# ECEN 377: Engineering Applications of AI

**Dr. Mahmoud Nabil Mahmoud**
*mnmahmoud@ncat.edu*

North Carolina A & T State University

September 26, 2024

# Outline

# Classification vs Regression

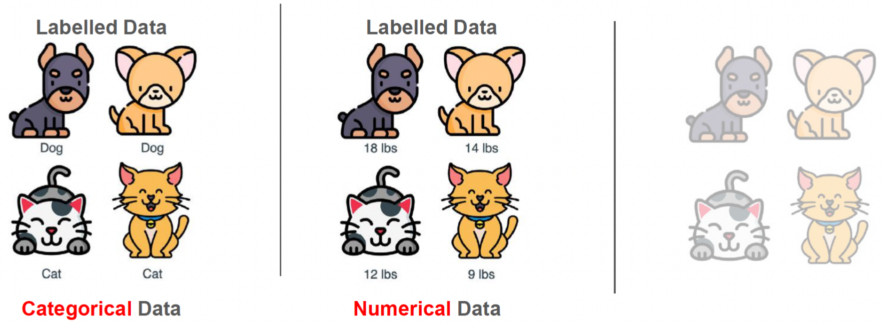
**Classification Models**

- The models that predict **categorical** output
- The output is **discrete**
- Example: Type of animal (cat or dog)
- Example: Email spam detection model

**Regression Models**

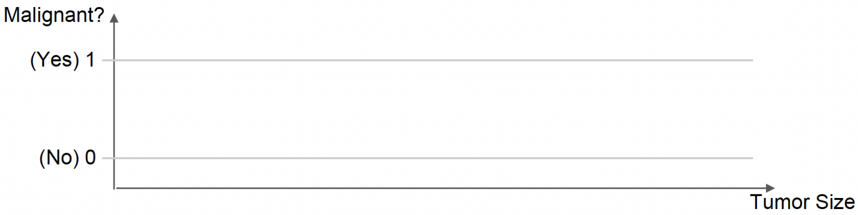
- The models that predict **numerical** output
- The output is a **number**
- Example: Housing prices model
- Example: Predicting the weight of an object

# Classification vs Regression



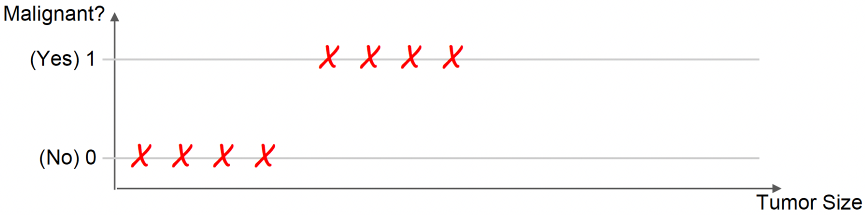
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



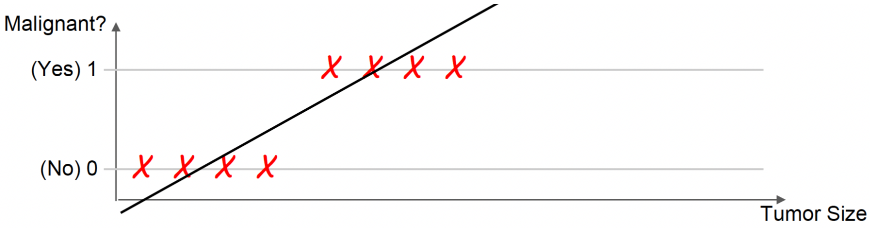
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



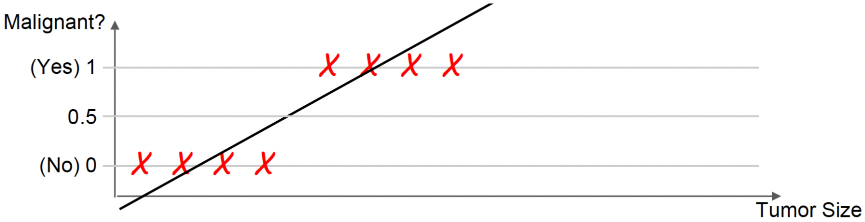
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



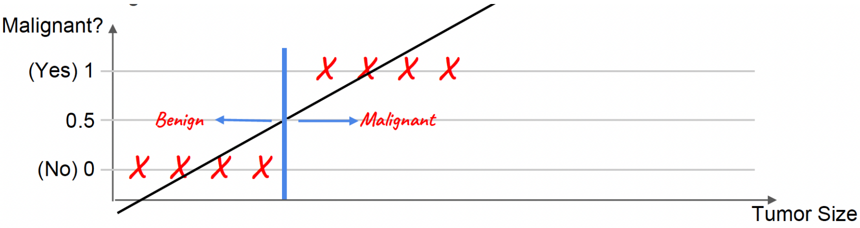
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



# Linear Regression As A Classifier!

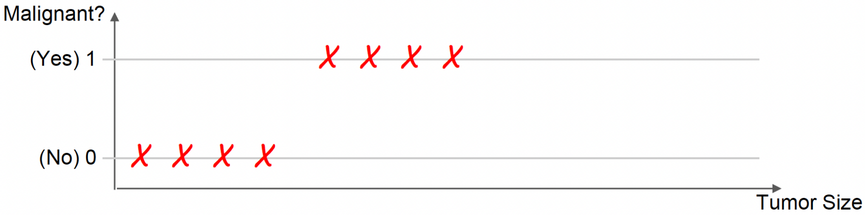
Can use linear regression as a classifier by using a threshold on the output of the linear regression model.




**Can it work for all cases?**

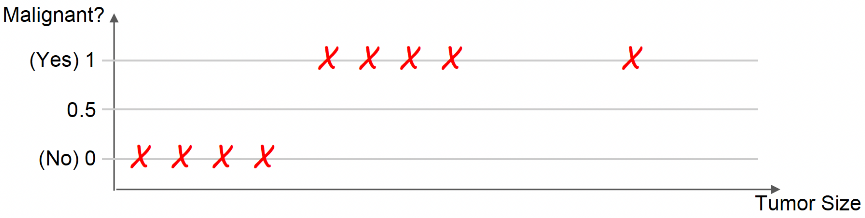
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



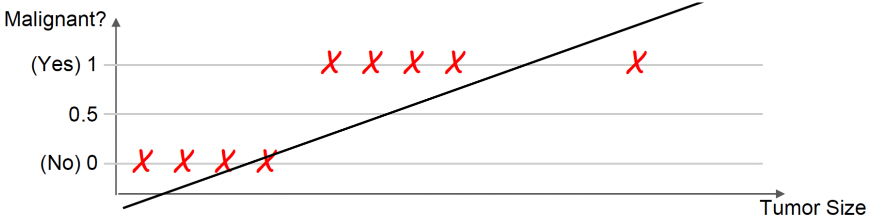
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



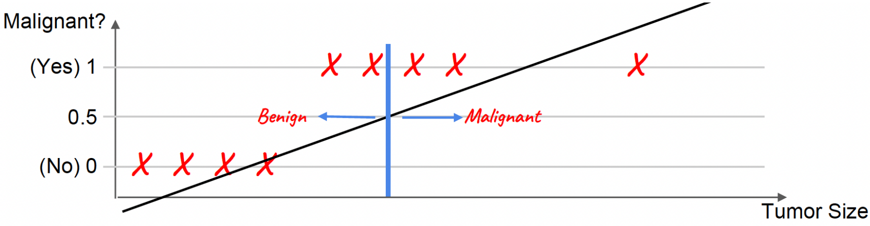
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



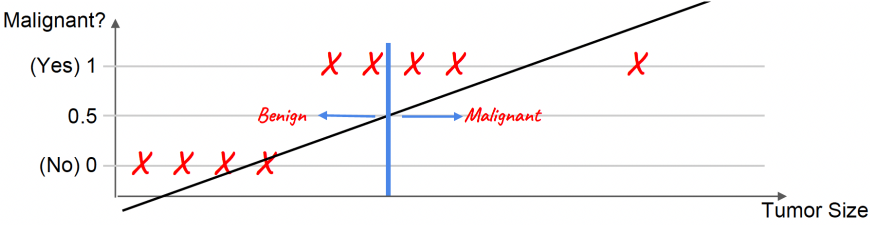
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



# Linear Regression As A Classifier!

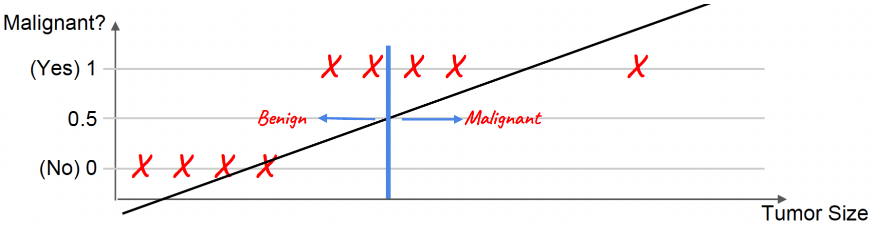
Can use linear regression as a classifier by using a threshold on the output of the linear regression model.




**Linear regression as a classifier will be sensitive to outliers**

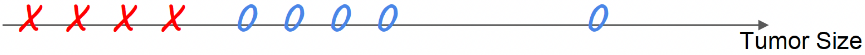
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.



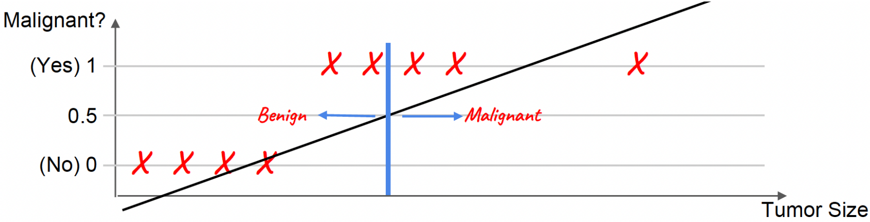

**Linear regression as a classifier will be sensitive to outliers**

As a classification problem:



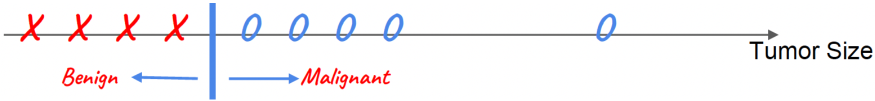
# Linear Regression As A Classifier!

Can use linear regression as a classifier by using a threshold on the output of the linear regression model.




**Linear regression as a classifier will be sensitive to outliers**

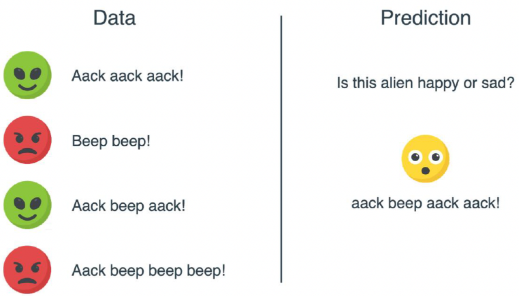
As a classification problem:



# Outline

# Problem Definition

- Imagine you invade a new planet
- The aliens have 2 words: "Aack" and "Beep"
- You want to know if an alien is happy or sad
- You got this data from your experience with some aliens




**What do you notice about this data?**

**How do we solve this?**

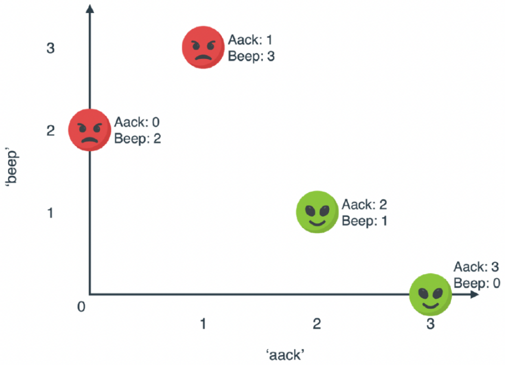
# Star Wars!

Let's build our data set:

| Sentence | Aack | Beep | Mood |
|---|---|---|---|
| Aack aack aack! | 3 | 0 | Happy |
| Beep beep! | 0 | 2 | Sad |
| Aack beep aack! | 2 | 1 | Happy |
| Aack beep beep beep! | 1 | 3 | Sad |

# Star Wars!

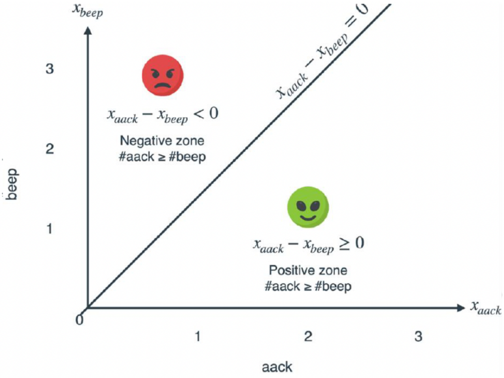
Draw a line that classifies the data correctly.



# Star Wars!

Draw a line that classifies the data correctly.

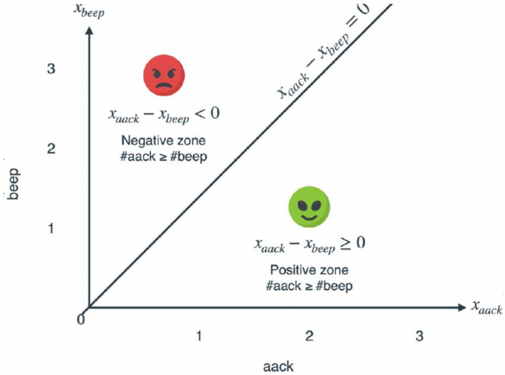
- Line: $X_{aack} = X_{beep}$
- Or
- Line: $X_{aack} - X_{beep} = 0$



# Star Wars!

Draw a line that classifies the data correctly.

- Line: $X_{aack} = X_{beep}$
- Or
- Line: $X_{aack} - X_{beep} = 0$




**We call this line the decision boundary.**

# Star Wars!

What do you notice about this planet?

# Star Wars!

What do you notice about this planet?

# Star Wars!

What do you notice about this planet?

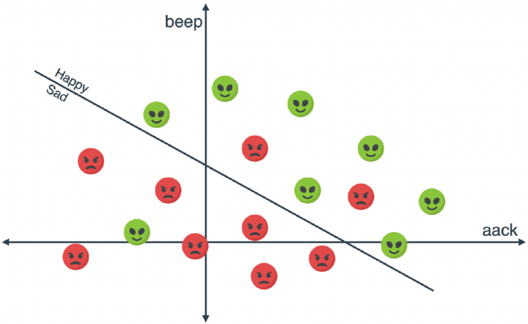
- $X_{crack} + X_{doink} - 3.5 = 0$
- Happy: $X_{crack} + X_{doink} - 3.5 > 0$
- Sad: $X_{crack} + X_{doink} - 3.5 < 0$

Is it always that simple?

# Another dataset example

Model could mistake some samples

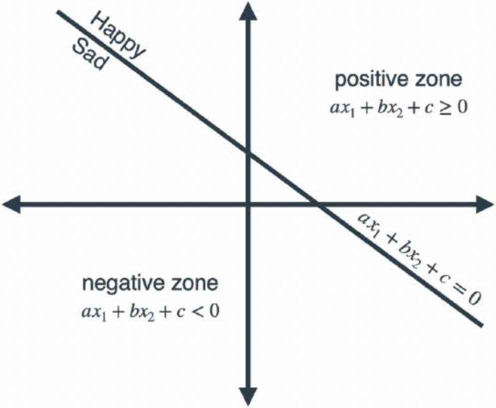
- We try to find the most general model **(linear decision boundary)**
- With least error



# Outline

# Perceptron Classifier [Formula]

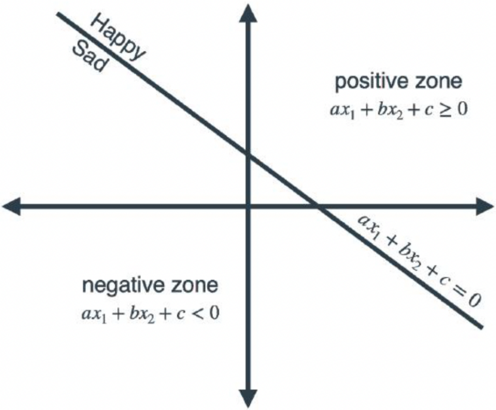
- The general form of classifiers using line: $ax_1 + bx_2 + c = 0$




An example of a plane model (3 features, plane model): $ax_1 + bx_2 + cx_3 + d = 0$

# Perceptron Classifier [Formula]

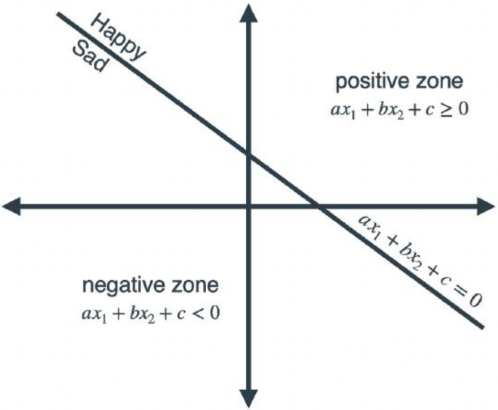
- The general form of classifiers using line: $ax_1 + bx_2 + c = 0$
- This equation purpose is different from linear regression purpose but they are equivalent




An example of a plane model (3 features, plane model): $ax_1 + bx_2 + cx_3 + d = 0$

# Perceptron Classifier [Formula]

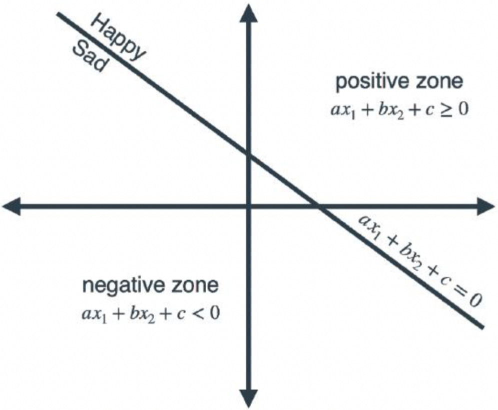
- The general form of classifiers using line: $ax_1 + bx_2 + c = 0$
- This equation purpose is different from linear regression purpose but they are equivalent
- **What do $a$, $b$, and $c$ represent?**




An example of a plane model (3 features, plane model): $ax_1 + bx_2 + cx_3 + d = 0$

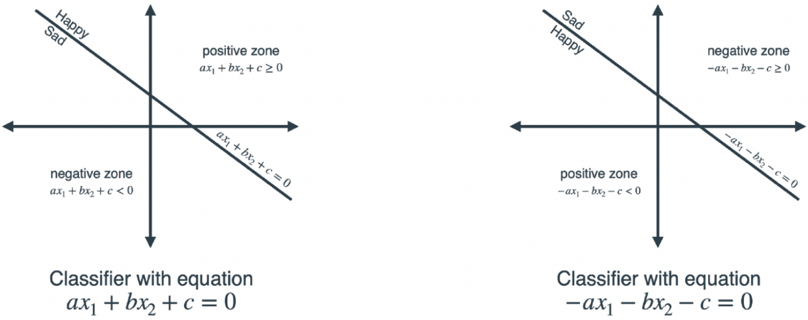
# Perceptron Classifier [Formula]

- The general form of classifiers using line: $ax_1 + bx_2 + c = 0$
- This equation purpose is different from linear regression purpose but they are equivalent
- **What do $a$, $b$, and $c$ represent?**
- This formula could be extended if features are more than 2



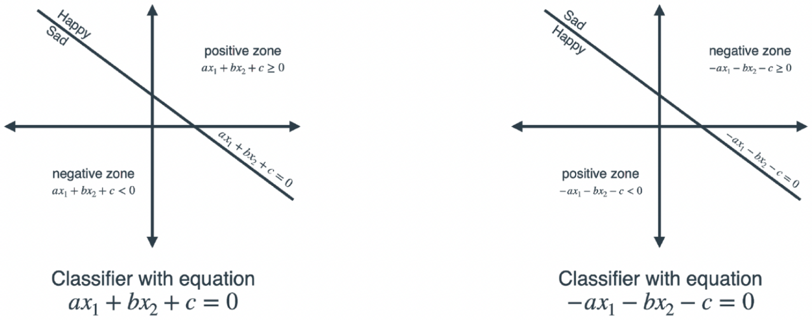

An example of a plane model (3 features, plane model): $ax_1 + bx_2 + cx_3 + d = 0$

# So why we use this formula?




if we multiply by negative one, the sign of the classification will be flipped

# So why we use this formula?




if we multiply by negative one, the sign of the classification will be flipped

**How to take the decision?**

# Perceptron Classifier [Step function]

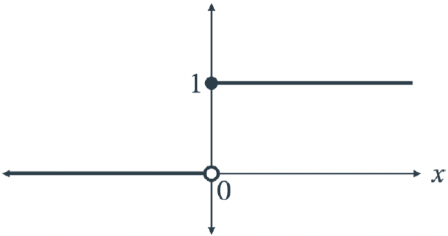
- The step function is used to make a binary decision
- It's defined as:

$$step(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- For perceptron, we use:

$$y' = step(ax_1 + bx_2 + c)$$

- 1 is happy
- 0 is sad



# Perceptron Classifier [Step function]

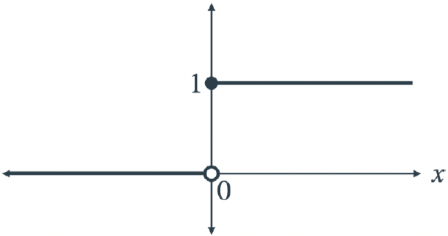
- The step function is used to make a binary decision
- It's defined as:

$$step(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- For perceptron, we use:

$$y' = step(ax_1 + bx_2 + c)$$

- 1 is happy
- 0 is sad

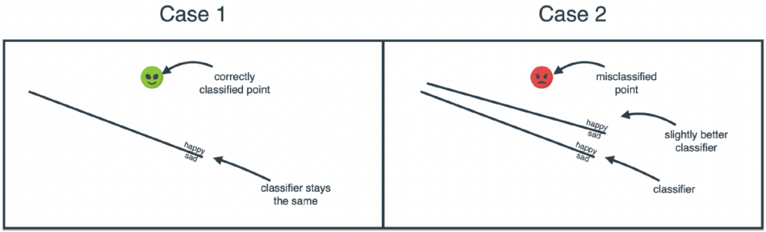



**How do we find $a$, $b$, and $c$?**

# Outline

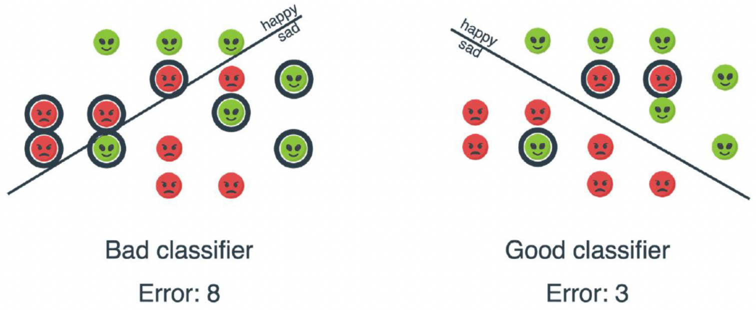
# Loss Defination

- At each Training iteration we get a data point from our dataset:
  - Case 1: If the point is correctly classified, no loss.
  - Case 2: If the point is incorrectly classified, that means it produces an error (loss).
    - Distant points that are misclassified incur greater loss
    - Nearby points that are misclassified incur lesser loss



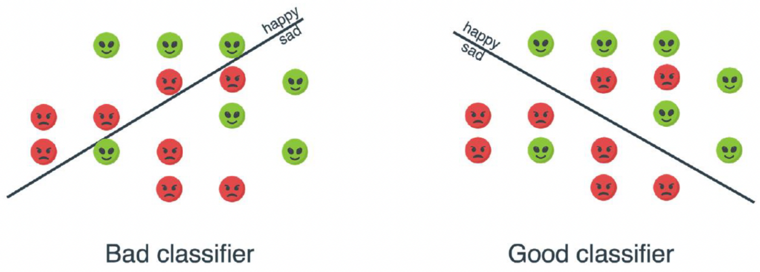
# Perceptron Classifier [loss function]

1. lets try the number of misclassified samples as a measure of loss



# Perceptron Classifier [loss function]

lets try the number of misclassified samples as a measure of loss



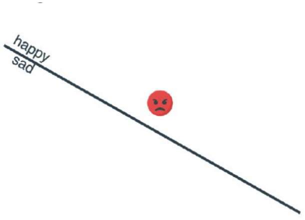
# Perceptron Classifier [loss function]

**What is the problem with this loss function?**

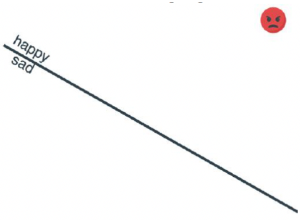
# Perceptron Classifier [loss function]

**What is the problem with this loss function?**

- It penalize close and far mistakes the same
- It is difficult to assess convergence or progress



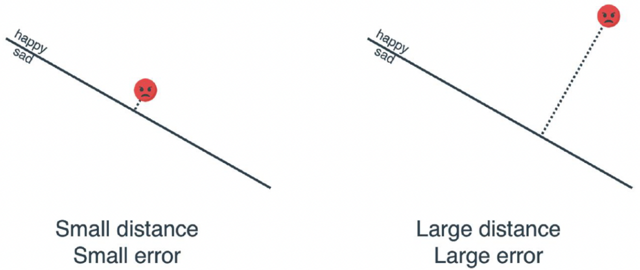

Poorly misclassified




Very poorly misclassified

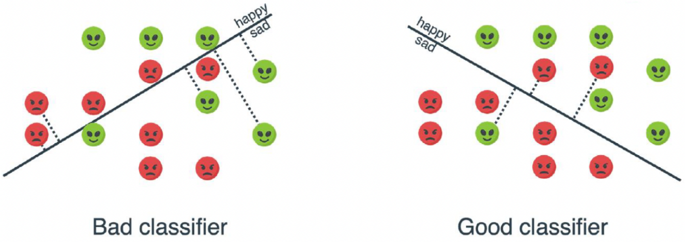
# Perceptron Classifier [loss function]

2. Let's distance from the decision boundary as a loss function



# Perceptron Classifier [loss function]

**What is the problem with this loss function?**




It is mathematically complex to compute

# Perceptron Classifier [loss function]

We need a loss function that is differentiable and easy to compute with gradient descent

**Requirements:**

- The points that are misclassified and far from the decision boundary should contribute more to the loss

# Perceptron Classifier [loss function]

We need a loss function that is differentiable and easy to compute with gradient descent

**Requirements:**

- The points that are misclassified and far from the decision boundary should contribute more to the loss
- The points that are misclassified and close to the decision boundary should contribute less to the loss

# Perceptron Classifier [loss function]

We need a loss function that is differentiable and easy to compute with gradient descent

**Requirements:**

- The points that are misclassified and far from the decision boundary should contribute more to the loss
- The points that are misclassified and close to the decision boundary should contribute less to the loss

# Perceptron Classifier [loss function]

We need a loss function that is differentiable and easy to compute with gradient descent

**Requirements:**

- The points that are misclassified and far from the decision boundary should contribute more to the loss
- The points that are misclassified and close to the decision boundary should contribute less to the loss

**Solution:**

$$ax_1 + bx_2 + c$$

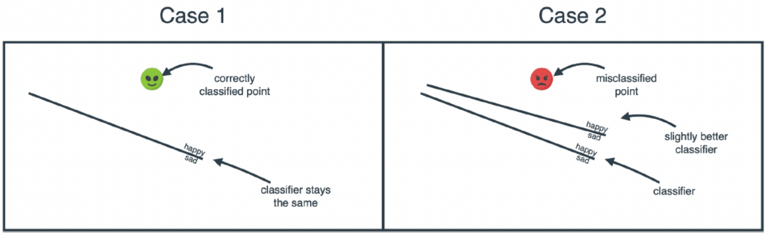
# Perceptron Classifier [loss function]

- Loss function:
  - If the sentence is correctly classified, the error is 0
  - If the sentence is misclassified, the error is $|ax_1 + bx_2 + c|$
- This scoring function satisfies our requirements:
  - Correctly classified points contribute zero to the error
  - Misclassified points contribute proportionally to their distance from the decision boundary ($|ax_1 + bx_2 + c|$)
  - It is simple to compute and can be used with gradient descent

# Outline

# Perceptron Classifier [Training]

- Training process:
  - Case 1: If the point is correctly classified, leave the line as it is.
  - Case 2: If the point is incorrectly classified, that means it produces an error. **Adjust the weights and the bias a small amount so that this error slightly decreases.**



# Perceptron Classifier [Derivative of the Loss]

- The loss function for the perceptron can be summarized as:

$$L = \left| w_1 x_1 + w_2 x_2 + b \right|$$

- The update rules for the parameters $a$, $b$, and $c$ are:

$$\begin{aligned} w_1' &= w_1 - \eta \frac{\partial E}{\partial w_1} \\ w_2' &= w_2 - \eta \frac{\partial E}{\partial w_2} \\ b' &= b - \eta \frac{\partial E}{\partial b} \end{aligned}$$

- The partial derivatives are:

$$\begin{aligned} \frac{\partial E}{\partial w_1} &= \text{sign}(w_1 x_1 + w_2 x_2 + b) \cdot x_1 \\ \frac{\partial E}{\partial w_2} &= \text{sign}(w_1 x_1 + w_2 x_2 + b) \cdot x_2 \\ \frac{\partial E}{\partial b} &= \text{sign}(w_1 x_1 + w_2 x_2 + b) \end{aligned}$$

# Perceptron Classifier [Training]

- Pick random weights $w_1, w_2$ and a random bias $b$.
- Repeat **many times**:
  1. Pick a random data point $(x_1^{(i)}, x_2^{(i)}, y^{(i)})$.
  2. Compute Model Prediction:

$$y'^{(i)} = \begin{cases} 1 & \text{if } w_1 x_1^{(i)} + w_2 x_2^{(i)} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

  3. If misclassified (i.e., $y'^{(i)} \neq y^{(i)}$), update the weights and bias:

**Case 1:** $y'^{(i)} = 1, y^{(i)} = 0$

$$\begin{aligned} w_1 &= w_1 - \eta x_1^{(i)} \\ w_2 &= w_2 - \eta x_2^{(i)} \\ b &= b - \eta \end{aligned}$$

**Case 2:** $y'^{(i)} = 0, y^{(i)} = 1$

$$\begin{aligned} w_1 &= w_1 + \eta x_1^{(i)} \\ w_2 &= w_2 + \eta x_2^{(i)} \\ b &= b + \eta \end{aligned}$$

where $\eta$ is the learning rate.

- Return the model you've obtained.

# Perceptron Classifier [Training (Compining cases)]

- Pick random weights $w_1, w_2$ and a random bias $b$.
- Repeat **many times**:
  1. Pick a random data point $(x_1^{(i)}, x_2^{(i)}, y^{(i)})$.
  2. Compute Model Prediction:

$$y'^{(i)} = \begin{cases} 1 & \text{if } w_1 x_1^{(i)} + w_2 x_2^{(i)} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

  3. If misclassified (i.e., $y'^{(i)} \neq y^{(i)}$), update the weights and bias:

$$w_1 = w_1 - \eta(y'^{(i)} - y^{(i)})x_1^{(i)}$$

$$w_2 = w_2 - \eta(y'^{(i)} - y^{(i)})x_2^{(i)}$$

$$b = b - \eta(y'^{(i)} - y^{(i)})$$

  where $\eta$ is the learning rate.
- Return the model you've obtained.

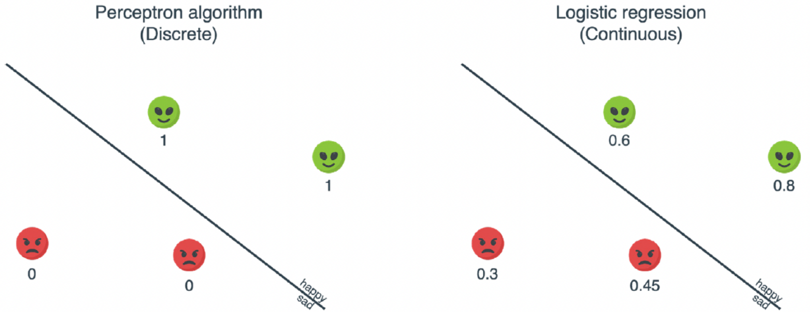
# Outline

# Perceptron Classifier [Drawbacks]

- The step function is discrete which causes:
  - It is not continous function. Derivative is undefined at zero.
  - It would be better if we can get a probablity output.

Our Logistic Regression Classifier solve these problems.

# Logistic Regression vs Perceptron



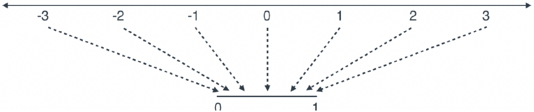

- **Why does it called logistic ‘regression’ while it is a classifier?!**

# Outline

# Logistic regression [ Logistic function(Sigmoid) ]

The output of this function should be continuous [0,1]

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



# Logistic regression [ Logistic function(Sigmoid) ]

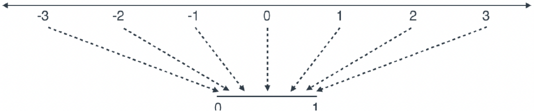
The output of this function should be continuous [0,1]

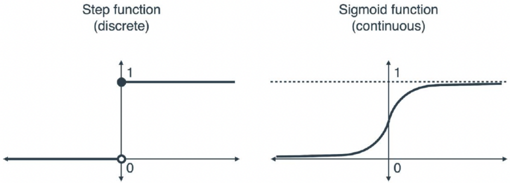
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

What is $\sigma(-\infty)$ ?

What is $\sigma(\infty)$ ?

What is $\sigma(0)$ ?



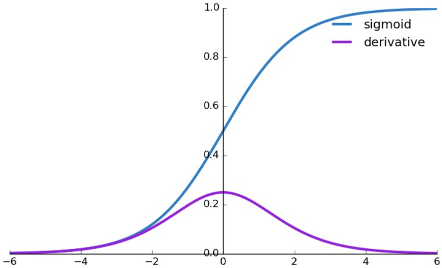
# Logistic regression [ Logistic function(Sigmoid) ]

- In order to map predicted values to probabilities, we use the sigmoid function.

$$sigmoid(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

Another advantage of sigmoid function is the simple derivative:

Derivative of $\sigma(z) =$ $\sigma(z) * (1 - \sigma(z))$



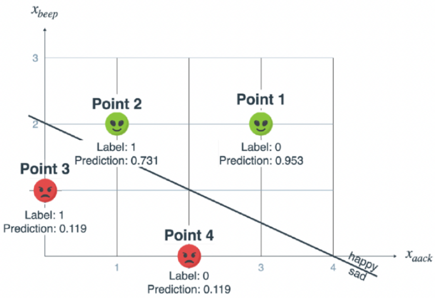
# Logistic regression [Star War Example]

$$\hat{y} = \sigma\left(w_1 \cdot x_{\text{aack}} + w_2 \cdot x_{\text{beep}} + w_0\right)$$

**Prediction:**

$\hat{y} = \sigma\left(1 \cdot x_{\text{aack}} + 2 \cdot x_{\text{beep}} - 4\right)$

- **Point 1:** $\hat{y} = \sigma\left(1 \cdot 3 + 2 \cdot 2 - 4\right) = \sigma(3) = 0.953$ (happy class > 0.5)



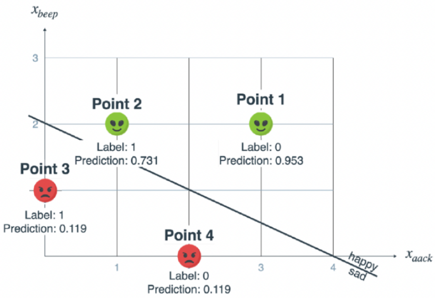
# Logistic regression [Star War Example]

$$\hat{y} = \sigma\left(w_1 \cdot x_{\text{aack}} + w_2 \cdot x_{\text{beep}} + w_0\right)$$

**Prediction:**

$\hat{y} = \sigma\left(1 \cdot x_{\text{aack}} + 2 \cdot x_{\text{beep}} - 4\right)$

- **Point 1:** $\hat{y} = \sigma\left(1 \cdot 3 + 2 \cdot 2 - 4\right) = \sigma(3) = 0.953$ (happy class > 0.5)
- **Point 2:** $\hat{y} = \sigma\left(1 \cdot 1 + 2 \cdot 2 - 4\right) = \sigma(1) = 0.731$ (happy class > 0.5)

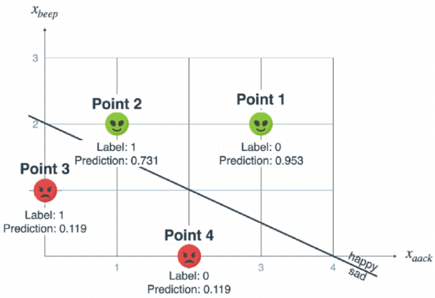


# Logistic regression [Star War Example]

$$\hat{y} = \sigma\left(w_1 \cdot x_{\text{aack}} + w_2 \cdot x_{\text{beep}} + w_0\right)$$

**Prediction:**
$\hat{y} = \sigma\left(1 \cdot x_{\text{aack}} + 2 \cdot x_{\text{beep}} - 4\right)$

- **Point 1:** $\hat{y} = \sigma(1 \cdot 3 + 2 \cdot 2 - 4) = \sigma(3) = 0.953$ (happy class > 0.5)
- **Point 2:** $\hat{y} = \sigma(1 \cdot 1 + 2 \cdot 2 - 4) = \sigma(1) = 0.731$ (happy class > 0.5)
- **Point 3:** $\hat{y} = \sigma(1 \cdot 0 + 2 \cdot 1 - 4) = \sigma(-2) = 0.119$ (sad class < 0.5)



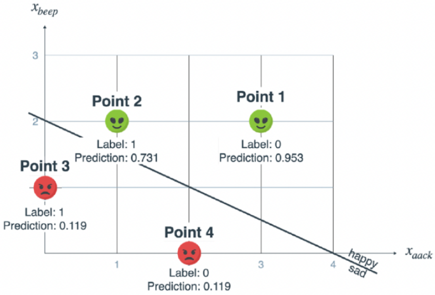
# Logistic regression [Star War Example]

$$\hat{y} = \sigma\left(w_1 \cdot x_{\text{aack}} + w_2 \cdot x_{\text{beep}} + w_0\right)$$

**Prediction:**

$\hat{y} = \sigma\left(1 \cdot x_{\text{aack}} + 2 \cdot x_{\text{beep}} - 4\right)$

- **Point 1:** $\hat{y} = \sigma(1 \cdot 3 + 2 \cdot 2 - 4) = \sigma(3) = 0.953$ (happy class > 0.5)
- **Point 2:** $\hat{y} = \sigma(1 \cdot 1 + 2 \cdot 2 - 4) = \sigma(1) = 0.731$ (happy class > 0.5)
- **Point 3:** $\hat{y} = \sigma(1 \cdot 0 + 2 \cdot 1 - 4) = \sigma(-2) = 0.119$ (sad class < 0.5)
- **Point 4:** $\hat{y} = \sigma(1 \cdot 1 + 2 \cdot 0 - 4) = \sigma(-2) = 0.119$ (sad class < 0.5)



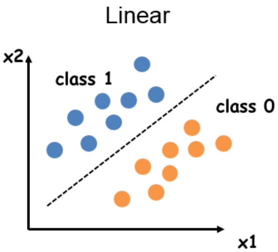
# Logistic Regression [Summary]

## Logistic Regression

It is a statistical model used for binary classification. The inputs are the features values and the output (y) is a probability from 0 to 1.

**Note that**

- Logistic regression is a linear classifier.
- The equation of the decesion boundry : $0 = w_2x_2 + w_1x_1 + w_0$
- Class 0 condition: $0 < w_2x_2 + w_1x_1 + w_0$
- Class 1 condition: $0 > w_2x_2 + w_1x_1 + w_0$




How get y' as probability given these conditions?

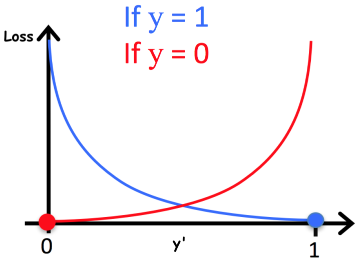
# Logistic Regression

- We can set decesion boundary $z = w_2 x_2 + w_1 x_1 + w_0$
- Then $y' = \sigma(z) = \frac{1}{1+e^{-z}}$
- What if point $(x_1, x_2)$ is below the decesion boundary?
- What if point $(x_1, x_2)$ is above the decesion boundary?
- $\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$

# Outline

# Logistic Loss Function [Log Loss (cross entropy loss)]

- Since y' in logistic regression is a probability between 0 and 1.
- Our loss can be defined with the following loss function.
  - if y = 1 : Loss = -log(y')
  - if y = 0 : Loss = -log(1-y')




$$\text{Loss} = \ell = \text{-ylog(y')-(1-y)log(1-y')}$$

Now we can train with gradient descent to find the weights

# Generalization and Gradient

- For n features: $z = \sum_{i=0}^{i=n} w_i x_i$ , ($w_0$ is the bias)
- vector representation z $= \mathbf{w}^T\mathbf{x}$
- y $= sigmoid(z) = \sigma(z)$
- $\ell = -ylog(y') - (1-y)log(1-y')$

# Gradient Derivation

$$\frac{d\ell}{dw_i} = \frac{d\ell}{dy'}\frac{dy'}{dw_i} = \frac{d\ell}{dy'}\frac{dy'}{dz}\frac{dz}{dw_i}$$

# Gradient Derivation

$$\frac{d\ell}{dw_i} = \frac{d\ell}{dy'}\frac{dy'}{dw_i} = \frac{d\ell}{dy'}\frac{dy'}{dz}\frac{dz}{dw_i}$$

$$= \underbrace{\left[\frac{-y}{\sigma(z)} + \frac{1-y}{1-\sigma(z)}\right]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}}$$

$$= \underbrace{\left[\frac{-y(1-\sigma(z)) + (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}}$$

$$= \underbrace{\left[\frac{-y(1-\sigma(z)) + (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]}_{\frac{d\ell}{dy'}} * \underbrace{\sigma(z)(1-\sigma(z))}_{\frac{dy'}{dz}} * \underbrace{x_i}_{\frac{dz}{dw_i}}$$

$$= (\sigma(z) - y) * x_i = (y' - y) * x_i$$

# Outline

# Perceptron Classifier [Training (Compining cases)]

- Pick random weights $w_1, w_2$ and a random bias $b$.
- Repeat **many times**:
  1. Pick a random data point $(x_1^{(i)}, x_2^{(i)}, y^{(i)})$.
  2. Compute Model Prediction:

$$y'^{(i)} = \sigma(w_1 x_1^{(i)} + w2 x_2^{(i)} + b)$$

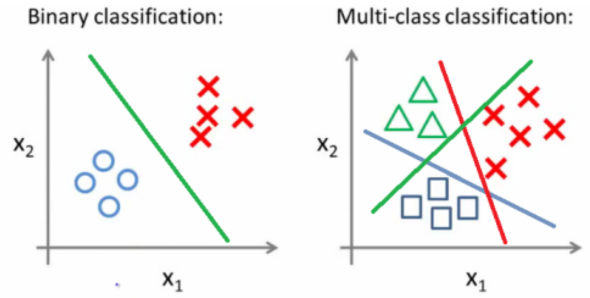
  3. **Directly update the weights and bias:**

$$w_1 = w_1 - \eta(y'^{(i)} - y^{(i)})x_1^{(i)}$$
$$w_2 = w_2 - \eta(y'^{(i)} - y^{(i)})x_2^{(i)}$$
$$b = b - \eta(y'^{(i)} - y^{(i)})$$

  where $\eta$ is the learning rate.
- Return the model you've obtained.

# Outline

# Multiclass Classification (One vs All)




For Three classes

$$\text{Result Class} = \underset{k \in \{1,2,3\}}{\text{argmax}}\, f_k(x)$$


Questions