Polynomial Regression

Tian Han

Outline

- Polynomial Regression
- Binary Classification - Logistic regression

Warm-up: Linear Regression

Linear Regression (Task)

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$

Output: a vector $\mathbf{w} \in \mathbb{R}^d$ and scalar $b \in \mathbb{R}$ such that $\mathbf{x}_i^T \mathbf{w} + b \approx y_i$.

assume y_i is a linear function of $\mathbf{x}_i$.

Tasks

Linear Regression

Least Squares Regression (Method)

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$

1. Add one dimension to $\mathbf{x} \in \mathbb{R}^d$: $\bar{\mathbf{x}}_j = [\mathbf{x}_j; 1] \in \mathbb{R}^{d+1}$.
2. Solve least squares regression: $\min_{\mathbf{w} \in \mathbb{R}^{d+1}} \left|\left|\bar{\mathbf{X}}\,\mathbf{w} - \mathbf{y}\right|\right|_2^2$.

Tasks

Methods

Linear Regression

Least Squares Regression

Least Squares Regression (Method)

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$

1. Add one dimension to $\mathbf{x} \in \mathbb{R}^d$: $\bar{\mathbf{x}}_j = [\mathbf{x}_j; 1] \in \mathbb{R}^{d+1}$.
2. Solve least squares regression: $\min_{\mathbf{w} \in \mathbb{R}^{d+1}} \| \bar{\mathbf{X}} \mathbf{w} - \mathbf{y} \|_2^2$.

Tasks	Methods	Algorithms
Linear Regression	Least Squares Regression	Analytical Solution Gradient Descent

Polynomial Regression

The Regression Task

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

Output: a function $f: \mathbb{R}^d \mapsto \mathbb{R}$ such that $f(\mathbf{x}) \approx y$.

Question: f is unknown! So how to learn f?

The Regression Task

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

Output: a function f: $\mathbb{R}^d \mapsto \mathbb{R}$ such that $f(\mathbf{x}) \approx y$.

Question: f is unknown! So how to learn f?

Answer: polynomial approximation; f is a polynomial function.

Taylor expansion: $f(x) = f(a) + f'(a)(a - x) + \frac{f''(a)}{2!}(a - x)^2 + \cdots$

Polynomial Regression: 1D Example

Input: scalars $x_1, \cdots, x_n \in \mathbb{R}$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

Output: a function $f: \mathbb{R} \mapsto \mathbb{R}$ such that $f(x) \approx y$.

One-dimensional example: $f(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_p x^p$.

Polynomial Regression: 1D Example

Input: scalars $x_1, \cdots, x_n \in \mathbb{R}$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

Output: a function $f: \mathbb{R} \mapsto \mathbb{R}$ such that $f(x) \approx y$.

One-dimensional example: $f(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_p x^p$.

Polynomial regression:

1. Define a feature map $\boldsymbol{\phi}(x) = [1,\ x,\ x^2,\ x^3, \cdots, x^p]$.
2. For $j = 1$ to n, do the mapping $x_j \mapsto \boldsymbol{\phi}(x_j)$.
    - Let $\boldsymbol{\Phi} = [\boldsymbol{\phi}(x_1); \cdots, \boldsymbol{\phi}(x_n)]^T \in \mathbb{R}^{n \times (p+1)}$
3. Solve the least squares regression $\min_{\mathbf{w} \in \mathbb{R}^{p+1}} \quad ||\boldsymbol{\Phi}\,\mathbf{w} - \mathbf{y}||_2^2$.

Polynomial Regression: 2D Example

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^2$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

Output: a function $f: \mathbb{R}^2 \mapsto \mathbb{R}$ such that $f(\mathbf{x}_i) \approx y_i$.

Two-dimensional example: how to do feature mapping?

Polynomial features:

$$\boldsymbol{\phi}(\mathbf{x}) = [\underbrace{1}_{\text{degree-0}}, \underbrace{x_1, \ x_2,}_{\text{degree-1}} \ \underbrace{x_1^2, \ x_2^2, \ x_1 x_2,}_{\text{degree-2}} \ \underbrace{x_1^3, \ x_2^3, \ x_1 x_2^2, \ x_1^2 x_2}_{\text{degree-3}}].$$

Polynomial Regression

```
import numpy
X = numpy.arange(6).reshape(3, 2)
print('X = ')
print(X)
```

```
X =
[[0 1]
 [2 3]
 [4 5]]
```

Polynomial Regression

```python
import numpy
X = numpy.arange(6).reshape(3, 2)
print('X = ')
print(X)
```

```
X =
[[0 1]
 [2 3]
 [4 5]]
```

```python
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=3)
Phi = poly.fit_transform(X)
print('Phi = ')
print(Phi)
```

```
Phi =
[[   1.    0.    1.    0.    0.    1.    0.    0.    0.    1.]
 [   1.    2.    3.    4.    6.    9.    8.   12.   18.   27.]
 [   1.    4.    5.   16.   20.   25.   64.   80.  100.  125.]]
```

degree-0 degree-1 degree-2 degree-3

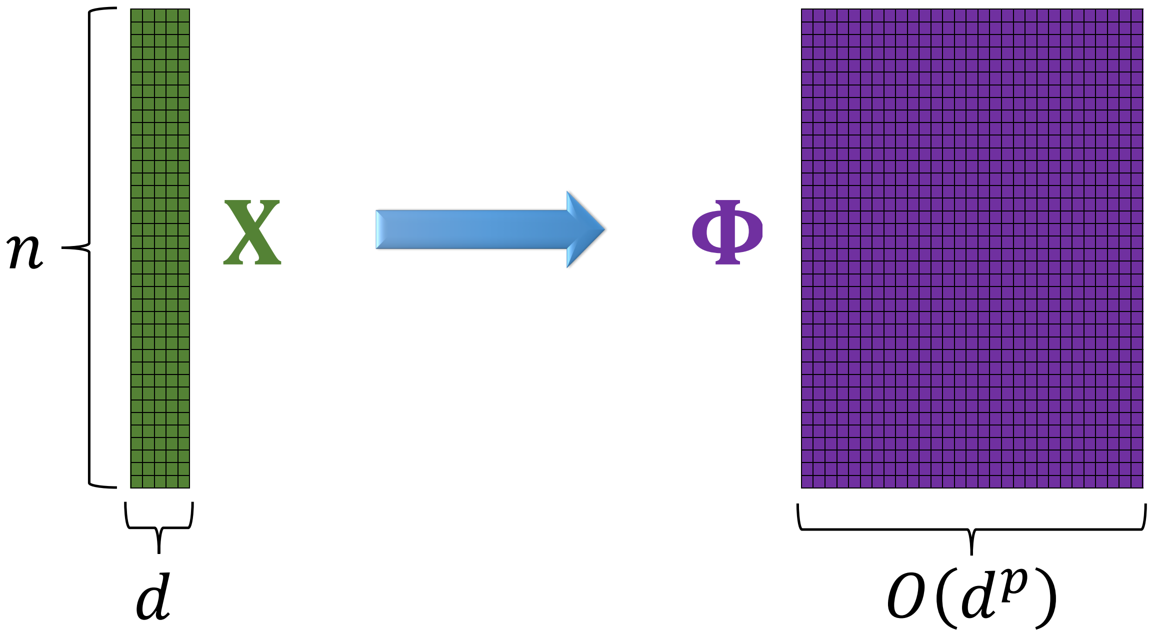
Polynomial Regression

- $\mathbf{x}$: d-dimensional
- $\boldsymbol{\phi}(\mathbf{x})$: degree-$p$ polynomial
- The dimension of $\boldsymbol{\phi}(\mathbf{x})$ is $O(d^p)$

Polynomial Regression

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

Output: a function $f: \mathbb{R}^d \mapsto \mathbb{R}$ such that $f(\mathbf{x}_i) \approx y_i$.



Training, Test, and Overfitting

Polynomial Regression: Training

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

Feature map: $\boldsymbol{\phi}(\mathbf{x})$. Its dimension is $O(d^p)$.

Least squares: $\min_{\mathbf{w}} \; ||\boldsymbol{\Phi}\,\mathbf{w} - \mathbf{y}||_2^2$.

Polynomial Regression: Training

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

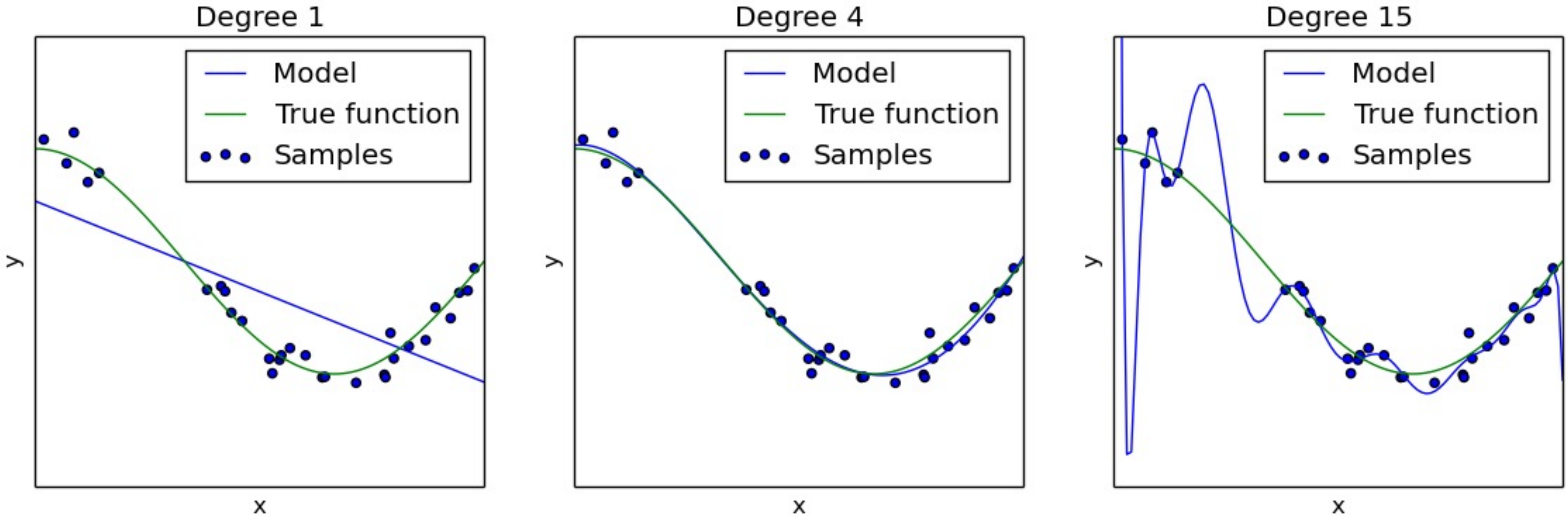
Feature map: $\boldsymbol{\phi}(\mathbf{x})$. Its dimension is $O(d^p)$.

Least squares: $\min_{\mathbf{w}} \; ||\boldsymbol{\Phi}\, \mathbf{w} - \mathbf{y}||_2^2$.

Question: what will happen as p grows?

1. For sufficiently large p, the dimension of the feature $\boldsymbol{\phi}(\mathbf{x})$ exceeds n.
2. Then you can find $\mathbf{w}$ such that $\boldsymbol{\Phi}\, \mathbf{w} = \mathbf{y}$. (Zero training error!)

Training and Testing



Underfitting

Overfitting

Training and Testing

Train:

Input: vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{R}$.

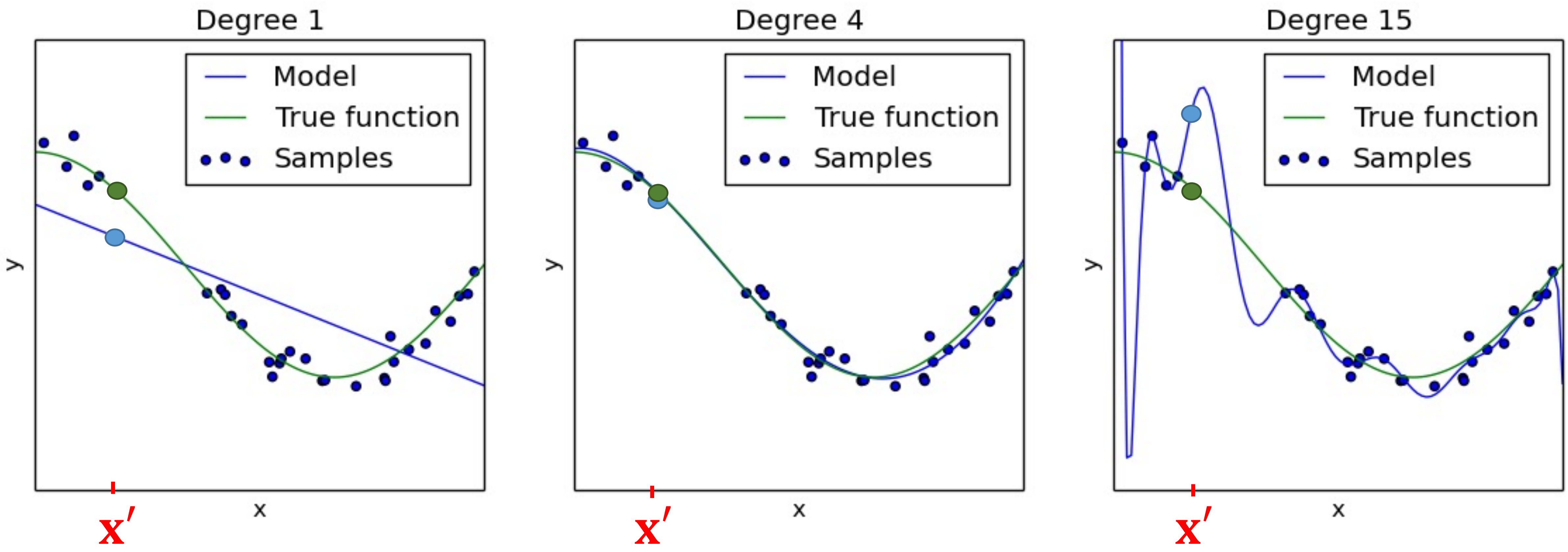
Output: a function $f: \mathbb{R}^d \mapsto \mathbb{R}$ such that $f(\mathbf{x}_i) \approx y_i$.

Test:

Input: a *never-seen-before* feature vectors $\mathbf{x}' \in \mathbb{R}^d$.

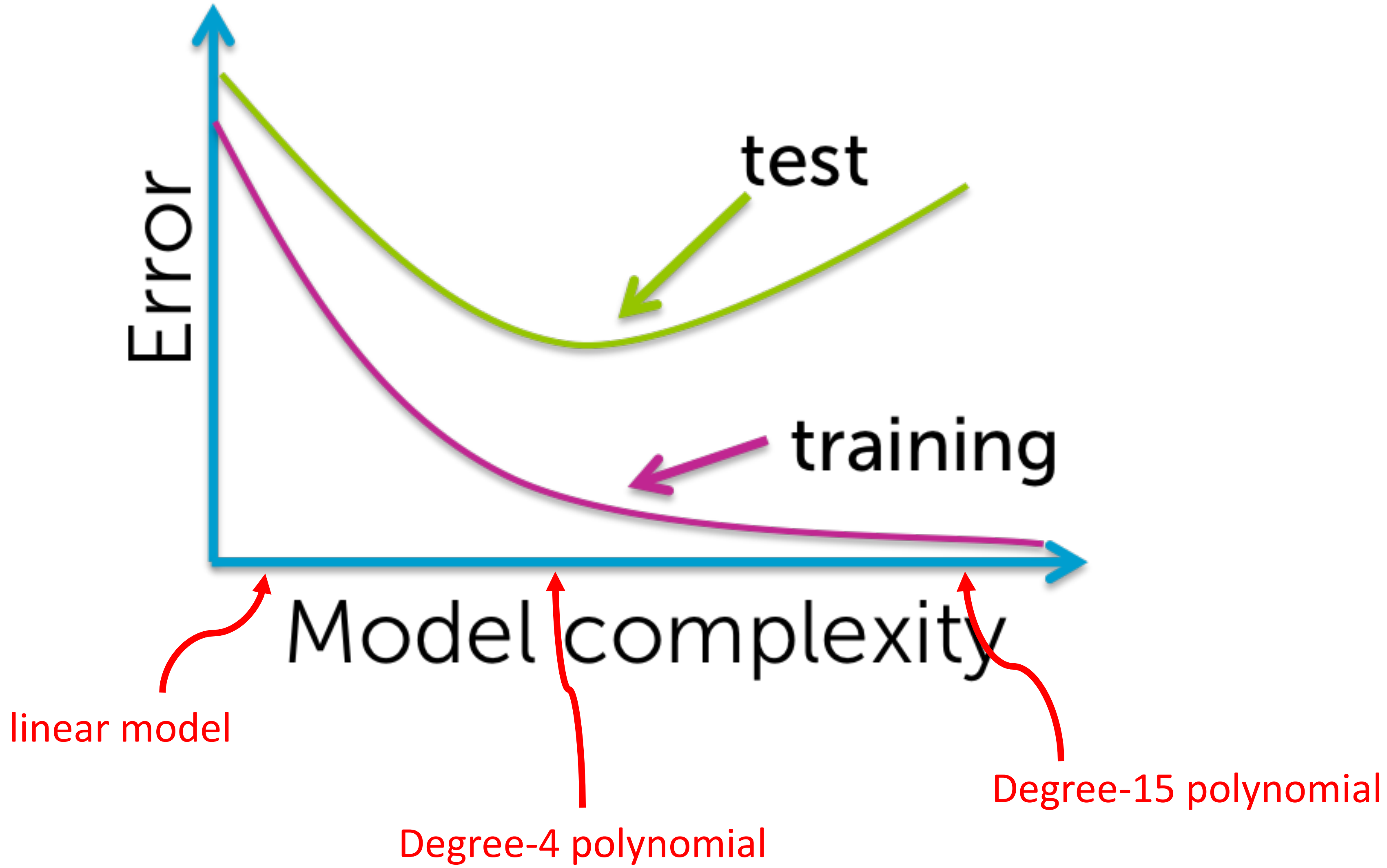
output: predict its label by $f(\mathbf{x}')$.

Training and Testing



BAD **GOOD** **BAD**

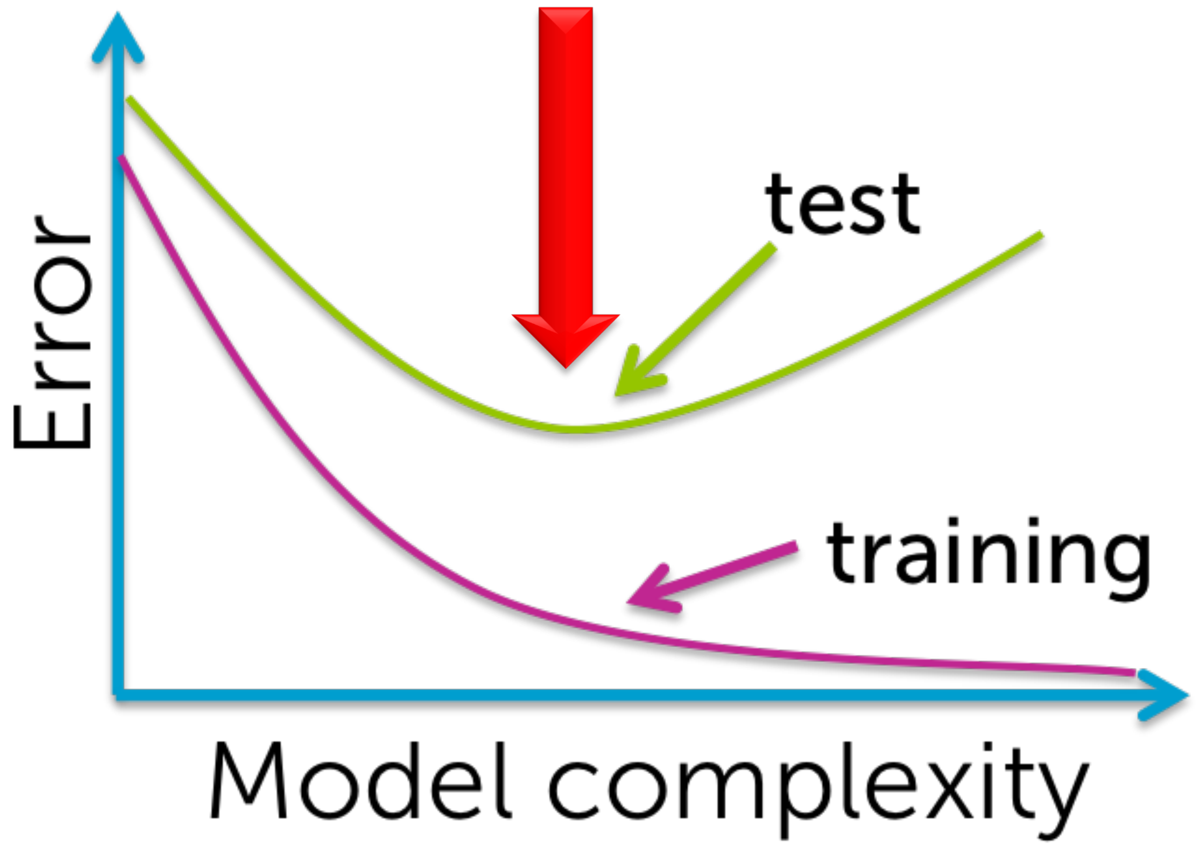
Training and Testing



Hyper-Parameter Tuning

Question: for the polynomial regression model, how to determine the degree p?

Answer: the degree p leads to the smallest test error.



Hyper-Parameter Tuning

Training Set	Test Set
Train a degree-1 polynomial regression	Test MSE = 23.2
Train a degree-2 polynomial regression	Test MSE = 19.0
Train a degree-3 polynomial regression	Test MSE = 16.7
Train a degree-4 polynomial regression	Test MSE = 12.2
Train a degree-5 polynomial regression	Test MSE = 14.8
Train a degree-6 polynomial regression	Test MSE = 25.1
Train a degree-7 polynomial regression	Test MSE = 39.4
Train a degree-8 polynomial regression	Test MSE = 53.0

Hyper-Parameter Tuning

Training Set		Test Set
Train a degree-1 polynomial regression	→	Test MSE = 23.2
Train a degree-2 polynomial regression	→	Test MSE = 19.0
Train a degree-3 polynomial regression	→	Test MSE = 16.7
Train a $\boxed{\text{degree-4 polynomial}}$ regression	→	$\boxed{\text{Test MSE = 12.2}}$
Train a degree-5 polynomial regression	→	Test MSE = 14.8
Train a degree-6 polynomial regression	→	Test MSE = 25.1
Train a degree-7 polynomial regression	→	Test MSE = 39.4
Train a degree-8 polynomial regression	→	Test MSE = 53.0

Hyper-Parameter Tuning

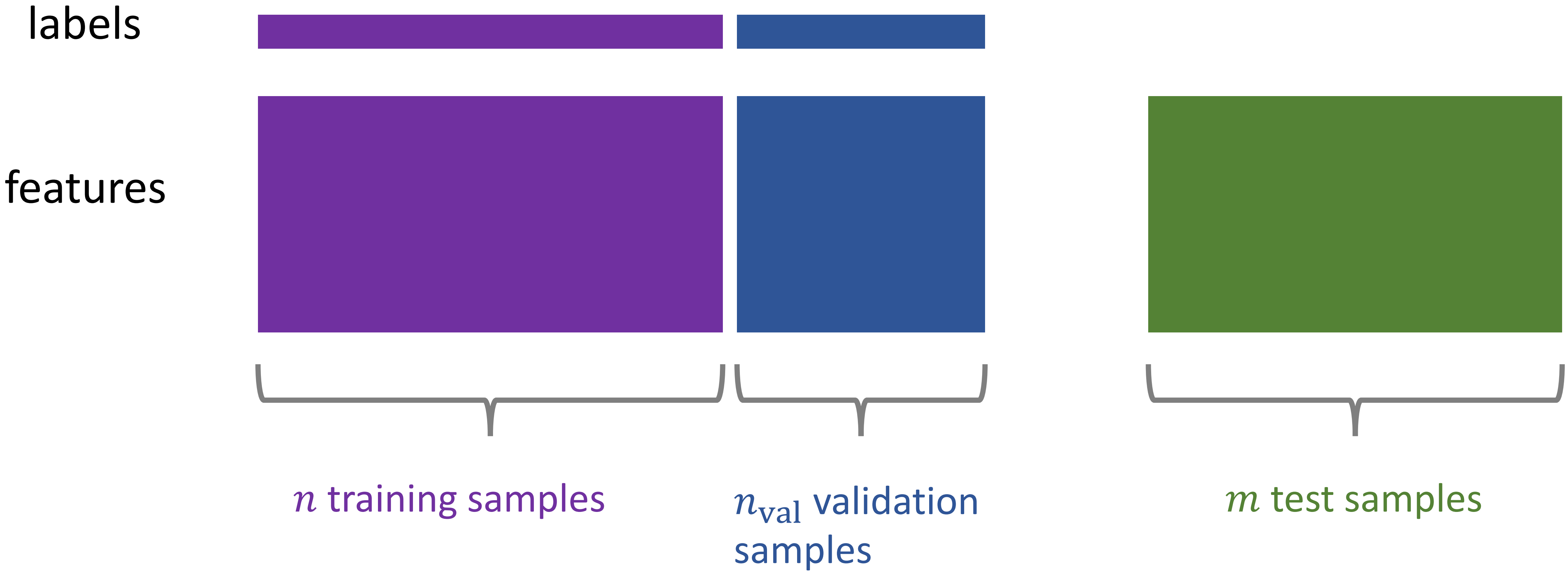
Training Set		Test Set
Train a degree-1 polynomial regression	→	Test MSE = 23.2
Train a degree-2 polynomial regression	→	Test MSE = 19.0
Train a degree-3 polynomial regression	→	Test MSE = 16.7
Train a degree-4 polynomial regression	→	Test MSE = 12.2
Train a degree-5 polynomial regression	→	Test MSE = 14.8
Train a degree-6 polynomial regression	→	Test MSE = 25.1
Train a degree-7 polynomial regression	→	Test MSE = 39.4
Train a degree-8 polynomial regression	→	Test MSE = 53.0

- **Wrong! The test labels are unavailable!**
- **Even if you have the test labels, never do this!**

Cross-Validation (Naïve Approach) for Hyper-Parameter Tuning

Cross-Validation (Naïve Approach)

Cross-Validation (Naïve Approach)



Cross-Validation (Naïve Approach)

Training Set		Test Set
Train a degree-1 polynomial regression	→	Test MSE = 23.2
Train a degree-2 polynomial regression	→	Test MSE = 19.0
Train a degree-3 polynomial regression	→	Test MSE = 16.7
Train a degree-4 polynomial regression	→	Test MSE = 12.2
Train a degree-5 polynomial regression	→	Test MSE = 14.8
Train a degree-6 polynomial regression	→	Test MSE = 25.1
Train a degree-7 polynomial regression	→	Test MSE = 39.4
Train a degree-8 polynomial regression	→	Test MSE = 53.0

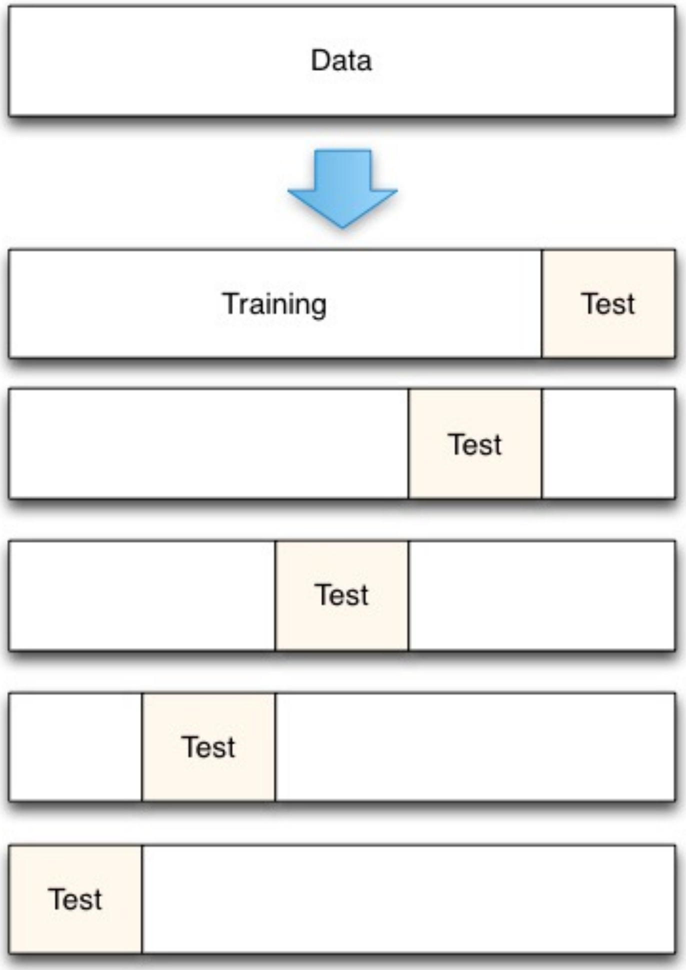
Cross-Validation (Naïve Approach)

Training Set		Validation Set ~~Test Set~~
Train a degree-1 polynomial regression	→	Valid. MSE = 23.1
Train a degree-2 polynomial regression	→	Valid. MSE = 19.2
Train a degree-3 polynomial regression	→	Valid. MSE = 16.3
Train a degree-4 polynomial regression	→	$\boxed{\text{Valid. MSE = 12.5}}$
Train a degree-5 polynomial regression	→	Valid. MSE = 14.4
Train a degree-6 polynomial regression	→	Valid. MSE = 25.0
Train a degree-7 polynomial regression	→	Valid. MSE = 39.1
Train a degree-8 polynomial regression	→	Valid. MSE = 53.5

k-Fold Cross-Validation

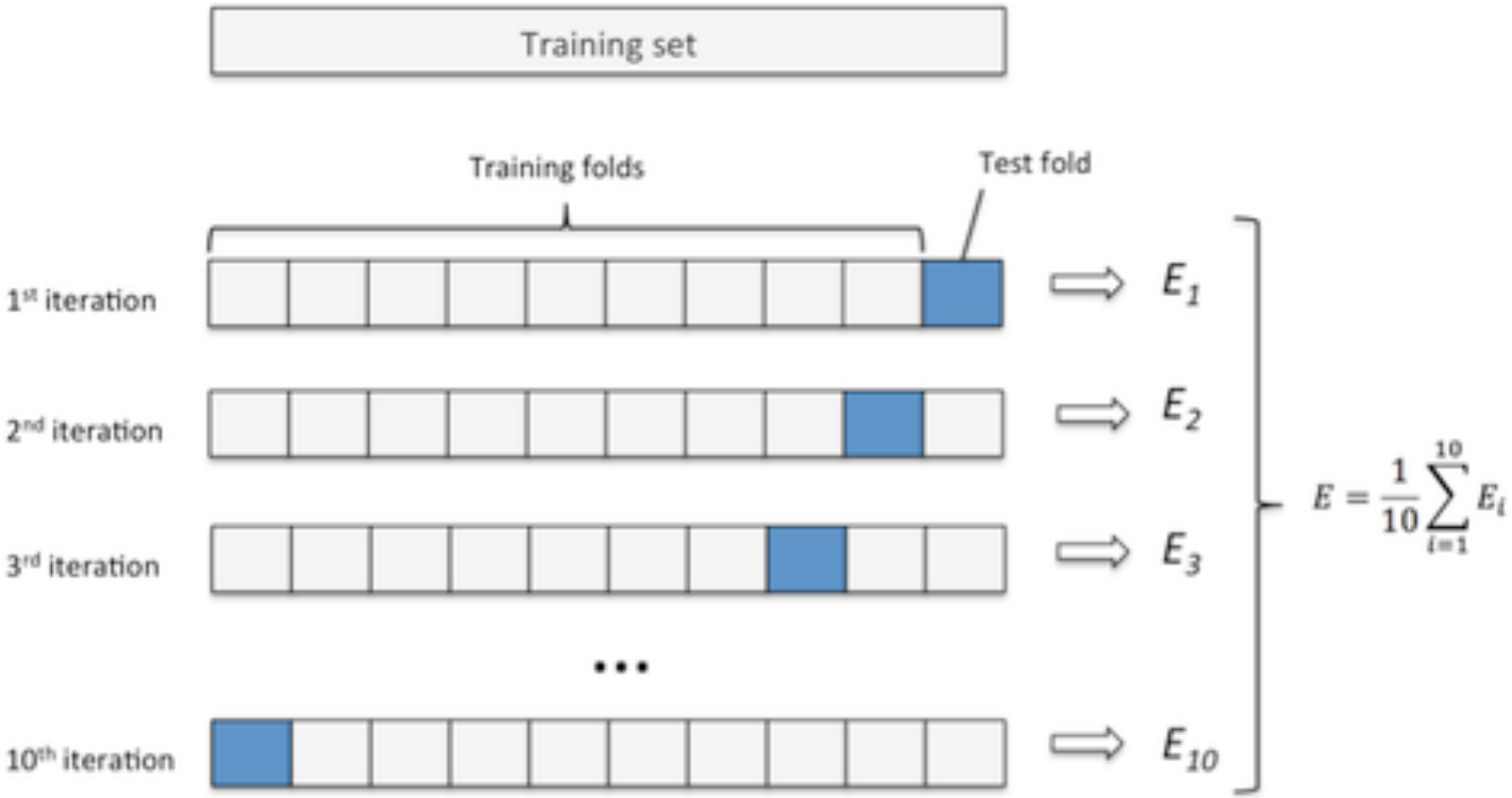
k-Fold Cross-Validation

1. Propose a grid of hyper-parameters.
   - E.g. $p \in \{1, 2, 3, 4, 5, 6\}$.
2. Randomly partition the training samples to k parts.
   - $k - 1$ parts for training.
   - One part for test.
3. Compute the averaged test errors of the k repeats.
   - The average is called the validation error.
4. Choose the hyper-parameter p that leads to the smallest validation error.



Example: 5-fold cross-validation

Example: 10-Fold Cross-Validation



validation error

Example: 10-Fold Cross-Validation

hyper-parameter	validation error
p=1	23.19
p=2	21.00
p=3	18.54
p=4	24.36
p=5	27.96
p=6	33.10

Real-World Machine Learning Competition

The Available Data

	Training	Public	Private
Labels:	y	unknown	unknown
Features:	X	X_{public}	$X_{private}$

$$\underbrace{X_{public} \quad X_{private}}_{\text{Test Data}}$$

The public and private are mixed;
Participants cannot distinguish them.

Train A Model

	Training	Public	Private
Labels:	$\mathbf{y}$	unknown	unknown
Features:	$\mathbf{X}$	$\mathbf{X}_{\text{public}}$	$\mathbf{X}_{\text{private}}$

Model

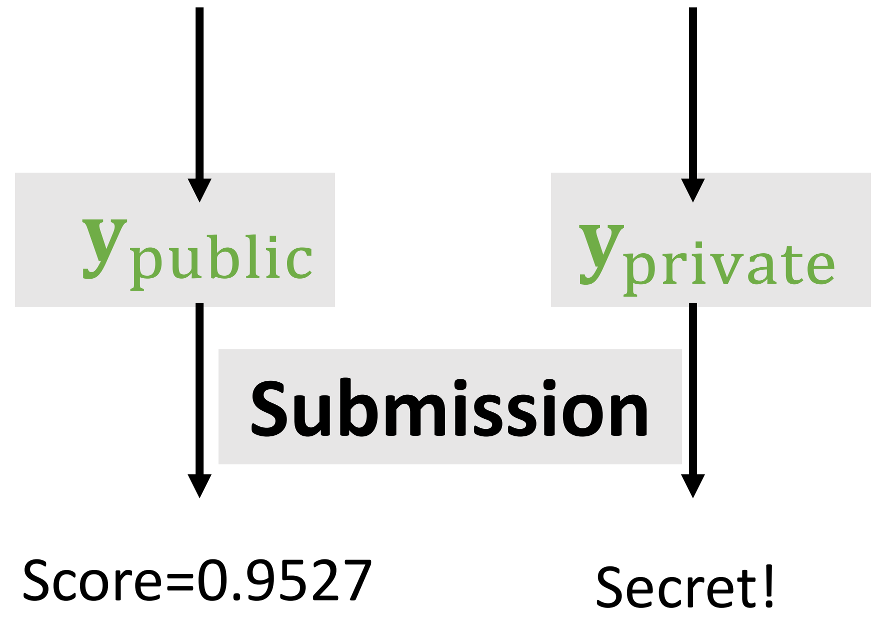
Prediction

	Training	Public	Private
Labels:	y	unknown	unknown
Features:	X	X_{public}	$X_{private}$
		y_{public}	$y_{private}$

Model

Submission to Leaderboard

	Training	Public	Private
Labels:	y	unknown	unknown
Features:	X	X_{public}	$X_{private}$



Submission to Leaderboard

	Training	Public	Private
Labels:	y	unknown	unknown
Features:	X	X_{public}	$X_{private}$
		y_{public}	$y_{private}$

Submission

Score=0.9527

Secret!

Question: Why two leaderboards?

Answer: The score can be evilly used for hyper-parameter tuning (cheating).

Summary

- Polynomial regression for non-linear problems.
- Polynomial regression has a hyper-parameter p.
- Underfitting (very small p) and overfitting (very big p) .
- Tune the hyper-parameters using cross-validation.
- Make your model parameters and hyper-parameters independent of the test set!!!

Binary Classification

(Logistic Regression)

Vector and Matrix Derivatives

Derivative of Scalar w.r.t. Scalar

Examples:

- $y = x^2; \;\; \frac{dy}{dx} = 2x.$
- $y = e^x; \;\; \frac{dy}{dx} = e^x.$

Derivative of Vector w.r.t. Scalar

- The derivative of a vector $\mathbf{y} \in \mathbb{R}^n$ w.r.t. a scalar $x \in \mathbb{R}$:

$$\frac{\partial\,\mathbf{y}}{\partial\,x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_n}{\partial x} \end{bmatrix}$$

- Example:

$$\mathbf{y} = \begin{bmatrix} 3x^2 \\ x+1 \\ \log x \\ e^x \end{bmatrix}, \qquad \frac{\partial\,\mathbf{y}}{\partial\,x} = \begin{bmatrix} 6x \\ 1 \\ 1/x \\ e^x \end{bmatrix}$$

Derivative of Scalar w.r.t. Vector

- The derivative of a scalar $y \in \mathbb{R}$ w.r.t. a vector $\mathbf{x} \in \mathbb{R}^m$:

$$\frac{\partial\, y}{\partial\, \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_m} \end{bmatrix}$$

- Example 1:

$$y = \|\mathbf{x}\|_2^2 = \sum_{i=1}^{m} x_i^2, \qquad \frac{\partial\, y}{\partial\, \mathbf{x}} = 2\mathbf{x}.$$

Derivative of Scalar w.r.t. Vector

- The derivative of a scalar $y \in \mathbb{R}$ w.r.t. a vector $\mathbf{x} \in \mathbb{R}^m$:

$$\frac{\partial\, y}{\partial\, \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_m} \end{bmatrix}$$

- Example 2:

$$y = \mathbf{x}^T \mathbf{z} = \sum_{i=1}^{m} x_i z_i, \qquad \frac{\partial\, y}{\partial\, \mathbf{x}} = \mathbf{z}.$$

Derivative of Scalar w.r.t. Vector

- The derivative of a scalar $y \in \mathbb{R}$ w.r.t. a vector $\mathbf{x} \in \mathbb{R}^m$:

$$\frac{\partial\, y}{\partial\, \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_m} \end{bmatrix}$$

- Example 3:

$$y = \sum_{i=1}^{m} \log(1 + e^{-x_i}), \qquad \frac{\partial\, y}{\partial\, \mathbf{x}} = \begin{bmatrix} \frac{\partial \log(1+e^{-x_1})}{\partial x_1} \\ \vdots \\ \frac{\partial \log(1+e^{-x_m})}{\partial x_m} \end{bmatrix} = \begin{bmatrix} -\frac{1}{1+e^{x_1}} \\ \vdots \\ -\frac{1}{1+e^{x_m}} \end{bmatrix}$$

Derivative of Vector w.r.t. Vector

- The derivative of a vector $\mathbf{y} \in \mathbb{R}^n$ w.r.t. a vector $\mathbf{x} \in \mathbb{R}^m$:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_m} & \frac{\partial y_2}{\partial x_m} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

$m \times n$ matrix

The (i, j)-th entry is $\frac{\partial y_j}{\partial x_i}$

- Example 1:

$$\frac{\partial \mathbf{x}}{\partial \mathbf{x}} = \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}}_{m \times m}$$

Derivative of Vector w.r.t. Vector

- The derivative of a vector $\mathbf{y} \in \mathbb{R}^n$ w.r.t. a vector $\mathbf{x} \in \mathbb{R}^m$:

$$\frac{\partial\, \mathbf{y}}{\partial\, \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_m} & \frac{\partial y_2}{\partial x_m} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

$m \times n$ matrix

- Example 2:

$$\mathbf{y} = \begin{bmatrix} a_1 x_1^2 \\ a_2 x_2^2 \\ \vdots \\ a_m x_m^2 \end{bmatrix} \in \mathbb{R}^m, \qquad \frac{\partial\, \mathbf{y}}{\partial\, \mathbf{x}} = \underbrace{\begin{bmatrix} 2a_1x_1 & 0 & \cdots & 0 \\ 0 & 2a_2x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2a_mx_m \end{bmatrix}}_{m \times m}$$

Derivative of Vector w.r.t. Vector

- The derivative of a vector $\mathbf{y} \in \mathbb{R}^n$ w.r.t. a vector $\mathbf{x} \in \mathbb{R}^m$:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_m} & \frac{\partial y_2}{\partial x_m} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

$m \times n$ matrix

- Example 3:

$$\mathbf{A} \in \mathbb{R}^{n \times m}, \qquad \mathbf{y} = \mathbf{A}\mathbf{x} \in \mathbb{R}^n, \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{A}^T \in \mathbb{R}^{m \times n}$$

Chain Rule

- Let $\mathbf{z} \in \mathbb{R}^{n_z}$ be a function of $\mathbf{y} \in \mathbb{R}^{n_y}$ and $\mathbf{y}$ be a function of $\mathbf{x} \in \mathbb{R}^{n_x}$.

$$\mathbf{x} \in \mathbb{R}^{n_x} \longrightarrow \mathbf{y} \in \mathbb{R}^{n_y} \longrightarrow \mathbf{z} \in \mathbb{R}^{n_z}$$

$$\underbrace{\frac{d\mathbf{z}}{d\mathbf{x}}}_{n_x \times n_z} = \underbrace{\frac{d\mathbf{y}}{d\mathbf{x}}}_{n_x \times n_y} \underbrace{\frac{d\mathbf{z}}{d\mathbf{y}}}_{n_y \times n_z}$$

Derivative of Scalar w.r.t. Matrix

- The derivative of a scalar $y \in \mathbb{R}$ w.r.t. a matrix $\mathbf{Z} \in \mathbb{R}^{p \times q}$:

1. Vectorization: $\mathbf{x} = \mathrm{vec}(\mathbf{Z}) \in \mathbb{R}^{pq \times 1}$.
2. Compute $\frac{\partial\, y}{\partial\, \mathbf{x}} \in \mathbb{R}^{pq \times 1}$.
3. Reshape the resulting $pq \times 1$ vector to $p \times q$ matrix.

Derivative of Vector w.r.t. Matrix

- The derivative of a vector $\mathbf{y} \in \mathbb{R}^{n}$ w.r.t. a matrix $\mathbf{Z} \in \mathbb{R}^{p\times q}$:
  1. Vectorization: $\mathbf{x} = \mathrm{vec}(\mathbf{Z}) \in \mathbb{R}^{pq\times 1}$.
  2. Compute $\frac{\partial\, \mathbf{y}}{\partial\, \mathbf{x}} \in \mathbb{R}^{pq\times n}$.
  3. Reshape the resulting $pq\times n$ matrix to $p\times q\times n$ tensor.

Binary Classification

Tasks

Methods

Algorithms

Binary Classification

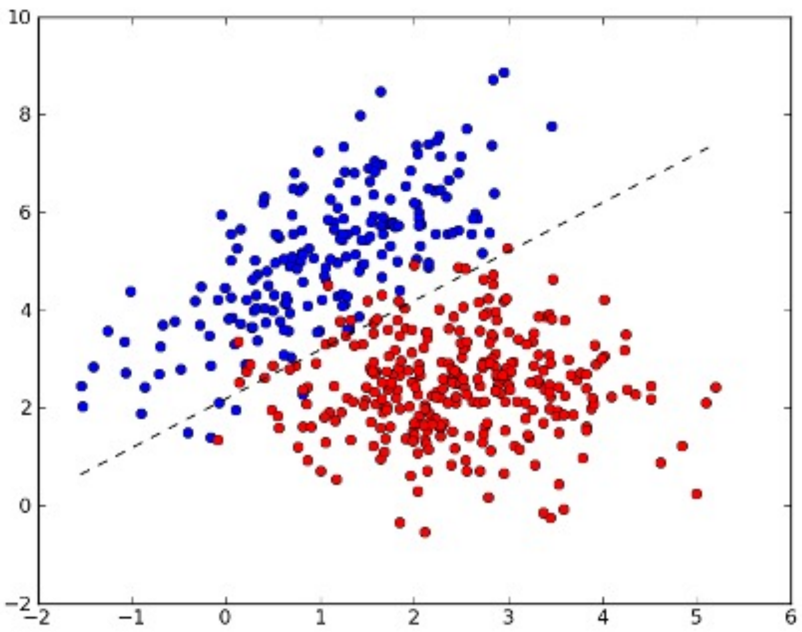
Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \{-1, +1\}$.

Output: a function $f: \mathbb{R}^d \mapsto \{-1, +1\}$.

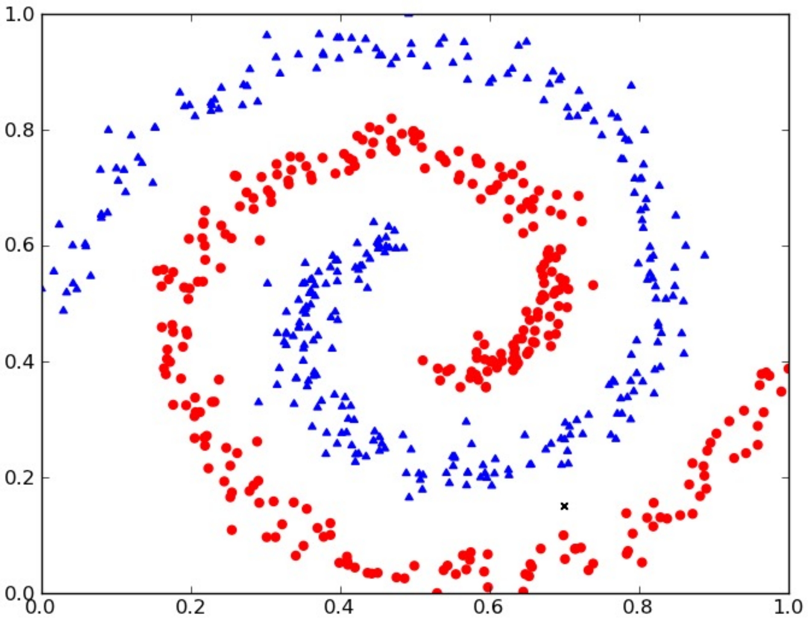
Binary Classification

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \{-1, +1\}$.

Output: a function $f: \mathbb{R}^d \mapsto \{-1, +1\}$.



Linear Classification



Nonlinear Classification

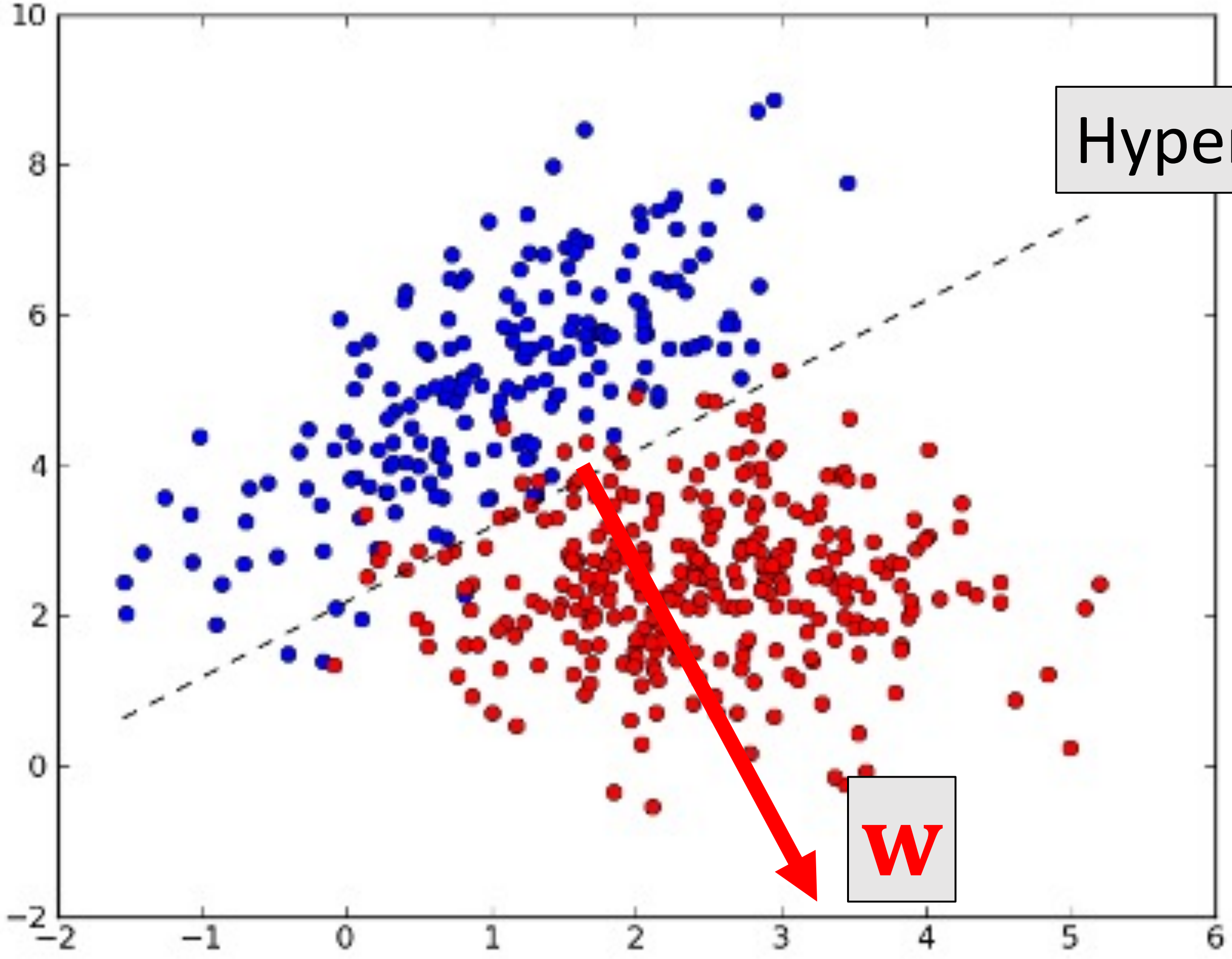
Logistic Regression (Linear Classifier)

Tasks

Methods

Algorithms

Linear Classifier



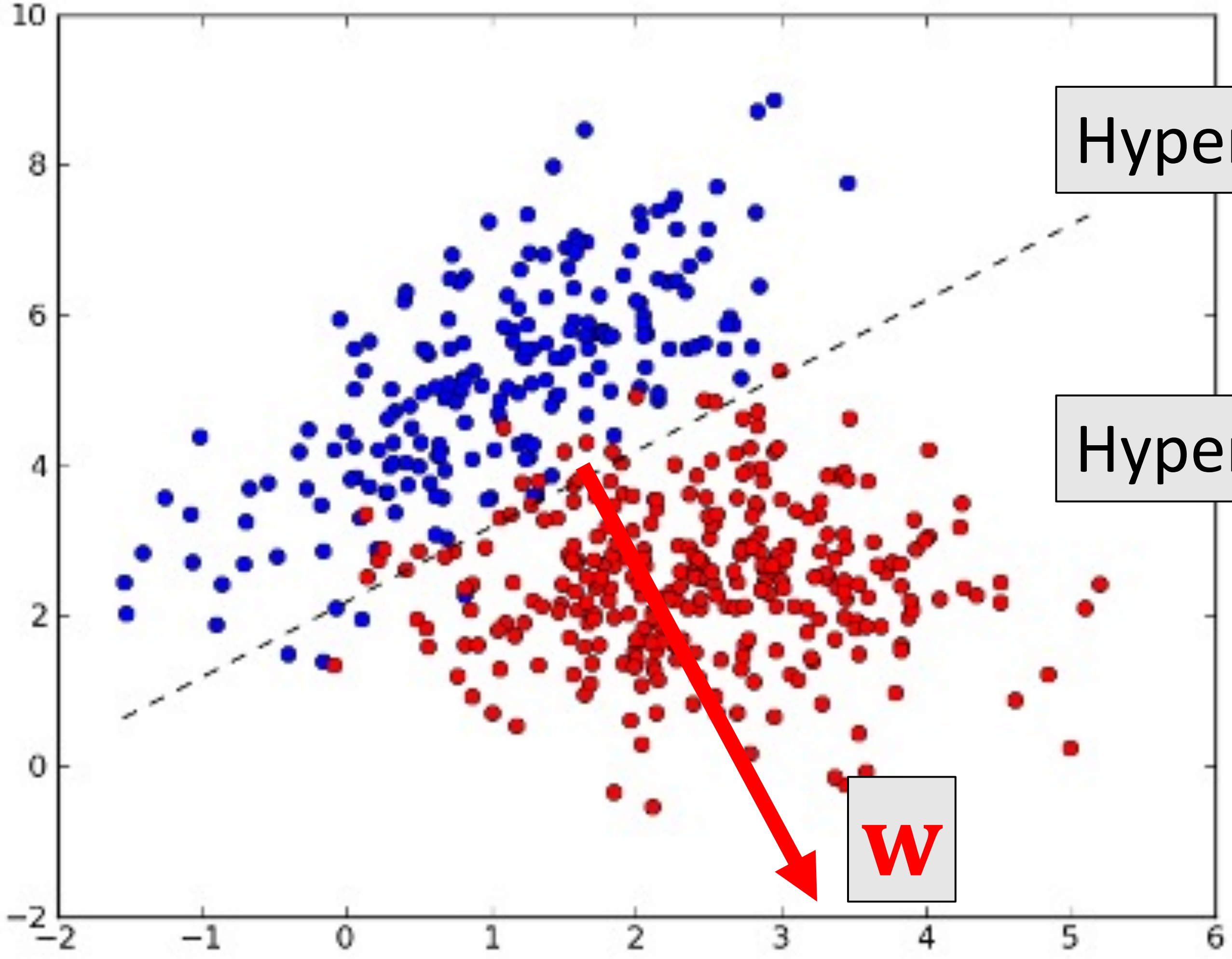
Hyperplane: $\mathbf{w}^T\mathbf{x} + b = 0$

Define $\bar{\mathbf{x}}_j = [\mathbf{x}_j; 1] \in \mathbb{R}^{d+1}$

Define $\bar{\mathbf{w}} = [\mathbf{w};\ b] \in \mathbb{R}^{d+1}$

➔ $\mathbf{x}_j^T\mathbf{w} + b = \bar{\mathbf{x}}_j^T\bar{\mathbf{w}}$

Linear Classifier

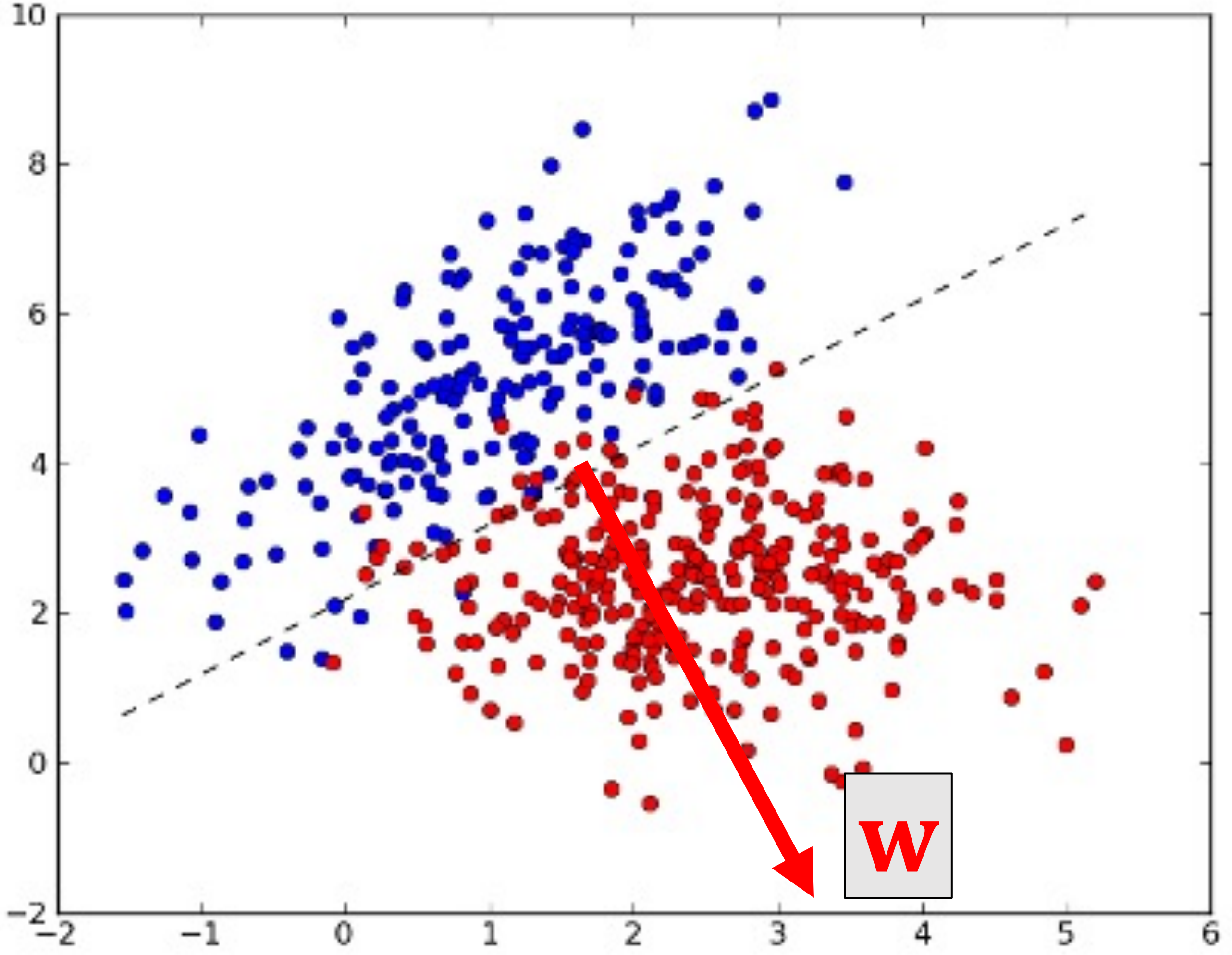


Hyperplane: $\mathbf{w}^T\mathbf{x} + b = 0$

Hyperplane: $\bar{\mathbf{w}}^T\bar{\mathbf{x}} = 0$

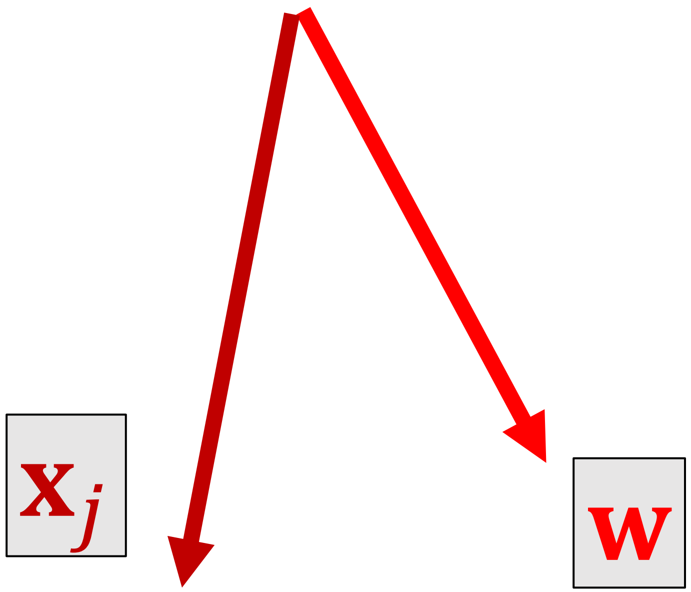
(The intercept is removed.)

Linear Classifier

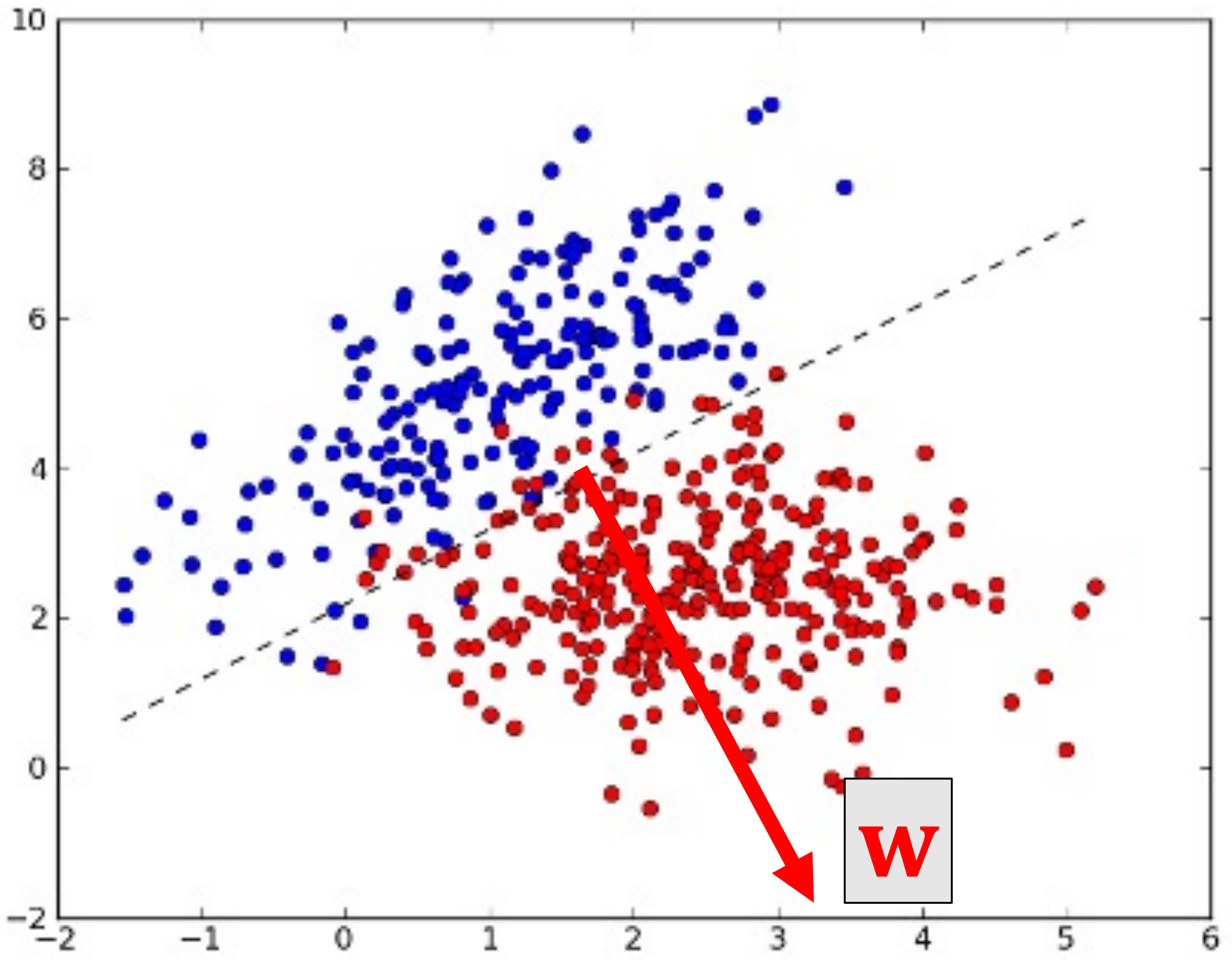


Learn a vector **w** such that

- If $y_j = +1$, then $\mathbf{w}^T \mathbf{x}_j > 0$.

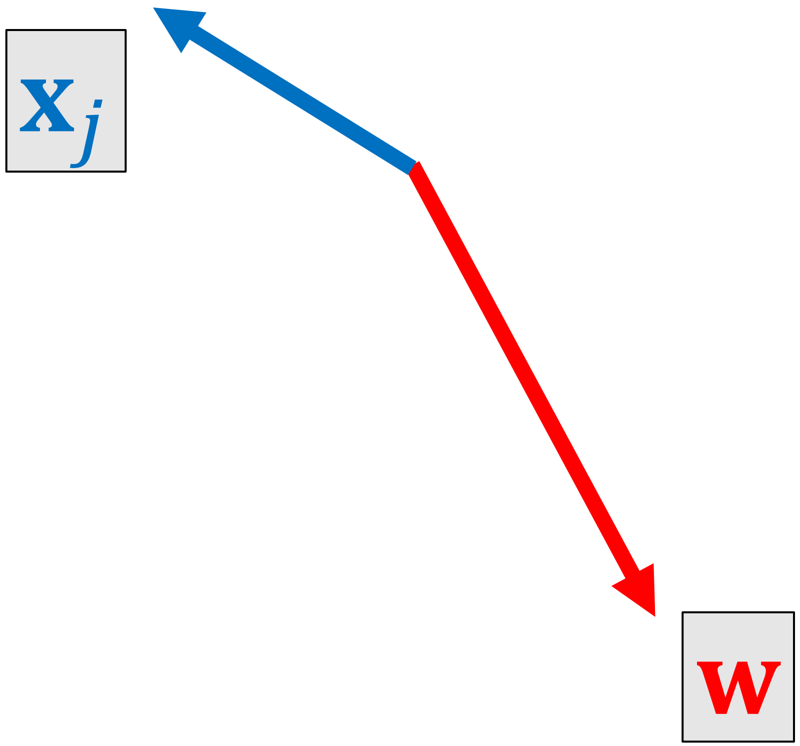


Linear Classifier

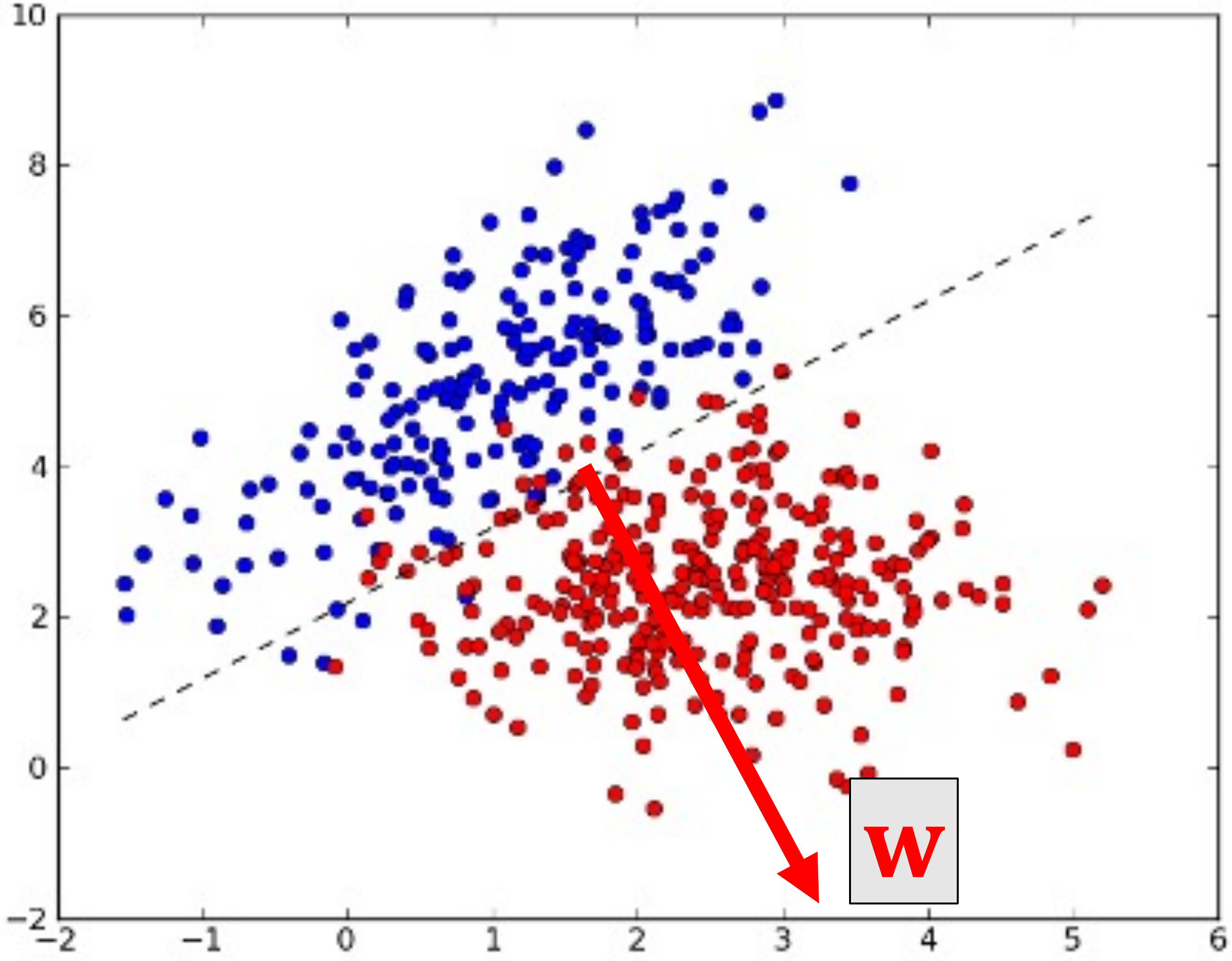


Learn a vector $\mathbf{w}$ such that

- If $y_j = +1$, then $\mathbf{w}^T \mathbf{x}_j > 0$.
- If $y_j = -1$, then $\mathbf{w}^T \mathbf{x}_j < 0$.



Linear Classifier



Learn a vector $\mathbf{w}$ such that

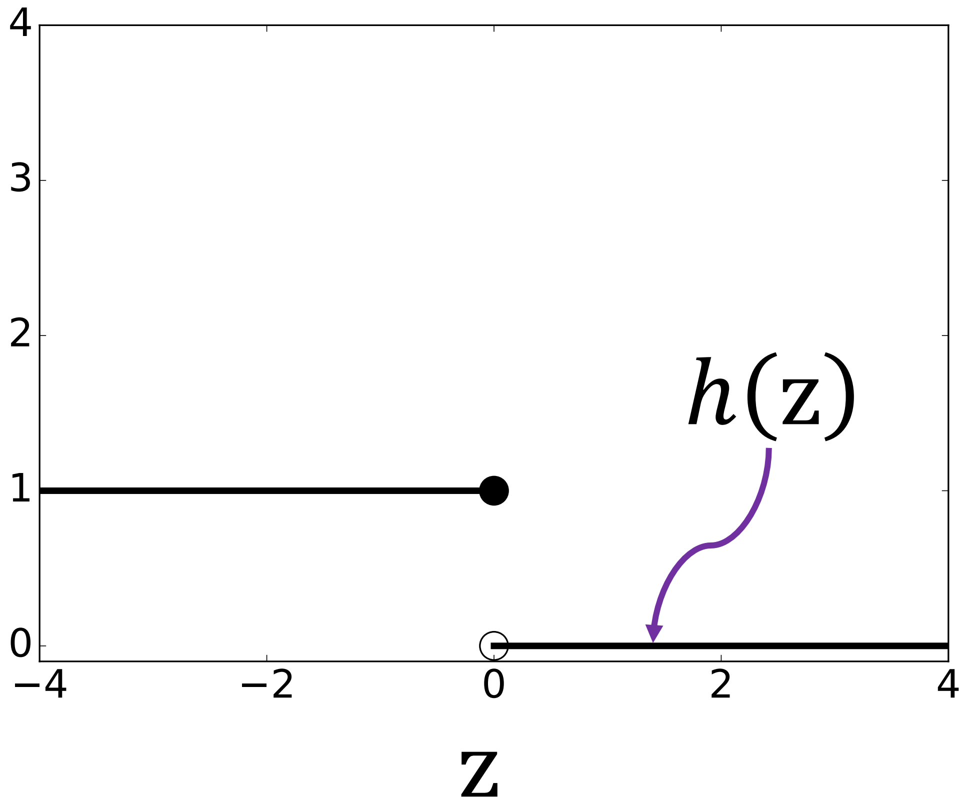
- If $y_j = +1$, then $\mathbf{w}^T\mathbf{x}_j > 0.$
- If $y_j = -1$, then $\mathbf{w}^T\mathbf{x}_j < 0.$

Key Idea:

Encourage $y_j\mathbf{w}^T\mathbf{x}_j$ to be positive

Directly Minimize the Classification Error?

Minimize $\sum_j h(y_j \mathbf{w}^T \mathbf{x}_j)$, where $h(\mathrm{z}) = \begin{cases} 1, & \text{if } z < 0; \\ 0, & \text{if } z \geq 0. \end{cases}$



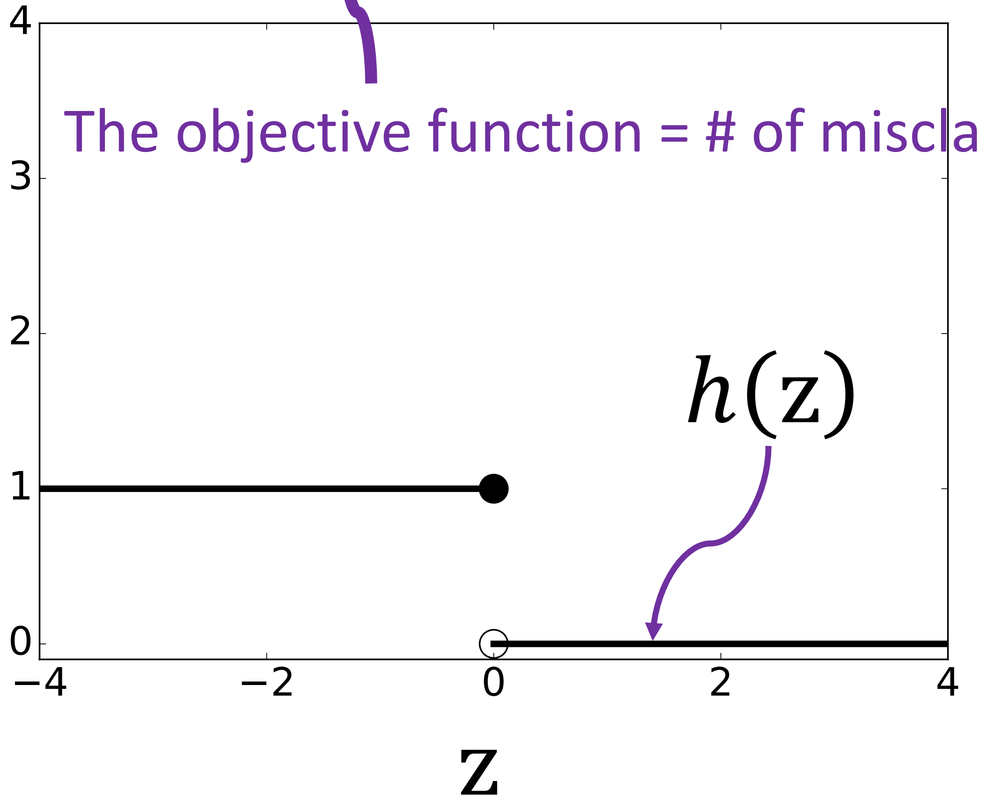
Key Idea:

Encourage $y_j \mathbf{w}^T \mathbf{x}_j$ to be positive

Directly Minimize the Classification Error?

Minimize $\sum_j h(y_j \mathbf{w}^T \mathbf{x}_j)$, where $h(z) = \begin{cases} 1, & \text{if } z < 0; \\ 0, & \text{if } z \geq 0. \end{cases}$

The objective function = # of misclassified training samples



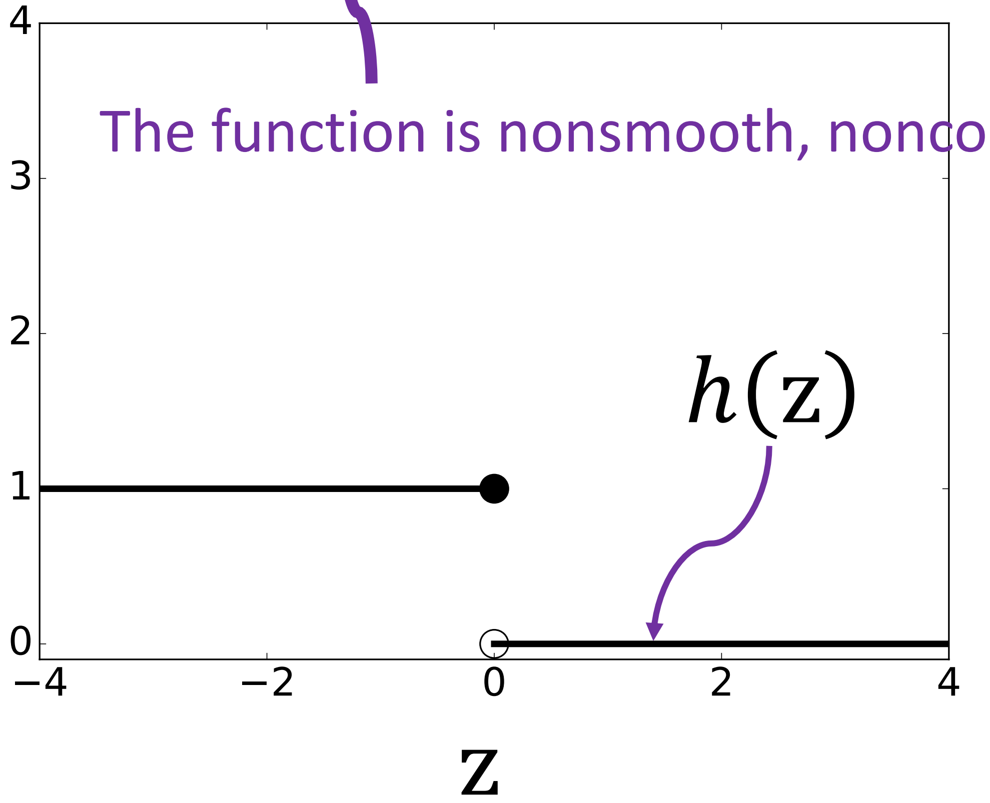
Key Idea:

Encourage $y_j \mathbf{w}^T \mathbf{x}_j$ to be positive

Directly Minimize the Classification Error?

Minimize $\sum_j h(y_j \mathbf{w}^T \mathbf{x}_j)$, where $h(\mathrm{z}) = \begin{cases} 1, & \text{if } z < 0; \\ 0, & \text{if } z \geq 0. \end{cases}$

The function is nonsmooth, nonconvex, and hard to optimize.

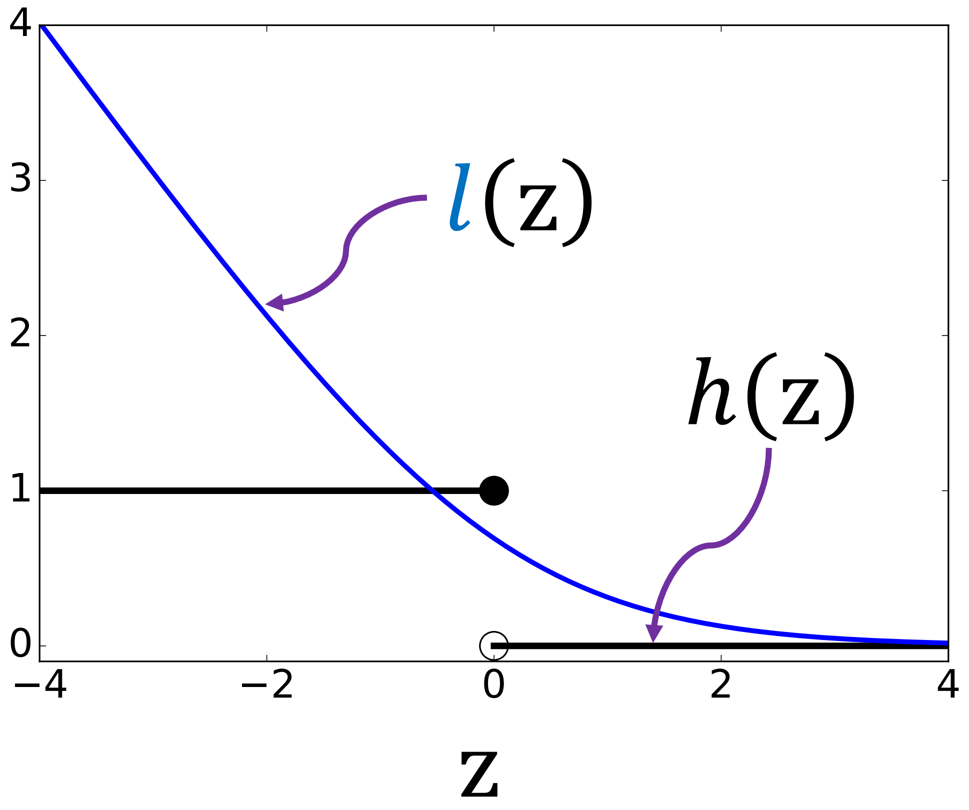


Key Idea:

Encourage $y_j \mathbf{w}^T \mathbf{x}_j$ to be positive

Logistic Regression

Minimize $\sum_j l(y_j \mathbf{w}^T \mathbf{x}_j)$, where $l(\mathrm{z}) = \log(1 + e^{-z})$.



Key Idea:

Encourage $y_j \mathbf{w}^T \mathbf{x}_j$ to be positive

Logistic Regression

Tasks

Methods

Algorithms

Logistic Regression

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(z) = \log(1 + e^{-z})$.

Tasks

Binary Classification

Multi-Class Classification

Methods

Logistic Regression

SVM

Neural Networks

Algorithms

Gradient Descent (GD)

Accelerated GD

Stochastic GD

Gradient

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n} \sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(\mathrm{z}) = \log(1 + e^{-z})$.

$$\mathbf{w} \longrightarrow z = y\, \mathbf{w}^T \mathbf{x} \longrightarrow l = \log(1 + e^{-z})$$

Gradient

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(z) = \log(1 + e^{-z})$.

$\mathbf{w}$ $\quad z = y\,\mathbf{w}^T\mathbf{x}$ $\quad l = \log(1 + e^{-z})$

- $\frac{\partial\, z}{\partial\, \mathbf{w}} = y\mathbf{x},$

Gradient

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(\mathrm{z}) = \log(1 + e^{-z})$.

$\mathbf{w}$ $\qquad$ $z = y\, \mathbf{w}^T \mathbf{x}$ $\qquad$ $l = \log\left(1 + e^{-z}\right)$

- $\frac{\partial\, z}{\partial\, \mathbf{w}} = y\mathbf{x}$, $\qquad \frac{\partial\, l(z)}{\partial\, z} = \frac{-e^{-z}}{1+e^{-z}} = -\frac{1}{1+e^{z}}$.

Gradient

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n} \sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(z) = \log(1 + e^{-z})$.

$$\mathbf{w} \longrightarrow z = y\, \mathbf{w}^T \mathbf{x} \longrightarrow l = \log(1 + e^{-z})$$

- $\frac{\partial\, z}{\partial\, \mathbf{w}} = y\mathbf{x}$, $\qquad \frac{\partial\, l(z)}{\partial\, z} = \frac{-e^{-z}}{1+e^{-z}} = -\frac{1}{1+e^{z}}$.
- Chain rule: $\frac{\partial\, l}{\partial\, \mathbf{w}} = \frac{\partial\, z}{\partial\, \mathbf{w}} \cdot \frac{\partial\, l}{\partial\, z} = (y\mathbf{x}) \left(-\frac{1}{1+e^{z}}\right) = -\frac{y\mathbf{x}}{1+\exp(y\mathbf{w}^T\mathbf{x})}$.

Gradient

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n} \sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(\mathrm{z}) = \log(1 + e^{-z})$.

- We have shown: $\frac{\partial\, l(y\mathbf{w}^T\mathbf{x})}{\partial\, \mathbf{w}} = \frac{-y\mathbf{x}}{1+\exp(y\mathbf{w}^T\mathbf{x})}$.
- Objective function: $f(\mathbf{w}) = \frac{1}{n} \sum_j l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$.

Gradient

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(\mathrm{z}) = \log(1 + e^{-z})$.

- We have shown: $\dfrac{\partial\, l(y\mathbf{w}^T\mathbf{x})}{\partial\, \mathbf{w}} = \dfrac{-y\mathbf{x}}{1+\exp(y\mathbf{w}^T\mathbf{x})}$.
- Objective function: $f(\mathbf{w}) = \frac{1}{n}\sum_j l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$.
- $\dfrac{\partial f(\mathbf{w})}{\partial\, \mathbf{w}} = \frac{1}{n}\sum_j \dfrac{\partial\, l(y_j\mathbf{w}^T\mathbf{x}_j)}{\partial\, \mathbf{w}} = \frac{1}{n}\sum_j \dfrac{-y_j\, \mathbf{x}_j}{1+\exp(y_j\mathbf{w}^T\mathbf{x}_j)}$.

Gradient

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(\mathrm{z}) = \log(1+e^{-z})$.

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n}\sum_{j=1}^{n} \frac{-y_j\,\mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

- We have shown: $\frac{\partial\, l(y\mathbf{w}^T\mathbf{x})}{\partial\, \mathbf{w}} = \frac{-y\mathbf{x}}{1+\exp(y\mathbf{w}^T\mathbf{x})}$.
- Objective function: $f(\mathbf{w}) = \frac{1}{n}\sum_j l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$.
- $\frac{\partial f(\mathbf{w})}{\partial\, \mathbf{w}} = \frac{1}{n}\sum_j \frac{\partial\, l(y_j \mathbf{w}^T \mathbf{x}_j)}{\partial\, \mathbf{w}} = \frac{1}{n}\sum_j \frac{-y_j\,\mathbf{x}_j}{1+\exp(y_j \mathbf{w}^T \mathbf{x}_j)}$.

Gradient Descent (GD) Algorithm

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n} \sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(z) = \log(1 + e^{-z})$.

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n} \sum_{j=1}^{n} \frac{-y_j \mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

GD repeat:

1. Compute gradient: $\mathbf{g}_t$
2. Update: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \, \mathbf{g}_t$

Tune the step size (learning rate) α

Algorithms

- Gradient Descent (GD)
- Accelerated GD
- Stochastic GD

AGD Algorithm

Logistic regression: $\min_{\mathbf{w}} \frac{1}{n}\sum_{j=1}^{n} l\left(y_j \mathbf{w}^T \mathbf{x}_j\right)$, where $l(\mathrm{z}) = \log(1 + e^{-z})$.

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n}\sum_{j=1}^{n} \frac{-y_j \mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

AGD repeat:

1. Compute gradient: $\mathbf{g}_t$
2. Update momentum: $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \mathbf{g}_t$
3. Update: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\, \mathbf{v}_{t+1}$

Tune α and β $(0 \le \beta < 1)$

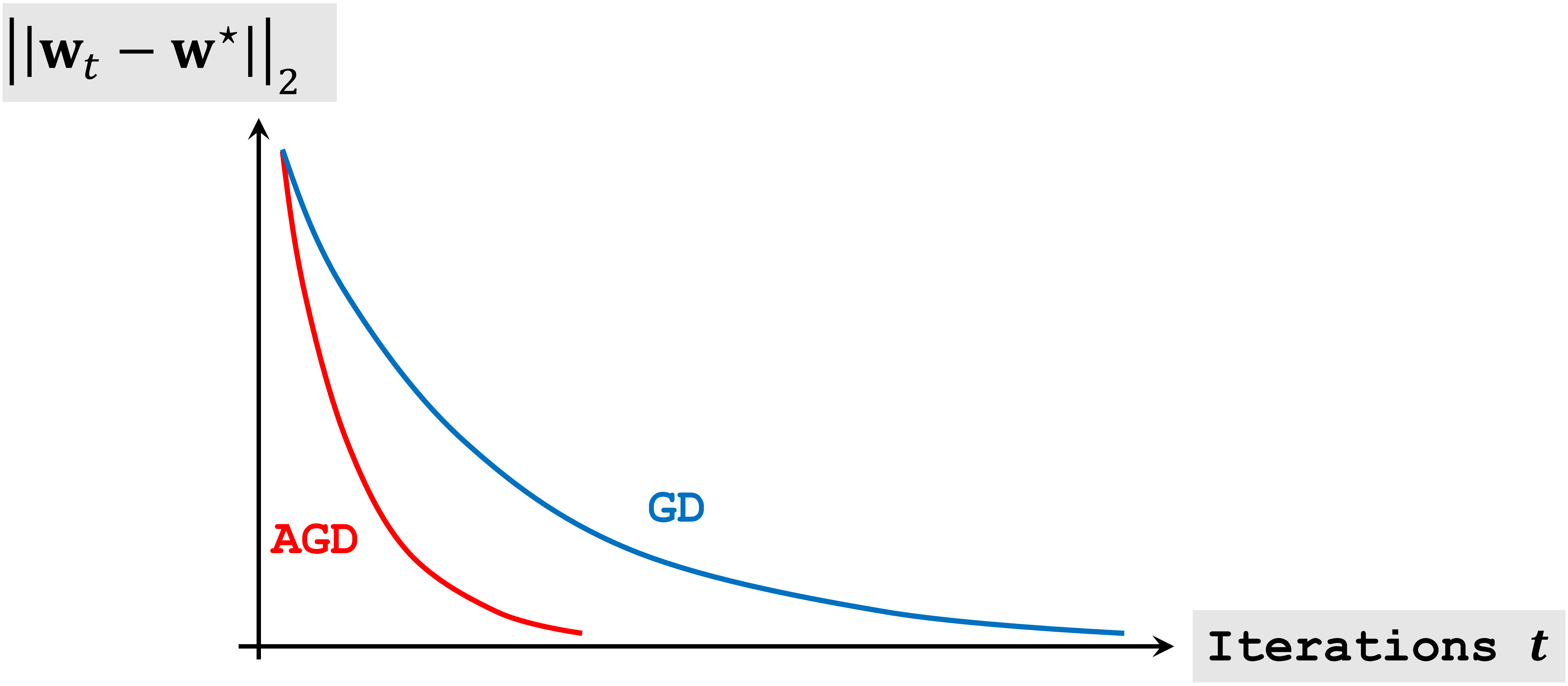
Algorithms

Gradient Descent (GD)

Accelerated GD

Stochastic GD

GD versus AGD



Time Complexity

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n}\sum_{j=1}^{n} \tilde{\mathbf{g}}_{t,j}$, where $\tilde{\mathbf{g}}_{t,j} = \frac{-y_j \mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

Per-iteration time complexity is $O(nd)$.

- $O(d)$ time for computing $\mathbf{w}_t^T \mathbf{x}_j$.
- $O(d)$ time for computing $\tilde{\mathbf{g}}_{t,j}$.
- $O(nd)$ time for computing all the $\tilde{\mathbf{g}}_{t,j}$.

Algorithms

Gradient Descent (GD)

Accelerated GD

Stochastic GD

SGD Algorithm

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n}\sum_{j=1}^{n} \tilde{\mathbf{g}}_{t,j}$, where $\tilde{\mathbf{g}}_{t,j} = \frac{-y_j \mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

The stochastic gradient is close to the full gradient:

$$\mathbf{g}_t = \mathbb{E}_j\left[\tilde{\mathbf{g}}_{t,j}\right],$$

where j is randomly sampled from $\{1, \cdots, n\}$.

SGD Algorithm

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n}\sum_{j=1}^{n} \tilde{\mathbf{g}}_{t,j}$, where $\tilde{\mathbf{g}}_{t,j} = \frac{-y_j \mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

SGD repeats

1. Randomly draw j from $\{1, 2, \cdots, n\}$.
2. Compute the stochastic gradient $\tilde{\mathbf{g}}_{t,j}$.
3. Update: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\, \tilde{\mathbf{g}}_{t,j}$.

Per-iteration time complexity is $O(d)$.

Algorithms

Gradient Descent (GD)

Accelerated GD

Stochastic GD

Accelerated SGD Algorithm

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n}\sum_{j=1}^{n} \tilde{\mathbf{g}}_{t,j}$, where $\tilde{\mathbf{g}}_{t,j} = \frac{-y_j \mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

Accelerated SGD repeats

1. Randomly draw j from $\{1, 2, \cdots, n\}$.
2. Compute the stochastic gradient $\tilde{\mathbf{g}}_{t,j}$.
3. Update momentum: $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \tilde{\mathbf{g}}_{t,j}$.
4. Update: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{v}_{t+1}$.

Algorithms

Gradient Descent (GD)

Accelerated GD

Stochastic GD

SGD Algorithm

Gradient at $\mathbf{w}_t$: $\mathbf{g}_t = \frac{1}{n}\sum_{j=1}^{n}\tilde{\mathbf{g}}_{t,j}$, where $\tilde{\mathbf{g}}_{t,j} = \frac{-y_j \mathbf{x}_j}{1+\exp(y_j \mathbf{w}_t^T \mathbf{x}_j)}$.

Output of SGD:

- Option 1: output the last iteration $\mathbf{w}_{T+1}$
- Option 2: output the average of $\mathbf{w}$ produced by the last tens of iteration.

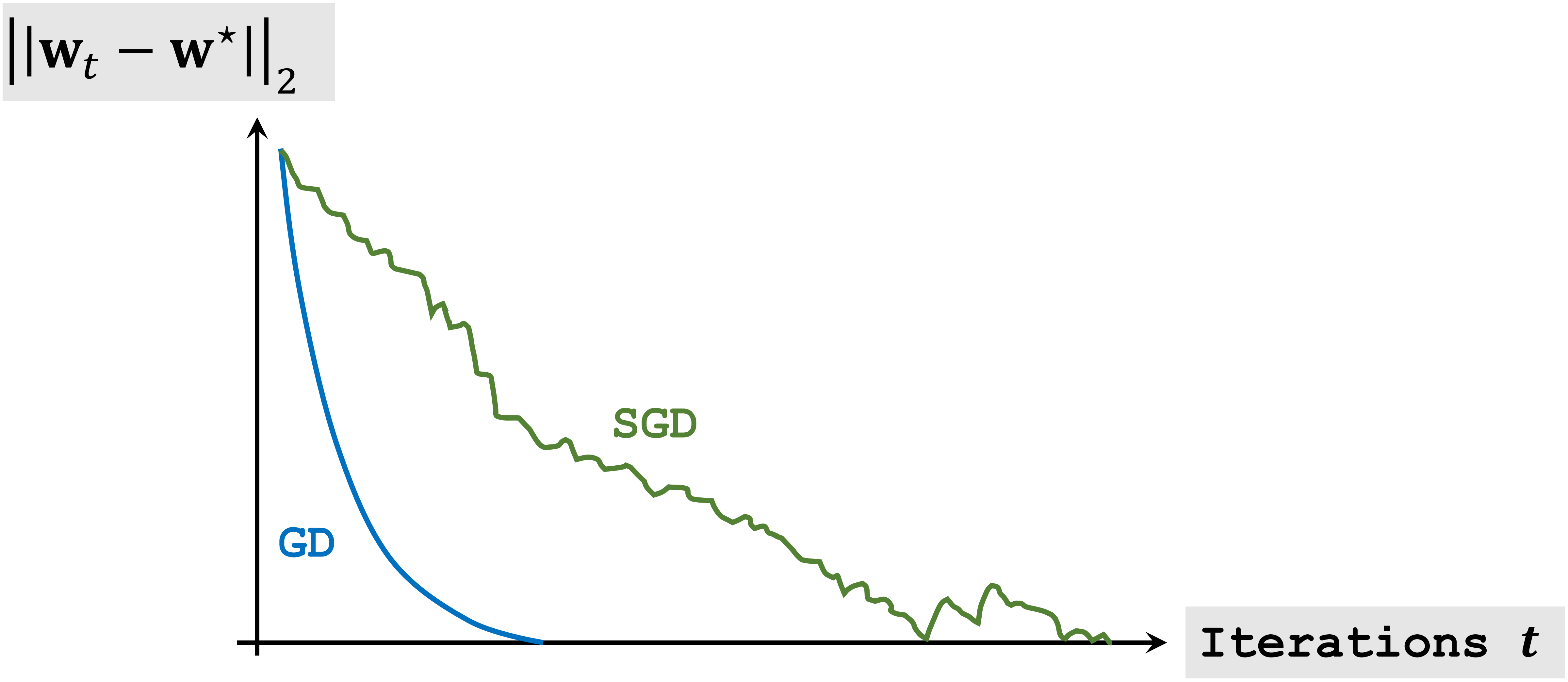
Algorithms

Gradient Descent (GD)

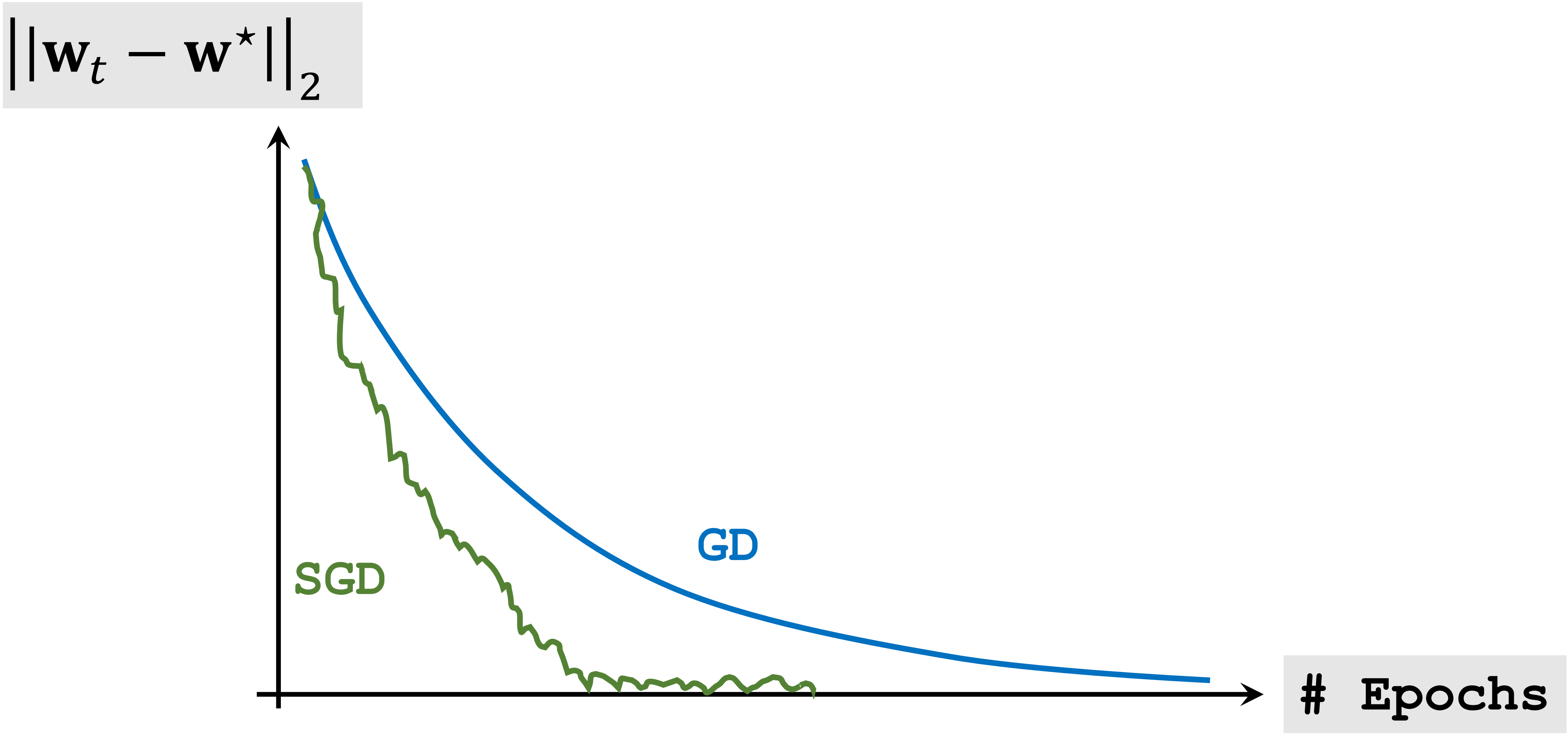
Accelerated GD

Stochastic GD

GD versus SGD



GD versus SGD



Training and Prediction

- Training:

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j l\left(y_j \mathbf{w}^T \mathbf{x}_j\right), \text{ where } l(\mathrm{z}) = \log(1 + e^{-z}).$$

- For a test feature vector $\mathbf{x}' \in \mathbb{R}^d$, make prediction by

$$\operatorname{sign}\left(\mathbf{x}'^T \mathbf{w}^\star\right).$$

Summary

- Logistic regression model for *linear binary* classification.

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j l\left(y_j \mathbf{w}^T \mathbf{x}_j\right), \text{ where } l(\mathrm{z}) = \log(1 + e^{-z}).$$

- Compute the gradient using vector derivatives and the chain rule.
- Gradient-based algorithms: GD, AGD, SGD, etc.
- Make prediction using $\mathrm{sign}\left(\mathbf{x}'^T \mathbf{w}^\star\right)$.

Evaluate Binary Classification

Evaluate Binary Classification

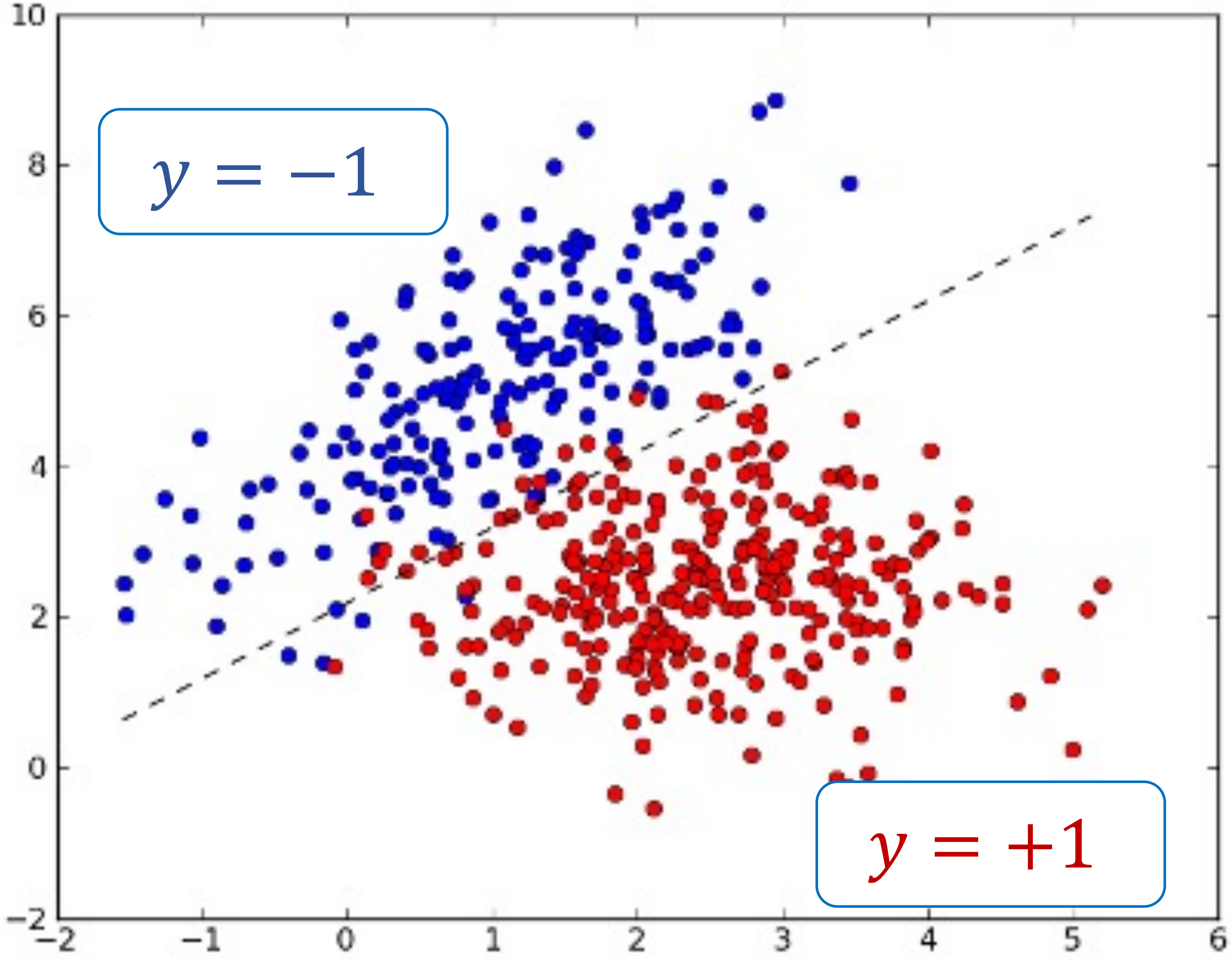
- Error Rate $= \dfrac{\#\ \text{Classification Errors}}{\#\ \text{Samples}}$
- Accuracy = 1 - Error Rate

Evaluate Binary Classification

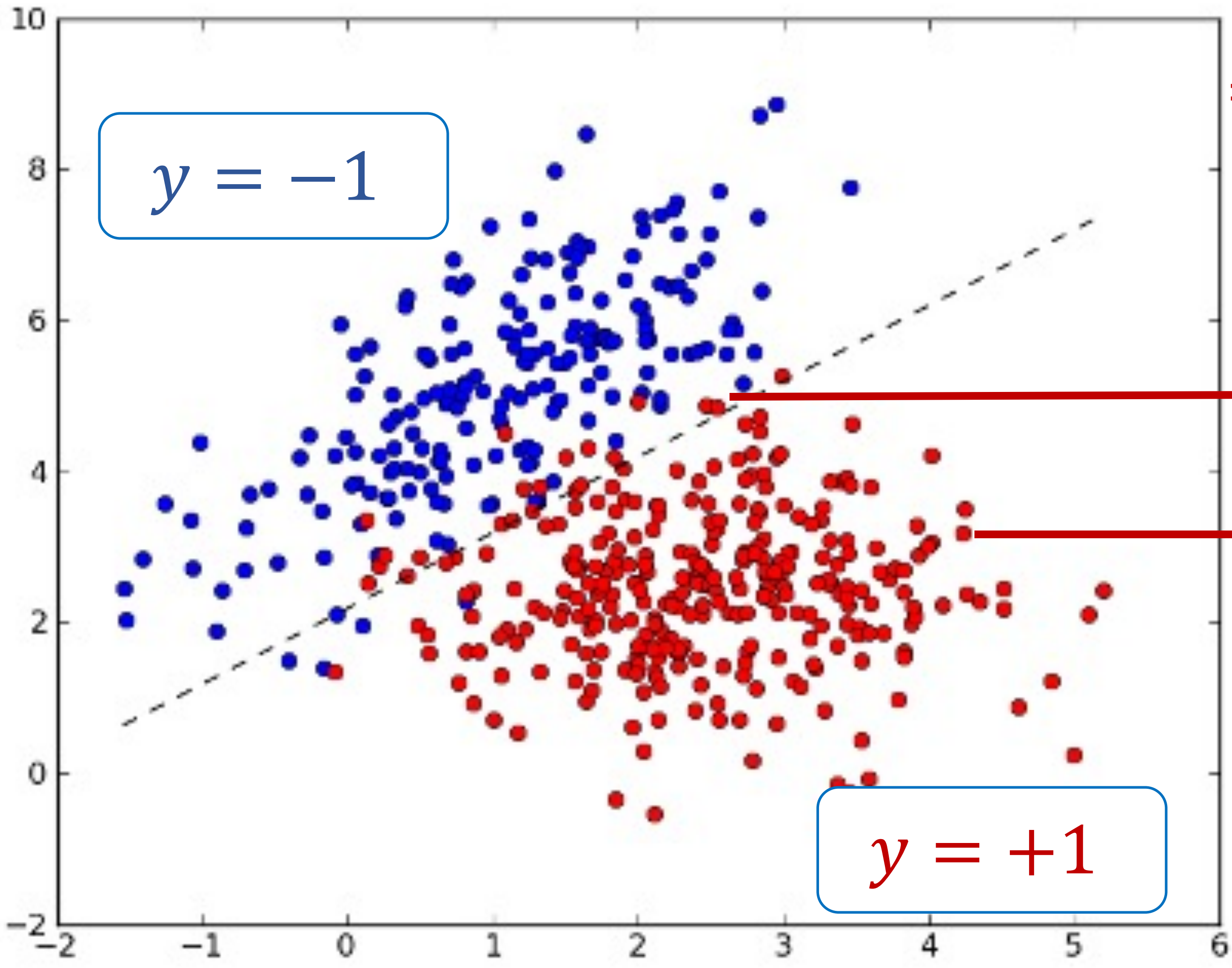
- Error Rate $= \dfrac{\text{\# Classification Errors}}{\text{\# Samples}}$
- Accuracy = 1 - Error Rate

Error rate and **Accuracy** are not meaningful in class-imbalanced problems.

Evaluate Binary Classification



Evaluate Binary Classification



True Positive Rate (TPR)

$$= \frac{\#TP}{\#TP + \#FN}$$

False Negative (FN)

True Positive (TP)

Evaluate Binary Classification

False Positive Rate (FPR)

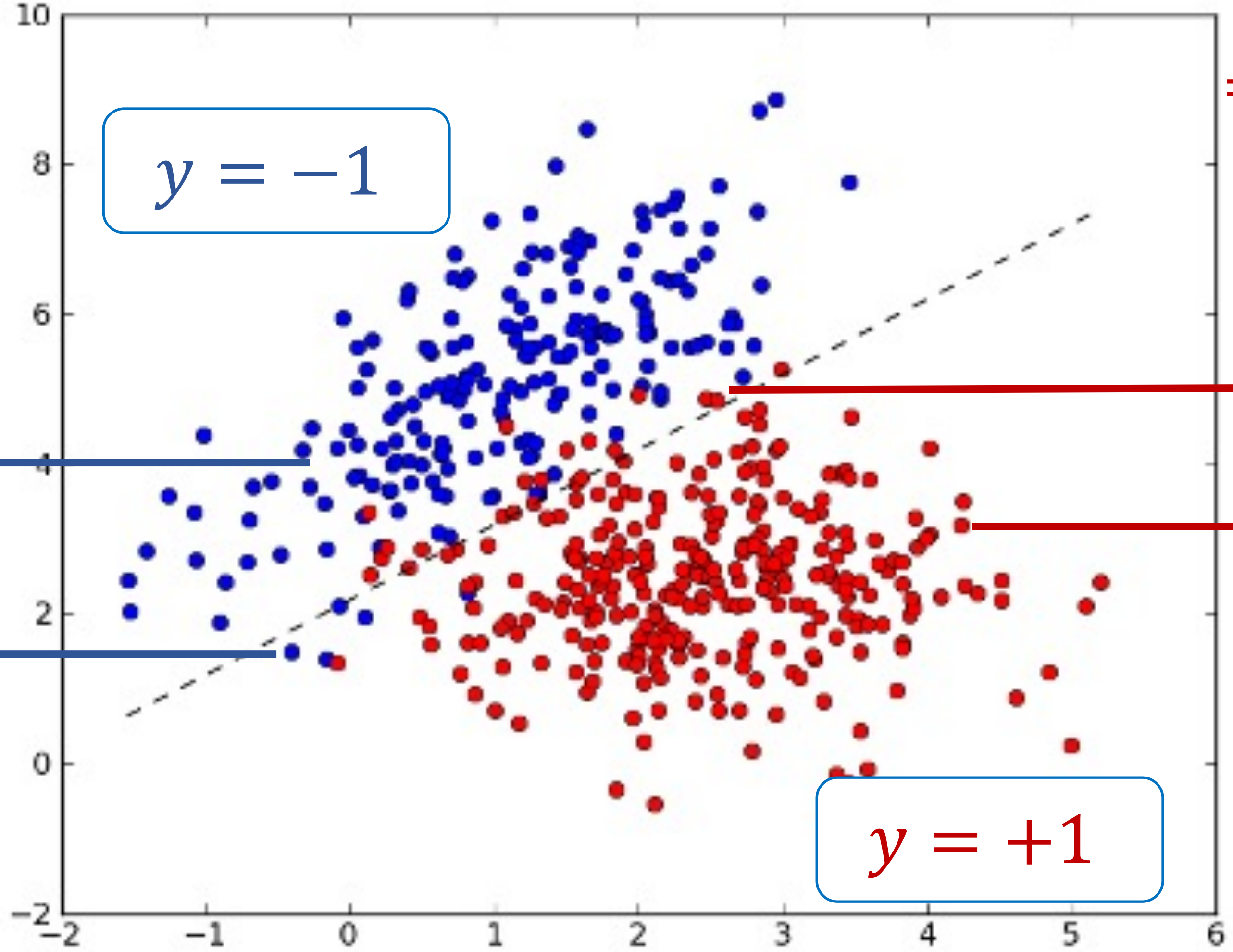
$$= \frac{\#\mathrm{FP}}{\#\mathrm{FP} + \#\mathrm{TN}}$$

True Positive Rate (TPR)

$$= \frac{\#\mathrm{TP}}{\#\mathrm{TP} + \#\mathrm{FN}}$$

True Negative (TN)

False Positive (FP)



False Negative (FN)

True Positive (TP)

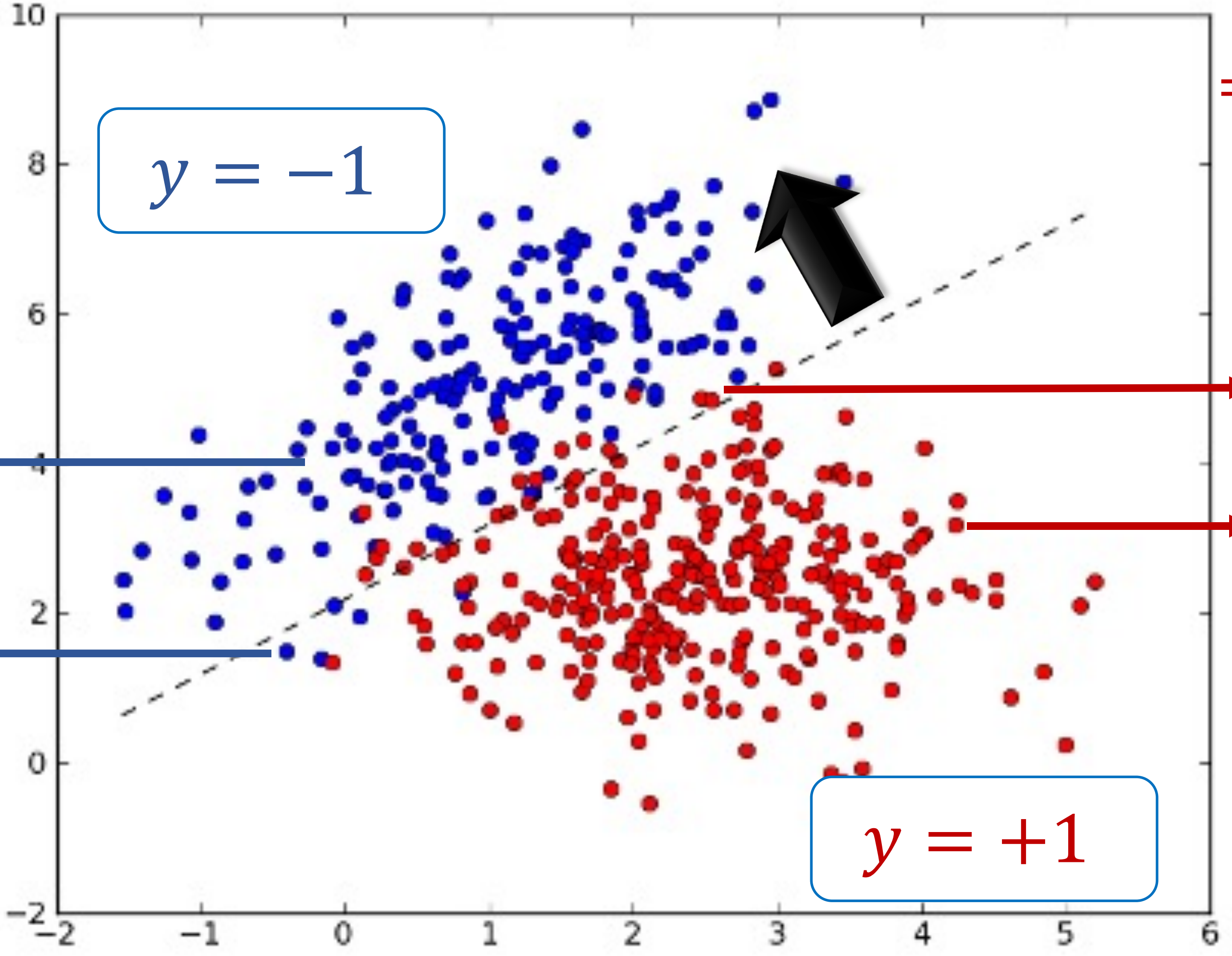
Evaluate Binary Classification

False Positive Rate (FPR)

$$= \frac{\#\mathrm{FP}}{\#\mathrm{FP} + \#\mathrm{TN}} \nearrow$$

True Positive Rate (TPR)

$$= \frac{\#\mathrm{TP}}{\#\mathrm{TP} + \#\mathrm{FN}} \nearrow$$



True Negative (TN)

False Positive (FP)

False Negative (FN)

True Positive (TP)

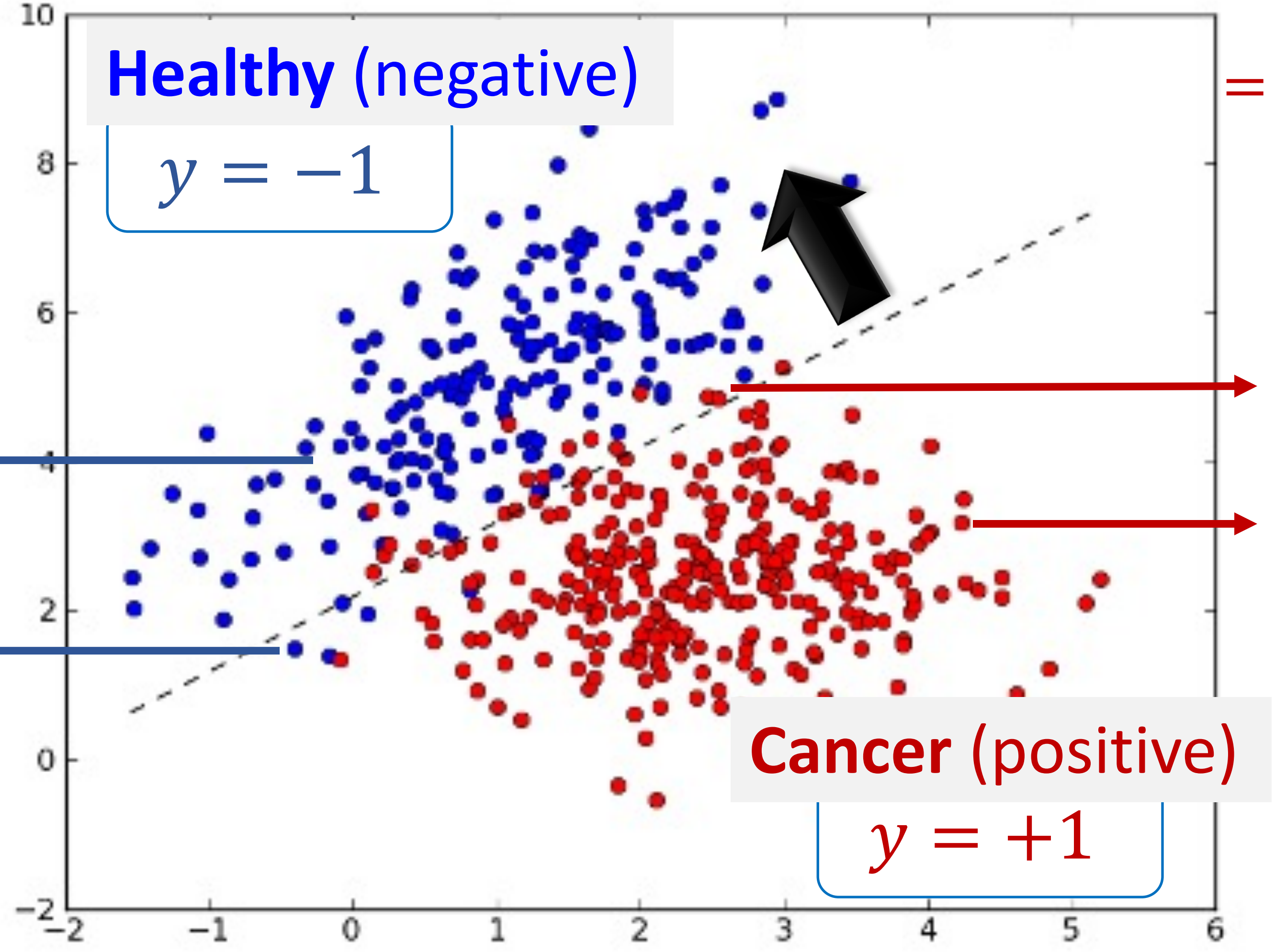
Evaluate Binary Classification

False Positive Rate (FPR)

$$= \frac{\#\mathrm{FP}}{\#\mathrm{FP} + \#\mathrm{TN}} \nearrow$$

True Positive Rate (TPR)

$$= \frac{\#\mathrm{TP}}{\#\mathrm{TP} + \#\mathrm{FN}} \nearrow$$



True Negative (TN)

False Positive (FP)

False Negative (FN)

True Positive (TP)

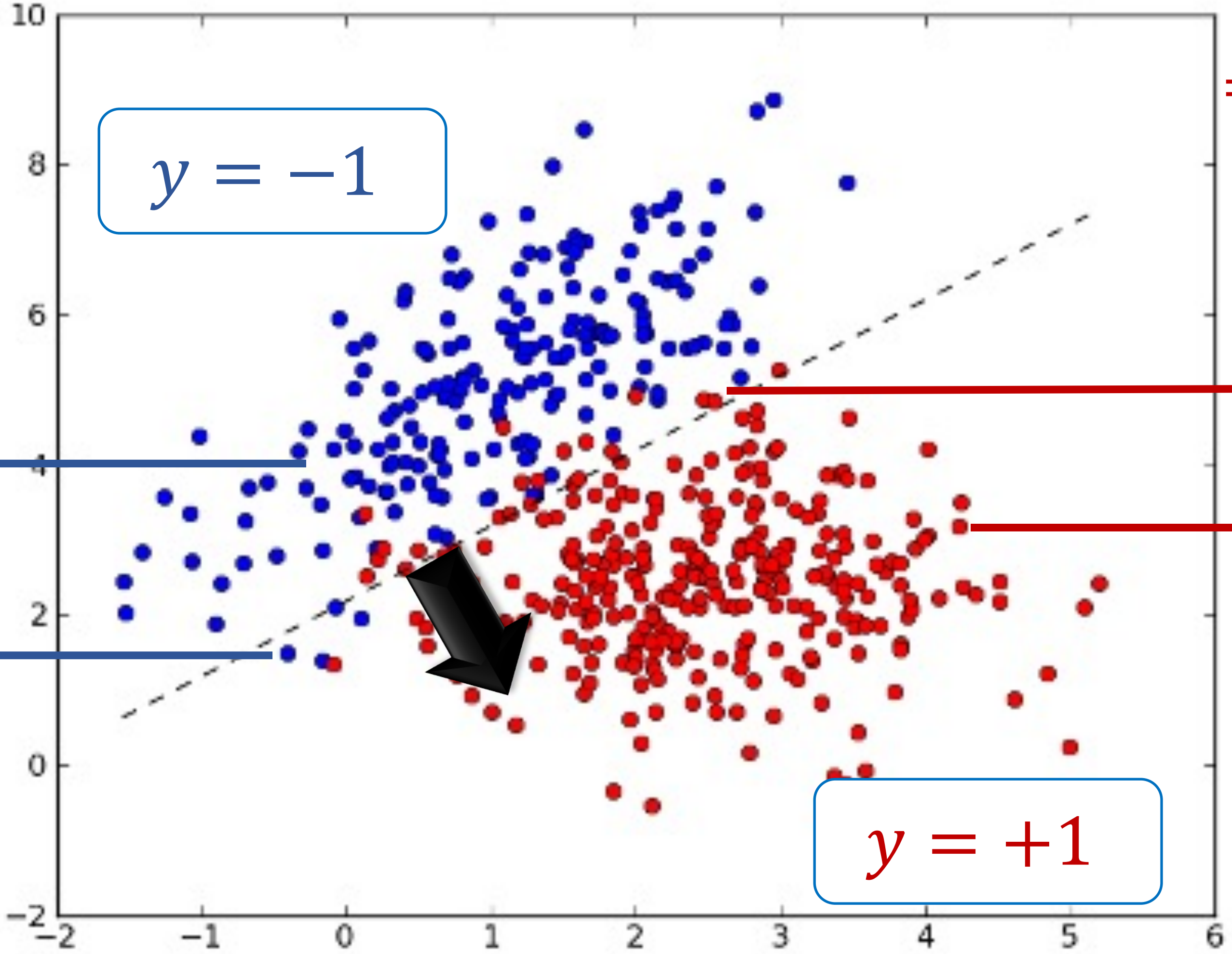
Evaluate Binary Classification

False Positive Rate (FPR)

$$= \frac{\#FP}{\#FP + \#TN}$$

True Positive Rate (TPR)

$$= \frac{\#TP}{\#TP + \#FN}$$



False Negative (FN)

True Negative (TN)

True Positive (TP)

False Positive (FP)

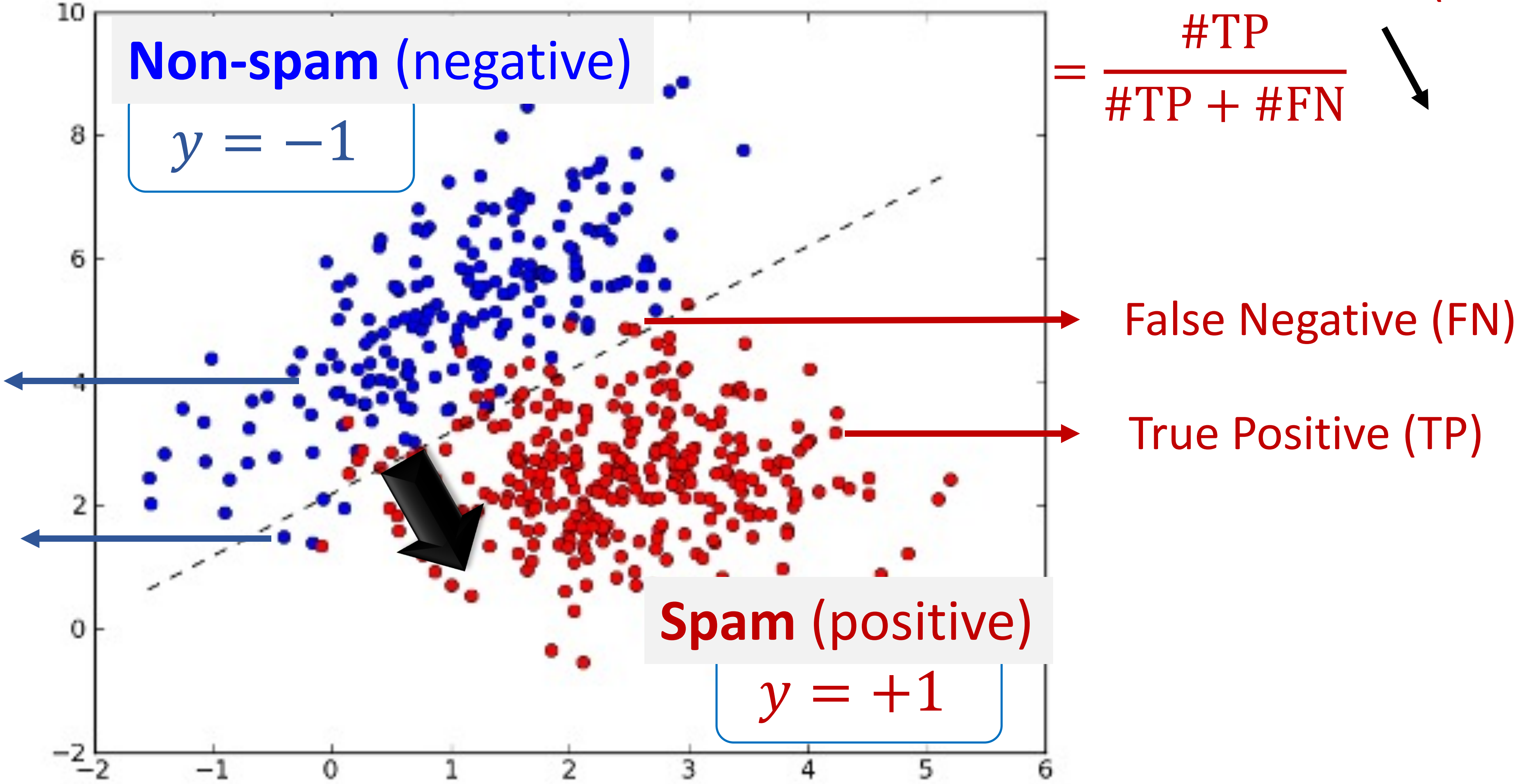
Evaluate Binary Classification

False Positive Rate (FPR)

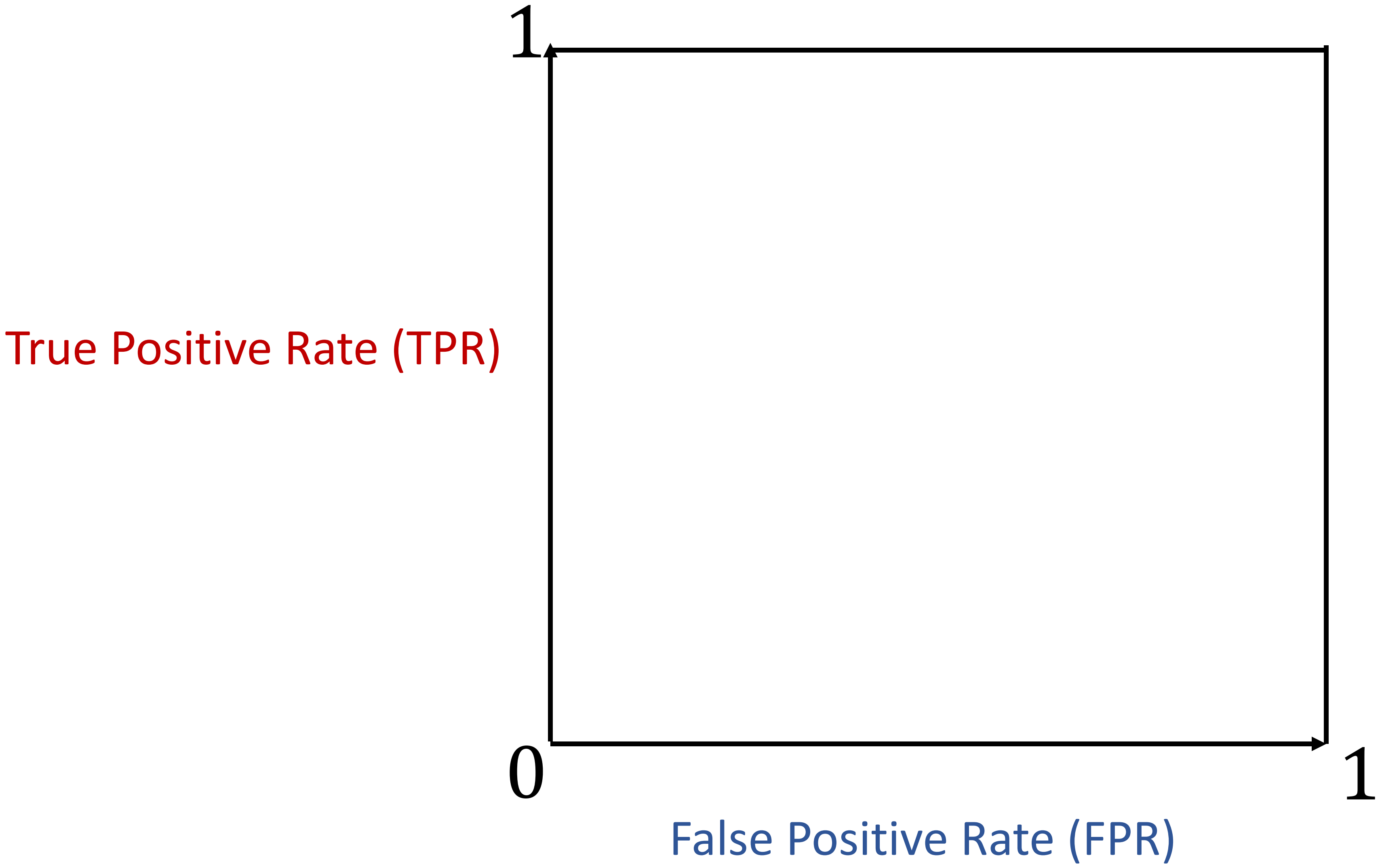
$$= \frac{\#FP}{\#FP + \#TN}$$

True Positive Rate (TPR)

$$= \frac{\#TP}{\#TP + \#FN}$$



Receiver Operating Characteristic (ROC) Curve

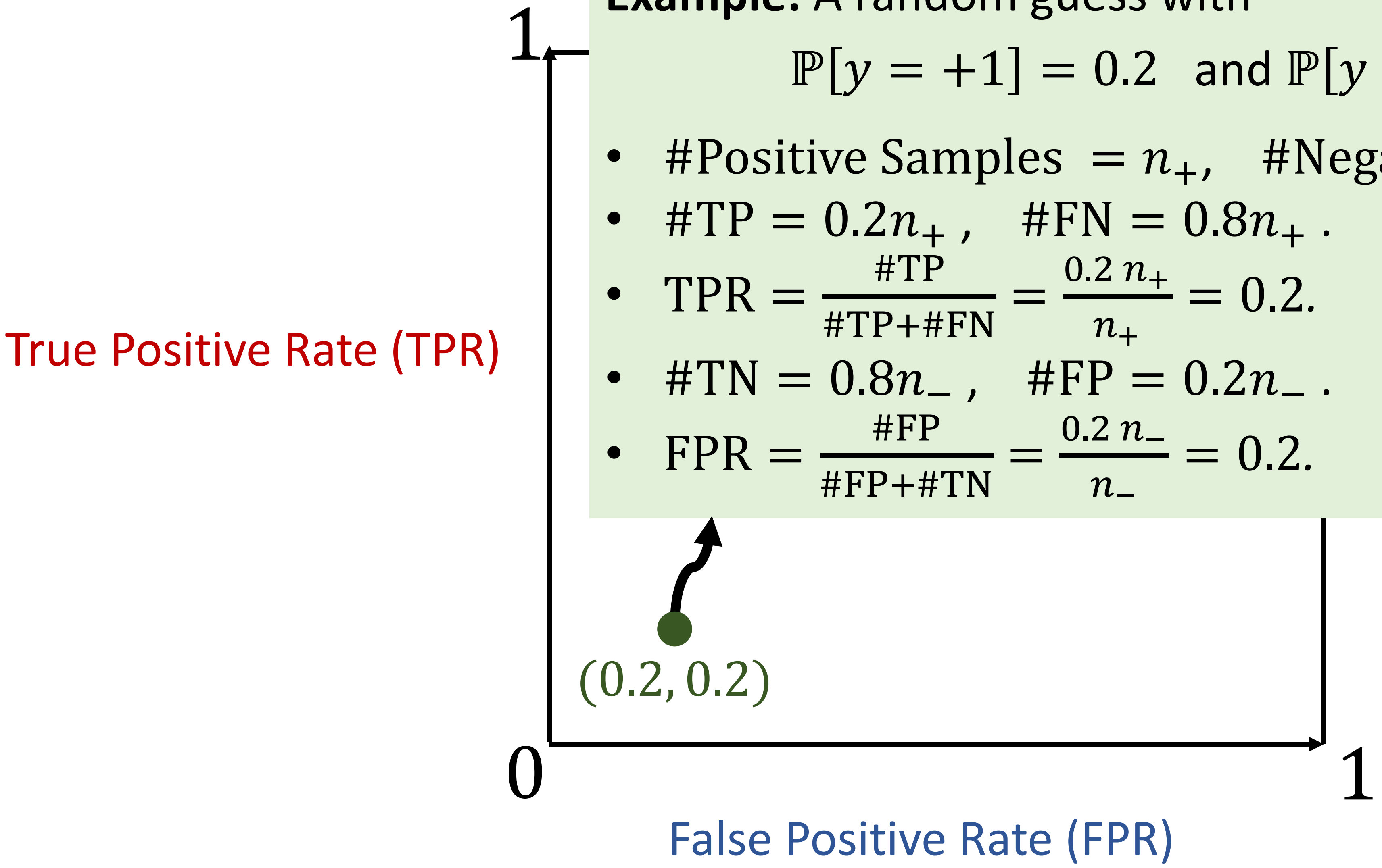


Receiver Operating Characteristic (ROC) Curve

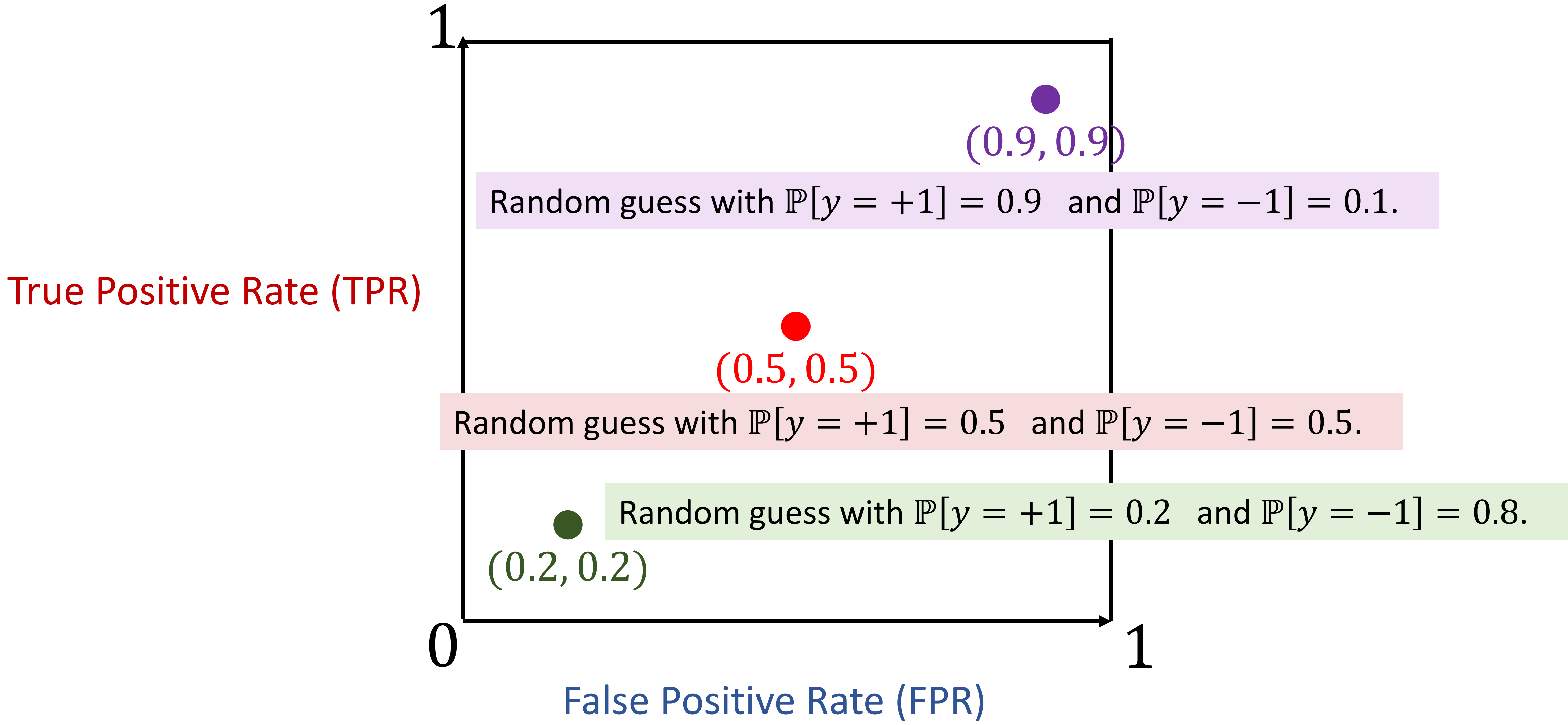
Example: A random guess with

$$\mathbb{P}[y = +1] = 0.2 \quad \text{and} \quad \mathbb{P}[y = -1] = 0.8.$$

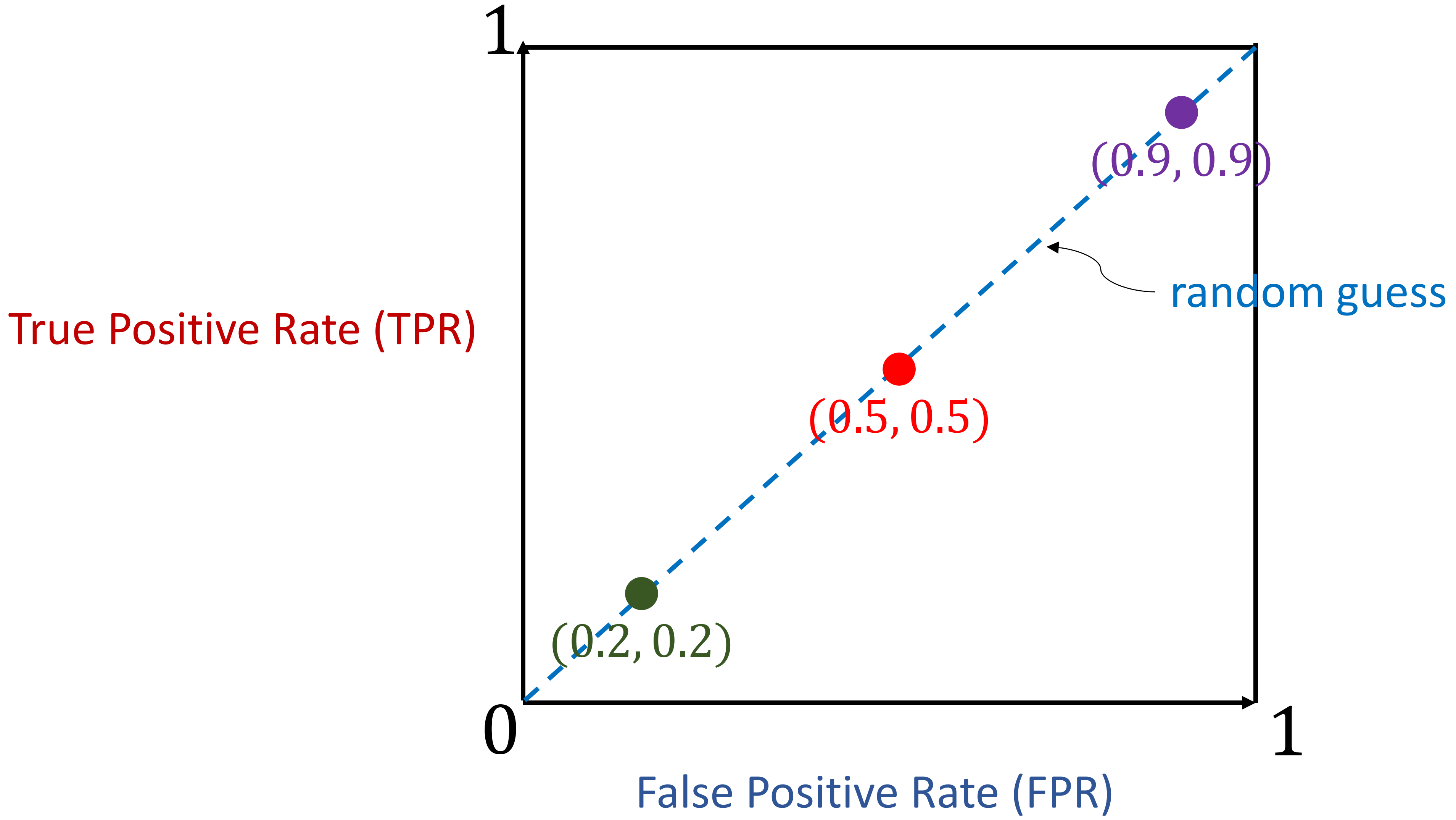
- $\#\text{Positive Samples} = n_+, \quad \#\text{Negative } Samples = n_-.$
- $\#\text{TP} = 0.2 n_+ , \quad \#\text{FN} = 0.8 n_+ .$
- $\text{TPR} = \frac{\#\text{TP}}{\#\text{TP}+\#\text{FN}} = \frac{0.2\, n_+}{n_+} = 0.2.$
- $\#\text{TN} = 0.8 n_- , \quad \#\text{FP} = 0.2 n_- .$
- $\text{FPR} = \frac{\#\text{FP}}{\#\text{FP}+\#\text{TN}} = \frac{0.2\, n_-}{n_-} = 0.2.$



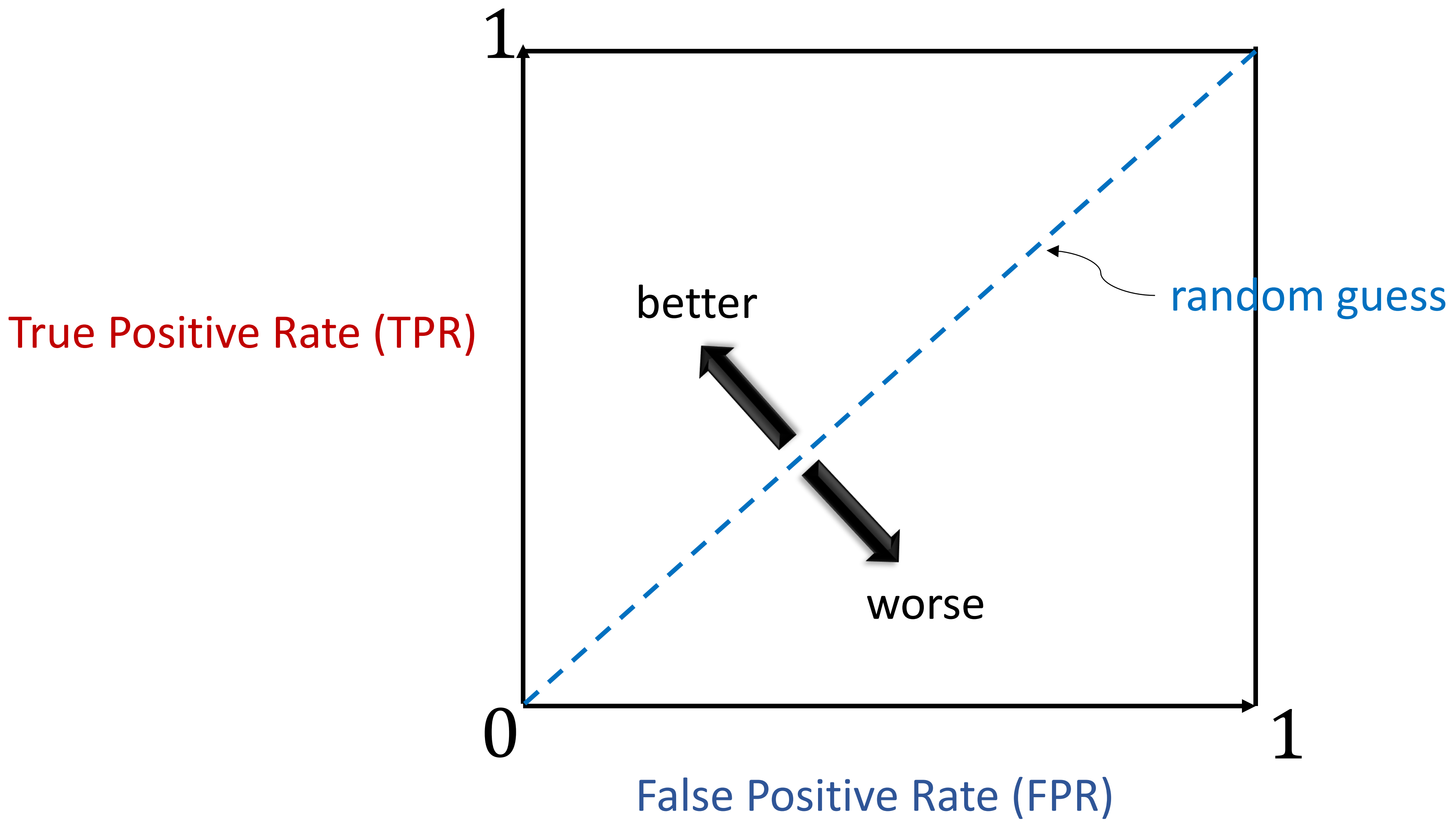
Receiver Operating Characteristic (ROC) Curve



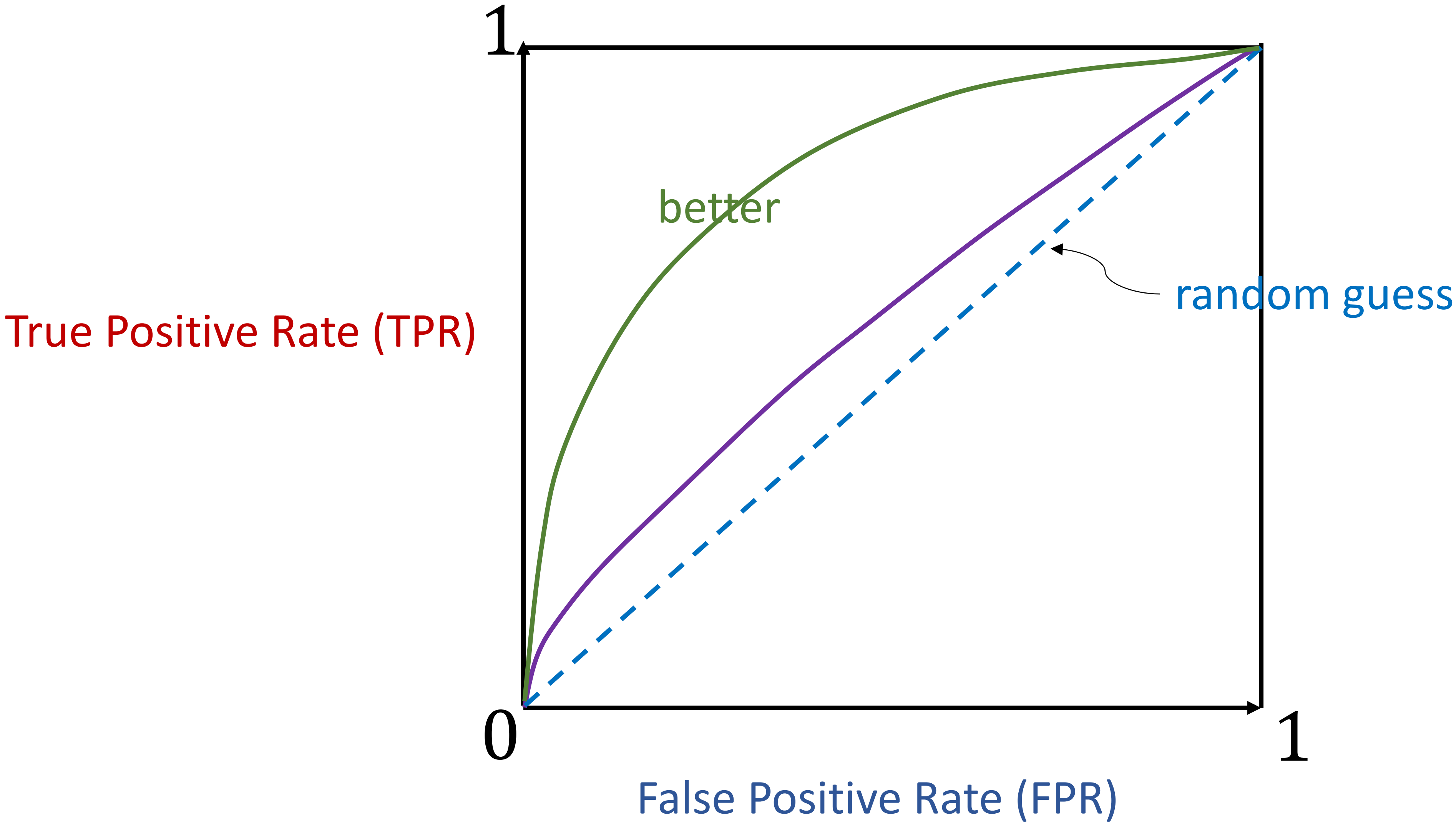
Receiver Operating Characteristic (ROC) Curve



Receiver Operating Characteristic (ROC) Curve



Receiver Operating Characteristic (ROC) Curve



Thank you!