

Introduction to Machine Learning (Fall 2022)

Recitation attendance check

Type in your section passcode to get attendance credit (within the first fifteen minutes of your scheduled section).

Passcode:

100.00%

{0: 'games', 1: 'animal', 2: 'border', 3: 'call', 4: 'dark', 5: 'elder', 6: 'football', 7: 'girl'}

Logistic Regression

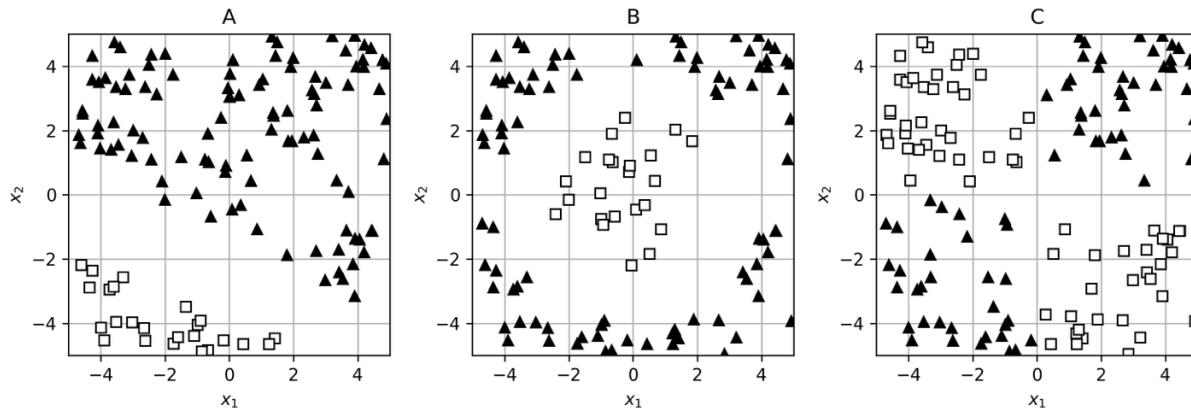
Due: Wednesday, October 05, 2022 at 11:00 PM

For this homework, it will be helpful to review the course notes on [Classification](#).

For your convenience, we have copied the sample test cases into a [colab notebook](#).

1) Linear Separability

Below we plot three data sets. The x_1 and x_2 axes are the features available for training. The different colors and shapes denote different classes. Data points marked with black triangles are labelled as "1"; data points marked with white squares are labelled as "-1". We want to train a model that can predict the class given the features.



1.1)

Which of these data sets are linearly separable? For those that are linearly separable, choose any vector θ and scalar θ_0 (written as $[[\theta_1, \theta_2], \theta_0]$) such that the classifier defined by $\text{sign}(\theta^T x + \theta_0)$ classifies every point correctly. For those that are not separable, choose "not separable".

A:

- [[2, 1], 0]
- [[1, 2], 4]
- not separable

100.00%*You have infinitely many submissions remaining.*

Solution: [[1, 2], 4]

Explanation:

One possible hyperplane that separates the data is oriented along the normal vector [1, 2] with an offset of 4. This corresponds to the line $x_2 = (-1/2) * x_1 - 2$. Two points on the boundary defined by this hyperplane are (-4, 0) and (4, -4).

B:

- [[1, 0], 0]
- [[0, 1], 0]
- not separable

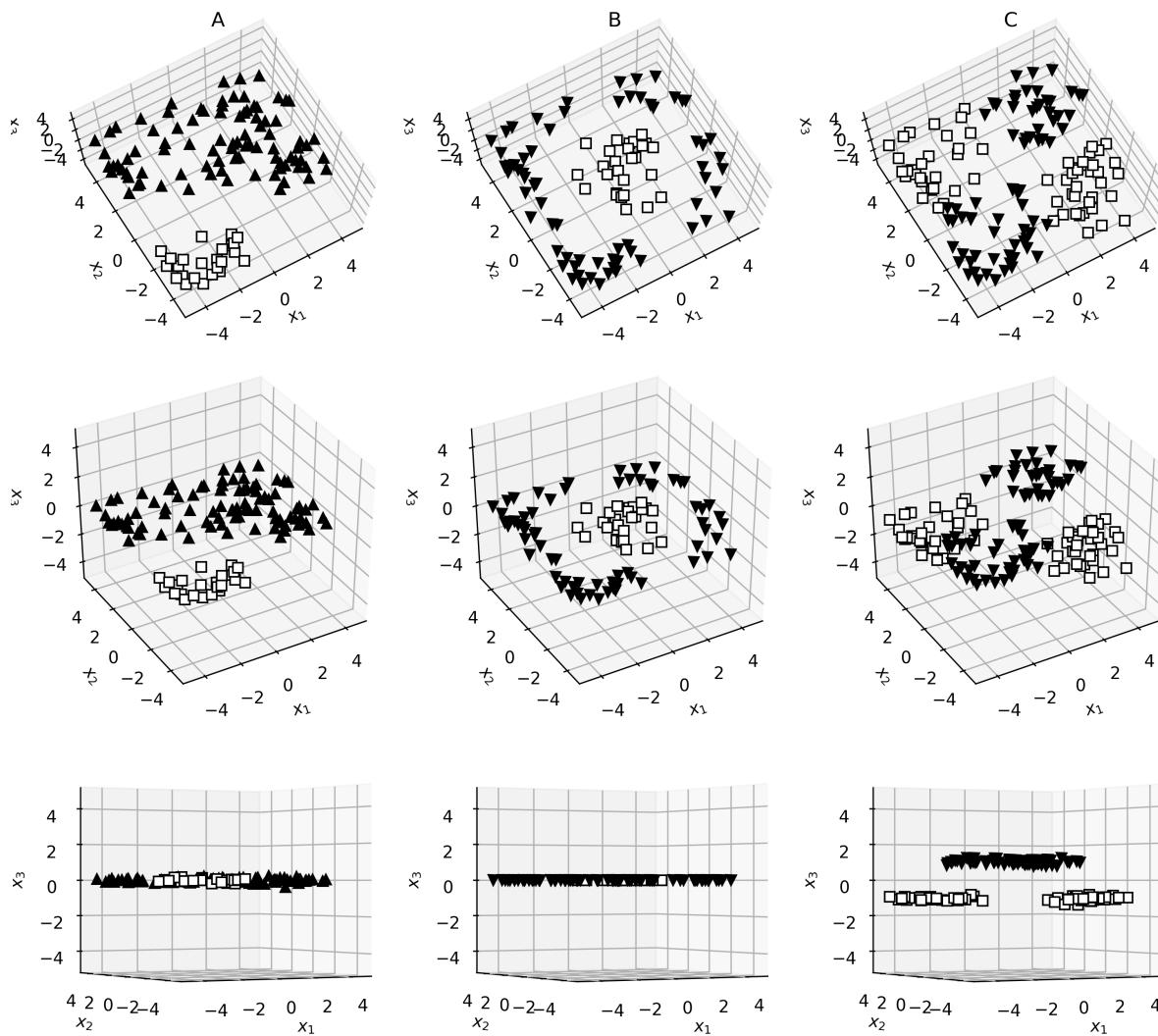
Submit**View Answer****100.00%***You have infinitely many submissions remaining.*

C:

- [[1, 0], 0]
- [[0, 1], 0]
- not separable

Submit**View Answer****100.00%***You have infinitely many submissions remaining.***1.2)**

The plots from the previous question had omitted one of the features available in the raw data. Below we plot all three features (in a 3D plot). Given that 3D plots shown in 2D can be confusing, each column plots the same data set viewed from different angles.



Given this new third feature, which of the data sets are linearly separable? For those that are, choose the vector θ and scalar θ_0 (written as $[\theta_1, \theta_2, \theta_3]$, θ_0) defining the classifier by $\text{sign}(\theta^T x + \theta_0)$. For those that are not separable, choose "not separable".

A:

- [[1, 2, 0], 0]
- [[1, 2, 0], 4]
- not separable

100.00%

You have infinitely many submissions remaining.

B:

- [[1, 0, 0], 0]
- [[0, 1, 0], 0]
- not separable

[Submit](#)[View Answer](#)**100.00%***You have infinitely many submissions remaining.*

C:

- [[0, 1, 0], 0]
- [[0, 0, 1], 0]
- not separable

100.00%*You have infinitely many submissions remaining.*

Solution: [[0, 0, 1], 0]

Explanation:

We can now separate the data along the x_3 feature. The classifier is $\text{sign}(x_3)$.

2) Simple Logistic Regression

We are interested in logistic regression for input vectors representing points in \mathbb{R}^d . We find a hypothesis of the form

$$y = \sigma(\theta^T x + \theta_0)$$

and, from it, derive a separator. Reminder: $\sigma(0) = 0.5$.

2.1)

What is the form of the **separator**?

- A d dimensional hyperplane
- A $d + 1$ dimensional hyperplane
- A $d - 1$ dimensional hyperplane

[Submit](#)[View Answer](#)**100.00%***You have infinitely many submissions remaining.*

2.2)

If $d = 1$, what is the separator?

- A point
- A line
- A plane

100.00%

You have infinitely many submissions remaining.

2.3)

For $d = 1$, what is the equation of the separator? Provide an expression in terms of theta and theta_0.

$x = \frac{-\theta_0}{\theta}$

100.00%

You have infinitely many submissions remaining.

2.4)

Here is a data set in 1D of the form $\{x^{(i)}, y^{(i)}\}$:

$$\{(1, 1), (2, 1), (4, 0), (5, 0)\}$$

2.4.1)

Is it linearly separable? Yes 

100.00%

You have infinitely many submissions remaining.

2.4.2)

Intuitively, it seems like the best separator for this data set is 3, because it is a separator and it maximizes the distance to the closest points. Give two different sets of finite-valued parameters (θ, θ_0) that result in a separator at 3 and correctly classify the points in the data set. Provide a list of lists.

100.00%

You have infinitely many submissions remaining.

2.4.3)

Are these parameters optimal according to \mathcal{L}_{nll} ? No

100.00%

You have infinitely many submissions remaining.

Solution: No

Explanation:

Recall that the negative-log likelihood (NLL) loss for a label $y \in \{0, 1\}$ and $g \in (0, 1)$ representing a predicted probability of the 1-class is

$$L_{\text{nll}}(g, y) = -[y \log g + (1 - y) \log(1 - g)].$$

Note that $L_{\text{nll}}(g, 1) = -\log g$ and $L_{\text{nll}}(g, 0) = -\log(1 - g)$, so minimizing the NLL-loss can be interpreted as maximizing the probability assigned to the label class y .

In logistic regression, $g(x) = \sigma(z(x))$, where

$$z(x) = \theta^T x + \theta_0.$$

In the case of this problem, $z(x)$ takes the form

$$z(x) = -\theta x + 3\theta,$$

where $\theta > 0$. Note that $-\theta x + 3\theta < 0$ (is assigned label 1) iff $x > 3$, so $z(x)$ perfectly separates the data. However, $z(x)$ is not optimal because **we can scale $z(x)$ up to get a new separator that achieves even lower NLL loss**. The separator

$$z'(x) = -2\theta x + 6\theta$$

will assign scores twice as positive as those of $z(x)$ to points labeled 1, and scores twice as negative as those of $z(x)$ to points labeled 0. Since sigmoid is monotonically increasing, this translates to assigning higher probabilities to points in the 1-class, and lower probabilities to points in the 0-class, achieving lower L_{nll} for every point in the dataset. Thus, there is no finite θ which achieves optimality (if there were, 2θ would do better).

2.5)

Recall that in logistic regression, $h(x; \theta, \theta_0) = \sigma(\theta^T x + \theta_0)$. Consider the following transformations on θ, θ_0 :

- (A) Multiply all parameters by 2
- (B) Multiply only θ_0 by 2
- (C) Multiply only θ by 2
- (D) Multiply all parameters by -2
- (E) Add 1 to θ_0
- (F) Subtract 1 from θ_0
- (G) None of the above

2.5.1)

Suppose that θ and θ_0 are non-zero. What transformation on θ, θ_0 makes the $h(x; \theta, \theta_0)$ function steeper but does not change which points $x \in R^d$ are classified as 0 and which points are classified 1?

100.00%

You have infinitely many submissions remaining.

Solution: A

Explanation:

The separator is defined by $\theta^T x + \theta_0 = 0$. We can represent the same separator with the equation $c\theta^T x + c\theta_0 = 0$ for any $c > 0$ (we don't let c be less than zero otherwise it would change the direction of the inequality for the classifier corresponding to this separator), since we can divide both sides of this equation by c and get the original equation. As we increase c , the input to h has a more positive value if $\theta^T x + \theta_0 > 0$ and a more negative value if $\theta^T x + \theta_0 < 0$. Because h is monotonically increasing, this means that inputs on the "+1" side of h will receive higher probabilities as we increase c , and inputs on the "-1" side of h will receive lower probabilities as we increase c . This makes h steeper without moving the separator. You can verify this behavior by comparing plots of $\sigma(z) = \frac{1}{1+e^{kz}}$ for different values of k . [This Desmos](#) may be helpful.

2.5.2)

What transformation on θ, θ_0 moves the separator to the left but does not change the steepness of $h(x; \theta, \theta_0)$?

100.00%

You have infinitely many submissions remaining.

Solution: G

Explanation:

We will explore this idea in detail below. Neither E nor F is completely correct because it depends on the sign of θ .

2.5.3)

If the separator classifies all the points correctly, what transformation on θ, θ_0 would decrease \mathcal{L}_{all} ?

100.00%

You have infinitely many submissions remaining.

2.6)

Consider a hypothesis θ, θ_0 in 1D. You want to move the separator in the positive direction by 1 while keeping the slope of the sigmoid the same. What are the updated values of the parameters you would use? Specify your answers in terms of θ, θ_0 .

Enter an expression for θ_{new} in terms of theta and theta_0.

$\theta_{new} = \boxed{\text{theta}}$

100.00%

You have infinitely many submissions remaining.

Enter an expression for $\theta_{0,new}$ in terms of theta and theta_0.

$\theta_{0,new} = \boxed{-\text{theta} + \text{theta_0}}$

100.00%

You have infinitely many submissions remaining.

Solution: $\text{theta_0} - \text{theta}$

$$\theta_0 - \theta$$

Explanation:

The equation of the separator in 1D in terms of x is given by $x = \frac{-\theta_0}{\theta}$. We want to move the separator to be at $x' = x + 1 = \frac{-\theta_0}{\theta} + 1 = \frac{-(\theta_0 - \theta)}{\theta} = \frac{(-\theta_0 + \theta)}{\theta}$. Recall that $\sigma(z + b)$ for any real-valued b shifts the sigmoid and does not change its slope. Decreasing θ_0 by θ is like setting $b = -\theta$, so we move the separator without changing the slope of the sigmoid.

3) Why Not Linear Regression?

We went to all this trouble to make a new loss function for classification. Was it necessary?

Here is another data set in 1D of the form $\{(x^{(i)}, y^{(i)})\}$:

$$\{(1, 0), (2, 0), (3, 1), (100, 1)\}$$

3.1)

Mark all of the following that are true about logistic regression on this data set:

- It will find a classifier with accuracy 1.0.
- The optimal parameters, if unregularized, are finite.
- The separator will be between 2 and 3.

100.00%

You have infinitely many submissions remaining.

Solution:

- It will find a classifier with accuracy 1.0.
- The optimal parameters, if unregularized, are finite.
- The separator will be between 2 and 3.

Explanation:

As discussed in the explanation to 2.4.3, one can find optimal parameters to a logistic regression by finding a perfect linear separator and scaling up the parameters of this separator. As the scale factor approaches infinity, the NLL loss approaches zero. Any separator that makes at least one classification error incurs NLL loss at least $-\log(1/2) = \log 2 > 0$, so if a perfect linear separator exists, any optimal logistic regression will use a perfect linear separator. In the case of this 1D dataset, the separator must lie between 2 and 3 to achieve perfect classification. Furthermore, any perfect linear separator with finite parameters θ, θ_0 suffers higher NLL loss than the separator with parameters $2\theta, 2\theta_0$, so no finite θ, θ_0 is optimal under NLL.

3.2)

We already know how to do linear regression! What if we treat this as a linear regression problem, with target y values of 0 and 1, and the objective of minimizing mean squared loss (instead of NLL)? We ran OLS regression on this data and got $\theta, \theta_0 = 0.007, 0.316$ with mean squared error (MSE) 0.163.

3.2.1)

What scalar value represents the separator (at $y=0.5$) corresponding to this hypothesis?

26.29

[Check Formatting](#)

[Submit](#)

[View Answer](#)

100.00%

You have infinitely many submissions remaining.

3.2.2)

If we predict 1 when the line y value is above 0.5 (and predict 0 otherwise), how many points do we predict correctly?

100.00%

You have infinitely many submissions remaining.

3.3)

Mark all of the following that are true:

- Logistic regression is able to accurately classify this data because the data are linearly separable.
- Linear regression has trouble classifying this data accurately because the distant point has a large influence on the hypothesis.
- Logistic regression doesn't work well on this data because the distant point is ignored.

100.00%

You have infinitely many submissions remaining.

4) Multi-class classification using binary classification

4.1) Part I: Confidence

First we're going to revisit binary classification. Suppose we have a classifier defined by parameters (θ, θ_0) . Recall that a separating hyperplane can be described as $\{x : \theta^T x + \theta_0 = 0\}$.

Assume within 4.1 that θ is not the zero vector. (But as you work through questions, do think about what happens if $\theta = 0$?) Also, in your answers within 4.1, you'll be writing out expression; you may use the following:

`x` for x

`theta` for θ

`theta_0` for θ_0

`max(a, b)` to indicate taking the max of a and b. Note: `max()` only accepts two values.

`abs()` to indicate an absolute value, e.g. `abs(a)` for the absolute value of a

`*` to indicate scalar multiplication

`@` to indicate matrix/vector multiplication

`transpose()` to indicate the transpose of a matrix/vector, e.g. `transpose(x)` for x^T

`sqrt(x)` for the non-negative square root of x (), e.g. `sqrt(4)=2`

4.1.1)

Write an expression for a vector g that is perpendicular to (a.k.a. normal to) the separating hyperplane defined above. Your vector should point in the direction of positive predictions and have length 1:

$$g = \frac{\theta}{\|\theta\|}$$

100.00%

You have infinitely many submissions remaining.

4.1.2)

Suppose we have some value $b \in \mathbb{R}$ with $b \neq 0$, and we draw a new hyperplane defined by $\theta^T x + \theta_0 = b$. Is this hyperplane parallel to the separator of the classifier?

Is this hyperplane parallel to the separator of the classifier?

- Yes
 No

100.00%

You have infinitely many submissions remaining.

Solution: Yes

Explanation:

If the two hyperplanes $\theta^T x + \theta_0 = b$ and $\theta^T x + \theta_0 = 0$ were not parallel, they would intersect at at least one point x . But then x satisfies $\theta^T x + \theta_0 = b$ and also $\theta^T x + \theta_0 = 0$. So $\theta^T x - b = \theta^T x$. Since θ is not the 0-vector, at least one element of this equation would imply $b = 0$, a contradiction. So the hyperplanes must be parallel.

4.1.3)

Take any $b \in \mathbb{R}$. Write an expression for the distance from the hyperplane defined by $\theta^T x + \theta_0 = b$ to the separator of the classifier.

Expression of the distance:

`abs(b)/sqrt(transpose(theta)@theta)`

[Check Syntax](#)

[Submit](#)

[View Answer](#)

100.00%

You have infinitely many submissions remaining.

4.1.4)

Let x be a point. Write an expression for the distance from x to the separator of the classifier.

`abs(transpose(theta)@x + theta_0)/sqrt(transpose(theta)@theta)`

[Check Syntax](#)

[Submit](#)

[View Answer](#)

100.00%

You have infinitely many submissions remaining.

4.1.5)

We now want to take a classifier and come up with a function for computing our "confidence" in its classification (+1 or -1) for a point x . We want our confidence to be a score that satisfies the following goals:

- For any point x , the confidence at x should be linearly proportional to the magnitude of the distance from x to the separating hyperplane.
- The confidence at any point in the training data should be between 0 and 1, scaled so that a training point that lies on the separating hyperplane has 0 confidence and the point in the training dataset that is farthest from the separating hyperplane has confidence ≤ 1 . Note that the confidence does not have to go all the way to 1 for a given dataset.

We will write our confidence as `conf(x; theta, theta0)`.

Assume that we're looking at a classifier that classifies at least one point as +1 and at least one point as -1. Which of the following choices could represent a confidence that satisfies the goals described above? Let R be the distance between the two points that are farthest from each other in the data set.

Which of the following choices could represent a confidence that satisfies the goals described above? Choose all that apply.

- $|\theta^T x + \theta_0|$
- $|\theta^T x + \theta_0|/(R||\theta||)$
- $|\theta^T x|/R$

[Submit](#)[View Answer](#)

100.00%

You have infinitely many submissions remaining.

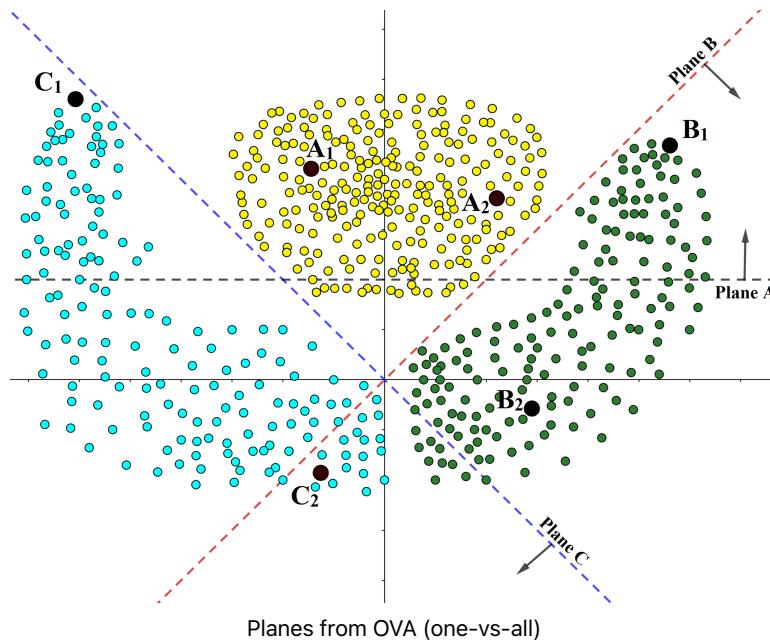
4.2) Part II:

Imagine that we have a dataset D where every point $x^{(i)}$ has a label $y^{(i)}$, which is one of k possible labels $\{Y^1, \dots, Y^k\}$. Let's consider two strategies: One-vs-All and One-vs-One.

4.2.1) One-vs-All (OVA)

Create k datasets. Let D_j denote the j th dataset. To create D_j , we include every data point from the original dataset -- but if a data point had label Y^j , it now has label +1. And if a data point had any other label, it now has label -1. For each dataset, we find a classifier. So we get k total classifiers. We'll call the set of these classifiers P_{ova} .

Below we show a dataset with 3 classes (A, B, C) where we have singled out some points ($A_1, A_2, B_1, B_2, C_1, C_2$). We also show the classifiers found by OVA. Note that each arrow points into the **positive** halfspace of its classifier; for example, for Plane A, the points in class A are (mostly) in the positive halfspace.



Next we describe the OVA multi-class classifier for a new point x .

$h_{ova}(x)$: Consider only the set of classifiers in P_{ova} that predict +1 for x . Each corresponds to a particular class j . Predict the class whose corresponding classifier has the largest confidence value (using a valid confidence value equation from above).

Select the points that will be classified as class A

A_1

A_2

B_1

B_2

C_1

C_2

100.00%

You have infinitely many submissions remaining.

Solution:

✓ A_1

✓ A_2

✓ B_1

✗ B_2

✓ C_1

✗ C_2

Select the points that will be classified as class B

- A_1
- A_2
- B_1
- B_2
- C_1
- C_2

100.00%

You have infinitely many submissions remaining.

Solution:

-  A_1
-  A_2
-  B_1
-  B_2
-  C_1
-  C_2

Select the points that will be classified as class C

- A_1
- A_2
- B_1
- B_2
- C_1
- C_2

100.00%

You have infinitely many submissions remaining.

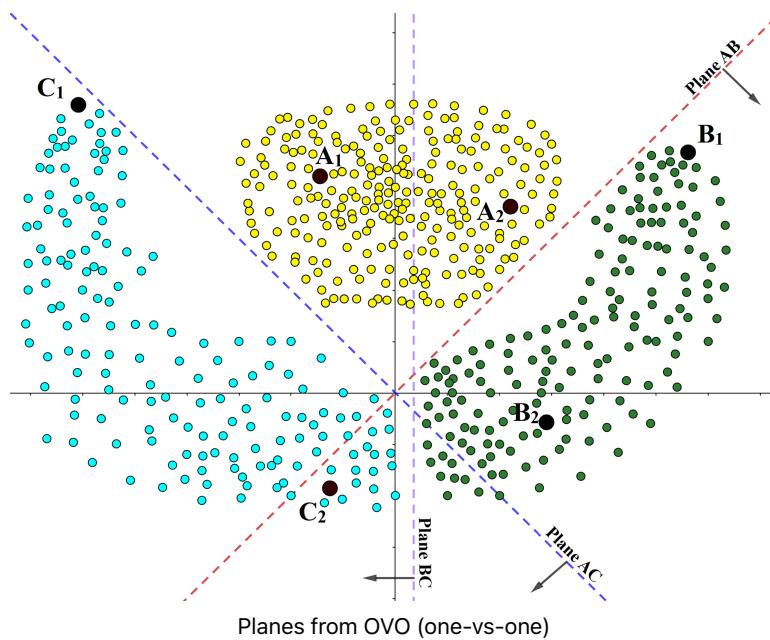
Solution:

- ✖ A_1
- ✖ A_2
- ✖ B_1
- ✖ B_2
- ✖ C_1
- ✓ C_2

4.2.2 One-vs-One (OVO)

We now create a new dataset for each (unordered) pair of distinct classes. Let $D_{\ell,m}$ denote the dataset for classes ℓ and m . So there are $k(k - 1)/2$ datasets in total. To create $D_{\ell,m}$, points from the original dataset with label Y^ℓ now have label -1; points from the original dataset with label Y^m now have label +1; and points from the original dataset with any other label do not appear in $D_{\ell,m}$. For each dataset, we find a classifier. So we get $k(k - 1)/2$ total classifiers. We'll call the set of these classifiers P_{ovo} .

Below we show the same dataset with 3 classes (A, B, C) where we have singled out some points ($A_1, A_2, B_1, B_2, C_1, C_2$) with the classifiers found by OVO. For the classifier with separating hyperplane "Plane ℓm ", the arrow is shown pointing into the halfspace where class m is predicted. E.g., for Plane AC , the arrow points toward the halfspace classified as class C , and the other halfspace would be classified as class A .



Next we describe the OVO multi-class classifier for a new point x .

$h_{ovo}(x)$: Consider the classifier between classes ℓ and m . If this classifier predicts class ℓ at x , it votes for ℓ . If it predicts class m , it votes for m . At the end, we predict the class with the most votes overall across all the classes.

Select the points that will be classified as class A

- A_1
- A_2
- B_1
- B_2
- C_1
- C_2

[Submit](#)

[View Answer](#)

100.00%

You have infinitely many submissions remaining.

Select the points that will be classified as class B

- A_1
- A_2
- B_1
- B_2
- C_1
- C_2

[Submit](#)

[View Answer](#)

100.00%

You have infinitely many submissions remaining.

Select the points that will be classified as class C

- A_1
- A_2
- B_1
- B_2
- C_1
- C_2

[Submit](#)

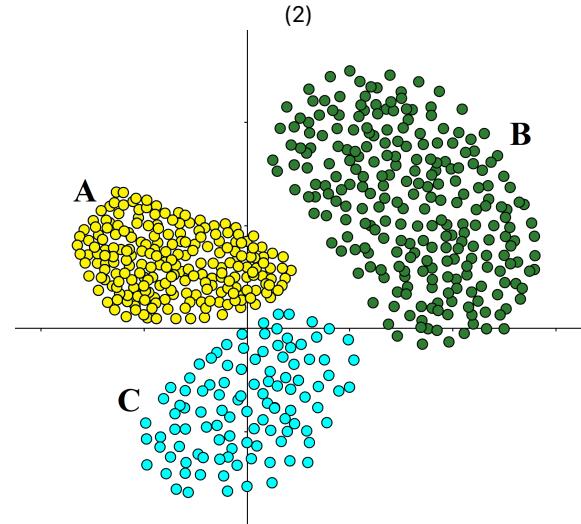
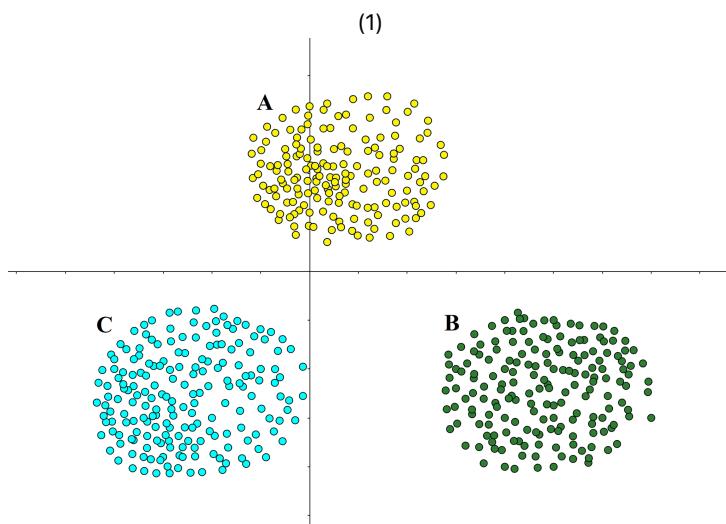
[View Answer](#)

100.00%

You have infinitely many submissions remaining.

4.3) Part III:

We look at two new datasets below. For each dataset, imagine running OVA and OVO with optimal binary classifiers along the way. Indicate which hypothesis would have lower error -- or if the two would be tied



4.3.1)

Which hypothesis has lower error for dataset 1?

- $h_{ova}(x)$
- $h_{ovo}(x)$
- tied

[Submit](#)

[View Answer](#)

100.00%

You have infinitely many submissions remaining.

4.3.2)

Which hypothesis has lower error for dataset 2?

- $h_{ova}(x)$
- $h_{ovo}(x)$
- tied

100.00%

You have infinitely many submissions remaining.

5) Deriving Me Crazy

Our eventual goal is to do gradient descent on the logistic regression objective J_{nll} .

In this problem, we'll take the first step toward deriving that gradient update. We'll focus on the gradient of the loss at a single point with respect to parameters θ and θ_0 .

5.1)

What is an expression for the derivative of the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ with respect to z , expressed as a function of z , its input? Enter a Python expression (use `**` for exponentiation) involving `e` and `z`.

$$\frac{\partial \sigma(z)}{\partial z} = \boxed{e^{**(-z)}/(1+e^{**(-z)})**2}$$

100.00%

You have infinitely many submissions remaining.

5.2)

What is an expression for the derivative of the sigmoid with respect to z , but this time expressed as a function of $o = \sigma(z) = \frac{1}{1+e^{-z}}$? (It's beautifully simple!)

Hint: Think about the expression $1 - \frac{1}{1+e^{-z}}$. (Here is a [review](#) of computing derivatives.)

Enter a Python expression (use `**` for exponentiation) involving only `o`. `e` and `z` are not allowed, and remember $o = \sigma(z)$.

$$\frac{\partial \sigma(z)}{\partial z} = \boxed{o*(1-o)}$$

100.00%

You have infinitely many submissions remaining.

5.3)

5.3.1)

What is the maximum value of $\frac{\partial \sigma(z)}{\partial z}$?

100.00%

You have infinitely many submissions remaining.

5.3.2)

What is the largest number that is always less than any actual value of $\frac{\partial \sigma(z)}{\partial z}$?

100.00%

You have infinitely many submissions remaining.

5.4)

Given the output of the model

$$g = \sigma(\theta^T x + \theta_0) = \frac{1}{1 + e^{-(\theta^T x + \theta_0)}}$$

what is the derivative of g with respect to θ ? Enter a Python expression involving g and x .

Hint: Use chain rule and the expression you found for the derivative of the sigmoid in part 5.2

$\frac{\partial g}{\partial \theta} = [g**2*(1/g-1)*x]$

100.00%

You have infinitely many submissions remaining.

5.5)

The loss, $L_{\text{nll}}(g, y)$ is defined as:

$$L_{\text{nll}}(g, y) = -\left(y \log g + (1 - y) \log(1 - g)\right)$$

What is the derivative of the loss with respect to g ? Enter a Python expression involving y and g .

$\frac{\partial L_{\text{nll}}}{\partial g} = [(1-y)/(1-g) - y/g]$

100.00%

You have infinitely many submissions remaining.

5.6)

What is the derivative of L_{nll} with respect to θ ? Enter a Python expression involving x , y , and g .

Hint: Use the chain rule and your expression from 5.3

$\frac{\partial L_{\text{nll}}}{\partial \theta} = [(1-y)/(1-g) - y/g] * (g**2*(1/g-1)*x)$

100.00%

You have infinitely many submissions remaining.

5.7)

What is the derivative of L_{nll} with respect to θ_0 ? Enter a Python expression involving x , y , and g .

$$\frac{\partial L_{\text{ml}}}{\partial \theta_0} = \frac{((1-y)/(1-g) - y/g) * (g**2*(1/g-1))}{\theta_0}$$

100.00%

You have infinitely many submissions remaining.

6) Multi-class Logistic Regression

You might want to solve using Python and numpy, or use the [colab file](#) for the following questions.

6.1)

Assume we are doing multi-class logistic regression with three possible categories. What probability distribution over the categories is represented by $z = [-1, 0, 1]^T$, where z is the vector of inputs to the softmax transformation?

Enter a distribution (a list of three non-negative numbers adding up to 1) for the three categories. Your answers should be numeric (please enter numbers, and do not use the symbol e):

100.00%

You have infinitely many submissions remaining.

6.2)

If our output $g = [.3, .5, .2]^T$ and $y = [0, 0, 1]^T$, what is $\mathcal{L}_{\text{NLLM}}(g, y)$ on this one point?

Enter an expression involving `log(.)` (for natural log) and constants:

100.00%

You have infinitely many submissions remaining.

6.2.1)

Suppose we are doing multi-class logistic regression with input dimension $d = 2$ and number of classes $K = 3$. Let the parameter matrix be

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

Assume $\theta_0 = [0, 0, 0]^T$, the input $x = [1, 1]^T$, and the target output $y = [0, 1, 0]^T$. These are numpied for your convenience below:

```
th = np.array([[1, -1, -2], [-1, 2, 1]])
x = np.array([[1, 1]]).T
y = np.array([[0, 1, 0]]).T
```

What is the predicted probability that x is in class 1, before any gradient updates? (Assume we have classes 0, 1, and 2.)

Enter a number (to at least 3 decimal places):

100.00%

You have infinitely many submissions remaining.

Solution: **0.665**

Explanation:

```
th = np.array([[1, -1, -2], [-1, 2, 1]])
x = np.array([[1, 1]]).T
y = np.array([[0, 1, 0]]).T

def softmax(z):
    return np.exp(z) / np.sum(np.exp(z))

z = th.T @ x
g = softmax(z)
print(g[1])
```

6.2.2)

To do gradient descent, we need to know $\nabla_{\theta} \mathcal{L}_{\text{NLLM}}(g, y)$. We will postpone doing the derivation, but just to let you know, it has an awesome form:

$$\nabla_{\theta} \mathcal{L}_{\text{NLLM}}(g, y) = x(g - y)^T$$

(Check the dimensions yourself to be sure that it's sensible.)

If you don't want to think about the whole matrix of partial derivatives at once, we can write the partial derivative with respect to a single component (i, j) of the parameter matrix:

$$\frac{\partial}{\partial \theta_{ij}} \mathcal{L}_{\text{NLLM}}(g, y) = x_j(g_i - y_i)$$

For the example we have developed in this question, what is the numeric value of the matrix $\nabla_{\theta} \mathcal{L}_{\text{NLLM}}(g, y)$?

Hint: You might want to solve using Python and numpy, or using the [colab notebook which may be found here](#).

Enter the matrix as a list of lists, one list for each row of the matrix. Please enter values with a precision of three decimal points.

[[0.24472847, -0.33475904, 0.09003057], [0.24472847, -0.33475904, 0.09003057]]

100.00%

You have infinitely many submissions remaining.

Solution: [[0.24472847, -0.33475904, 0.09003057], [0.24472847, -0.33475904, 0.09003057]]

Explanation:

```
dL = x @ (g - y).T
print(dL.tolist())
```

6.3)

Using a step size of 0.5, what is θ after one gradient update step?

Enter the matrix as a list of lists, one list for each row of the matrix. Please enter values with at least precision of three decimal points:

[[0.87763576, -0.83262048, -2.04501529], [-1.12236424, 2.16737952, 0.95498471]]

Check Formatting

Submit

View Answer

100.00%

You have infinitely many submissions remaining.

6.4)

What is the predicted probability that x is in class 1, given the new weight matrix?

Enter a number: 0.772

Check Formatting

Submit

View Answer

100.00%

You have infinitely many submissions remaining.

7) Applying gradient descent to Linear Logistic Classifier objective

Last week we implemented gradient descent in the general case, so now we will apply it to negative log-likelihood (NLL). Our goal in this section will be to derive and implement appropriate gradient calculations that we can use with gd for optimization of the LLC objective. In the derivations below, we'll consider linear binary classifiers with offset; i.e., our collection of parameters is θ, θ_0 .

Recall that NLL loss for binary classification is defined as:

$$L_{\text{nll}}(g, y) = -\left(y \log g + (1 - y) \log(1 - g)\right)$$

The objective function for linear logistic classification (LLC), a.k.a. logistic regression (LR), takes the mean of the NLL loss over all points and introduces a regularization term to this equation to make sure that the magnitude of θ stays small.

$$J_{lr}(\theta, \theta_0) = \left[\frac{1}{n} \sum_{i=1}^n L_{\text{nll}}(\sigma(\theta \cdot x^{(i)} + \theta_0), y^{(i)}) \right] + \lambda \|\theta\|^2$$

We're interested in applying our gradient descent procedure to this function in order to find the 'best' separator for our data, where 'best' is measured by the lowest possible LLC objective.

7.1) Calculating the LLC objective

First, implement the sigmoid function and implement NLL loss over the data points and separator. Using the latter function, implement the LLC objective. Note that these functions should work for matrix/vector arguments, so that we can compute the objective for a whole dataset with one call.

Note that we're going to let X represent the full set of features across all the data points and y represent the full set of labels across all the data points. So X is $d \times n$, y is $1 \times n$, θ is $d \times 1$, θ_0 is 1×1 , λ is a scalar.

Hint: Look at `np.exp`, `np.log`

In the test cases for this problem, we'll use the following `super_simple_separable` test dataset and `sep_e_separator` test separator. A couple of the test cases are also shown below.

```
def super_simple_separable():
    X = np.array([[2, 3, 9, 12],
                  [5, 2, 6, 5]])
    y = np.array([[1, 0, 1, 0]])
    return X, y

sep_e_separator = np.array([-0.40338351, 1.1849563]), np.array([-2.26910091])
```

Test case 1

```
x_1, y_1 = super_simple_separable()
th1, th1_0 = sep_e_separator
ans = llc_obj(x_1, y_1, th1, th1_0, .1)
```

Test case 2

```
ans = llc_obj(x_1, y_1, th1, th1_0, 0.0)
```

Note: In this section, you will code many individual functions, each of which depends on previous ones. We **strongly recommend** that you test each of the components on your own to debug.

Please use `np.sum` to take the sum of a matrix if needed.

```

1 # returns a vector of the same shape as z
2 def sigmoid(z):
3     pass
4
5 # X is dxn, y is 1xn, th is dx1, th0 is 1x1
6
7 # returns a (1,n) array for the nll loss for each data point given th and th0
8 import math
9 def sigmoid(z):
10     return 1/(1+math.e**(-z))
11 def nll_loss(X, y, th, th0):
12     func = th.T@X + th0

```

100.00%

You have infinitely many submissions remaining.

7.2) Calculating the gradients

Define a function `llc_obj_grad` that returns the gradient of the LLC objective function with respect to θ and θ_0 in a single column vector. The last component of the gradient vector should be the partial derivative with respect to θ_0 . Look at `np.vstack` as a simple way of stacking two matrices/vectors vertically. We have broken it down into pieces that mimic steps in the chain rule; this leads to code that is a bit inefficient but easier to write and debug. We can worry about efficiency later.

Some test cases that may be of use are shown below:

Inputs to Test Cases

```

X1 = np.array([[1, 2, 3, 9, 10]])
y1 = np.array([[1, 1, 1, 0, 0]])
th1, th10 = np.array([-0.31202807]), np.array([1.834])
X2 = np.array([[2, 3, 9, 12],
               [5, 2, 6, 5]])
y2 = np.array([[1, 0, 1, 0]])
th2, th20=np.array([-3., 15.]).T, np.array([2.])

```

d_sigmoid Tests

Test Case 1:
`d_sigmoid(np.array([[71.]]).tolist()`

Test Case 2:
`d_sigmoid(np.array([[-23.]]).tolist()`

Test Case 3:
`d_sigmoid(np.array([[71, -23]]).tolist()`

d_nll_loss_th Tests

Test Case 4:
`d_nll_loss_th(X2[:,0:1], y2[:,0:1], th2, th20).tolist()`

Test Case 5:
`d_nll_loss_th(X2, y2, th2, th20).tolist()`

d_nll_loss_th0 Tests

Test Case 6:
`d_nll_loss_th0(X2[:,0:1], y2[:,0:1], th2, th20).tolist()`

Test Case 7:
`d_nll_loss_th0(X2, y2, th2, th20).tolist()`

d_llc_obj_th Tests

Test Case 8:
`d_llc_obj_th(X2[:,0:1], y2[:,0:1], th2, th20, 0.01).tolist()`

Test Case 9:
`d_llc_obj_th(X2, y2, th2, th20, 0.01).tolist()`

d_llc_obj_th0 Tests

Test Case 10:
`d_llc_obj_th0(X2[:,0:1], y2[:,0:1], th2, th20, 0.01).tolist()`

Test Case 11:
`d_llc_obj_th0(X2, y2, th2, th20, 0.01).tolist()`

llc_obj_grad Tests

Test Case 12:
`llc_obj_grad(X2, y2, th2, th20, 0.01).tolist()`

Test Case 13:
`llc_obj_grad(X2[:,0:1], y2[:,0:1], th2, th20, 0.01).tolist()`

Note: In this section, you will code many individual functions, each of which depends on previous ones. We **strongly recommend** that you test each of the components on your own to debug.

Hint: Make sure to fully simplify the gradients in your implementation!

If using `np.sum` or `np.mean`, the optional `keepdims` argument (see [documentation](#)) preserves the dimension of the output matrix.

```

1 # returns an array of the same shape as z for the gradient of sigmoid(z)
2 import math
3 def sigmoid(z):
4     return 1/(1+math.e**(-z))
5 def d_sigmoid(z):
6     return 1/(1+math.e**(-z))**2*(math.e**(-z))
7
8 # returns a (d,n) array for the gradient of nll_loss(X, y, th, th0) with respect to th for each dat
9
10 def d_nll_loss_th(X, y, th, th0):
11     g = sigmoid(th.T@X + th0)
12     return np.multiply(X, (g-y))

```

100.00%

You have infinitely many submissions remaining.

7.3) LLC minimize

Putting it all together, use the functions you built earlier to write a gradient descent minimizer for the LLC objective. You do not need to paste in your previous definitions; you can just call the ones you've defined above. You will need to call gd the gradient descent function which you implemented in HW 3; your function llc_min should return the values that gd does. We have provided you with a sequence of step sizes already below.

- Initialize all the separator parameters to zero,
- use the step size function provided below, and
- specify 10 iterations.

Hint: the f that we feed into gd can only have a single column vector as its parameter; however, to call the objective function you've written, you need both theta and theta_0. Think of a way to pass both of them into f and then unpack them to call the objective function. Look back at the structure of what llc_obj_grad returns in the previous problem.

```

1 def llc_min(data, labels, lam):
2     """
3
4     Parameters:
5         data: dxn
6         labels: 1xn
7         lam: scalar
8
9     Returns:
10        same output as gd
11    """
12
13    def llc_min_step_size_fn(i):
14        return 2/(i+1)**0.5
15    return np.array([[-1.897383932746237], [3.3962308285238088], [-0.3185808665118801], 0.304061]

```

0.00%

You have infinitely many submissions remaining.

Some test cases are shown below, where an additional separable test dataset has been specified.

```

def separable_medium():
    X = np.array([[2, -1, 1, 1],
                  [-2, 2, 2, -1]])
    y = np.array([[1, 0, 1, 0]])

```

```
return X, y
sep_m_separator = np.array([[ 2.69231855], [ 0.67624906]]), np.array([[-3.02402521]])
```

Test Case 1:

```
x_1, y_1 = super_simple_separable()
ans = package_ans(llc_min(x_1, y_1, 0.0001))
```

Test Case 2:

```
x_1, y_1 = separable_medium()
ans = package_ans(llc_min(x_1, y_1, 0.0001))
```

8) Vectorized Random Binary Classifier

Let's implement the random linear classifier from the notes. But rather than use the `for` loop in the notes, we will be able to evaluate our random hyperplanes all at once. The process of using matrix algebra to do things we would otherwise express with loops is called "vectorization" and helps make our algorithms (for more complicated models) run fast on GPUs. Vectorization will become increasingly important as our models become more complex, as we will see when we start studying Neural Networks in Week 6.

We will implement a binary linear classifier. For any single data point's feature vector x , we have:

$$h(x; \theta, \theta_0) = \begin{cases} 1 & \text{if } \theta^T x + \theta_0 > 0 \\ -1 & \text{if } \theta^T x + \theta_0 \leq 0 \end{cases}$$

8.1) Dim check!

First, let's check our dimensions. For the next 4 questions, assume the number of features is $d = 5$, the number of examples is $n = 7$, and the number of outputs is $k = 1$.

We're now going to have x collect all of the features for all of our data points (instead of representing just a single data point); see 7.1 for the last time we collected all of the dataset features in one place. What is the shape of x ? Submit your answer as a python list.

[Check Formatting](#)

[Submit](#)

[View Answer](#)

You have infinitely many submissions remaining.

What is the shape of θ ? Submit your answer as a python list.

[Check Formatting](#)

[Submit](#)

[View Answer](#)

You have infinitely many submissions remaining.

What is the shape of $\theta^T x$? Submit your answer as a python list.

[Check Formatting](#)

[Submit](#)

[View Answer](#)

You have infinitely many submissions remaining.

What is the shape of θ_0 ? Submit your answer as a python list.

[Check Formatting](#)[Submit](#)[View Answer](#)

You have infinitely many submissions remaining.

8.2) Things are getting tensor

First, let's remind ourselves how to implement a binary linear classifier as given by this equation.

```
def binary_classifier(x, theta, theta0):
    y = theta.T @ x + theta0
    return np.where(y>0, 1, -1)
```

Now, we want to assess k binary classifiers on our data x at once without using a `for` loop like in the notes. There is more than one way to do this. One simple way is to prepend a third index to our model parameters, making them tensors instead of just matrices. (The tensors must flow!) This will require us to rewrite our function. Again, there is more than one way to do this.

In the [Matrix Derivative Notes](#), we introduced "implicit summation" notation, also known as "Einstein summation notation". This is a very helpful technique especially when you have tensors with many indices. Here's how we can "vectorize" our binary classifier to apply multiple binary classifiers at once using Einstein summation notation with `np.einsum`.

```
def vectorized_binary_classifier(x, theta, theta0):
    """
    n = num of examples, d = dimensions of features, k = model index
    input before -> now vectorized!
    x      [d, n] -> [d, n] (no change here since we are vectorizing over models)
    theta  [d, 1] -> [k, d, 1]
    theta0 [1, 1] -> [k, 1, 1]
```

We will use the letter 'b' below to indicate our binary output index.

```
"""
# Because the 'd' index is repeated and does not appear in the final 'kbn',
# it is summed over just like we do with our regular matrix multiply @.
y = np.einsum('kdb,dn->kbn', theta, x) + theta0
return np.where(y>0, 1, -1)
```

One of the many nice things about `np.einsum` is that it handles index transposes for you!

We are also going to implement a function that takes `theta` and `theta0` and returns the "slope" and "intercept" of our hyperplane. This is a helpful function for plotting!

```
def slope_intercept(theta, theta0):
    return (-theta[:-1] / theta[-1], theta0/ theta[-1])
```

8.2.1)

Now, let's create and plot randomly-generated classifiers and our train and test set. We'll start with the number of hyperplanes $k = 5$. See what happens when you increase the number of hyperplanes (e.g. $k = 10, 20, 50$).

```

1 def run():
2     return visualize_random_classifiers(k=5)
3

```

[Run Code](#)[Submit](#)

You have infinitely many submissions remaining.

8.2.2)

Next, we'll make a new classifier by choosing the best performing randomly-generated classifier from among those we've generated so far.

Consider the new classifier that takes the best randomly-generated classifier so far. Imagine we increase the number of hyperplanes in this classifier by always keeping all of the hyperplanes we've generated so far and then adding new hyperplanes. How does the new classifier's performance improve as we increase the number of randomly-generated classifiers?

- The new classifier is random, so we can't say if the performance will generally improve or not as we create more randomly-generated classifiers.
- Performance generally improves, but the improvement might not be monotonic (i.e., performance might go down sometimes but generally trends up)
- Performance monotonically improves (might stay the same)
- Performance is guaranteed to improve by a non-zero amount every time we increase the number of randomly-generated classifiers

[Submit](#)[View Answer](#)

You have infinitely many submissions remaining.

8.3) How much are we losing?

Now, we're going to evaluate the loss. Given the model output $h(x)$ and labels y , we will use the average 0-1 loss across the dataset here to evaluate our binary classifier:

$$E_n(h) = \frac{1}{n} \sum_{i=1}^n \begin{cases} 1 & h(x^{(i)}) \neq y^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

Write a function, `binary_loss`, that has the following input arguments:

- `output`: the output of the random classifier ($k \times 1 \times n$)
- `labels`: the true labels of the data points, copied k times ($k \times 1 \times n$)

Your function should return a $k \times 1$ matrix, the loss calculated for each of the k classifiers.

To write this function, you may find the numpy function `np.count_nonzero` or `np.mean` useful.

```
1 def binary_loss(output, labels):  
2     pass  
3
```

You have infinitely many submissions remaining.

8.4) Trying things many times

Now let's plot the mean and min error across our randomly-generated classifiers as a function of the number of random hypotheses generated so far. We're going to do this for a few different sequences of randomly-generated classifiers, so we can see what changes across these sequences. That's why we will see multiple plotted points for each particular number of randomly-generated classifiers. Note that when we use our `binary_loss` function, we will need to prepend a new axis to our original labels $[1, n] \rightarrow [1, 1, n]$, so our expression can broadcast the appropriate dimensions. Run the following code block and inspect the plots generated.

```
1 def run():  
2     return plot_loss()  
3
```

You have infinitely many submissions remaining.

8.4.1)

What do you notice about the mean of the error calculated across all of the randomly-generated classifiers?

As the number of randomly-generated classifiers increases, the mean error

- Decreases non-monotonically
- Decreases monotonically
- Fluctuates between 0.35 and 0.65
- Converges to 0.5

[Submit](#)

[View Answer](#)

You have infinitely many submissions remaining.

8.4.2)

Which of these statements most accurately reflects what's going on with the minimum test and train errors as a function of the number of randomly-generated classifiers?

As the number of randomly-generated classifiers increases,

- The minimum test and train errors converge to the same value
- The minimum train error converges to a lower value than the minimum test error .
- There is large uncertainty on the accuracy on the test data.

[Submit](#)

[View Answer](#)

You have infinitely many submissions remaining.