UCL ARTIFICIAL
INTELLIGENCE SOCIETY

TUTORIAL #3

Session 3

Fundamentals of supervised AI models

UCL ARTIFICIAL
INTELLIGENCE SOCIETY

LECTURE OVERVIEW

01

Linear Regression

What is it and how to use it

02

Model Training

Overview of the most important terms in ML

03

Logistic Regression

Our first trully relevant ML model

04

Other Supervised Models

Support Vector Machine
Decision Trees

Theory

LINEAR REGRESSION RECAP

What is Linear Regression?

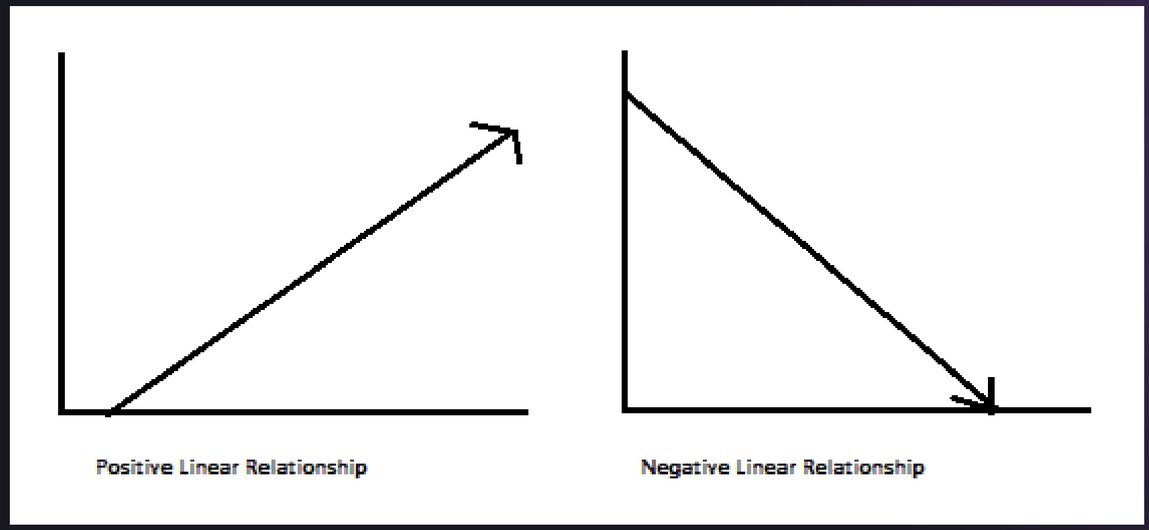
- ML method that is based on purely calculatable statistic

Theory

LINEAR REGRESSION RECAP

What is Linear Regression?

- Is only able to capture **LINEAR RELATIONSHIP**
- Tries to find a best function to fit on the provided data

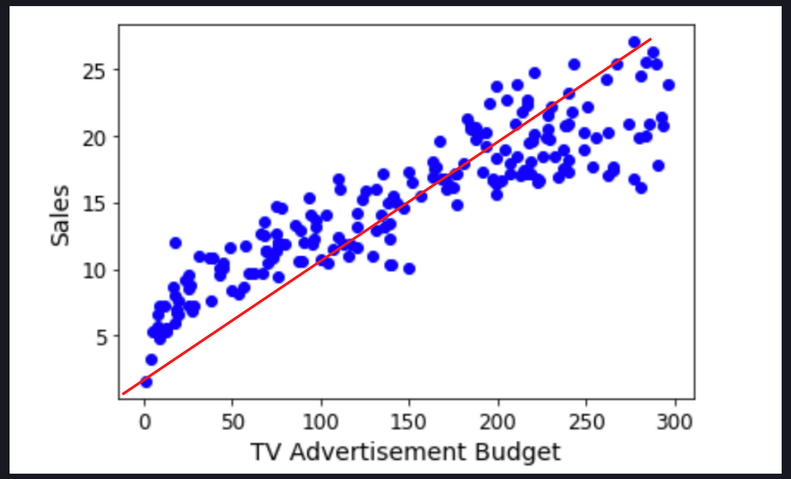


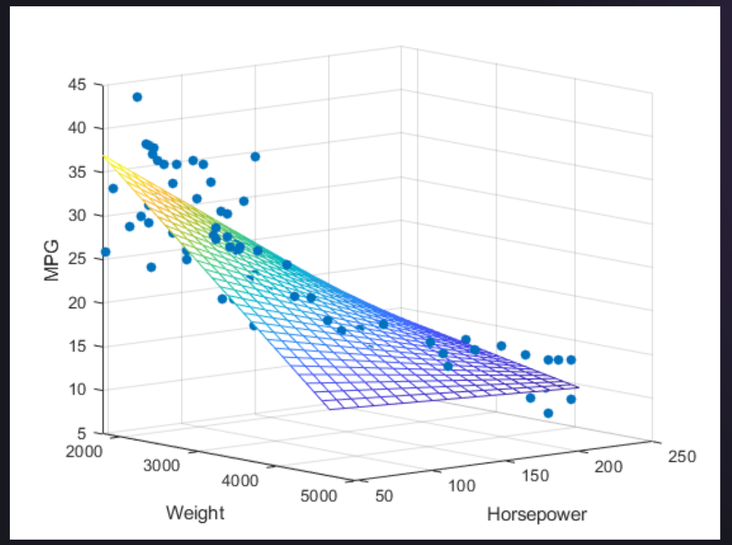
Theory

LINEAR REGRESSION RECAP

What is Linear Regression?

- It can exist in just 2 dimensions (linear function), but it is not limited to them





Theory

LINEAR REGRESSION RECAP

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$$\hat{y} = \theta^T x$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Theory

LINEAR REGRESSION RECAP

Math is FUN😎

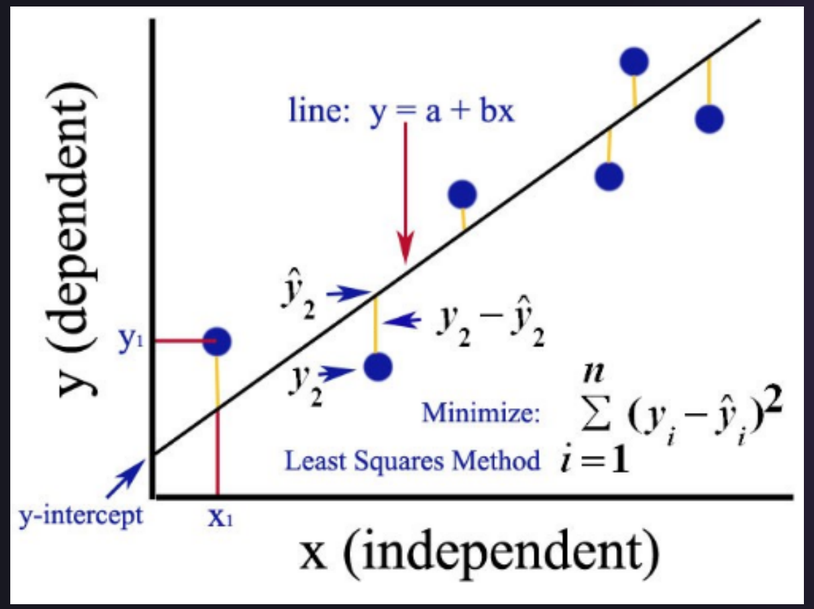
- Ordinary least squares

$$y = m * x + b$$

$$m = \frac{\sum_{i=0}^{n}(x_i - \bar{x}) * (y_i - \bar{y}))}{\sum_{i=0}^{n}(x_i - \bar{x})^2}$$

$$b = \bar{y} - m * \bar{x}$$

$$\bar{x} = \sum_{i=0}^{n} \frac{x_i}{n}$$



Theory

LINEAR REGRESSION RECAP

Math is FUN 😭

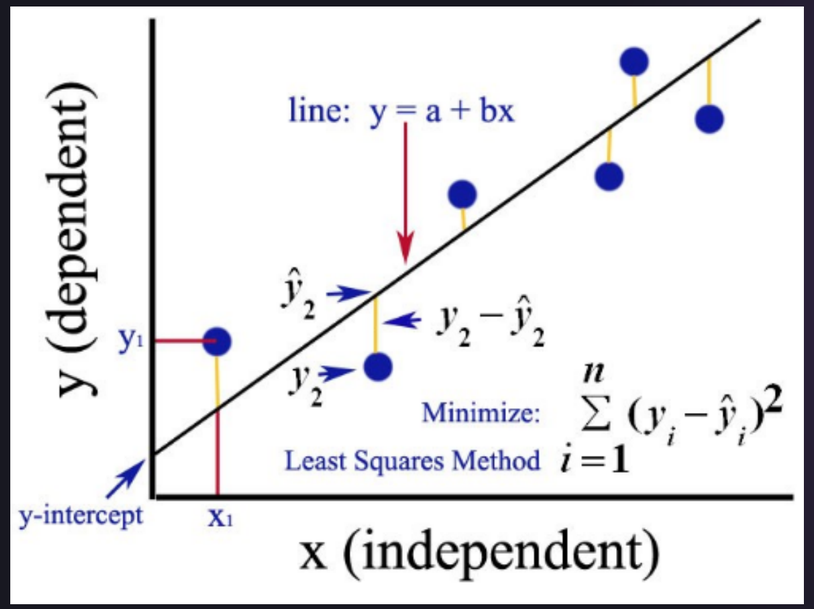
- Ordinary least squares

$$y = m * x + b$$

$$m = \frac{\sum_{i=0}^{n}(x_i - \bar{x}) * (y_i - \bar{y}))}{\sum_{i=0}^{n}(x_i - \bar{x})^2}$$

$$b = \bar{y} - m * \bar{x}$$

$$\bar{x} = \sum_{i=0}^{n} \frac{x_i}{n}$$



Theory

LINEAR REGRESSION RECAP

Code is very easy to implement

```
# Import the packages and classes needed in this example:
import numpy as np
from sklearn.linear_model import LinearRegression

# Create a numpy array of data:
temperature = np.array([0, 4, 35, 28, 21,24, 17, 22]).reshape((-1, 1))
icecream_sold = np.array([0, 1,250,150,90,115,40,99])

# Create an instance of a linear regression model and fit it to the data with the fit() function:
model = LinearRegression().fit(temperature,icecream_sold)

# The following section will get results by interpreting the created instance:
# Print the Intercept:
print('b:', model.intercept_)

# Print the Slope:
print('m:', model.coef_)
```

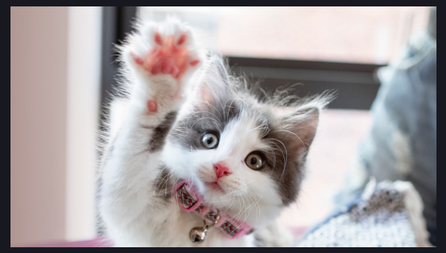
```
b: -32.075009716284484
m: [6.6331131]
```

Gradient Descent

- First-order iterative optimization algorithm for finding the local minimum of a differentiable function

Gradient Descent

Machine Learning Model



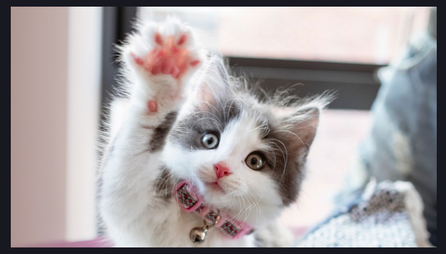
$$f\ (x)$$

"cat" or 1

"dog" or 0

Gradient Descent

First iteration

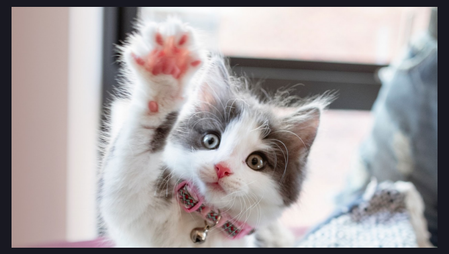


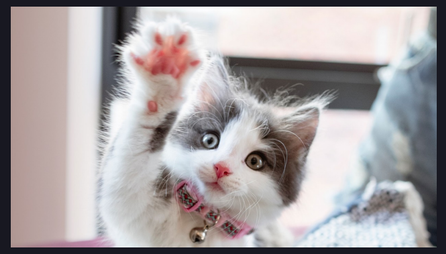
$$f\ (x)$$

0.6

Gradient Descent

First iteration

Remember  = 1



$$f\ (x)$$

0.6

How do we tell the model how well it's doing?

Loss function!

$$L\ (x)$$

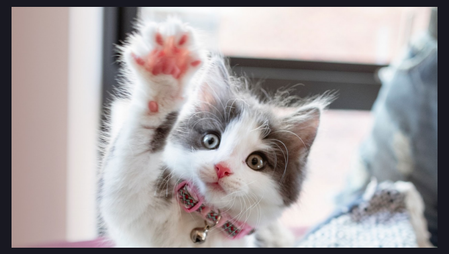
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

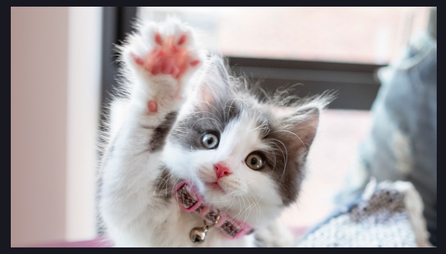
$$\mathrm{MAE} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{test set}} \underbrace{|y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i|}_{\text{actual value}}$$

$$
\begin{aligned}
LOSS = 1 - IoU + \frac{\rho^2\left(b, b^{gt}\right)}{c^2} + \alpha v - \\
\sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log\left(1 - C_i\right) \right] - \\
\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log(1 - C_i) \right] - \\
\sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in classes} \left[\hat{p}_i(c) \log(p_i(c)) + \left(1 - \hat{p}_i(c)\right) \log(1 - p_i(c)) \right]
\end{aligned}
$$

Gradient Descent

First iteration

Remember  = 1



$$f\ (x)$$

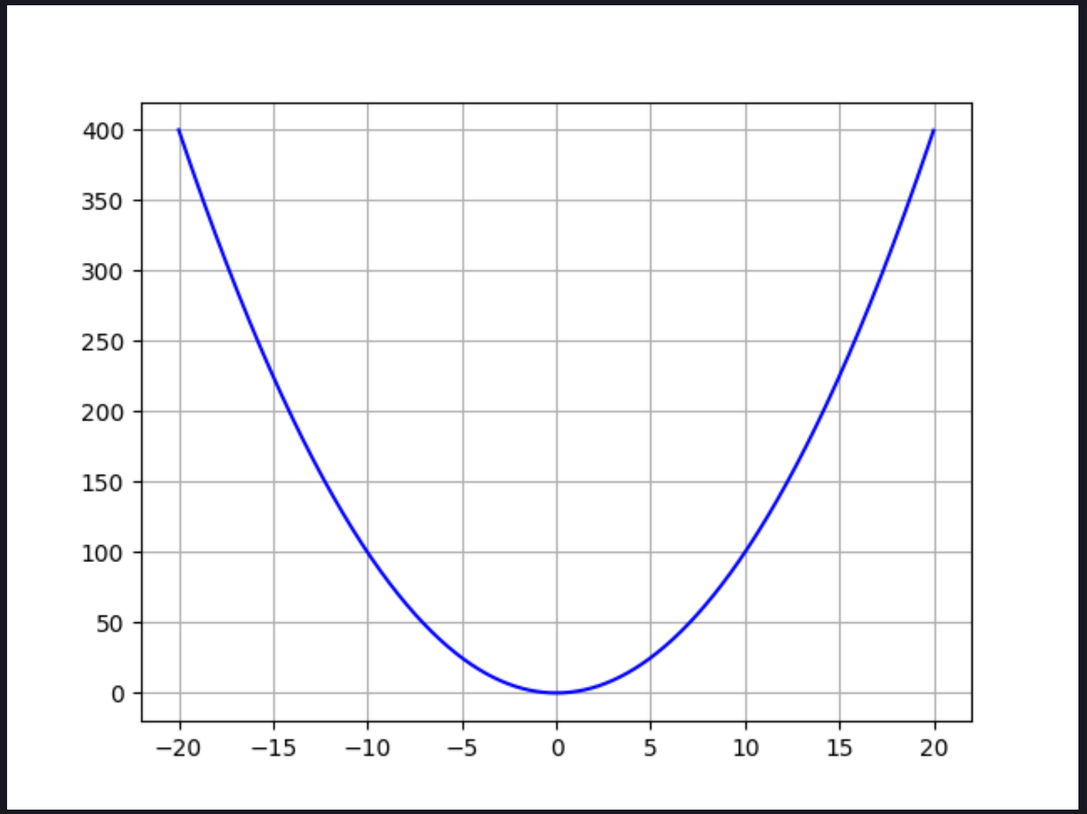
0.1 Bad!

1 Good!

Gradient Descent

Visualize the loss function

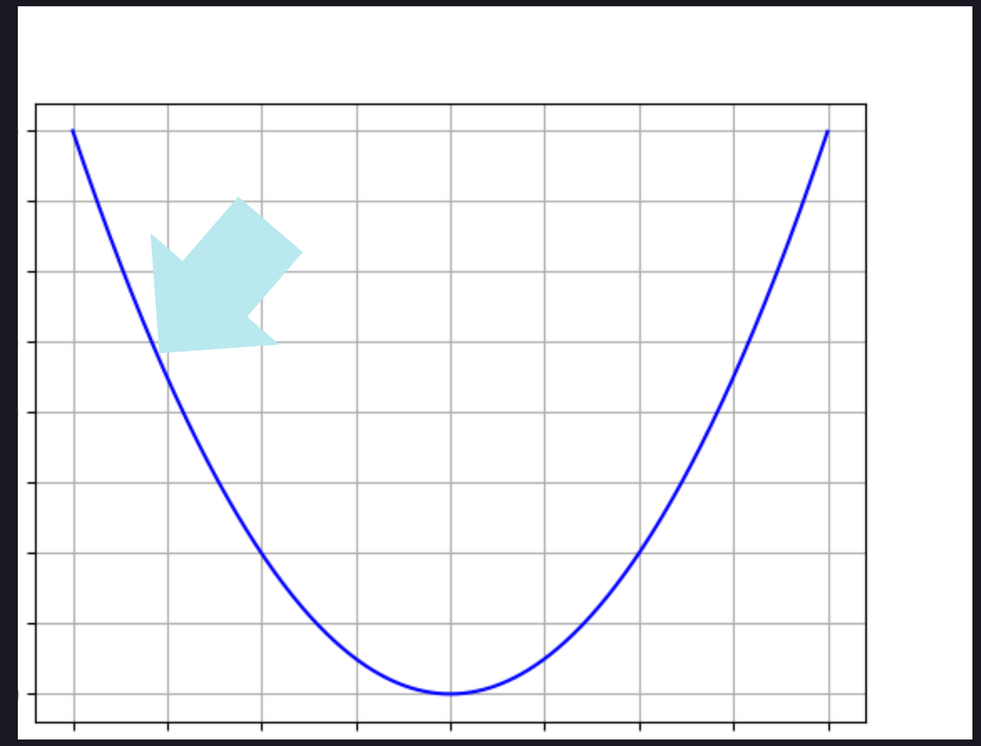
$$L(x)$$



Gradient Descent

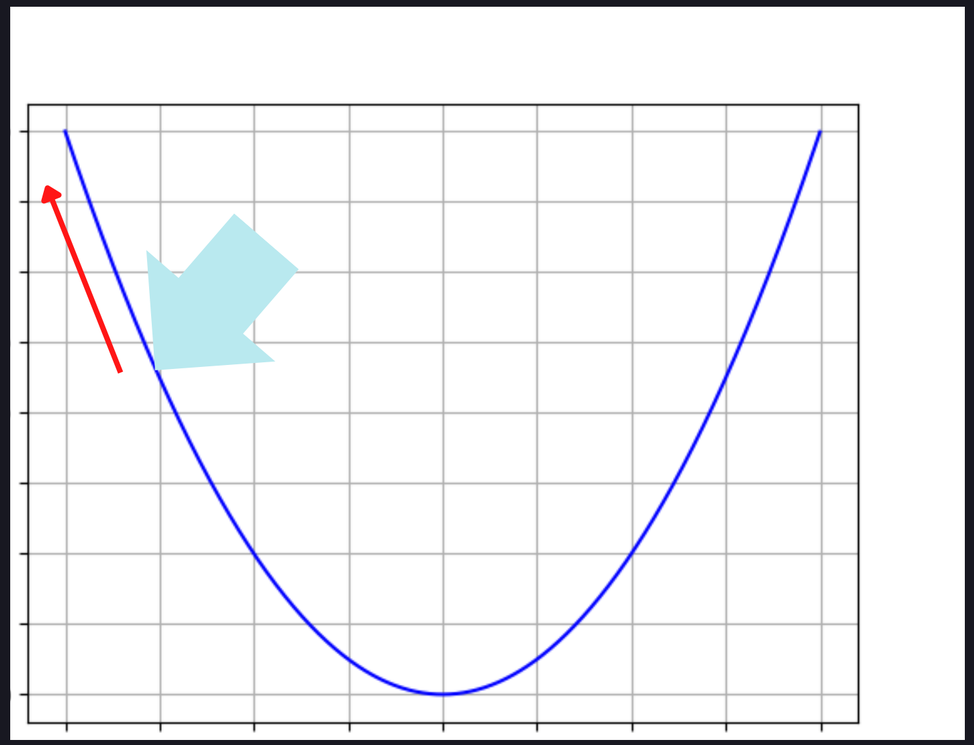
Visualize the loss function

$$L(x)$$



Gradient Descent

"direction and rate of fastest increase"

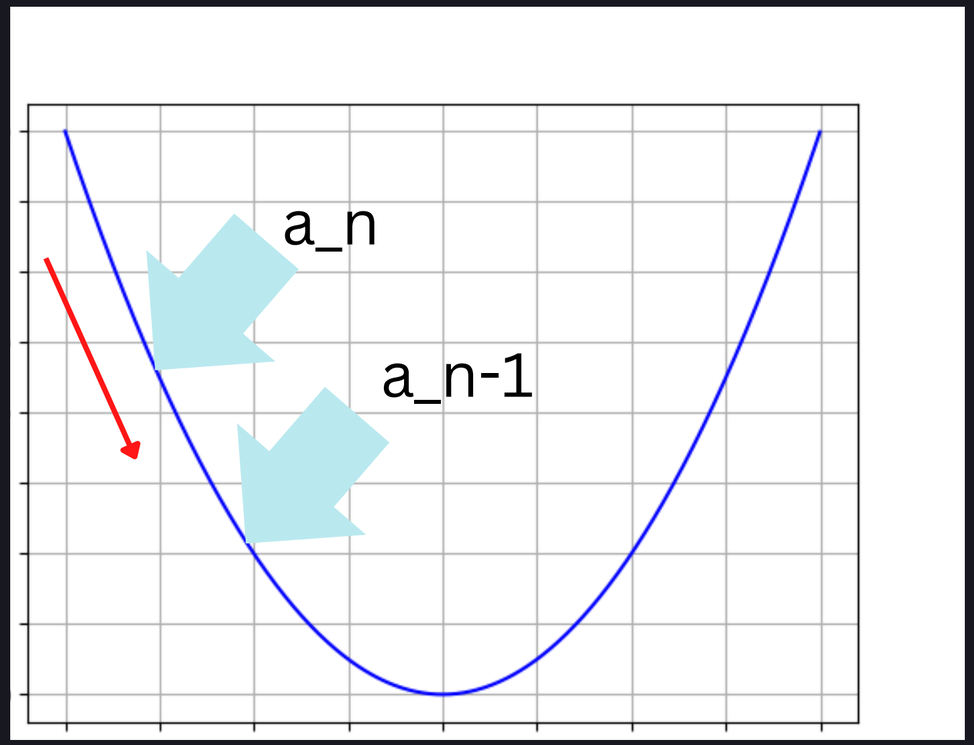


$$\nabla f$$

We want to go down!

Gradient Descent

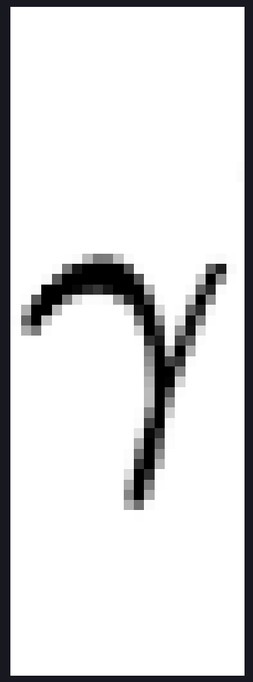
"direction and rate of fastest increase"



$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

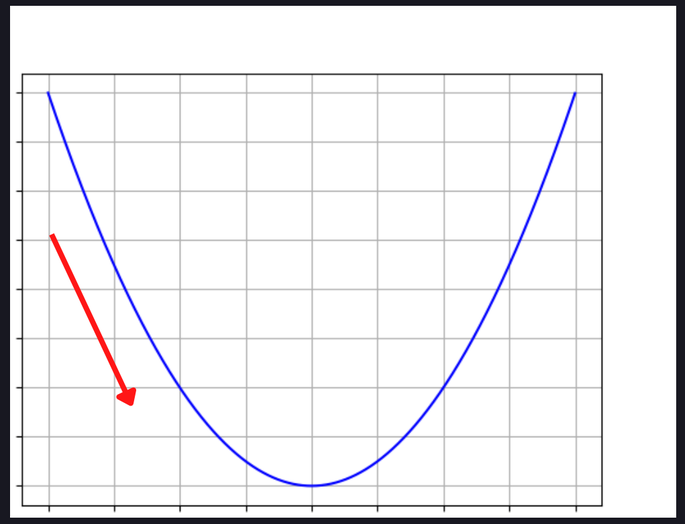
We want to go down!

But what is this?



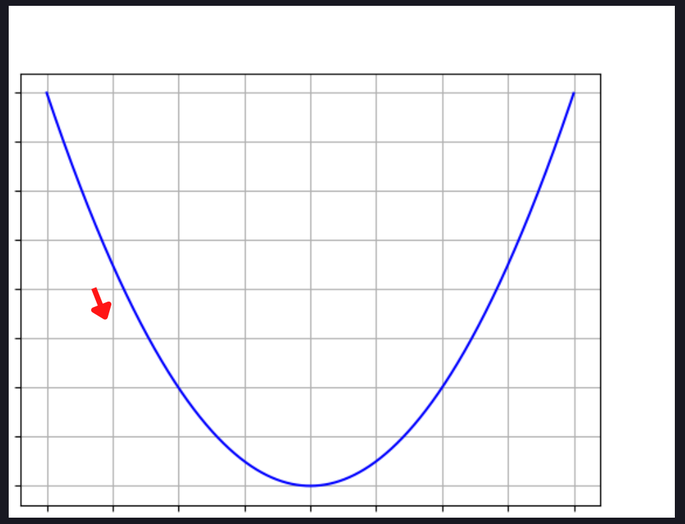
Learning Rate!

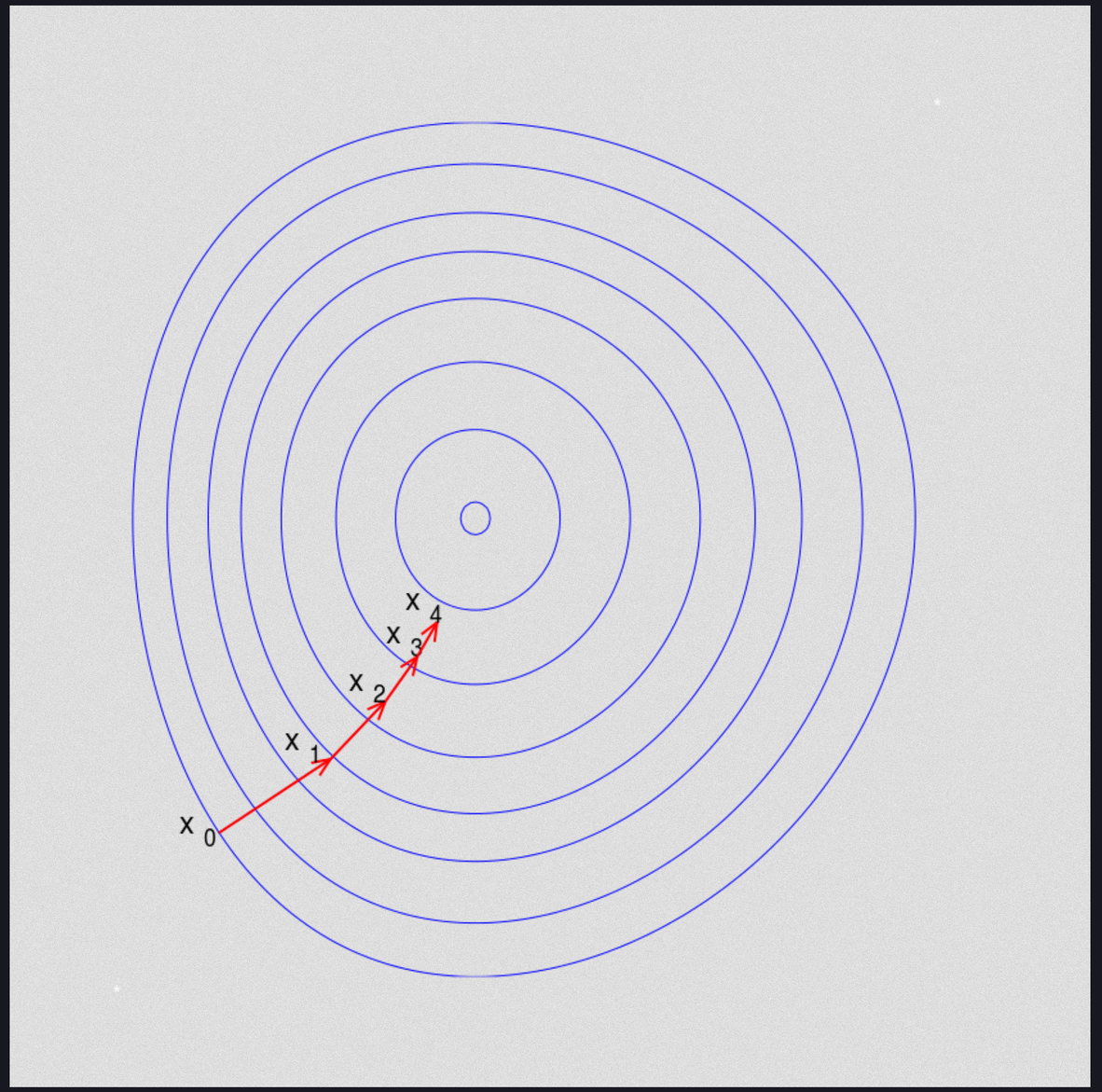
This...



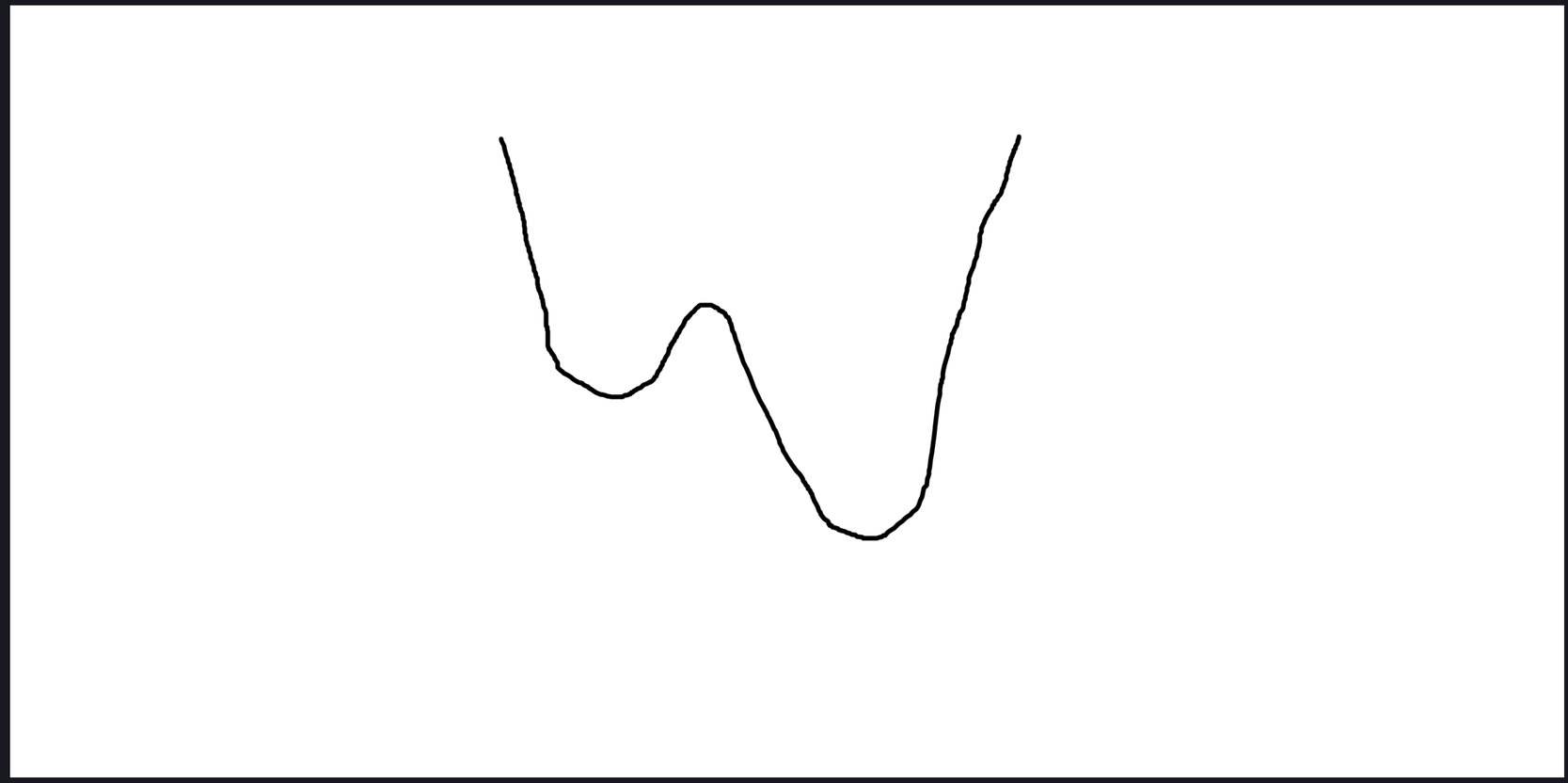
Not this...

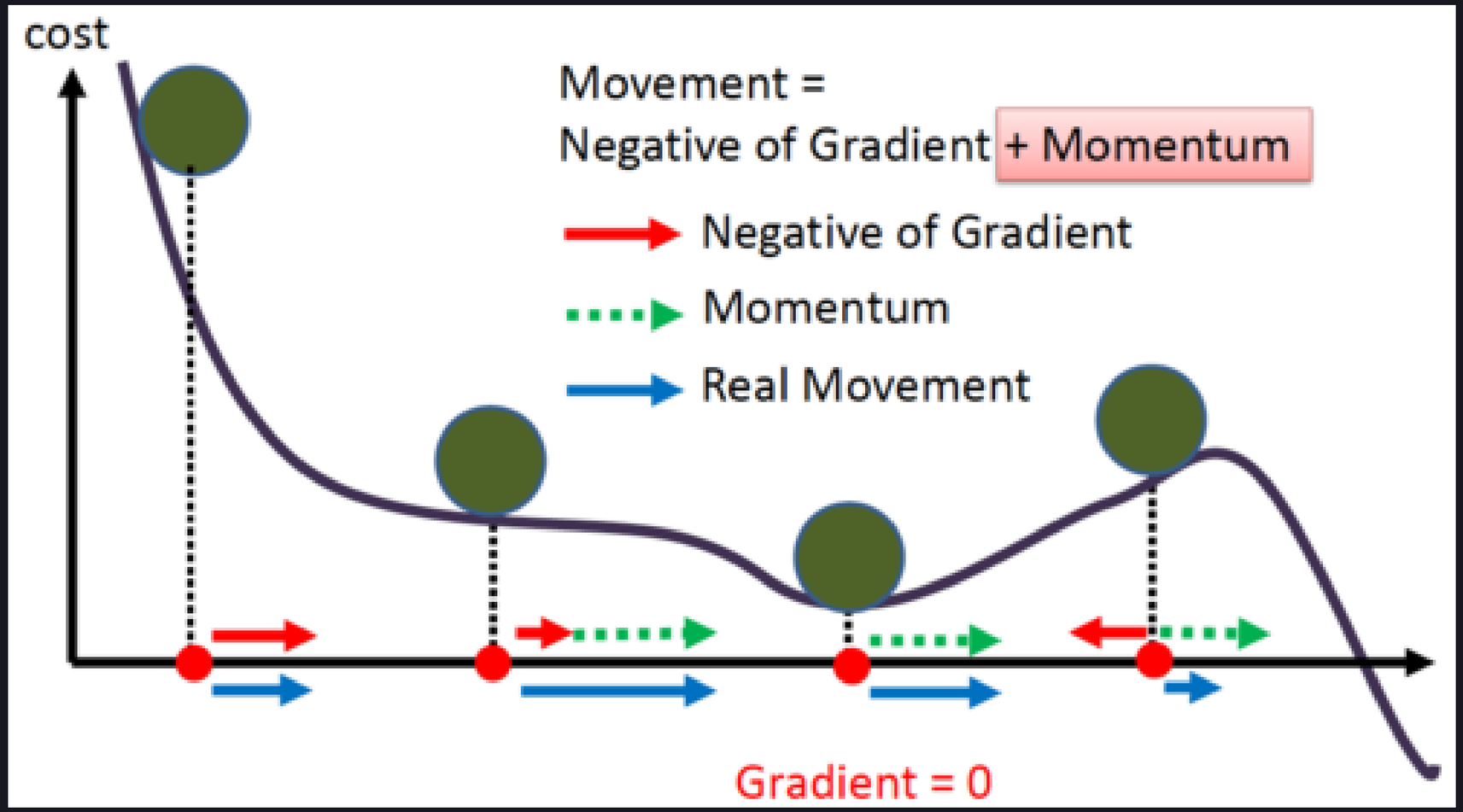
or...




x 0
x 1
x 2
x 3
x 4

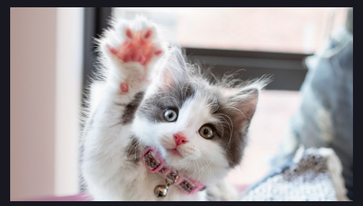
Problems with Gradient descent?

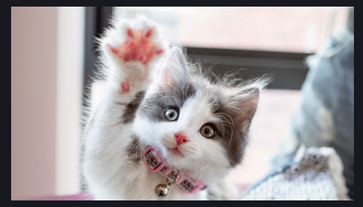



cost
Movement =
Negative of Gradient + Momentum
Negative of Gradient
Momentum
Real Movement
Gradient = 0

So far, what we have talked about is the training of the model

First iteration

Remember  = 1

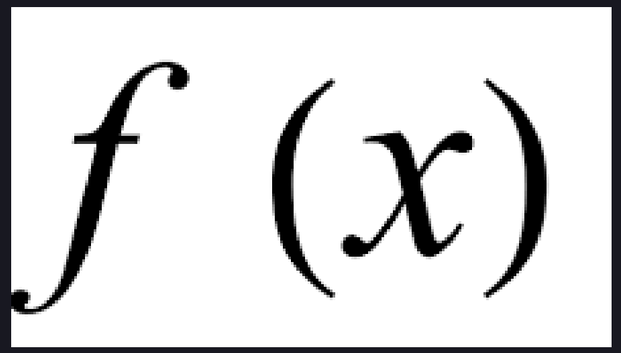


$f(x)$

0.1 Bad!

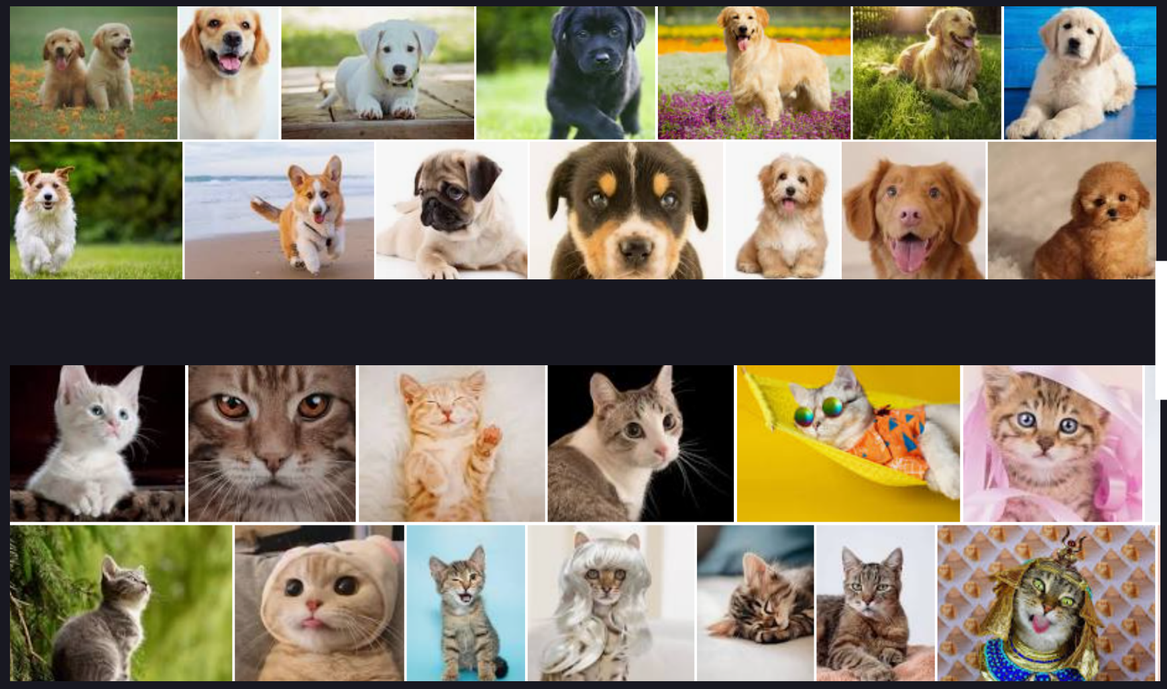
1 Good!

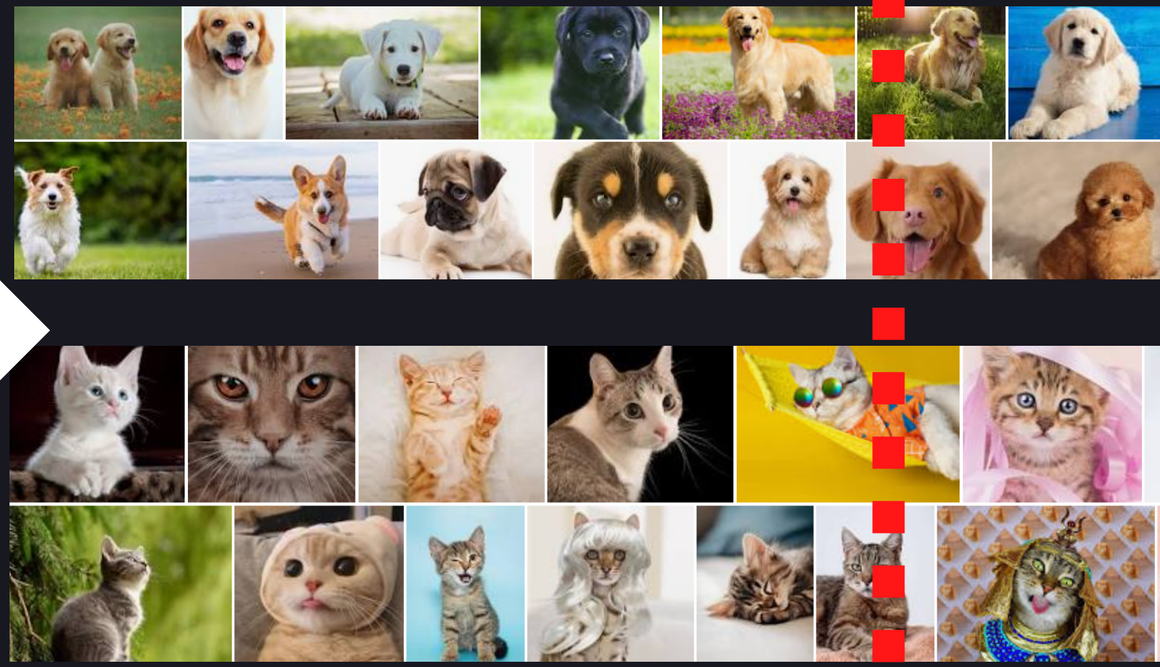
How do we test performance?



Accuracy = ?

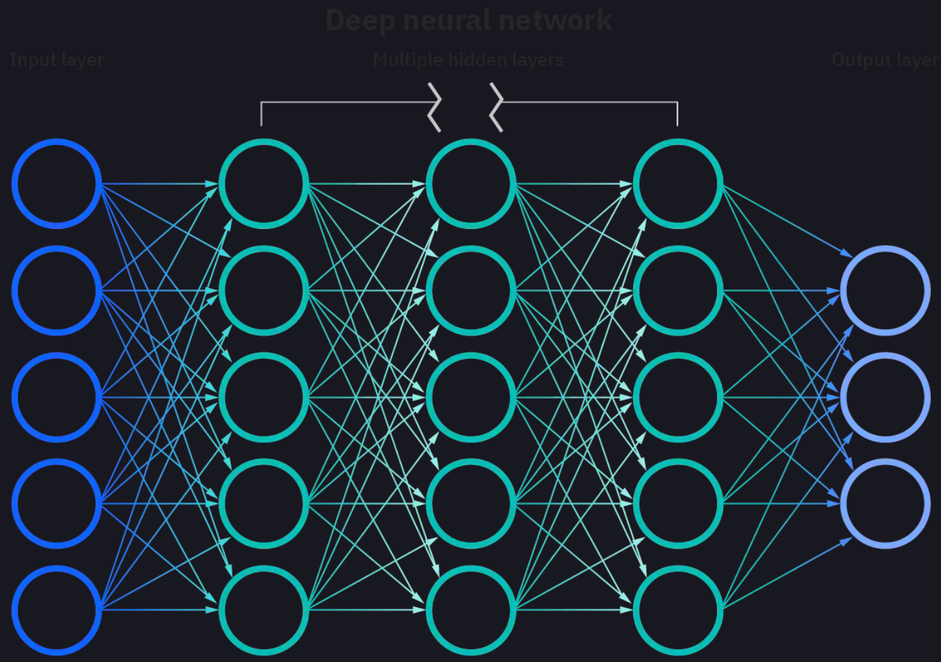
Train/Test split





Other considerations

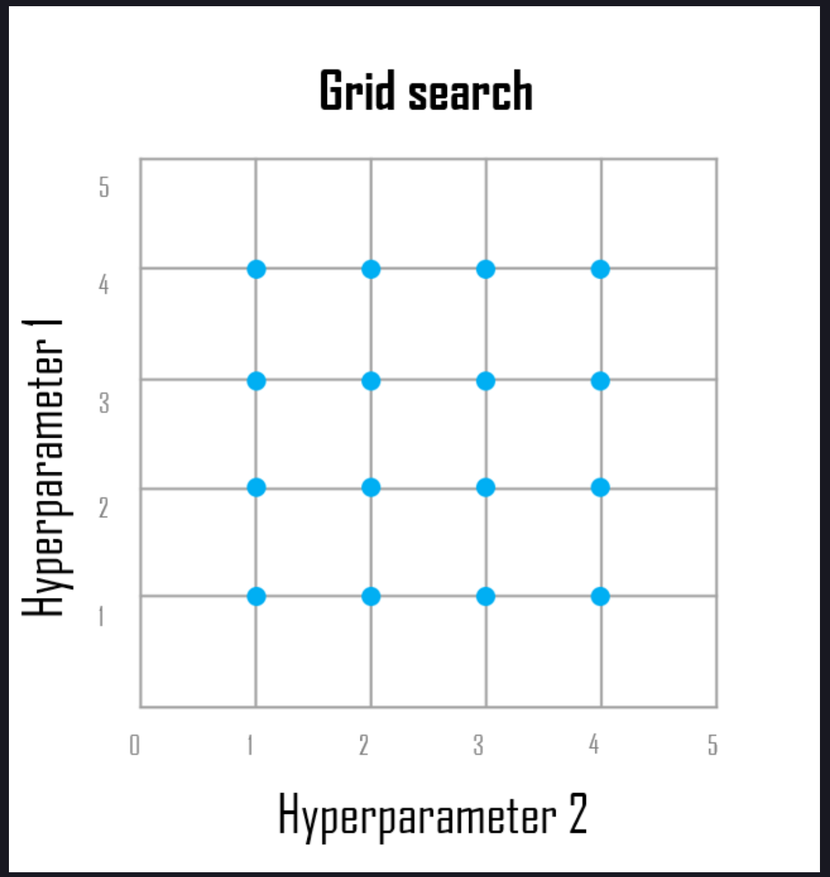
In machine learning, a hyperparameter is a parameter whose value is set before the learning process begins.



Do we just guess?

...Kinda

Grid Search

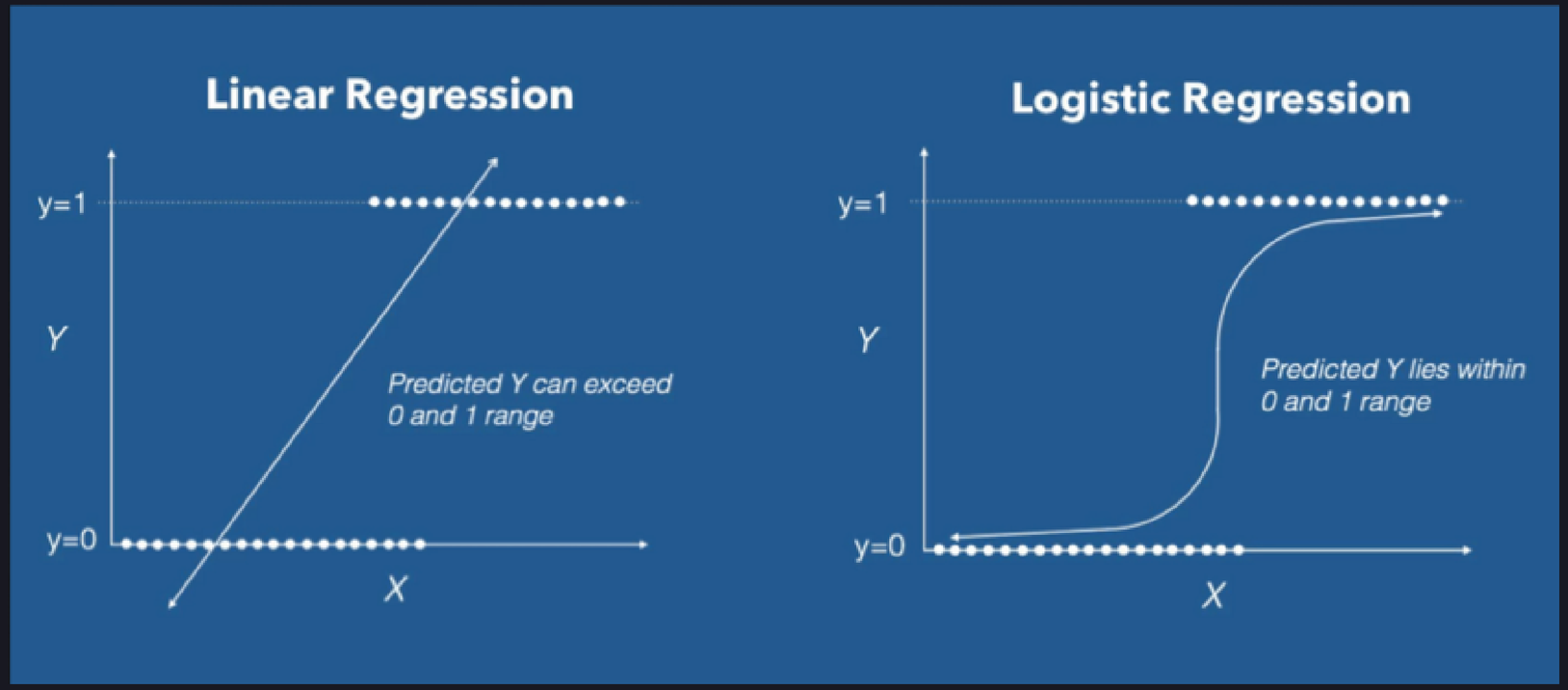


So what do we do for classification problems?

Theory

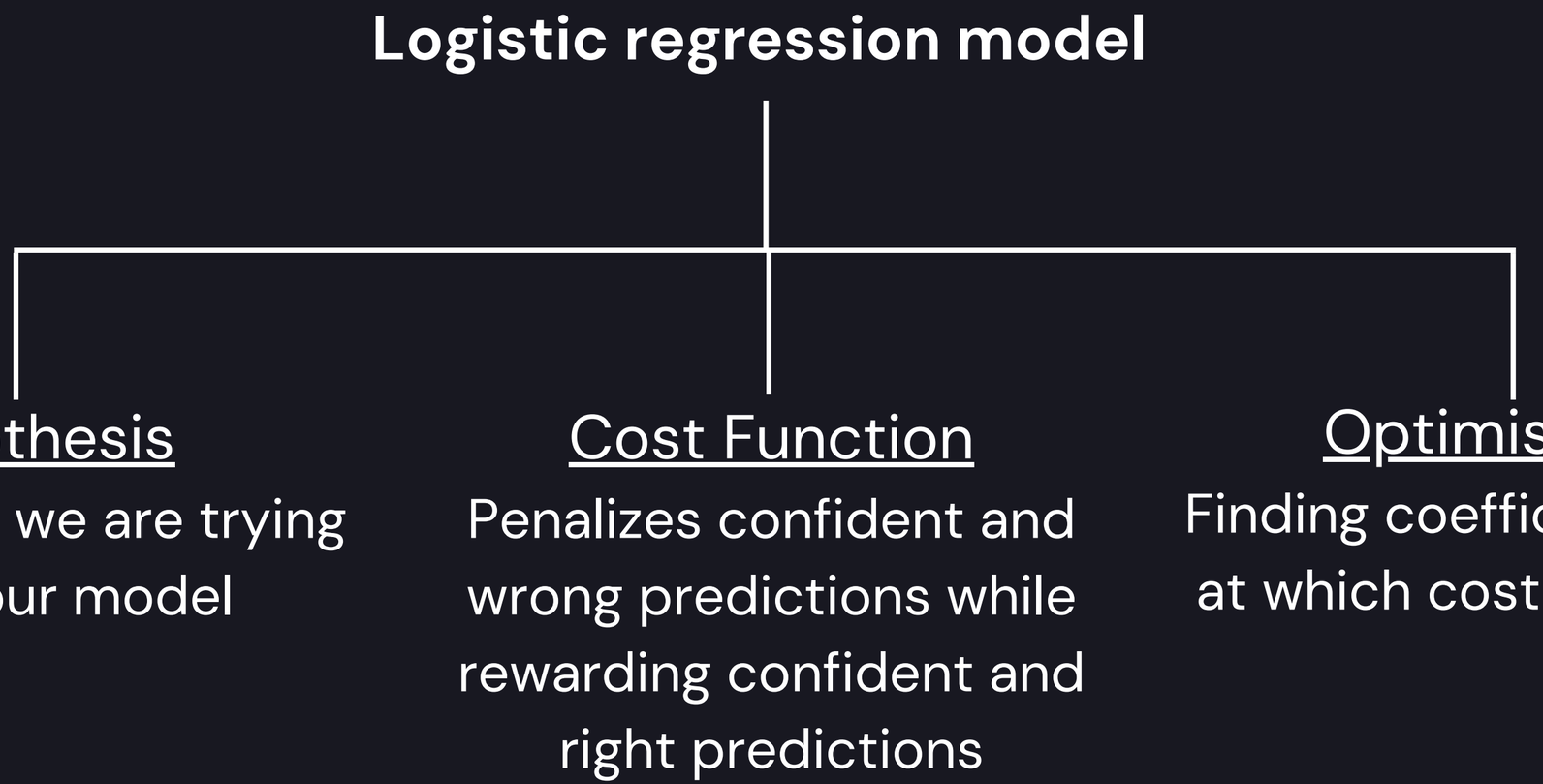
LOGISTIC REGRESSION

- Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring.



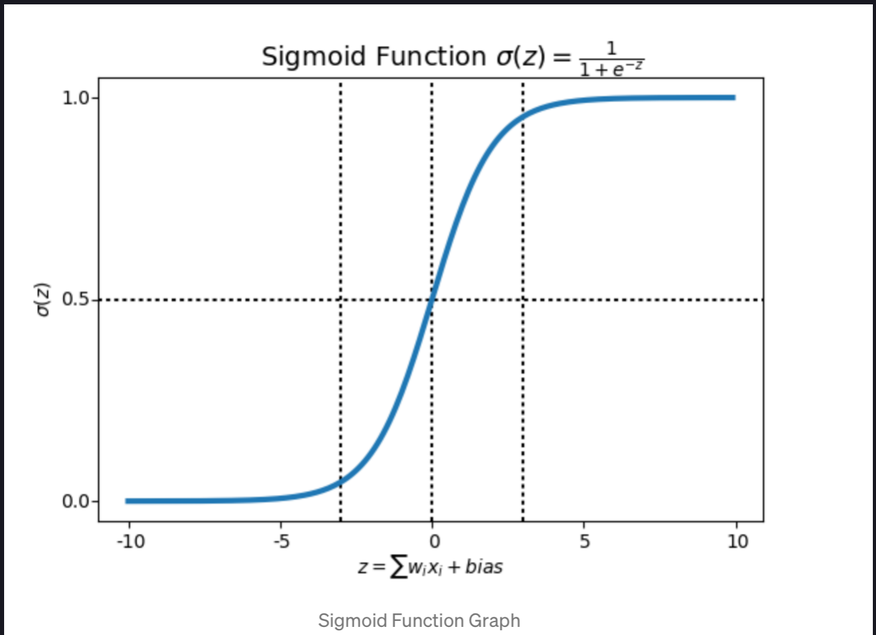
Theory

What makes a logistic regression model?



HYPOTHESIS

Sigmoid Function Graph



Sigmoid Function Graph

Logistic Regression Hypothesis :

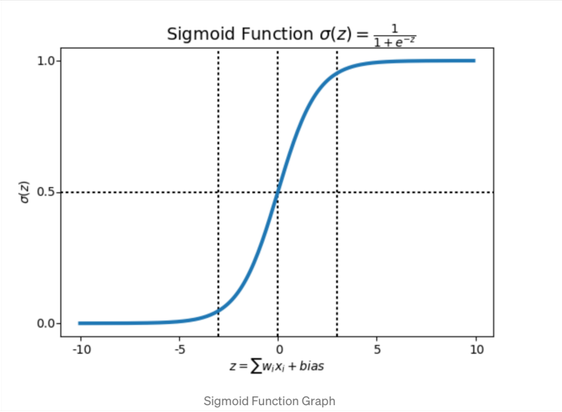
$$\sigma(Z) = \sigma(\beta_0 + \beta_1 X)$$

Expected Hypothesis :

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

HYPOTHESIS

Sigmoid Function Graph



Sigmoid Function Graph

Code Example

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

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

```python
import numpy as np

def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))
```

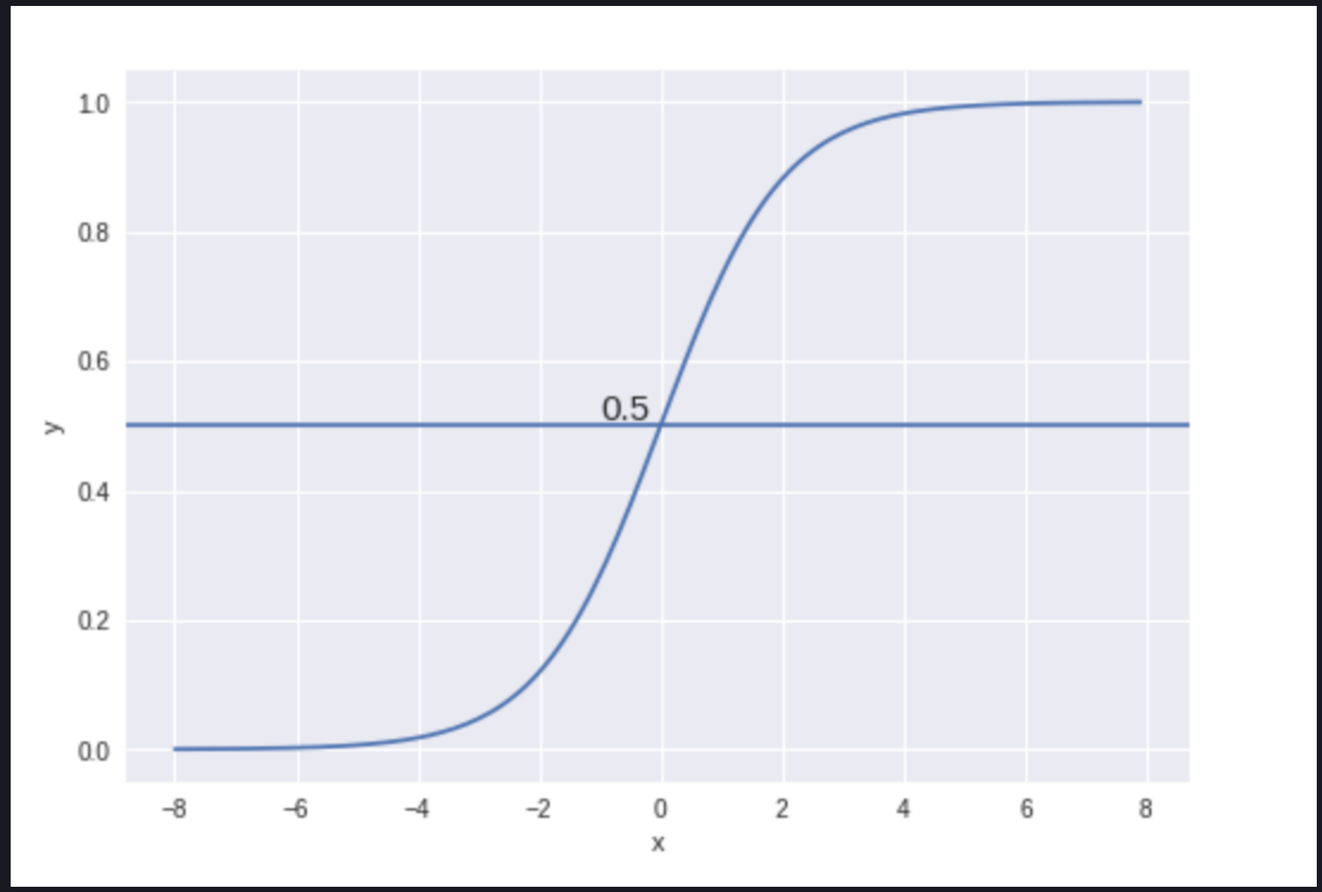
```python
def hypothesis(X, theta)
    z = np.dot(X, theta)
    y_hat = sigmoid(z)

    return y_hat
```

DECISION BOUNDARY

Example

1 — dog , 0 — cats

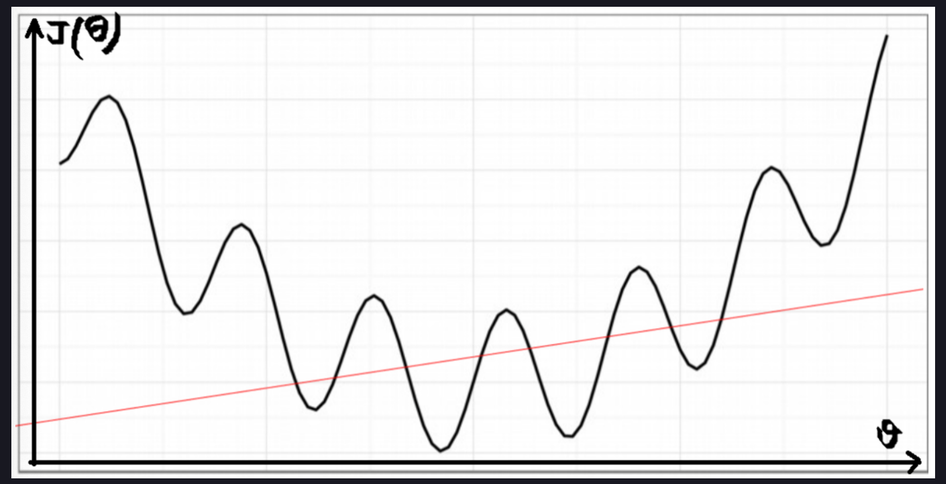


COST FUNCTION

Cost function of Linear regression

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Non-Convex Function



COST FUNCTION

For logistic regression, the Cost function is defined as:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Aboce functions compressed into one:

$$J(\theta) = -\frac{1}{m}\sum \left[y^{(i)} \log(h\theta(x(i))) + \left(1 - y^{(i)}\right) \log(1 - h\theta(x(i))) \right]$$

OPTIMIZATION

- Finding the optimal coefficients with the smallest cost
- Differentiating the cost function

$$\theta = \theta - lr.\frac{1}{m}X^T(\hat{y} - y)$$

```python
def gradient_descent(x_train, y_train, lr, epochs):
    intercept = np.ones((x_train.shape[0], 1))
    x_train = np.concatenate((intercept, x_train), axis=1)
    theta = np.zeros(x_train.shape[1])

    m = len(x_train)

    for i in range(epochs):
        y_hat = hypothesis(x_train, theta)
        gradient = np.dot(x_train.T, (y_hat - y_train)) / m

        theta -= lr * gradient

    return theta
```

Supervised Model

SUPPORT VECTOR MACHINE

A type of supervised model which is mainly used for binary or multi-class classification problem

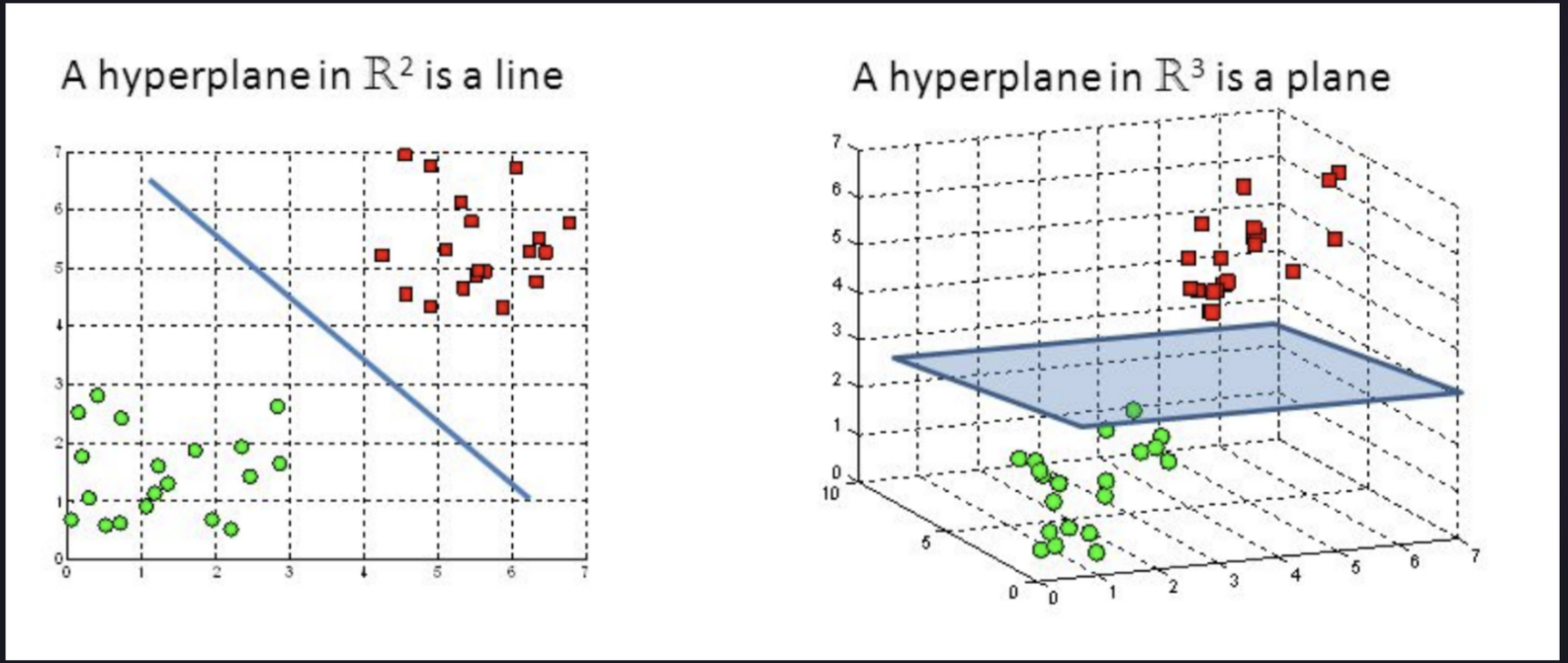
SPAM	Identifying email is spam or not
DISEASE	Identify a person has contracted a disease or not
PENGUIN	Classify the species of penguin
IRIS (FLOWER)	Classify the species of flower

HOW DOES IT WORK?

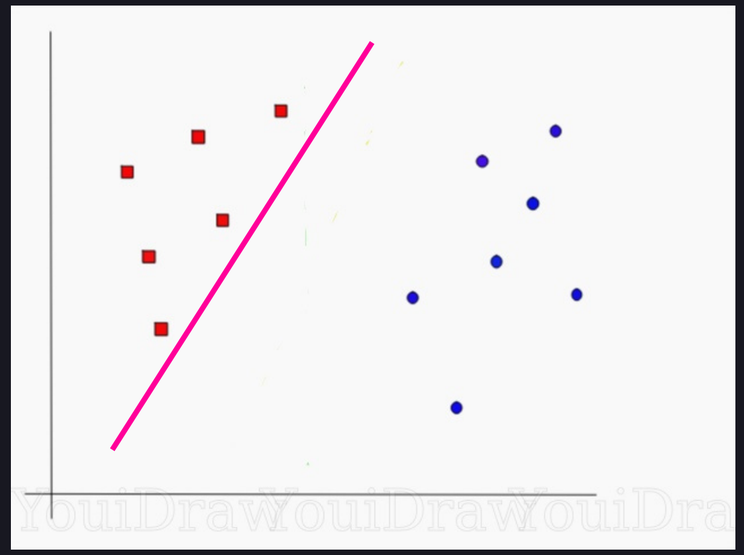
Finding a hyperplane that best separate data points into its corresponding classes

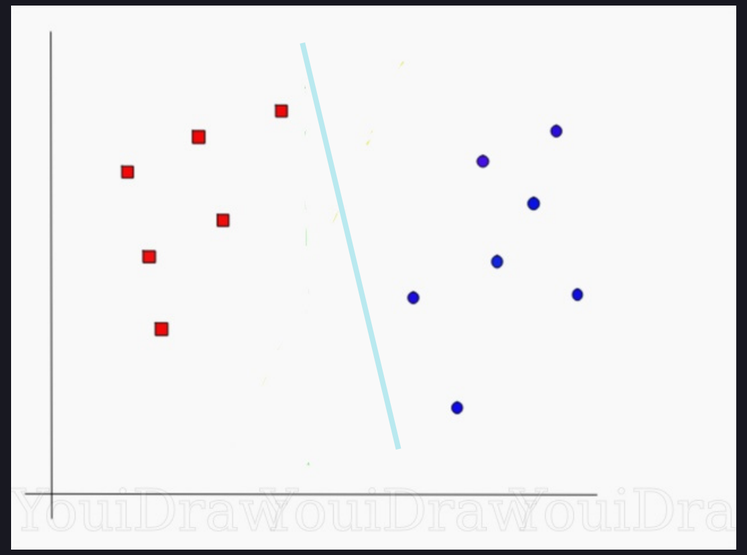
HYPERPLANE

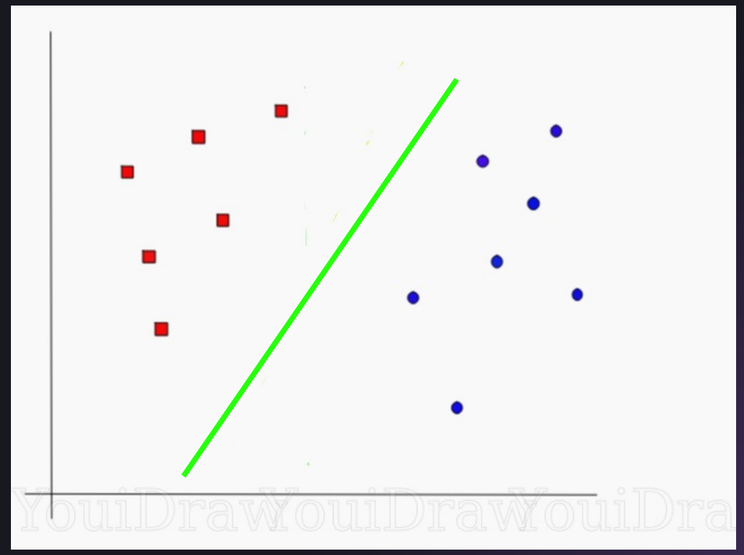
A plane whose dimension is one less than the dimension of data space



WHICH HYPERPLANE?

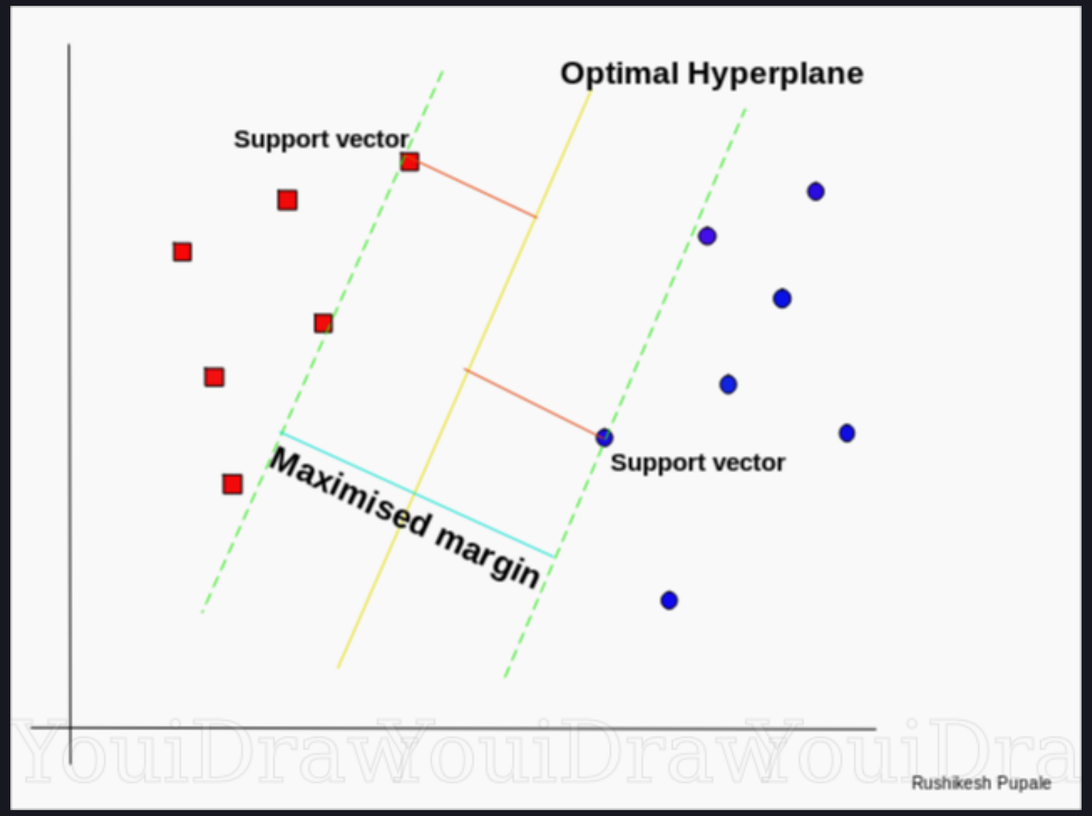






There are lots of hyperplane we can draw that is able to classify the datasets given correctly, but which one is the best?

WHAT IS CONSIDERED AS THE 'BEST' HYPERPLANE?



- Correctly classify all the red square into one class, and blue circle into another class
  - It creates separation with the maximum margin between variable classes

MATHEMATICS

Cost Function

$$J(w) = \frac{[w]^2}{2} + C[\frac{1}{N}\sum_i^n max(0, 1 - y_i(wx_1 + b))]$$

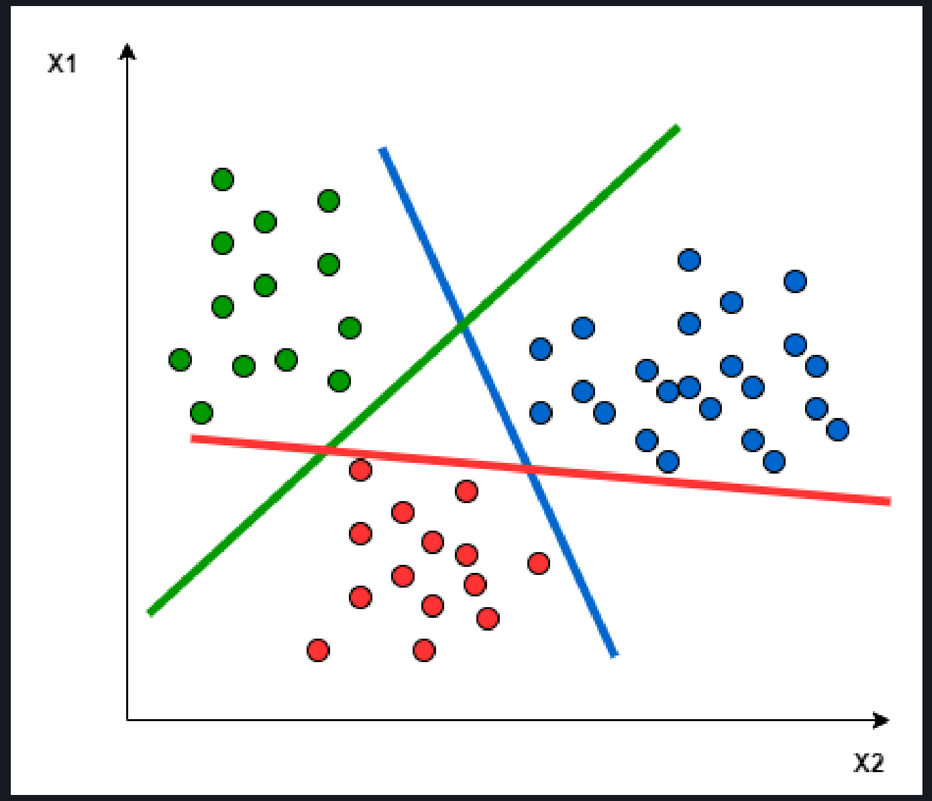
Margin width

Hinge Loss

Optimization (Gradient Descent)

- $\frac{1}{N}\sum_i^n w$ if $max(0, 1 - y_i * (wx_i)) = 0$
- $\frac{1}{N}\sum_i^n w - Cy_ix_i$ otherwise

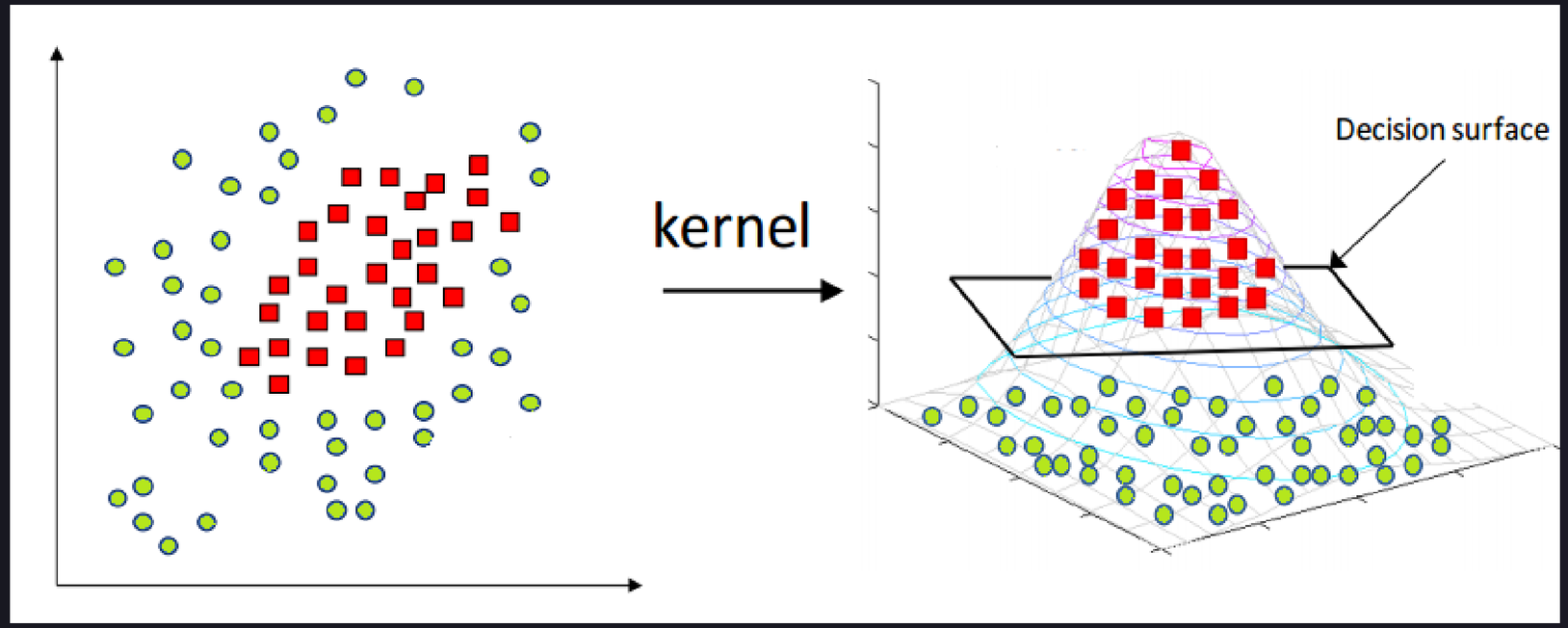
MULTI-CLASS PROBLEM



Just draw more hyperplane!

NON-LINEAR PATTERN

But, what if our data is not linearly separable?



1 - Map dataset into a relatively high-dimensional space

2 - Find a hyperplane that best separate data points into its classes

KERNEL

Kernel is a function that projects our data into a high-dimensional space. At a much deeper level, kernel tries to investigate the relationships that exist between each pair of observations (data)

POLYNOMIAL KERNEL

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} * \mathbf{y} + 1)^d$$

RADIAL BASIS FUNCTION KERNEL

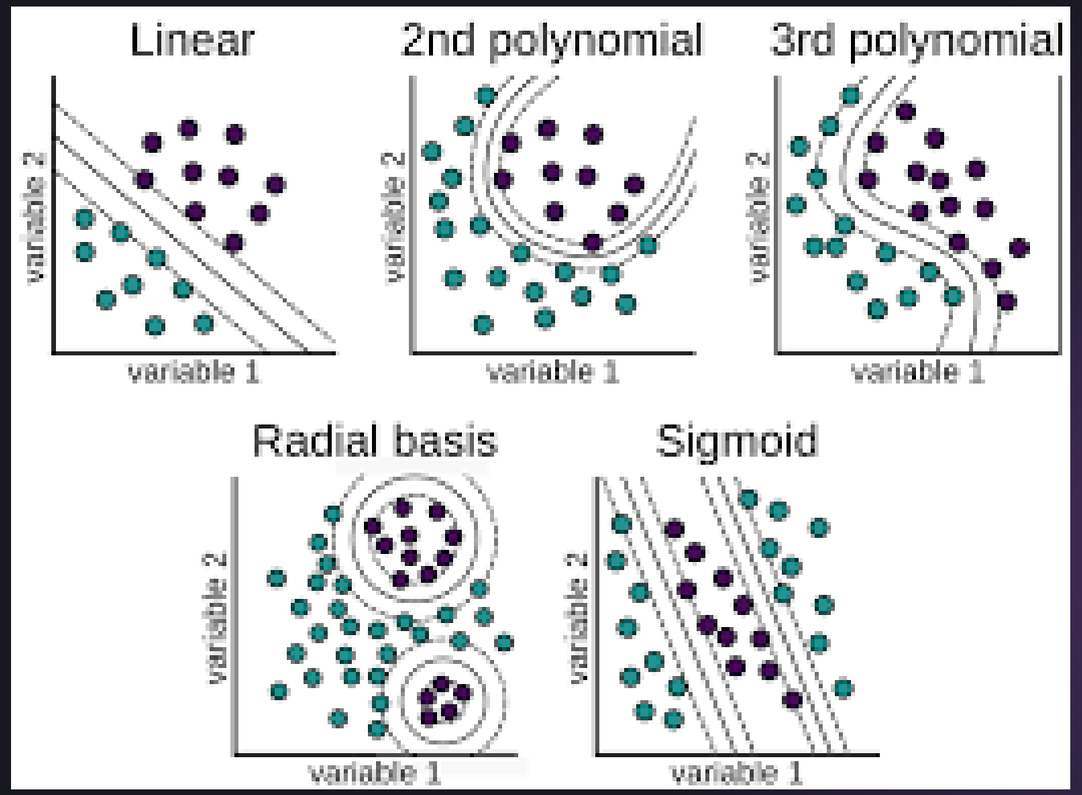
$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^2}, \gamma > 0$$

SKLEARN IMPLEMENTATION

```
from sklearn import svm

clf = svm.SVC(kernel = 'linear')
# clf = svm.SVC(kernel = 'poly', degree = 2)
# clf = svm.SVC(kernel = 'rbf')

clf.fit(X_train, y_train)
```

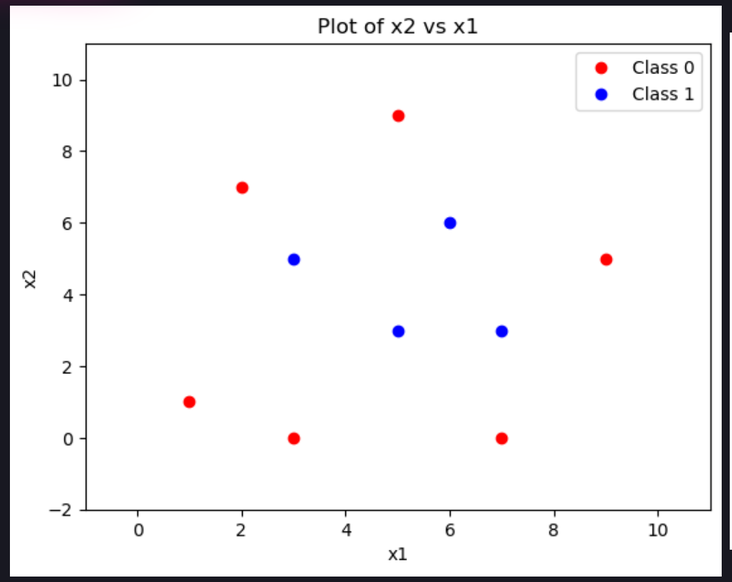


Supervised Model

DECISION TREE

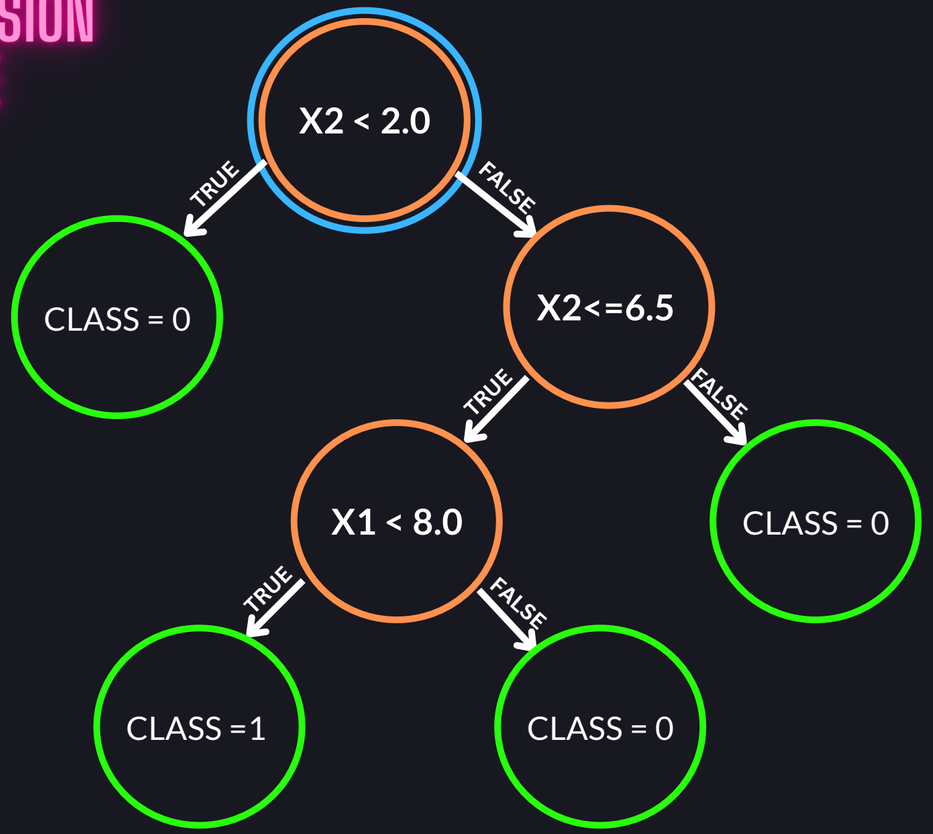
A tree-like model that can be used to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data(training data).

DATASET

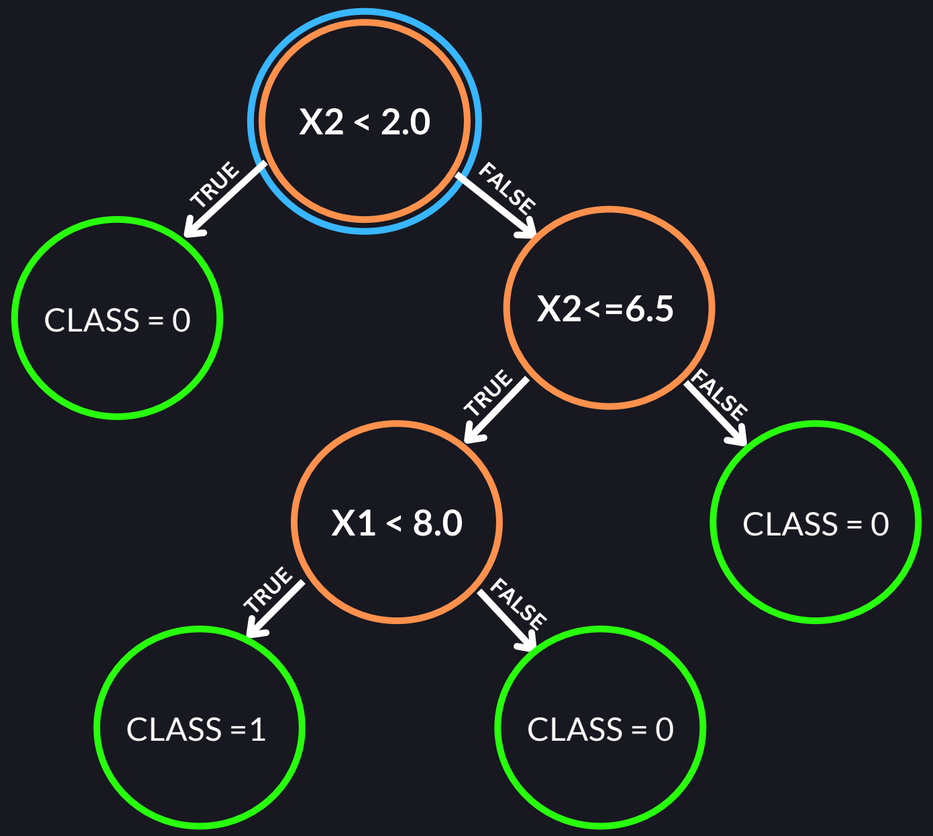


x1	x2	Class
1	1	0
3	0	0
7	0	0
9	5	0
5	9	0
2	7	0
3	5	1
5	3	1
6	6	1
7	3	1

DECISION TREE



DECISION TREES ANATOMY



ROOT NODE

- The node that starts the graph

DECISION NODE

- Nodes that show a decision to be made

LEAF NODE

- Nodes that don't split into more nodes.
- This node classifies which class a particular sample belongs to

BUILDING DECISION TREE

1. For each feature, find all possible decisions that we can make

x1	x2	Class
1	1	0
3	0	0
7	0	0
9	5	0
5	9	0
2	7	0
3	5	1
5	3	1
6	6	1
7	3	1

x1 feature:

1, 2, 3, 5, 6, 7, 9

Possible Decision (for x1):

- x1 < 1.5
- x1 < 2.5
- x1 < 4.0
- ...

x2 feature:

0, 1, 3, 5, 6, 7, 9

Possible Decision (for x2):

- x2 < 0.5
- x2 < 2.0
- x2 < 4.0
- x2 < 5.5
- ...

*Note that the middle point is chosen as the threshold for our decision

BUILDING DECISION TREE

2. Figure out the quality of each decision is by using either **Gini impurity**, **Entropy**, **Log-loss**, etc..

Gini impurity

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Entropy

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

x1	x2	Class
1	1	0
2	7	0
3	0	0
3	5	1
5	3	1
5	9	0
6	6	1
7	0	0
7	3	1
9	5	0

*x1 in ascending order

x1	x2	Class
3	0	0
7	0	0
1	1	0
5	3	1
7	3	1
3	5	1
9	5	0
6	6	1
2	7	0
5	9	0

*x2 in ascending order

BUILDING DECISION TREE

3. Select the best split as the current node
(right now because our tree is empty, it will be chosen as the root node)

x1	x2	Class
3	0	0
7	0	0
1	1	0
5	3	1
7	3	1
3	5	1
9	5	0
6	6	1
2	7	0
5	9	0

*x2 in ascending order

Now, we can proceed to the right child, and repeat the process all over again with an updated dataset

BUILDING DECISION TREE

SUMMARY

1. For each feature, find all possible decisions that we can make

2. Figure out what the gini index for each decision is
   (a measure of how good a decision/split is)

3. Pick the best decision (the one that has lowest gini index) as the current node

4. Traverse to the left child, start from STEP 1

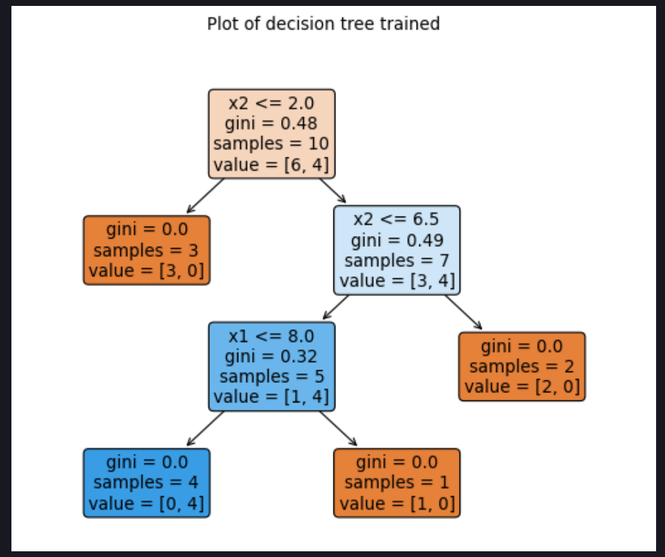
5. If the left child is a leaf, go to the right child, start from STEP 1

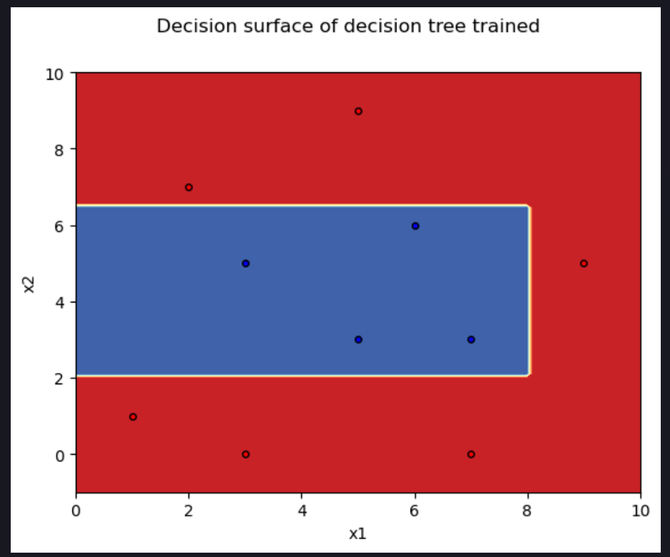
SKLEARN IMPLEMENTATION

```
from sklearn import tree
from sklearn.inspection import DecisionBoundaryDisplay

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)                                  # Train a decision tree based on training set 'X', and target class 'y'
tree.plot_tree(clf)                                  # Plot the decision tree
DecisionBoundaryDisplay.from_estimator(clf, X)       # Plot decision surface of decision trees

# For further information of what each function does, refer the documentation of Scikit-learn
```





BEAR IN MIND BOTH SVM AND DECISION TREE CAN GENERALIZE WELL INTO PROBLEM WITH MORE FEATURES (HIGHER DIMENSIONAL DATASET)

QUESTIONS?

SEE YOU NEXT TIME