AI VIETNAM
All-in-One Course
(TA Session)

Logistic Regression

Exercise

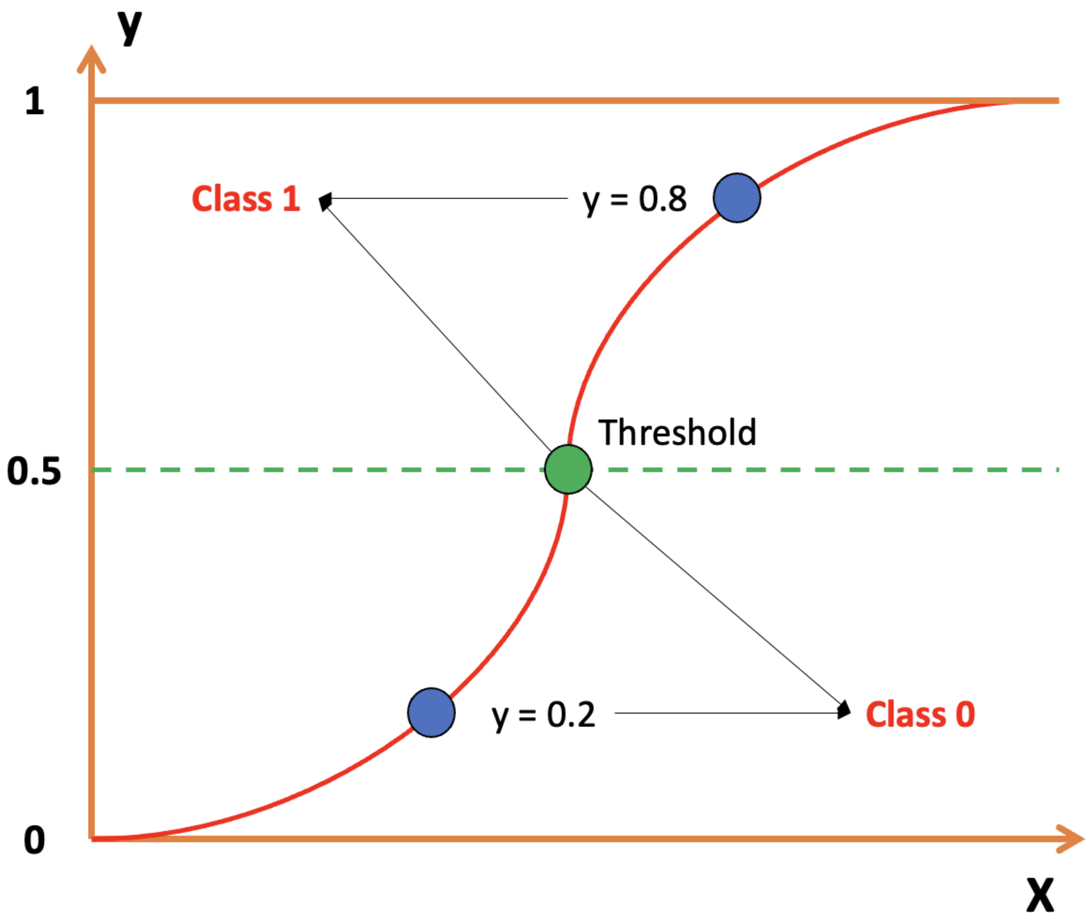
AI VIET NAM
@aivietnam.edu.vn

Dinh-Thang Duong – TA

Year 2024

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

Review

❖ Introduction

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

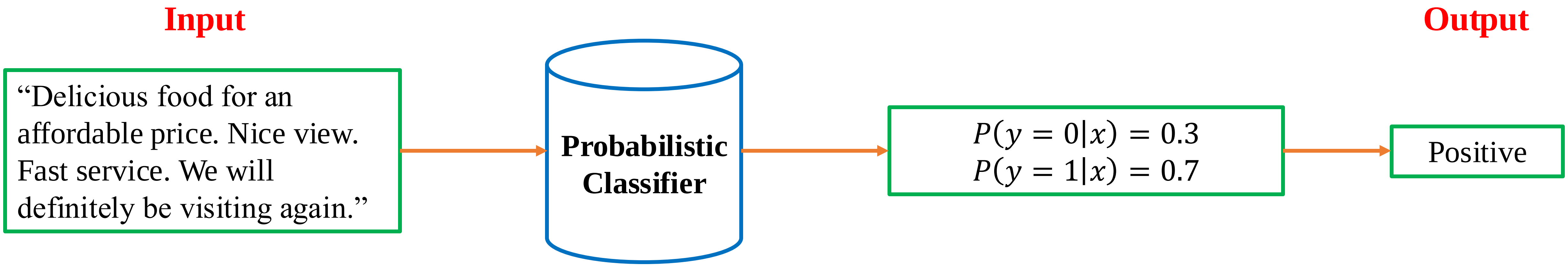
Problem Statement: Given a review, classify it into one of the three classes: Positive, Negative or Neutral.

Input: "Delicious food for an affordable price. Nice view. Fast service. We will definitely be visiting again."
Output: "Positive"

Sentiment Analysis Problem

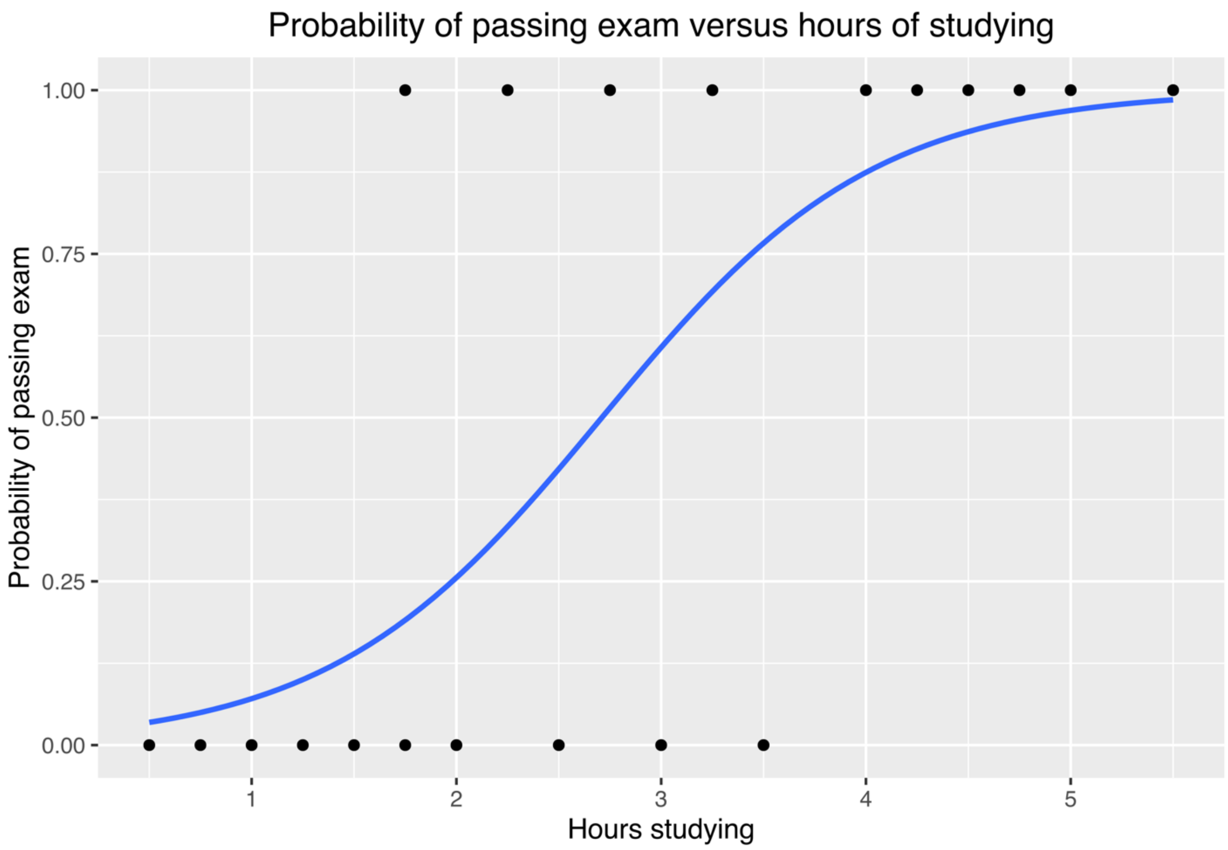
Review

❖ Probabilistic Classifier



Review

❖ 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.

https://en.wikipedia.org/wiki/Logistic_regression

Review

❖ Logistic Regression

Input

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

Logistic Regression

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

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

Output

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

learning

Features X

x_1	x_2	x_3	…	x_n

Weights W

w_1	w_2	w_3	…	w_n

The bias term b

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

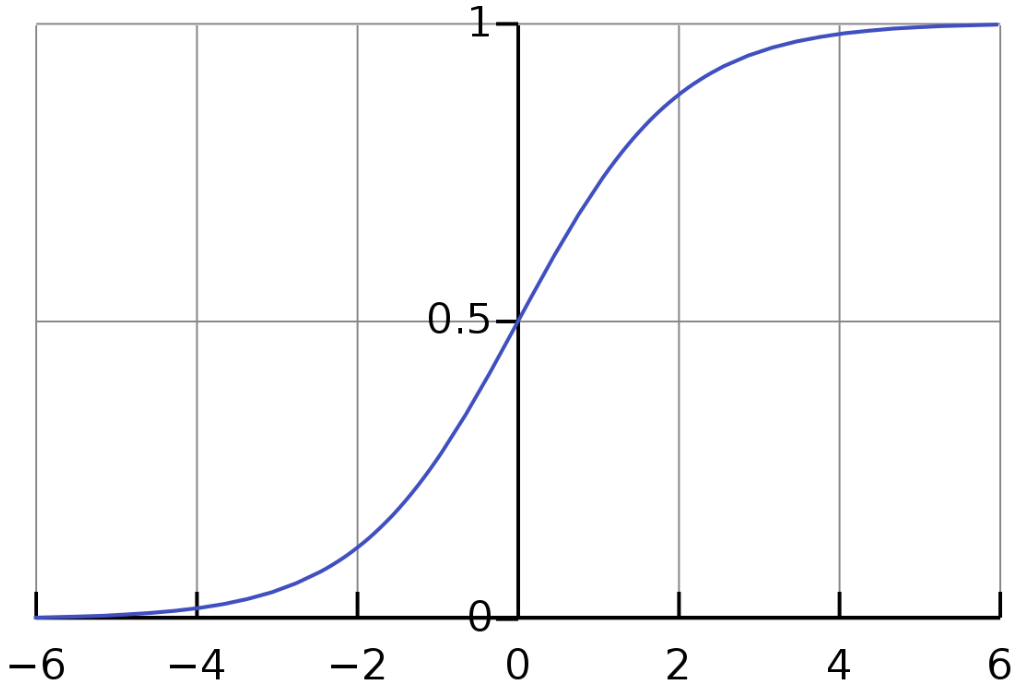
Review

❖ 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.**



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

Review

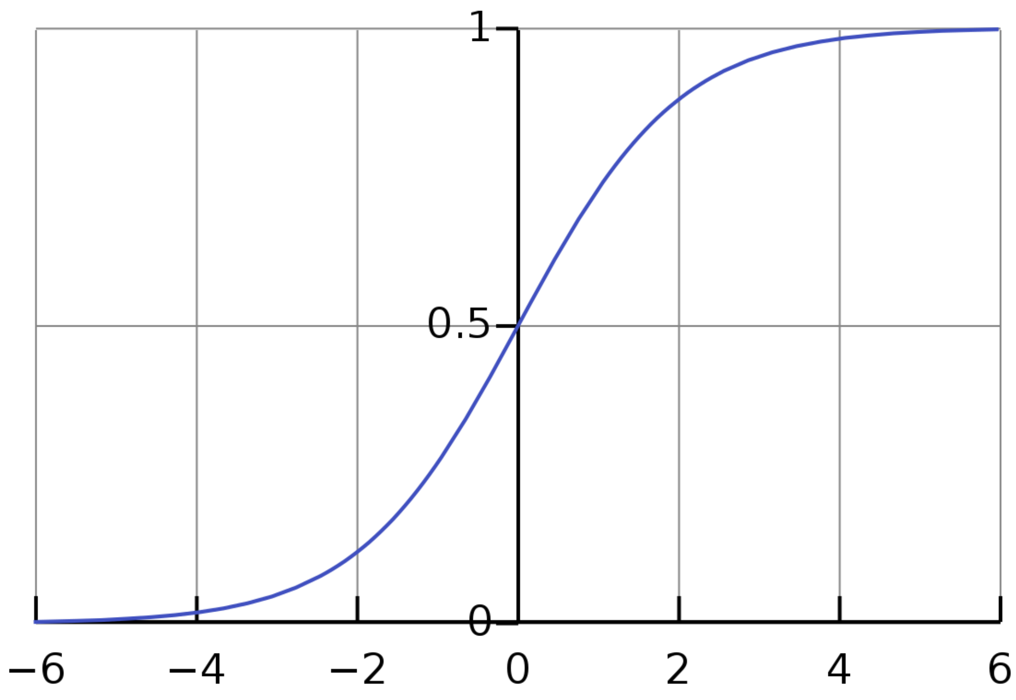
❖ Logistic Regression property

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

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

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

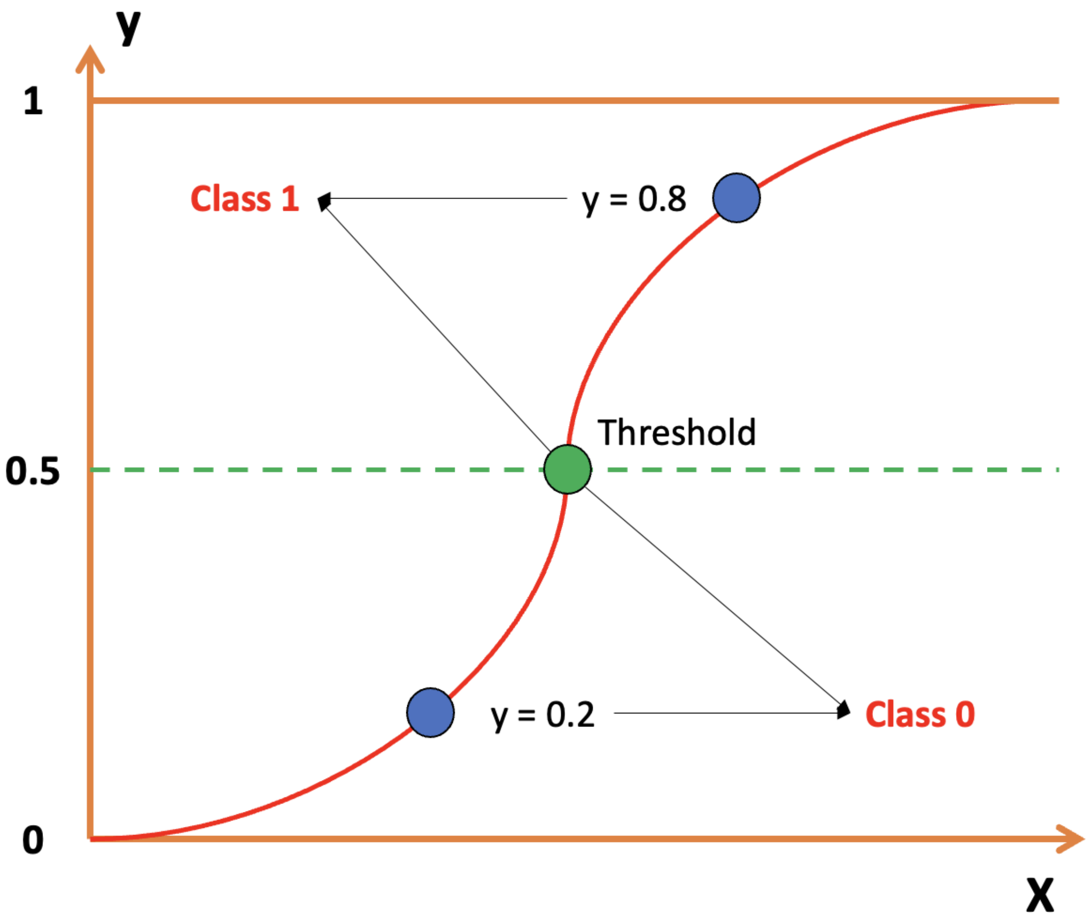
=> $P(y=0) = 1 - \sigma(z) = \sigma(-z)$



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

Review

❖ Logistic Regression Prediction



➢ **When making a prediction:**

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

Review

❖ 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 ŷ (prediction) and y (true label) => $L(\hat{y}, y)$

- $p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$ (Bernoulli Distribution)
- $$\begin{aligned}\log\big(p(y|x)\big) &= \log\left(\hat{y}^y (1-\hat{y})^{1-y}\right) \\ &= \log(\hat{y}^y) + \log\left((1-\hat{y})^{1-y}\right) \\ &= y\log(\hat{y}) + (1-y)\log(1-\hat{y})\end{aligned}$$
- $L(\hat{y}, y) = -\log\big(p(y|x)\big) = -[y log(\hat{y}) + (1-y)\log(1-\hat{y})]$

=> **Cross-entropy loss.**

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

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

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

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

Review

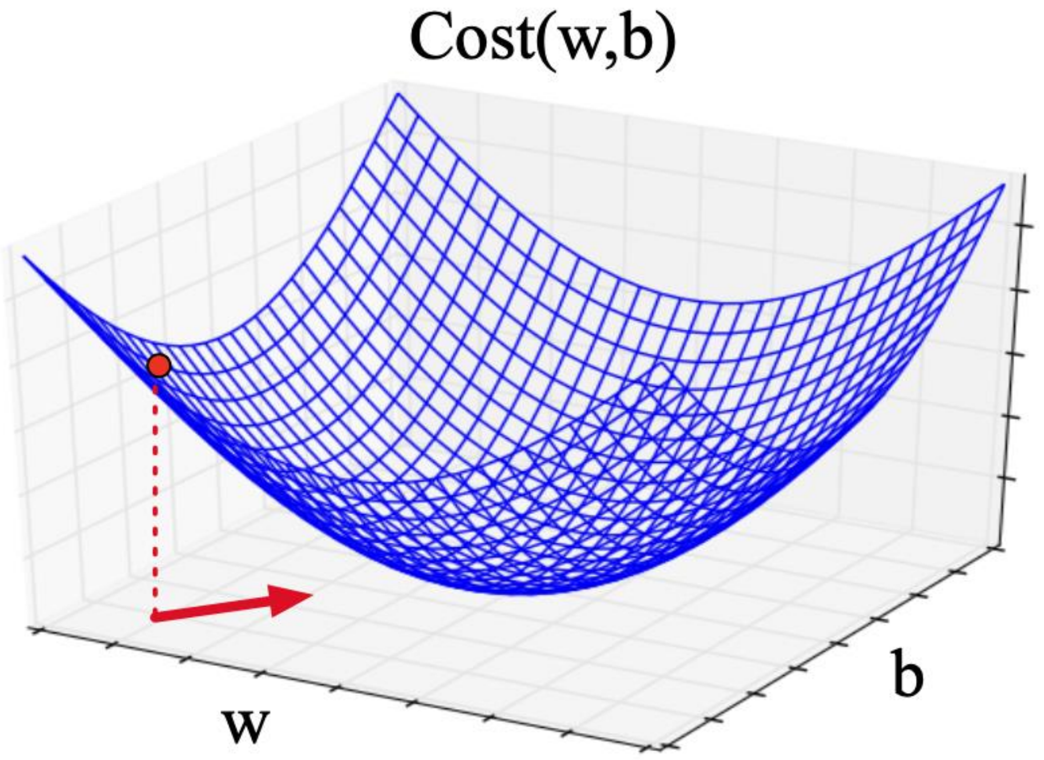
❖ Logistic Regression objective

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

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

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

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



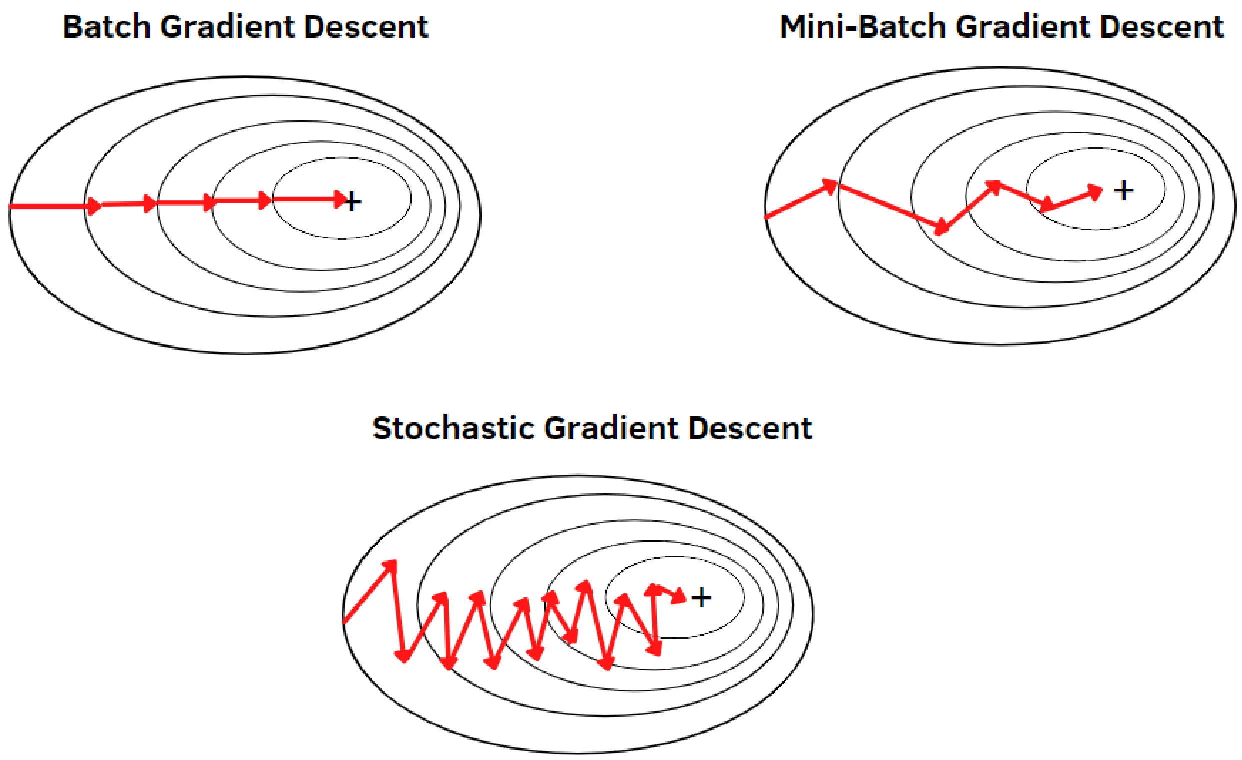
Gradient Descent

Review

❖ 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 < batch_size < N
- **Batch GD:** batch_size = N

AI VIETNAM
All-in-One Course
(TA Session)

Review

❖ Training Logistic Regression

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

2) Compute output $\hat{y}$

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

3) Compute loss

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

4) Compute derivative

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

5) Update parameters

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

η is learning rate

```python
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
```

```python
# Given X and y
```

```python
# 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)
```

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 $(\boldsymbol{x}, y)$ from training data

2) Compute output $\hat{y}$

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

3) Compute loss

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

4) Compute derivative

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

5) Update parameters

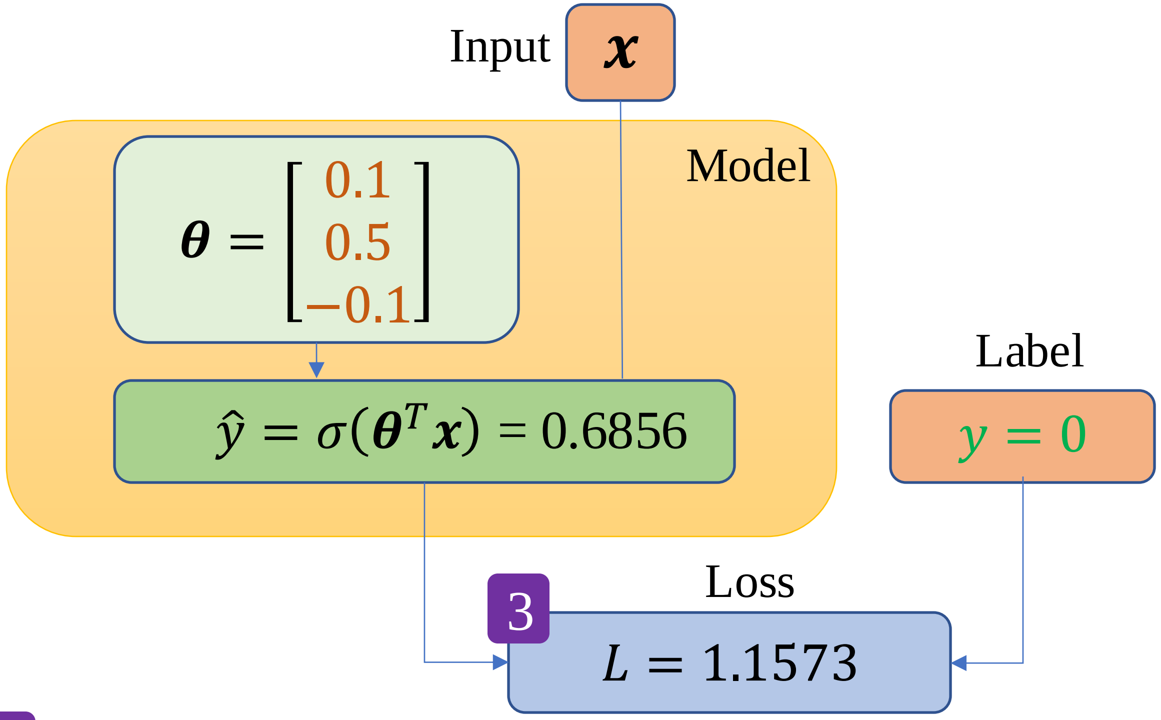
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L$$

1

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.4 \\ 0.2 \end{bmatrix}$$

$$\text{Given } \boldsymbol{\theta} = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.5 \\ -0.1 \end{bmatrix}$$

$$\eta = 0.01$$



4

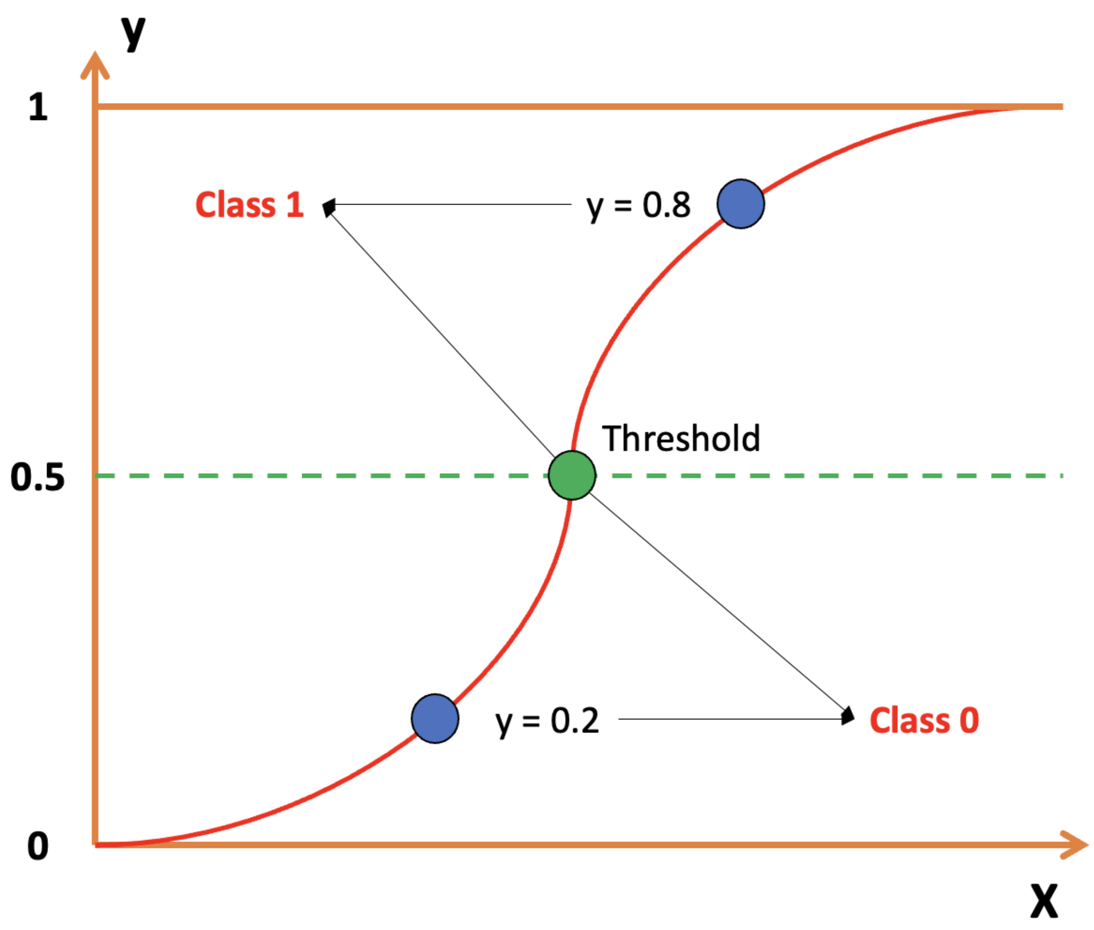
$$\nabla_{\boldsymbol{\theta}} L = \boldsymbol{x}(\hat{y} - y) = \begin{bmatrix} 1 \\ 1.4 \\ 0.2 \end{bmatrix} [0.6856] = \begin{bmatrix} 0.6856 \\ 0.9599 \\ 0.1371 \end{bmatrix} = \begin{bmatrix} L'_b \\ L'_{w_1} \\ L'_{w_2} \end{bmatrix}$$

5

$$\boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}} = \begin{bmatrix} 0.1 \\ 0.5 \\ -0.1 \end{bmatrix} - \eta \begin{bmatrix} 0.6856 \\ 0.9599 \\ 0.1371 \end{bmatrix} = \begin{bmatrix} 0.093 \\ 0.499 \\ -0.101 \end{bmatrix}$$

Review

❖ Logistic Regression



Logistic Regression (LR)

Sigmoid function

$$z = b_0 + \sum_{i=1}^{n\_features} b_i x_i$$

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

Binary Cross Entropy:

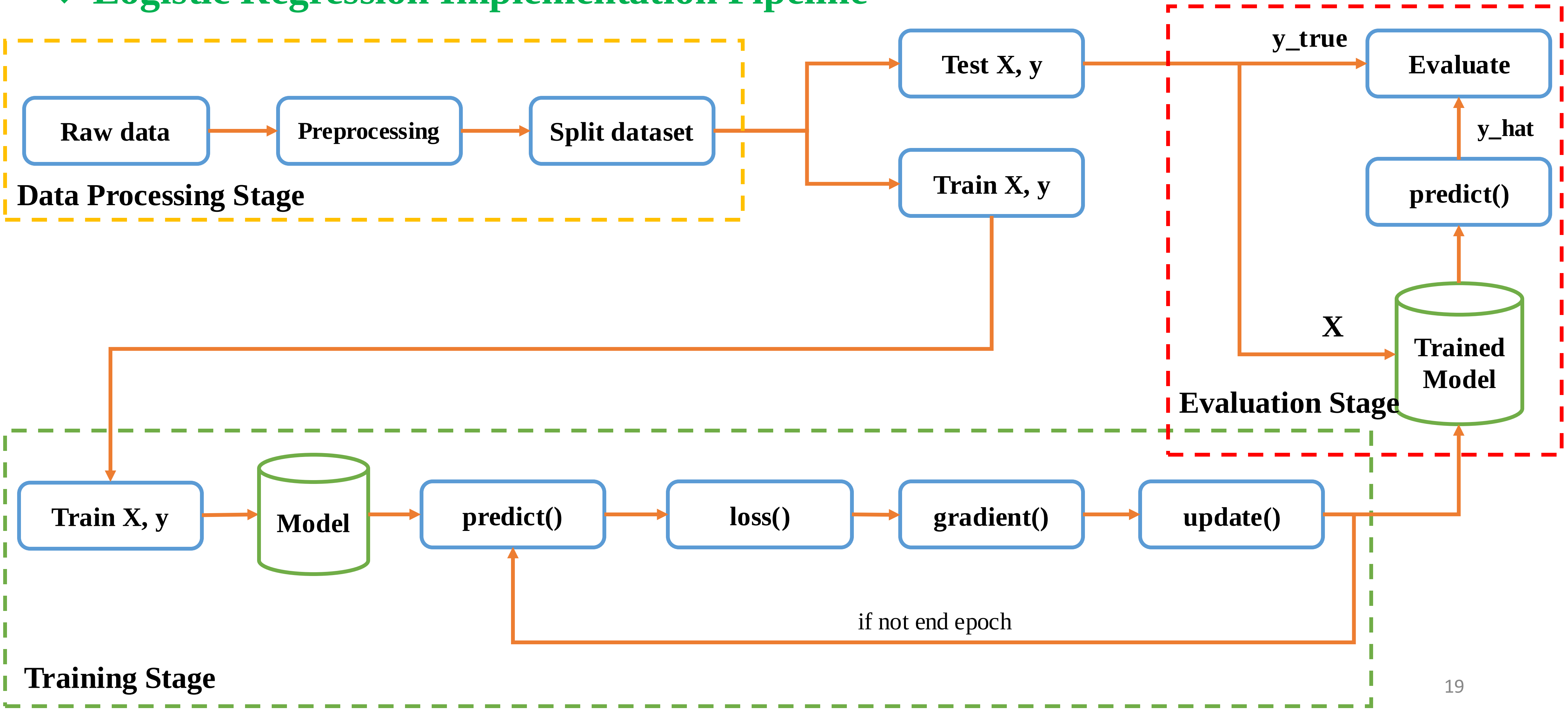
$$L(\hat{y}, y) = -\frac{1}{N}\left[y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})\right]$$

Gradient:

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

Review

❖ Logistic Regression Implementation Pipeline



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

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
dataset_path = 'titanic_modified_dataset.csv'
df = pd.read_csv(
    dataset_path,
    index_col='PassengerId'
)
df
```

PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Survived
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 × 9 columns

Titanic Survival Prediction

❖ Step 2: Get dataset information

```
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

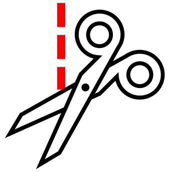
❖ Step 3: Split X, y

X

y

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

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)

```
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
```

```
intercept = np.ones((
    X.shape[0], 1)
)
X_b = np.concatenate(
    (intercept, X),
    axis=1
)
```

```
intercept
```

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

```
X
```

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

New shape of X after adding intercepts:

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

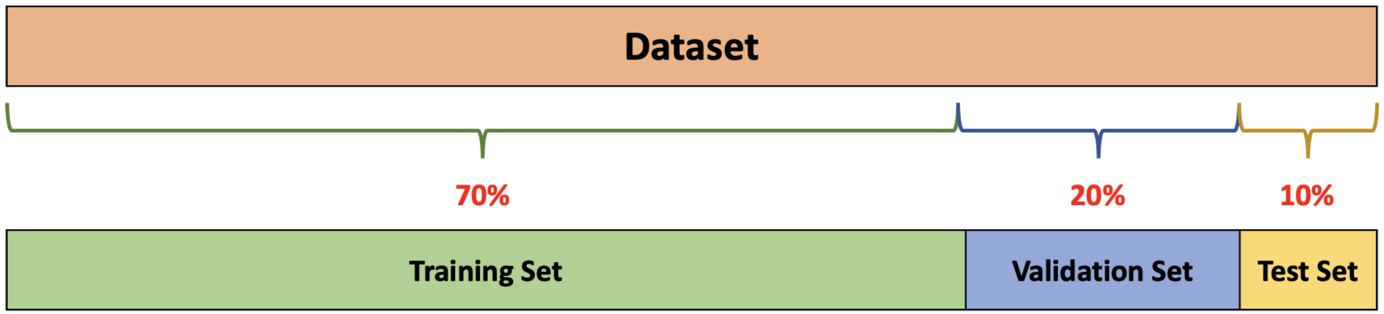
```
(891, 9)
```

```
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




```python
val_size = 0.2
test_size = 0.125
random_state = 2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X_b, y,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

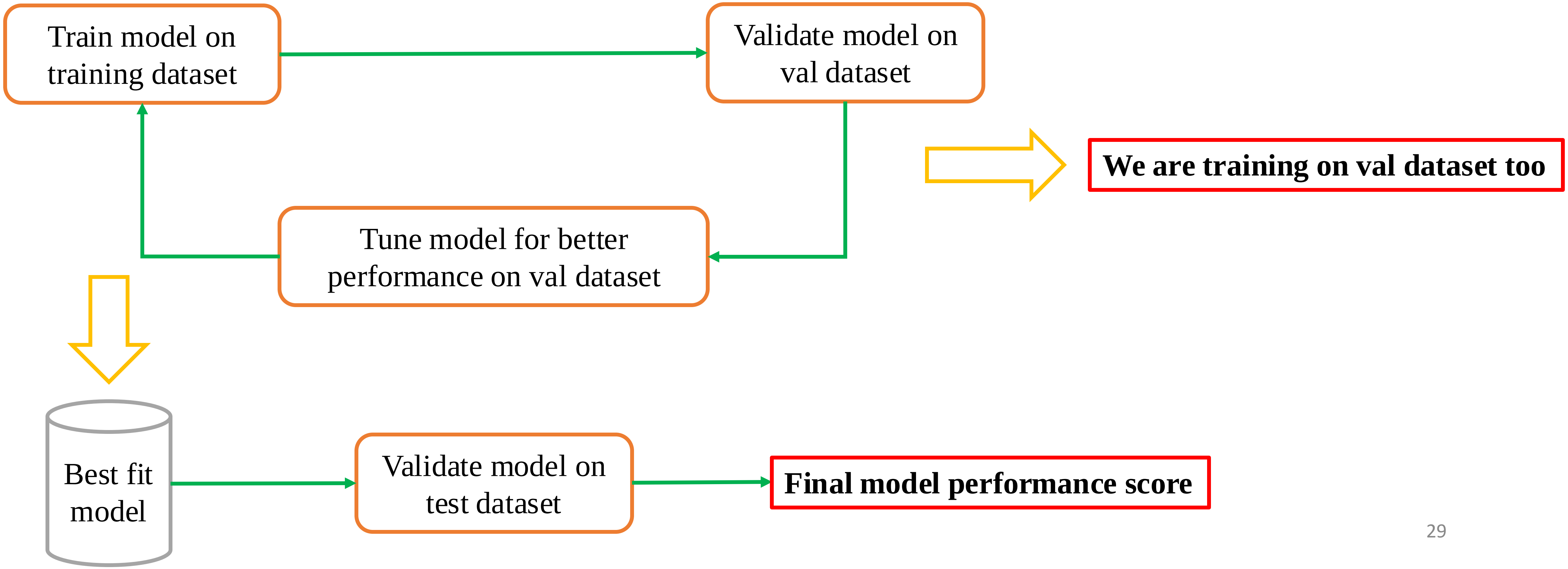
```python
print(f'Number of training samples: {X_train.shape[0]}')
print(f'Number of val samples: {X_val.shape[0]}')
print(f'Number of test samples: {X_test.shape[0]}')
```

```
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

Why we need test set?



Titanic Survival Prediction

❖ Step 5: Normalization

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

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

```
normalizer = StandardScaler()
X_train = normalizer.fit_transform(X_train)
X_val = normalizer.transform(X_val)
X_test = normalizer.transform(X_test)
```

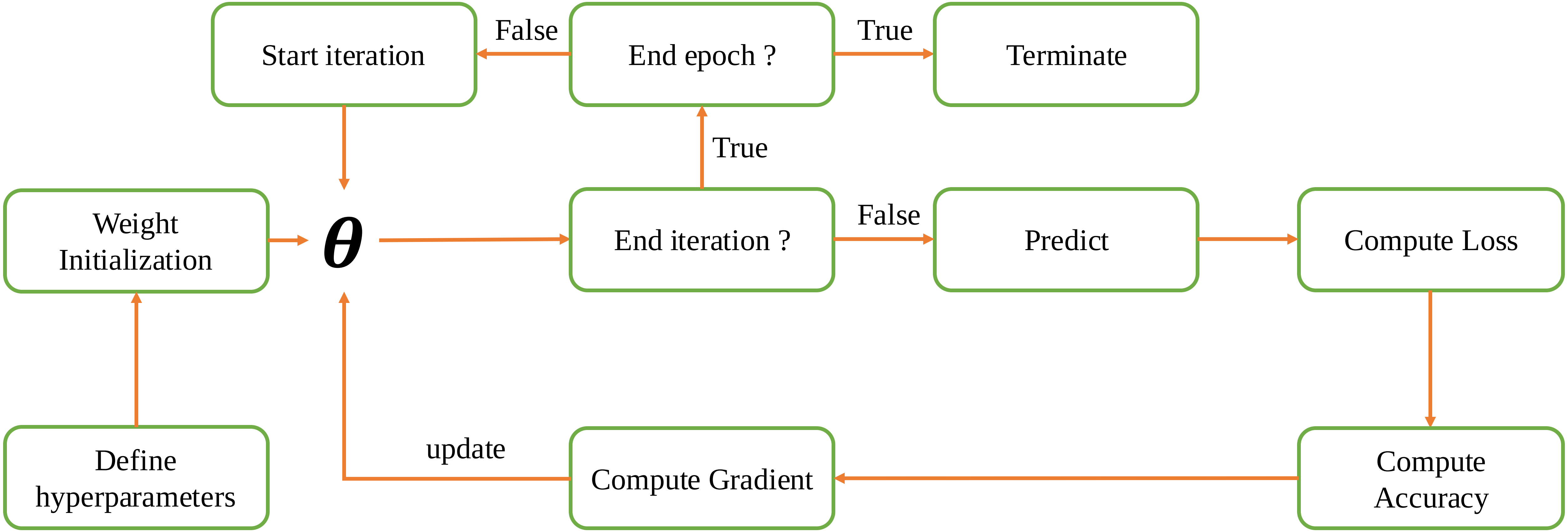
Note: We only use the train set to fit the scaler.

```
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.        ]])
```

Titanic Survival Prediction

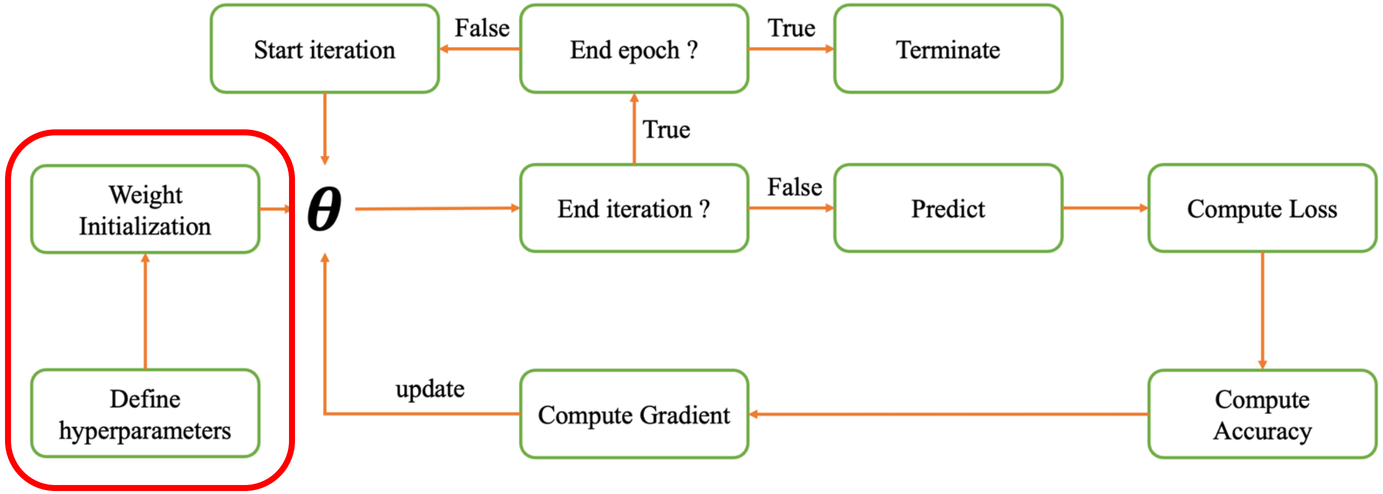
❖ Step 6: Training



Titanic Survival Prediction

❖ Step 6: Training

```
lr = 0.01
epochs = 100
batch_size = 16

np.random.seed(random_state)
theta = np.random.uniform(
    size=X_train.shape[1]
)
```



Define essential hyperparameters and initialize weights.

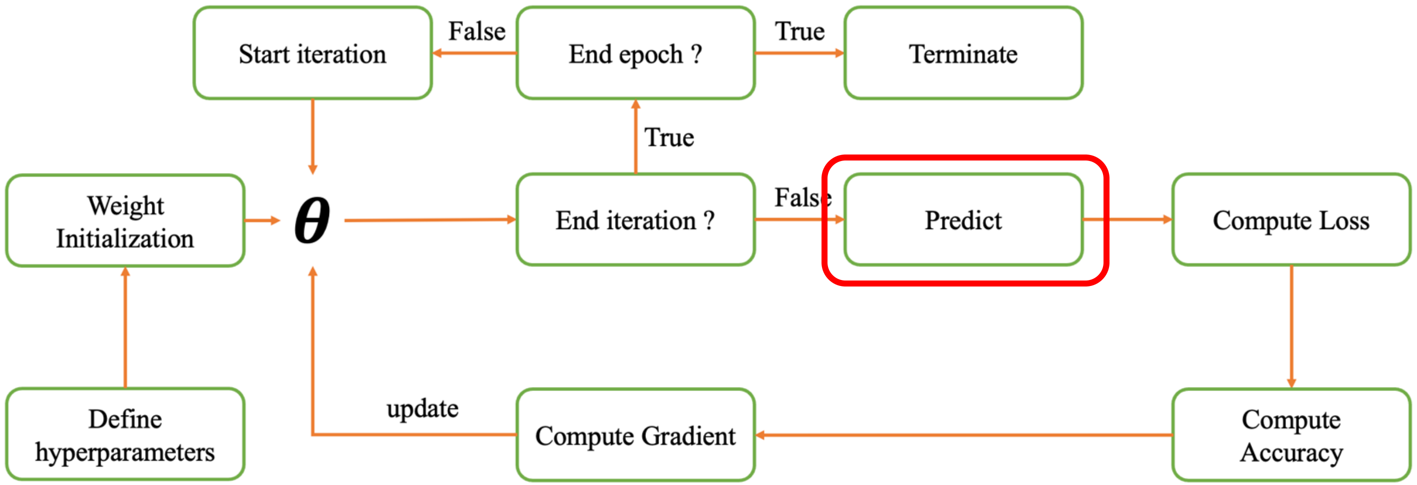
Titanic Survival Prediction

❖ Step 6: Training

Hypothesis function

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

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


def predict(X, theta):
    dot_product = np.dot(X, theta)
    y_hat = sigmoid(dot_product)

    return y_hat
```



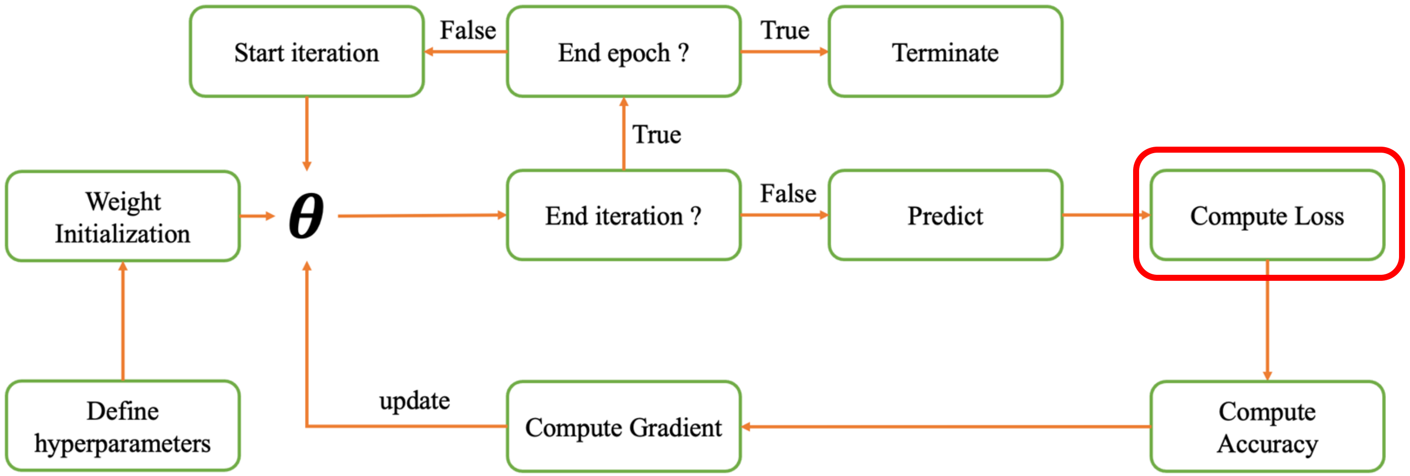
Titanic Survival Prediction

❖ Step 6: Training

Logistic Regression Loss (Cross-entropy)

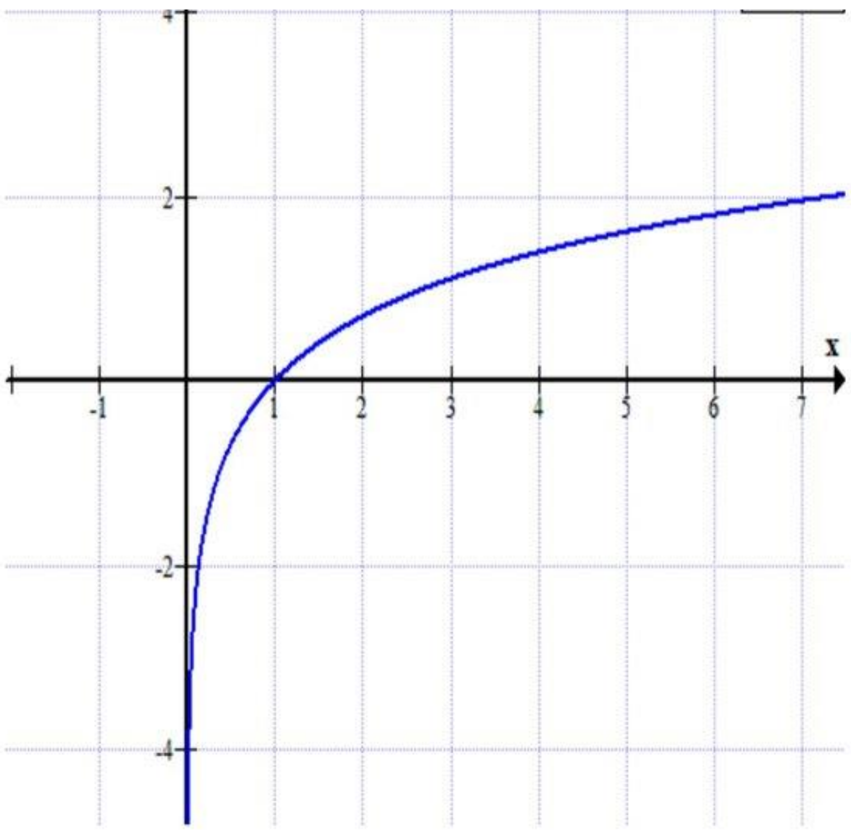
$$L(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

```python
def loss(y_hat, y):
    y_hat = np.clip(
        y_hat, 1e-7, 1 - 1e-7
    )

    return (
        -y * \
        np.log(y_hat) - (1 - y) * \
        np.log(1 - y_hat)
    ).mean()
```



Titanic Survival Prediction

❖ Step 6: Training




```python
def loss(y_hat, y):
    y_hat = np.clip(
        y_hat, 1e-7, 1 - 1e-7
    )

    return (
        -y * \
        np.log(y_hat) - (1 - y) * \
        np.log(1 - y_hat)
    ).mean()
```

→ Can cause nan if log(0).

Avoid prediction values too small by limiting the value range.

Titanic Survival Prediction

❖ Step 6: Training

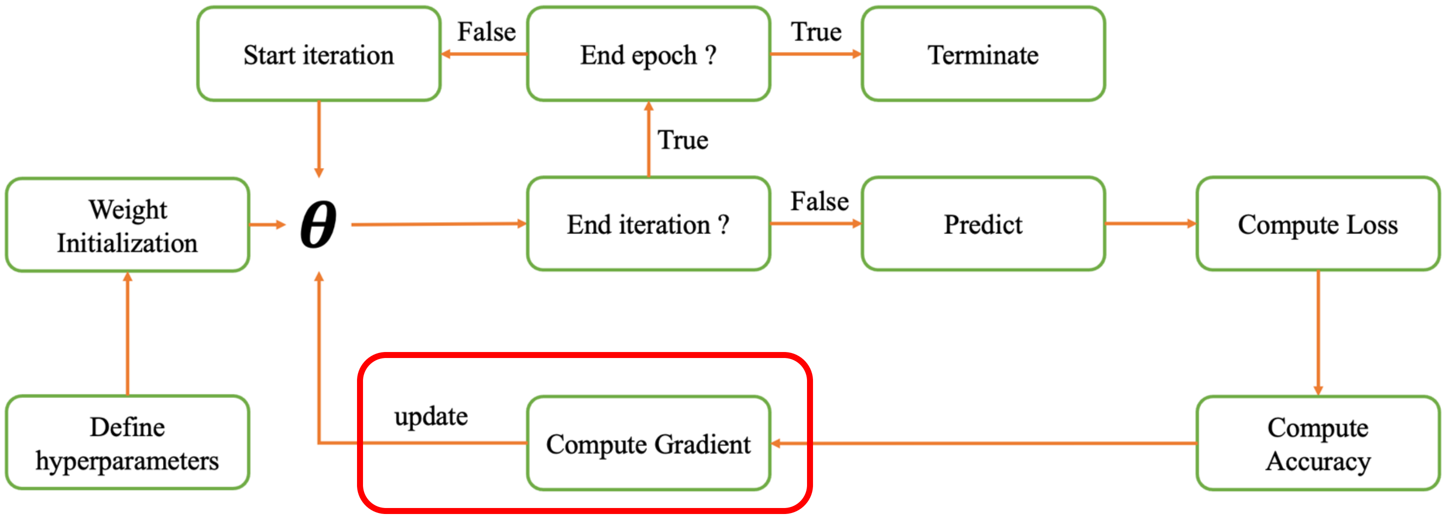
- **Gradient computation formula:**

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

- **Weights update formula:**

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

```python
def compute_gradient(X, y, y_hat):
    return np.dot(
        X.T, (y_hat - y)
    ) / y.size

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



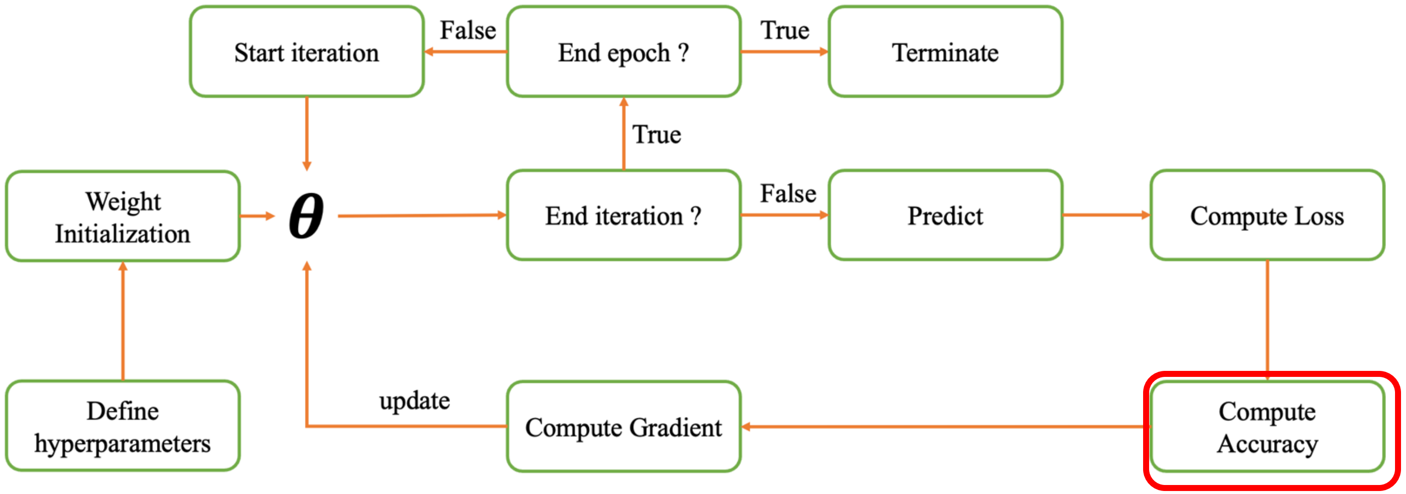
Titanic Survival Prediction

❖ Step 6: Training

Accuracy Formula

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

```python
def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta).round()
    acc = (y_hat == y).mean()

    return acc
```



Titanic Survival Prediction

❖ Step 6: 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)
```

Line 6: Loop over number of epochs.

Line 7, 8, 9, 10: Declare empty lists to stor batche accuracies, losses of train and val sets.

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)
```

AI VIETNAM
All-in-One Course
(TA Session)

Titanic Survival Prediction

❖ Step 6: Training

Line 16: Do prediction on X_i and theta.

Line 17: Compute the loss of y_hat 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

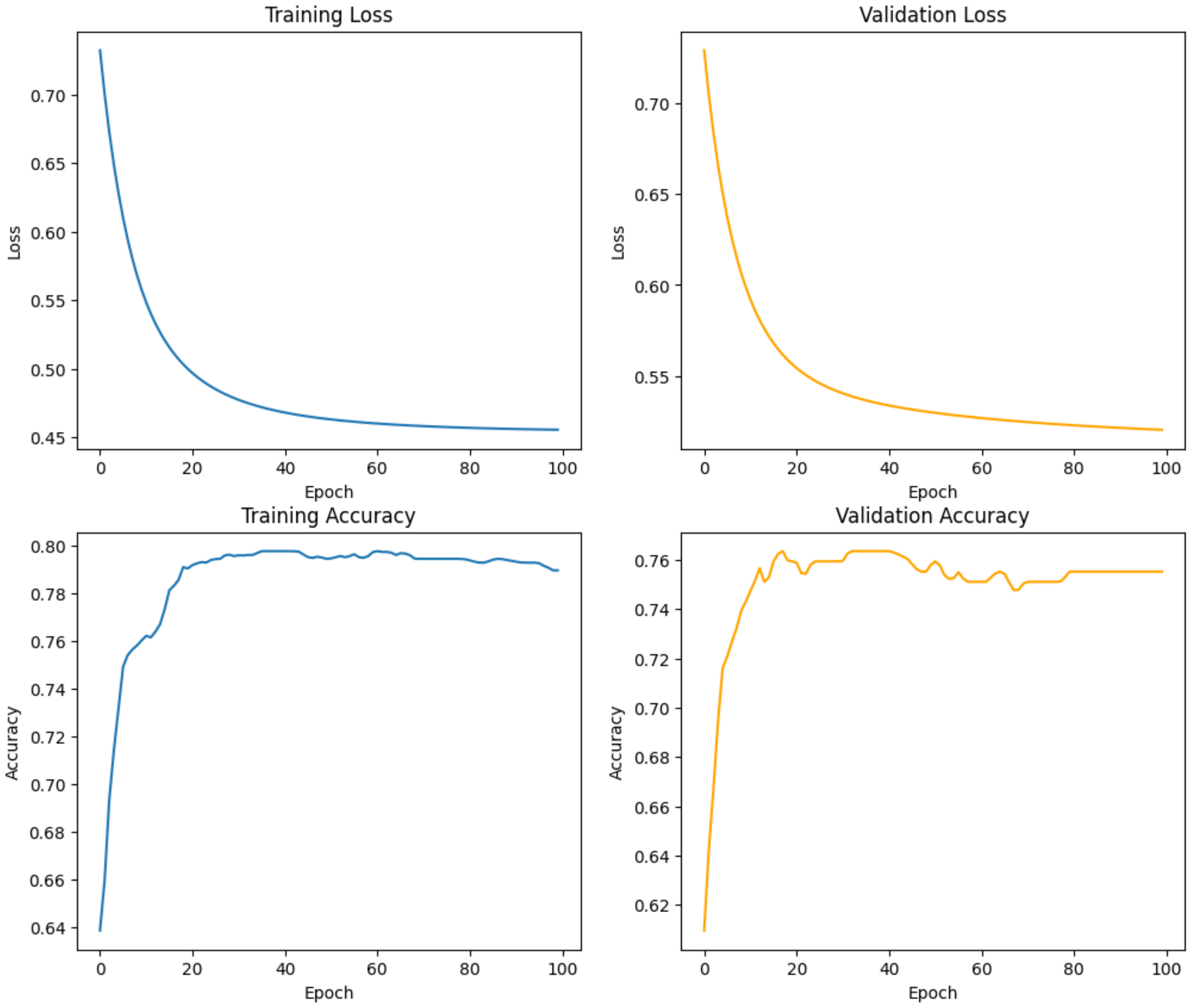
From line 32 to line 35: Compute and store batch accuracies and losses of train and val sets.

```
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)

```python
fig, ax = plt.subplots(2, 2, figsize=(12, 10))
ax[0, 0].plot(train_losses)
ax[0, 0].set(xlabel='Epoch', ylabel='Loss')
ax[0, 0].set_title('Training Loss')

ax[0, 1].plot(val_losses, 'orange')
ax[0, 1].set(xlabel='Epoch', ylabel='Loss')
ax[0, 1].set_title('Validation Loss')

ax[1, 0].plot(train_accs)
ax[1, 0].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 0].set_title('Training Accuracy')

ax[1, 1].plot(val_accs, 'orange')
ax[1, 1].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 1].set_title('Validation Accuracy')

plt.show()
```



Titanic Survival Prediction

❖ Step 7: Evaluation

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

```python
def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta).round()
    acc = (y_hat == y).mean()

    return acc
```

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

```
Evaluation on validation set:
Accuracy: 0.770949720670391
```

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

```
Evaluation on test set:
Accuracy: 0.7752808988764045
```

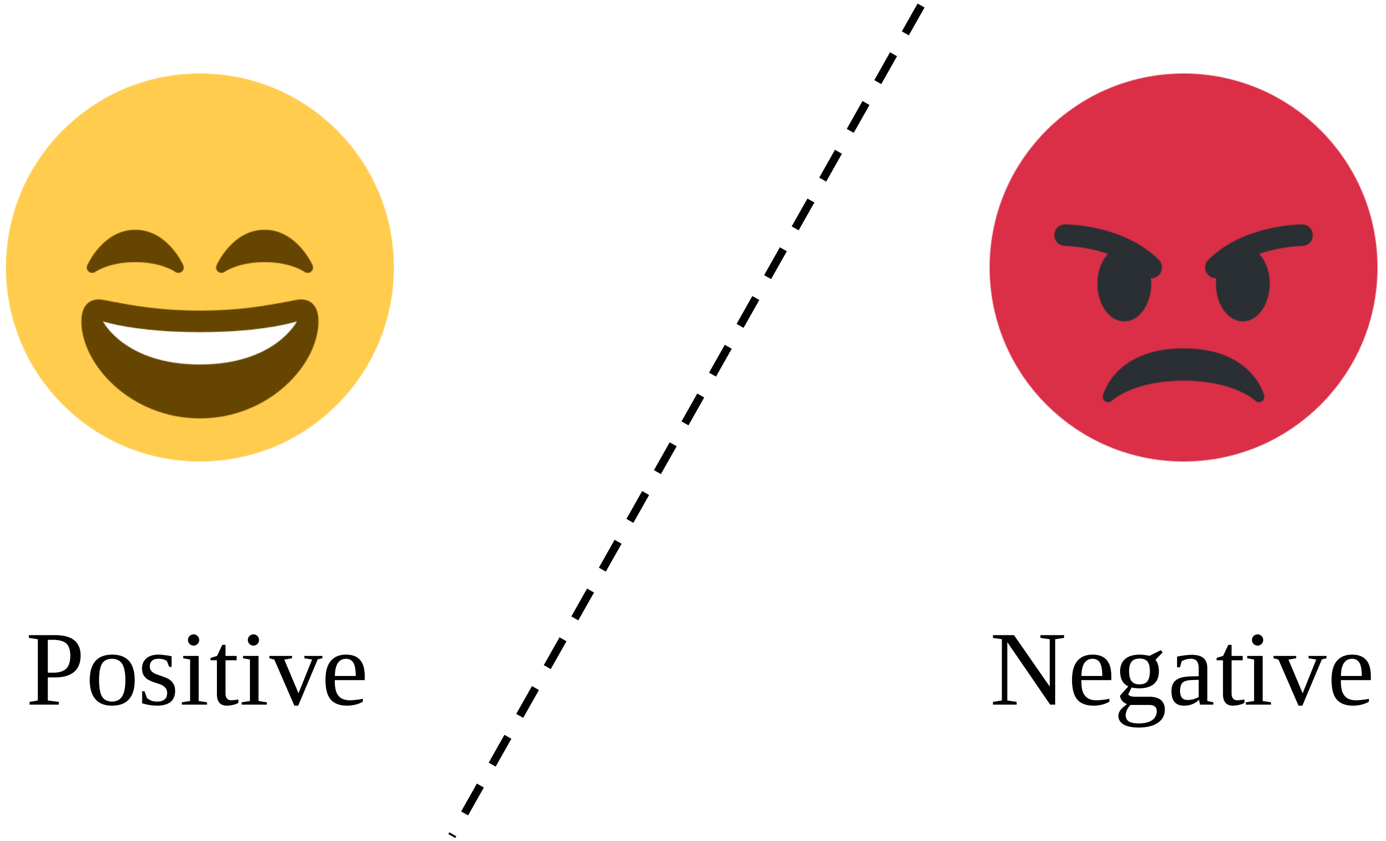
QUIZ

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.



Sentiment Analysis

❖ Step 1: Import libraries

```python
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from nltk.tokenize import TweetTokenizer
from collections import defaultdict
```

Sentiment Analysis

❖ Step 2: Read dataset

```
dataset_path = 'sentiment_analysis.csv'
df = pd.read_csv(
    dataset_path,
    index_col='id'
)
df
```

	label	tweet
id		
1	0	#fingerprint #Pregnancy Test https://goo.gl/h1...
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 × 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

↓

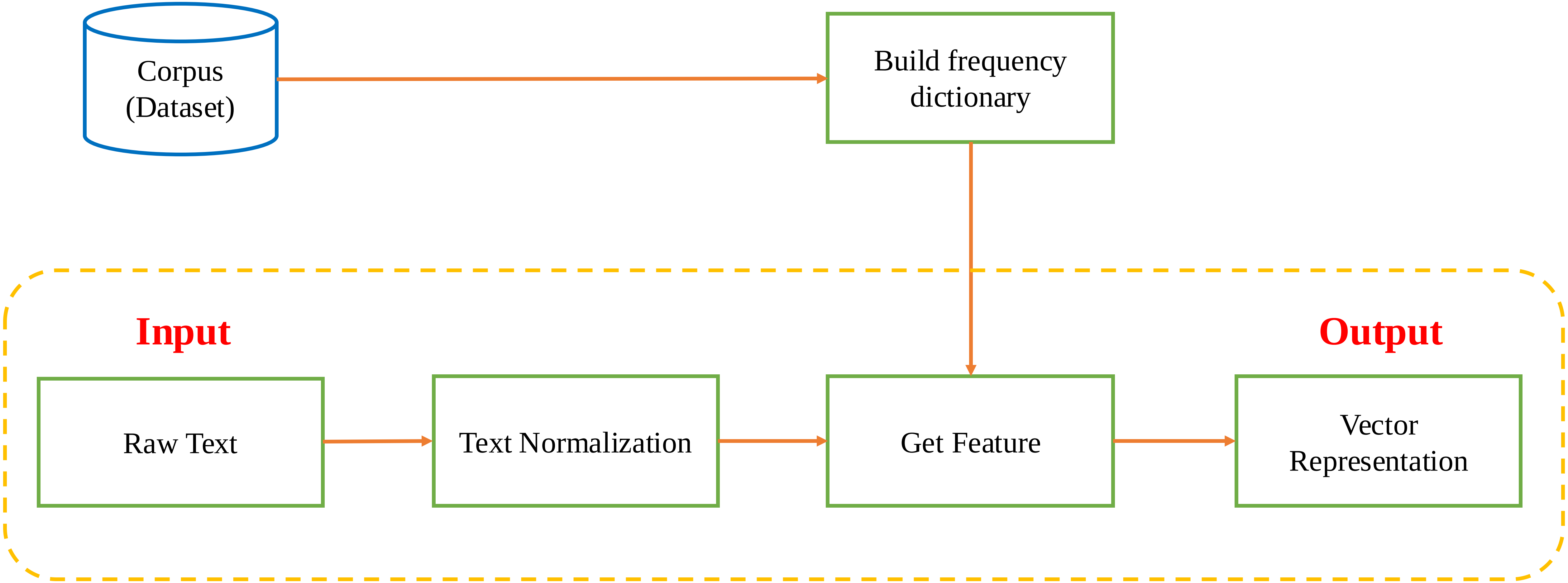
Vector Representation: *array([1.0, 4768.0, 1425.0])*

Vector Representation Meaning

0	1	2
Bias	Positive Word Count	Negative Word Count

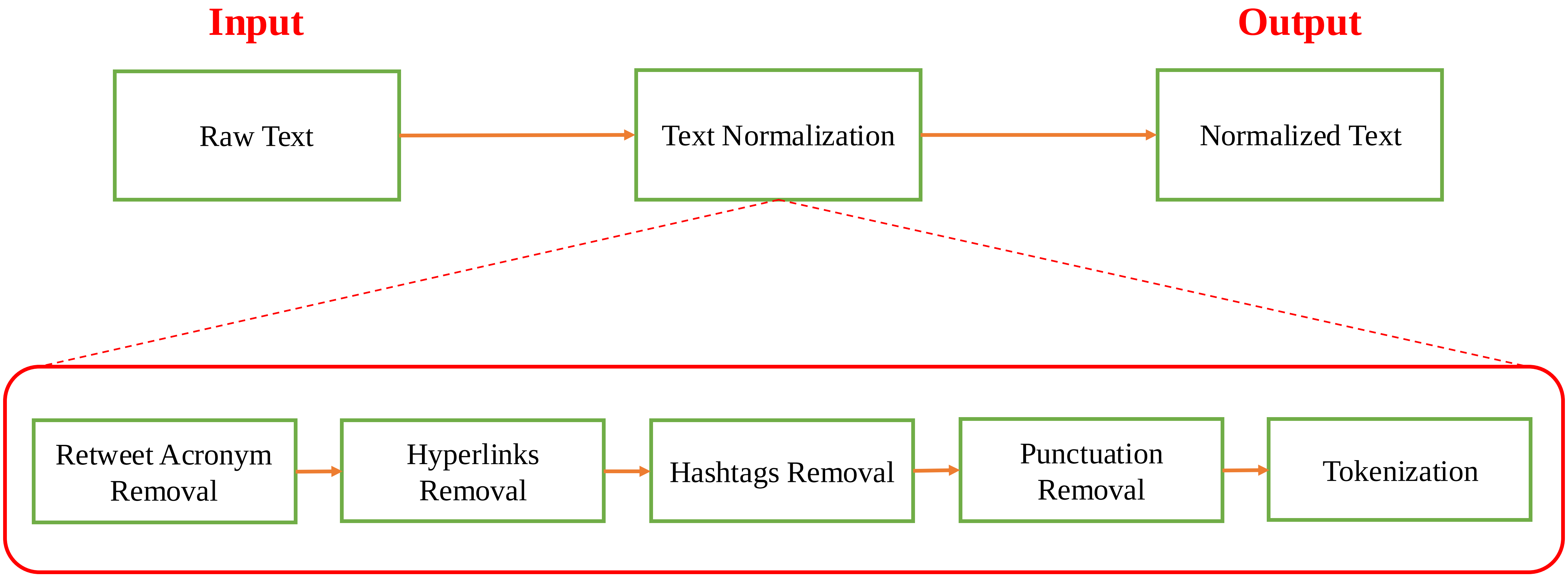
Sentiment Analysis

❖ Step 4: Data preprocessing



Sentiment Analysis

❖ Step 4: Data preprocessing



Sentiment Analysis

❖ Step 4: Data preprocessing

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

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

```
Hello World
```

Hyperlinks Removal: Remove all hyperlinks in the string.

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

Hashtag Removal: Remove all hashtag symbols "#".

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

```
xinchao hello konnichiwa
```

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

```
text = "hello world"
tokenizer = TweetTokenizer(
    preserve_case=False,
    strip_handles=True,
    reduce_len=True
)
text_tokens = tokenizer.tokenize(text)
print(text_tokens)
```

```
['hello', 'world']
```

Sentiment Analysis

❖ Step 4: Data preprocessing

```python
def text_normalize(text):
    # Retweet old acronym "RT" removal
    text = re.sub(r'^RT[\s]+', '', text)

    # Hyperlinks removal
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text)

    # Punctuation removal
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenization
    tokenizer = TweetTokenizer(
        preserve_case=False,
        strip_handles=True,
        reduce_len=True
    )
    text_tokens = tokenizer.tokenize(text)

    return text_tokens
```

```python
text = """We love this! Would you go?
#talk #makememories #unplug
 #relax #iphone #smartphone #wifi #connect...
http://fb.me/6N3LsUpCu
"""
text = text_normalize(text)
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 occurences of each word in positive and negative sentences.

Sentiment Analysis

❖ Step 4: Data preprocessing

Build a dictionary containing number of occurences 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]

Get Feature

Output: [1, 65, 5]

Sentiment Analysis

❖ Step 4: Data preprocessing

```python
def get_freqs(df):
    freqs = defaultdict(lambda: 0)
    for idx, row in df.iterrows():
        tweet = row['tweet']
        label = row['label']

        tokens = text_normalize(tweet)
        for token in tokens:
            pair = (token, label)
            freqs[pair] += 1

    return freqs

freqs = get_freqs(df)
```

```
{('fingerprint', 0): 4,
 ('pregnancy', 0): 1,
 ('test', 0): 8,
 ('finally', 0): 168,
 ('a', 0): 727,
 ('transparant', 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

```python
def get_freqs(df):
    freqs = defaultdict(lambda: 0)
    for idx, row in df.iterrows():
        tweet = row['tweet']
        label = row['label']

        tokens = text_normalize(tweet)
        for token in tokens:
            pair = (token, label)
            freqs[pair] += 1

    return freqs

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.

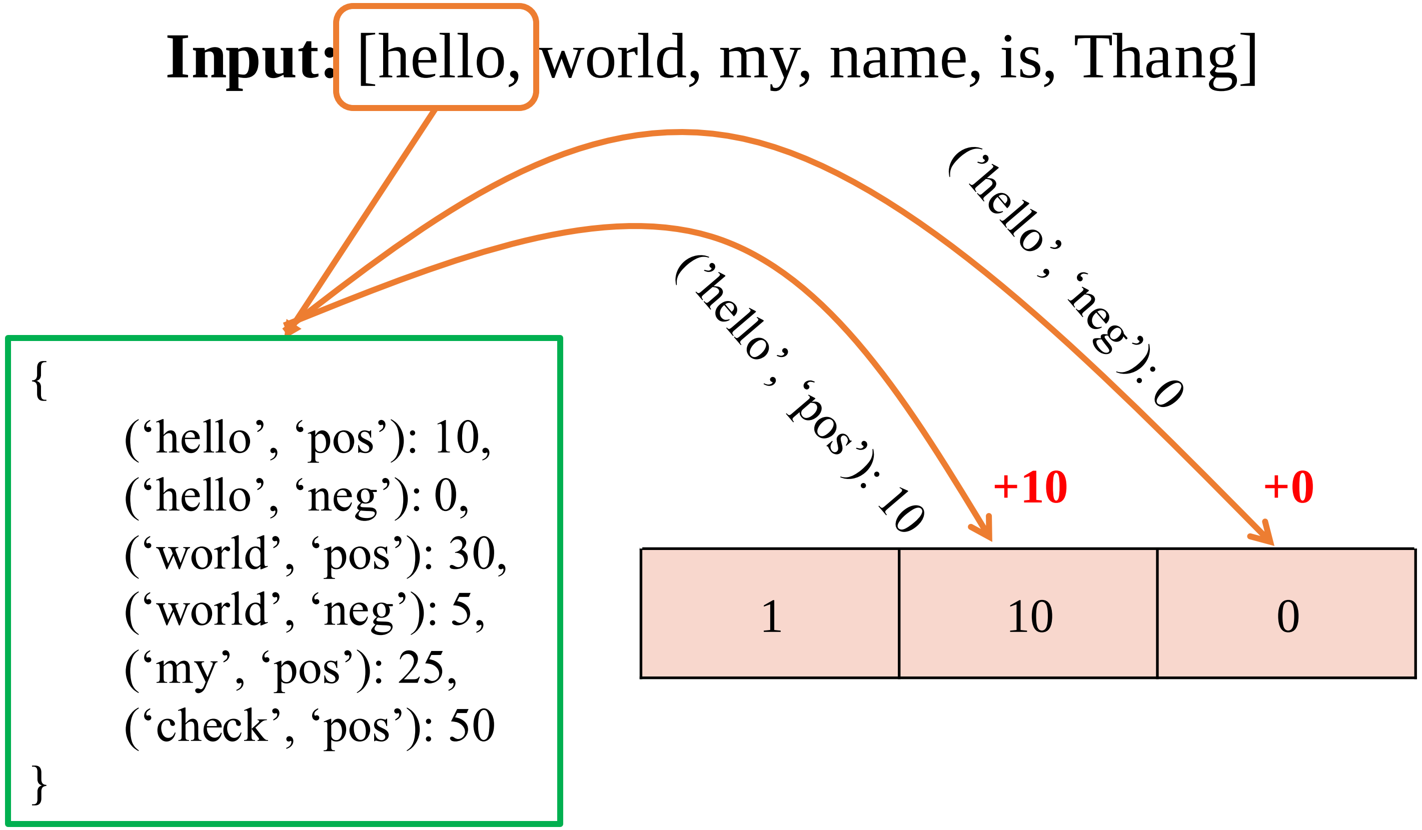
defaultdict benefit: Assign a default value for non-existing keys. For e.g., **lambda: 0** will assign value 0 to keys excluded from the dictionary.

Sentiment Analysis

❖ Step 4: Data preprocessing

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

```python
def get_feature(text, freqs):
    tokens = text_normalize(text)

    X = np.zeros(3)
    X[0] = 1

    for token in tokens:
        X[1] += freqs[(token, 0)]
        X[2] += freqs[(token, 1)]

    return X
```



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

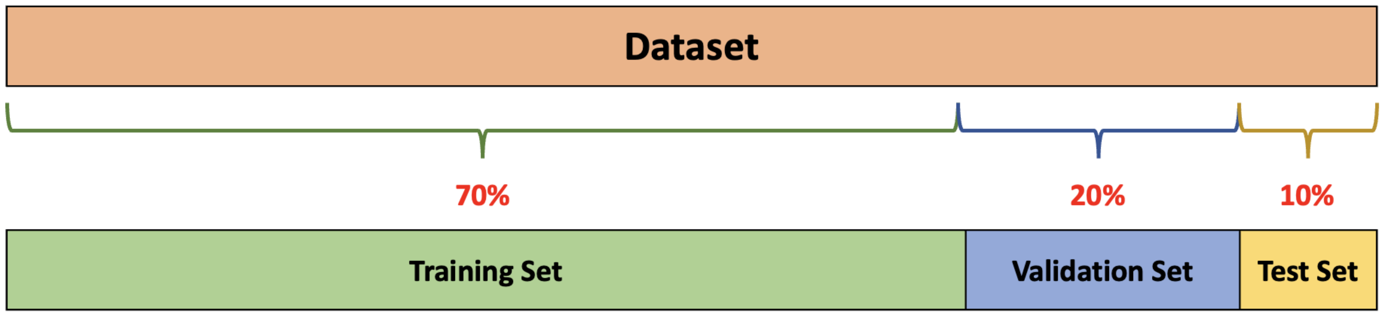
Vector Representation: *array([1.0, 4768.0, 1425.0])*

```python
X = []
y = []

for idx, row in df.iterrows():
    tweet = row['tweet']
    label = row['label']

    X_i = get_feature(tweet, freqs)
    X.append(X_i)
    y.append(label)

X = np.array(X)
y = np.array(y).reshape(-1, 1)

print(f'X shape: {X.shape}')
print(f'y shape: {y.shape}')
```

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

Sentiment Analysis

❖ Step 5: Split train, val, test set




```python
val_size = 0.2
test_size = 0.125
random_state = 2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X, y,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

```python
print(f'Number of training samples: {X_train.shape[0]}')
print(f'Number of val samples: {X_val.shape[0]}')
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.

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

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

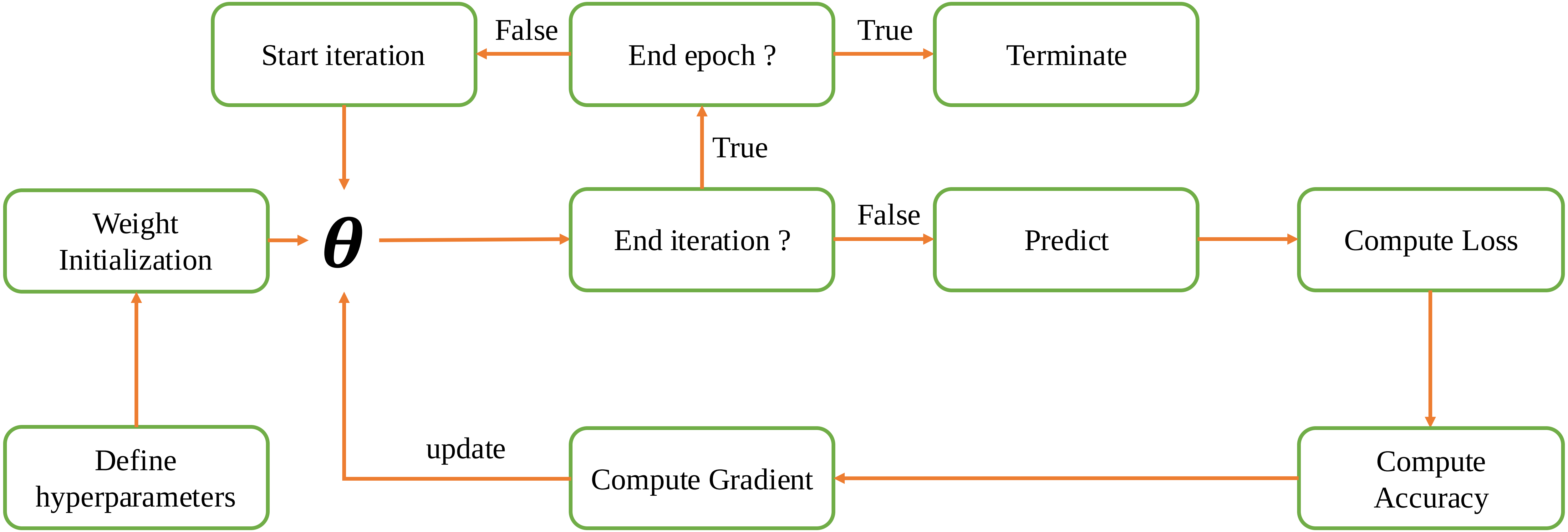
```python
X_train
```

```
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

```python
lr = 0.01
epochs = 200
batch_size = 128

np.random.seed(random_state)
theta = np.random.uniform(
    size=X_train.shape[1]
)
```

Define essential functions and hyperparameters

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

def predict(X, theta):
    dot_product = np.dot(X, theta)
    y_hat = sigmoid(dot_product)

    return y_hat

def compute_loss(y_hat, y):
    y_hat = np.clip(
        y_hat, 1e-7, 1 - 1e-7
    )

    return (-y * np.log(y_hat) - (1 - y) * np.log(1 - y_hat)).mean()

def compute_gradient(X, y, y_hat):
    return np.dot(
        X.T, (y_hat - y)
    ) / y.size

def update_theta(theta, gradient, lr):
    return theta - lr * gradient

def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta).round()
    acc = (y_hat == y).mean()

    return acc
```

Sentiment Analysis

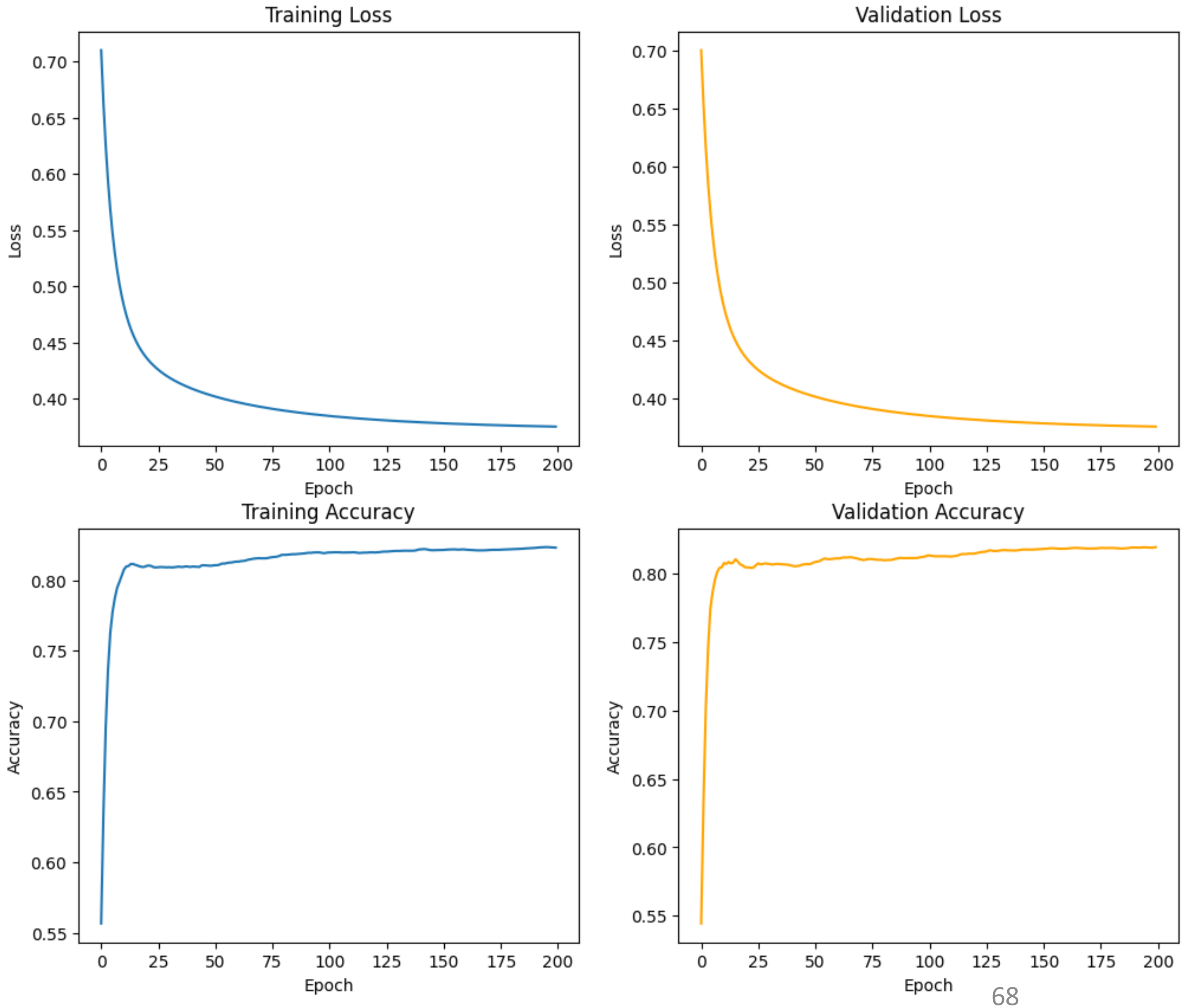
❖ Step 7: Training

```python
for epoch in range(epochs):
    train_batch_losses = []
    train_batch_accs = []
    val_batch_losses = []
    val_batch_accs = []

    for i in range(0, X_train.shape[0], batch_size):
        X_i = X_train[i:i+batch_size]
        y_i = y_train[i:i+batch_size]

        y_hat = predict(X_i, theta)
        train_loss = compute_loss(y_hat, y_i)
        gradient = compute_gradient(X_i, y_i, y_hat)
        theta = update_theta(theta, gradient, lr)
        train_batch_losses.append(train_loss)

        train_acc = compute_accuracy(X_train, y_train, theta)
        train_batch_accs.append(train_acc)

        y_val_hat = predict(X_val, theta)
        val_loss = compute_loss(y_val_hat, y_val)
        val_batch_losses.append(val_loss)

        val_acc = compute_accuracy(X_val, y_val, theta)
        val_batch_accs.append(val_acc)

    train_batch_loss = sum(train_batch_losses) / len(train_batch_losses)
    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
```

Sentiment Analysis

❖ Step 7: Training (Visualization)

```python
fig, ax = plt.subplots(2, 2, figsize=(12, 10))
ax[0, 0].plot(train_losses)
ax[0, 0].set(xlabel='Epoch', ylabel='Loss')
ax[0, 0].set_title('Training Loss')

ax[0, 1].plot(val_losses, 'orange')
ax[0, 1].set(xlabel='Epoch', ylabel='Loss')
ax[0, 1].set_title('Validation Loss')

ax[1, 0].plot(train_accs)
ax[1, 0].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 0].set_title('Training Accuracy')

ax[1, 1].plot(val_accs, 'orange')
ax[1, 1].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 1].set_title('Validation Accuracy')

plt.show()
```



Sentiment Analysis

❖ Step 8: Evaluation

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

```python
def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta).round()
    acc = (y_hat == y).mean()

    return acc
```

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

```
Evaluation on validation set:
Accuracy: 0.821969696969697
```

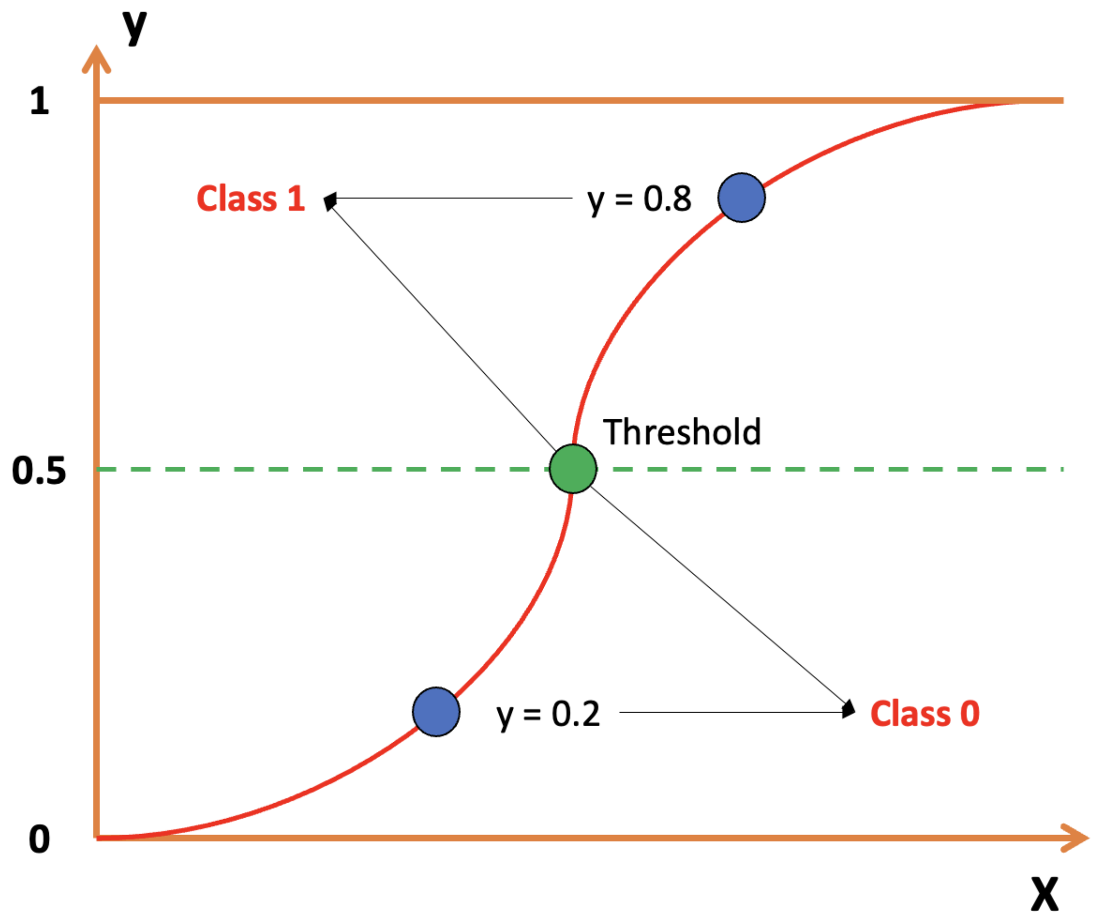
```python
# Test set
test_set_acc = compute_accuracy(X_test, y_test, theta)
print('Evaluation on test set:')
print(f'Accuracy: {test_set_acc}')
```

```
Evaluation on test set:
Accuracy: 0.8434343434343434
```

Summarization and QA

Summarization

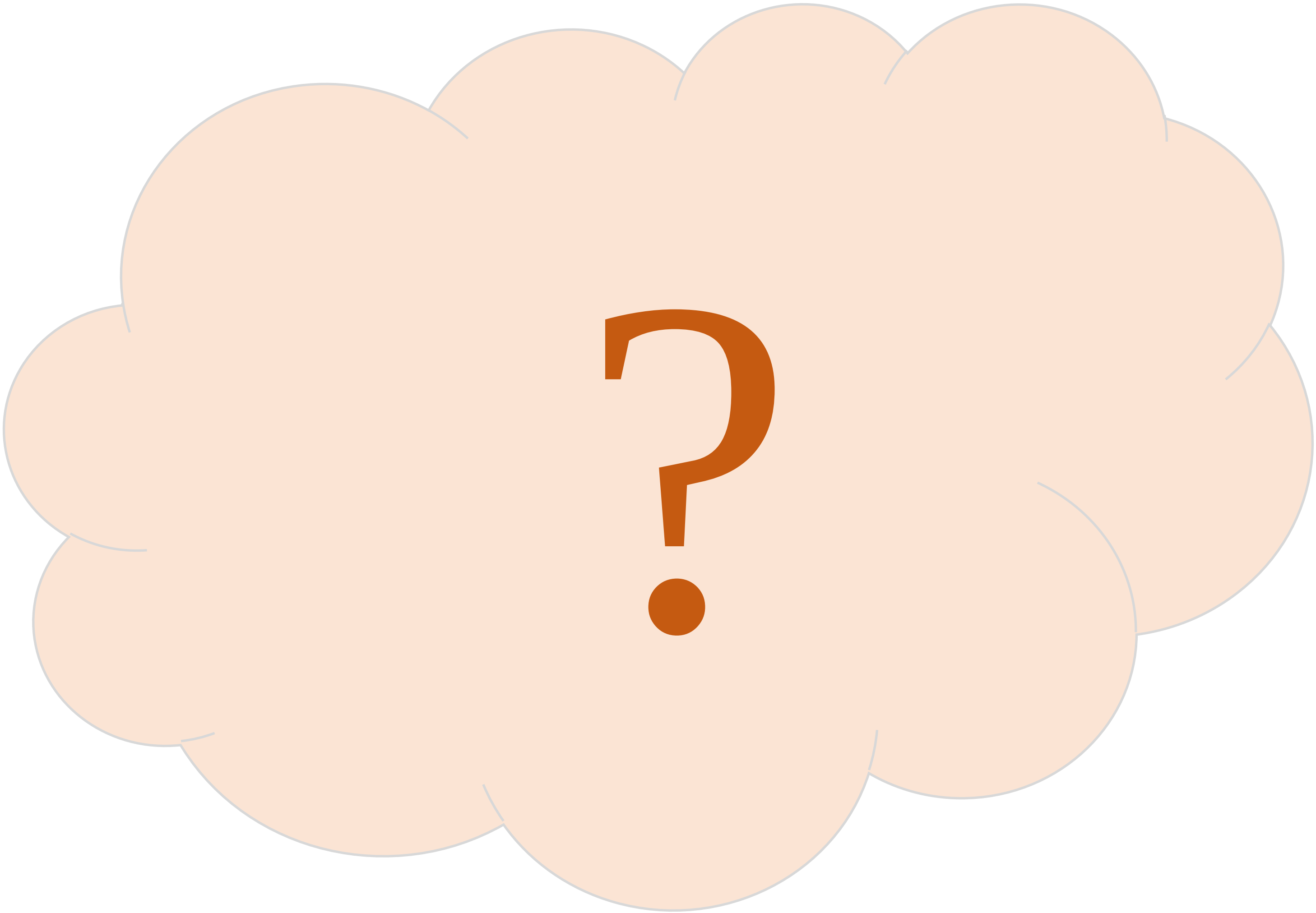
❖ Content



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




Thank you!