



AI VIET NAM  
@aivietnam.edu.vn

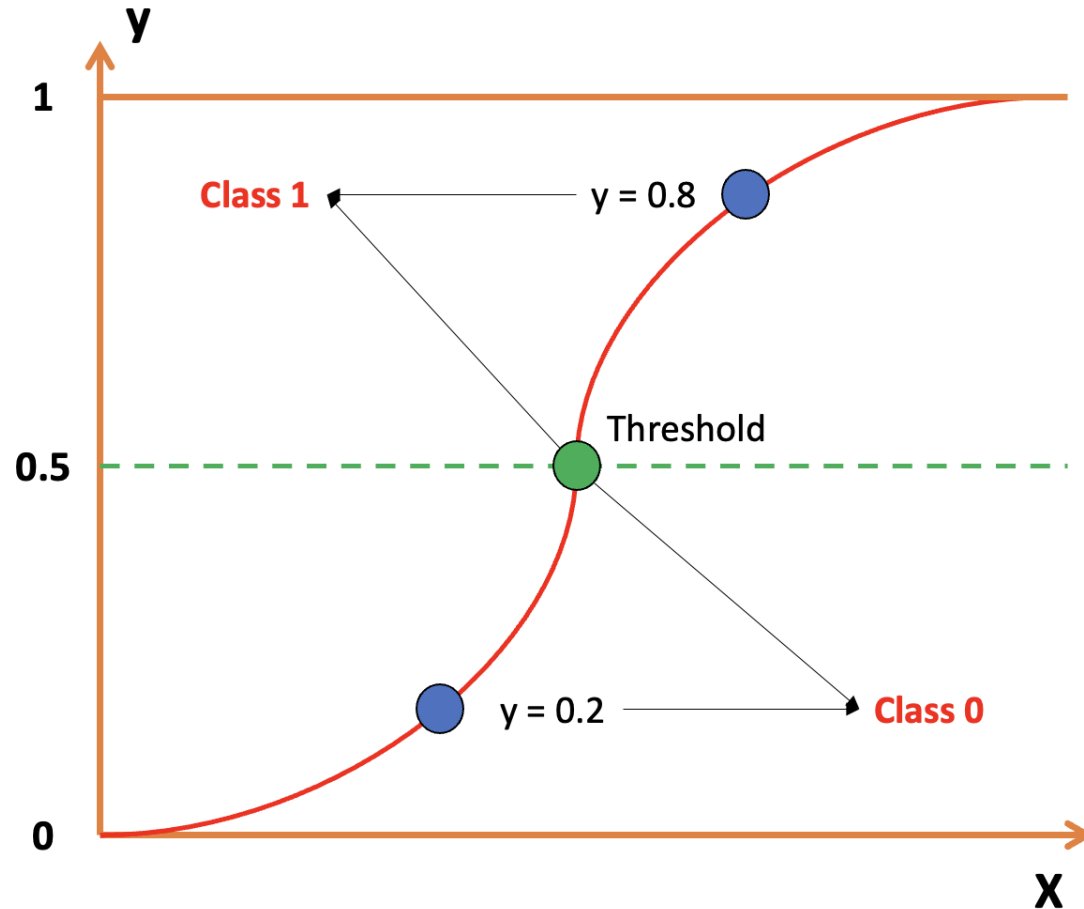
# Logistic Regression

## Exercise

Dinh-Thang Duong – TA

# Getting Started

## ❖ Objectives



### Our objectives:



- Review the Logistic Regression.
- Practice how to implement Logistic Regression with Titanic Survival Prediction and Sentiment Analysis problems.

# Outline

- **Review**
- **Titanic Survival Prediction**
- **Sentiment Analysis**
- **Question**

# Review

## ❖ Introduction

Positive	Negative
 <p>“Delicious food for an affordable price. Nice view. Fast service. We will definitely be visiting again.”</p>	 <p>“Food was terrible with unprofessional service. I will never revisit this restaurant again.”</p>

**Problem Statement:** Given a review, classify it into one of the three classes: Positive, Negative or Neutral.

**Input:** “Delicious food for an affordable price. Nice view. Fast service. We will definitely be visiting again.”

**Output:** “Positive”

**Sentiment Analysis Problem**

## ❖ Probabilistic Classifier

**Input**

“Delicious food for an affordable price. Nice view. Fast service. We will definitely be visiting again.”

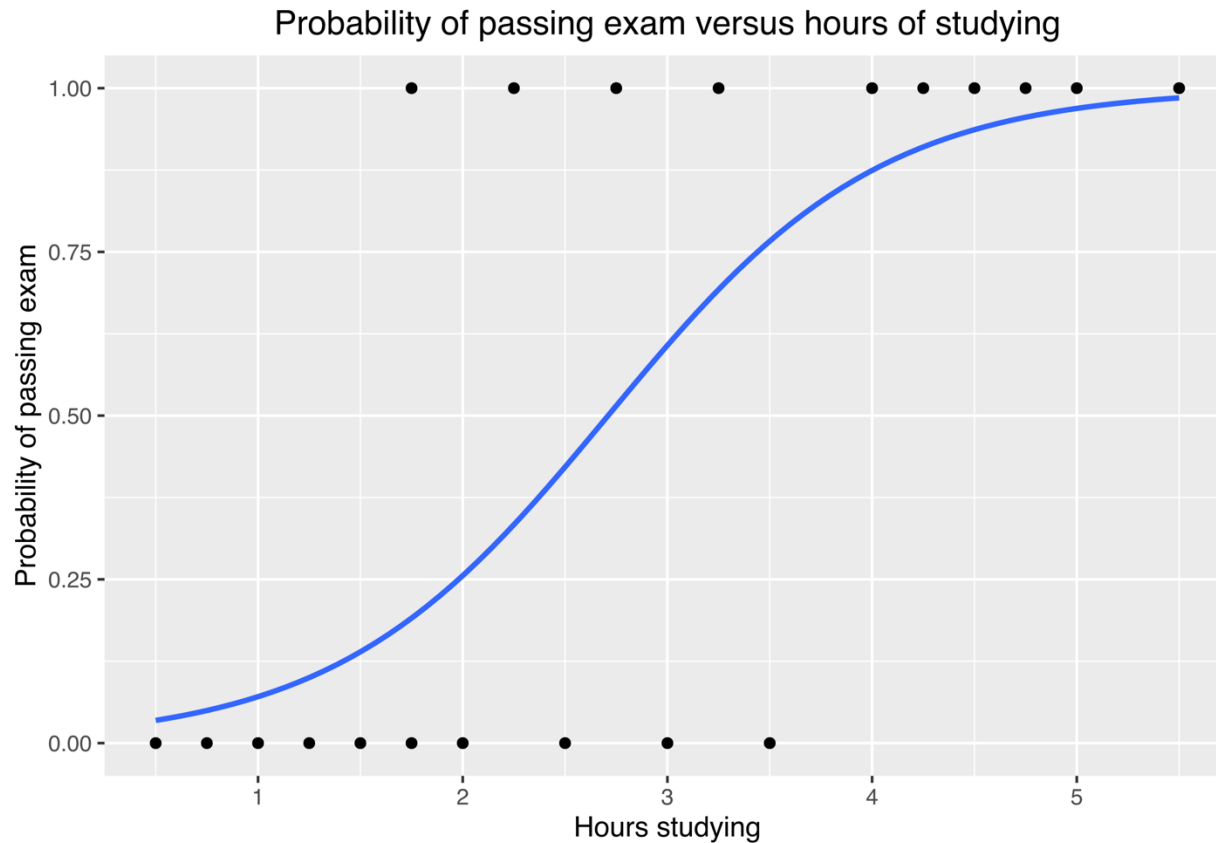
**Probabilistic Classifier**

$$\begin{aligned}P(y = 0|x) &= 0.3 \\P(y = 1|x) &= 0.7\end{aligned}$$

**Output**

Positive

## ❖ Logistic Regression



**Logistic Regression:** A statistical method used for binary classification and probability estimation. This is one of the fundamental tools in ML and statistics that models the relationship between one or more features and binary outcome variable.

## ❖ Logistic Regression

**Input**

“Delicious food for an affordable price. Nice view. Fast service. We will definitely be visiting again.”

**Logistic  
Regression**

**Output**

$$P(y = 1|x) = 0.7$$
$$P(y = 0|x) = 1 - P(y = 1|x)$$

Positive



# Review

## ❖ Logistic Regression

**Raw Text**

“Delicious food for an affordable price. Nice view. Fast service. We will definitely be visiting again.”



$x_1$   $x_2$   $x_3$  ...  $x_n$

**Features Vector Representation**

**Training Data**

X	y
$x^{(1)}$	$y^{(1)}$
$x^{(2)}$	$y^{(2)}$
$x^{(3)}$	$y^{(3)}$
...	...
$x^{(n)}$	$y^{(n)}$

**Features X**

$x_1$	$x_2$	$x_3$	...	$x_n$
-------	-------	-------	-----	-------



**Weights W**

$w_1$	$w_2$	$w_3$	...	$w_n$
-------	-------	-------	-----	-------



**The bias term**

$b$
-----

learning

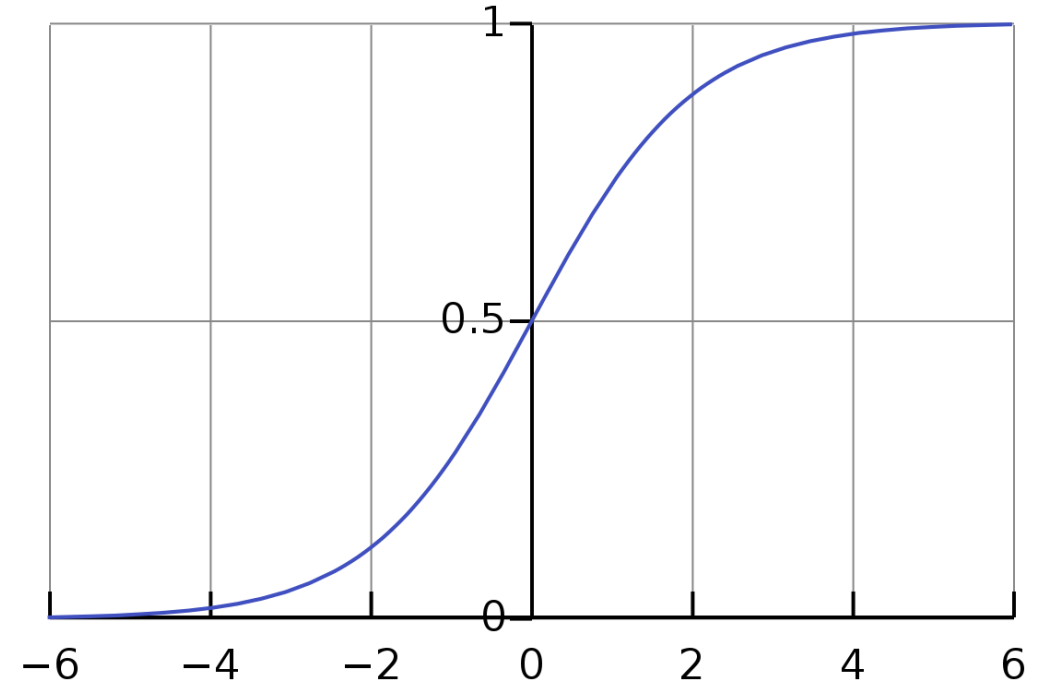
When make a decision: 
$$z = \sum_{i=1}^n w_i x_i + b = w \cdot x + b$$

## ❖ Logistic Regression: Sigmoid function

When make a decision:  $z = \sum_{i=1}^n w_i x_i + b = w \cdot x + b$

The value of this formula ranges from  $(-\infty, +\infty)$ .  
=> Not introduce probability.

To create probability, we will pass  $z$  to **sigmoid function**.



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

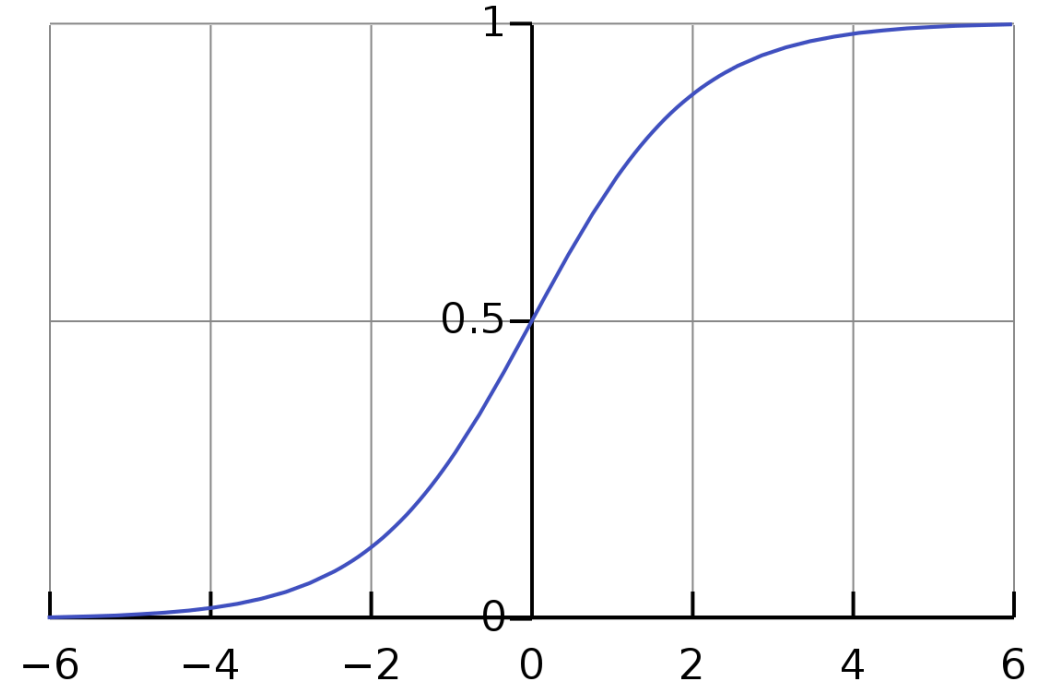
## ❖ Logistic Regression property

➤ We need to make sure the sum of two cases = 1:

- $z = \sum_{i=1}^n w_i x_i + b = w \cdot x + b$
- $P(y = 1) = \sigma(z) = \frac{1}{1 + \exp(-z)}$
- $P(y = 0) = 1 - \sigma(z) = 1 - \frac{1}{1 + \exp(-z)} = \frac{\exp(-z)}{1 + \exp(-z)}$

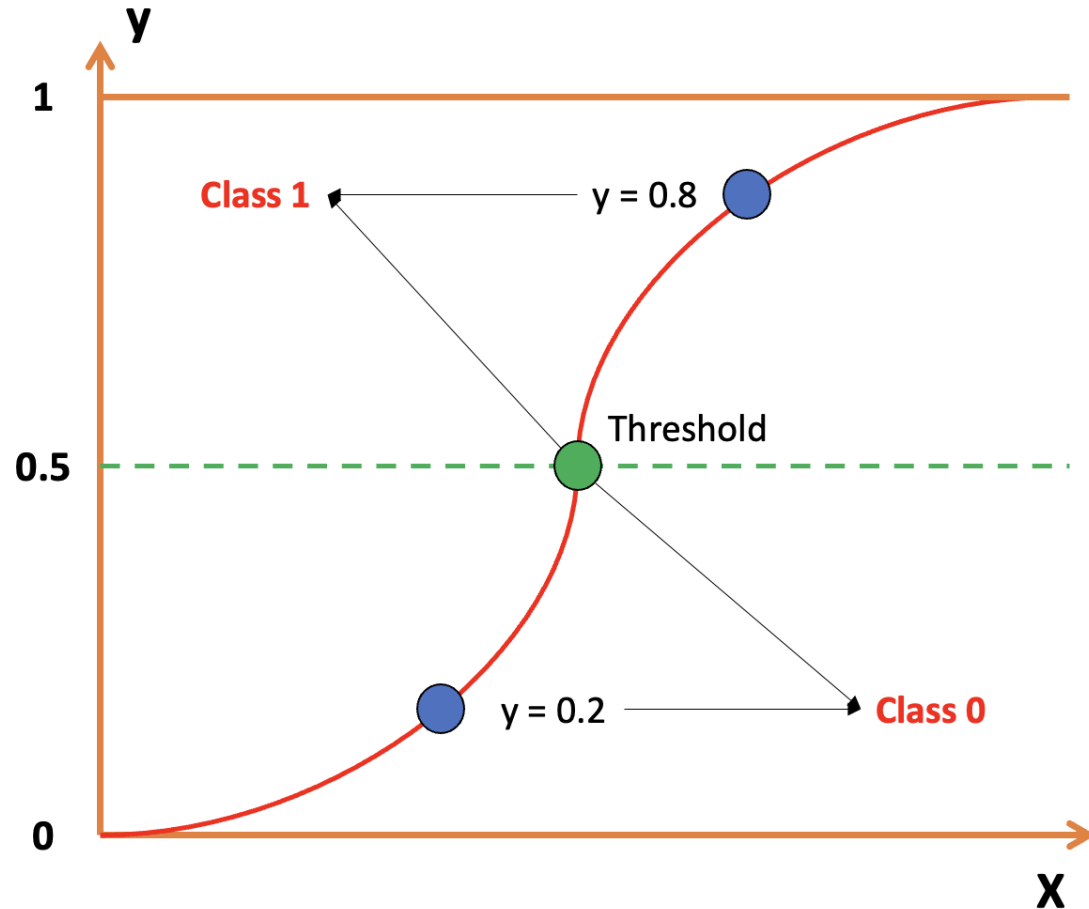
$$1 - \sigma(z) = \frac{\exp(-z)}{1 + \exp(-z)} = \frac{1}{\frac{1}{\exp(-z)} + 1} = \frac{1}{1 + \exp(z)} = \sigma(-z)$$

$\Rightarrow P(y = 0) = 1 - \sigma(z) = \sigma(-z)$



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

## ❖ Logistic Regression Prediction



➤ When making a prediction:

$$\text{predict}(x) = \begin{cases} 1 & \text{if } P(y = 1|x) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

## ❖ Logistic Regression loss function

- $z = \sum_{i=1}^n w_i x_i + b = w \cdot x + b$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$

Now we want to measure how different between  $\hat{y}$  (prediction) and  $y$  (true label)  $\Rightarrow L(\hat{y}, y)$

- $p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$  (Bernoulli Distribution)
- $\begin{aligned} \log(p(y|x)) &= \log(\hat{y}^y (1 - \hat{y})^{1-y}) \\ &= \log(\hat{y}^y) + \log((1 - \hat{y})^{1-y}) \\ &= y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \end{aligned}$
- $L(\hat{y}, y) = -\log(p(y|x)) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$   
 $\Rightarrow$  **Cross-entropy loss.**

**Case 1 (Correct prediction):**  $\hat{y} = 0.7$  and  $y = 1$

$$\begin{aligned} L(\hat{y}, y) &= -\log(p(y|x)) \\ &= -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \\ &= -\log(\hat{y}) = -\log(0.7) = 0.36 \end{aligned}$$

**Case 2 (Wrong prediction):**  $\hat{y} = 0.7$  and  $y = 0$

$$\begin{aligned} L(\hat{y}, y) &= -\log(p(y|x)) \\ &= -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \\ &= -\log(1 - \hat{y}) = -\log(0.3) = 1.2 \end{aligned}$$

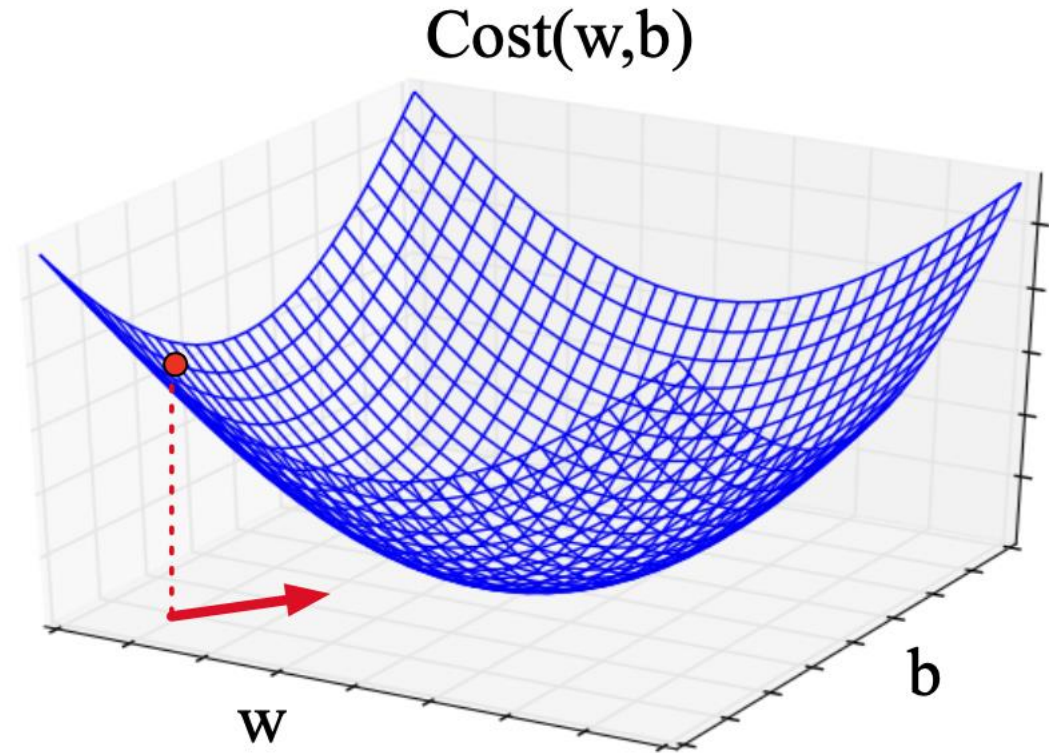
## ❖ Logistic Regression objective

Our objective is to find the optimal weights and bias (theta) that minimize the loss function:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L(\hat{y}^i, y^i)$$

For each step, compute the gradient and update the current theta:

- $\nabla_{\theta_s} L = \frac{1}{N} X^{iT} (\hat{y}^i - y^i)$
- $\theta_{s+1} = \theta_s - \eta \nabla_{\theta_s} L$



Gradient Descent

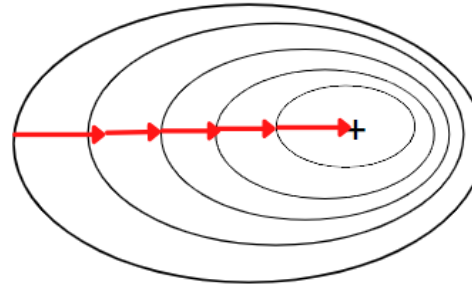
## ❖ Logistic Regression objective

For each step, compute the gradient and update the current theta:

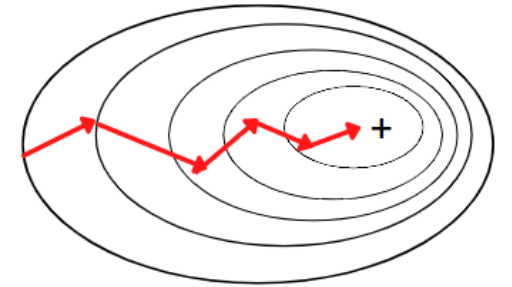
- $\nabla_{\theta_s} L = \frac{1}{N} X^i{}^T (\hat{y}^i - y^i)$
- $\theta_{s+1} = \theta_s - \eta \nabla_{\theta_s} L$

- **Stochastic GD:** batch\_size = 1
- **Mini-Batch GD:**  $1 < \text{batch\_size} < N$
- **Batch GD:** batch\_size = N

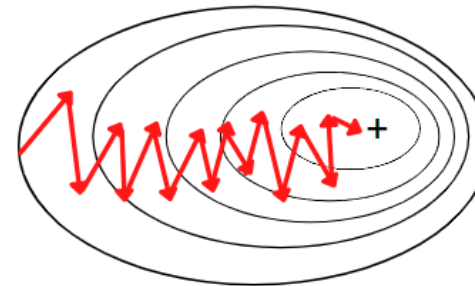
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent



# Review

## ❖ Training Logistic Regression

→ 1) Pick a sample  $(\mathbf{x}, y)$  from training data



2) Compute output  $\hat{y}$



$$z = \boldsymbol{\theta}^T \mathbf{x} = \mathbf{x}^T \boldsymbol{\theta} \quad \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

3) Compute loss



$$L(\hat{y}, y) = (-y \log \hat{y} - (1 - y) \log (1 - \hat{y}))$$

4) Compute derivative



$$\nabla_{\boldsymbol{\theta}} L = \mathbf{x}(\hat{y} - y)$$

→ 5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L$$

$\eta$  is learning rate

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

```
def predict(X, theta):  
    return sigmoid_function( np.dot(X.T, theta) )
```

```
def loss_function(y_hat, y):  
    return -y*np.log(y_hat) - (1 - y)*np.log(1 - y_hat)
```

```
def compute_gradient(X, y_hat, y):  
    return X*(y_hat - y)
```

```
def update(theta, lr, gradient):  
    return theta - lr*gradient
```

```
# compute output
```

```
y_hat = predict(X, theta)
```

```
# compute loss
```

```
loss = loss_function(y_hat, y)
```

```
# compute mean of gradient
```

```
gradient = compute_gradient(X, y_hat, y)
```

```
# update
```

```
theta = update(theta, lr, gradient)
```

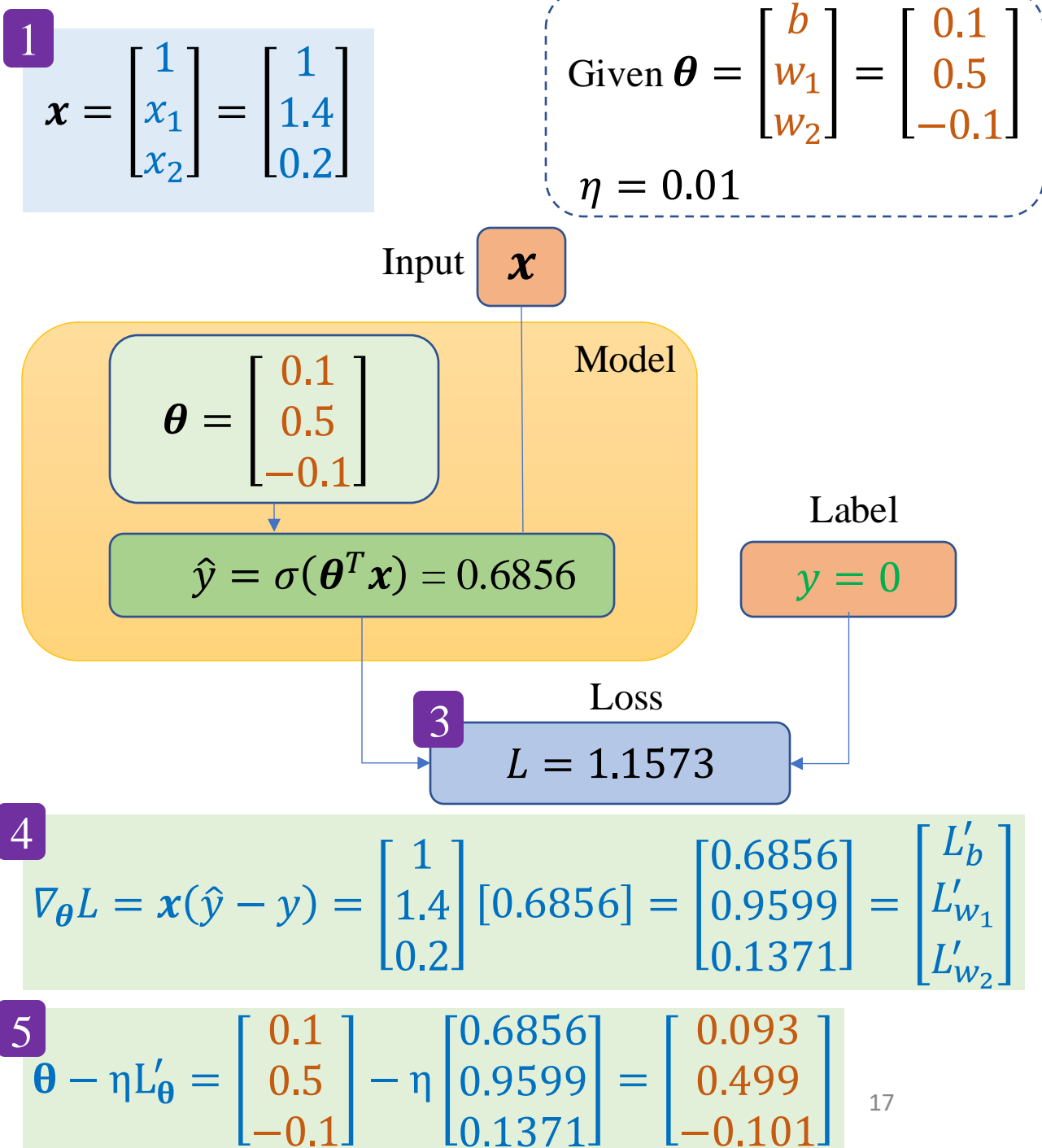
# Given X and y



Dataset

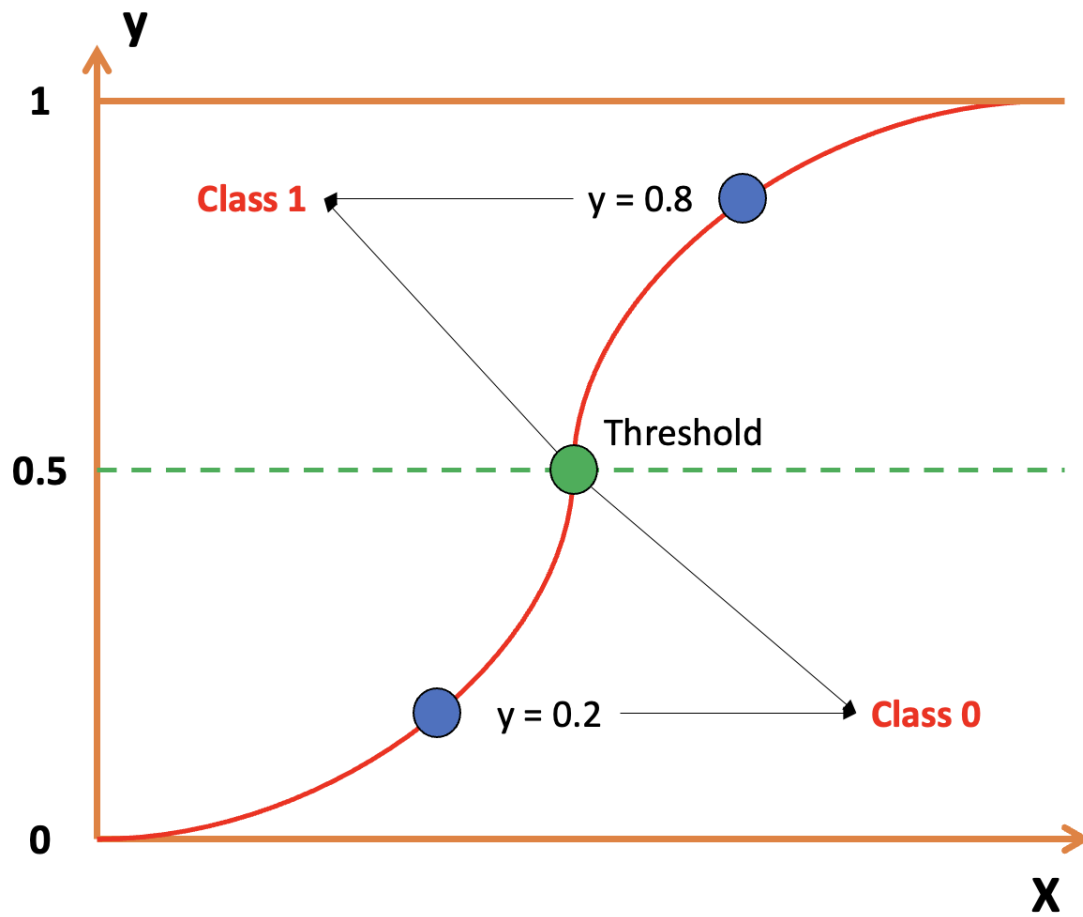
Petal_Length	Petal_Width	Label
1.4	0.2	0
1.5	0.2	0
3	1.1	1
4.1	1.3	1

- 1) Pick a sample  $(\mathbf{x}, y)$  from training data
- ↓
- 2) Compute output  $\hat{y}$
- ↓  $z = \boldsymbol{\theta}^T \mathbf{x} = \mathbf{x}^T \boldsymbol{\theta} \quad \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$
- 3) Compute loss
- ↓  $L(\hat{y}, y) = (-y \log \hat{y} - (1-y) \log(1-\hat{y}))$
- 4) Compute derivative
- ↓  $\nabla_{\boldsymbol{\theta}} L = \mathbf{x}(\hat{y} - y)$
- 5) Update parameters
- $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L$



# Review

## ❖ Logistic Regression



Logistic Regression (LR)

**Sigmoid function**

$$z = b_0 + \sum_{i=1}^{n\_features} b_i x_i$$
$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

**Binary Cross Entropy:**

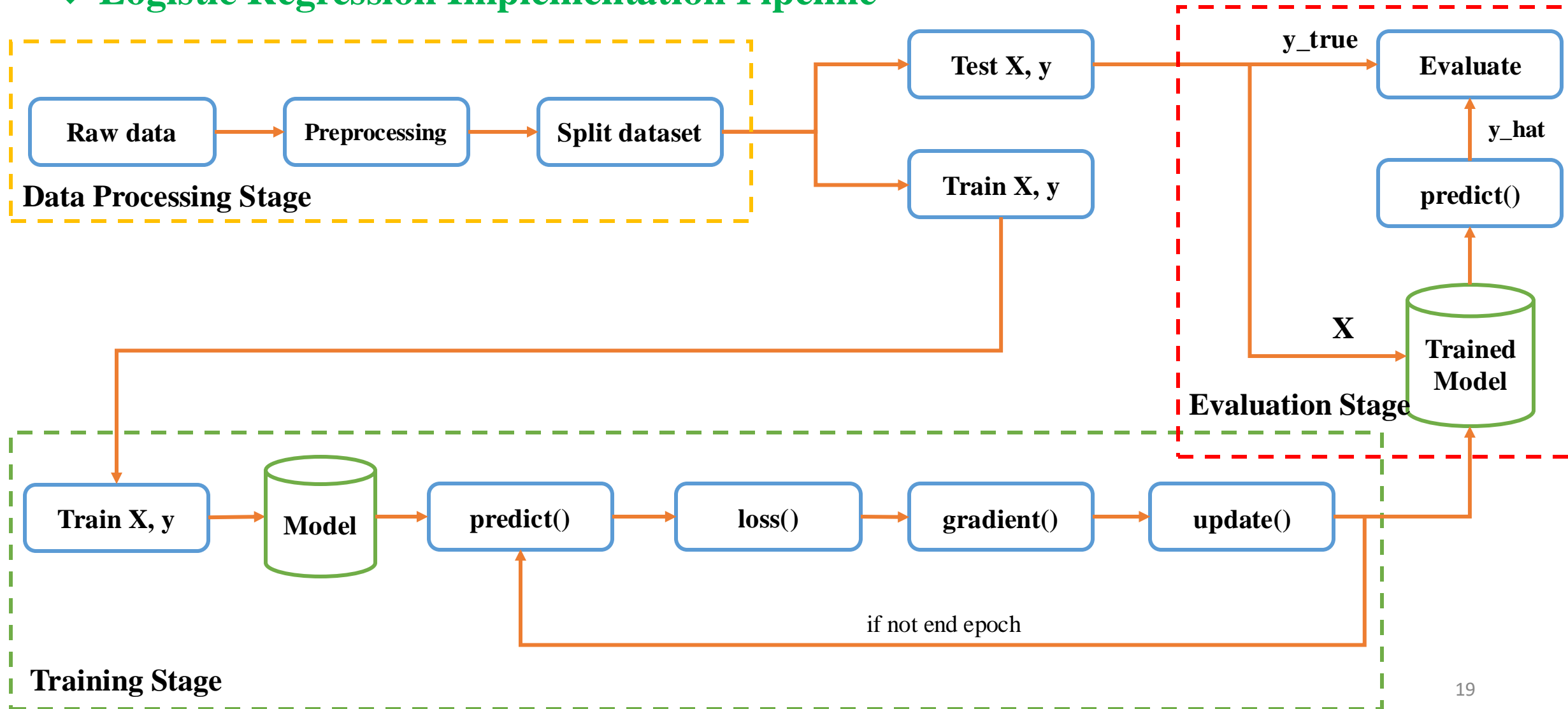
$$L(\hat{y}, y) = -\frac{1}{N} [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

**Gradient:**

$$\nabla_{\theta} L = \frac{1}{N} X^T (\hat{y} - y)$$

# Review

## ❖ Logistic Regression Implementation Pipeline



## ❖ Short Quiz

Câu 1: Trong Logistic Regression, hàm nào được sử dụng để biến đổi đầu ra của một phép tính tuyến tính thành xác suất?

- A. Hàm Sigmoid.
- B. Hàm ReLU.
- C. Hàm Softmax.
- D. Hàm Tanh.

Câu 2: Tại sao Logistic Regression không phù hợp cho các bài toán hồi quy liên tục?

- A. Vì Logistic Regression chỉ dự đoán giá trị nhị phân hoặc xác suất.
- B. Vì Logistic Regression quá phức tạp cho dữ liệu liên tục.
- C. Vì Logistic Regression không thể tối ưu hóa trọng số.
- D. Vì Logistic Regression yêu cầu dữ liệu không có nhiễu.

Câu 3: Trong Logistic Regression, tham số nào được tối ưu hóa để giảm thiểu lỗi dự đoán?

- A. Xác suất đầu ra  $\hat{y}$ .
- B. Trọng số  $w$  và bias  $b$ .
- C. Tham số Sigmoid  $\sigma$ .
- D. Giá trị nhãn  $y$ .

# Titanic Survival Prediction

# Titanic Survival Prediction

## ❖ Introduction

**Description:** Given [Titanic Survival dataset](#), build a Logistic Regression model to determine whether a passenger was survived or not in the Titanic incident.

PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Survived
1	3	0	22	1	0	7.25	0	0	0
2	1	1	38	1	0	71.2833	1	1	1
3	3	1	26	0	0	7.925	0	2	1
4	1	1	35	1	0	53.1	0	1	1
5	3	0	35	0	0	8.05	0	0	0
6	3	0	28	0	0	8.4583	2	0	0
7	1	0	54	0	0	51.8625	0	0	0
8	3	0	2	3	1	21.075	0	3	0
9	3	1	27	0	2	11.1333	0	1	1

# Titanic Survival Prediction

## ❖ Dataset information

PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Survived
1	3	0	22	1	0	7.25	0	0	0
2	1	1	38	1	0	71.2833	1	1	1
3	3	1	26	0	0	7.925	0	2	1
4	1	1	35	1	0	53.1	0	1	1
5	3	0	35	0	0	8.05	0	0	0
6	3	0	28	0	0	8.4583	2	0	0
7	1	0	54	0	0	51.8625	0	0	0
8	3	0	2	3	1	21.075	0	3	0
9	3	1	27	0	2	11.1333	0	1	1

- **Number of samples:** 891
- **Number of features:** 8
- **Number of classes:** 2

Feature	Description
PassengerID	Dataset Index
Pclass	Ticket class
Sex	Sex
Age	Age in years
SibSp	Number of passenger's siblings / spouses aboard the Titanic
Parch	Number of passenger's parents / children aboard the Titanic
Fare	Passenger fare
Embarked	Port of Embarkation
Title	Passenger title
Survived	Survival

# Titanic Survival Prediction

## ❖ Step 1: Import libraries and read dataset

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 from sklearn.model_selection import train_test_split
6 from sklearn.preprocessing import StandardScaler
```

```
1 dataset_path = 'titanic_modified_dataset.csv'
2 df = pd.read_csv(
3     dataset_path,
4     index_col='PassengerId'
5 )
6 df
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Survived
PassengerId									
1	3	0	22.0	1	0	7.2500	0	0	0
2	1	1	38.0	1	0	71.2833	1	1	1
3	3	1	26.0	0	0	7.9250	0	2	1
4	1	1	35.0	1	0	53.1000	0	1	1
5	3	0	35.0	0	0	8.0500	0	0	0
...	...	...	...	...	...	...	...	...	...
887	2	0	27.0	0	0	13.0000	0	5	0
888	1	1	19.0	0	0	30.0000	0	2	1
889	3	1	28.0	1	2	23.4500	0	2	0
890	1	0	26.0	0	0	30.0000	1	0	1
891	3	0	32.0	0	0	7.7500	2	0	0

891 rows x 9 columns



# Titanic Survival Prediction

## ❖ Step 2: Get dataset information

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 891 entries, 1 to 891
```

```
Data columns (total 9 columns):
```

#	Column	Non-Null Count	Dtype
0	Pclass	891 non-null	int64
1	Sex	891 non-null	int64
2	Age	891 non-null	float64
3	SibSp	891 non-null	int64
4	Parch	891 non-null	int64
5	Fare	891 non-null	float64
6	Embarked	891 non-null	int64
7	Title	891 non-null	int64
8	Survived	891 non-null	int64

```
dtypes: float64(2), int64(7)
```

```
memory usage: 69.6 KB
```

# Titanic Survival Prediction

## ❖ Step 2: Get dataset information

```
1 df.describe()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Survived
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	2.308642	0.352413	29.361582	0.523008	0.381594	32.204208	0.359147	0.936027	0.383838
std	0.836071	0.477990	13.019697	1.102743	0.806057	49.693429	0.638707	1.725341	0.486592
min	1.000000	0.000000	0.420000	0.000000	0.000000	0.000000	-1.000000	0.000000	0.000000
25%	2.000000	0.000000	22.000000	0.000000	0.000000	7.910400	0.000000	0.000000	0.000000
50%	3.000000	0.000000	28.000000	0.000000	0.000000	14.454200	0.000000	0.000000	0.000000
75%	3.000000	1.000000	35.000000	1.000000	0.000000	31.000000	1.000000	2.000000	1.000000
max	3.000000	1.000000	80.000000	8.000000	6.000000	512.329200	2.000000	16.000000	1.000000

# Titanic Survival Prediction

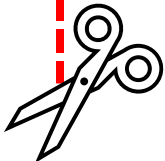
## ❖ Step 3: Split X, y

X

Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
3	0	22	1	0	7.25	0	0
1	1	38	1	0	71.2833	1	1
3	1	26	0	0	7.925	0	2
1	1	35	1	0	53.1	0	1
3	0	35	0	0	8.05	0	0

Y

Survived
0
1
1
1
0



### Split raw dataset to X, y

- ❖ X: first 8 columns (except PassengerID)
- ❖ y: last column

```
dataset = df.to_numpy().astype(np.float64)
```

```
X, y = dataset[:, :-1], dataset[:, -1]
```

# Titanic Survival Prediction

## ❖ Step 3: Split X, y (add intercepts)

$$z = b_0 + \sum_{i=1}^{n_{features}} b_i x_i$$

Currently our X only has n\_features, thus we need to add b (intercept)

```
1 intercept = np.ones((
2 |     X.shape[0], 1)
3 | )
4 X_b = np.concatenate(
5 |     (intercept, X),
6 |     axis=1
7 | )
```

New shape of X after adding intercepts:

```
1 print(X_b.shape)
```

(891, 9)

```
1 print(X.shape)
```

(891, 8)

shape[1] == 8 means 8 features

#	Column
---	-----
0	Pclass
1	Sex
2	Age
3	SibSp
4	Parch
5	Fare
6	Embarked
7	Title

```
1 intercept
```

```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

```
1 X
```

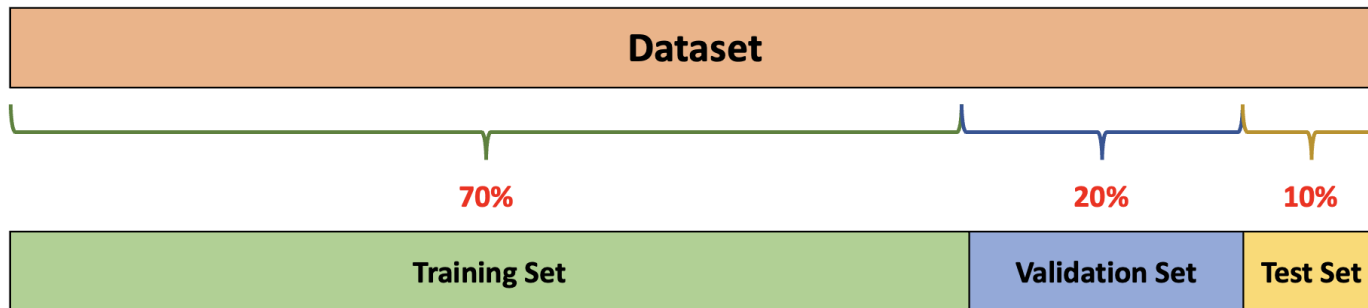
```
array([[ 3.,  0.,
        [ 1.,  1.,
        [ 3.,  1.,
        ...,
        [ 1.,  3.,
        [ 1.,  1.,
        [ 1.,  3.]])
```

```
1 X_b
```

```
array([[ 1.,  3.,
        [ 1.,  1.,
        [ 1.,  3.,
        ...,
        [ 1.,  3.,
        [ 1.,  1.,
        [ 1.,  3.]])
```

# Titanic Survival Prediction

## ❖ Step 4: Split train, val, test set



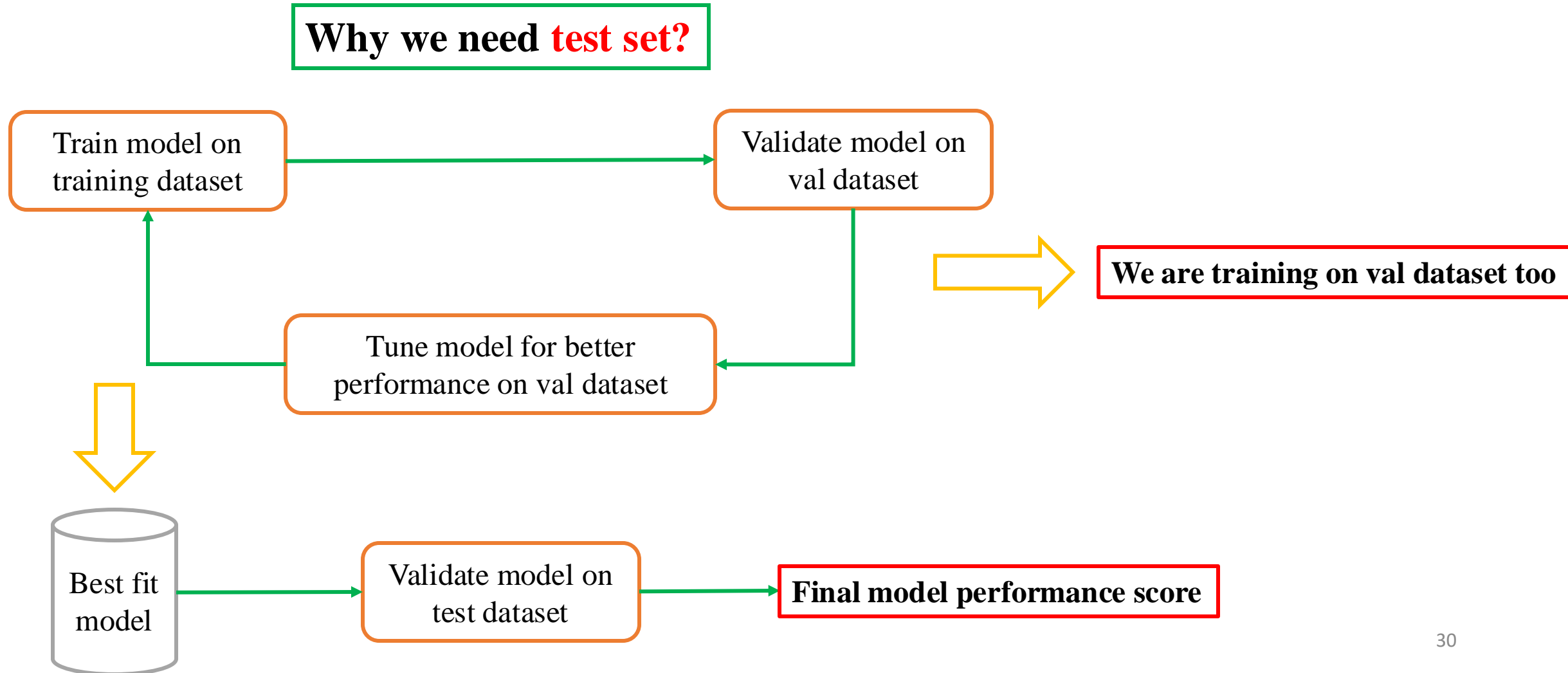
```
1 val_size = 0.2
2 test_size = 0.125
3 random_state = 2
4 is_shuffle = True
5
6 X_train, X_val, y_train, y_val = train_test_split(
7     X_b, y,
8     test_size=val_size,
9     random_state=random_state,
10    shuffle=is_shuffle
11 )
12
13 X_train, X_test, y_train, y_test = train_test_split(
14     X_train, y_train,
15     test_size=test_size,
16     random_state=random_state,
17     shuffle=is_shuffle
18 )
```

```
1 print(f'Number of training samples: {X_train.shape[0]}')
2 print(f'Number of val samples: {X_val.shape[0]}')
3 print(f'Number of test samples: {X_test.shape[0]}')
```

Number of training samples: 623  
Number of val samples: 179  
Number of test samples: 89

# Titanic Survival Prediction

## ❖ Step 4: Split train, val, test set



# Titanic Survival Prediction

## ❖ Step 5: Normalization

Using `sklearn.preprocessing.StandardScaler()` to scale all values in dataset.

$$z = \frac{x_i - \mu}{\sigma}$$

```
1 normalizer = StandardScaler()
2 X_train = normalizer.fit_transform(X_train)
3 X_val = normalizer.transform(X_val)
4 X_test = normalizer.transform(X_test)
```

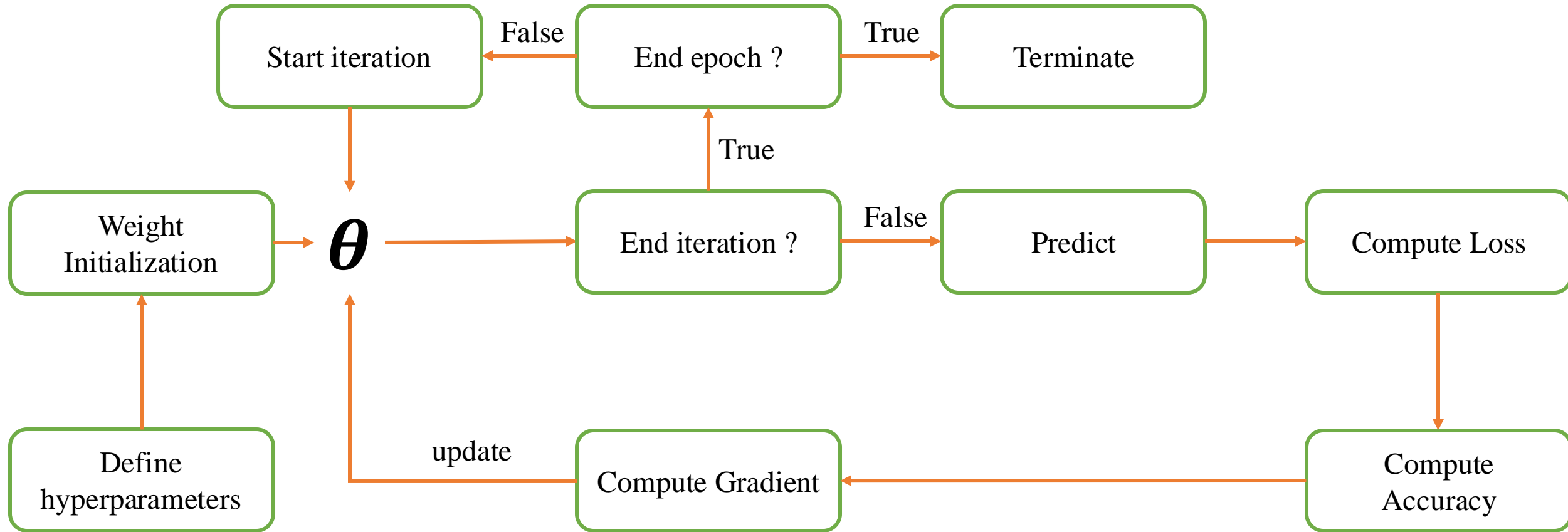
1 X\_train

```
array([[ 0.82313781, -0.75449952, -0.09246773, ..., -0.5944735 ,
        -0.56180558,  0.          ],
       [-0.37502774,  1.3253819 , -0.40715477, ..., -0.5944735 ,
         0.64719226,  0.          ],
       [ 0.82313781, -0.75449952, -0.80051356, ..., -0.5944735 ,
        -0.56180558,  0.          ],
       ...,
       [-1.57319329, -0.75449952,  3.29041791, ...,  0.91104272,
        -0.56180558,  0.          ],
       [-1.57319329, -0.75449952, -0.09246773, ..., -0.5944735 ,
        -0.56180558,  0.          ],
       [-1.57319329, -0.75449952, -0.09246773, ..., -0.5944735 ,
        -0.56180558,  0.          ]])
```

**Note:** We only use the train set to fit the scaler.

# Titanic Survival Prediction

## ❖ Step 6: Training

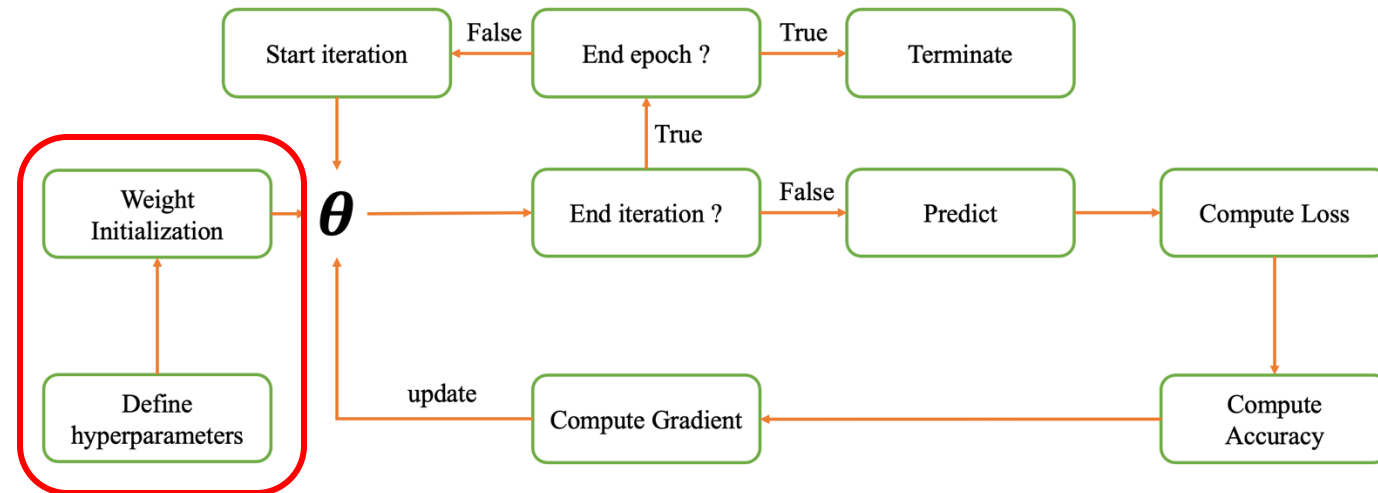




# Titanic Survival Prediction

## ❖ Step 6: Training

```
1 lr = 0.01
2 epochs = 100
3 batch_size = 16
4
5 np.random.seed(random_state)
6 theta = np.random.uniform(
7     size=X_train.shape[1]
8 )
```



Define essential hyperparameters and initialize weights.

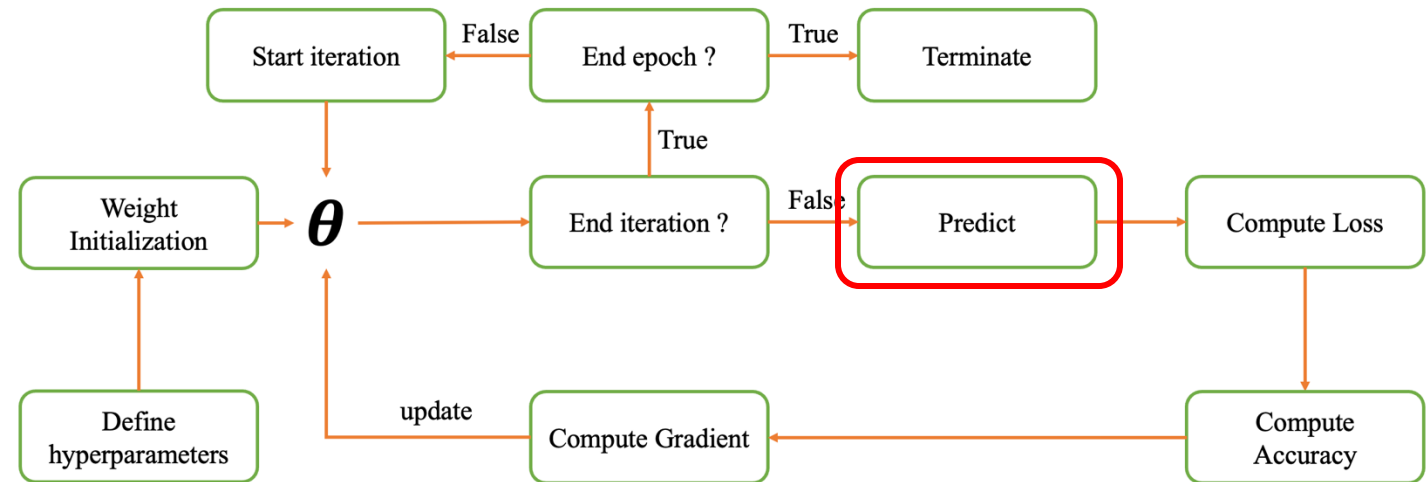
# Titanic Survival Prediction

## ❖ Step 6: Training

### Hypothesis function

- $z = b_0 + \sum_{i=1}^{n\_features} b_i x_i$
- $y_{hat} = h(z) = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$

```
1 def sigmoid(z):
2     return 1 / (1 + np.exp(-z))
3
4
5 def predict(X, theta):
6     dot_product = np.dot(X, theta)
7     y_hat = sigmoid(dot_product)
8
9     return y_hat
```



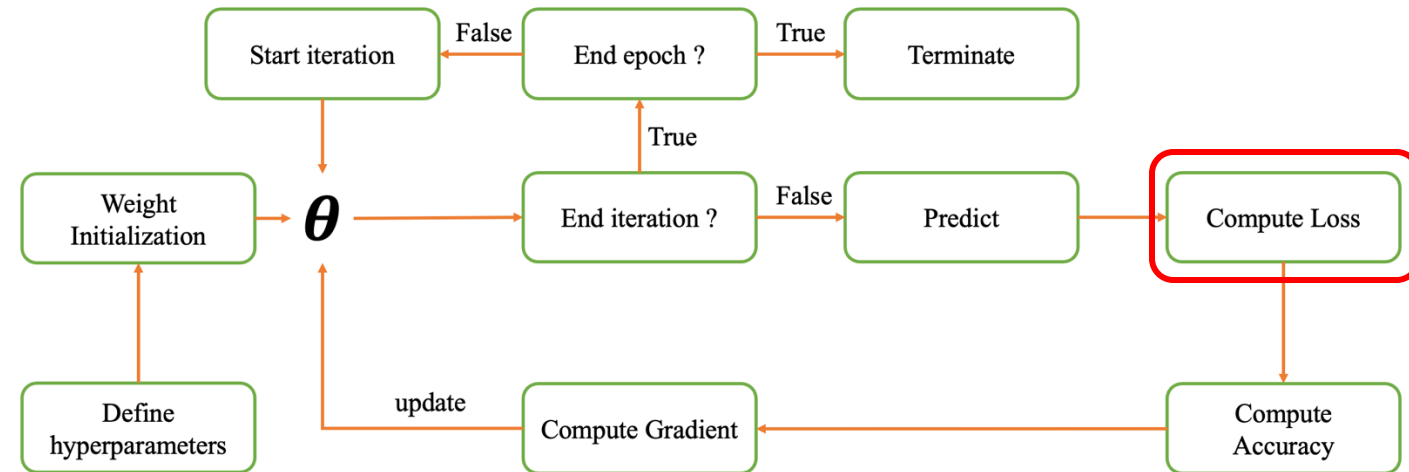
# Titanic Survival Prediction

## ❖ Step 6: Training

### Logistic Regression Loss (Cross-entropy)

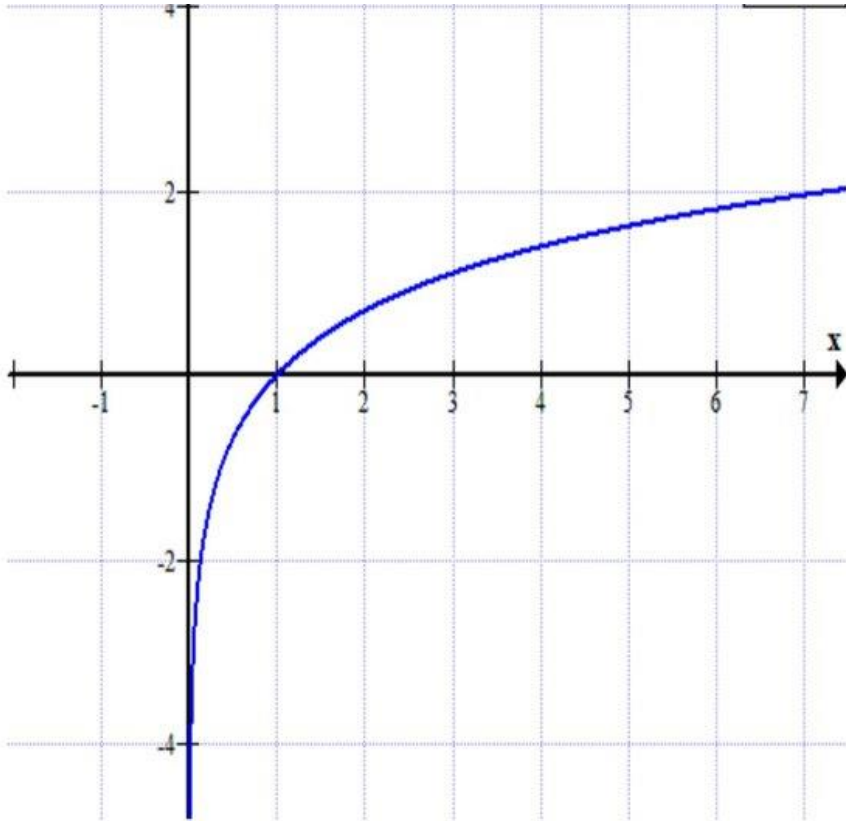
$$L(\hat{y}, y) = -\frac{1}{N} \sum_{i=1}^N y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

```
1 def loss(y_hat, y):
2     y_hat = np.clip(
3         y_hat, 1e-7, 1 - 1e-7
4     )
5
6     return (
7         -y * \
8         np.log(y_hat) - (1 - y) * \
9         np.log(1 - y_hat)
10    ).mean()
```



# Titanic Survival Prediction

## ❖ Step 6: Training



→ Can cause nan if  $\log(0)$ .

```
1 def loss(y_hat, y):
2     y_hat = np.clip(
3         y_hat, 1e-7, 1 - 1e-7
4     )
5
6     return (
7         -y * \
8         np.log(y_hat) - (1 - y) * \
9         np.log(1 - y_hat)
10    ).mean()
```

Avoid prediction values too small by limiting the value range.

# Titanic Survival Prediction

## ❖ Step 6: Training

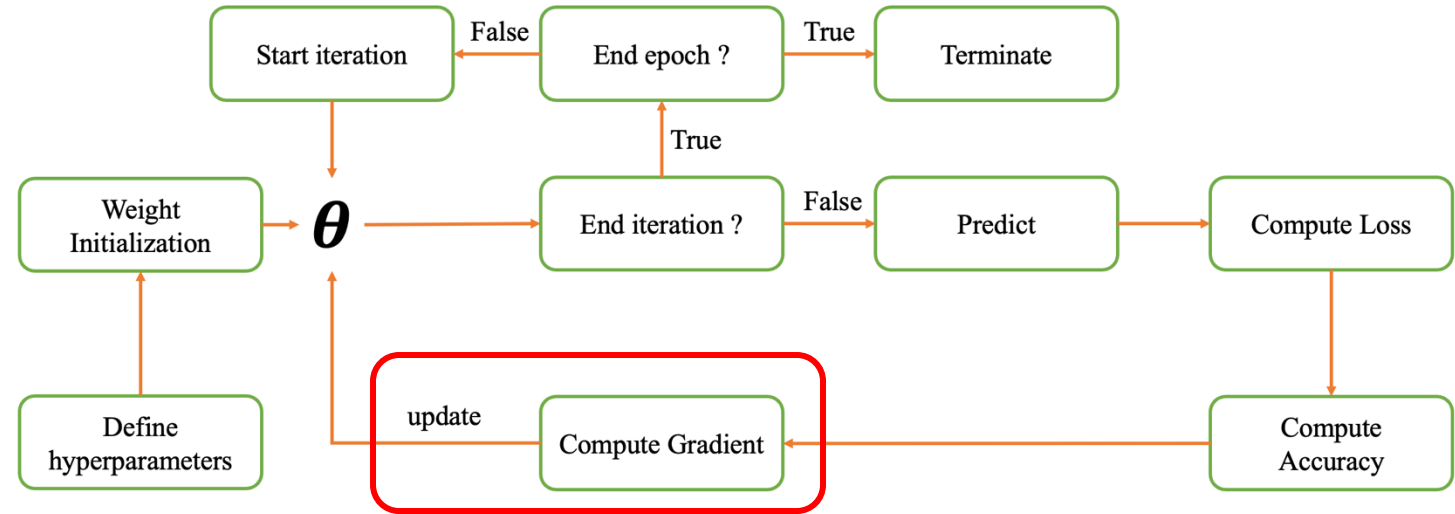
- Gradient computation formula:

$$\nabla_{\theta} L = \frac{1}{N} X^T (\hat{y} - y)$$

- Weights update formula:

$$\theta = \theta - \eta \nabla_{\theta} L$$

```
1 def compute_gradient(X, y, y_hat):
2     return np.dot(
3         X.T, (y_hat - y)
4     ) / y.size
5
6 def update_theta(theta, gradient, lr):
7     return theta - lr * gradient
```



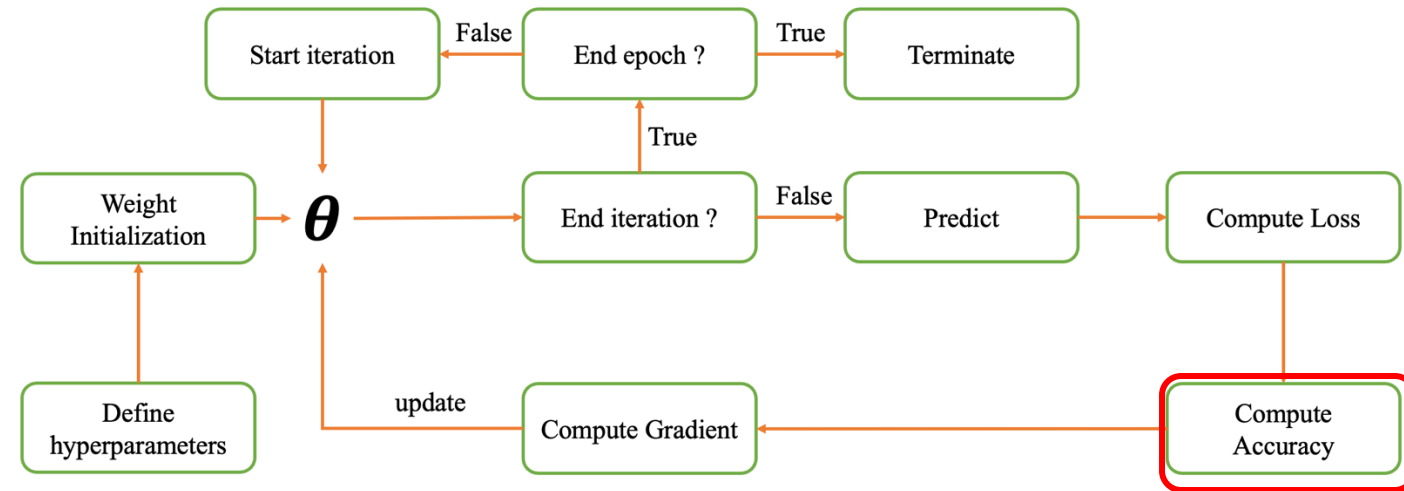
# Titanic Survival Prediction

## ❖ Step 6: Training

### Accuracy Formula

$$accuracy = \frac{true\_predictions}{n\_samples}$$

```
1 def compute_accuracy(X, y, theta):  
2     y_hat = predict(X, theta).round()  
3     acc = (y_hat == y).mean()  
4  
5     return acc
```



# Titanic Survival Prediction

## ❖ Step 6: Training

**Line 6:** Loop over number of epochs.

**Line 7, 8, 9, 10:** Declare empty lists to store batch accuracies, losses of train and val sets.

```
6 for epoch in range(epochs):
7     train_batch_losses = []
8     train_batch_accs = []
9     val_batch_losses = []
10    val_batch_accs = []
11
12    for i in range(0, X_train.shape[0], batch_size):
13        X_i = X_train[i:i+batch_size]
14        y_i = y_train[i:i+batch_size]
15
16        y_hat = predict(X_i, theta)
17        train_loss = compute_loss(y_hat, y_i)
18        gradient = compute_gradient(X_i, y_i, y_hat)
19        theta = update_theta(theta, gradient, lr)
20        train_batch_losses.append(train_loss)
21
22        train_acc = compute_accuracy(X_train, y_train, theta)
23        train_batch_accs.append(train_acc)
24
25        y_val_hat = predict(X_val, theta)
26        val_loss = compute_loss(y_val_hat, y_val)
27        val_batch_losses.append(val_loss)
28
29        val_acc = compute_accuracy(X_val, y_val, theta)
30        val_batch_accs.append(val_acc)
31
32    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

# Titanic Survival Prediction

## ❖ Step 6: Training

**Line 12:** Loop over number of batches (based on batch size).

**Line 13, 14:** Get X and y data of current batch.

```
6 for epoch in range(epochs):
7     train_batch_losses = []
8     train_batch_accs = []
9     val_batch_losses = []
10    val_batch_accs = []
11
12    for i in range(0, X_train.shape[0], batch_size):
13        X_i = X_train[i:i+batch_size]
14        y_i = y_train[i:i+batch_size]
15
16        y_hat = predict(X_i, theta)
17        train_loss = compute_loss(y_hat, y_i)
18        gradient = compute_gradient(X_i, y_i, y_hat)
19        theta = update_theta(theta, gradient, lr)
20        train_batch_losses.append(train_loss)
21
22        train_acc = compute_accuracy(X_train, y_train, theta)
23        train_batch_accs.append(train_acc)
24
25        y_val_hat = predict(X_val, theta)
26        val_loss = compute_loss(y_val_hat, y_val)
27        val_batch_losses.append(val_loss)
28
29        val_acc = compute_accuracy(X_val, y_val, theta)
30        val_batch_accs.append(val_acc)
31
32    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```



# Titanic Survival Prediction

## ❖ Step 6: Training

**Line 16:** Do prediction on  $X_i$  and  $\theta$ .

**Line 17:** Compute the loss of  $\hat{y}$  and  $y_i$ .

**Line 18:** Compute the gradient.

**Line 19:** Update  $\theta$  using the computed gradient.

```
6 for epoch in range(epochs):
7     train_batch_losses = []
8     train_batch_accs = []
9     val_batch_losses = []
10    val_batch_accs = []
11
12    for i in range(0, X_train.shape[0], batch_size):
13        X_i = X_train[i:i+batch_size]
14        y_i = y_train[i:i+batch_size]
15
16        y_hat = predict(X_i, theta)
17        train_loss = compute_loss(y_hat, y_i)
18        gradient = compute_gradient(X_i, y_i, y_hat)
19        theta = update_theta(theta, gradient, lr)
20        train_batch_losses.append(train_loss)
21
22        train_acc = compute_accuracy(X_train, y_train, theta)
23        train_batch_accs.append(train_acc)
24
25        y_val_hat = predict(X_val, theta)
26        val_loss = compute_loss(y_val_hat, y_val)
27        val_batch_losses.append(val_loss)
28
29        val_acc = compute_accuracy(X_val, y_val, theta)
30        val_batch_accs.append(val_acc)
31
32    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

# Titanic Survival Prediction

## ❖ Step 6: Training

**From line 20 to line 30:** Compute and store accuracies and losses on train and val sets.

```
6 for epoch in range(epochs):
7     train_batch_losses = []
8     train_batch_accs = []
9     val_batch_losses = []
10    val_batch_accs = []
11
12    for i in range(0, X_train.shape[0], batch_size):
13        X_i = X_train[i:i+batch_size]
14        y_i = y_train[i:i+batch_size]
15
16        y_hat = predict(X_i, theta)
17        train_loss = compute_loss(y_hat, y_i)
18        gradient = compute_gradient(X_i, y_i, y_hat)
19        theta = update_theta(theta, gradient, lr)
20        train_batch_losses.append(train_loss)
21
22        train_acc = compute_accuracy(X_train, y_train, theta)
23        train_batch_accs.append(train_acc)
24
25        y_val_hat = predict(X_val, theta)
26        val_loss = compute_loss(y_val_hat, y_val)
27        val_batch_losses.append(val_loss)
28
29        val_acc = compute_accuracy(X_val, y_val, theta)
30        val_batch_accs.append(val_acc)
31
32    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

# Titanic Survival Prediction

## ❖ Step 6: Training

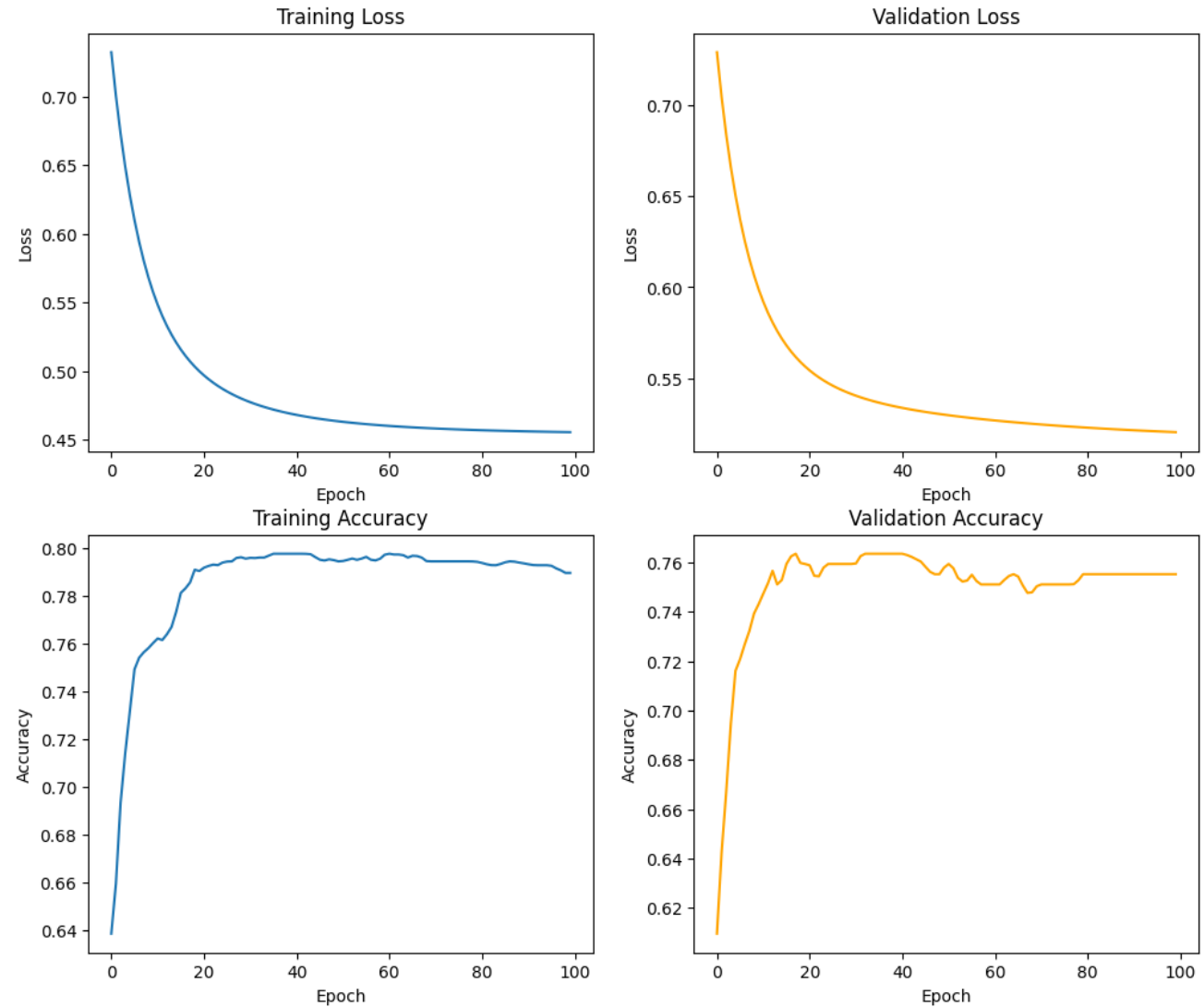
**From line 32 to line 35:** Compute and store batch accuracies and losses of train and val sets.

```
6 for epoch in range(epochs):
7     train_batch_losses = []
8     train_batch_accs = []
9     val_batch_losses = []
10    val_batch_accs = []
11
12    for i in range(0, X_train.shape[0], batch_size):
13        X_i = X_train[i:i+batch_size]
14        y_i = y_train[i:i+batch_size]
15
16        y_hat = predict(X_i, theta)
17        train_loss = compute_loss(y_hat, y_i)
18        gradient = compute_gradient(X_i, y_i, y_hat)
19        theta = update_theta(theta, gradient, lr)
20        train_batch_losses.append(train_loss)
21
22        train_acc = compute_accuracy(X_train, y_train, theta)
23        train_batch_accs.append(train_acc)
24
25        y_val_hat = predict(X_val, theta)
26        val_loss = compute_loss(y_val_hat, y_val)
27        val_batch_losses.append(val_loss)
28
29        val_acc = compute_accuracy(X_val, y_val, theta)
30        val_batch_accs.append(val_acc)
31
32    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

# Titanic Survival Prediction

## ❖ Step 6: Training (Visualization)

```
1 fig, ax = plt.subplots(2, 2, figsize=(12, 10))
2 ax[0, 0].plot(train_losses)
3 ax[0, 0].set_xlabel='Epoch', ylabel='Loss'
4 ax[0, 0].set_title('Training Loss')
5
6 ax[0, 1].plot(val_losses, 'orange')
7 ax[0, 1].set_xlabel='Epoch', ylabel='Loss'
8 ax[0, 1].set_title('Validation Loss')
9
10 ax[1, 0].plot(train_accs)
11 ax[1, 0].set_xlabel='Epoch', ylabel='Accuracy'
12 ax[1, 0].set_title('Training Accuracy')
13
14 ax[1, 1].plot(val_accs, 'orange')
15 ax[1, 1].set_xlabel='Epoch', ylabel='Accuracy'
16 ax[1, 1].set_title('Validation Accuracy')
17
18 plt.show()
```



# Titanic Survival Prediction

## ❖ Step 7: Evaluation

$$accuracy = \frac{true\_predictions}{n\_samples}$$

```
1 def compute_accuracy(X, y, theta):  
2     y_hat = predict(X, theta).round()  
3     acc = (y_hat == y).mean()  
4  
5     return acc
```

```
1 # Val set  
2 val_set_acc = compute_accuracy(X_val, y_val, theta)  
3 print('Evaluation on validation set:')  
4 print(f'Accuracy: {val_set_acc}')
```

Evaluation on validation set:  
Accuracy: 0.770949720670391

```
1 # Test set  
2 test_set_acc = compute_accuracy(X_test, y_test, theta)  
3 print('Evaluation on test set:')  
4 print(f'Accuracy: {test_set_acc}')
```

Evaluation on test set:  
Accuracy: 0.7752808988764045

# QUIZ

# Sentiment Analysis

# Sentiment Analysis

## ❖ Introduction

**Description:** Given [Twitter Sentiment Analysis](#), build a Logistic Regression model to determine whether a tweet (text) has a positive sentiment or not.



Positive



Negative



# Sentiment Analysis

## ❖ Step 1: Import libraries

```
1 import pandas as pd
2 import numpy as np
3 import re
4 import nltk
5 import matplotlib.pyplot as plt
6
7 from sklearn.model_selection import train_test_split
8 from sklearn.preprocessing import StandardScaler
9 from nltk.tokenize import TweetTokenizer
10 from collections import defaultdict
```



# Sentiment Analysis

## ❖ Step 2: Read dataset

```
1 dataset_path = 'sentiment_analysis.csv'
2 df = pd.read_csv(
3     dataset_path,
4     index_col='id'
5 )
6 df
```

	label	tweet
id		
1	0	#fingerprint #Pregnancy Test <a href="https://goo.gl/h1...">https://goo.gl/h1...</a>
2	0	Finally a transparant silicon case ^^ Thanks t...
3	0	We love this! Would you go? #talk #makememorie...
4	0	I'm wired I know I'm George I was made that wa...
5	1	What amazing service! Apple won't even talk to...
...	...	...
7916	0	Live out loud #lol #liveoutloud #selfie #smile...
7917	0	We would like to wish you an amazing day! Make...
7918	0	Helping my lovely 90 year old neighbor with he...
7919	0	Finally got my #smart #pocket #wifi stay conne...
7920	0	Apple Barcelona!!! #Apple #Store #BCN #Barcelo...

7920 rows x 2 columns

# Sentiment Analysis

## ❖ Step 3: Get dataset information

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7920 entries, 1 to 7920
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  -
 0   label   7920 non-null   int64  
 1   tweet   7920 non-null   object  
dtypes: int64(1), object(1)
memory usage: 185.6+ KB
```

```
1 df.describe()
```

	label
count	7920.000000
mean	0.255808
std	0.436342
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

# Sentiment Analysis

## ❖ Step 4: Data preprocessing

label	tweet
0	#fingerprint #Pregnancy Test https://goo.gl/h1...
0	Finally a transparant silicon case ^^ Thanks t...
0	We love this! Would you go? #talk #makememorie...
0	I'm wired I know I'm George I was made that wa...
1	What amazing service! Apple won't even talk to...
...	...
0	Live out loud #lol #liveoutloud #selfie #smile...
0	We would like to wish you an amazing day! Make...
0	Helping my lovely 90 year old neighbor with he...
0	Finally got my #smart #pocket #wifi stay conne...
0	Apple Barcelona!!! #Apple #Store #BCN #Barcelo...

**Raw Text:** We love this! Would you go? #talk #makememories #unplug #relax #iphone #smartphone #wifi #connect...  
http://fb.me/6N3LsUpCu

**Data  
Preprocessing**

**Vector Representation:** `array([1.0, 4768.0, 1425.0])`

# Sentiment Analysis

## ❖ Step 4: Data preprocessing

**Raw Text:** We love this! Would you go? #talk #makememories  
#unplug #relax #iphone #smartphone #wifi #connect...  
<http://fb.me/6N3LsUpCu>

**Data  
Preprocessing**

**Vector Representation:** `array([1.0, 4768.0, 1425.0])`

**Vector Representation Meaning**

**0**

Bias

**1**

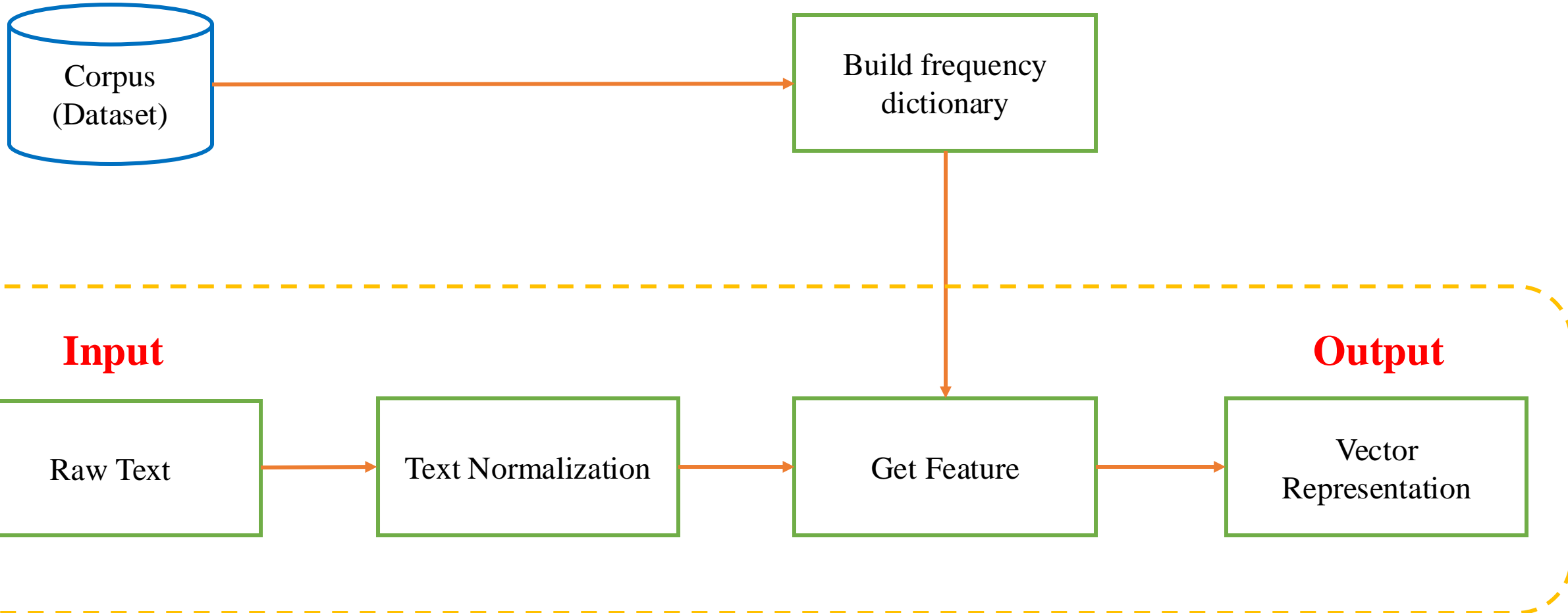
Positive  
Word Count

**2**

Negative  
Word Count

# Sentiment Analysis

## ❖ Step 4: Data preprocessing



# Sentiment Analysis

## ❖ Step 4: Data preprocessing

**Input**

**Output**

Raw Text

Text Normalization

Normalized Text

Retweet Acronym  
Removal

Hyperlinks  
Removal

Hashtags Removal

Punctuation  
Removal

Tokenization

# Sentiment Analysis

## ❖ Step 4: Data preprocessing

**Retweet Acronym Removal:** Remove "RT" at the start of the string

```
1 text = "RT Hello World"
2 text = re.sub(r'^RT[\s]+', '', text)
3 print(text)
```

Hello World

**Hyperlinks Removal:** Remove all hyperlinks in the string.

```
1 text = "http://fb.me/6N3LsUpCu"
2 text = re.sub(
3     r'https?:\/\/\.[\r\n]*',
4     '',
5     text
6 )
7 print(text)
```

**Hashtag Removal:** Remove all hashtag symbols "#".

```
1 text = "#xinchao, #hello, #konnichiwa"
2 text = re.sub(r'^\w\s', '', text)
3 print(text)
```

xinchao hello konnichiwa

**Tokenization:** Tokenize the string into list of tokens (list of words).

```
1 text = "hello world"
2 tokenizer = TweetTokenizer(
3     preserve_case=False,
4     strip_handles=True,
5     reduce_len=True
6 )
7 text_tokens = tokenizer.tokenize(text)
8 print(text_tokens)
```

['hello', 'world']



# Sentiment Analysis

## ❖ Step 4: Data preprocessing

```
1 def text_normalize(text):
2     # Retweet old acronym "RT" removal
3     text = re.sub(r'^RT[\s]+', '', text)
4
5     # Hyperlinks removal
6     text = re.sub(r'https?:\/\/\.[^\s]*', '', text)
7
8     # Punctuation removal
9     text = re.sub(r'[\^\w\s]', '', text)
10
11    # Tokenization
12    tokenizer = TweetTokenizer(
13        preserve_case=False,
14        strip_handles=True,
15        reduce_len=True
16    )
17    text_tokens = tokenizer.tokenize(text)
18
19    return text_tokens
```

```
1 text = """We love this! Would you go?
2 #talk #makememories #unplug
3 #relax #iphone #smartphone #wifi #connect...
4 http://fb.me/6N3LsUpCu
5 """
6 text = text_normalize(text)
7 text
```

```
['we',
 'love',
 'this',
 'would',
 'you',
 'go',
 'talk',
 'makememories',
 'unplug',
 'relax',
 'iphone',
 'smartphone',
 'wifi',
 'connect']
```

# Sentiment Analysis

## ❖ Step 4: Data preprocessing

**Raw Text:** We love this! Would you go? #talk #makememories  
#unplug #relax #iphone #smartphone #wifi #connect...  
<http://fb.me/6N3LsUpCu>

**Data  
Preprocessing**

**Vector Representation:** `array([1.0, 4768.0, 1425.0])`

### Vector Representation Meaning

0	1	2
Bias	Positive Word Count	Negative Word Count



Build a dictionary containing number of occurrences of each word in positive and negative sentences.

# Sentiment Analysis

## ❖ Step 4: Data preprocessing

**Build a dictionary containing number of occurrences of each word in both pos and neg sentences:**

```
{  
    ('hello', 'pos'): 10,  
    ('hello', 'neg'): 0,  
    ('world', 'pos'): 30,  
    ('world', 'neg'): 5,  
    ('my', 'pos'): 25,  
    ('check', 'pos'): 50  
}
```

**Input:** [hello, world, my, name, is, Thang]

```
graph TD; Input["Input: [hello, world, my, name, is, Thang]"] --> GetFeature["Get Feature"]; GetFeature --> Output["Output: [1, 65, 5]"]
```

Get Feature

**Output:** [1, 65, 5]

# Sentiment Analysis

## ❖ Step 4: Data preprocessing

```
1 def get_freqs(df):
2     freqs = defaultdict(lambda: 0)
3     for idx, row in df.iterrows():
4         tweet = row['tweet']
5         label = row['label']
6
7         tokens = text_normalize(tweet)
8         for token in tokens:
9             pair = (token, label)
10            freqs[pair] += 1
11
12    return freqs
13
14 freqs = get_freqs(df)
```

```
{('fingerprint', 0): 4,
 ('pregnancy', 0): 1,
 ('test', 0): 8,
 ('finally', 0): 168,
 ('a', 0): 727,
 ('transparent', 0): 1,
 ('silicon', 0): 1,
 ('case', 0): 228,
 ('thanks', 0): 94,
 ('to', 0): 876,
 ('my', 0): 1227,
 ('uncle', 0): 4,
 ('yay', 0): 63,
 ('sony', 0): 701,
 ('xperia', 0): 54,
 ('s', 0): 38,
 ('sonyexperias', 0): 1,
 ('we', 0): 159,
 ('love', 0): 385,
```

# Sentiment Analysis

## ❖ Step 4: Data preprocessing

```
1 def get_freqs(df):
2     freqs = defaultdict(lambda: 0)
3     for idx, row in df.iterrows():
4         tweet = row['tweet']
5         label = row['label']
6
7         tokens = text_normalize(tweet)
8         for token in tokens:
9             pair = (token, label)
10            freqs[pair] += 1
11
12    return freqs
13
14 freqs = get_freqs(df)
```

```
class collections.defaultdict(default_factory=None, /[, ...])
```

Return a new dictionary-like object. `defaultdict` is a subclass of the built-in `dict` class. It overrides one method and adds one writable instance variable. The remaining functionality is the same as for the `dict` class and is not documented here.

The first argument provides the initial value for the `default_factory` attribute; it defaults to `None`. All remaining arguments are treated the same as if they were passed to the `dict` constructor, including keyword arguments.

**defaultdict benefit:** Assign a default value for non-existing keys. For e.g., `lambda: 0` will assign value 0 to keys excluded from the dictionary.

# Sentiment Analysis

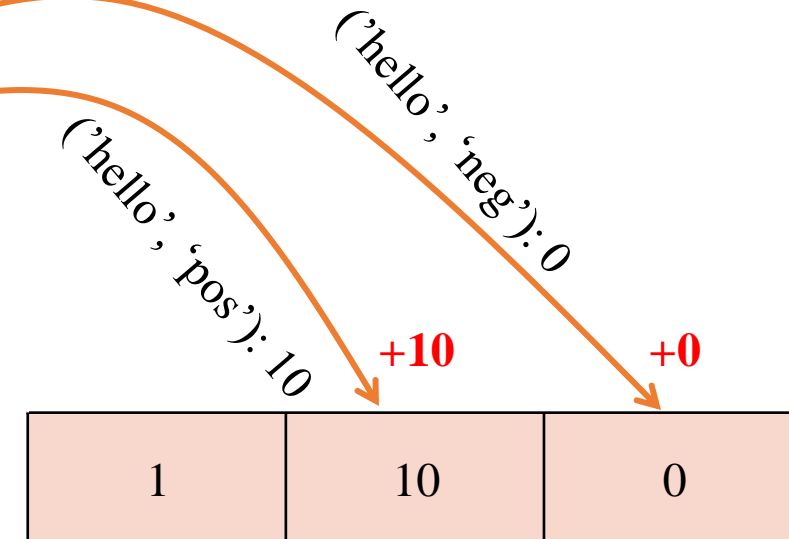
## ❖ Step 4: Data preprocessing

With frequency dictionary, we can create new vector representation for raw texts as following:

```
1 def get_feature(text, freqs):
2     tokens = text_normalize(text)
3
4     X = np.zeros(3)
5     X[0] = 1
6
7     for token in tokens:
8         X[1] += freqs[(token, 0)]
9         X[2] += freqs[(token, 1)]
10
11     return X
```

**Input:** [hello, world, my, name, is, Thang]

```
{
    ('hello', 'pos'): 10,
    ('hello', 'neg'): 0,
    ('world', 'pos'): 30,
    ('world', 'neg'): 5,
    ('my', 'pos'): 25,
    ('check', 'pos'): 50
}
```



# Sentiment Analysis

## ❖ Step 4: Data preprocessing

**Raw Text:** We love this! Would you go? #talk #makememories  
#unplug #relax #iphone #smartphone #wifi #connect...  
<http://fb.me/6N3LsUpCu>

**Data  
Preprocessing**

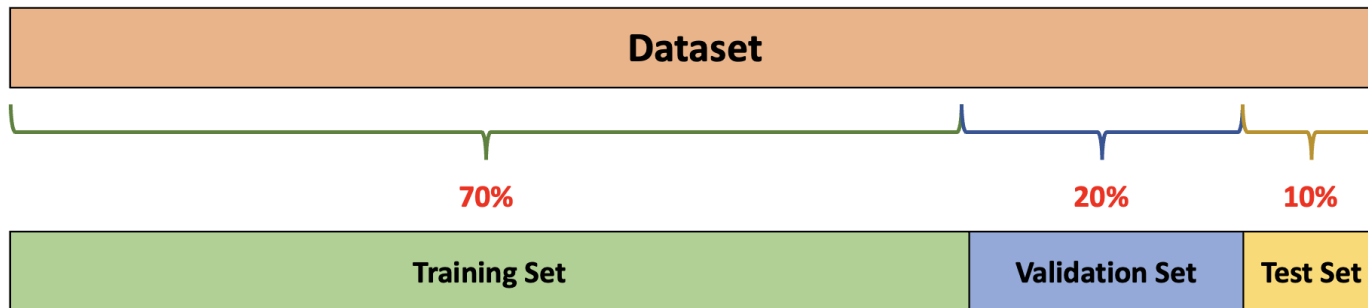
**Vector Representation:** `array([1.0, 4768.0, 1425.0])`

```
1 X = []
2 y = []
3
4 for idx, row in df.iterrows():
5     tweet = row['tweet']
6     label = row['label']
7
8     X_i = get_feature(tweet, freqs)
9     X.append(X_i)
10    y.append(label)
11
12 X = np.array(X)
13 y = np.array(y).reshape(-1, 1)
14
15 print(f'X shape: {X.shape}')
16 print(f'y shape: {y.shape}')
```

```
X shape: (7920, 3)
y shape: (7920, 1)
```

# Sentiment Analysis

## ❖ Step 5: Split train, val, test set



```
1 val_size = 0.2
2 test_size = 0.125
3 random_state = 2
4 is_shuffle = True
5
6 X_train, X_val, y_train, y_val = train_test_split(
7     X, y,
8     test_size=val_size,
9     random_state=random_state,
10    shuffle=is_shuffle
11 )
12
13 X_train, X_test, y_train, y_test = train_test_split(
14     X_train, y_train,
15     test_size=test_size,
16     random_state=random_state,
17     shuffle=is_shuffle
18 )
```

```
1 print(f'Number of training samples: {X_train.shape[0]}')
2 print(f'Number of val samples: {X_val.shape[0]}')
3 print(f'Number of test samples: {X_test.shape[0]}')
```

Number of training samples: 5544  
Number of val samples: 1584  
Number of test samples: 792



# Sentiment Analysis

## ❖ Step 6: Normalization

Using `sklearn.preprocessing.StandardScaler()` to scale all values in dataset.

$$z = \frac{x_i - \mu}{\sigma}$$

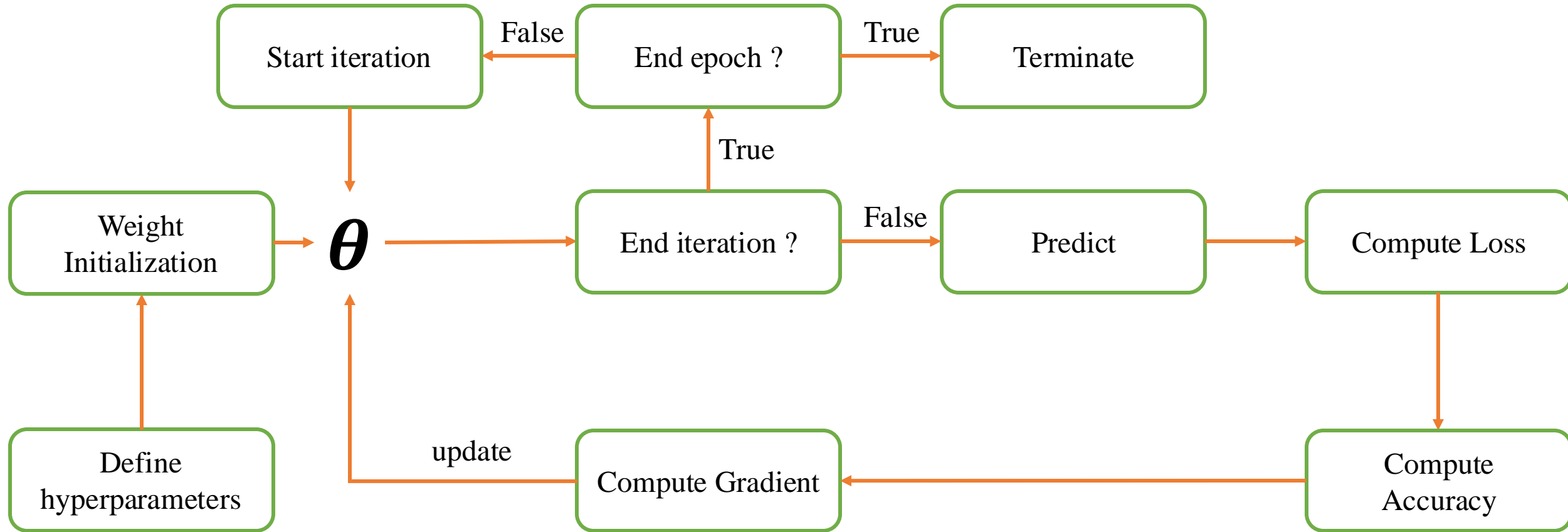
```
1 normalizer = StandardScaler()  
2 X_train[:, 1:] = normalizer.fit_transform(X_train[:, 1:])  
3 X_val[:, 1:] = normalizer.transform(X_val[:, 1:])  
4 X_test[:, 1:] = normalizer.transform(X_test[:, 1:])
```

```
1 X_train  
  
array([[ 1.          , -0.04496429, -0.80123214],  
       [ 1.          ,  0.83177357,  0.04495408],  
       [ 1.          , -0.78519562, -0.93920268],  
       ...,  
       [ 1.          , -0.37919422,  0.26878398],  
       [ 1.          ,  0.33852057, -0.10989055],  
       [ 1.          ,  1.73384255,  1.86239334]])
```

**Note:** We only use the train set to fit the scaler.

# Sentiment Analysis

## ❖ Step 7: Training



# Sentiment Analysis

## ❖ Step 7: Training

```
1 lr = 0.01
2 epochs = 200
3 batch_size = 128
4
5 np.random.seed(random_state)
6 theta = np.random.uniform(
7     size=X_train.shape[1]
8 )
```

Define essential functions and hyperparameters

```
1 def sigmoid(z):
2     return 1 / (1 + np.exp(-z))
3
4 def predict(X, theta):
5     dot_product = np.dot(X, theta)
6     y_hat = sigmoid(dot_product)
7
8     return y_hat
9
10 def compute_loss(y_hat, y):
11     y_hat = np.clip(
12         y_hat, 1e-7, 1 - 1e-7
13     )
14
15     return (-y * np.log(y_hat) - (1 - y) * np.log(1 - y_hat)).mean()
16
17 def compute_gradient(X, y, y_hat):
18     return np.dot(
19         X.T, (y_hat - y)
20     ) / y.size
21
22 def update_theta(theta, gradient, lr):
23     return theta - lr * gradient
24
25 def compute_accuracy(X, y, theta):
26     y_hat = predict(X, theta).round()
27     acc = (y_hat == y).mean()
28
29     return acc
```

# Sentiment Analysis

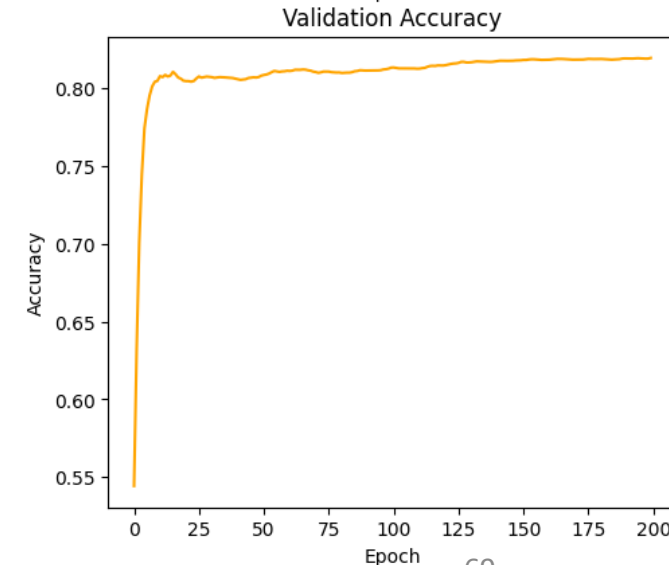
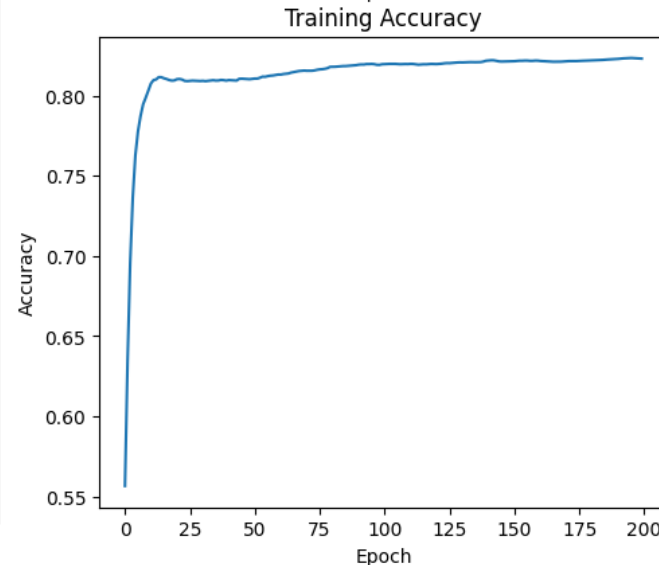
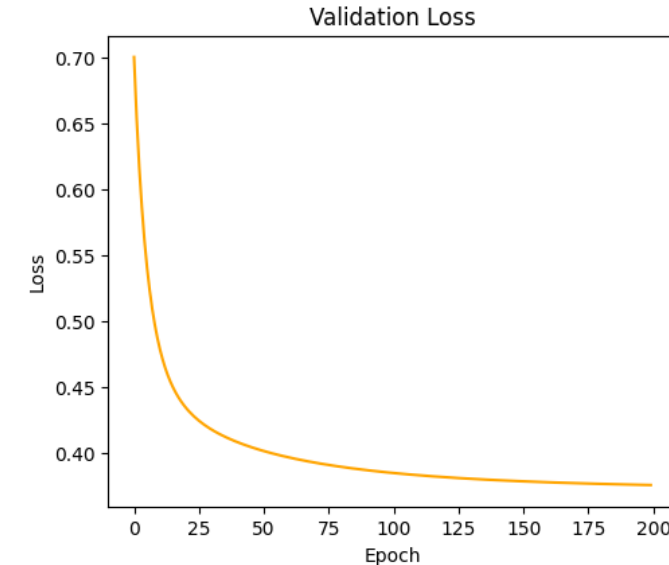
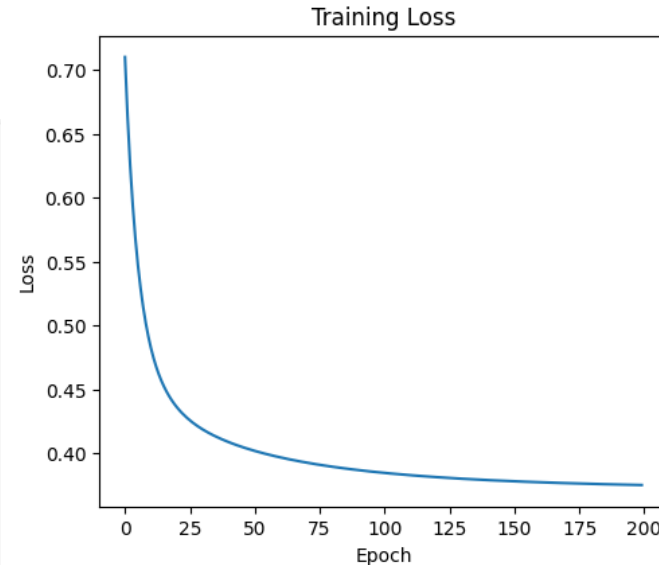
## ❖ Step 7: Training

```
6 for epoch in range(epochs):
7     train_batch_losses = []
8     train_batch_accs = []
9     val_batch_losses = []
10    val_batch_accs = []
11
12    for i in range(0, X_train.shape[0], batch_size):
13        X_i = X_train[i:i+batch_size]
14        y_i = y_train[i:i+batch_size]
15
16        y_hat = predict(X_i, theta)
17        train_loss = compute_loss(y_hat, y_i)
18        gradient = compute_gradient(X_i, y_i, y_hat)
19        theta = update_theta(theta, gradient, lr)
20        train_batch_losses.append(train_loss)
21
22        train_acc = compute_accuracy(X_train, y_train, theta)
23        train_batch_accs.append(train_acc)
24
25        y_val_hat = predict(X_val, theta)
26        val_loss = compute_loss(y_val_hat, y_val)
27        val_batch_losses.append(val_loss)
28
29        val_acc = compute_accuracy(X_val, y_val, theta)
30        val_batch_accs.append(val_acc)
31
32    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
33    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
34    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
35    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

# Sentiment Analysis

## ❖ Step 7: Training (Visualization)

```
1 fig, ax = plt.subplots(2, 2, figsize=(12, 10))
2 ax[0, 0].plot(train_losses)
3 ax[0, 0].set_xlabel='Epoch', ylabel='Loss'
4 ax[0, 0].set_title('Training Loss')
5
6 ax[0, 1].plot(val_losses, 'orange')
7 ax[0, 1].set_xlabel='Epoch', ylabel='Loss'
8 ax[0, 1].set_title('Validation Loss')
9
10 ax[1, 0].plot(train_accs)
11 ax[1, 0].set_xlabel='Epoch', ylabel='Accuracy'
12 ax[1, 0].set_title('Training Accuracy')
13
14 ax[1, 1].plot(val_accs, 'orange')
15 ax[1, 1].set_xlabel='Epoch', ylabel='Accuracy'
16 ax[1, 1].set_title('Validation Accuracy')
17
18 plt.show()
```



# Sentiment Analysis

## ❖ Step 8: Evaluation

$$accuracy = \frac{true\_predictions}{n\_samples}$$

```
1 def compute_accuracy(X, y, theta):
2     y_hat = predict(X, theta).round()
3     acc = (y_hat == y).mean()
4
5     return acc
```

```
1 # Val set
2 val_set_acc = compute_accuracy(X_val, y_val, theta)
3 print('Evaluation on validation set:')
4 print(f'Accuracy: {val_set_acc}')
```

Evaluation on validation set:  
Accuracy: 0.821969696969697

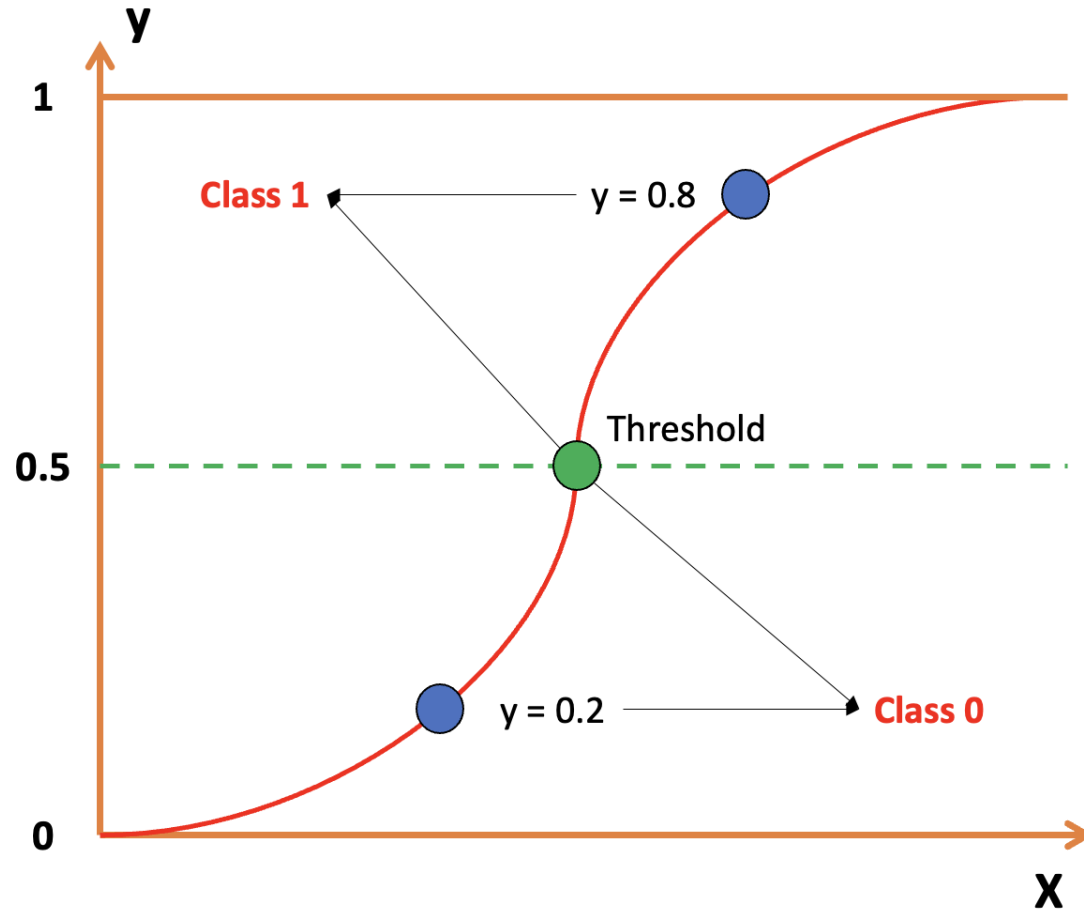
```
1 # Test set
2 test_set_acc = compute_accuracy(X_test, y_test, theta)
3 print('Evaluation on test set:')
4 print(f'Accuracy: {test_set_acc}')
```

Evaluation on test set:  
Accuracy: 0.8434343434343434

# Summarization and QA

# Summarization

## ❖ Content



**In this lecture, we have discussed:**

- What is Logistic Regression? How does it work? How to train it?
- Practice how to implement Logistic Regression with Titanic Survival Prediction and Sentiment Analysis problems.



# Question

---

