

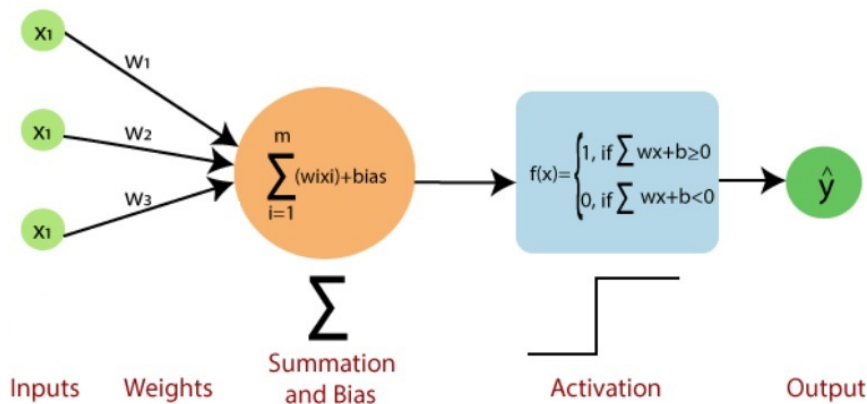
Module 1 Assignment

[Start Assignment](#)

Due Sep 19 by 11:59pm **Points** 50

Submitting a text entry box, a website url, or a file upload

Available until Sep 23 at 11:59pm



Module 1 Assignment

Due: Sept 19

Grace Period: Yes + 24 hours

Late Policy: After Grace Period Ends, -10% per day for up to 3 days.

Cannot be submitted after Sept 23

All due dates are always 11:59pm MT

Overview:

This assignment will enable you to practice with logistic regression, coding in Python, Loss functions (binary cross entropy in this case), learning rates, gradient descent and derivatives, activation function (sigmoid in this case), and drawing an architecture.

All concepts here will extend into perceptrons and ANNs with layers. For this reason, it is critical to gain a solid foundation, to practice, to code these concepts by hand (using an easy and small dataset), and to work through the math.

All requirements, concepts, and math here will be on Exam 1.

This is an individual assignment. This means that you should do your own work and your submission should be yours alone. While it is OK to discuss concepts, it is not OK to duplicate work. If you ever have any questions about cheating, plagiarism, sharing, etc. please ask.

Requirements:

1) Create your own small, 3D, labeled dataset. Your labels should be binary. Save your dataset as A1_Data_YourName.csv. You will be submitting this raw dataset with your Assignment. A 3D dataset has three columns (variables/features). The label will be in the 4th column. Recall that labels are not part of the dataset but rather are the categories (also called classes or groups) that each data row (example, observation) is a member of. Your dataset should have 5 - 7 rows. Please invent the dataset BUT be sure that it makes sense. In other words, the labels should have a correlation with the data. You do not need to mathematically test the correlation :) just do it intuitively. **All of your data must be numeric.** The label (which is not part of the dataset) can be any binary label such as POS and NEG, YES and NO, BBPlayer and NotBBPlayer. There are an infinite number of options.

Example: This is an example of **4D data** with a binary label called "Children".

AGE	HEIGHT	WEIGHT	WAIST	Children
23	64	114	67	Yes
32	66	165	82	No
25	62	107	66	Yes
26	62	160	80	No
29	63	123	75	Yes
25	59	147	73	No
22	64	156	81	Yes
41	67	218	99	No
32	64	110	67	No
31	66	188	100	Yes

2)

(a) Write out a general linear equation (call it z) to represent a linear function that multiplies individual weights by each x value.

Example: For a 4D dataset, a general linear equation is:

$$z = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

(b) Write your linear equation using **vector format**.

Example:

$$z = \mathbf{w}^T \mathbf{x} + b$$

(c) What do the w values represent?

(d) What does the " b " represent?

3) Use Python to read in your dataset and to plot the data (not the label) in 3D.

Hint: (This is an option, not the only option :)

```
import matplotlib.pyplot as plt
```

```
fig = plt.figure()
```

```
ax = plt.axes(projection='3d')
```

```
x1 = ....
```

```
x2 = ....
```

```
x3 = ....
```

```
# plotting
```

```
ax.scatter(x1, x2, x3, s=50)
```

```
ax.set_title('Dataset')
```

```
plt.show()
```

4) Does your dataset appear to have clusters or groups? Does it appear to be separable - especially with respect to your labeling?

(Side note: If your data appears chaotic and/or does not appear separable with respect to your labeling, make adjustments to your dataset and do the above again. The goal here is to have an easy and simple dataset that will do what is expected.)

5)

(a) Write out the Sigmoid function - include both ways to write it.

(b) Prove that $1 / (1 + e^{-x})$ is the same as $e^x / (1 + e^x)$ are the same. **Show all steps carefully.**

(c) Use Python to create a plot (from -100 to 100 with steps of 1) of the Sigmoid function. Include the plot here.

(d) Let $S(z) = 1 / (1 + e^{-z})$ be the Sigmoid function. Prove that the first derivative of $S(z)$ is $S(z) * (1 - S(z))$. **Show every step carefully.**

6) Using your dataset and an initial weight vector of all ones, an initial b value of 0, do the following. (Before you start, be sure to convert your labels to 0 and 1. For example, if my labels are POS and NEG, then I can change all POS to 1 and all NEG to 0.)

(a) What is your weight vector? Show it here. (You can draw it if you wish)

(b) Use your weight vectors of all ones and your b of 0 and your dataset to calculate z for all of your data. **Show every step and do this by hand.**

(c) Once you have your z values, apply the Sigmoid function to each and show your $y^{\wedge} = S(z)$ vector. **Do this by hand and show every step.**

(d) Are your current parameters (\mathbf{w} and b) doing a good job predicting the correct labels? Show the correct label (y) and the predicted label (y^{\wedge}). How many are incorrect? (Note that y^{\wedge} is pronounced as y hat and is the value predicted by applying the Sigmoid to the linear equation). **Show all of your work.**

(e) What can you do to improve your predictions? In other words, what can you do to reduce the differences between the y (known label) and the y^{\wedge} (predicted value).

Required: For the above, do this in vector/matrix form.

Example: (Yes I know this is not pretty :)

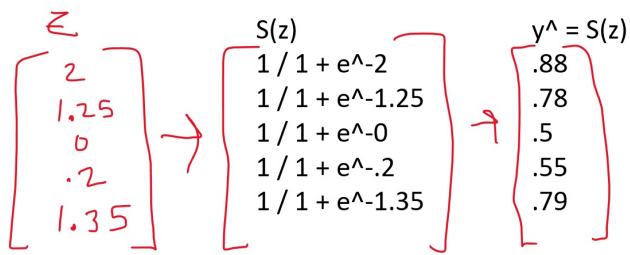
$$\begin{pmatrix} 1 \\ 0.5 \\ 0 \\ 0.1 \\ 0.6 \end{pmatrix} \odot \begin{pmatrix} 1 \\ 0.75 \\ 0 \\ 0.1 \\ 0.75 \end{pmatrix} \odot \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + b = \begin{pmatrix} 1w_1 + 1w_2 + b \\ 0.5w_1 + 0.75w_2 + b \\ 0w_1 + 0w_2 + b \\ 0.1w_1 + 0.1w_2 + b \\ 0.6w_1 + 0.75w_2 + b \end{pmatrix}$$

Using \mathbf{w} as $[1 \ 1]$ and b as 0 then \mathbf{z} is:

$$\begin{aligned} (1)(1) + (1)(1) + 0 &= 2 \\ .5(1) + .75(1) + 0 &= 1.25 \\ 0(1) + 0(1) + 0 &= 0 \\ .1(1) + .1(1) + 0 &= .2 \\ .6(1) + .75(1) + 0 &= 1.35 \end{aligned}$$

$$\mathbf{z} = \begin{pmatrix} 2 \\ 1.25 \\ 0 \\ .2 \\ 1.35 \end{pmatrix}$$

AND then...



7) [Note - LCE and L_{CE} will mean the same thing]

(a) Write out the cross-entropy function for binary labels here.

$$L_{CE}(y, \hat{y}) = \dots$$

(b) For all of your data rows, calculate the LCE. **Show all of your work and do this by hand.**

Include ALL steps. Continue to use the weight vector of all ones and a b of 0.

(c) Using your result from (b), what would you say your current "Loss" or "error" is? How might you improve/reduce this? What does it mean if the LCE is 0?

8) To reduce the error (or loss), we need to adjust (change) the parameters. Recall that in this case, our parameters are \mathbf{w} and b . If we change the values of \mathbf{w} and/or b , this will result in a change in the LCE and in the overall difference between \hat{y} and y .

Do this by hand.

Change your \mathbf{w} to values other than 1. Leave b as 0. Recalculate LCE.

Show every step and all of your work. Do this by hand. Your \mathbf{w} values are invented by you.

9) To make changes to parameters, such as \mathbf{w} and b so that we get closer and closer to the minimum (in other words to optimize the values of \mathbf{w} and b) we need to take the partial derivatives of LCE with respect to both \mathbf{w} and b . The **Jacobian** is a vector or matrix that contains all partial derivatives of a function with respect to all possible parameters. Here, our Jacobian has two values, one for $dLCE/d\mathbf{w}$ and one for $dLCE/db$, where "d" here is partial derivative.

(a) Find the partial derivative of the LCE with respect to \mathbf{w} . **Show all steps and work and use the chain rule. HINT (chain rule):** $dLCE/d\mathbf{w} = dLCE/d\hat{y} * d\hat{y}/dz * dz/d\mathbf{w}$

HINT2: The two derivatives are:

$$dLCE/d\mathbf{w} = 1/n (\hat{y} - y)^T \mathbf{x}$$

$$dLCE/db = 1/n (y^{\wedge} - y)$$

You will be graded on the work as you already have the final answer :)

(b) In words, how would you describe the derivatives here? What do that "mean"? (3 - 4 sentences is enough :)

10)

(a) Using $dLCE/d\mathbf{w} = 1/n (y^{\wedge} - y)^T \mathbf{x}$ and using the original \mathbf{w} of all ones, calculate and update \mathbf{w} . **Show all of the work and do this by hand vector style.** (For now we will assume that the learning rate, LR, is 1).

Example:

Update for \mathbf{w}

(without the LR yet – so LR=1)

To get the update for \mathbf{w} :

$$\frac{1}{5} \begin{bmatrix} -.12 & -.12 & .5 & .55 & -.21 \end{bmatrix} \begin{bmatrix} 1 \\ .5 \\ 0 \\ .1 \\ -.6 \end{bmatrix}$$

$\hat{y} - y$

$$= 1/5 [(-.12*1 + -.22*.5 + .5*0 + .55*.1 + -.21*.6) \\ (-.12*1 + -.22*.75 + .5*0 + .55*.1 + -.21*.75)]$$

$$\rightarrow 1/5 [-.3 \quad -.39] \rightarrow$$

$$[-.06 \quad -.08] \text{ is the update for } \mathbf{w}$$

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial \mathbf{w}} = \frac{1}{m} (\hat{y} - y) \mathbf{x}_i^T$$

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial b} = \frac{1}{m} (\hat{y} - y)$$

y^{\wedge} is	y is
.88	1
.78	1
.5	0
.55	0
.79	1

(b) Once you update \mathbf{w} and keeping b as 0, recalculate LCE and compare your new LCE to the original LCE. Did it reduce? **Show all work and steps and do this by hand.**

11) Code logistic regression using Python. DO NOT use a "package" but rather, code it by hand. Yes you can use simple packages like Numpy, etc. You cannot use a Logistic regression package.

You must write code that will:


(a) Read in any .csv labeled dataset where the Label is called "LABEL".


(b) Your code should create "y" (the label) as a numpy array of 0 and 1. So, if the labels are originally something like POS and NEG, your code will need to determine this, and then update all POS to 1 and all NEG to 0. Your code should be able to do this for any two label options.

(c) Your code will also create X which is the entire dataset (without the labels) as a numpy array.

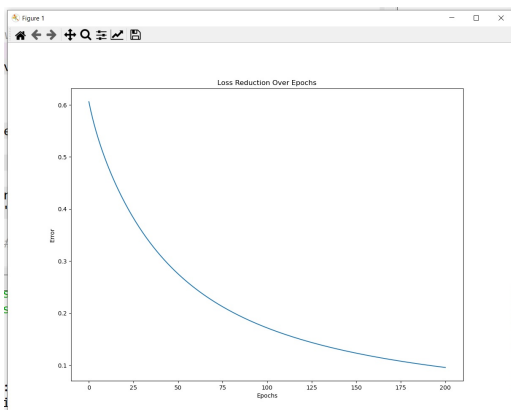
- (d) Your code will initialize w and b and a learning rate.
- (e) Your code will need to contain a function for Sigmoid and for z , etc.
- (f) Your code will then need to perform gradient descent to optimize (reduce) LCE.
- (g) Be sure that your code can run iteratively for as many epochs as you wish.
- (h) Show a final confusion matrix (see example below) AND show a graph of how the LCE reduces as epochs iterate (see example below)

Recommended: Use your code to see if your by-hand results from above are right. The goal here is to really understand all the steps, which steps are repeated (epochs), the loss function (LCE here), the activation function (Sigmoid here), updating the parameters, etc.

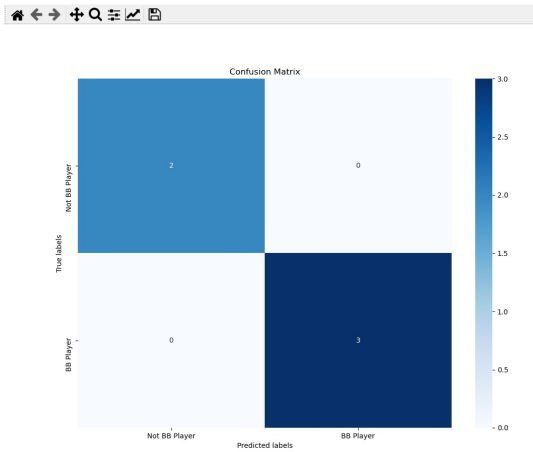
Here is [example code that I wrote](https://gatesboltonanalytics.com/?page_id=873)  (https://gatesboltonanalytics.com/?page_id=873). I strongly suggest NOT using this code first - but rather - try to understand the above and write your own code from scratch. If you get stuck, you can use my code as a reference.

(Direct link to code and a tutorial: https://gatesboltonanalytics.com/?page_id=873  (https://gatesboltonanalytics.com/?page_id=873).

Example: Optimization of LCE over epochs (iterations of gradient descent)



Example: Confusion Matrix



Hints:

```
cm = confusion_matrix(labels, Prediction)
print(cm)
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap='Blues')
#annot=True to annotate cells, fmt='g' to disable scientific notation

# labels, title and ticks
ax.set_xlabel("Predicted labels")
ax.set_ylabel("True labels")
ax.set_title("Confusion Matrix")
ax.xaxis.set_ticklabels(["Not BB Player", "BB Player"])
ax.yaxis.set_ticklabels(["Not BB Player", "BB Player"])
```

Some Rubric

Criteria	Ratings	Pts
Q1		1 pts
Q2		1 pts
Q3		4 pts
Q4		1 pts
Q5		7 pts
Q6		7 pts
Q7		2 pts
Q8		2 pts
Q9		10 pts
Q10		5 pts
Q11		10 pts
Total Points: 50		