Midterm 1, CS-UY 4563

Sample Questions

Show all of your work to receive full (and partial) credit.

Always, Sometimes, Never

Indicate whether each of the following statements is **always** true, **sometimes** true, or **never** true. Provide a one or two short justification or example to explain your choice.

1. For random events $p(x \mid y) < p(x, y)$.

   ALWAYS SOMETIMES **NEVER**

   By definition $p(x \mid y) = p(x, y)/p(y)$. Since $p(y) \leq 1$ (its a probability) $p(x \mid y) \geq p(x, y)$ always.

2. You use gradient descent to find parameters $\vec{\beta}_{GD}$ for a multiple linear regression problem under ℓ_2 loss: $L(\vec{\beta}) = \|X\vec{\beta} - \vec{y}\|_2^2$. You are short on time, so you only run gradient descent for 10 iterations. Your friend finds parameters $\vec{\beta}_M$ using the equation $\vec{\beta}_M = (X^T X)^{-1} X^T y$. Is $L(\vec{\beta}_m) \leq L(\vec{\beta}_{GD})$?

   ALWAYS SOMETIMES NEVER

   For linear regression, $\vec{\beta}_M = (X^T X)^{-1} X^T y$ *minimizes the loss* as shown in class (by setting the gradient of the loss to 0). So $L(\vec{\beta}_m) \leq L(\vec{\beta})$ for *any* β.

3. Does $\vec{\beta}_m$ acheive better population risk than $\vec{\beta}_{GD}$?

   ALWAYS **SOMETIMES** NEVER

   Typically we might expect $\vec{\beta}_m$ to obtain better population risk -- we minize loss on the training set with the goal of achieving good population risk, so doing a better job minimizing loss often leads to better risk. However, as we saw from studying regularization **this is not always the case**. In some cases, a parameter vector which has worse test loss (like one found with regularization) can achieve better population risk.

4. To evaluate machine learning models, you should use a train-test split instead of k-fold cross validation.

   ALWAYS **SOMETIMES** NEVER

   Train-test split is computationally faster by a factor of k, k-fold cross validation gives a better estimate of the model performance for a given amount of data. So depending on if you have a limited amount of data, finite computational resources, etc. you might choose one or the other.

Short Answer

1. You are trying to develop a machine learning algorithm for classifying data $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ into catagories $1, \ldots, q$. You have decided to use linear classification for the problem.

   (a) You know you can find a good linear classifier for *binary* classification (dividing into $q = 2$ classes) using logistic regression. You are considering using either the **one-vs-all** or **one-vs-one** approach to adapting this approach to the multiclass problem. In a few sort sentences describe why you might use one over the other.

   One-vs-all is faster -- it only requires training q classifiers. This is one of the primary reasons its used in practice, especially for many-class problems like image classification. One-vs.-one is slower ($O(q^2)$ classifiers need to be trained), but can generally lead to better classification (see example from class).

   (b) Your coworker suggests the following alternative approach: let's try to learn a parameter vector $\vec{\beta} \in \mathbb{R}^d$ and classify using the following model:

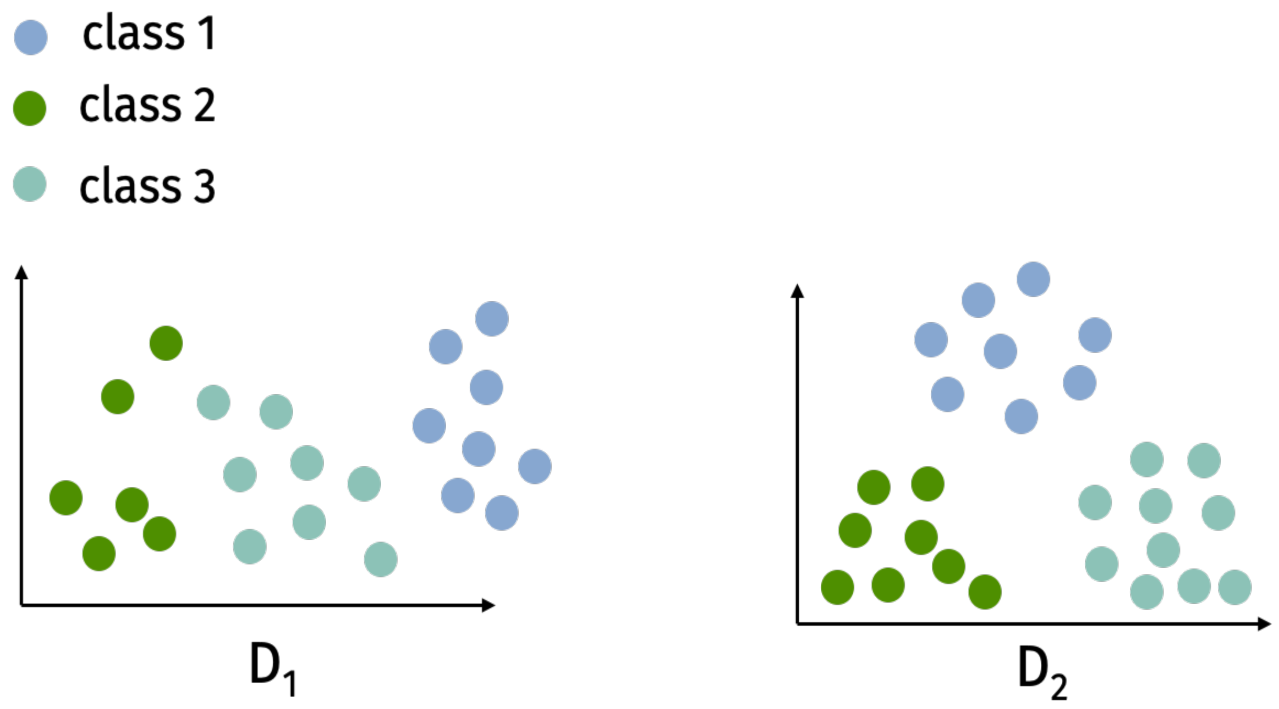
$$f_{\vec{\beta}}(\vec{x}) = \begin{cases} 1 \text{ if} & \langle \vec{\beta}, \vec{x} \rangle \leq 1 \\ 2 \text{ if} & 2 < \langle \vec{\beta}, \vec{x} \rangle \leq 3 \\ 3 \text{ if} & 3 < \langle \vec{\beta}, \vec{x} \rangle \leq 4 \\ \vdots & \\ q-1 \text{ if} & q-2 < \langle \vec{\beta}, \vec{x} \rangle \leq q-1 \\ q \text{ if} & q-1 < \langle \vec{\beta}, \vec{x} \rangle \end{cases} \tag{1}$$

Describe **one potential issue** and **one potental benefit** of your coworker's method over the approaches mentioned in (a). There is no one "right" answer here.

One major issue with your coworkers model is that it bakes in an assumption that class $i-1$ lies inbetween class i and class $i+1$. This is the same issue that motivated us to use one-hot encoding. For example, suppose our data only had single variable (draw for your self on a line if it helps) then it would need to be that examples in class two have values in between those of class 1 and 3 to get good prediction.

One potential benefit is that it leads to a very simple and fast classifier -- just compute one inner product and threshold. Both other methods required learning and classifying with at least q, up to $O(q^2)$ classifiers.

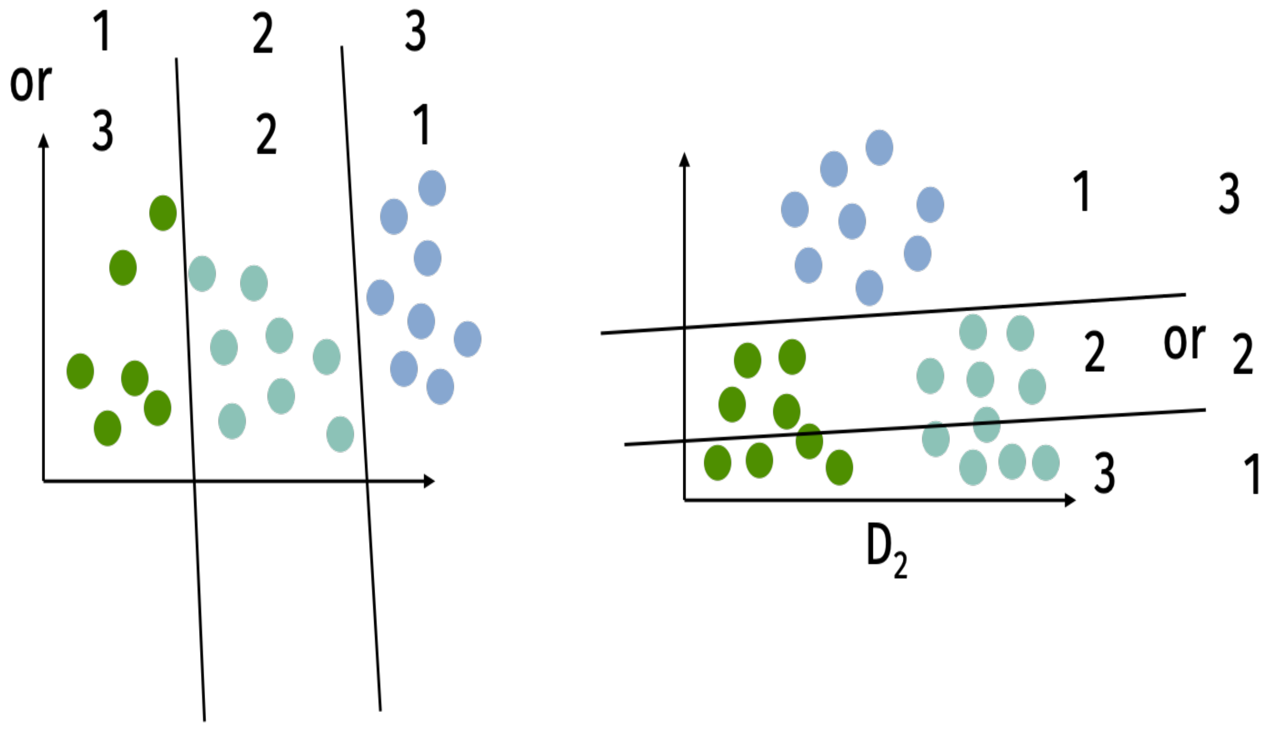
(c) For the two datasets D_1 and D_2 below, indicate which of the three approaches (**one-vs-one**, **one-vs-all**, or your **coworkers approach**) would lead to an accurate solution to the multiclass classification problem. No explanation is required, but having one might help you earn partial credit.



One-vs.-on: Gets both D_1, D_2.

One-vs.-all: Gets D_2 but not D_1 (same example we saw in class)

Coworkers approach: Neither. One way to look at the coworkers approach is that it will learn a single serperating hyperplane and then seperate classes using different shifts of that plane up and down. This leads to a grid of different regions, but it has to classify those *in order*. Meaning either the regions will be classified 1,2,3 left to right, or right to left. No classifier of that type can seperate this data, which I try to visualize below:



2. We are given data with just one predictor variable and one target: $(x_1, y_1), \ldots, (x_n, y_n)$, with the goal of fitting a degree two polynomial model using unregularized multiple linear regression with data transformation. The goal is to find the best coefficients $\beta_0, \beta_1, \beta_2$ for predicting y as $\beta_0 + \beta_1 x + \beta_2 x^2$.

   Consider the following three transformed data matrices:

$$X_1 = \begin{bmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}, X_2 = \begin{bmatrix} 1 & x_1^2 - x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n^2 - x_n & x_n^2 \end{bmatrix}, \text{ and } X_3 = \begin{bmatrix} 1 & 2x_1^2 - x_1 & 2x_1 - 4x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & 2x_n^2 - x_n & 2x_n - 4x_n^2 \end{bmatrix}$$

   Which of the above matrices can be used to solve this problem? In other words, if we train a multiple linear regression problem with X_i can we obtain an optimal degree two polynomial fit for $y_1, \ldots, y_n$. Justify your answer in words, or with equations.

One and two can be used to solve the problem. Three cannot be. For any degree 2 polynomial $p(x) = \beta_0 + \beta_1 x + \beta_2 x^2$ we can express the polynomail as $X_1 \vec{\beta}$ where $\vec{\beta} = [\beta_0, \beta_1, \beta_2]$. We can alternatively express the polynomial as $X_2 \vec{\tilde{\beta}}$ where $\vec{\tilde{\beta}} = [\beta_0, -\beta_1, \beta_2 - \beta_1]$. You can check this because $\beta_0 + (-\beta_1)(x^2 - x) + (\beta_2 - \beta_1)(x^2) = \beta_0 + \beta_1 x + \beta_2 x^2$.

On the other hand, for X_3, the third column is simply a scaled copy of the second (multiply by -2). As a consequence, using X_3 we can only express polynomials of the form $\alpha_0 + \alpha_1(2x^2 - x)$. In other words, the only polynomials $p(x) = \beta_0 + \beta_1 x + \beta_2 x^2$ we can express are those where $\beta_2 = 2\beta_1$.

3. Write each of the following models as transformed linear models. That is, find a parameter vector $\vec{\beta}$ in terms of the given parameters a_i and a set basis functions of functions $\phi(\vec{x})$ such that $y = \langle \vec{\beta}, \phi(\vec{x}) \rangle$. Also, show how to recover the original parameters a_i from the parameters β_j:

   (a) $y = (a_1 x_1 + a_2 x_2) e^{-x_1 - x_2}$.

   (b) $y = \begin{cases} a_1 + a_2 x & \text{if } x < 1 \\ a_3 + a_4 x & \text{if } x \geq 1 \end{cases}$

   (c) $y = (1 + a_1 x_1) e^{-x_2 + a_2}$.

(a) $y = a_1(x_1 e^{-x_1 - x_2}) + a_2(x_2 e^{-x_1 - x_2}))$. We can use basis functions $\phi_1(x_1, x_2) = x_1 e^{-x_1 - x_2}$ and $\phi_2(x_1, x_2) = x_2 e^{-x_1 - x_2}$. Once we obtain parameter vector $[\beta_1, \beta_2]$, we simply set $\alpha_1 = \beta_1$ and $\alpha_2 = \beta_2$.

(b) We can use basis functions are $\phi_1(x) = \mathbb{I}(x < 1)$, $\phi_2(x) = x \cdot \mathbb{I}(x < 1)$, $\phi_3(x) = \mathbb{I}(x \geq 1)$, and $\phi_4(x) = x \cdot \mathbb{I}(x \geq 1)$. Again if we use these basis function we can just set $\alpha_1, \alpha_2, \alpha_3, \alpha_4 = \beta_1, \beta_2, \beta_3, \beta_4$.

(c) $y = e^{a_2} e^{-x_2} + a_1 e^{a_2} x_1 e^{-x_2}$. We can use basis functions are $\phi_1(x_1, x_2) = e^{-x_2}$ and $\phi_2(x_1, x_2) = x_1 e^{-x_2}$. From parameters β_1, β_2, we set $\alpha_2 = \ln(\beta_2)$ and $\alpha_1 = \beta_2 / \alpha_2$.

4. For data with one predictor variable and one target: $(x_1, y_1), \ldots, (x_n, y_n)$, consider a simple linear regression model:

$$f_{\beta_0, \beta_1}(x) = \beta_0 + \beta_1 x \tag{2}$$

with a *logarithmically transformed* ℓ_2 loss:

$$L(\beta_0, \beta_1) = \sum_{i=1}^{n} (\log(y_i) - \log(f_{\beta_0, \beta_1}(x_i)))^2 \tag{3}$$

This sort of model makes sense when trying to predict a value that is more naturally expressed on a logarithmic scale (e.g. pH level, volume in decimals, etc.)

(a) Write down an expression for the gradient of the loss L.

Using chain rule we obtain.

$$\nabla L(\beta_0, \beta_1) = \begin{bmatrix} \sum_{i=1}^{n} -2(\log(y_i) - \log(f_{\beta_0, \beta_1}(x_i))) \cdot \frac{1}{\beta_0 + \beta_1 x_i} \\ \sum_{i=1}^{n} -2x_i(\log(y_i) - \log(f_{\beta_0, \beta_1}(x_i))) \cdot \frac{1}{\beta_0 + \beta_1 x_i} \end{bmatrix} \tag{4}$$

(b) Using your expression and the gradient descent update rule, write pseudocode for finding parameters β_0^*, β_1^* which approximately minimize L.

- $\vec{\beta} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ #choose an arbitrary starting point

- $\eta = .1$ #choose a step size. This might need to be tuned later.
- For $i = 1, \ldots, T$
  - Compute $\vec{y}_{pred} = \vec{\beta}[0] + \vec{\beta}[1]\vec{x}$.
  - Compute $\vec{g} = \begin{bmatrix} \sum_{i=1}^{n} -2(\log(\vec{y}[i]) - \log(\vec{y}_{pred}[i]) \cdot \frac{1}{\vec{y}_{pred}[i]} \\ \sum_{i=1}^{n} -2\vec{x}[i](\log(\vec{y}[i]) - \log(\vec{y}_{pred}[i]) \cdot \frac{1}{\vec{y}_{pred}[i]} \end{bmatrix}$
  - $\vec{\beta} \leftarrow \vec{\beta} - \eta \cdot \vec{g}$.

(c) What other method could have been used to find nearly optimal β_0^*, β_1^*?

Brute force search.