INF721

2024/2

UFV

Deep Learning

L4: Logistic Regression

Logistics

Announcements

- PA1: Logistic Regression will be out by the end of today.

Last Lecture

- Univariate Linear regression
  - Hypothesis space
  - MSE loss function
- Gradient Descent

Lecture Outline

- Linear regression with multiple features
- Vectorization
- Logistic Regression
  - Hypothesis space
  - Binary Cross-Entropy (BCE) Loss Function
  - Gradients

Univariate Linear Regression

Dataset D

x (size m)	y (Price in 1000's USD)
55	144
61	200
84	293
95	196
...	...

- Univariate Linear Regression

$$h(x) = wx + b$$

UFV

Multiple Linear Regression

Dataset D

x_1 (size m)	x_2 (# of beds)	x_3 (age in years)	y (price in 1000's USD)
152	4	24	1550
229	3	35	2286
84	1	10	293
95	3	14	196
...	...	...	...

- Univariate Linear Regression

$$h_{w,b}(x) = wx + b$$

- Generaly (for d input features)

$$h_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \ldots + \mathbf{w}_d\mathbf{x}_d + b$$

- Example:

$$h_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \mathbf{w}_3\mathbf{x}_3 + b$$

$$h_{\mathbf{w},b}(\mathbf{x}) = 0.1\mathbf{x}_1 + 4\mathbf{x}_2 + -2\mathbf{x}_3 + 80$$

size   # of beds   years   base price

UFV

Dot Product Notation

- Multiple Linear Regression

$$h(\mathbf{x}) = w_1x_1 + w_2x_2 + \ldots + w_dx_d + b \xrightarrow{\text{Dot product}} h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

- $\mathbf{w} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_d]$ is a weight vector
- $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d]$ is an input vector
- b is a scalar (called bias)

- Dot product

$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \ldots \mathbf{w}_d\mathbf{x}_d$$

UFV

Gradient Descent for Multiple Linear Regression

```python
def optimize(X, y, lr, n_iter):
  # Init weights to zero
  w, b = np.zeros(len(X[0])), 0

  # Optimize weihts iteratively
  for t in range(n_iter):
    # Predict x labels with w and b
    y_hat = np.dot(X, w) + b

    # Compute gradients
    dw = np.dot(X.T, (y_hat - y)) / len(y)
    db = np.mean(y_hat - y)

    # Update weights
    w = w - lr * dw
    b = b - lr * db

  return w, b
```

Multiple Linear Regression

$$h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

Loss Function

$$L(h_{\mathbf{w},b}) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\mathbf{w},b}(\mathbf{x}^{(\mathbf{i})}) - y^{(i)})^2$$

Gradient

$$\frac{\partial L}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} (h_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})\mathbf{x}_j^{(i)}$$

$$\frac{\partial L}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (h_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})$$

Vectorization

UFV

Vectorization

Vectorization in ML programming is the process of optimizing code to perform operations on entire vectors or matrices at once, rather than using explicit loops.

Benefits:

- Significantly faster execution (takes advantage of SIMD instructions)
- More concise and readable code
- Better utilization of modern CPU/GPU architectures
- Improved scalability for large datasets

Vectorizing multiple linear regression

Without vectorization 😞

```python
# Input features as a list
x = [152, 4, 24]

# Weights as a list
w = [0.1, 4.0, -2.0]

# Bias term as a float
b = 4

def model(x, w, b):
    y_hat = 0
    for i in range(len(x)):
        y_hat += w[i] * x[I]

    return y_hat + b
```

Vectorization 😃

```python
import numpy as np

# Input features as a vector
x = np.array([152, 4, 24])

# Weights as a vector
w = np.array([0.1, 4.0, -2.0])

# Bias term as a float
b = 4

def model(x, w, b):
     return np.dot(w, x) + b
```

UFV

Why vectorization speeds up ML code

Without vectorization 😞

```python
def model(x, w, b):
    d = len(x)
    y_hat = 0
    for i in range(d):
        y_hat += w[i] * x[i]

    return y_hat + b
```

t_0 `y_hat + w[0]*x[0]`

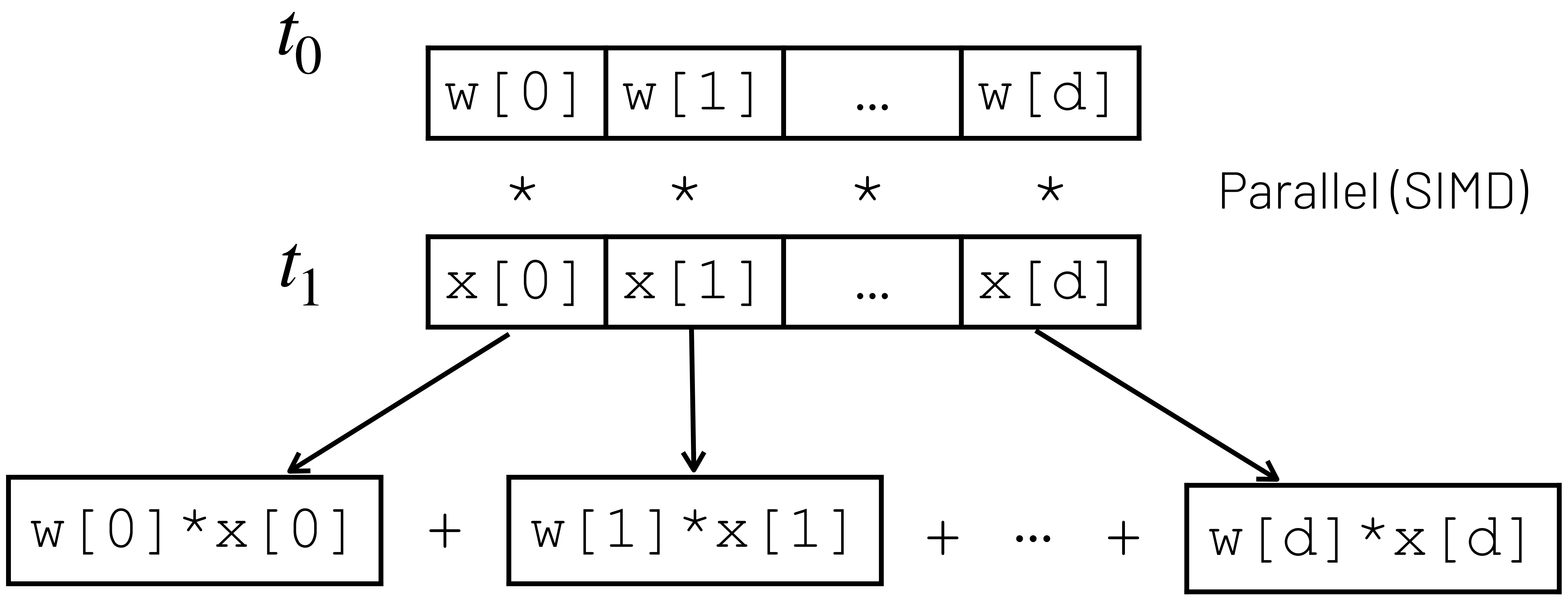
t_1 `y_hat + w[1]*x[1]`

$\dots$

t_d `y_hat + w[d]*x[d]`

Vectorization 😃

```python
def model(x, w, b):
    return np.dot(w, x) + b
```



Vectorizing loss function (MSE)

Without vectorization 😞

```python
# Labels as a list
y = [30, 70, 120]

# Predictions as a list
y_hat = [27, 92, 33]

def loss(y, y_hat):
    l = 0

    m = len(y)
    for i in range(m):
        l += (y_hat[i] - y[i]) ** 2

    return l / (2 * m)
```

Vectorization 😃

```python
import numpy as np

# Labels as a list
y = np.array([30, 70, 120])

# Predictions as a list
y_hat = np.array([27, 92, 33])

def loss(y, y_hat):
    return np.mean((y - y_hat) ** 2)
```

Vectorizing gradient descent

Without vectorization 😞

```
X = [[152, 4, 24], [229, 3, 35], [84,  1, 10]]
y = [1550, 2286, 293]

def optimize(X, y, n_iter, alpha):
  m = len(X), d = len(X[0])
  w = [0.0] * d
  b = 0.0

  for i in range(n_iter):
    # Compute predictions

    # Compute gradients
    dw = [0.0] * d
    db = 0.0

    for i in range(m):
      for j in range(d):
        dw[j] += (y_hat[i] - y[i]) * X[i][j]
      db += (y_hat[i] - y[i])

    # Update weights and bias
```

Vectorization 😃

```
import numpy as np

X = np.array([[152, 4, 24],
              [229, 3, 35],
              [84,  1, 10]]

y = np.array([1550, 2286, 293])

def optimize(X, y, n_iter, alpha):
    d = X.shape[1]
    w = np.zeros(d)
    b = 0.0

    # Compute predictions

    # Compute gradients
    dw = np.dot(X.T, (y_hat - y)) / len(y)
    db = np.mean(y_hat - y)

    # Update weights and bias
```

Numpy

NumPy (Numerical Python) is a library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

```
# Create arrays
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])


# Element-wise operations
z = x + y  # [3, 6, 9, 12, 15]
w = x * y  # [2, 8, 18, 32, 50]

# Matrix multiplication
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.dot(A, B)  # [[19, 22], [43, 50]]
```

```
# Statistical operations
mean = np.mean(x)  # 3.0
std = np.std(x)    # 1.41421356...

# Reshaping
D = np.arange(6) # [0, 1, 2, 3, 4, 5]
E = D.reshape(2, 3) # [[0, 1, 2], [3, 4, 5]]

# Broadcasting
F = np.array([[1, 2, 3], [4, 5, 6]])
G = G + 10  # [[11, 12, 13], [14, 15, 16]]
```

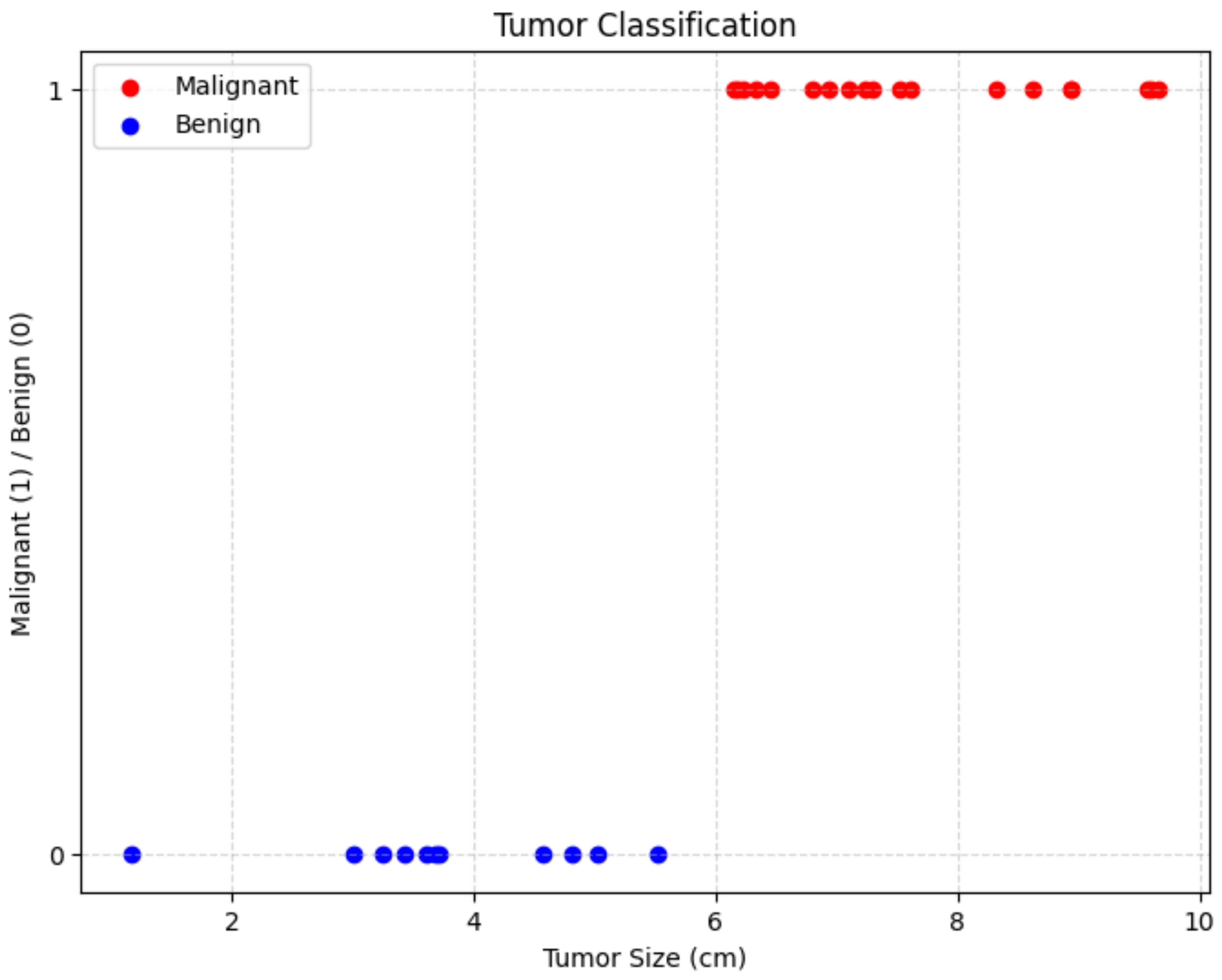
Logistic Regression

Problem 2: Tumor classification

Consider the problem of predicting whether a tumor is malignant or not based on its size:

Dataset D

x (size cm)	**y (malignant)**
9.63	1
4.32	0
5.42	0
9.52	1
...	...



UFV

Why not using linear regression?

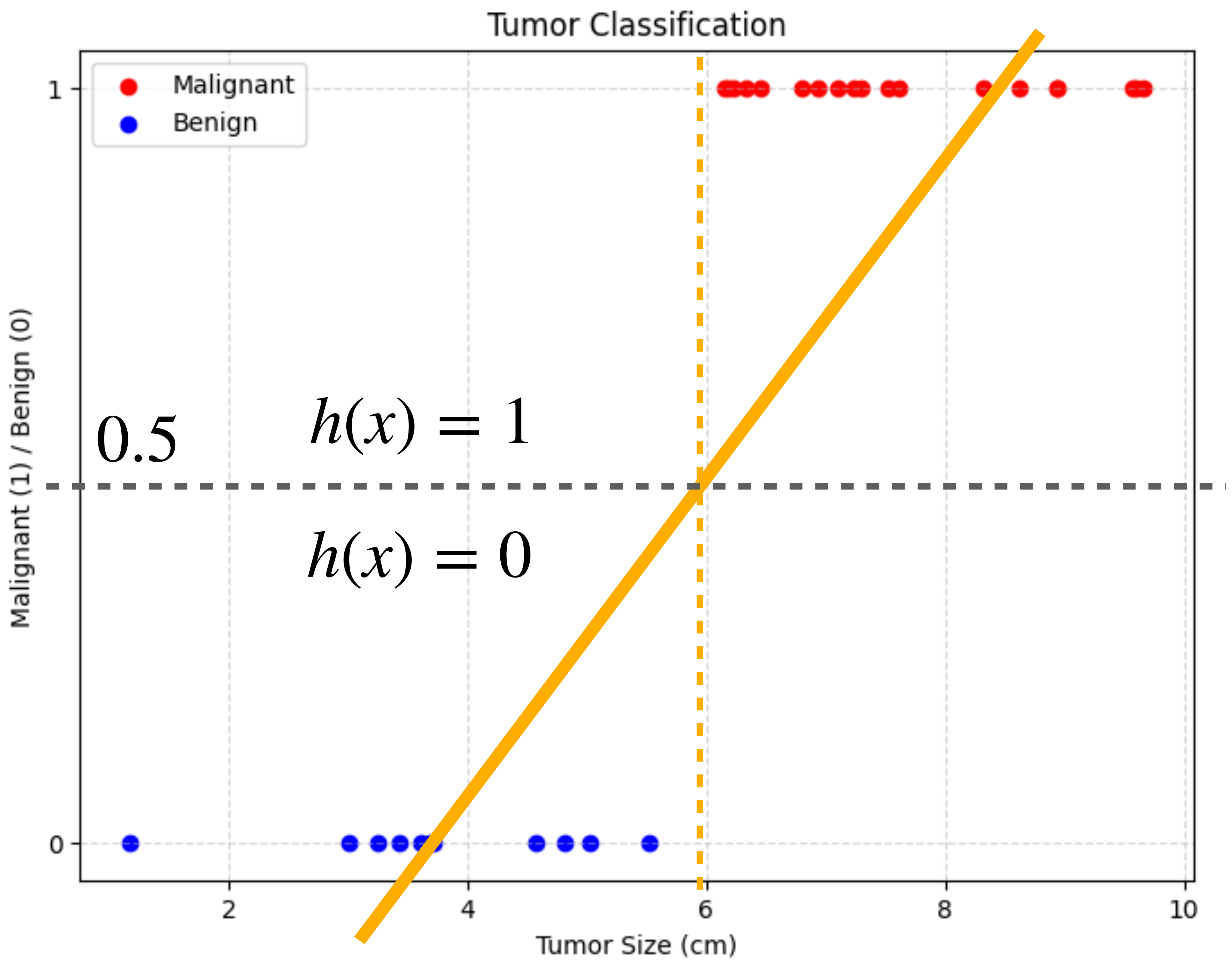
What happens if we try to use linear regression to solve this problem?

$h(x) = wx + b$

- Unbounded output $h(x) \in R$: produces outputs outside $[0,1]$ interval
- Idea – define a prediction **threshold**:

$$\hat{y} = \begin{cases} 0, \text{ if } h(x) < 0.5 \\ 1, \text{ if } h(x) \geq 0.5 \end{cases}$$

- Sensitive to outliers: extreme values can significantly skew the decision boundary



Why not using linear regression?

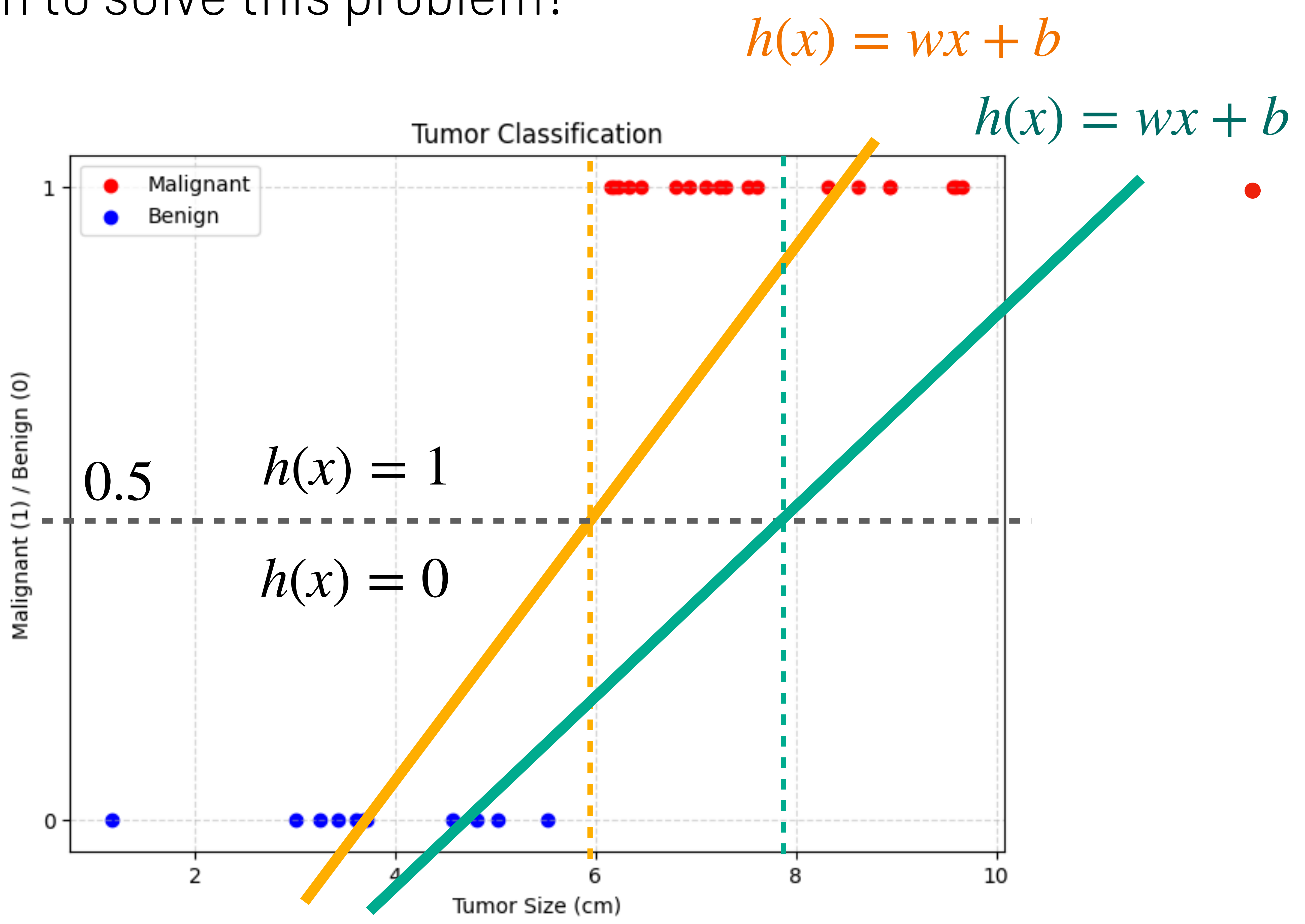
What happens if we try to use linear regression to solve this problem?

- Unbounded output $h(x) \in R$: produces outputs outside $[0,1]$ interval

- Idea – define a prediction **threshold**:

$$\hat{y} = \begin{cases} 0, \text{ if } & h(x) < 0.5 \\ 1, \text{ if } & h(x) \geq 0.5 \end{cases}$$

- Sensitive to outliers: extreme values can significantly skew the decision boundary



UFV

Logistic Regression

In **Logistic Regression**, we want to find a logistic function $h_{\mathbf{w},b}(\mathbf{x})$ that best fits the dataset D

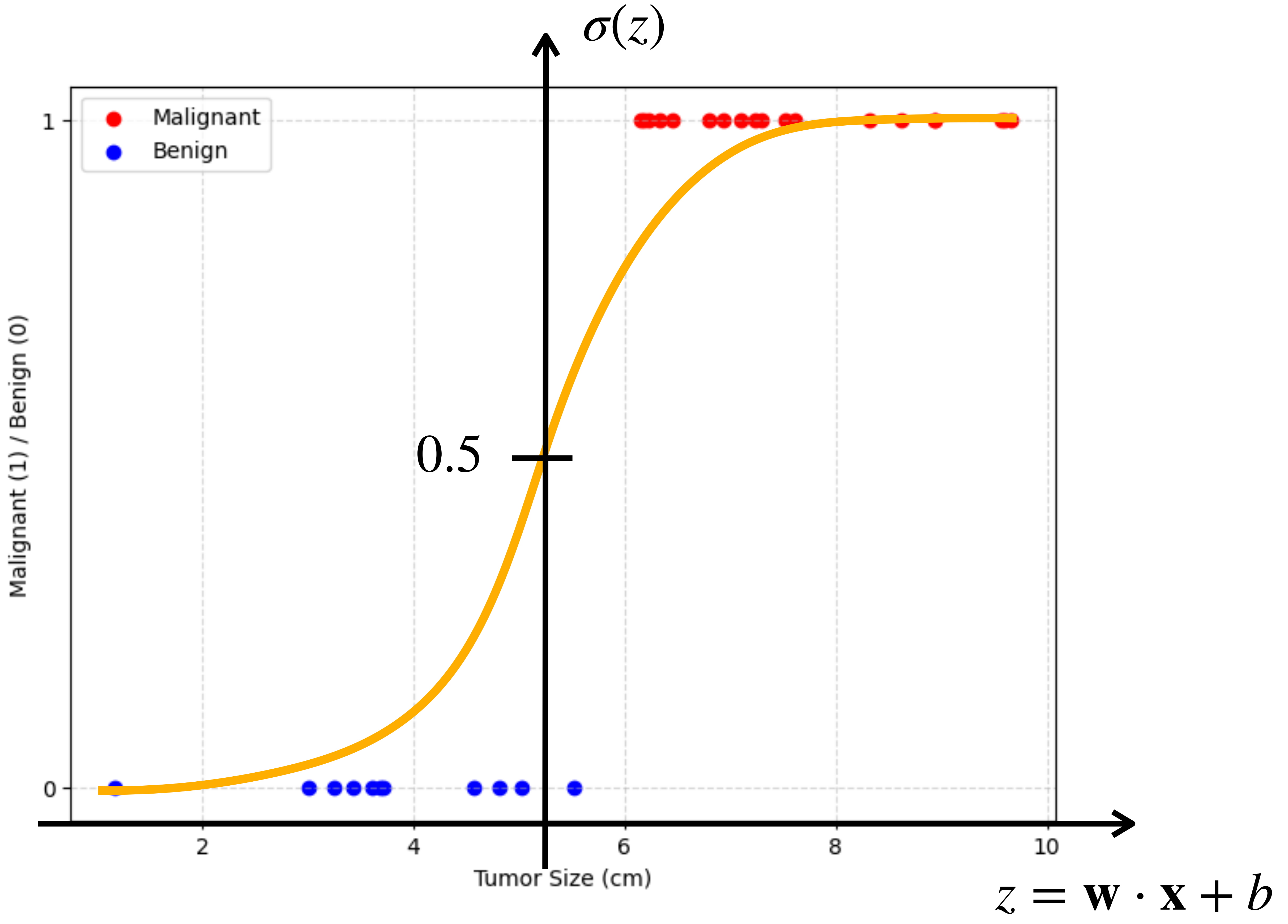
- Hypothesis space H:

  $h(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$, where

  $\sigma(z) = \dfrac{1}{1 + e^{-z}}$ (logistic/sigmoid)

- Bounded output $0 \leq h(\mathbf{x}) \leq 1$
- Still use threshold for prediction:

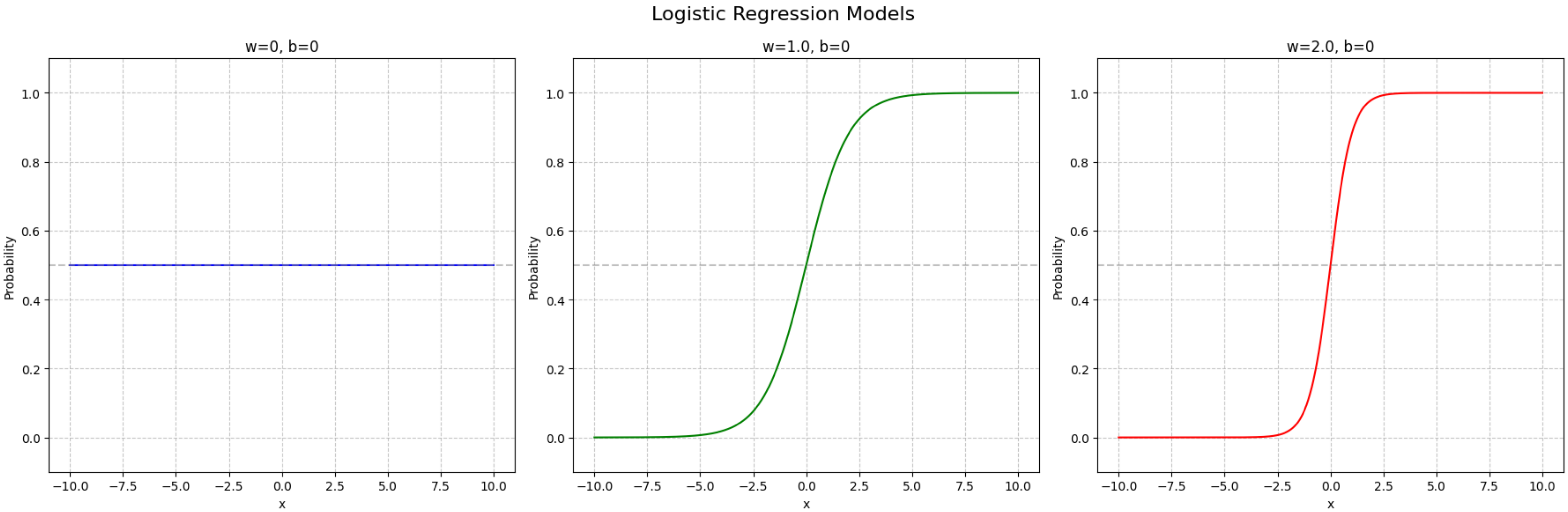
$$\hat{y} = \begin{cases} 0, \text{ if } h(x) < 0.5 \\ 1, \text{ if } h(x) \geq 0.5 \end{cases}$$



Hypothesis Space (w)

Hypothesis space

$$h(x) = \frac{1}{1 + e^{-(wx+b)}}$$



$$h(x) = \frac{1}{1 + e^{-(0)}}$$

$$h(x) = \frac{1}{1 + e^{-(x)}}$$

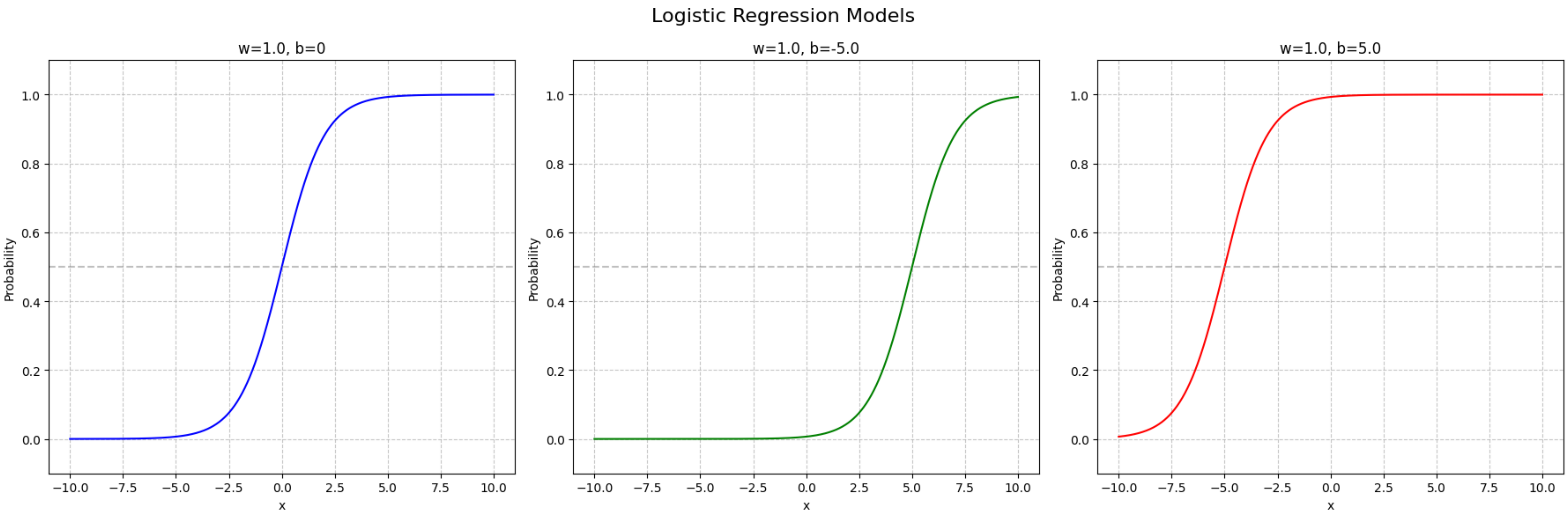
$$h(x) = \frac{1}{1 + e^{-(2x)}}$$

UFV

Hypothesis Space (b)

Hypothesis space

$$h(x) = \frac{1}{1 + e^{-(wx+b)}}$$



$$h(x) = \frac{1}{1 + e^{-(x)}}$$

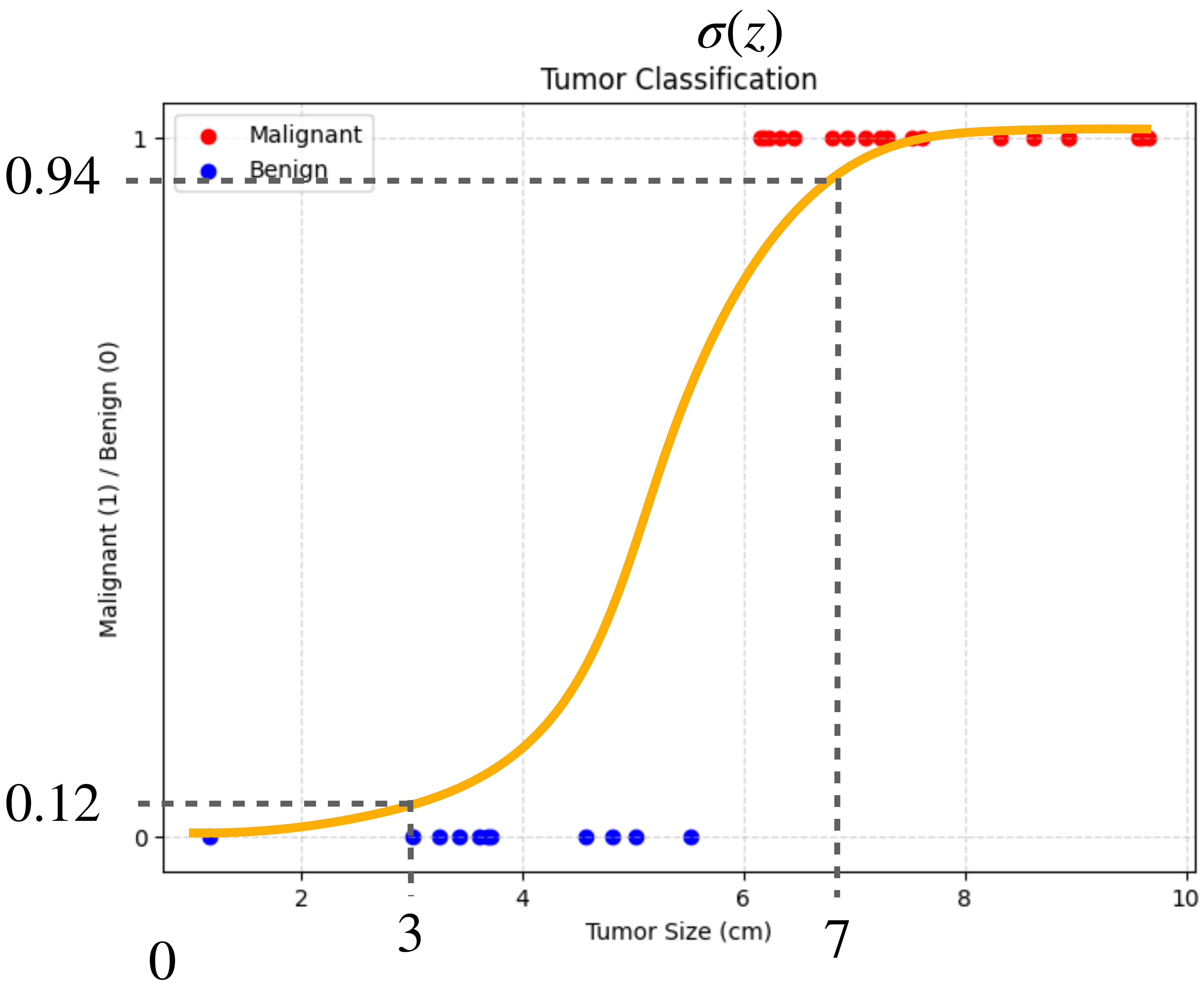
$$h(x) = \frac{1}{1 + e^{-(x-5)}}$$

$$h(x) = \frac{1}{1 + e^{-(x+5)}}$$

UFV

Probability interpretation

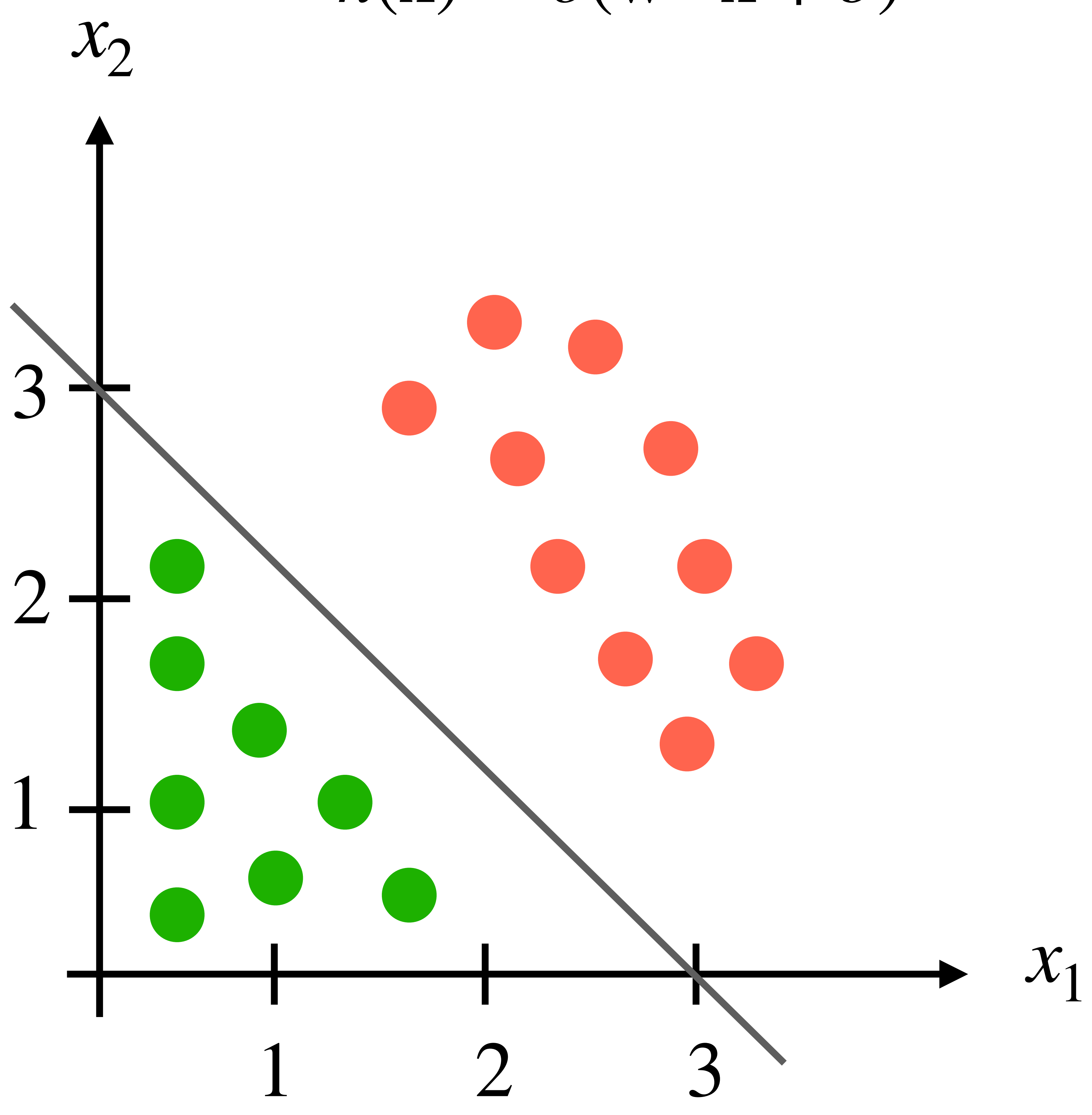
- Logistic Regression: $h(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$
- Since $0 \leq h(\mathbf{x}) \leq 1$, we can interpret $h(\mathbf{x})$ as $h(\mathbf{x}) = P(y = 1 \mid \mathbf{x})$, the probability that the label of the feature vector $\mathbf{x}$ is 1
- For example:
  - $h(3) = P(y = 1 \mid x = 3) = 0.12$
    12% of being malignant
  - $h(7) = P(y = 1 \mid x = 7) = 0.94$
    94% of being malignant
- If we want to know the probability of benign:
  - $P(y = 0 \mid \mathbf{x}) = 1 - P(y = 1 \mid \mathbf{x}) = 1 - h(\mathbf{x})$



Decision Boundary

Logistic Regression

$h(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$



To make a prediction $\hat{y} = h(x)$, we use a threshold:

$$\hat{y} = \begin{cases} 0, \text{ if } h(x) < 0.5 \\ 1, \text{ if } h(x) \geq 0.5 \end{cases}$$

Consider the following trained hypothesis:

$h(\mathbf{x}) = \sigma(\mathbf{x}_1 + \mathbf{x}_2 - 3) \quad w = [1,1], b = -3$

$$\hat{y} = \begin{cases} 0, \text{ if } x_1 + x_2 - 3 < 0 \\ 1, \text{ if } x_1 + x_2 - 3 \geq 0 \end{cases}$$

The line $x_1 + x_2 = 3$ is called the **decision boundary** of the the logistic regression.

UFV

Loss Function

Given a dataset $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$, want to measure how far the predictions $h(\mathbf{x}^{(\mathbf{i})})$ are from labels $y^{(i)}$ of examples $(\mathbf{x}^{(i)}, y^{(i)}) \in D$

We could try to use the MSE loss as in linear regression:

$$L(h) = \frac{1}{m} \sum_{i=1}^{m} (h(\mathbf{x}^{(i)}) - y^{(i)})^2$$

However, for logistic regression this loss is **not convex**!

Binary Cross-Entropy Loss Function

Logistic Regression $h(\mathbf{x})$ gives the probability of a feature vector $\mathbf{x}$ having label $y = 1$:

$$P(y = 1 \,|\, \mathbf{x}) = h(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}+b}}$$

Given a dataset $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$ maximize $P(y^{(i)} \,|\, \mathbf{x}^{(\mathbf{i})})$ for each $(\mathbf{x}^{(i)}, y^{(i)}), \in D$:

1. Probabilities for a given feature vector $\mathbf{x}^{\mathbf{i}}$:

$$P(y^{(i)} = 1 \,|\, \mathbf{x}^{(\mathbf{i})}) = h(\mathbf{x}^{(\mathbf{i})})$$

$$P(y^{(i)} = 0 \,|\, \mathbf{x}^{(\mathbf{i})}) = 1 - h(\mathbf{x}^{(\mathbf{i})})$$

2. Grouping this two probabilities in one expression:

$$P(y^{(i)} \,|\, \mathbf{x}^{(\mathbf{i})}) = h(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - h(\mathbf{x}^{(i)}))^{(1-y^{(i)})}$$

3. Since we want to maximize $P(y^{(i)} \,|\, \mathbf{x}^{(\mathbf{i})})$ for each $(\mathbf{x}^{(i)}, y^{(i)}), \in D$:

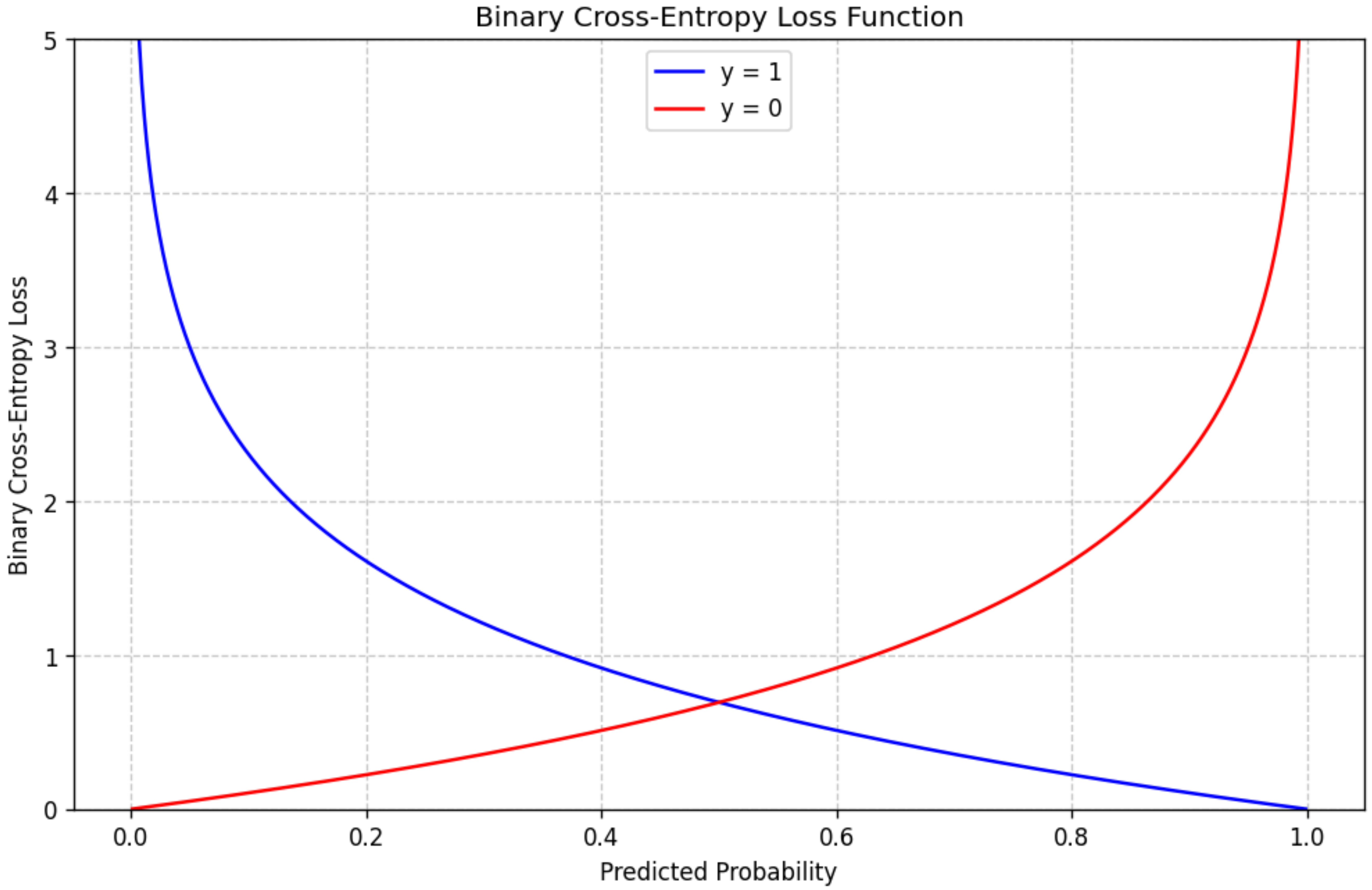
$$L(h) = \prod_{i=1}^{m} h(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - h(\mathbf{x}^{(i)}))^{(1-y^{(i)})}$$

4. Applying log and negating to transform into error:

$$L(h) = -\underbrace{\frac{1}{m}\sum_{i}^{m} y^{(i)}\, log(h(\mathbf{x}^{(\mathbf{i})})) + (1 - y^{(i)})\, log(1 - h(\mathbf{x}^{(\mathbf{i})}))}_{\textbf{Binary Cross-Entropy (BCE)}}$$

Binary Cross-Entropy Loss Function

$$L(h) = -\frac{1}{m}\sum_{i}^{m} y^{(i)}\, log(h(\mathbf{x^{(i)}})) + (1 - y^{(i)})\, log(1 - h(\mathbf{x^{(i)}}))$$



UFV

Calculating the gradients for logistic regression

Logistic Regression

$\hat{y} = \sigma(wx + b) = \frac{1}{1+e^{-(wx+b)}}$

Binary Cross-Entropy for a single sample

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

Partial derivative of L with respect to w

$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$

$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$

$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \cdot \hat{y}(1-\hat{y}) = \hat{y} - y$

$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial w} = (\hat{y} - y) \cdot x$

Partial derivative of L with respect to b

$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial b} = \hat{y} - y$

Gradient Descent for Logistic Regression

```
def optimize(x, y, lr, n_iter):
  # Init weights to zero
  w, b = 0, 0

  # Optimize weihts iteratively
  for t in range(n_iter):
    # Predict x labels with w and b
    y_hat = sigmoid(np.dot(w,x) + b)

    # Compute gradients
    dw = (1 / m) * np.sum((y_hat - y) * x)
    db = (1 / m) * np.sum(y_hat - y)

    # Update weights
    w = w - lr * dw
    b = b - lr * db

  return w, b
```

Logistic Regression

$$z = w \cdot x + b$$

$$\hat{y} = h(x) = \frac{1}{1 + e^{-z}}$$

BCE Loss Function

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

Gradient

$$\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{n}(\hat{y^{(i)}} - y^{(i)})x^{(i)}$$

$$\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})$$

UFV

Next Lecture

L5: MLP

Multilayer Perceptron for non-linearly separable problems