

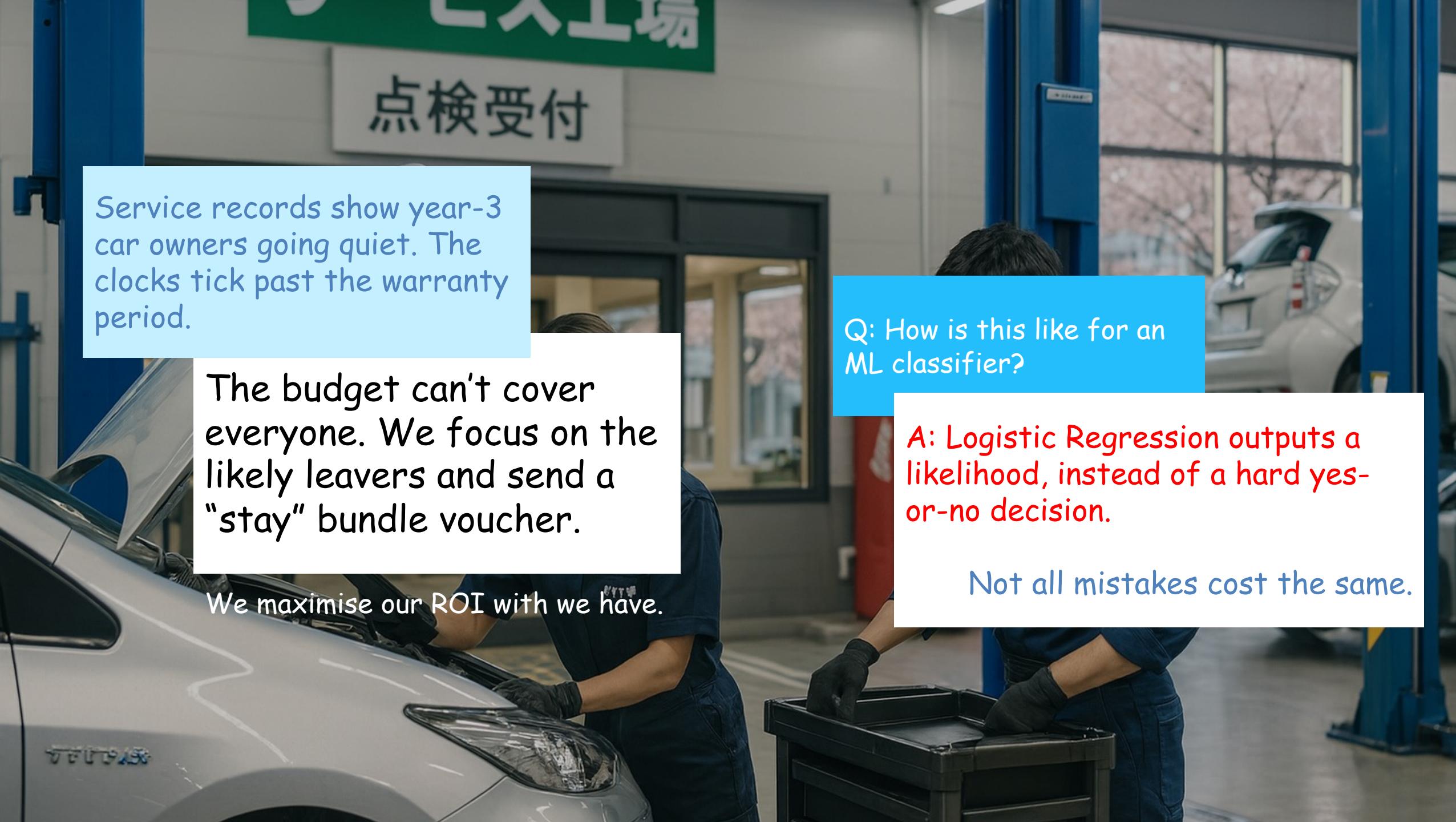
# Machine Learning

## Logistic Regression

Tarapong Sreenuch

8 February 2024

克明峻德，格物致知



Service records show year-3 car owners going quiet. The clocks tick past the warranty period.

The budget can't cover everyone. We focus on the likely leavers and send a "stay" bundle voucher.

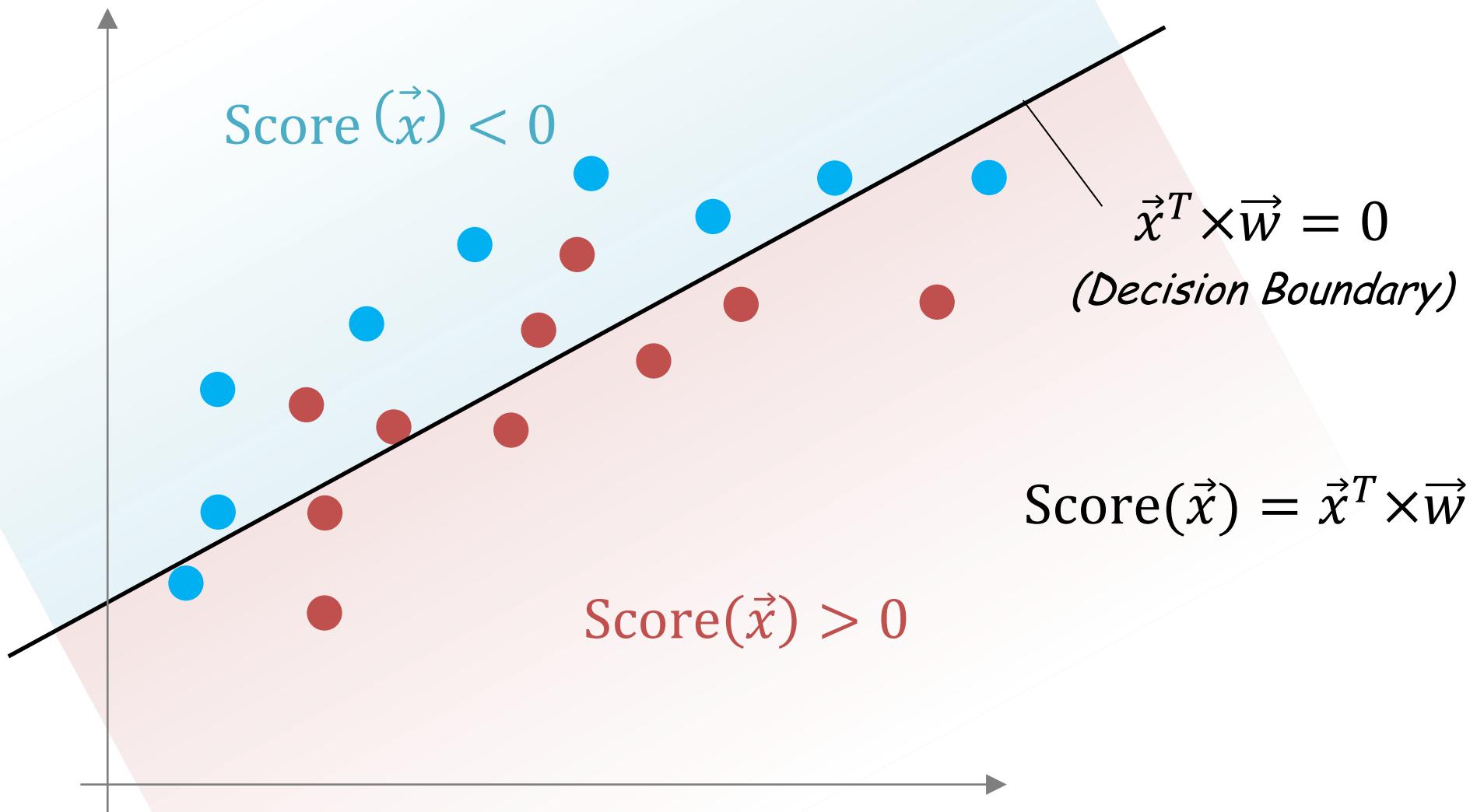
We maximise our ROI with what we have.

Q: How is this like for an ML classifier?

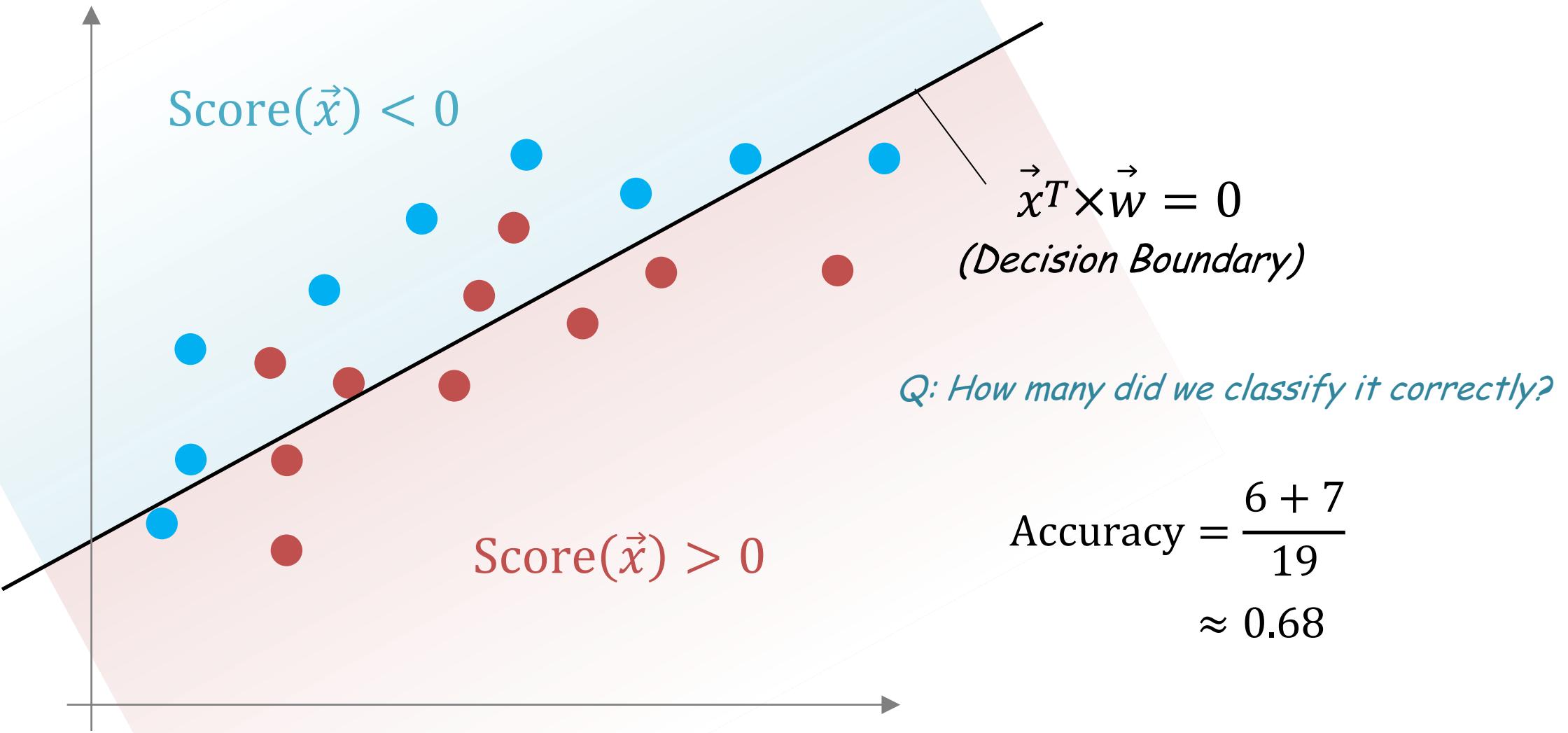
A: Logistic Regression outputs a likelihood, instead of a hard yes-or-no decision.

Not all mistakes cost the same.

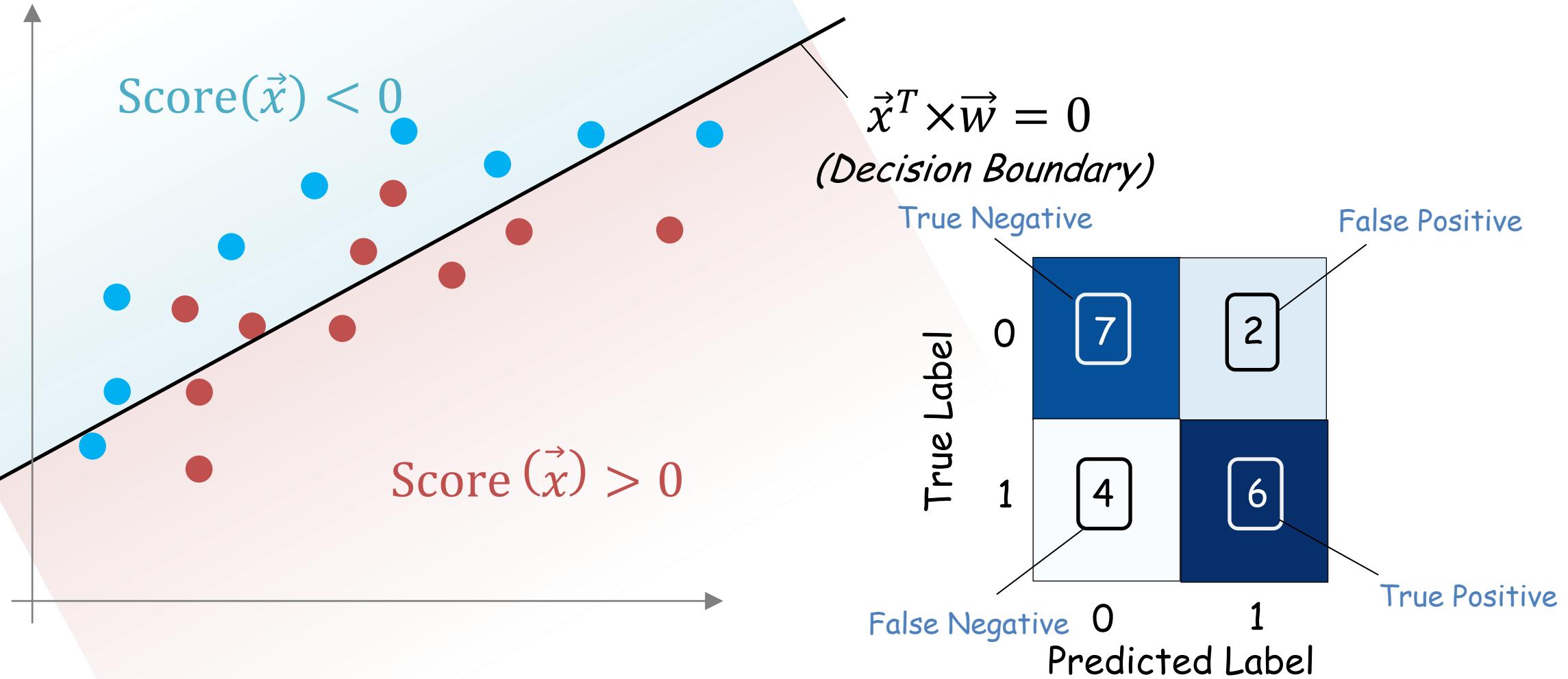
# Linear Classifier



## Recap: Accuracy



## Recap: Confusion Matrix



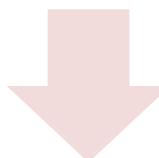
# Prediction Confidence

---

*The ramen & everything else were awesome!*



*Definitely Positive*



$$P(y = 1 \mid \vec{x} = \text{"The ramen & everything else were awesome!"}) = 0.99$$

*The ramen was good, the service was OK.*



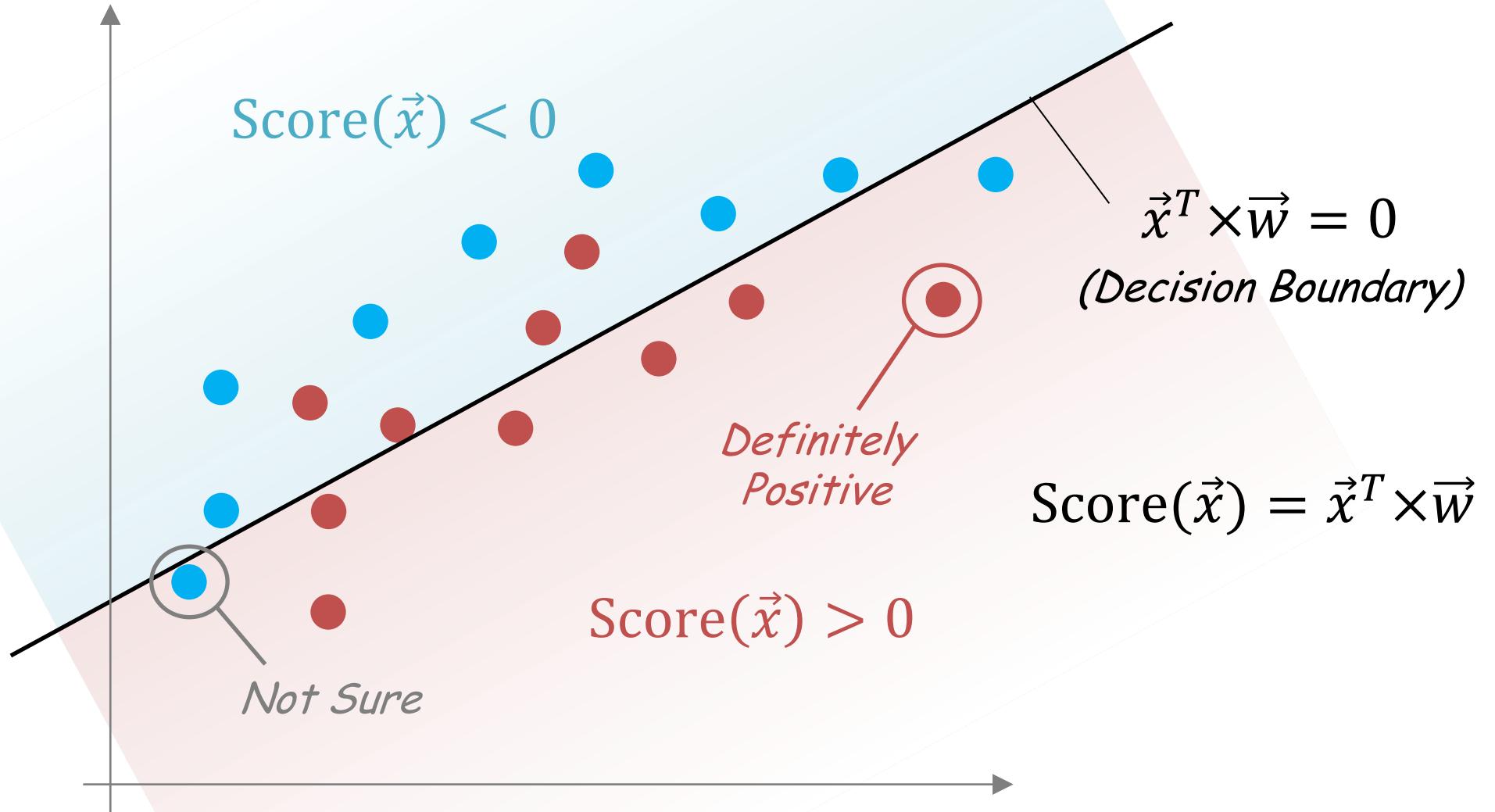
*Not Sure*



$$P(y = 1 \mid \vec{x} = \text{"The ramen was good, the service was OK."}) = 0.55$$

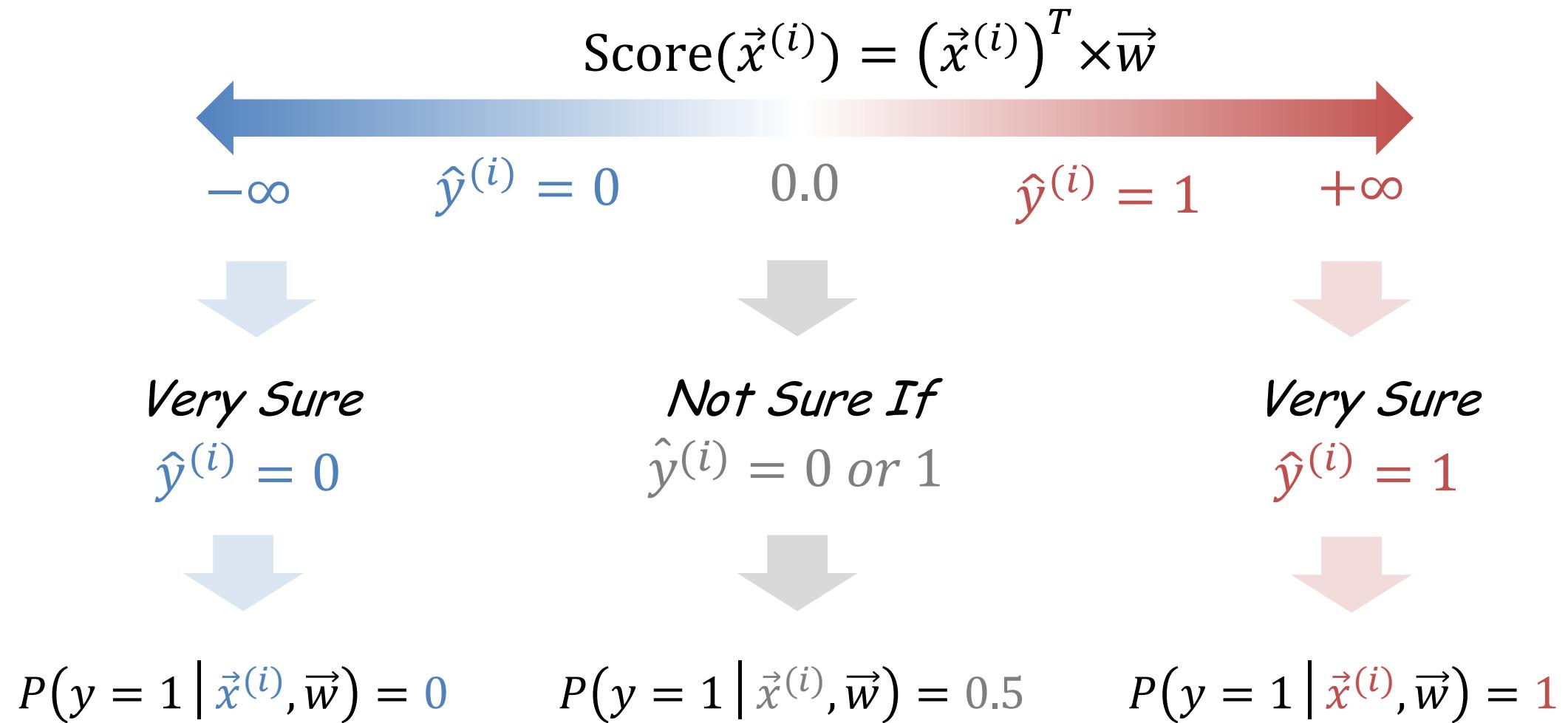
We read the conditional probability  $P(y|\vec{x})$  as “the probability of  $y$  given  $\vec{x}$ ”.

# How Confidence?



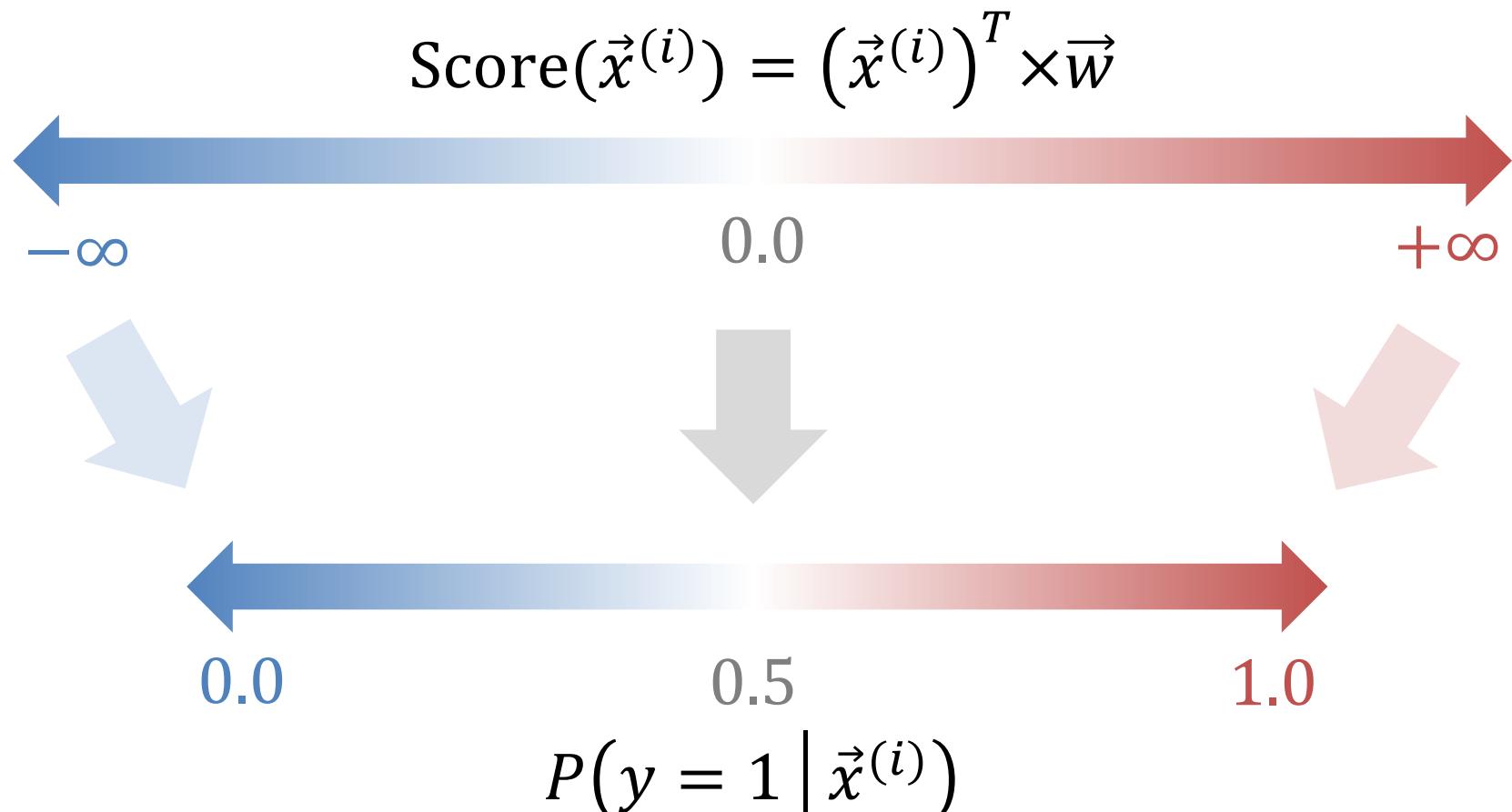
# Scoring Interpretation

---



# Scoring Interpretation

---



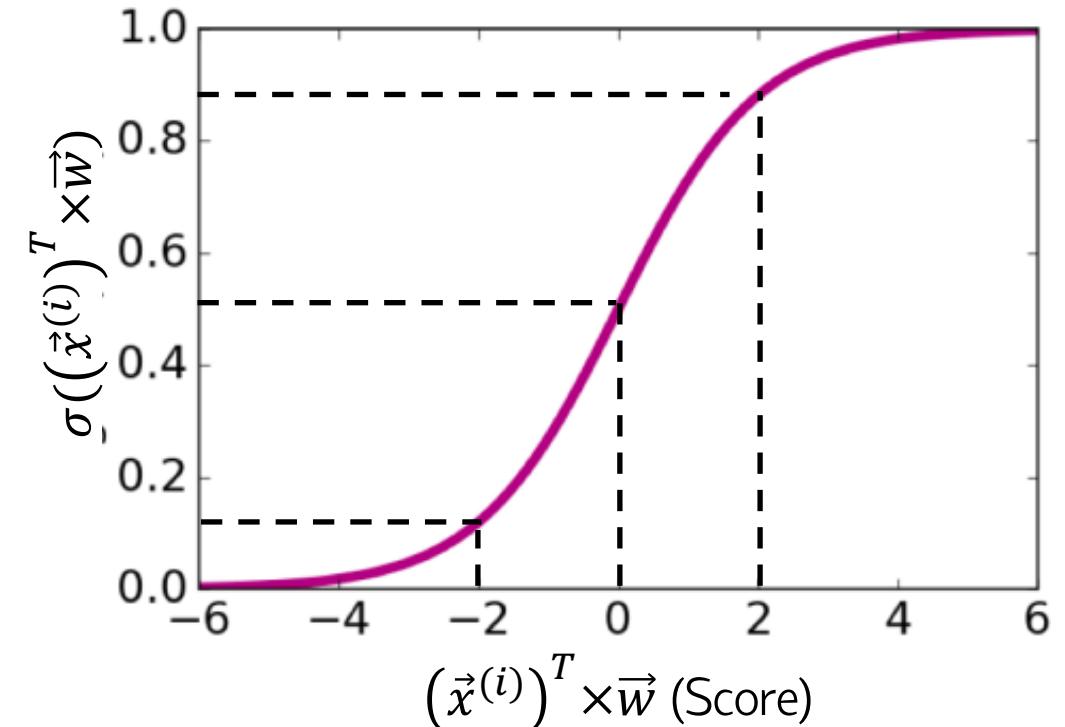
*Q: How do we map  $(-\infty, +\infty)$  to  $[0.0, 1.0]$ ?*

# Sigmoid (or Logistic) Function

Sigmoid Function:

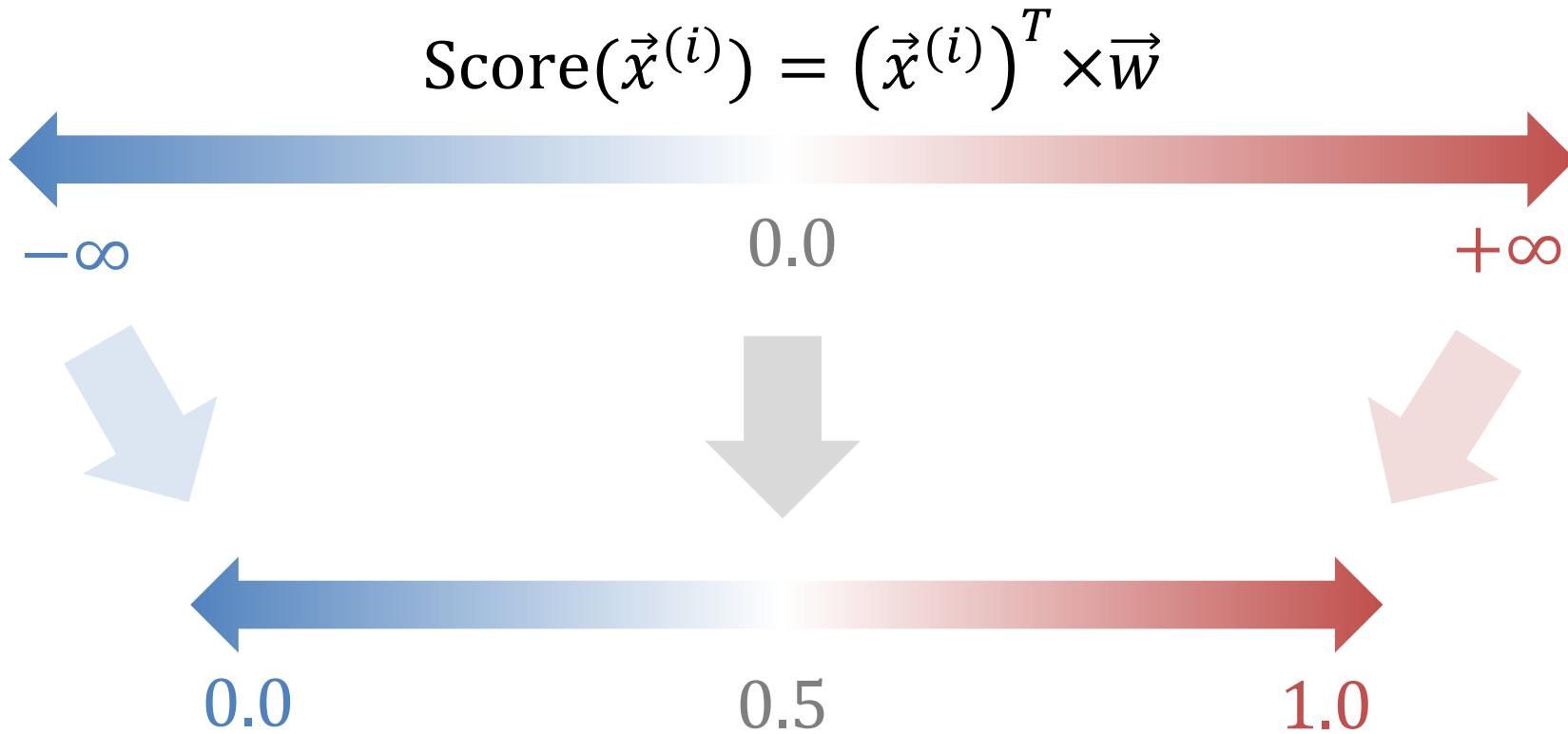
$$\sigma((\vec{x}^{(i)})^T \times \vec{w}) = \frac{1}{1 + e^{-(\vec{x}^{(i)})^T \times \vec{w}}}$$

$(\vec{x}^{(i)})^T \times \vec{w}$ (Score)	$\sigma((\vec{x}^{(i)})^T \times \vec{w})$
$-\infty$	0.0
-2.0	0.12
0.0	0.5
+2.0	0.88
$+\infty$	1.0



# Logistic Regression Model

---



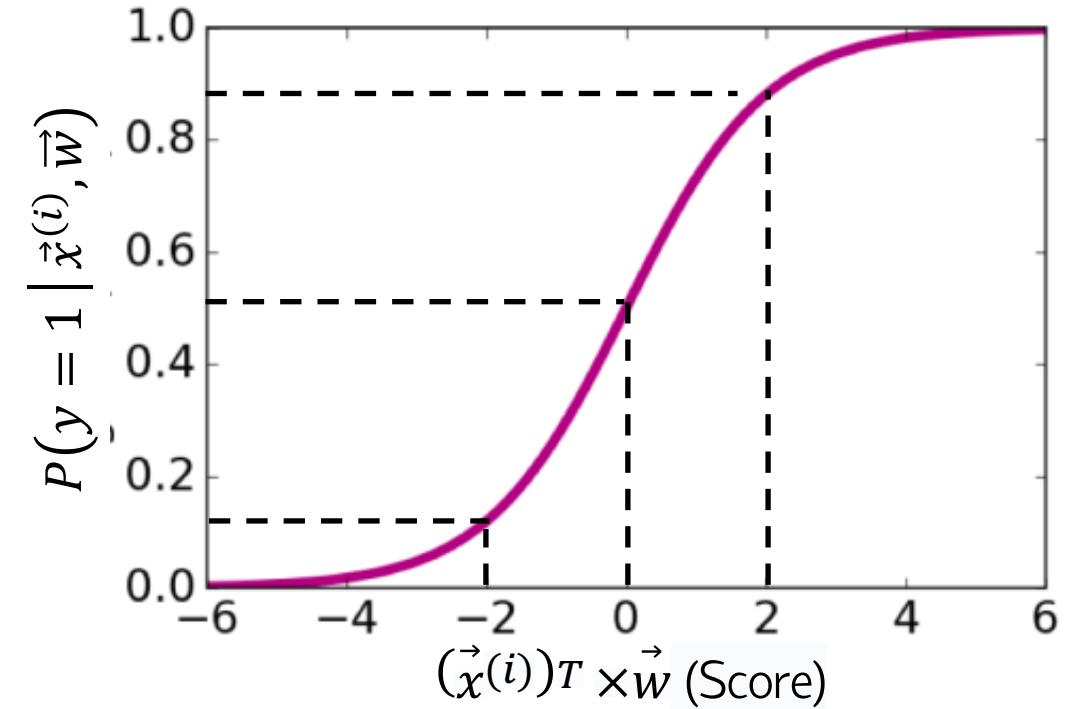
$$P(y = 1 \mid \vec{x}^{(i)}, \vec{w}) = \frac{1}{1 + e^{-(\vec{x}^{(i)})^T \times \vec{w}}}$$

# Understand Logistic Regression

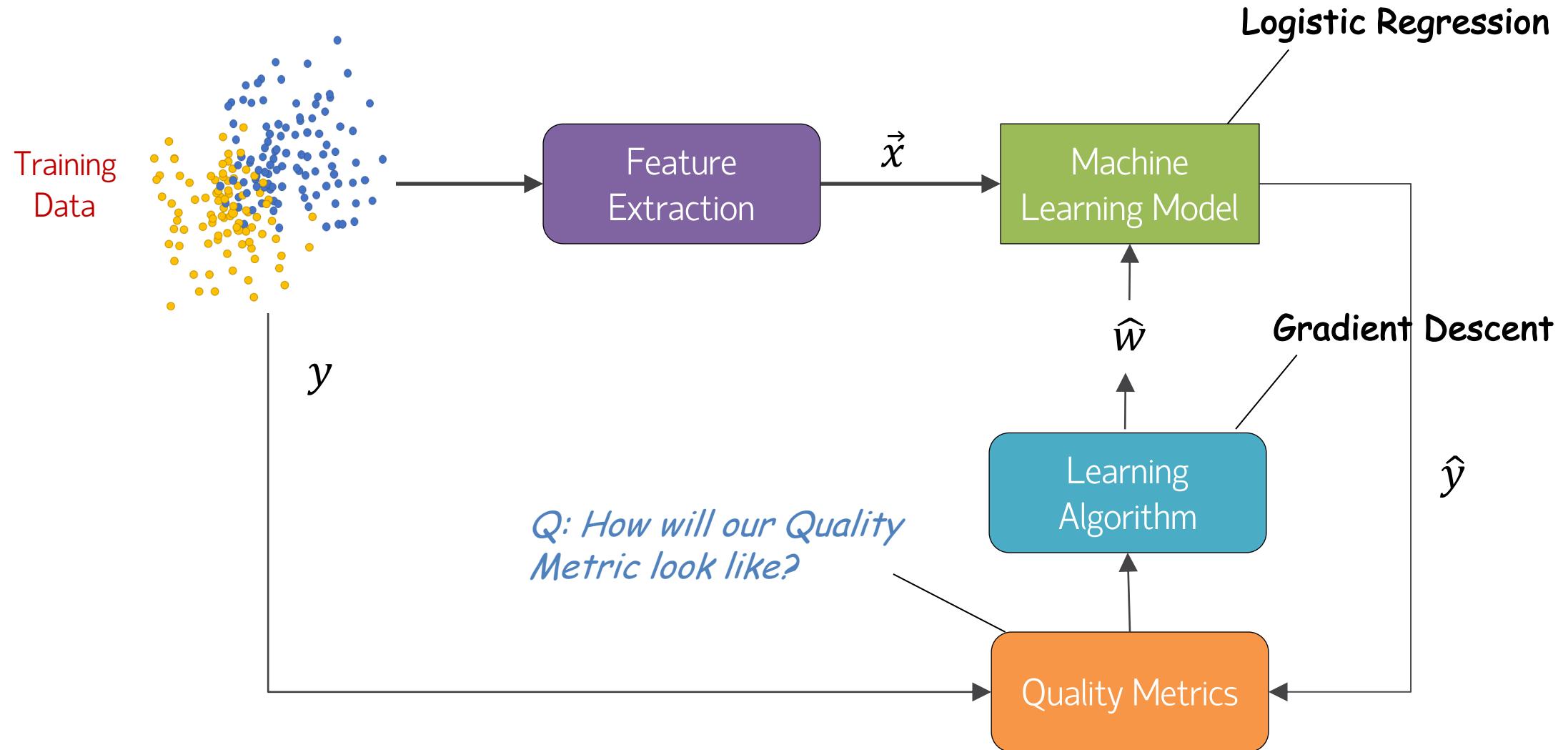
Probabilistic Score:

$$P(y = 1 \mid \vec{x}^{(i)}, \vec{w}) = \frac{1}{1 + e^{-(\vec{x}^{(i)})^T \times \vec{w}}}$$

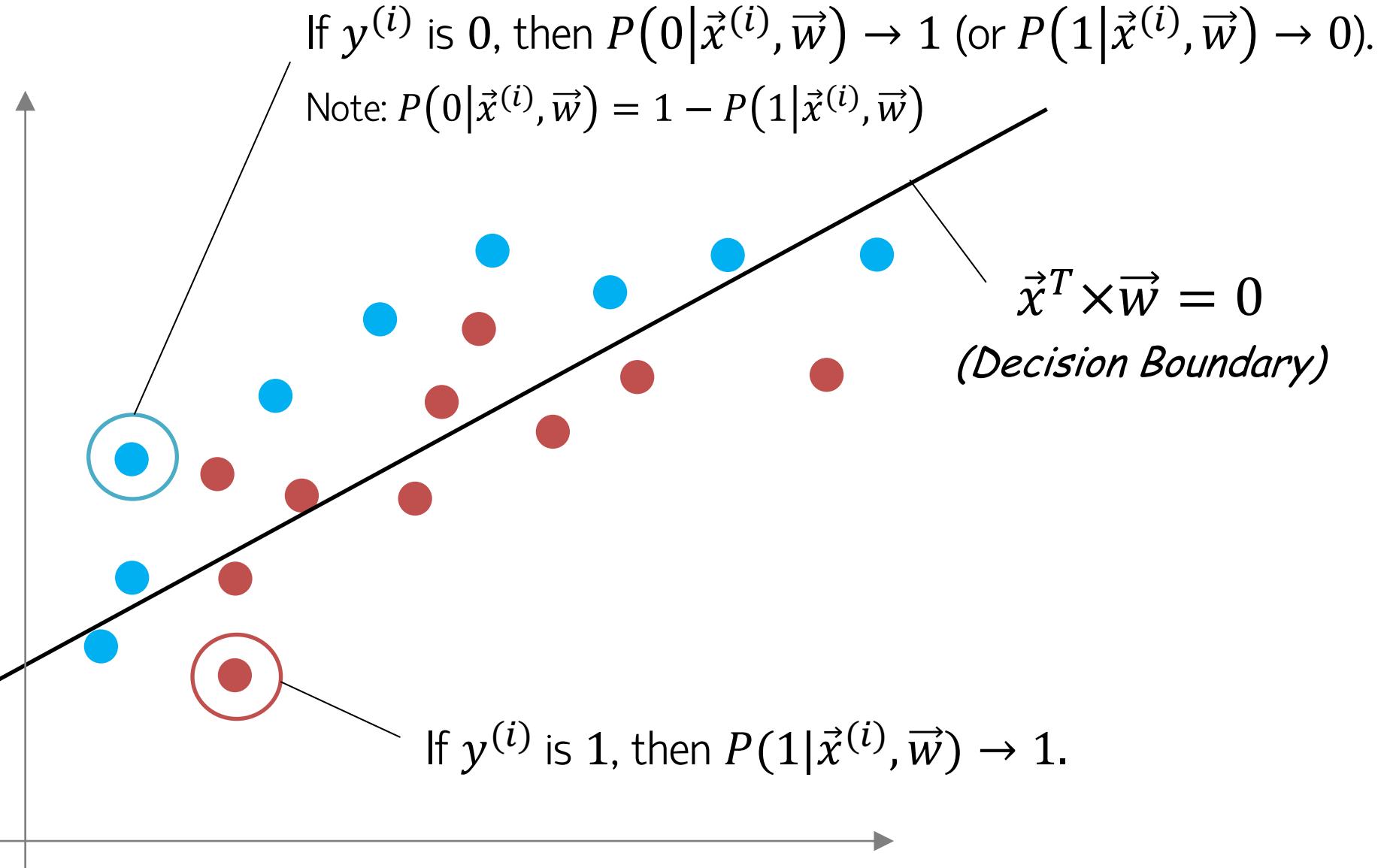
$(\vec{x}^{(i)})^T \times \vec{w}$ (Score)	$P(y = 1 \mid \vec{x}^{(i)}, \vec{w})$
$-\infty$	0.0
-2.0	0.12
0.0	0.5
+2.0	0.88
$+\infty$	1.0



# Workflow: Logistic Regression



# What We Are Optimising



# Recall: Joint Probability



Q: What is the probability (or likelihood) of rolling a six on both dice?

$$A: \frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$$

$$P(A, B) = P(A) \times P(B)$$

Probability of Rolling  
a Six on Dice A

Probability of Rolling  
a Six on Dice B

## Quality Metric: Likelihood Function

Likelihood Function:

$$L(\vec{w}) = \prod_{i=1}^N P(1|\vec{x}^{(i)}, \vec{w})^{y^{(i)}} \times [1 - P(1|\vec{x}^{(i)}, \vec{w})]^{(1-y^{(i)})}$$

$\leq 1 \text{ if } y^{(i)} = 0$        $\leq 1 \text{ if } y^{(i)} = 1$

If  $y_i$  is 1, then  $P(1|\vec{x}_i, \vec{w}) \rightarrow 1$ .

If  $y_i$  is 0, then  $P(1|\vec{x}^{(i)}, \vec{w}) \rightarrow 0$  (or  $P(0|\vec{x}^{(i)}, \vec{w}) \rightarrow 1$ ).

Note:  $P(0|\vec{x}^{(i)}, \vec{w}) = 1 - P(1|\vec{x}^{(i)}, \vec{w})$

# Log Likelihood Function

---

Likelihood Function:

$$L(\vec{w}) = \prod_{i=1}^N P(1|\vec{x}^{(i)}, \vec{w})^{y^{(i)}} \times [1 - P(1|\vec{x}^{(i)}, \vec{w})]^{(1-y^{(i)})}$$

Log Likelihood Function:

$$\log L(\vec{w}) = \sum_{i=1}^N [y^{(i)} \log P(1|\vec{x}^{(i)}, \vec{w}) + (1 - y^{(i)}) \log(1 - P(1|\vec{x}^{(i)}, \vec{w}))]$$

Relevant Logarithmic Rules:

$$\log(A \times B) = \log A + \log B \quad \log A^N = N \log A$$

## Numerical Underflow

---

$P(1|\vec{x}^{(i)}, \vec{w})$  and  $1 - P(1|\vec{x}^{(i)}, \vec{w})$  are less than 1. Hence,  $L(\vec{w}) \rightarrow 0$ .

*This could lead to numerical underflow. Try:  $0.1**1000$  (i.e.  $1.0 \times 10^{-1000}$ ) in Python.*

*Moreover, our gradients would be vanishing as  $L(\vec{w}_k) \approx 0$ . Consequently, if no gradient, then gradient descent will not work.*

*Q: How can we handle very very small numbers?*

*A: We take Log.  $1000 \times \text{Log}(0.1) = 1,000 \times (-1) = -1,000$ .*

# Likelihood Function as an On/Off Switch

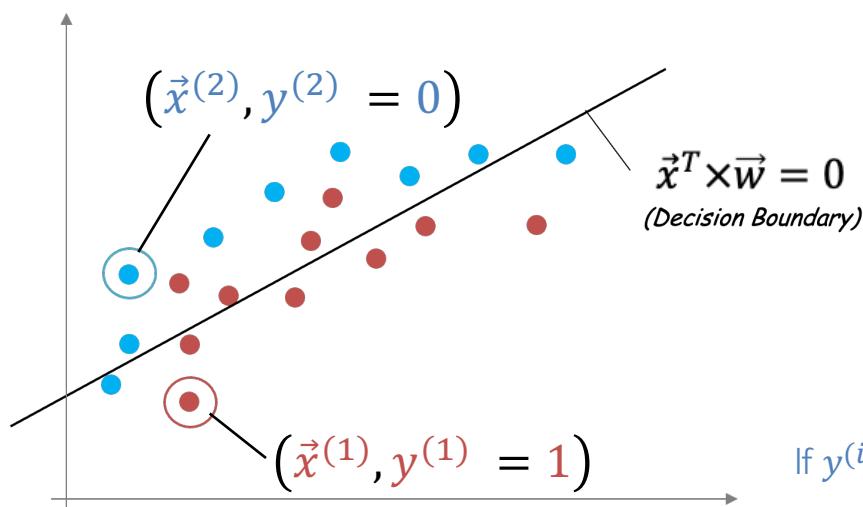
Likelihood Function:

$$L(\vec{w}, b) = - \sum_{i=1}^N [y^{(i)} \log P(1|\vec{x}^{(i)}, \vec{w}, b) + (1 - y^{(i)}) \log(1 - P(1|\vec{x}^{(i)}, \vec{w}, b))]$$

*0 (OFF) if  $y^{(i)} = 0$*   
||  
*0 (OFF) if  $y^{(i)} = 1$*

Related Properties:

$P(y = 1 \vec{x}, \vec{w}, b) \in (0,1)$	$P(y = 1 \vec{x}, \vec{w}, b) \rightarrow 1^-$
$\log P(y = 1 \vec{x}, \vec{w}, b) \in (-\infty, 0)$	$\log P(y = 1 \vec{x}, \vec{w}, b) \rightarrow 0^-$
$-\log P(y = 1 \vec{x}, \vec{w}, b) \in (0, +\infty)$	$-\log P(y = 1 \vec{x}, \vec{w}, b) \rightarrow 0^+$



If  $y^{(i)}$  is 1, then  $-\log P(1|\vec{x}^{(i)}, \vec{w}) \rightarrow 0^+$ .

$$\begin{aligned} L(\vec{x}^{(1)}, \vec{w}, b) &= -[y^{(1)} \log P(1|\vec{x}^{(1)}, \vec{w}, b) + (1 - y^{(1)}) \log(1 - P(1|\vec{x}^{(1)}, \vec{w}, b))] \\ &= -[1 \times \log P(1|\vec{x}^{(1)}, \vec{w}, b) + (1 - 1) \log(1 - P(1|\vec{x}^{(1)}, \vec{w}, b))] \\ &= -\log P(1|\vec{x}^{(1)}, \vec{w}, b) \end{aligned}$$

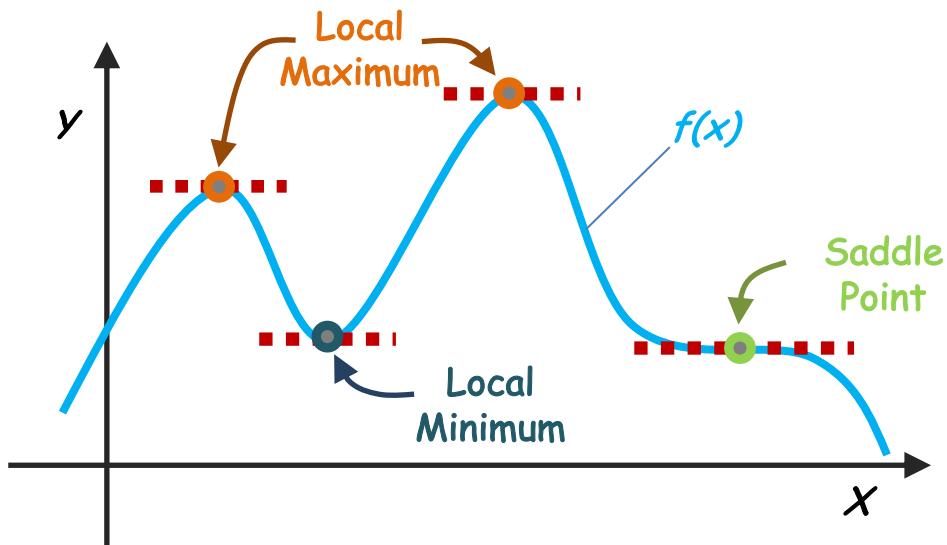
If  $y^{(i)}$  is 0, then  $-\log(1 - P(1|\vec{x}^{(2)}, \vec{w}, b)) \rightarrow 0^+$  (or  $-\log P(0|\vec{x}^{(i)}, \vec{w}, b) \rightarrow 0^+$ ).

$$\begin{aligned} L(\vec{x}^{(2)}, \vec{w}, b) &= -[y^{(2)} \log P(1|\vec{x}^{(2)}, \vec{w}, b) + (1 - y^{(2)}) \log(1 - P(1|\vec{x}^{(2)}, \vec{w}, b))] \\ &= -[0 \times \log P(1|\vec{x}^{(2)}, \vec{w}, b) + (1 - 0) \log(1 - P(1|\vec{x}^{(2)}, \vec{w}, b))] \\ &= -\log(1 - P(1|\vec{x}^{(2)}, \vec{w}, b)) \\ &= -\log P(0|\vec{x}^{(2)}, \vec{w}, b) \end{aligned}$$

# Best Decision Boundary

Maximising Log Likelihood:

$$\max_{\vec{w}} \sum_{i=1}^N [y^{(i)} \log P(1|\vec{x}^{(i)}, \vec{w}) + (1 - y^{(i)}) \log(1 - P(1|\vec{x}^{(i)}, \vec{w}))]$$



*Q: Which value of  $x$  will  $f(x)$  be maximum?*

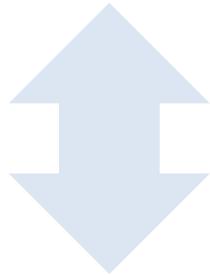
*A: ... Slope (or Gradient) = 0 ...*



# Gradient Descent: Minimisation

Maximising Log Likelihood:

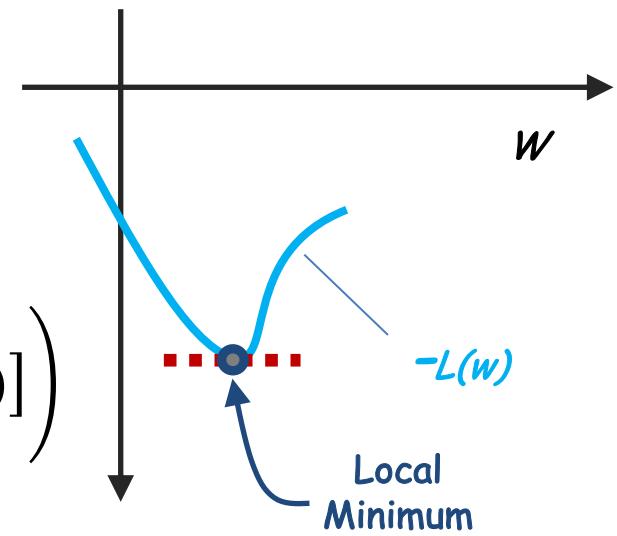
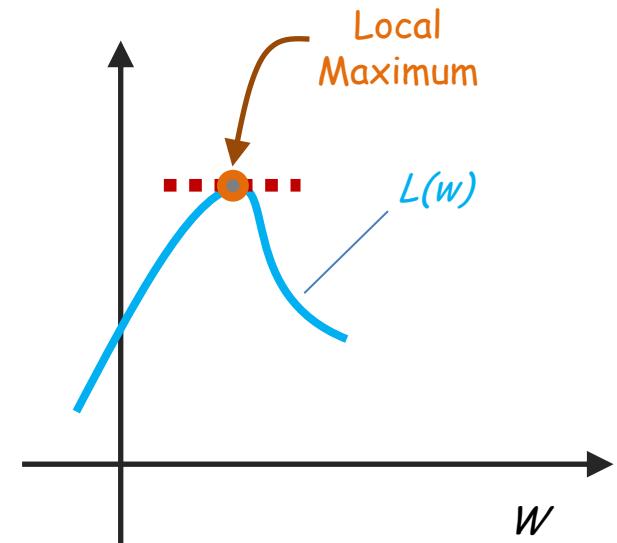
$$\max_{\vec{w}} \sum_{i=1}^N [y^{(i)} \log P(1|\vec{x}^{(i)}, \vec{w}) + (1 - y^{(i)}) \log(1 - P(1|\vec{x}^{(i)}, \vec{w}))]$$



*equivalent to*

Minimising Negative Log Likelihood:

$$\min_{\vec{w}} - \left( \sum_{i=1}^N [y^{(i)} \log P(1|\vec{x}^{(i)}, \vec{w}) + (1 - y^{(i)}) \log(1 - P(1|\vec{x}^{(i)}, \vec{w}))] \right)$$



# Gradient Vector

---

Negative Log Likelihood:

$$J(\vec{w}) = - \sum_{i=1}^N [y^{(i)} \log P(1|\vec{x}^{(i)}, \vec{w}) + (1 - y^{(i)}) \log(1 - P(1|\vec{x}^{(i)}, \vec{w}))]$$

Derivative of Log Likelihood:

$$\frac{\partial J(\vec{w})}{\partial w_j} = - \sum_{i=1}^N (y^{(i)} - \log P(1|\vec{x}^{(i)}, \vec{w})) x_j^{(i)}$$

Gradient Vector:

$$\nabla_{\vec{w}} J(\vec{w}) = -X^T(Y - P(1|X, \vec{w}))$$

# Pseudocode for Logistic Regression

```
# Logistic Regression – Gradient Descent on NLL

# Inputs
# data      ← (X, y) with y ∈ {0,1}
# n         ← learning rate
# max_iter ← maximum iterations
# tol       ← stop when ||∇L(w)|| ≤ tol
# X_query   ← examples to predict

# ----- fit -----
Φ ← concat_column(ones(N), X)           # design matrix with bias
w ← zeros(columns(Φ))                  # initialize
# NLL: L(w) = -  $\sum [y \log p + (1-y) \log(1-p)]$ , p = σ(Φw)
FOR t = 1 TO max_iter DO
    z ← Φ · w
    p ← 1 / (1 + exp(-z))              # sigmoid
    g ← transpose(Φ) · (p - y)          # ∇L(w)
    IF norm(g) ≤ tol THEN BREAK
    w ← w - n · g                      # GD step
END FOR

# ----- predict -----
Φ* ← concat_column(ones(|X_query|), X_query)
p* ← 1 / (1 + exp(-Φ* · w))
ŷ ← 1 if p* ≥ 0.5 else 0

RETURN p*, ŷ
```

# Logistic Regression from Scratch

```
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np
from scipy.special import expit # ← stable sigmoid

class MyLogisticRegression(BaseEstimator, ClassifierMixin):
    def __init__(self, eta=0.1, max_iter=2000, tol=1e-6, callback=None):
        self.eta, self.max_iter, self.tol = eta, max_iter, tol
        self.callback = callback

    def fit(self, X, y):
        Phi = np.c_[np.ones(X.shape[0]), X] # [1 | X]

        def grad(w): # ∇(avg NLL)
            p = expit(Phi @ w)
            return Phi.T @ (p - y)

        w0 = np.zeros(D)
        w, n_iter, gn = gradient_descent(
            grad, w0, eta=self.eta, max_iter=self.max_iter, tol=self.tol, callback=self.callback
        )

        self.weights_, self.n_iter_, self.grad_norm_ = w, n_iter, gn
        return self

    def predict_proba(self, X):
        Xs = np.c_[np.ones(X.shape[0]), X]
        z = Xs @ self.weights_
        p1 = expit(z)
        return np.column_stack([1 - p1, p1])

    def predict(self, X):
        return (self.predict_proba(X)[:, 1] ≥ 0.5).astype(int)
```

# Usage Example

```
import numpy as np
# from your_module import MyLogisticRegression # class already defined earlier

# ----- data generation -----
m, n = 100, 2
np.random.seed(0)

class_0 = np.hstack((1.5 + np.random.randn(m, 1), -1.5 + np.random.randn(m, 1)))
class_1 = np.hstack((-1.5 + np.random.randn(m, 1), 1.5 + np.random.randn(m, 1)))

X_train = np.vstack((class_0, class_1))          # shape (2m, 2)
y_train = np.concatenate([np.zeros(m), np.ones(m)]) # shape (2m,)

# one (or more) test points (no bias term needed; the model adds it)
X_test = np.array([[1.0, -1.0]])                # shape (1, 2)

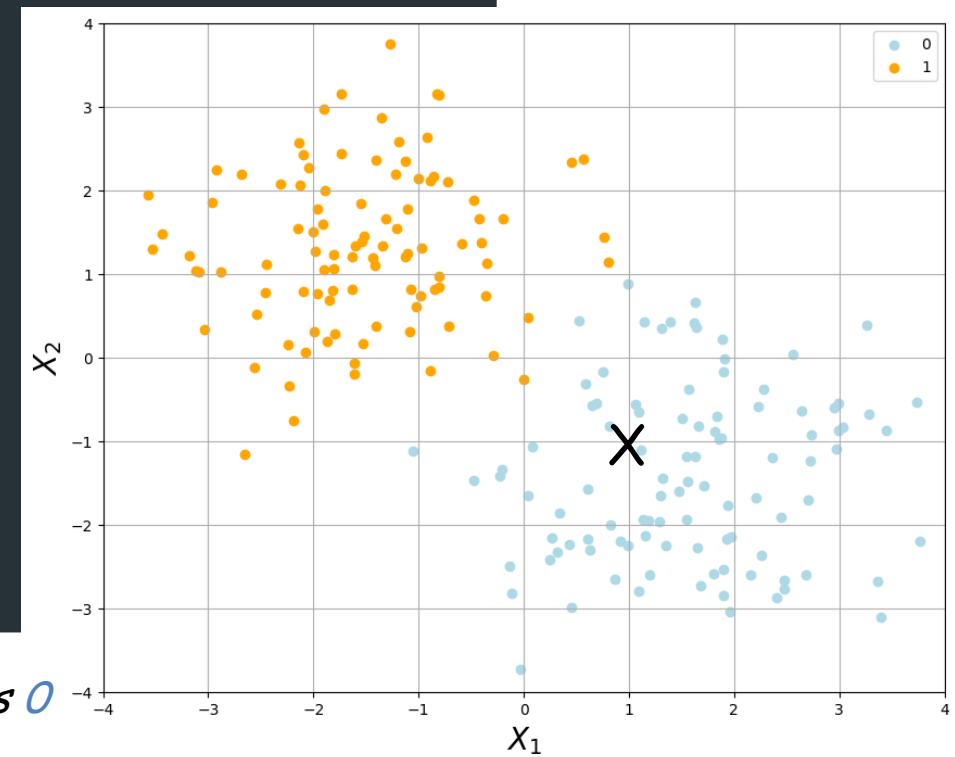
# ----- train -----
eta, max_iter, tol = 0.1, 1000, 1e-6
model = MyLogisticRegression(eta=eta, max_iter=max_iter, tol=tol)
model.fit(X_train, y_train)

# ----- predict on X_test -----
proba = model.predict_proba(X_test)[:, 1]          # P(y=1 | x)

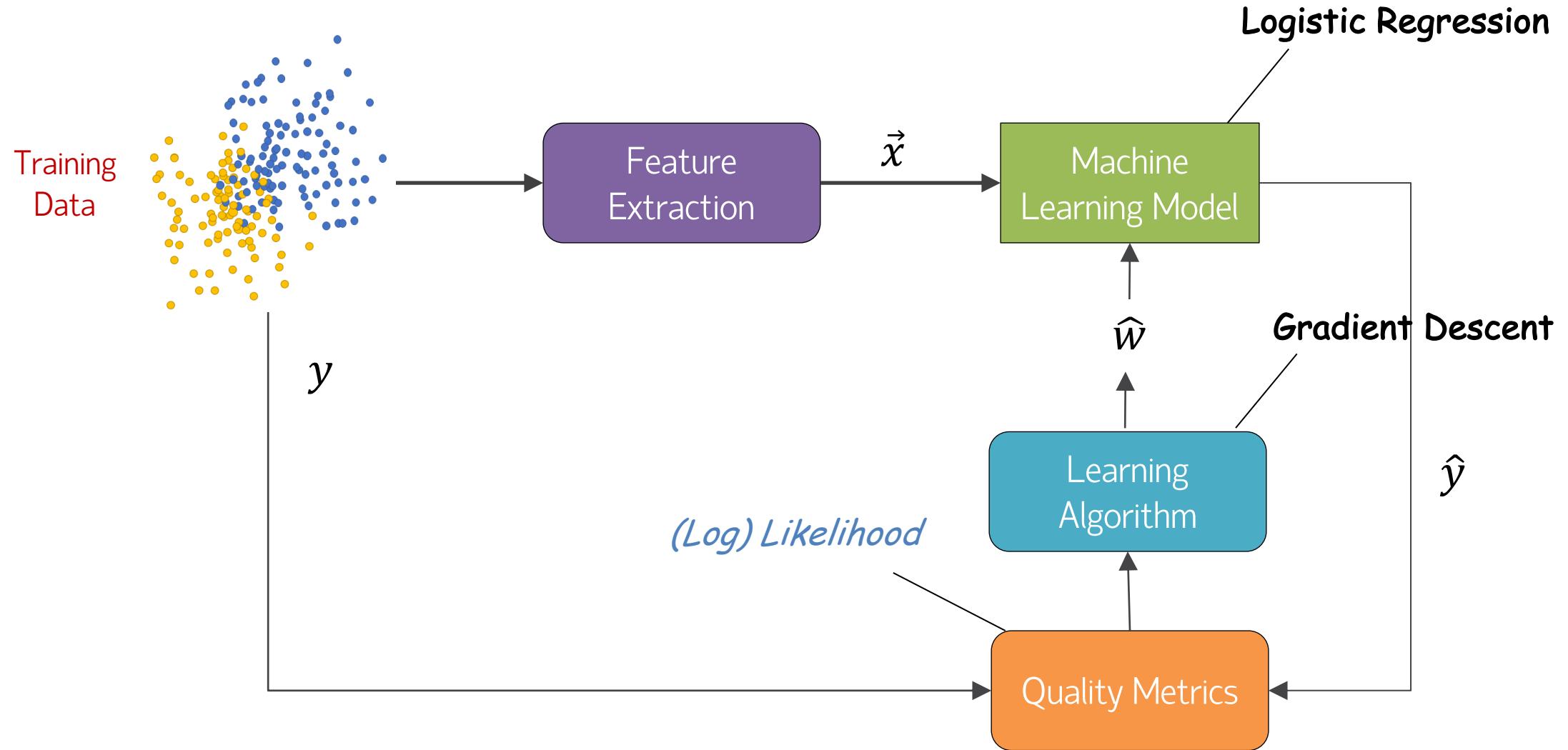
print(f"Predicted probability for X_test: {proba[0]:.4f}")
```

Predicted probability for X\_test: 0.0069173]

$\sim 0.007 \rightarrow \text{Class } 0$

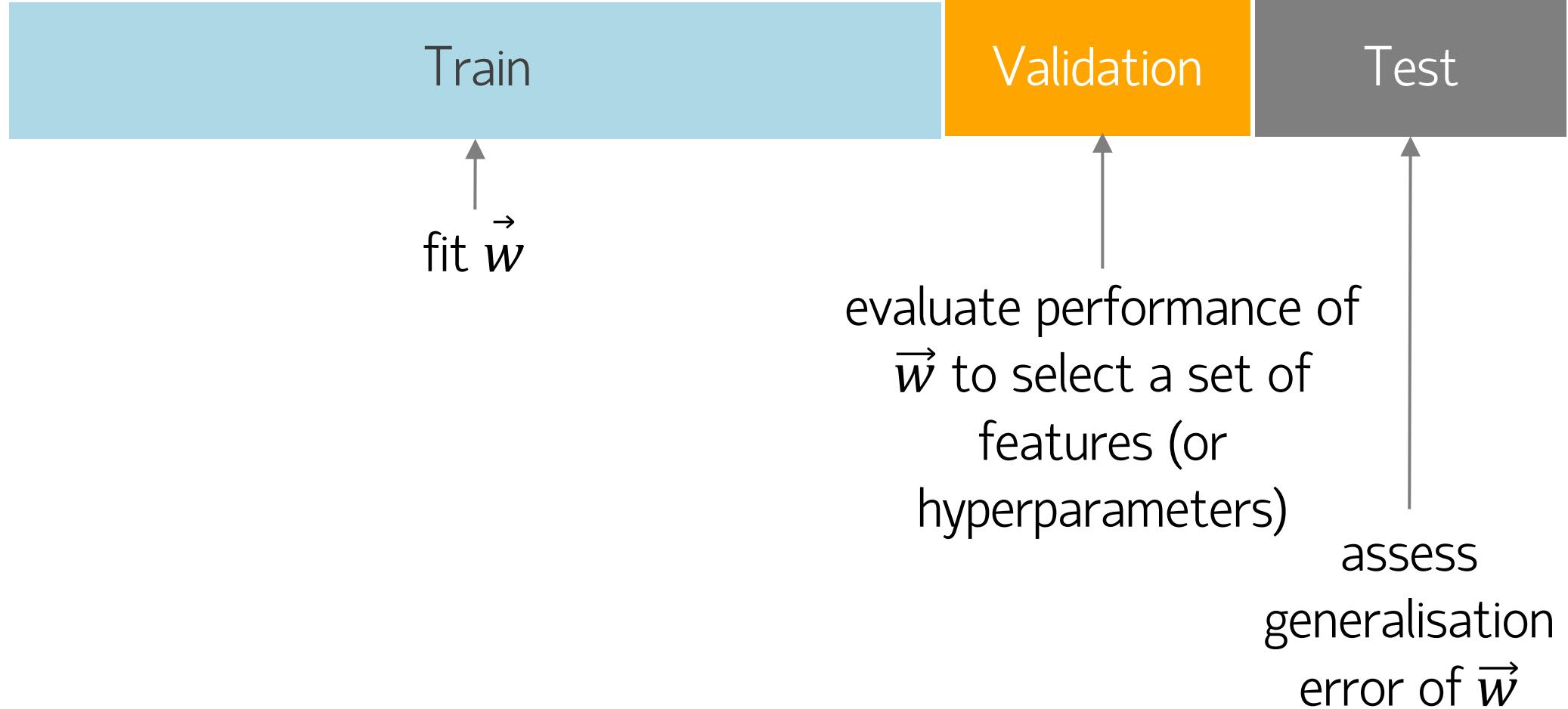


# Workflow: Logistic Regression



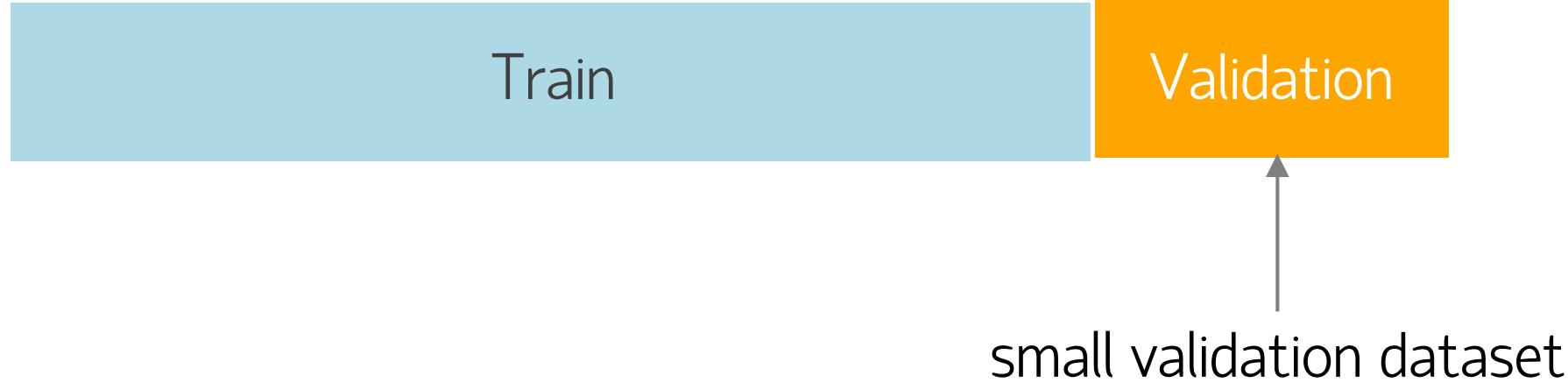
# Large Dataset

---



# Small Dataset

---



*Q: Is validation set enough to compare performance of  $\vec{w}$  across a set of features (or hyperparameters)?*

*A: No.*

# Choosing Validation Dataset

---



*Q: Which subset should we use?*

*A: All. We can average performance over all choices.*

# K-Fold Cross Validation

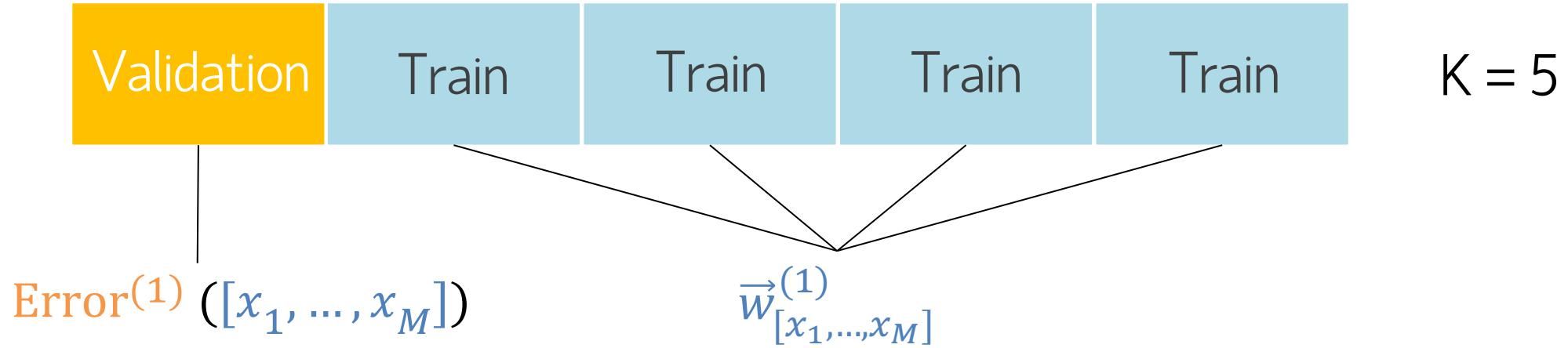
---



$k = 5$

We commonly use either 70% or 80% of the overall data for training. For K-fold, we randomly split the data into K groups (or folds), where K is typically either 3, 5 or 10.

# K-Fold Cross Validation



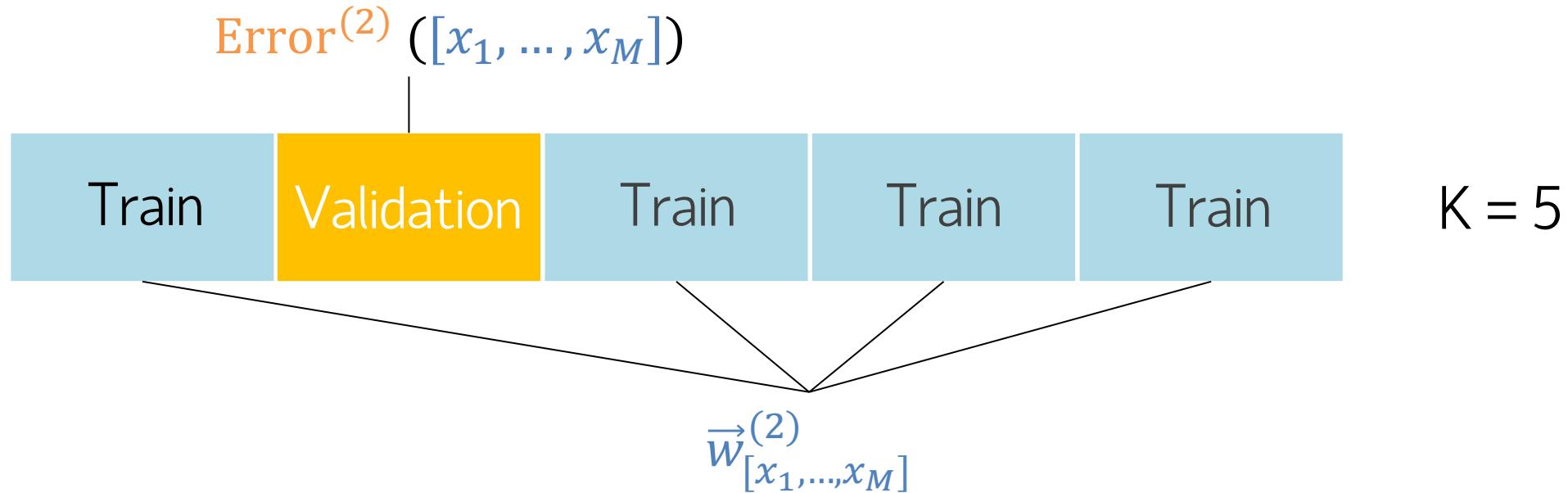
For k = 1, ..., K

[x<sub>1</sub>, ..., x<sub>M</sub>] is a set of features.

Step 1: Estimate  $\vec{w}_{[x_1, \dots, x_M]}^{(k)}$  on the training blocks.

Step 2: Compute error on Validation Block:  $\text{Error}^{(k)}([x_1, \dots, x_M])$ .

# K-Fold Cross Validation

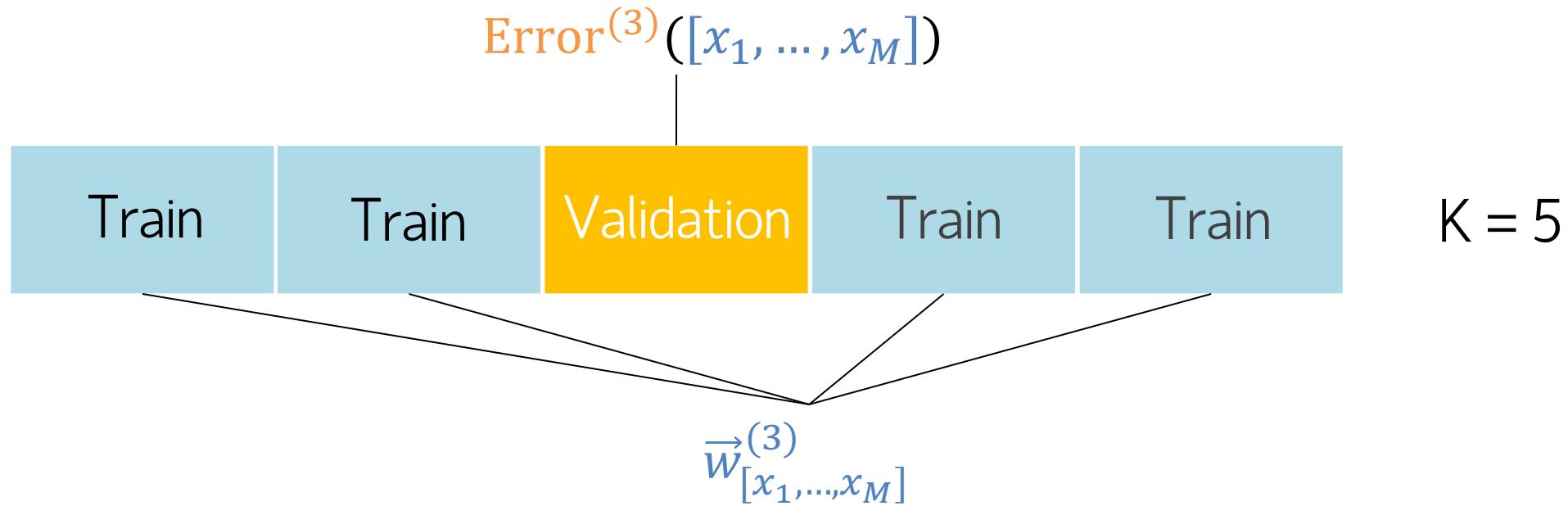


For  $k = 1, \dots, K$

Step 1: Estimate  $\vec{w}_{[x_1, \dots, x_M]}^{(k)}$  on the training blocks.

Step 2: Compute error on Validation Block: Error<sup>(k)</sup> ( $[x_1, \dots, x_M]$ ).

# K-Fold Cross Validation

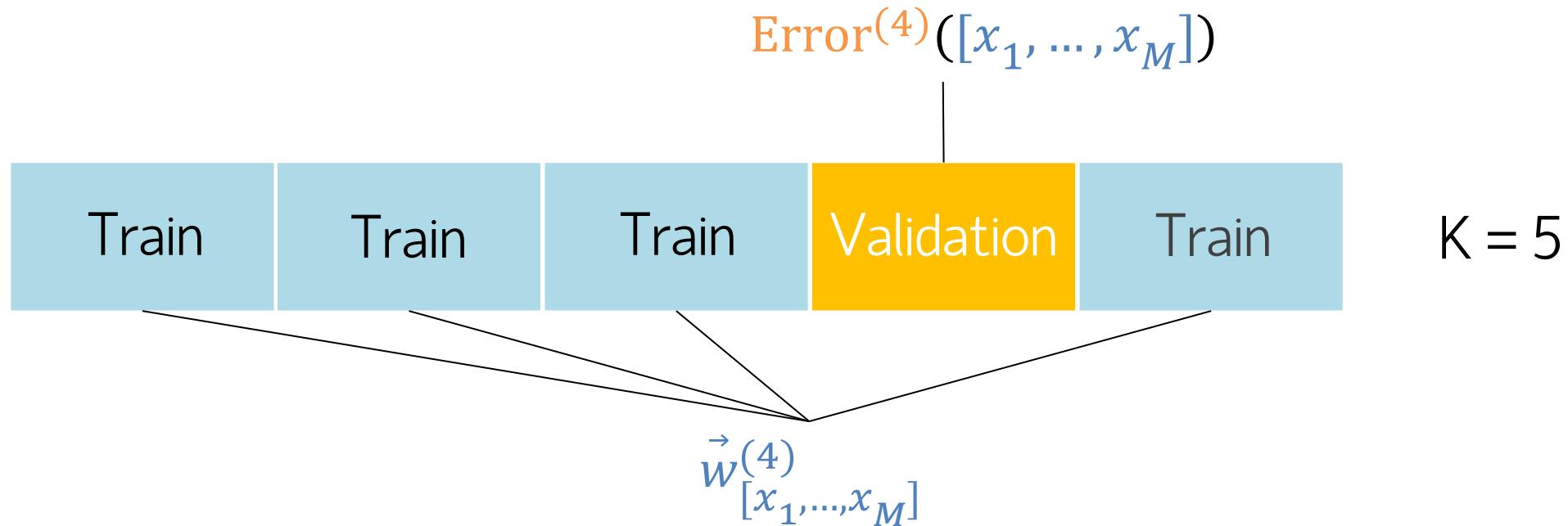


For  $k = 1, \dots, K$

Step 1: Estimate  $\vec{w}_{[x_1, \dots, x_M]}^{(k)}$  on the training blocks.

Step 2: Compute error on Validation Block:  $\text{Error}^{(k)}([x_1, \dots, x_M])$ .

# K-Fold Cross Validation

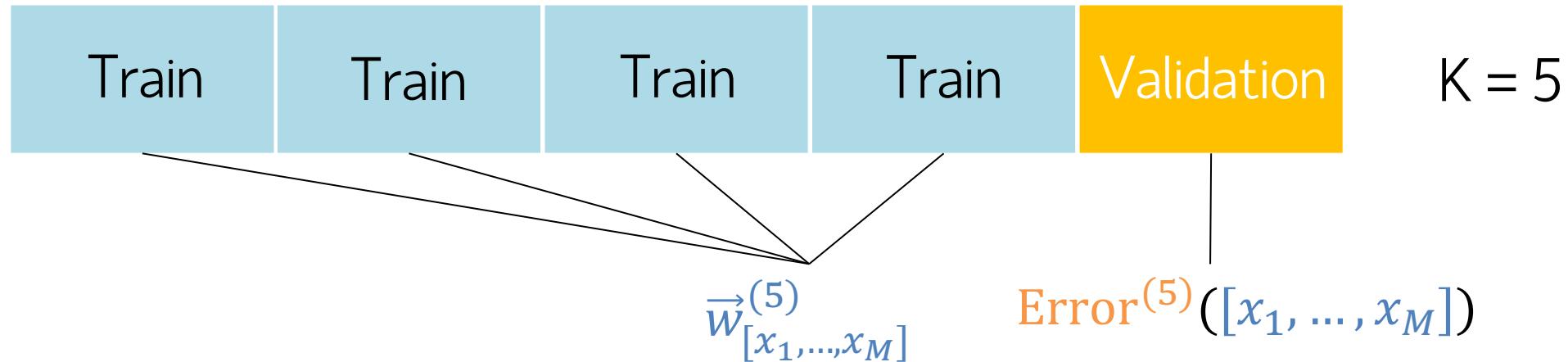


For k = 1, ..., K

Step 1: Estimate  $\vec{w}_{[x_1, \dots, x_M]}^{(k)}$  on the training blocks.

Step 2: Compute error on Validation Block:  $\text{Error}^{(k)}([x_1, \dots, x_M])$ .

# K-Fold Cross Validation



For  $k = 1, \dots, K$

Step 1: Estimate  $\vec{w}_{[x_1, \dots, x_M]}^{(k)}$  on the training blocks.

Step 2: Compute error on Validation Block:  $\text{Error}^{(k)}([x_1, \dots, x_M])$ .

Compute Average Error:  $\text{CV}([x_1, \dots, x_M]) = \frac{1}{K} \sum_{k=1}^K \text{Error}^{(k)}([x_1, \dots, x_M])$ .

# Pseudocode for Cross-Validation

```
# Inputs
#   model    ← untrained estimator (cloneable)
#   X, y     ← dataset
#   cv       ← splitter yielding (train_idx, val_idx) e.g., KFold
#   scoring  ← string name or callable scorer(est, X, y)

# ----- evaluate across folds -----
scores ← empty list

FOR (train_idx, val_idx) IN cv.split(X, y) DO
    est ← clone(model)                      # fresh copy per fold

    X_tr ← X[train_idx] ; y_tr ← y[train_idx]
    X_va ← X[val_idx] ; y_va ← y[val_idx]

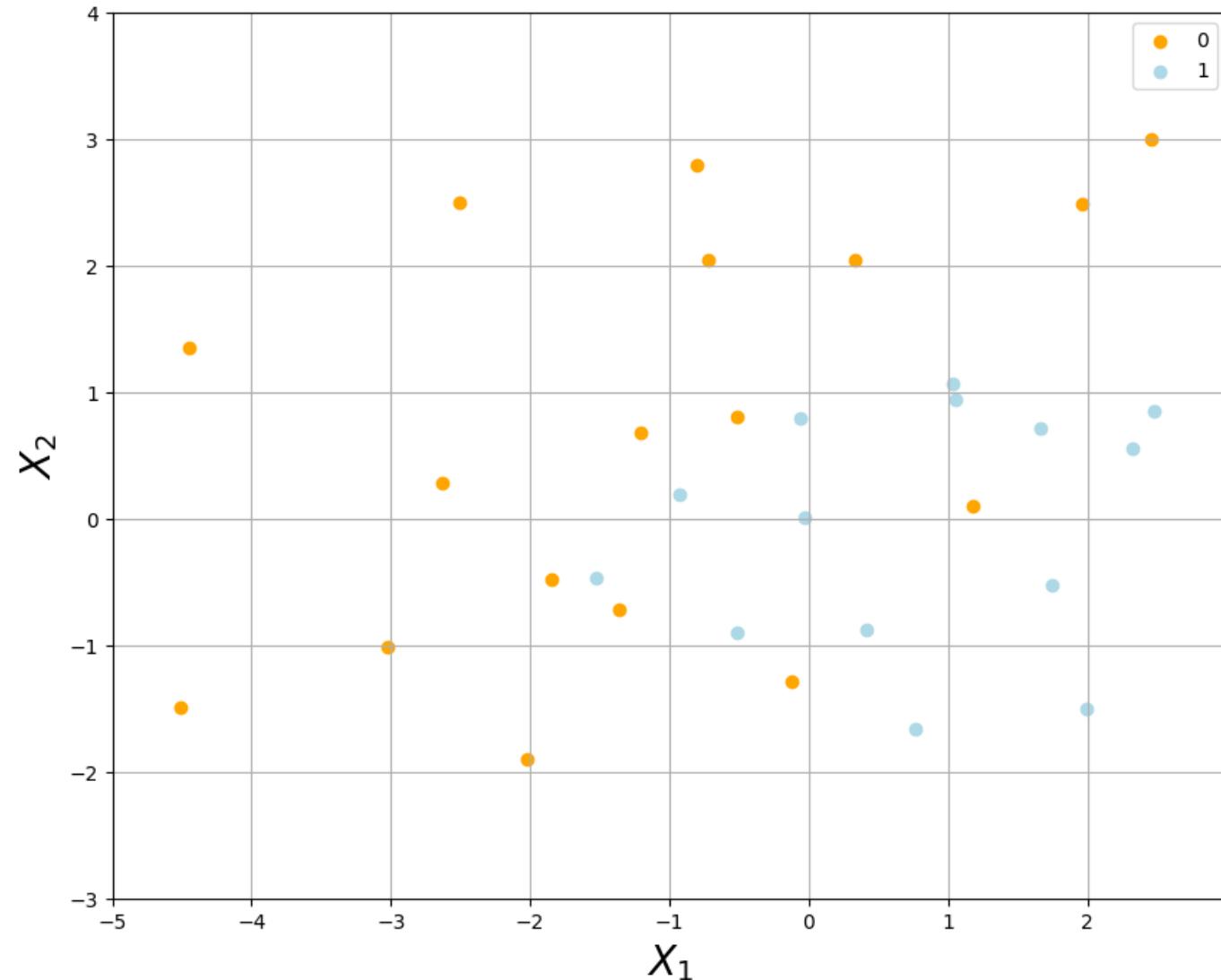
    fit est on (X_tr, y_tr)

    IF is_callable(scoring) THEN
        s ← scoring(est, X_va, y_va)
    ELSE
        s ← LOOKUP_SCORER(scoring)(est, X_va, y_va)
    END IF

    append s to scores
END FOR

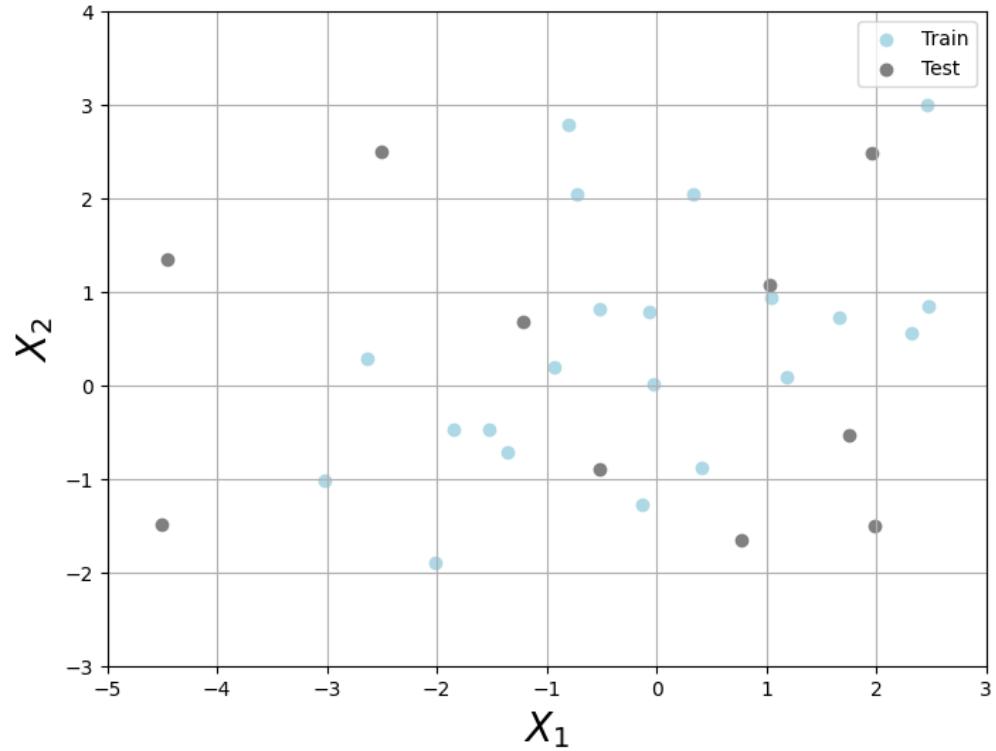
RETURN scores                                # like sklearn's cross_val_score
# (optional) mean_score ← mean(scores)
```

# Example Dataset



# Train, Validation and Test Datasets

```
from sklearn.model_selection import train_test_split  
  
# First, split the data into train (70%) and test (30%)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

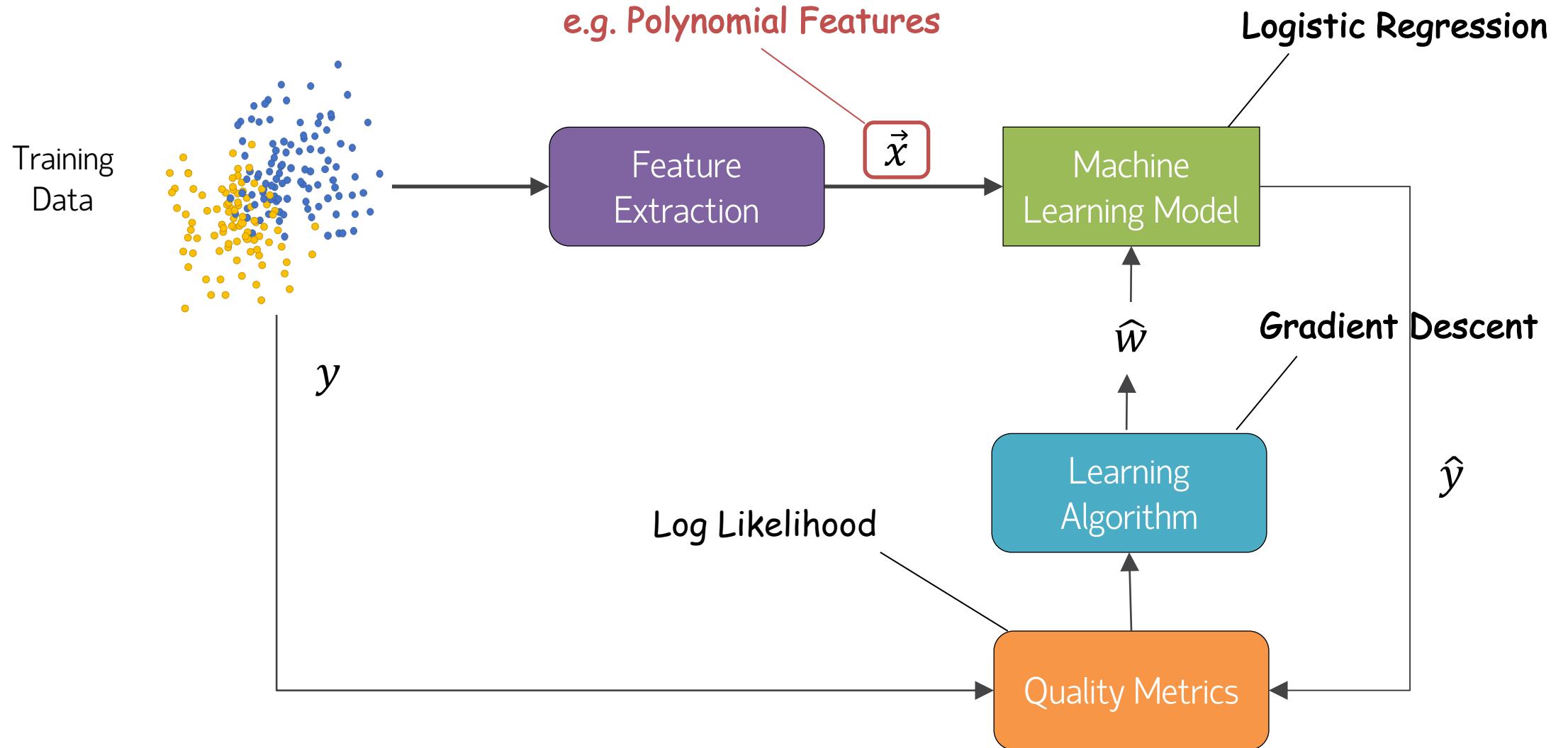


Train

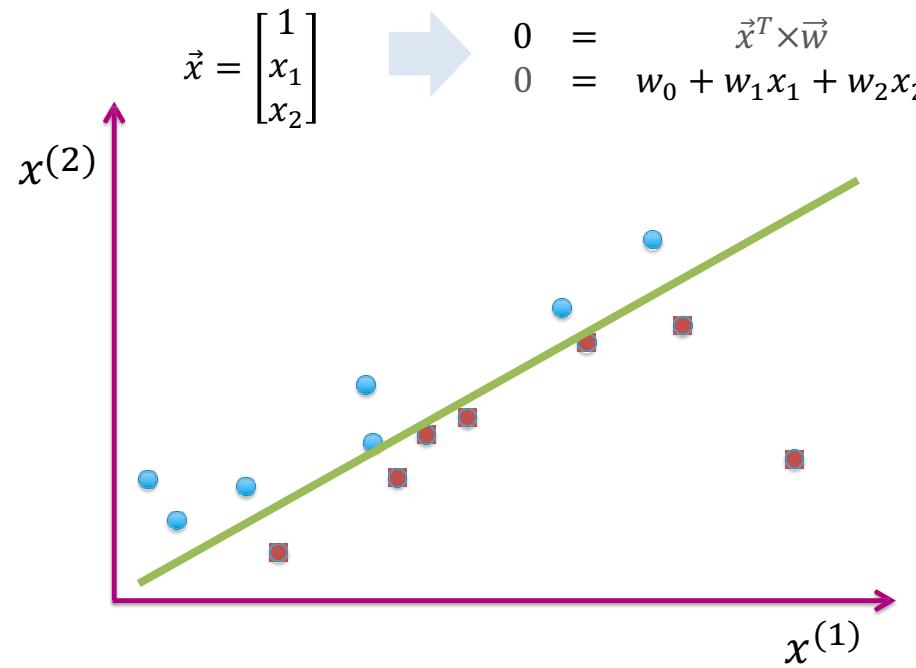
Test

- Train:Test is typically either 0.8:0.2 or 0.7:0.3.
- Test dataset is a proxy of unseen data, and it will only be used in the final evaluation.
- Train dataset is further divided into k folds (e.g., 5 or 10). For each fold, the model is trained on  $k-1$  folds and validated on the remaining fold.
- We use **K-Fold Cross-Validation** to fine-tune or optimize the ML model. Here, we find a set of hyper-parameters (or features) based on the average performance across folds.

# Workflow: Logistic Regression



# Feature Extraction: Polynomial Features



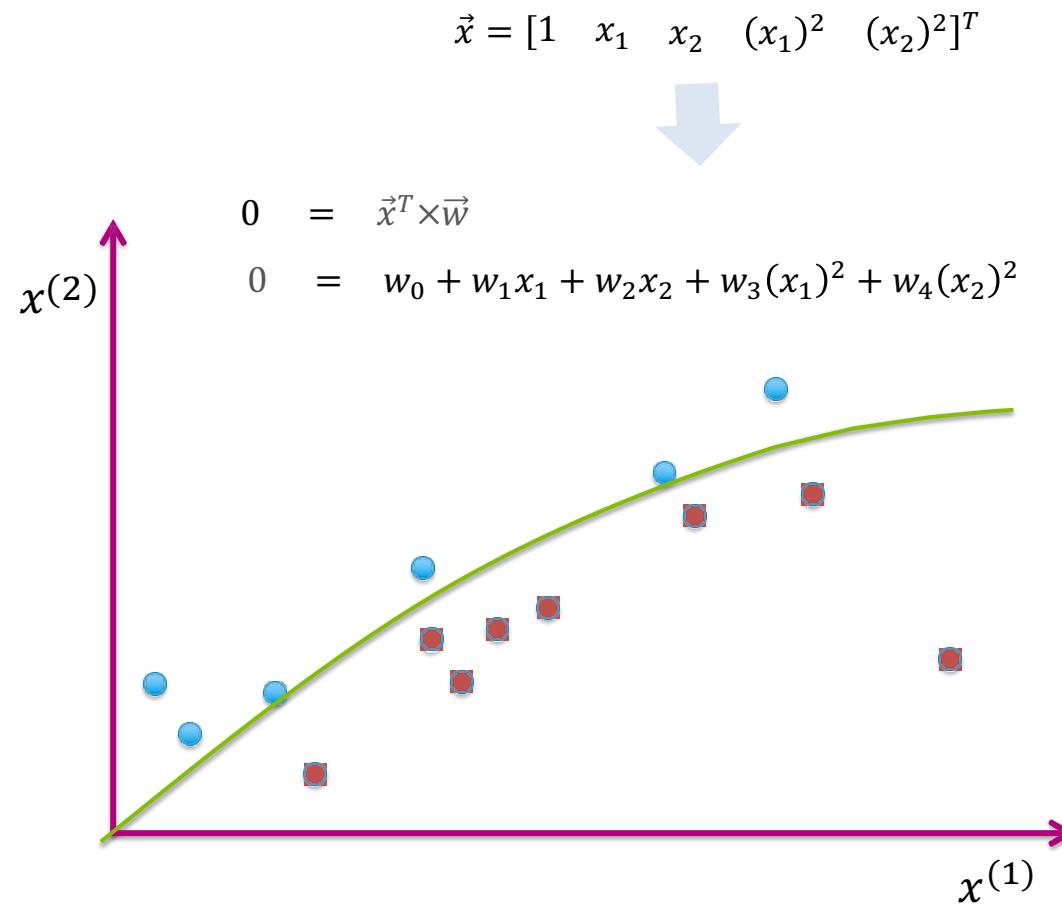
$$1^{\text{st}} \text{ Degree: } \vec{x} = [1 \ x_1 \ x_2]^T$$

$$2^{\text{nd}} \text{ Degree: } \vec{x} = [1 \ x_1 \ x_2 \ (x_1)^2 \ (x_2)^2]^T$$

$$3^{\text{rd}} \text{ Degree: } \vec{x} = [1 \ x_1 \ x_2 \ (x_1)^2 \ (x_2)^2 \ (x_1)^3 \ (x_2)^3]^T$$

⋮

$$N^{\text{th}} \text{ Degree: } \vec{x} = [1 \ x_1 \ x_2 \ \dots \ (x_1)^N \ (x_2)^N]^T$$



# scikit-learn: Logistic Regression

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score

# Step 1: Create an instance of LogisticRegression
model = LogisticRegression(penalty=None)

# Step 2: Define cross-validation strategy
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

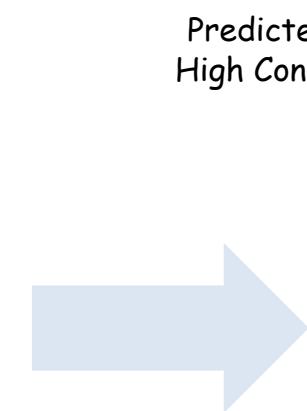
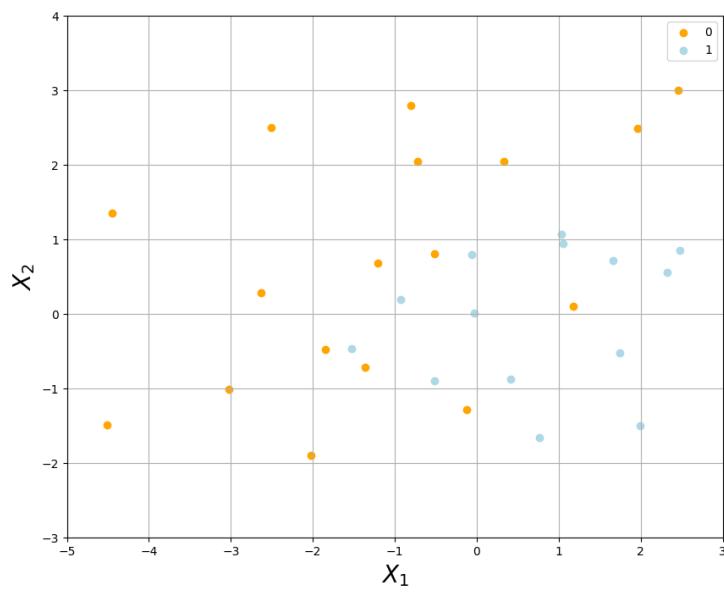
# Step 3: Perform cross-validation on the training data
cv_scores = cross_val_score(model, X_train, y_train, cv=kf, scoring='accuracy')
print("Mean Cross-Validation Accuracy: {:.2f}".format(np.mean(cv_scores)))
# optional: print per-fold scores
# print("Per-fold:", cv_scores)
```

Mean Cross-Validation Accuracy: 0.781

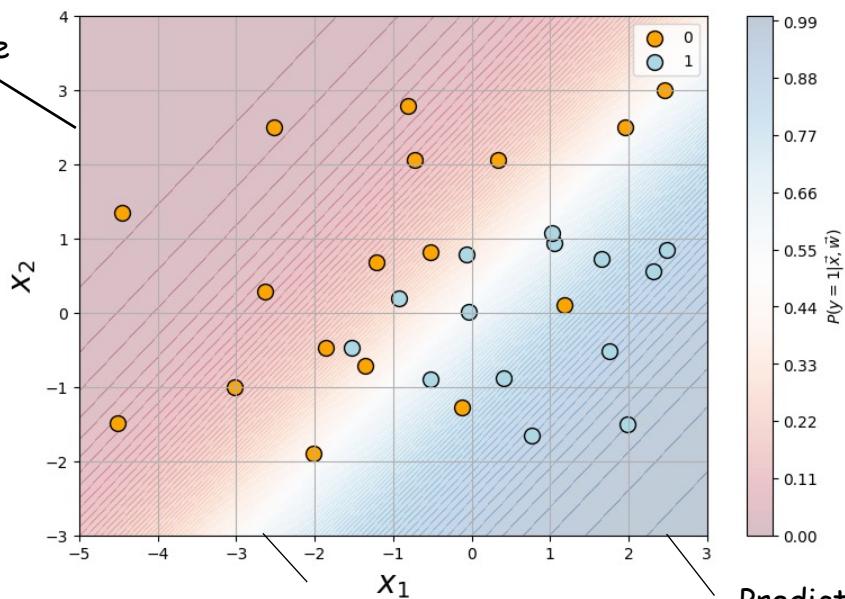


# Learned Decision Boundary

Feature	Coefficient
1	0.246
$x_1$	1.113
$x_2$	-1.062

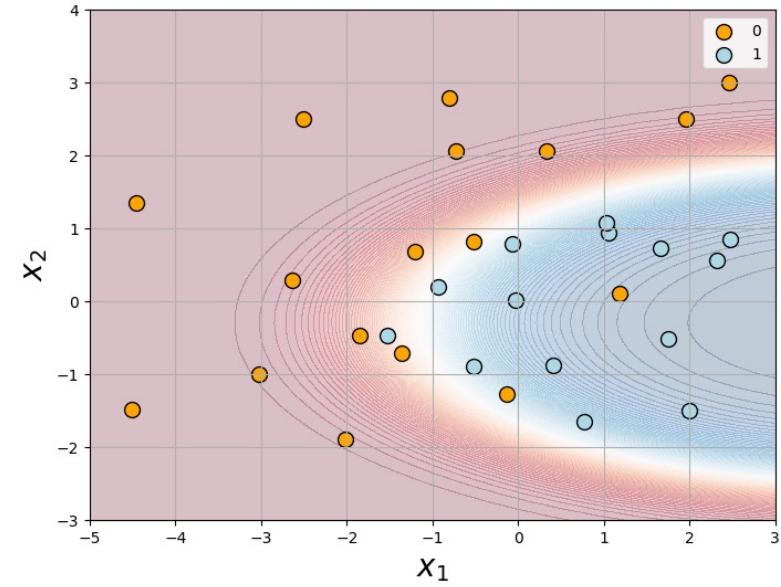
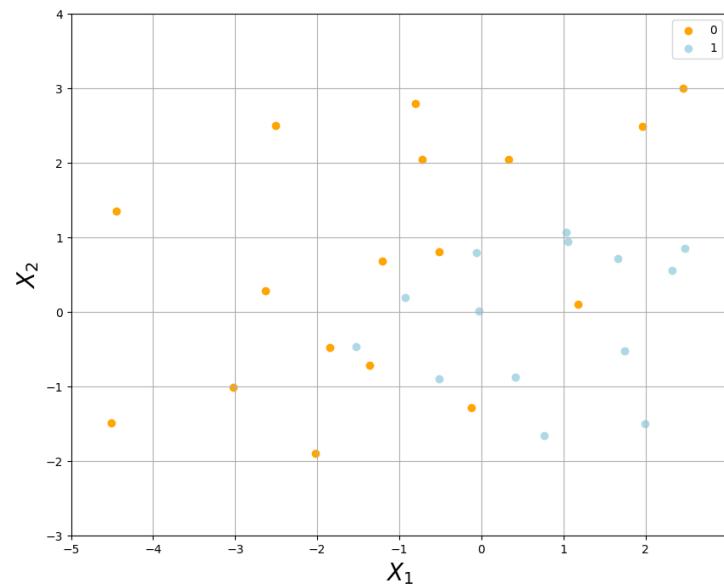


Predicted with  
High Confidence

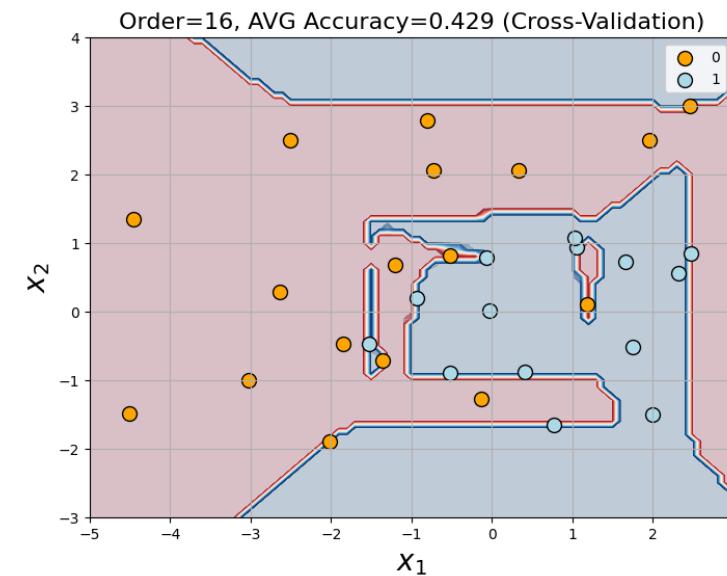
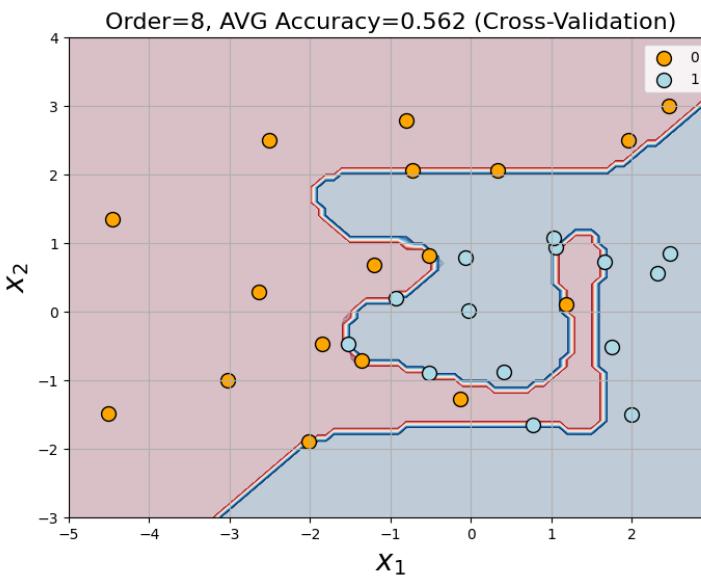
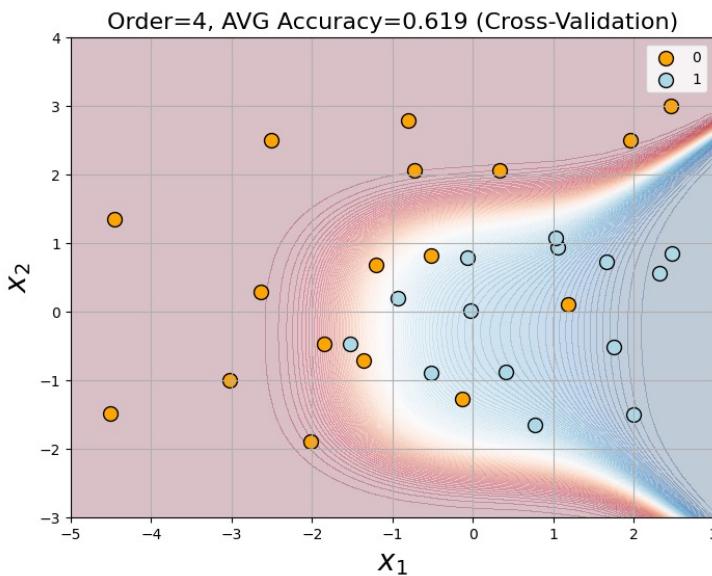
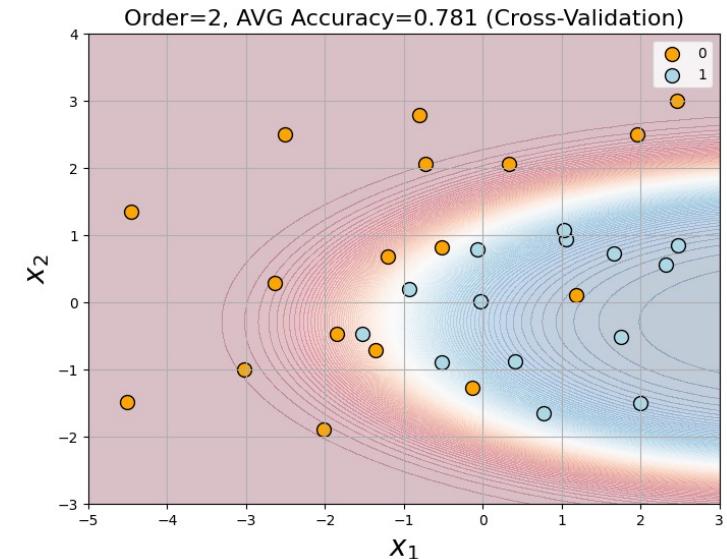
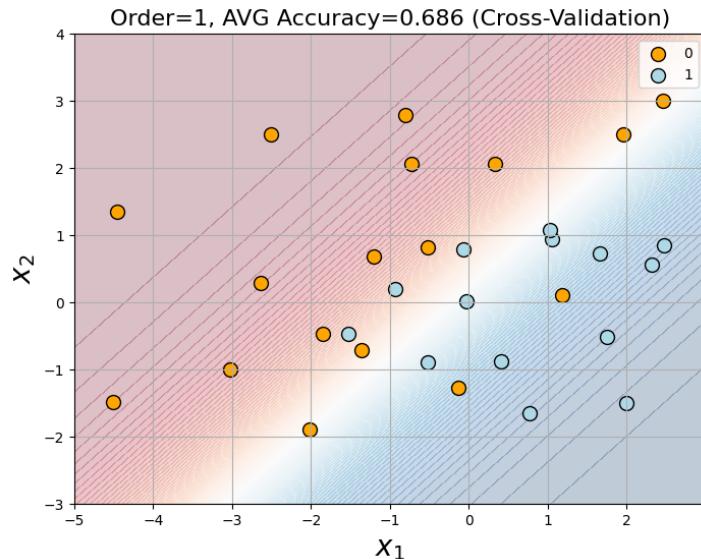
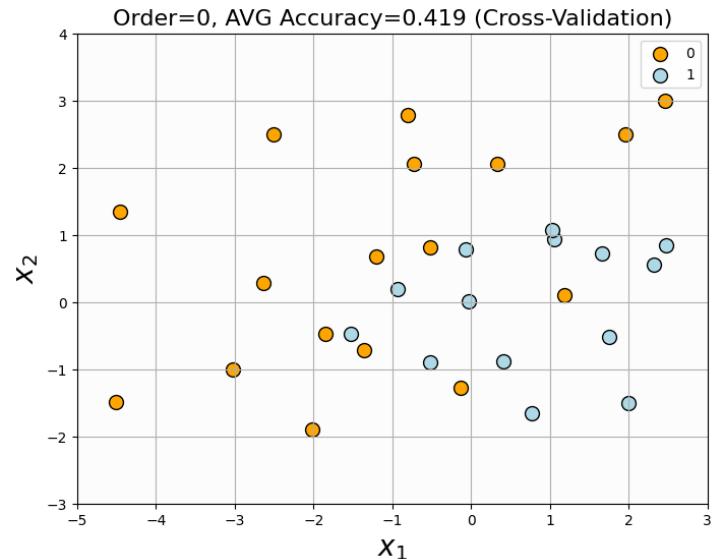


# Learned Decision Boundary (2<sup>nd</sup> Order)

Feature	Coefficient
1	1.717
$x_1$	1.385
$x_2$	-0.579
$x_1^2$	-0.167
$x_2^2$	0.978



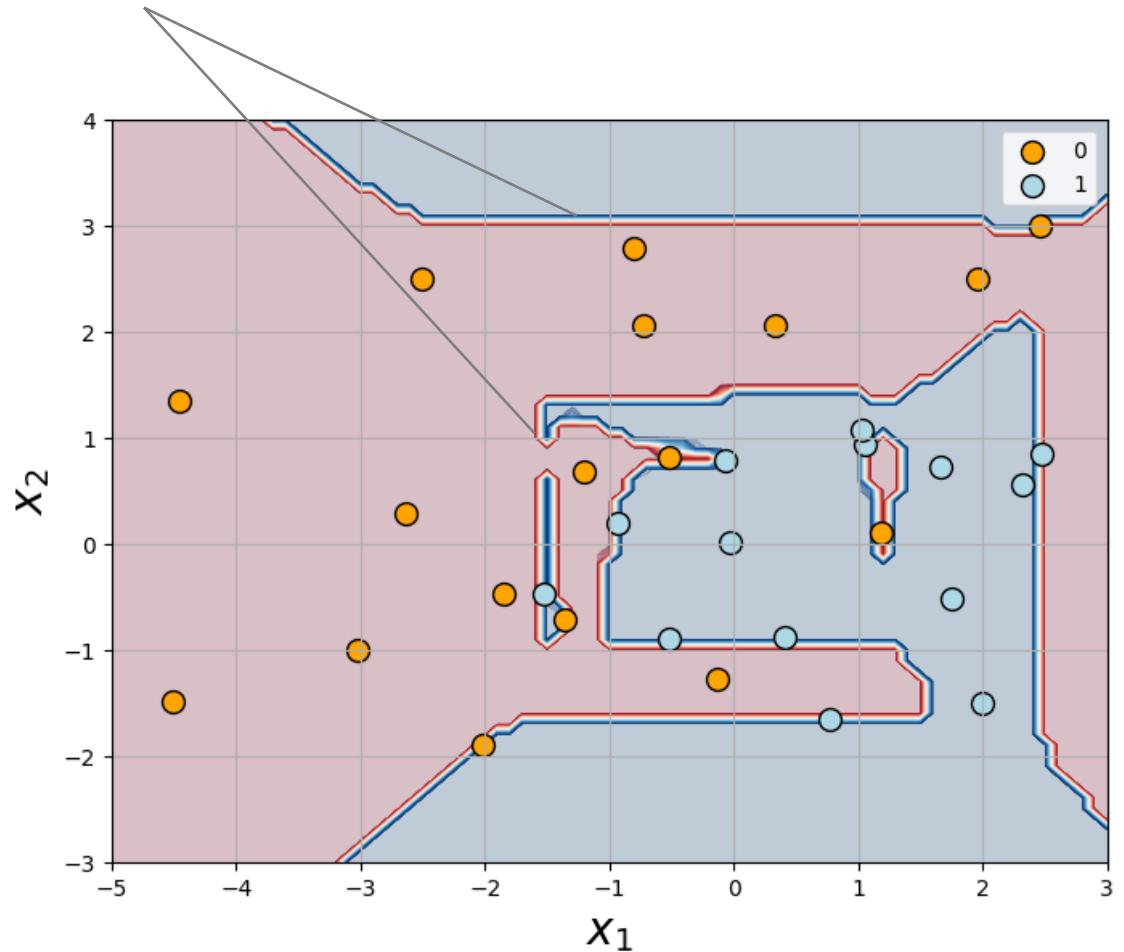
# Model Complexity



# Over-Confident Predictions

Tiny Uncertain Regions → Overfitting & Over-Confident Predictions

*We are sure we are right, when  
we are surely wrong.*



# Credit Card Fraud

---



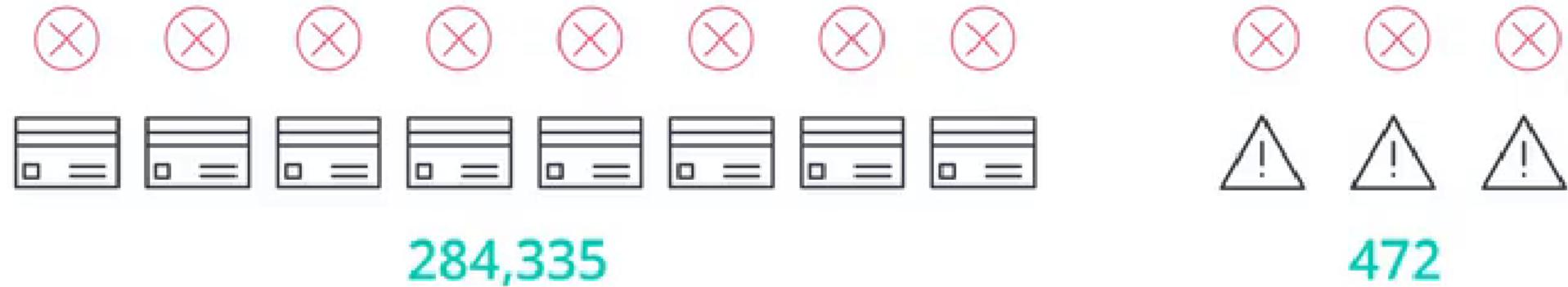
Our model predicted all transaction are good.

$$\begin{aligned} \text{Accuracy} &= \frac{284,335}{284,887} \\ &= 99.83\% \end{aligned}$$

Problem: We're not catching any of the bad ones.

# Credit Card Fraud

---



Our model predicted all transactions are fraudulent.

Great! Now, we're catching all of the fraudulent transactions.

Problem: We're accidentally catching all of the good ones.

# Quiz: False Positives and Negatives

		DIAGNOSIS	
		Diagnosed Sick	Diagnosed Healthy
PATIENTS	Diagnosed Sick		
	Diagnosed Healthy		

## Quiz: False Positives and Negatives

Which one do you think is worse, a false positive or a false negative? In other words, what is the worst mistake, to misdiagnose a healthy patient as sick or a sick patient as healthy?

- False Positive
- False Negative

		Diagnosis	
		DIAGNOSED SICK	DIAGNOSED HEALTHY
Patients	SHIELD	SICK	 FALSE NEGATIVE
	SICK	 FALSE POSITIVE	
	HEALTHY		

# Quiz: False Positives and Negatives

		FOLDER	
		Sent to Spam Folder	Sent to Inbox
EMAIL	Spam	 True Positive	 False Negative
	Not Spam	 False Positive	 True Negative

## Quiz: False Positives and Negatives

Which one do you think is worse, a false positive or a false negative? In other words, what is the worst mistake, to accidentally send your grandma's email to the spam folder or to accidentally send the spam email into your inbox?

- False Positive
- False Negative

		Folder	
		SENT TO SPAM	SENT TO INBOX
Emails	SPAM		
	NOT SPAM		
		<b>FALSE POSITIVE</b>	<b>FALSE NEGATIVE</b>

# Solution: False Positives and Negatives



Medical Model

FALSE POSITIVES OK

FALSE NEGATIVES NOT OK

OK IF NOT ALL ARE SICK  
FIND ALL THE SICK PEOPLE

*High Recall*



Spam Detector

FALSE POSITIVES NOT OK

FALSE NEGATIVES OK

DON'T NECESSARILY NEED  
TO FIND ALL THE SPAM

*High Precision*

# Precision

---

		DIAGNOSIS	
		Diagnosed Sick	Diagnosed Healthy
PATIENTS	Sick	1000	200 
	Healthy	800	9000

PRECISION: OUT OF THE PATIENTS WE DIAGNOSED WITH AN ILLNESS, HOW MANY DID WE CLASSIFY CORRECTLY?

$$\text{PRECISION} = \frac{1,000}{1,000 + 800} = 55.6\%$$

# Precision

---

		FOLDER
EMAIL		Sent to Spam Folder
	Spam	100
	Not Spam	30 
		Sent to Inbox
170	700	

OUT OF ALL THE E-MAILS  
SENT TO THE SPAM FOLDER,  
HOW MANY WERE ACTUALLY SPAM?

$$\text{PRECISION} = \frac{100}{100 + 30}$$

# Recall

---

		DIAGNOSIS	
		Diagnosed Sick	Diagnosed Healthy
PATIENTS	Sick	1000	200 
	Healthy	800	8000

OUT OF THE SICK PATIENTS,  
HOW MANY DID WE CORRECTLY  
DIAGNOSE AS SICK?

$$\text{RECALL} = \frac{1,000}{1,000 + 200} = 83.3\%$$

# Recall

---

		FOLDER	
		Sent to Spam Folder	Sent to Inbox
EMAIL	Spam	100	170
	Not Spam	30 	700

OUT OF ALL THE SPAM E-MAILS,  
HOW MANY WERE CORRECTLY  
SENT TO THE SPAM FOLDER?

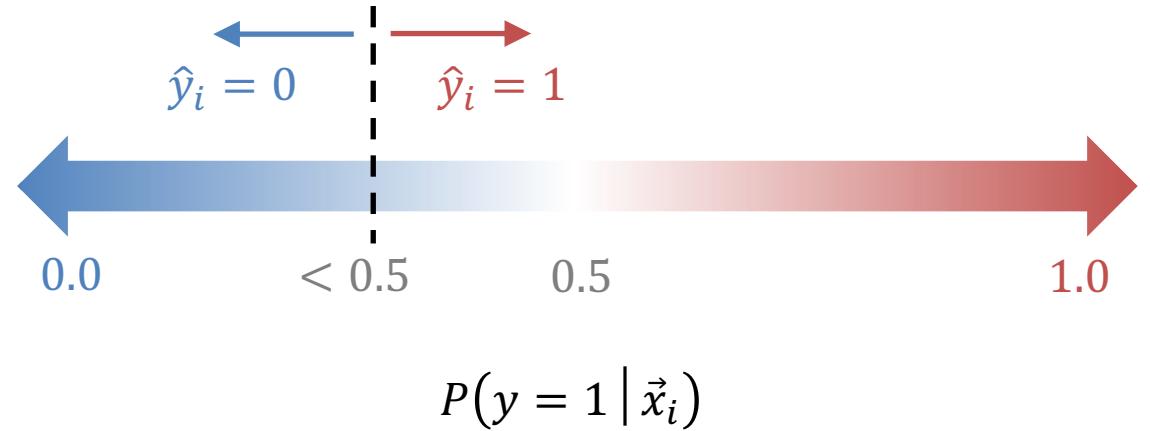
$$\text{Recall} = \frac{100}{100 + 170} = 37\%$$

# Output Probability and Recall Rate



*Q: How can we improve recall?*

*A: We set the threshold, i.e. cut-off probability, to be lower than 0.5.*



$$\text{Recall} = \frac{TP}{TP + FN}$$

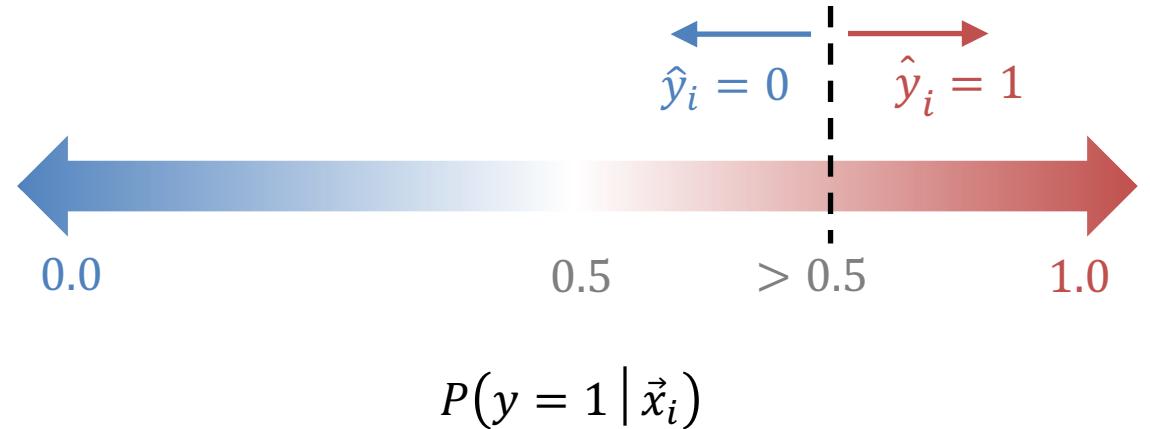
Lower Threshold → Higher TP → Higher Recall.  
(also Higher FP)

# Output Probability and Precision Rate



*Q: How can we improve precision?*

*A: We set the threshold, i.e. cut-off probability, to be higher than 0.5.*



$$\text{Precision} = \frac{TP}{TP + FP}$$

Higher Threshold  $\rightarrow$  Lower FP  $\rightarrow$  Higher Precision.  
(also Higher FN)

# $F_1$ Score

---

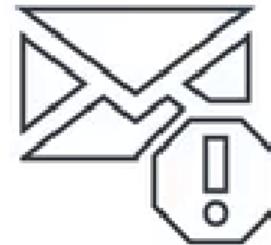


MEDICAL MODEL

PRECISION: 55.7%

RECALL: 83.3%

*One Score?*



SPAM DETECTOR

PRECISION: 76.9%

RECALL: 37%

$$F_1 \text{ Score} = \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad \left. \right\} \text{ Geometric Mean}$$

# Credit Card Fraud

---



Our model predicted all transaction are good.

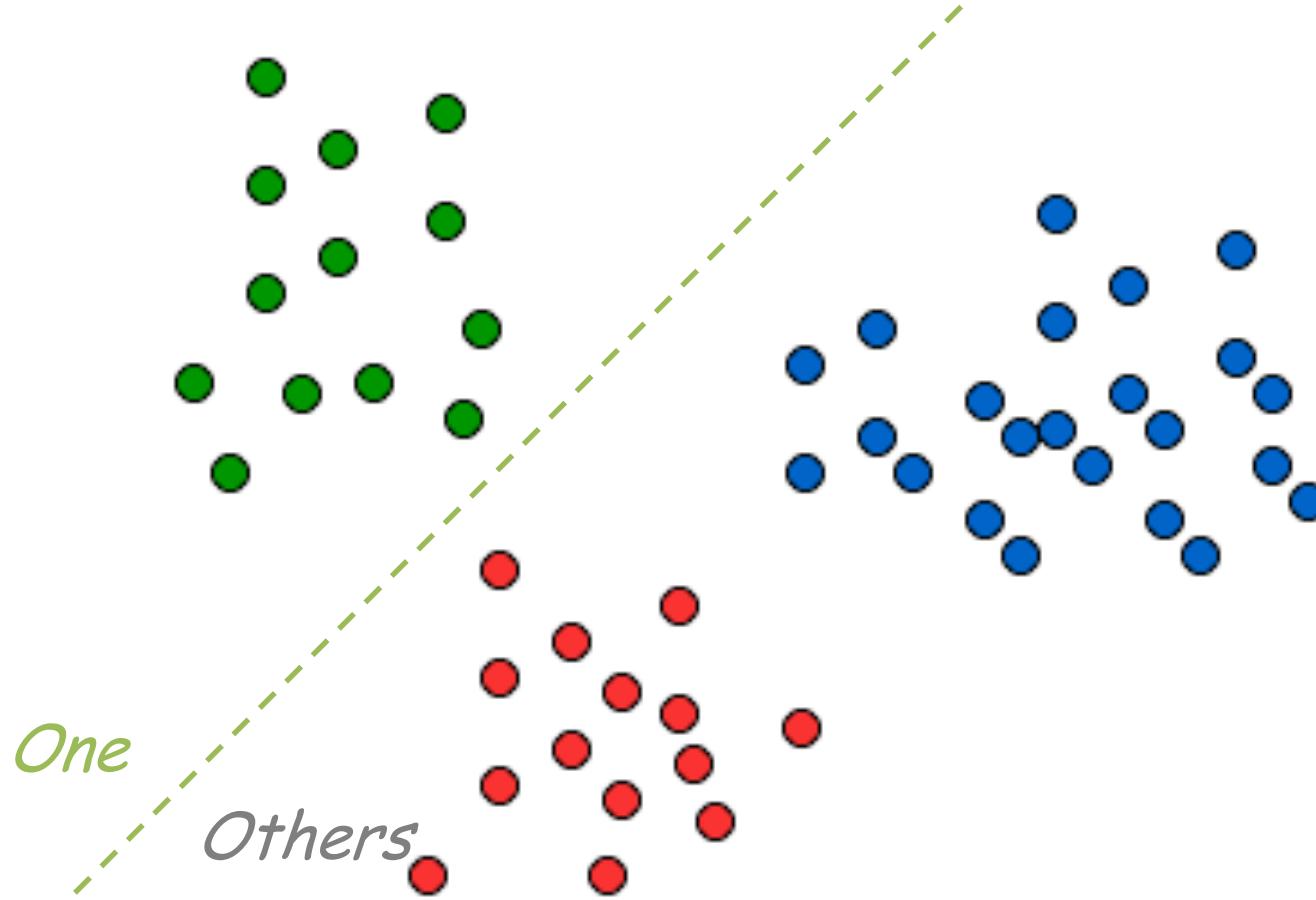
Prediction = 100%

$F_1$  Score = 0

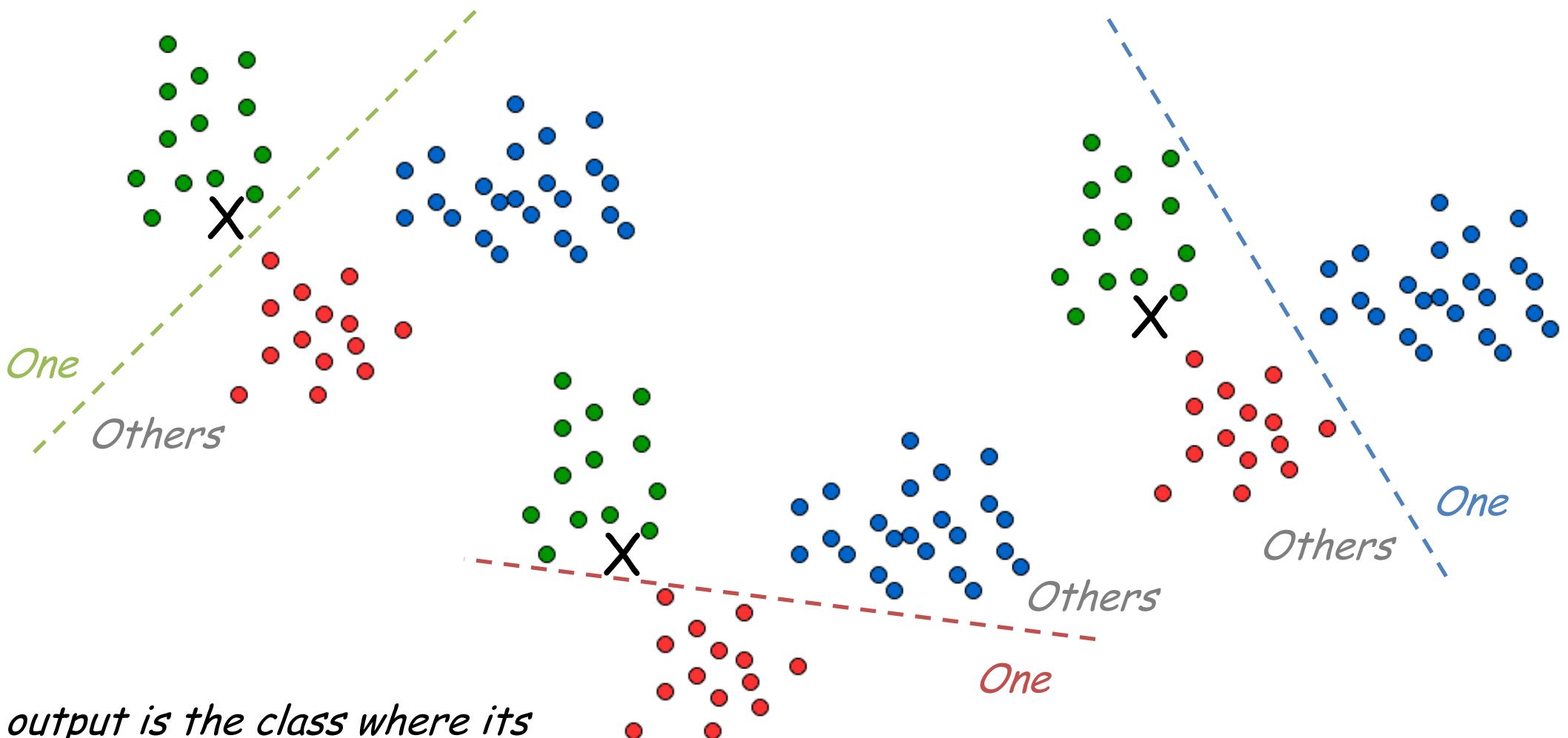
Recall = 0%

# Multiclass Classification

---



## 3 Logistic Classifiers



*The output is the class where its output probability is the highest (among the 3 classifiers).*

*Ex. [0.6, 0.4, 0.1]  $\rightarrow$  Green Class*

# Pseudocode for Multi-Class Classifiers

```
# Inputs
#   models  ← list of C trained binary classifiers (one per class)
#   C       ← number of classes (C = length(models))
#   X_query ← set of examples to predict

# ----- predict -----
ŷ    ← list of length |X_query|                                # predicted class per example
P    ← matrix of size |X_query| × C                            # per-class probabilities (optional)

FOR i = 1 TO |X_query| DO
  x* ← X_query[i]

  FOR c = 1 TO C DO
    P[i, c] ← models[c].predict_probability(x*)  # p(y = c | x*)
  END FOR

  ŷ[i] ← argmax_index(P[i, :])                                # class with highest prob (1..C)
END FOR

RETURN ŷ, P
```



# Summary

---

- Logistic regression models the probability of a binary outcome by applying a logistic (sigmoid) function to a linear combination of features. This function ensures predicted probabilities are bounded between 0 and 1, making it suitable for classification tasks.
- The decision boundary in logistic regression is defined by a linear combination of features. Data points on either side of this boundary are assigned different class labels, making it useful in applications with linearly separable classes.
- Maximum likelihood estimation (MLE) is used to determine the optimal weights. By maximizing the likelihood (or equivalently, the log-likelihood) of the observed data, logistic regression finds parameters that best explain the class distribution. This is typically achieved through iterative optimization methods like gradient descent.
- Threshold tuning is critical for practical applications. Adjusting the decision threshold allows control over the sensitivity to false positives vs. false negatives. This adjustment makes logistic regression a versatile choice for tasks like fraud detection, medical diagnostics, and risk assessment.
- K-Fold Cross Validation enhances evaluation by splitting data into multiple folds, with each fold used once for validation and the rest for training. This method, often with K=5, maximizes data usage, making it ideal for small datasets. It provides a reliable performance measure by reducing overfitting and ensuring the model generalizes well to unseen data.