Introduction to Machine Learning

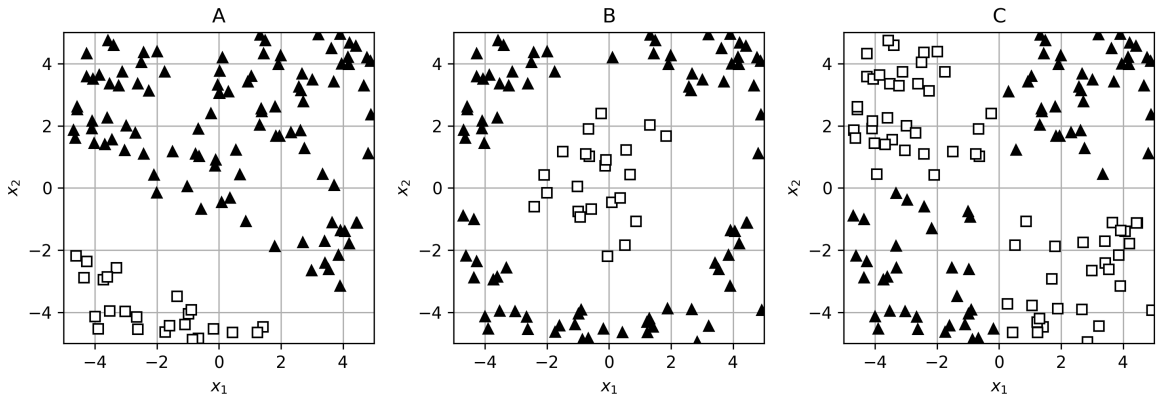
Logistic Regression

Due: Wednesday, March 06, 2024 at 11:00 PM

For this homework, it will be helpful to review the course notes on Classification.

1) Linear Separability

Below we plot three data sets. The x_1 and x_2 axes are the features available for training. 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 [[θ_1, θ_2], θ_0]) such that the classifier defined by $sign(\theta^T x + \theta_0)$ classifies every point correctly. For those that are not separable, choose "not separable".

A:

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

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

B:

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

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

C:

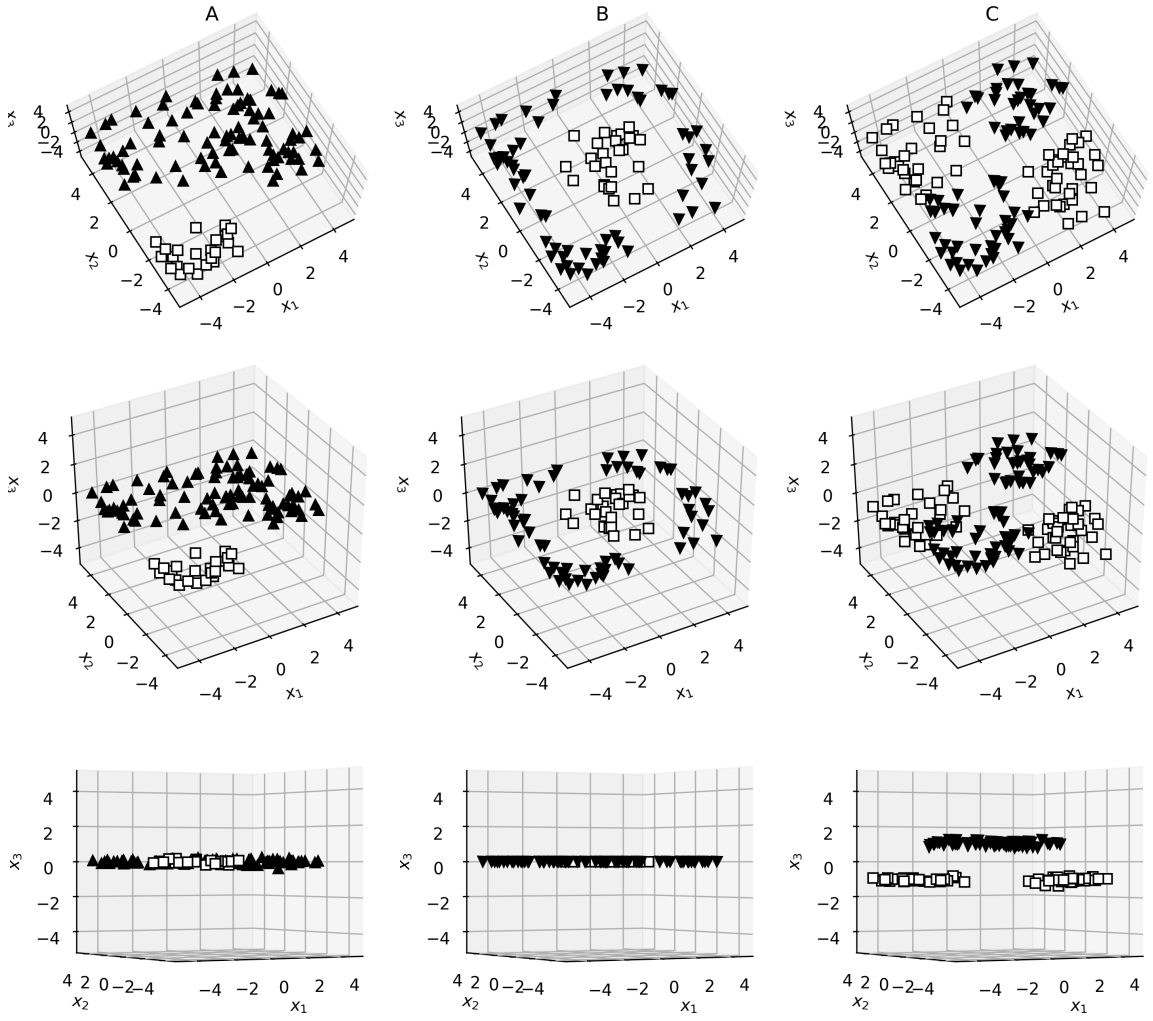
- [[1, 0], 0]
- [[0, 1], 0]
- not separable

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 [[θ_1, θ_2, θ_3], θ_0]) defining the classifier by $sign(\theta^T x + \theta_0)$. For those that are not separable, choose "not separable".

A:

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

Submit View Answer **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

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

2) Simple Linear Logistic Regression

We are interested in linear 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

Submit | View Answer **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`. That is, we're looking for an expression of x in terms of θ and θ_0, such that $\sigma(\theta x + \theta_0) = 0.5$.

$x =$ -theta_0/theta

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2.4)

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

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

(It might be helpful to draw out these data points for this problem.)

2.4.1)

Is it linearly separable? Yes

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2.4.2)

Intuitively, it seems like $x = 3$ might be a good separator for this data set, because it is a separator and it maximizes the distance to the closest points. Provide two different pairs of values for (θ, θ_0), such that in both cases, $\sigma(\theta x + \theta_0)$ models the desired linear separator at $x = 3$, and **correctly classifies** the data set (i.e. positive-labeled data is classified as positive, and vise-versa). Enter your answer as a list of two lists, where each (sub)list is a $[\theta, \theta_0]$ pair. [[-2, 6],[-1, 3]]

Check Formatting | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2.4.3)

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

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2.5)

Recall that in linear 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? A

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2.5.2)

Consider the case where $d = 2$. What transformation on θ, θ_0 moves the separator to the left but does not change the steepness of $h(x; \theta, \theta_0)$? (If you have trouble thinking about this, you may find it helpful to take a look at the Desmos demo linked in the previous question explanation for 1D intuition.) G

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2.5.3)

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

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2.6)

Consider a hypothesis $h(x) = \sigma(\theta x + \theta_0)$ 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. (You may find question 2.3 helpful for this question.)

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

θ_{new} = theta

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

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

$\theta_{0_{new}}$ = theta_0 - theta

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

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:

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

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

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? That is, at what x value would our hypothesis predict a y value of 0.5? (0.5-0.316)/0.007

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, out of four, do we predict correctly?

3

Check Formatting | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

3.3)

Mark all of the following that are true:

- [x] Logistic regression is able to accurately classify this data because the data are linearly separable.
- [x] 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.

Submit | View Answer **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 formulaic expressions; 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:

$g =$ theta/sqrt(transpose(theta)@theta)

Check Syntax | Submit | View Answer **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

Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

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.

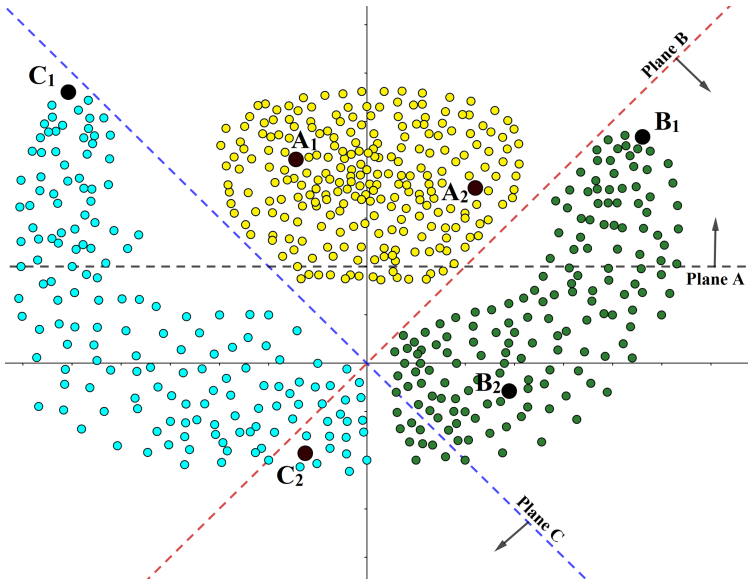
4.2) Part II:

Imagine that we have a dataset D where every point $x^{(i)}$ has a label $y^{(i)}$, which is one of k possible labels $\{Y^1, \ldots, Y^k\}$. 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 jth 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.



Planes from OVA (one-vs-all)

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

- [x] A_1
- [x] A_2
- [x] B_1
- [ ] B_2
- [x] C_1
- [ ] C_2

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

Among the points that will be classified as class A, which data point is the classifier most confident about?

- ○ 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
- [x] 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
- [x] C_2

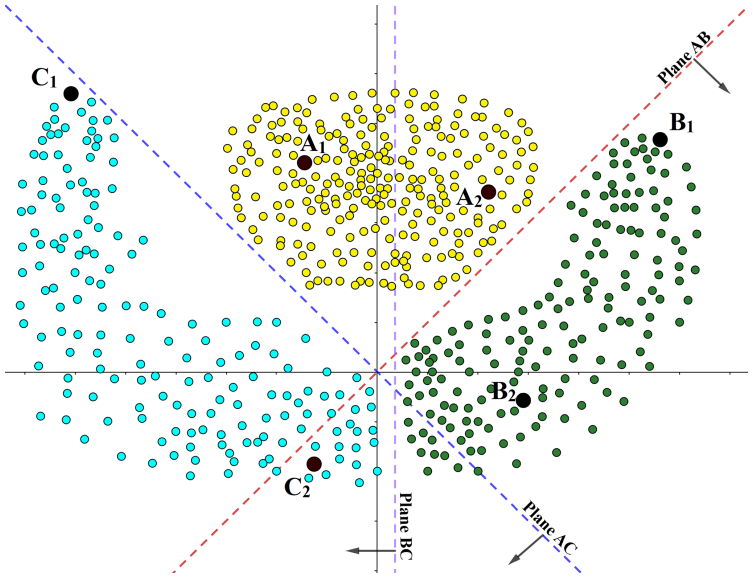
Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

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 $D_{l,m}$, 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 $D_{l,m}$. 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.



Planes from OVO (one-vs-one)

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

- [x] A_1
- [x] 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
- [x] B_1
- [x] 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
- [x] C_1
- [x] C_2

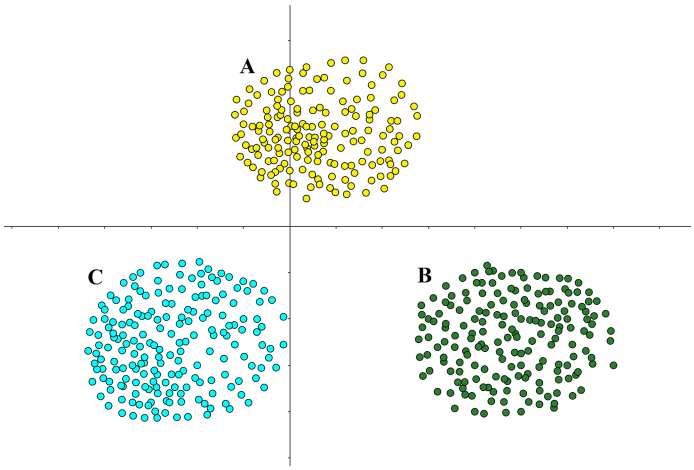
Submit View Answer **100.00%**

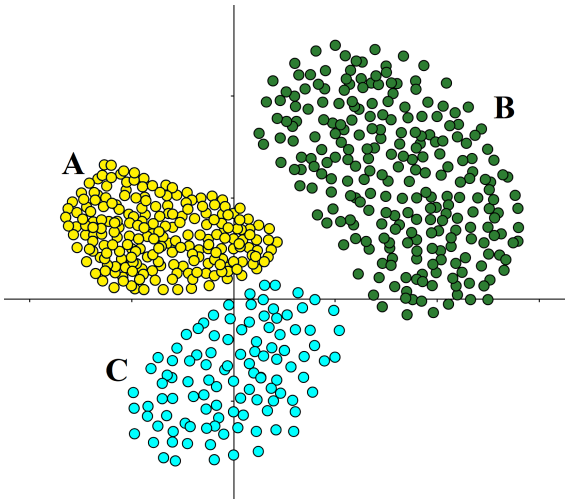
You have infinitely many submissions remaining.

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

(1) (2)





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

Submit | View Answer **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`.

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

Check Syntax | Submit | View Answer **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 $o = \sigma(z)$.

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

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

5.3)

5.3.1)

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

.25

Check Formatting | Submit | View Answer **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 $\frac{\partial \sigma(z)}{\partial z}$? In other words, $\frac{\partial \sigma(z)}{\partial z}$ has an unreachable lower bound, what is the value of this lower bound?

0

Check Formatting | Submit | View Answer **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 gradient 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

$\nabla_\theta g =$ g*(1-g)*(x)

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

5.5)

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

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

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

$\nabla_g L_{\text{nll}} =$ -y/g + (1-y)/(1-g)

Check Syntax | Submit | View Answer 100.00%

You have infinitely many submissions remaining.

5.6)

What is the gradient 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

$\nabla_\theta L_{\text{nll}} =$ -y/g*g*(1-g)*(x) + (1-y)/(1-g)*g*(1-g)*(x)

Check Syntax | Submit | View Answer 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`.

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

Check Syntax | Submit | View Answer 100.00%

You have infinitely many submissions remaining.

6) Multi-class Logistic Regression

You might want to refer to Section 4.5 in the lecture notes about how we can make use of **softmax** function and handle multiple classes in classification.

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): [.09, .245, .665]

Check Formatting | Submit | View Answer 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: -log(0.2)

Check Syntax | Submit | View Answer 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 (confidence) 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): .665

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 $\nabla_\theta \mathcal{L}_{\text{NLLM}}(g, y)$?

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.

[[.245, -.335, .09],[.245, -.335, .09]]

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 (confidence) that x is in class 1, given the new weight matrix (the offset θ_0 is kept the same and not updated)?

Enter a number: .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:

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

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. For your convenience, we have copied the sample test cases into a colab notebook.

For this question, you may find Section 4.4 of the notes particularly helpful.

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 $1 \times n$, `th` is $d \times 1$, `th0` is 1×1, `lam` 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.

```python
# returns a vector of the same shape as z
def sigmoid(z):
    def sigma(x):
        return 1/(1+np.e**-x)
    vectorized_sigmoid = np.vectorize(sigma)
    return vectorized_sigmoid(z)

# X is dxn, y is 1xn, th is dx1, th0 is 1x1

# returns a (1,n) array for the nll loss for each data point given th and th0

def nll_loss(X, y, th, th0):
    g = sigmoid(th.T@X + th0)
    loss = -(y*np.log(g) + (1-y)*(np.log(1-g)))
    return loss

# X is dxn, y is 1xn, th is dx1, th0 is 1x1, lam is a scalar

# returns a scalar for the llc objective over the dataset

def llc_obj(X, y, th, th0, lam):
    loss = nll_loss(X, y, th, th0)
    return ((1/X.shape[1])*(np.sum(loss)) + lam * (th.T@th))[0][0]

```

100.00%

You have infinitely many submissions remaining.

Here is the solution we wrote:

```python
# returns a vector of the same shape as z
def sigmoid(z):
    return 1/(1+np.exp(-z))

# X is dxn, y is 1xn, th is dx1, th0 is 1x1

# returns the nll loss for each data point given th and th0

def nll_loss(X, y, th, th0):
    pre_result = np.dot(th.T, X) + th0
    g = sigmoid(pre_result)

    cost = -(y*np.log(g) + (1-y)*np.log(1-g))
    return cost

# X is dxn, y is 1xn, th is dx1, th0 is 1x1, lam is a scalar

# returns the llc objective over the dataset

def llc_obj(X, y, th, th0, lam):
    return np.mean(nll_loss(X, y, th, th0)) + lam*np.linalg.norm(th)**2
```

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()
```

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. And a **heads-up**: You may notice that one of the test results returns an "exact" zero; in case you're surprised, an explanation can be found below:

▶ (click to expand) Explanation for numerical versus exact zero

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

```python
# returns a (1,1) array for the gradient of sigmoid with respect to its input z
def d_sigmoid(z):
    return np.exp(-z)*(1+np.exp(-z))**(-2)

# returns a (d,n) array for the gradient of nll_loss(X, y, th, th0) with respect to th for each da
def d_nll_loss_th(X, y, th, th0):
    g = sigmoid((np.dot(th.T,X) + th0))
    return (g-y)*X

# returns a (1,n) array for the gradient of nll_loss(X, y, th, th0) with respect to th0
def d_nll_loss_th0(X, y, th, th0):
    g = sigmoid((np.dot(th.T,X) + th0))
    return g-y

# returns a (d,1) array for the gradient of llc_obj(X, y, th, th0) with respect to th
def d_llc_obj_th(X, y, th, th0, lam):
    #return np.mean(np.sum(d_nll_loss_th(X, y, th, th0))) + 2*lam*th
    return np.mean(d_nll_loss_th(X, y, th, th0), axis=1, keepdims=True)+ 2 * lam * th


# returns a (1,1) array for the gradient of llc_obj(X, y, th, th0) with respect to th0
def d_llc_obj_th0(X, y, th, th0, lam):
    return np.mean(d_nll_loss_th0(X, y, th, th0), keepdims=True)

# returns a (d+1, 1) array for the full gradient as a single vector (which includes both th, th0)
def llc_obj_grad(X, y, th, th0, lam):
    return np.vstack((d_llc_obj_th(X, y, th, th0, lam), d_llc_obj_th0(X, y, th, th0, lam)))
```

Run Code | Submit | View Answer 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 θ, θ_0 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.

```python
def llc_min(data, labels, lam):
    """
    Parameters:
        data: dxn
        labels: 1xn
        lam: scalar
    Returns:
        same output as gd
    """
    def llc_min_step_size_fn(i):
       return 2/(i+1)**0.5
    d  = data.shape[0]
    th = np.zeros((d,1))
    th_0 = np.zeros((1,1))
    x0 = np.vstack((th, th_0))
    def f(x):
        return llc_obj(data, labels, x[0:d,:], x[d:,:], lam)
    def df(x):
        return llc_obj_grad(data, labels, x[0:d,:], x[d:,:], lam)
    return gd(f, df, x0, llc_min_step_size_fn, 10)
```

Run Code | Submit | View Answer 100.00%

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))
```

Survey

(The form below is to help us improve/calibrate for future assignments; submission is encouraged but not required. Thanks!)

How did you feel about the **length** of this homework?

- ○ Too long.
- ◉ About right.
- ○ Too short.

How did you feel about the **difficulty** of this homework?

- ○ Too hard. We should tone it down.
- ◉ About right.
- ○ Too easy. I want more challenge.

Do you have any feedback or comments about any questions in this homework? Anything else you want us to know?

Submit