55]$

● $[0.36, 0.34, 0.25, 0.19, -0.55, -0.6]$

○ $[-0.41, 0.41, 0.41, -0.41, -0.41, 0.41]$

○ $[0.41, 0.41, 0.41, 0.41, 0.41, 0.41]$

The Fiedler vector is $v_2 = [0.36, 0.34, 0.25, 0.19, -0.55, -0.6]$. Since we know these are all eigenvectors of the Laplacian matrix $L$, one way to identify it is to explicitly multiply the first row of $L$ by an eigenvector, and divide by the first component of the eigenvector to reveal its eigenvalue. (The Fiedler vector has eigenvalue $\lambda_2 = 6.5$; all the others except $v_1$ have larger eigenvalues.) An easier way is to notice that only the Fiedler vector is monotonic in one direction (decreasing in this example).

**(4)** [5 pts] **How does the sweep cut decide** how to cut this graph into two clusters? (Explain in clear English sentences.) For every cut considered by the algorithm, **write down the "score"** it is assigned by the sweep cut algorithm. (You may use fractions; decimal numbers aren't required.) **Identify the chosen cut** by writing down two sets of vertex indices.

The sweep cut sorts the values in the Fiedler vector, then decides which pair of consecutive vertices to cut between by explicitly computing the sparsity of each of the five possible cuts. The cut with the lowest sparsity wins.

From left to right in the Fiedler vector, the sparsity of each cut is 20, $\frac{50}{8} = 6.25$, $\frac{100}{9} \doteq 11.11$, $\frac{10}{8} = 1.25$, and 20.

The clusters are $\{5, 6\}$ and $\{4, 3, 1, 2\}$.

(Cut between the values 0.19 and $-0.55$.)

**(5)** [4 pts] Suppose we have computed the four eigenvectors $v_1, v_2, v_3, v_4$ corresponding to the four largest eigenvalues. (For simplicity, assume no two eigenvectors have the same eigenvalue.) We want to **write a constrained optimization problem that identifies the eigenvector corresponding to the fifth-largest eigenvalue**. Explain how to modify the optimization problem at the beginning of this question so that the vector $y$ it finds is the desired eigenvector. (You may change the objective function and/or add constraints, but they must be mathematical, and you cannot write things like "subject to $y$ being the eigenvector corresponding to the fifth-largest eigenvalue.")

Add the following constraints: $v_2^\top y = 0$, $v_3^\top y = 0$, $v_4^\top y = 0$.

# Q5. [12 pts] Hierarchical Spectral Graph Multi-Clustering

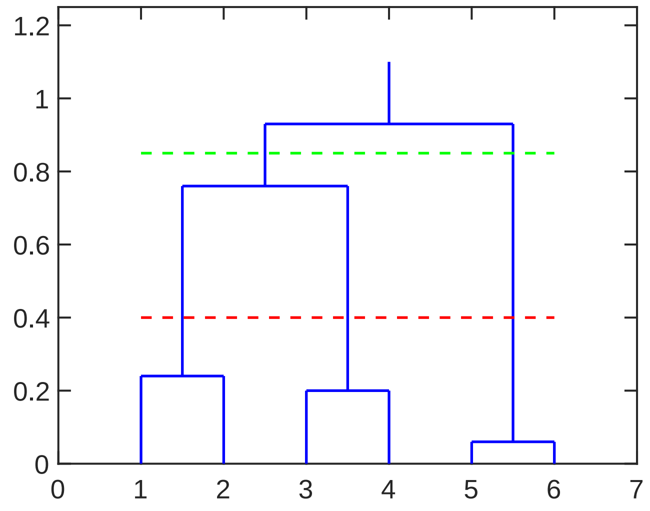
In this problem, we shall consider the same graph as in the previous question, but we use multiple eigenvectors to perform 3-cluster clustering with the algorithm of Ng, Jordan, and Weiss (as opposed to the 2-cluster clustering we performed in the last question).

**(1)** [4 pts] Based on the six eigenvectors of $L$ given in the previous question, **write down the spectral vector** (as defined in the lecture notes) **for each vertex** $1, \ldots, 6$ **in that order**.

The matrix of spectral vectors is $\begin{bmatrix} 0.41 & 0.36 & -0.56 \\ 0.41 & 0.34 & -0.32 \\ 0.41 & 0.25 & 0.43 \\ 0.41 & 0.19 & 0.62 \\ 0.41 & -0.55 & -0.06 \\ 0.41 & -0.6 & -0.1 \end{bmatrix}$.

**(2)** [8 pts] We shall now cluster the six raw, unnormalized spectral vectors obtained above using **hierarchical agglomerative clustering with the Euclidean distance metric and single linkage**. In contrast to what was discussed in class, we are not normalizing the six spectral vectors (because we don't want you to work that hard). Draw the **complete single linkage dendrogram** on paper. The six integer points on the $x$-axis, $1, \ldots, 6$, should represent the vertices of the graph in the order of their indices. The $y$-axis should indicate the linkage distances, as is standard for dendrograms. **The numerical distance at which each fusion happens should be clearly marked on your figure.**

Points 5 and 6 merge first; the Euclidean distance between them is 0.06. Points 3 and 4 merge next; the Euclidean distance between them is 0.2. Points 1 and 2 merge next; the Euclidean distance between them is 0.24. The single linkage distance between $\{1, 2\}$ and $\{3, 4\}$ is 0.76. The single linkage distance between $\{1, 2\}$ and $\{5, 6\}$ is 0.93. That between $\{3, 4\}$ and $\{5, 6\}$ is 0.94. Therefore, the fourth merger is between $\{1, 2\}$ and $\{3, 4\}$. The clusters after the fourth merger are $\{1, 2, 3, 4\}$ and $\{5, 6\}$, matching what we obtained in the previous question. $\{1, 2, 3, 4\}$ and $\{5, 6\}$ merge at 0.93. The green dotted line indicates the 2-cluster clustering. The red dotted line indicates the 3-cluster clustering. (Note tha students aren't asked to specify those lines.)

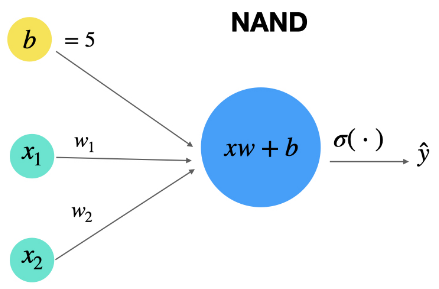


# Q6. [10 pts] A Miscellany

**(1)** [4 pts] Consider a single unit in a neural network that receives two binary inputs $x_1, x_2 \in \{0,1\}^2$ and computes a linear combination followed by a threshold activation function, namely,

$$\sigma(z) = \begin{cases} 1, & z \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

The unit is illustrated below. We have chosen a bias term of $b = 5$. **Provide values for the two weights $w_1$ and $w_2$** that allow you to compute the NAND function (which is 0 if and only if both inputs are 1).



$w_1 = -3$, $w_2 = -3$ will work.

**(2)** [6 pts] We are drawing sample points from a distribution with the probability density function (PDF) $f(x) = \frac{1}{2}e^{-|x-\mu|}$, but we do not know the mean $\mu \in \mathbb{R}$. We decide to estimate $\mu$ with maximum likelihood estimation (MLE). Unfortunately, we have only two sample points $X_1, X_2 \in \mathbb{R}$.

**Derive the likelihood and the log-likelihood** for this problem. Then **show that every value of $\mu$ between $X_1$ and $X_2$ is a maximum likelihood estimate**.

The likelihood is

$$\mathcal{L}(\mu; X_1, X_2) = \frac{1}{4}e^{-|X_1-\mu|}e^{-|X_2-\mu|},$$

and the log-likelihood is

$$\ell(\mu; X_1, X_2) = -|X_1 - \mu| - |X_2 - \mu| - \ln 4.$$

For any $\mu$ between $X_1$ and $X_2$ (inclusive), the log-likelihood is $-|X_1 - X_2| - \ln 4$. For any $\mu$ outside that range, it is lesser. (For example if $\mu$ is less than $\min\{X_1, X_2\}$, then the log-likelihood is $-|X_1 - X_2| - 2|\mu - \min\{X_1, X_2\}| - \ln 4$).

# Q7. [16 pts] Dual Ridge Regression & Leave-One-Out Error

[This question has four independent parts. If you get stuck on one, try the others. Each part depends on the statements made in the previous parts, but not on your answer to the previous parts. Please show your work!]

Let $X$ be an $n \times d$ design matrix representing $n$ sample points with $d$ features. (The last column of $X$ may or may not be all 1's, representing a fictitious dimension; it won't affect this question.) Let $\mathbf{y} \in \mathbb{R}^n$ be a vector of labels. As usual, $X_i$ denotes the $i$th sample point expressed as a column vector ($X_i^\top$ is row $i$ of $X$) and $y_i$ denotes the $i$th scalar component of $\mathbf{y}$. Recall that **ridge regression** finds the weight vector $\mathbf{w}^*$ minimizing the cost function

$$J(\mathbf{w}) = \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|^2$$

where $\lambda > 0$ is the regularization hyperparameter. Because $\lambda > 0$, every regression problem we will consider here has exactly one unique minimizer. For $X$ and $\mathbf{y}$, the unique minimizer of $J$ is denoted by $\mathbf{w}^*$, giving a unique linear hypothesis $h(\mathbf{z}) = \mathbf{w}^* \cdot \mathbf{z}$.

**(1)** [4 pts] Regression doesn't usually have zero training error; we would like to check the value $h(X_i) = \mathbf{w}^* \cdot X_i$ to see how close it is to $y_i$. **Recall the dual form of ridge regression and use it to show** that $\mathbf{w}^* \cdot X_i = \mathbf{y}^\top(K + \lambda I)^{-1}K_i$, where $K$ is the kernel matrix and $K_i$ is column $i$ of $K$. Show your work. *(Note: we are not lifting the sample points to another feature space; we are just doing dual ridge regression with kernel matrix $K = XX^\top$.)*

In dual ridge regression, we set $\mathbf{w} = X^\top\mathbf{a}$ where $\mathbf{a} \in \mathbb{R}^n$ is a vector of dual weights, and the optimal dual solution is $\mathbf{a}^* = (K+\lambda I)^{-1}\mathbf{y}$, so $\mathbf{w}^* = X^\top(K+\lambda I)^{-1}\mathbf{y}$. Thus $\mathbf{w}^* \cdot X_i = \mathbf{y}^\top(K+\lambda I)^{-1}XX_i$. Column $i$ of $K = XX^\top$ is $XX_i$, so $\mathbf{w}^* \cdot X_i = \mathbf{y}^\top(K+\lambda I)^{-1}K_i$.

The **Leave-One-Out (LOO)** error of a regression algorithm is the expected loss on a randomly chosen training point when you train on the other $n-1$ points, leaving the chosen point out of training. Let $X^i$ denote the $(n-1) \times d$ design matrix obtained by removing the sample point $X_i$ (the $i$th row of $X$) from $X$, and let $\mathbf{y}^i \in \mathbb{R}^{n-1}$ denote the vector obtained by removing $y_i$ from $\mathbf{y}$. Let $J^i$ be the cost function of ridge regression on $X^i$ and $\mathbf{y}^i$, and let $\mathbf{w}^i$ be the optimal weight vector that minimizes $J^i(\mathbf{w})$.

**(2)** [4 pts] Suppose that after we perform ridge regression on $X^i$ and $\mathbf{y}^i$, we discover that our linear hypothesis function just happens to fit the left-out sample point perfectly; that is, $\mathbf{w}^i \cdot X_i = y_i$.

**Prove that** $\mathbf{w}^* = \mathbf{w}^i$. That is, removing the sample point $X_i$ did not change the weights or the linear hypothesis. *(Hint: find the difference between $J(\mathbf{w})$ and $J^i(\mathbf{w})$ (for an arbitrary $\mathbf{w}$), then reason about the relationships between $J(\mathbf{w})$, $J^i(\mathbf{w})$, $J^i(\mathbf{w}^i)$, and $J(\mathbf{w}^i)$.)*

$$J(\mathbf{w}) = \sum_{j=1}^{n}(X_j \cdot \mathbf{w} - y_j)^2 + \lambda\|\mathbf{w}\|^2 \quad \text{and} \quad J^i(\mathbf{w}) = \sum_{j \neq i}(X_j \cdot \mathbf{w} - y_j)^2 + \lambda\|\mathbf{w}\|^2.$$

Hence $J(\mathbf{w}) - J^i(\mathbf{w}) = (X_i \cdot \mathbf{w} - y_i)^2$. Therefore, $J(\mathbf{w}) \geq J^i(\mathbf{w})$ for all $\mathbf{w}$. By assumption, $\mathbf{w}^i \cdot X_i = y_i$, so $J(\mathbf{w}^i) = J^i(\mathbf{w}^i)$. Recall that $\mathbf{w}^i$ is the weight vector that minimizes $J^i$.

It follows that for every $\mathbf{w} \in \mathbb{R}^d$, $J(\mathbf{w}) \geq J^i(\mathbf{w}) \geq J^i(\mathbf{w}^i) = J(\mathbf{w}^i)$. Hence $\mathbf{w}^i$ minimizes $J$. $J$ has only one unique minimizer, which we call $\mathbf{w}^*$, so $\mathbf{w}^* = \mathbf{w}^i$.

Suppose we are not so lucky, and it turns out that $\mathbf{w}^i \cdot X_i \neq y_i$. Let $\mathbf{y}^{(i)} \in \mathbb{R}^n$ denote the vector obtained by taking $\mathbf{y}$ and changing the $i$th component, replacing $y_i$ with $\mathbf{w}^i \cdot X_i$. Let $\mathbf{w}^{(i)}$ be the optimal weight vector that minimizes the ridge regression cost function on the inputs $X$ and $\mathbf{y}^{(i)}$. Our result from part (b) shows that $\mathbf{w}^{(i)} = \mathbf{w}^i$.

**(3)** [4 pts] From part (a), **show that** $\mathbf{w}^{(i)} \cdot X_i - \mathbf{w}^* \cdot X_i = (\mathbf{w}^{(i)} \cdot X_i - y_i)\,(K + \lambda I)_i^{-1}K_i$, where $(K + \lambda I)_i^{-1}$ denotes row $i$ of $(K + \lambda I)^{-1}$. *(Hint: The result from part (a) implies that $\mathbf{w}^{(i)} \cdot X_i = \mathbf{y}^{(i)} \cdot (K + \lambda I)^{-1}K_i$. What does $\mathbf{y}^{(i)} - \mathbf{y}$ look like?)*

$\mathbf{y}^{(i)} - \mathbf{y} = [0 \quad \dots \quad 0 \quad \mathbf{w}^i \cdot X_i - y_i \quad 0 \quad \dots \quad 0]^\top$, a vector of all zeros except in component $i$, so by part (a),

$$\mathbf{w}^{(i)} \cdot X_i - \mathbf{w}^* \cdot X_i = (\mathbf{y}^{(i)} - \mathbf{y})^\top(K + \lambda I)^{-1}K_i = (\mathbf{w}^i \cdot X_i - y_i)\,(K + \lambda I)_i^{-1}K_i = (\mathbf{w}^{(i)} \cdot X_i - y_i)\,(K + \lambda I)_i^{-1}K_i.$$

The Leave-One-Out error is defined to be

$$R_{\text{LOO}} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^{i} \cdot X_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^{(i)} \cdot X_i - y_i)^2.$$

The LOO error is often an excellent estimator for the regression loss on unseen data. In general, the computation of LOO error can be very costly because it requires training the algorithm $n$ times. But for dual ridge regression, remarkably, the LOO error can be computed by training the algorithm only once! Let's see how to compute the terms in the summation quickly.

**(4)** [4 pts] **Show that** $\mathbf{w}^{(i)} \cdot X_i - y_i = \dfrac{\mathbf{w}^{*} \cdot X_i - y_i}{1 - (K + \lambda I)_i^{-1} K_i}.$

From part (c), we have

$$\begin{aligned}
(\mathbf{w}^{(i)} \cdot X_i - y_i) - (\mathbf{w}^{*} \cdot X_i - y_i) &= (\mathbf{w}^{(i)} \cdot X_i - y_i)(K + \lambda I)_i^{-1} K_i \\
(\mathbf{w}^{(i)} \cdot X_i - y_i)(1 - K + \lambda I)_i^{-1} K_i) &= \mathbf{w}^{*} \cdot X_i - y_i \\
\mathbf{w}^{(i)} \cdot X_i - y_i = \frac{\mathbf{w}^{*} \cdot X_i - y_i}{1 - (K + \lambda I)_i^{-1} K_i}.
\end{aligned}$$

Postscript: We can add the kernel trick to this method if we want; it adds no difficulties, though for speed we usually want to use the kernel function to compute $K$ and each $\mathbf{w}^{*} \cdot X_i$. The technique also requires us to compute the diagonal of $(K + \lambda I)^{-1} K$, which is probably best done by a Cholesky factorization of $(K + \lambda I)^{-1}$ and backsubstitution.