

***COMP 7990***  
***Principles and Practices of***  
***Data Analytics***

*Lecture 4: Multivariate Linear Regression,  
Correlation and Perceptron*

Dr. Eric Lu Zhang

# Outline for Data Preprocessing and Data Mining

- **Data Preprocessing**

- **Supervised learning**

- ❖ Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

- ❖ Classification

1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

# Outline for Data Preprocessing and Data Mining

- Data Preprocessing

- **Supervised learning**

- ❖ **Regression**

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

- ❖ **Classification**

1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

# Multiple variables

Size (feet <sup>2</sup> )	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

Notation:

$n$  = number of variables ( $n=4$ )

$x^{(i)}$  = input (variable) of  $i^{th}$  training example.

$x_j^{(i)}$  = value of variable  $j$  in  $i^{th}$  training example.

# Hypothesis

Previously:  $h_{\theta}(x) = \theta_0 + \theta_1 x$

Multivariate linear regression:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define  $x_0 = 1$  .

# Gradient descent (Optional)

**Hypothesis:**  $h_{\theta}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

**Parameters:**  $\theta_0, \theta_1, \dots, \theta_n$

**Cost function:**

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

**Gradient descent:**

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

} (simultaneously update for every  $j = 0, \dots, n$  )

# Gradient descent (Optional)

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update  $\theta_0, \theta_1$  )

}

New algorithm ( $n \geq 1$ ) :

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update  $\theta_j$   
for  $j = 0, \dots, n$ )

}

---

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

...

# Matrix Multiplication

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \\ \end{bmatrix}$$

The diagram illustrates the calculation of the first element of the resulting matrix. A yellow curved arrow labeled "Dot Product" connects the first row of the first matrix (1, 2, 3) to the first column of the second matrix (7, 9, 11). The result of this dot product, 58, is highlighted in a yellow circle within the first row of the resulting matrix.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \end{bmatrix}$$

The diagram illustrates the calculation of the second element of the resulting matrix. A yellow curved arrow connects the first row of the first matrix (1, 2, 3) to the second column of the second matrix (8, 10, 12). The result of this dot product, 64, is highlighted in a yellow circle within the second column of the resulting matrix.



# Matrix Transpose

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Input

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Output

# Matrix Inverse

The product of A and its inverse is the identity:

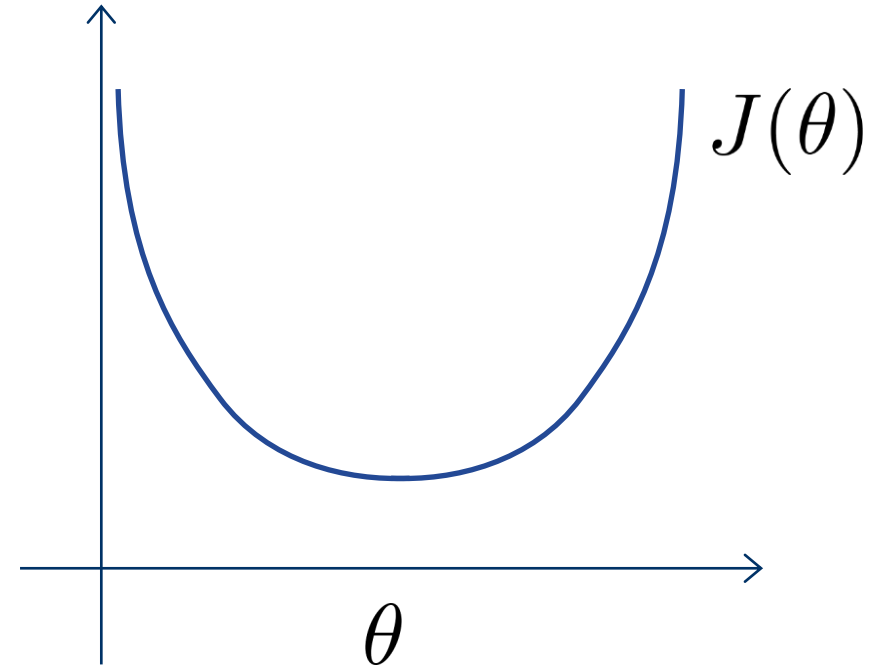
$$\begin{bmatrix} -3 & 1 \\ 5 & 0 \end{bmatrix} \begin{bmatrix} 0 & \frac{1}{5} \\ 1 & \frac{3}{5} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

matrix A                      matrix  $A^{-1}$                       2 x 2 identity matrix

# Normal equation

Gradient Descent:  
Method to solve for  $\theta$  numerically.

Normal equation:  
Method to solve for  $\theta$  analytically.



Gradient descent:

Repeat {  
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$   
}



$$\theta = (X^T X)^{-1} X^T y$$

# Normal equation (optional)

Linear Regression in Matrix Format

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x^{(11)} & \cdots & x^{(1n)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x^{(m1)} & \cdots & x^{(mn)} \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

- $\mathbf{y}$ :  $m \times 1$ ;  $\mathbf{X}$ :  $m \times (n+1)$ ;  $\boldsymbol{\theta}$ :  $(n+1) \times 1$
- The weighted sum of squared errors can be written as

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^m (X^{(i)}\boldsymbol{\theta} - y^{(i)})^2 = \frac{1}{2m} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

# Linear Regression Solution (optional)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (X^{(i)}\theta - y^{(i)})^2 = \frac{1}{2m} (\mathbf{X}\theta - \mathbf{y})^T (\mathbf{X}\theta - \mathbf{y})$$

- Taking derivative with respect to  $\theta$ , and equating to zero, we get

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} \mathbf{X}^T (\mathbf{X}\theta - \mathbf{y}) = 0 \quad \longrightarrow \quad \mathbf{X}^T \mathbf{X} \theta = \mathbf{X}^T \mathbf{y}$$

$$\longrightarrow \quad \theta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

# Gradient descent and normal equation

$m$  training cases,  $n$  variables.

## Gradient Descent

- Need to choose  $\alpha$ .
- Needs many iterations.
- Works well even when  $n$  is large.

## Normal Equation

- No need to choose  $\alpha$ .
- Don't need to iterate.
- Need to compute  $(X^T X)^{-1}$
- Slow if  $n$  is very large.

# Normal equation

Examples:  $m = 4$ .

$x_0$	Size (feet <sup>2</sup> )	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y$$

# Example

Student	Test score	IQ	Study hours
1	100	110	40
2	90	120	30
3	80	100	20
4	70	90	0
5	60	80	10

develop a regression equation to predict test scores ( $y$ ), based on student IQs ( $x_1$ ) and the number of hours that the student studied ( $x_2$ ).

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$



# Example

$$\mathbf{X} = \begin{bmatrix} 1 & 110 & 40 \\ 1 & 120 & 30 \\ 1 & 100 & 20 \\ 1 & 90 & 0 \\ 1 & 80 & 10 \end{bmatrix}$$

$$\mathbf{X}^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 110 & 120 & 100 & 90 & 80 \\ 40 & 30 & 20 & 0 & 10 \end{bmatrix}$$

$$\mathbf{X}^T \mathbf{X} = \begin{bmatrix} 5 & 500 & 100 \\ 500 & 51,000 & 10,800 \\ 100 & 10,800 & 3,000 \end{bmatrix}$$

$$(\mathbf{X}^T \mathbf{X})^{-1} = \begin{bmatrix} 101/5 & -7/30 & 1/6 \\ -7/30 & 1/360 & -1/450 \\ 1/6 & -1/450 & 1/360 \end{bmatrix}$$

# Calculation Steps

$$\mathbf{Y} = \begin{bmatrix} 100 \\ 90 \\ 80 \\ 70 \\ 60 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 20 \\ 0.5 \\ 0.5 \end{bmatrix}$$

$$y = 20 + 0.5x_1 + 0.5x_2$$

During the exam you are not required to perform numeric computation for linear regression with multiple variables. i.e., you would not need to perform matrix multiplication or inverse.

The derivation of the formulas are also not required.

# Outline for Data Preprocessing and Data Mining

- Data Preprocessing

- **Supervised learning**

- ❖ **Regression**

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

- ❖ **Classification**

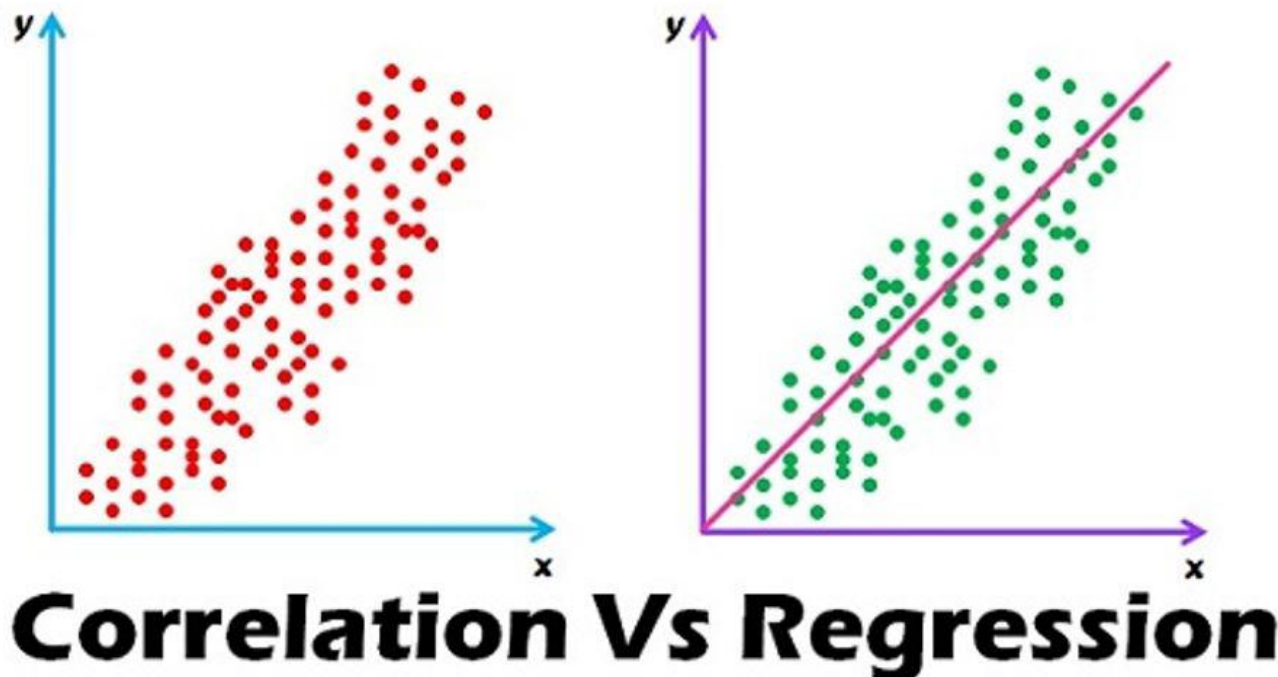
1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

# Linear regression and Correlation

- Correlation coefficient: measure of the strength and direction of linear relationship between two quantitative variables
- Linear regression fits a model predicting a quantitative response (dependent) variable ( $y$ ) based on a quantitative explanatory (independent) variable ( $x$ )



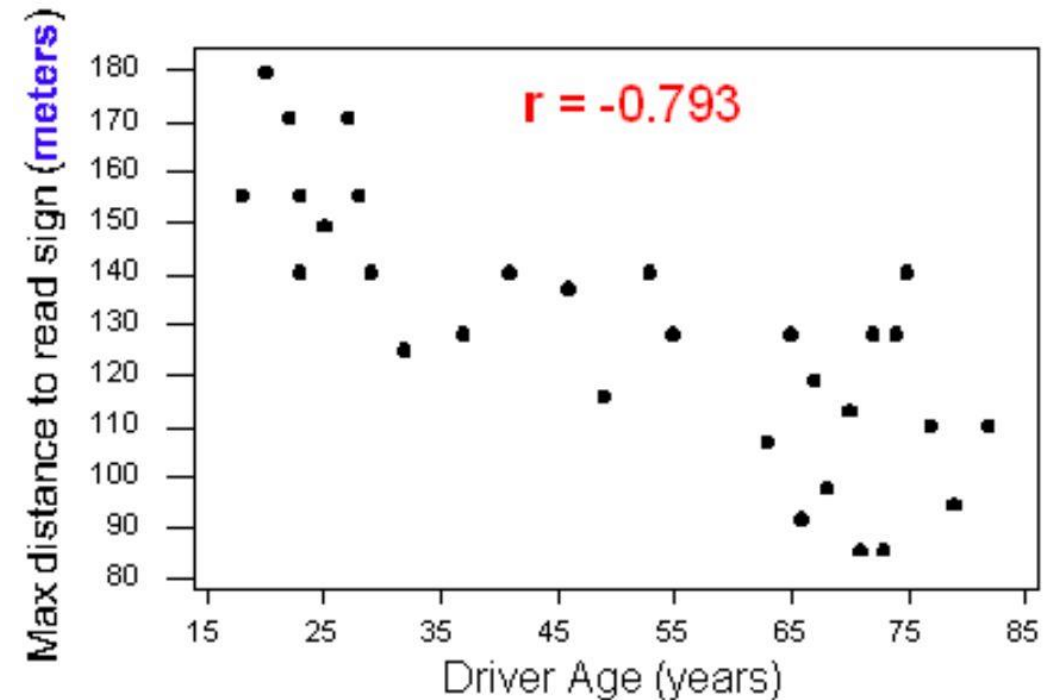
# Pearson Correlation Coefficient

- The correlation coefficient ( $r$ ) is a numeric measure of the strength and direction of a linear relationship between two quantitative variables

$$r_{XY} = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

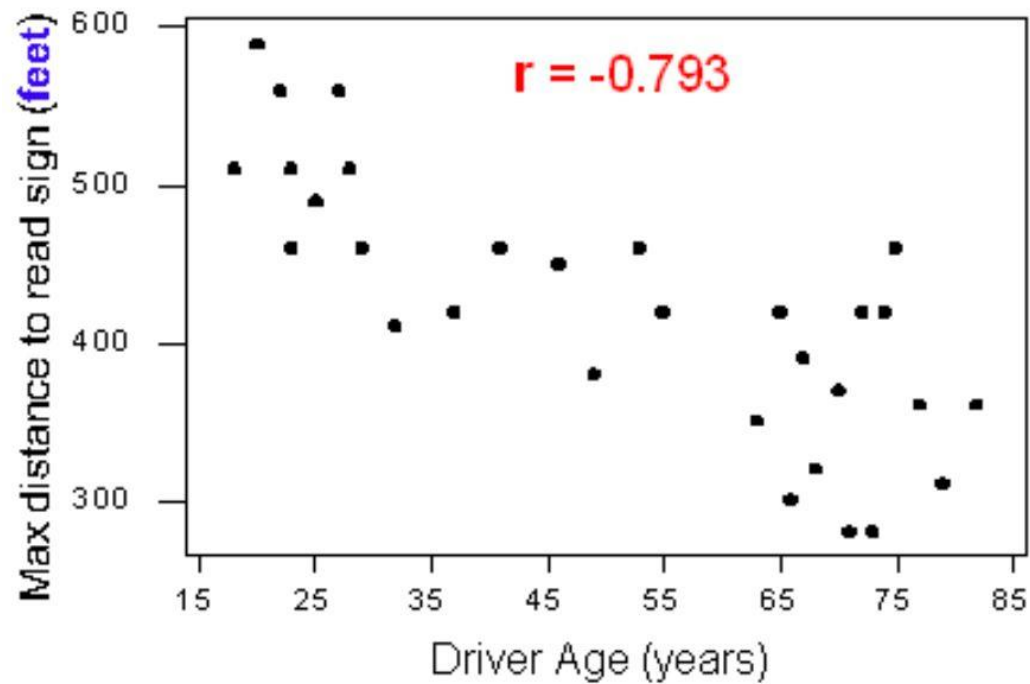
# Pearson Correlation Coefficient

- $-1 \leq r \leq 1$
- The sign indicates the direction of relationship
  - positive relationship:  $r > 0$
  - negative relationship:  $r < 0$
  - no linear relationship:  $r \approx 0$
- The closer  $r$  is to  $\pm 1$ , the stronger the linear relationship
- The correlation between  $X$  and  $Y$  is the same as the correlation between  $Y$  and  $X$

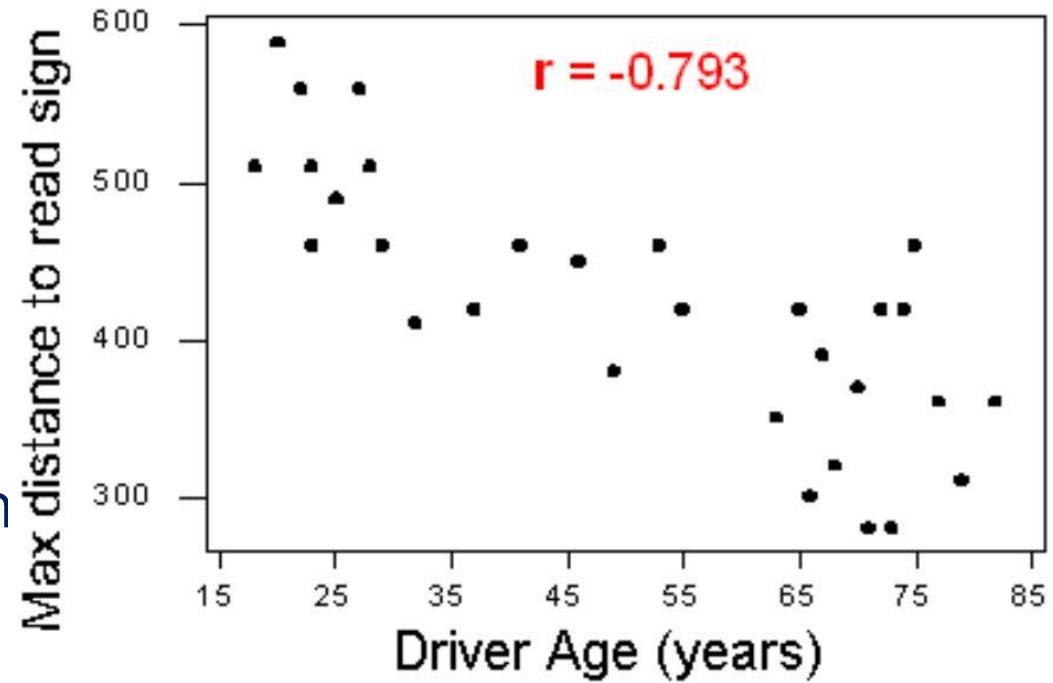


# Properties of r

- r has no units and does not depend on the units of measurement



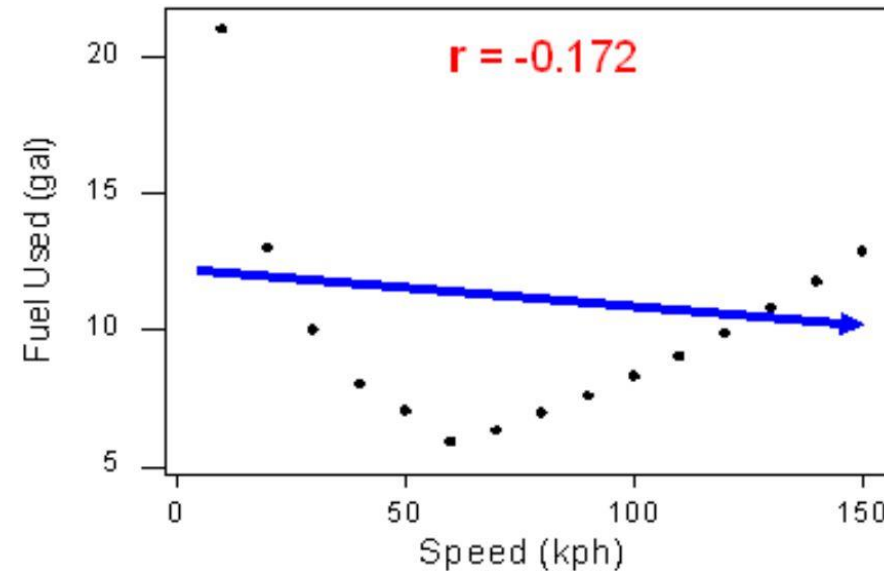
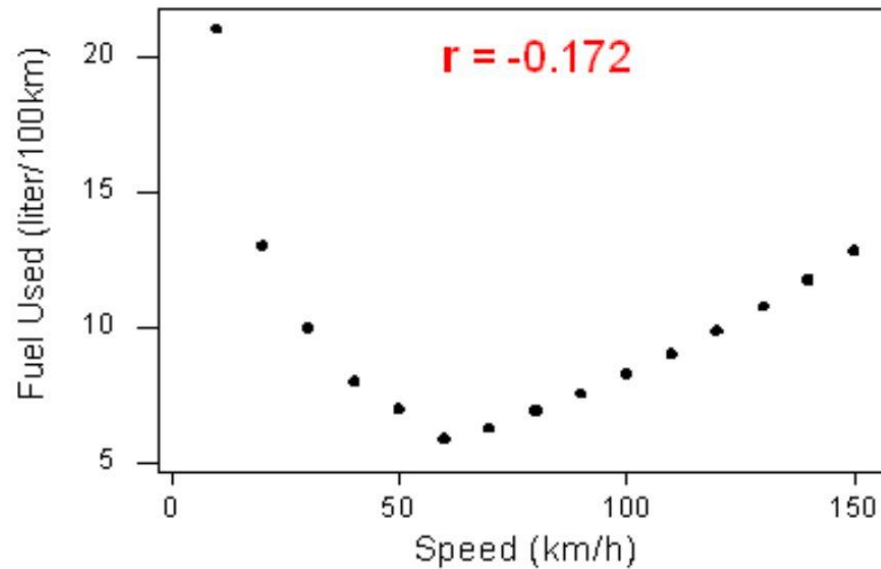
an



e.

# Properties of r

- The correlation measures only the strength of a linear relationship between two variables.



- Notice that the correlation  $r = -0.172$  indicates a weak linear relationship. This makes sense because the data does not closely follow a linear form.
- Always make a scatterplot of the data before calculating and interpreting the meaning of  $r$ .



# Spearman's Correlation Coefficient

- Spearman's rank correlation coefficient can be defined as a special case of Pearson coefficient applied to ranked variables.
  - ❖ not restricted to linear relationships.
  - ❖ measures monotonic association (only strictly increasing or decreasing, but not mixed) between two ranked variables.
  - ❖ In other words, rather than comparing means and variances, Spearman's coefficient looks at the relative order of values for each variable.
  - ❖ can be used with both continuous and discrete data.

# Steps To Calculate Spearman's Rank Correlation Coefficient

- Step1: Assign ranks 1, 2, 3, ..., n to the value of each variable.
  - ❖ Ranking can be descending in order or ascending in order.
  - ❖ However, both data sets should use the same ordering.
- Step2: For each pair of values (x, y), we will calculate
$$d = \text{rank}(x) - \text{rank}(y)$$
  - ❖ We call the difference d.
- Step3: We calculate Spearman's Rank Order Correlation Coefficient as follows:

$$r_s = 1 - \frac{6 \sum d^2}{n(n^2 - 1)}$$

- Step4: Compare the obtained  $r_s$  with the values in the Spearman's Rank Table. The values in this table are the minimum values of r from a sample that need to be reached for Spearman's Rank Correlation Coefficient value to be significant at the level shown.

Spearman's Rank Table

Sample size (n)	p = 0.05	p = 0.025	p = 0.01
4	1.0000	-	-
5	0.9000	1.0000	1.0000
6	0.8286	0.8857	0.9429
7	0.7143	0.7857	0.8929
8	0.6429	0.7381	0.8333
9	0.6000	0.7000	0.7833
10	0.5636	0.6485	0.7455
11	0.5364	0.6182	0.7091
12	0.5035	0.5874	0.6783
13	0.4825	0.5604	0.6484
14	0.4637	0.5385	0.6264
15	0.4464	0.5214	0.6036

# Exercise

- The following is a small selection of countries ranked according to the Human Development Index (HDI) and Income per capita (GDP per capita at PPP in USD). These data came from UNDP-HDR 2003. [Note: Income rank is a rank with respect to these ten countries and does not correspond to the GNP per capita ranking provided by UNDP.]

Country	Income rank (x)	HDI rank (y)	d = rank x - rank y	d <sup>2</sup>
Norway	3	1	2	4
Iceland	2	2	0	0
Sweden	10	3	7	49
Australia	8	4	4	16
Netherlands	5	5	0	0
Belgium	7	6	1	1
United States	1	7	-6	36
Canada	6	8	-2	4
Japan	9	9	0	0
Switzerland	4	10	-6	36
				$\sum d^2 = 146$

# Outline for Data Preprocessing and Data Mining

- Data Preprocessing

- **Supervised learning**

- ❖ Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

- ❖ **Classification**

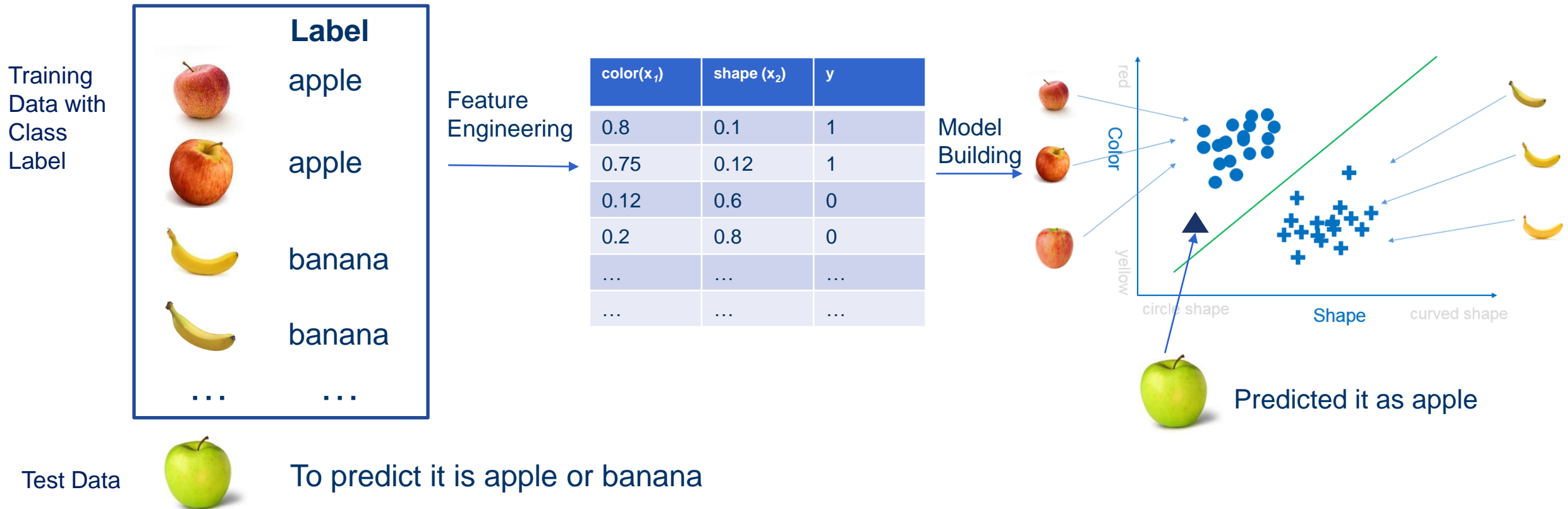
1. **Perceptron**
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

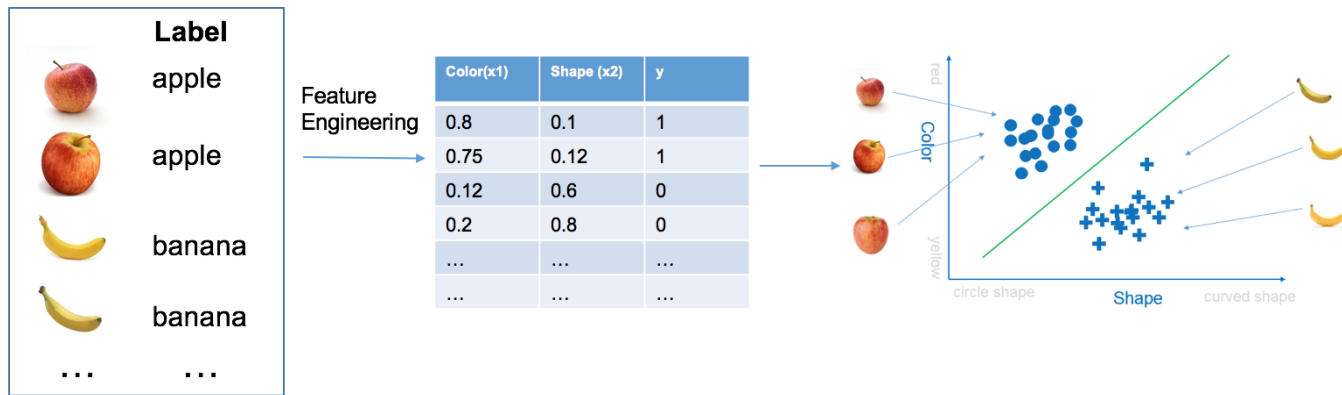
1. K-means Clustering
2. Hierarchical Clustering

# An Example of Classification Problem

- Learn to recognize apple or banana



# A Typical Classification Pipeline



Collecting  
Labelled Data

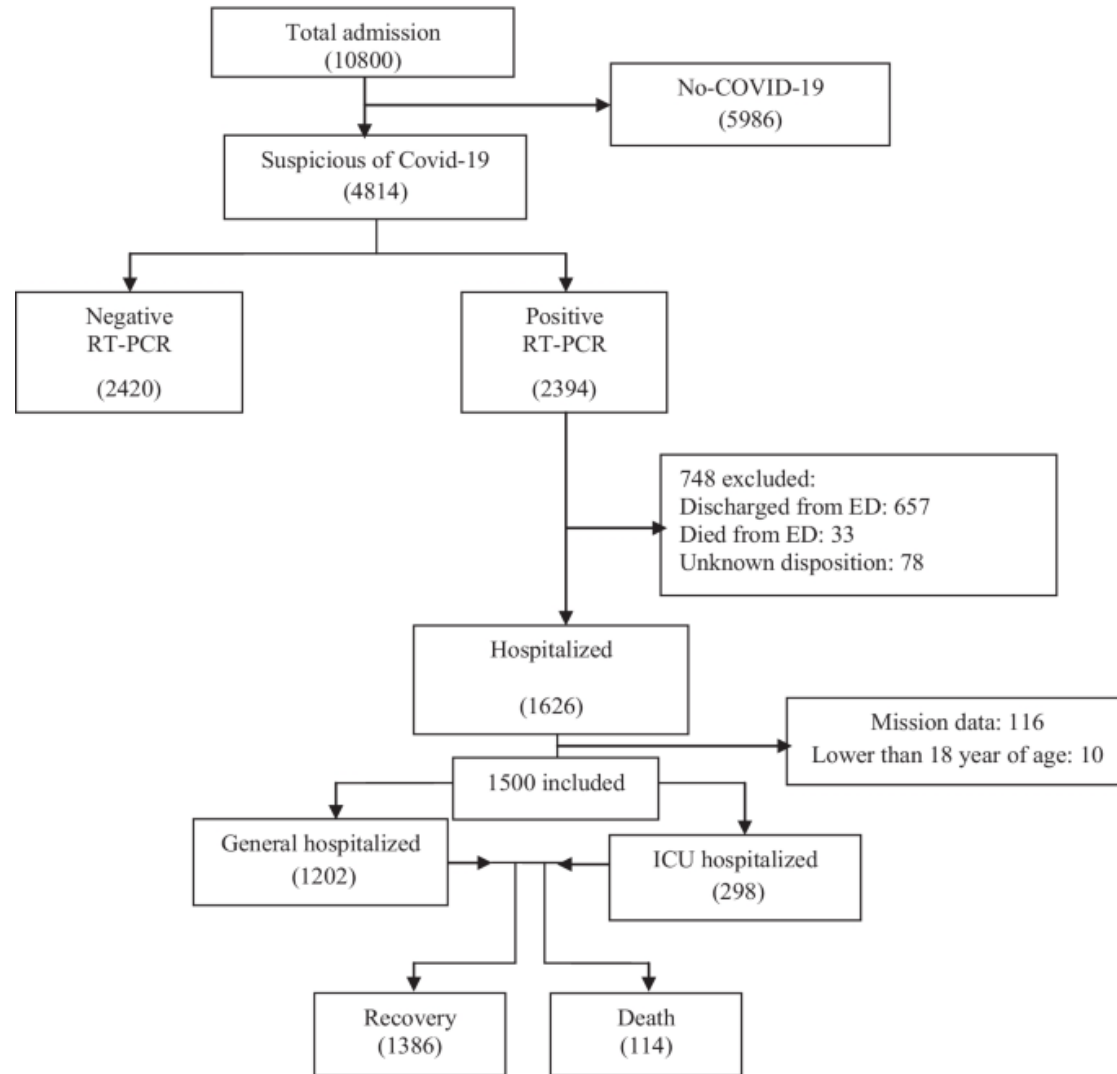
Feature  
Engineering

Building  
Classification  
model

Model  
Evaluation

Model  
Deployment

# Application: Covid-19 Mortality Prediction (Data Preprocessing)



# Application: Covid-19 Mortality Prediction (Feature Selection)

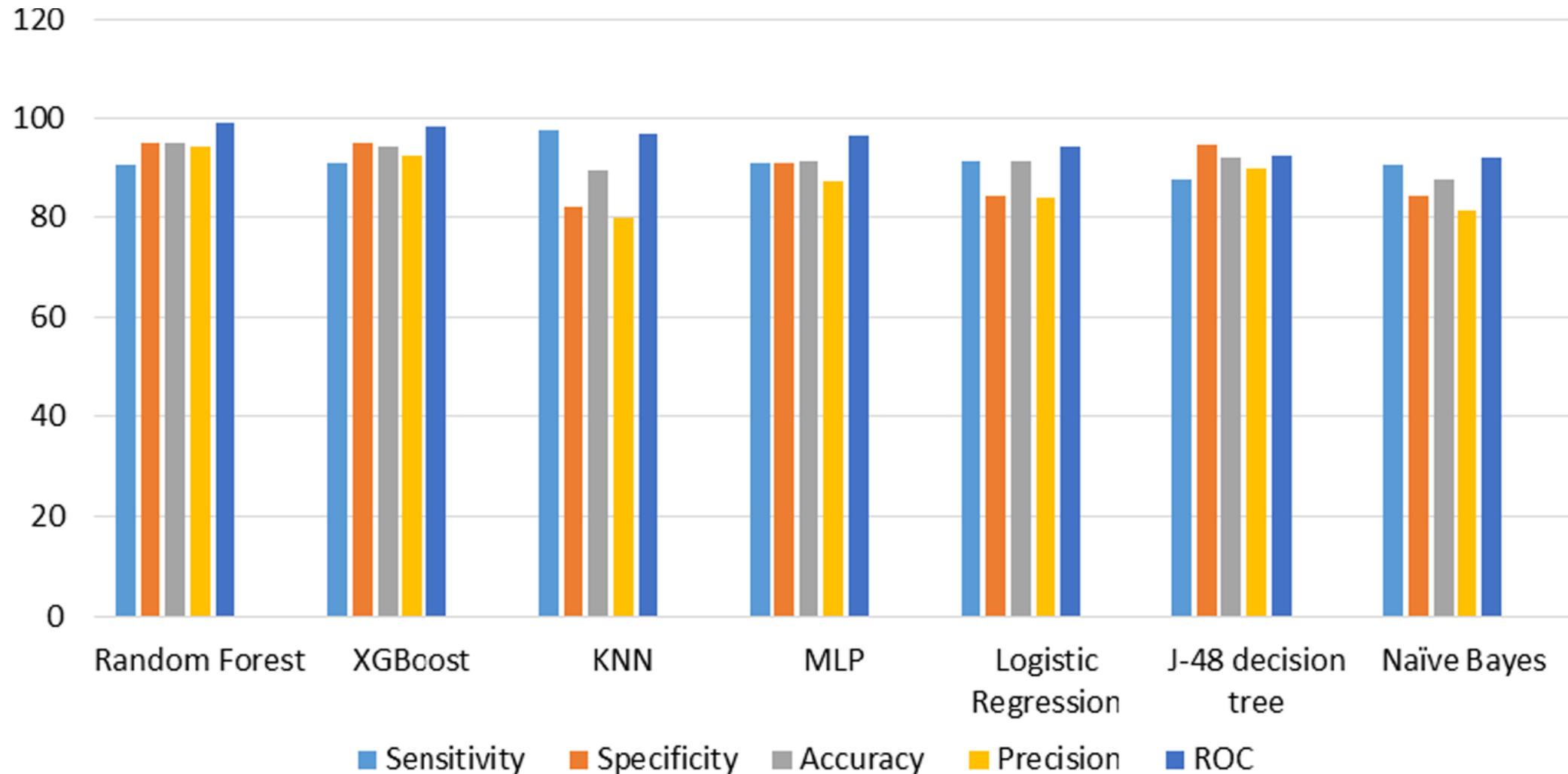
Row	Features name	Degree of importance	Row	Features name	Degree of importance
1	Dyspnea	0.5532	21	Chest pain and pressure	0.2256
2	ICU admission	0.5409	22	Absolute neutrophil count	0.2123
3	Oxygen therapy	0.3789	23	Headache	0.1992
4	Age	0.3207	24	Gender	0.186
5	Fever	0.3142	25	Gastrointestinal symptoms	0.1802
6	Cough	0.3072	26	White cell count	0.1702
7	Loss of taste	0.2944	27	C-reactive protein	0.1574
8	Loss of smell	0.2923	28	Hypersensitive troponin	0.1428
9	Hypertension	0.2768	29	Pneumonia	0.1066
10	Contusion	0.2744	30	Glucose	0.0906
11	Muscular Pain	0.2731	31	Erythrocyte sedimentation rate	0.0826
12	Chill	0.2537	32	Creatinine	0.0716
13	Runny nose	0.2532	33	Alkaline phosphatase	0.0678
14	Blood urea nitrogen	0.2524	34	Length of hospitalization	0.0626
15	Diabetes	0.2506	35	Aspartate aminotransferase	0.0445
16	Sore throat	0.25	36	Smoking	0.0427
17	Absolute lymphocyte count	0.2339	37	Alanine aminotransferase	0.0319
18	Nausea/vomiting	0.2301	38	Platelet count	0.0210
19	Other under line disease	0.2282	39		
20	Cardiac disease	0.2274			



# Application: Covid-19 Mortality Prediction (Feature Statistics)

Features (quantitative)	Range	Mean (SD)
Age (year)	18–100	57.25 (17.8)
Leng of hospitalization	1–32	61.89 (13.25)
Creatinine (mg/dL)	0.1–17.9	51.39 (14.4)
White-cell count	1300–63,000	82.34 (4897.4)
Platelet count	108,000–691,000	66.2 (38.1)
Absolute lymphocyte count	2–95	23.74 (11.8)
Absolute neutrophil count	8–98	74.52 (12.3)
Blood urea nitrogen	0.5–251	42.52 (31.7)
Aspartate aminotransferase	3.8–924	44.45 (53.5)
Alanine aminotransferase	2–672	38.29 (41.6)
Glucose	18–994	36.09 (74.2)
Lactate dehydrogenase	4.6–6973	55.68 (339.0)
Prothrombin time	0.9–46.8	42.82 (23.9)
Alkaline phosphatase	9.6–2846	21.12 (39.2)
Erythrocyte sedimentation rate	2–258	40.65 (28.8)

# Application: Covid-19 Mortality Prediction (Model Selection)

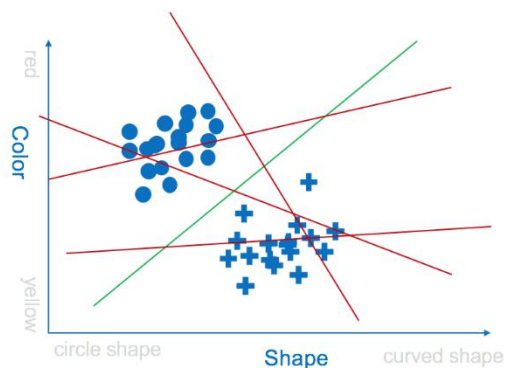


## Application: Covid-19 Mortality Prediction (Benefit)

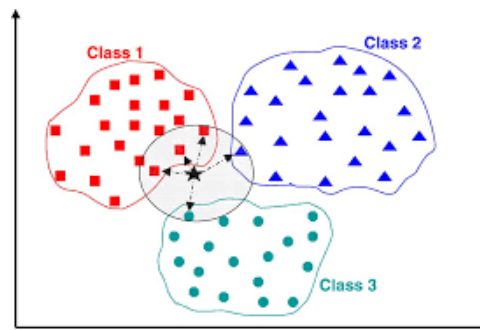
- Optimal use of hospital resources for
  - treating the patients with more critical conditions and
  - assisting in providing more qualitative care and
  - reducing medical errors due to fatigue and long working hours in the ICU

# Overview of Classification Models

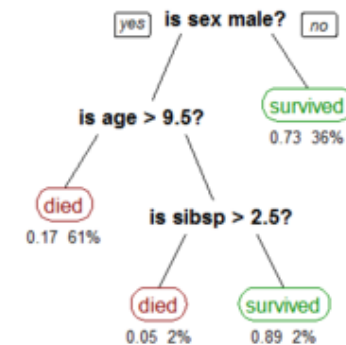
## ■ Simple Models



*Perceptron*

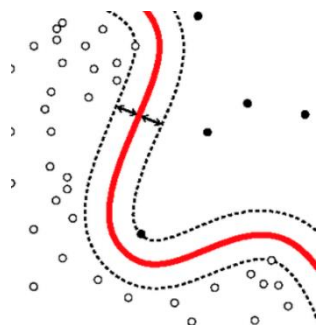


*K nearest Neighbors*

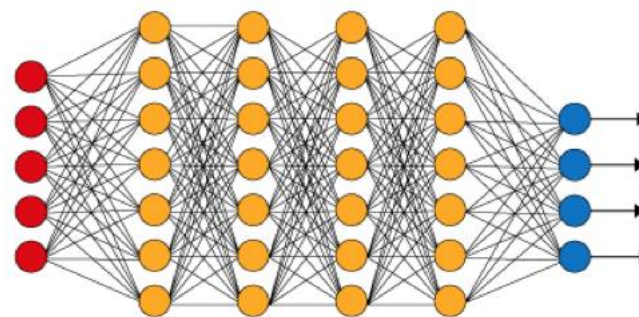


*Decision Tree*

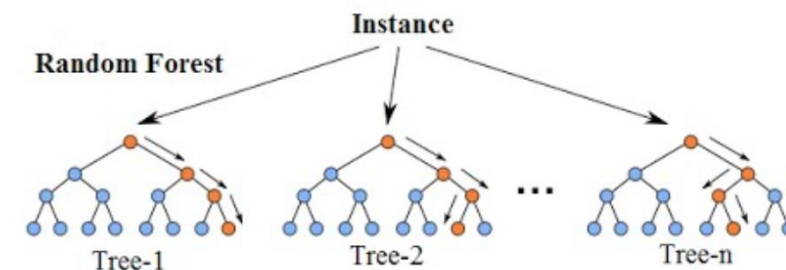
## ■ Complex Models



*Support Vector Machine*



*(Deep) Neural Networks*



*Ensemble methods:  
Random Forest; Gradient Boosting Tree*

# Notation

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
7	3.2	4.7	1.4	versicolor
6.3	3.3	6	2.5	virginica
...	...	...	...	...

- **$\mathbf{X}$** : the  $m \times n$  feature matrix
  - $m$ : the number of data samples
  - $n$ : the dimensionality of each data sample
- **$\mathbf{y}$** :  $m \times 1$  label vector
- **$\mathbf{X}(i,:)$** : the  $i$ -th row in matrix  **$\mathbf{X}$** 
  - i.e., the  $i$ -th data sample
- **$\mathbf{x}_i$** : the  $i$ -th sample in column vector representation
  - $\mathbf{x}_i = \mathbf{X}(i,:)^T$
- $x_{(i,j)}$ : the  $j$ -th feature of the  $i$ -th sample
- $y_i$ : the label of the  $i$ -th sample

Capital bold letter for matrix, small bold letter for vector, small (italic) letter for scalar. Vectors are column vectors

# Notation

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
7	3.2	4.7	1.4	versicolor
6.3	3.3	6	2.5	virginica
...	...	...	...	...

$x_{11}$

$x_{13}$

$y_1$

$\mathbf{X}(2,:) = [4.9, 3, 1.4, 0.2]$  it is a row vector

$$\mathbf{x}_2 = \begin{bmatrix} 4.9 \\ 3 \\ 1.4 \\ 0.2 \end{bmatrix}$$

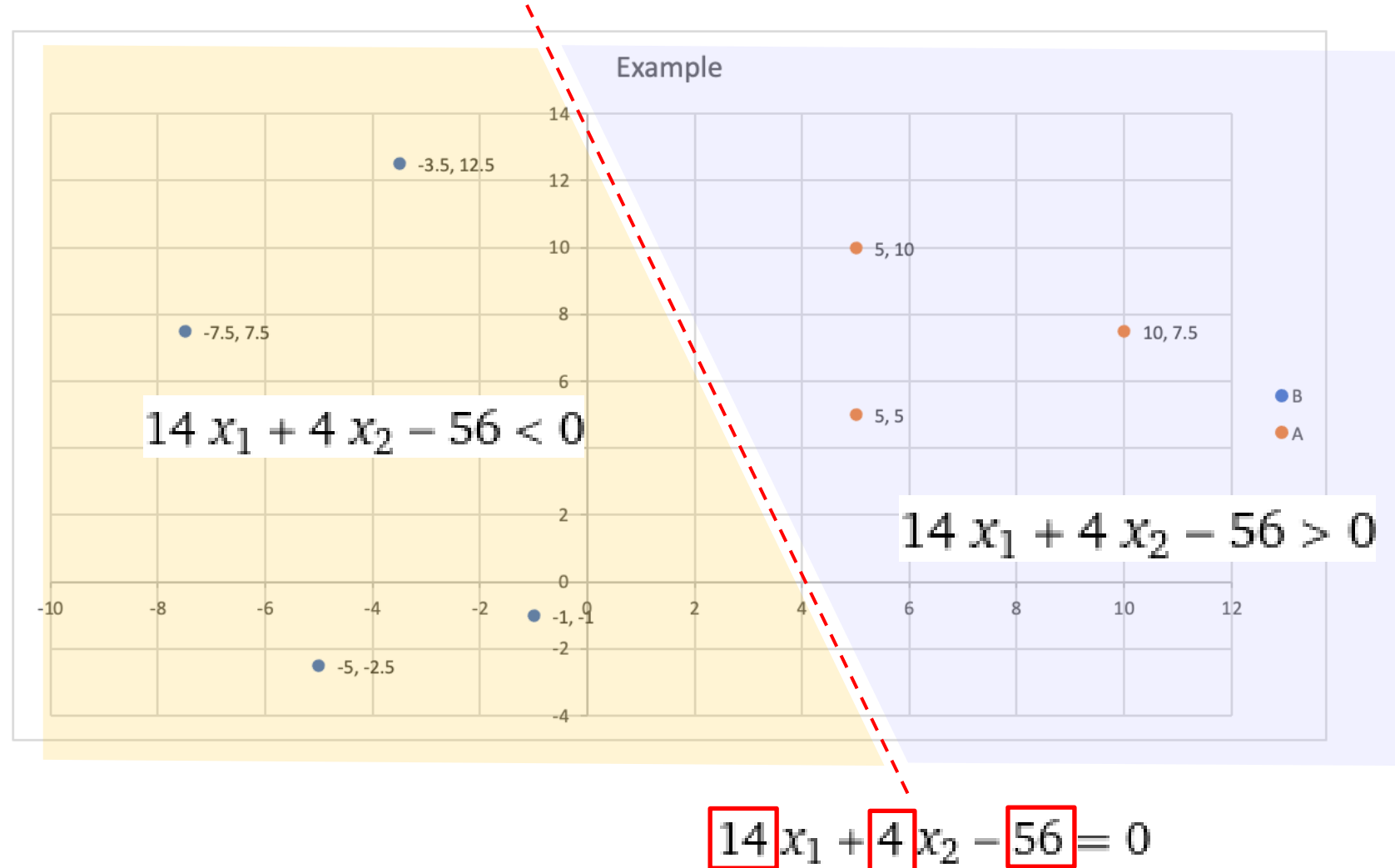
The subscript T means transpose.

Both of them are presenting the second sample for X.  $\mathbf{x}_i = \mathbf{X}(i,:)^T$

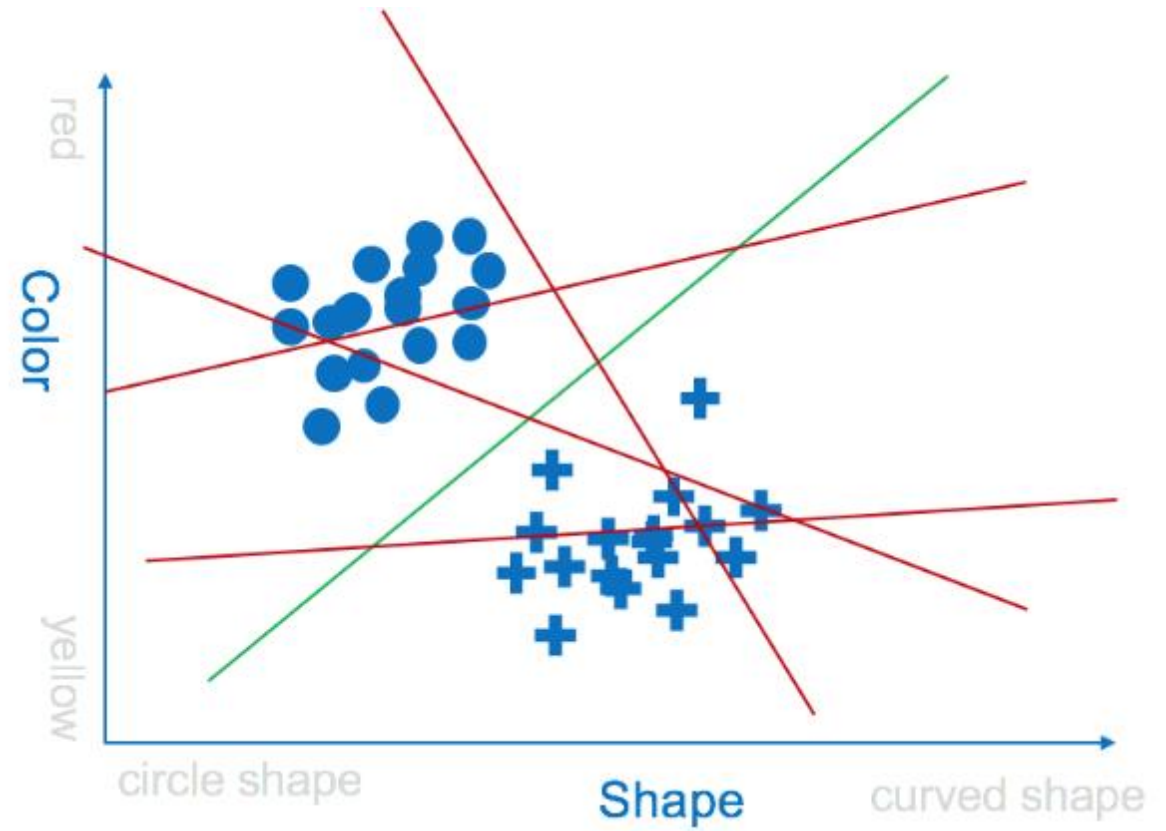
# Linear Classification Task – An Illustrated Example

*How about non-linear?*

$x_1$	$x_2$	$y$
-1	-1	0
-5	-2.5	0
-7.5	7.5	0
10	7.5	1
-2.5	12.5	0
5	10	1
5	5	1



# Perceptron Algorithm

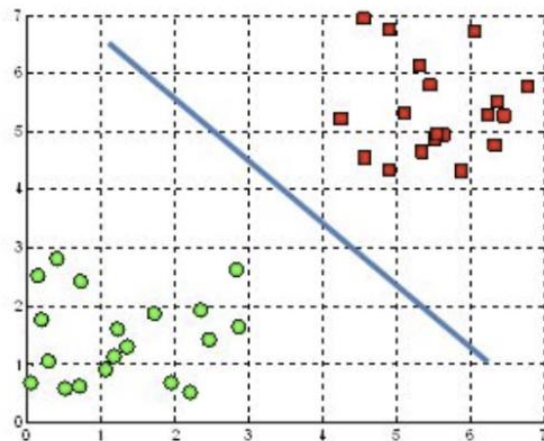




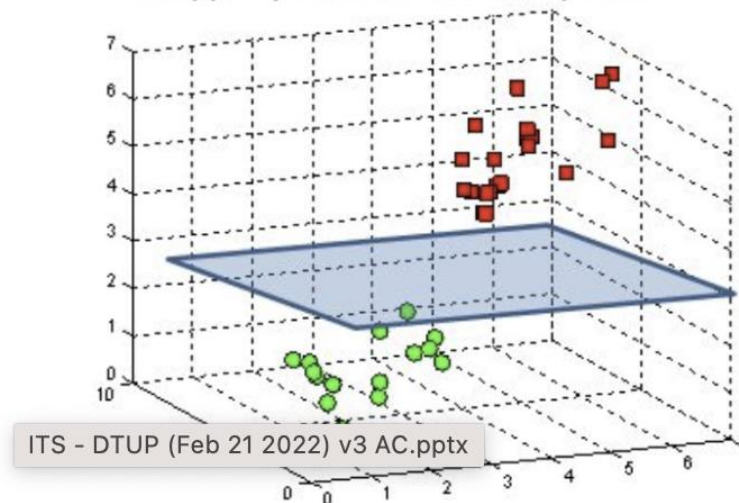
# Introduction to Perceptron Algorithm

- One of the earliest algorithms for linear classification (Rosenblatt, 1958)
- Try to find a **hyperplane** separating the labeled data
- Guaranteed to find a separating hyperplane if the data is linearly separable

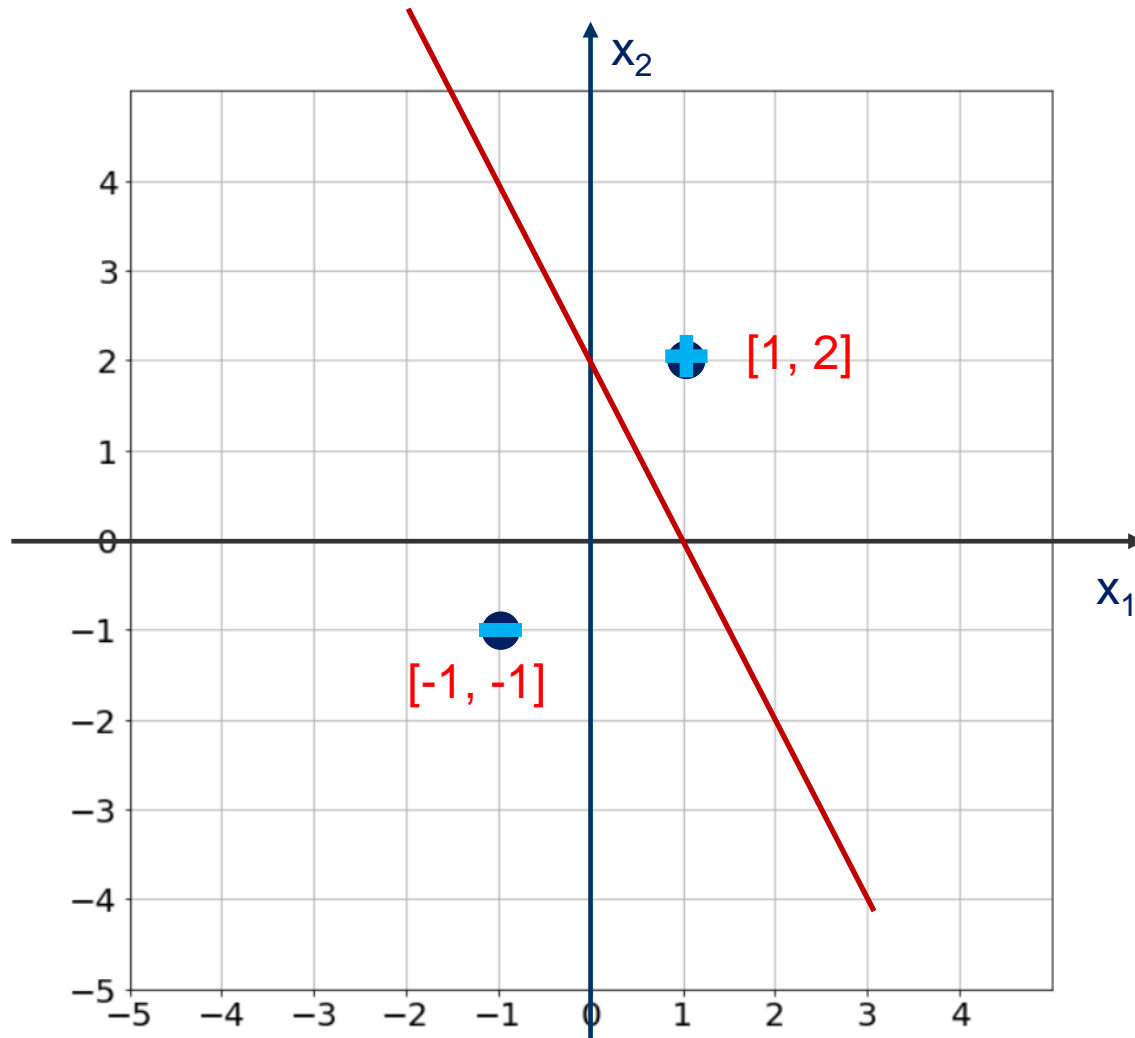
A hyperplane in  $\mathbb{R}^2$  is a line



A hyperplane in  $\mathbb{R}^3$  is a plane



# Linear Decision Boundary for Classification: Example



- What is the formula for this linear boundary?
  - $x_2 = -2x_1 + 2 \Rightarrow 2x_1 + x_2 - 2 = 0$
  - General form:
$$f(\mathbf{x}) = \sum_{j=1}^d w_j x_j + b = \mathbf{w}^T \mathbf{x} + b = 0$$
- What label would we predict for a new data point  $\mathbf{x}$ ?
  - $[1, 2]$ 
    - $2 \cdot 1 + 2 - 2 = 2 > 0$
    - predicted it as **positive**
  - $[-1, -1]$ 
    - $2 \cdot -1 + -1 - 2 = -5 < 0$
    - Predicted it as **negative**

# Introduction to Perceptron Algorithm

## ■ Problem Definition

- Given a training dataset  $\{\mathbf{x}_i, y_i\}_{i=1}^m$ , where  $\mathbf{x}_i$  is a  **$n$  dimensional input feature vector**,  $y_i \in \{-1, 1\}$  is the corresponding **class label**.
- Objective: Train a linear classifier that can separate positive and negative samples.
- **Training:** Learn a linear classification function  **$f(\mathbf{x}) = \sum_{j=1}^n w_j x_j + b$**  from training data
  - $y = 1$  if  $f(\mathbf{x}) = \sum_{j=1}^n w_j x_j + b > 0$
  - $y = -1$  if  $f(\mathbf{x}) = \sum_{j=1}^n w_j x_j + b < 0$
  - $w_j, b$  are the **model parameters** that we need to learn from training data
- **Prediction:** for any new input  $\mathbf{x}$ , predict its class label as  $y = \text{sign}(f(\mathbf{x}))$

## ■ Simplify the notation

- By introducing an artificial feature  **$x_0 = 1$** ,  $\mathbf{x} \rightarrow [x_0, x_1, \dots, x_n]$ ,  $f(\mathbf{x})$  can be rewritten in vector form as

$$f(\mathbf{x}) = \sum_{j=1}^n w_j x_j + b = \sum_{j=1}^n w_j x_j + b x_0 = \sum_{j=0}^n w_j x_j = \mathbf{w}^T \mathbf{x}$$

# Introduction to Perceptron Algorithm

Initialize  $\mathbf{w} = \mathbf{0}$

Repeat

if  $y_i(\mathbf{w}^T \mathbf{x}_i) \leq 0$  then

$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$

end if

*Iteratively do prediction on the training data,  
If the current data point is correctly classified*

**do nothing**

*If the current point is wrongly classified*

**Update the model**

$y_i(\mathbf{w}^T \mathbf{x}_i) \leq 0$  means the training data point  $\mathbf{x}_i$  is misclassified.

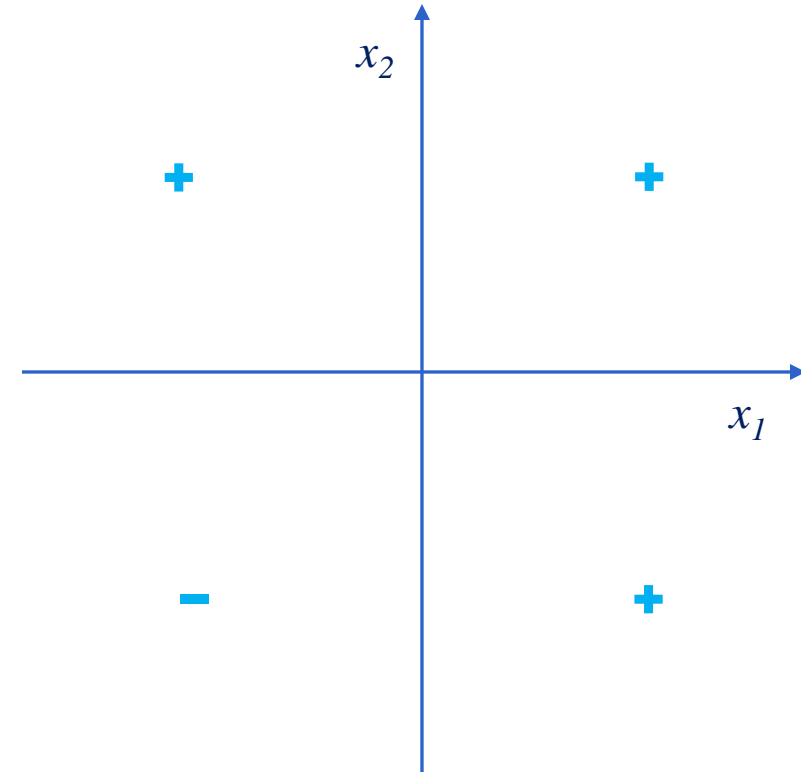
- true label  $y_i = 1$ , but the prediction  $\text{sign}(\mathbf{w}^T \mathbf{x}) = -1$ , or
- true label  $y_i = -1$ , but the prediction  $\text{sign}(\mathbf{w}^T \mathbf{x}) = 1$

# Perceptron: An Illustrated Example

Suppose we have a small training data with only 4 labelled data samples. Each data sample has only two features.

$x_1$	$x_2$	$y$
1	-1	1
1	1	1
-1	1	1
-1	-1	-1

$x_0$	$x_1$	$x_2$	$y$
1	1	-1	1
1	1	1	1
1	-1	1	1
1	-1	-1	-1



# An Illustrated Example

- Initialize  $\mathbf{w} = [0, 0, 0]^T$
- Cyclically go through the data

The 1<sup>st</sup> pass

$$([1, 1, -1], 1): y_i(\mathbf{w}^T \mathbf{x}_i) = 1 * (0 * 1 + 0 * 1 + 0 * (-1)) = 0 \leq 0$$

$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i = [0, 0, 0]^T + 1 * [1, 1, -1]^T = [1, 1, -1]^T$$

$$([1, 1, 1], 1): y_i(\mathbf{w}^T \mathbf{x}_i) = 1 * (1 * 1 + 1 * 1 + (-1) * 1) = 1 > 0$$

do not update  $\mathbf{w}$

$$([1, -1, 1], 1): y_i(\mathbf{w}^T \mathbf{x}_i) = 1 * (1 * 1 + 1 * (-1) + (-1) * 1) = -1 \leq 0$$

$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i = [1, 1, -1]^T + 1 * [1, -1, 1]^T = [2, 0, 0]^T$$

$$([1, -1, -1], -1): y_i(\mathbf{w}^T \mathbf{x}_i) = -1 * (2 * 1 + 0 * (-1) + 0 * (-1)) = -2 \leq 0$$

$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i = [2, 0, 0]^T + (-1) * [1, -1, -1]^T = [1, 1, 1]^T$$

The 2<sup>nd</sup> pass

$$([1, 1, -1], 1): y_i(\mathbf{w}^T \mathbf{x}_i) = 1 * (1 * 1 + 1 * 1 + 1 * (-1)) = 1 > 0$$

do not update  $\mathbf{w}$

$$([1, 1, 1], 1): y_i(\mathbf{w}^T \mathbf{x}_i) = 1 * (1 * 1 + 1 * 1 + 1 * (1)) = 3 > 0$$

do not update  $\mathbf{w}$

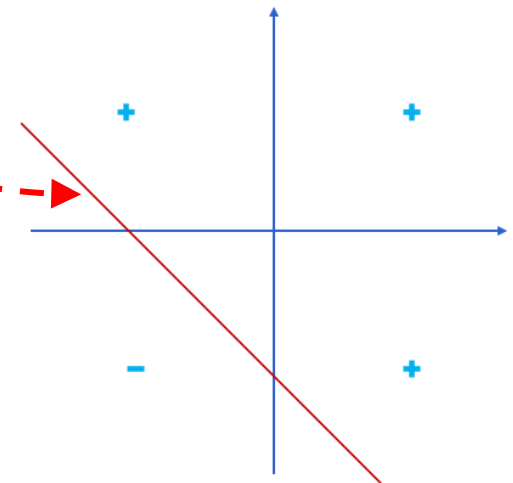
$$([1, -1, 1], 1): y_i(\mathbf{w}^T \mathbf{x}_i) = 1 * (1 * 1 + 1 * (-1) + 1 * (1)) = 1 > 0$$

do not update  $\mathbf{w}$

$$([1, -1, -1], -1): y_i(\mathbf{w}^T \mathbf{x}_i) = -1 * (1 * 1 + 1 * (-1) + 1 * (-1)) = 1 > 0$$

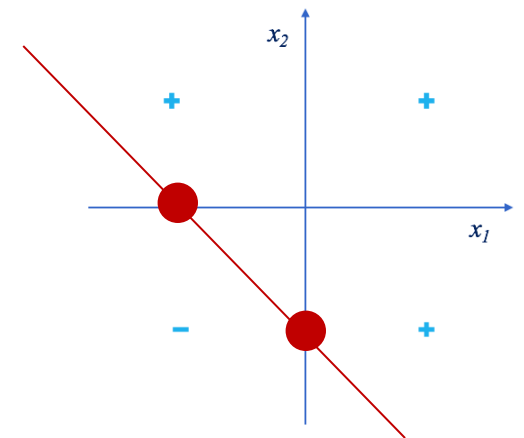
do not update  $\mathbf{w}$

$x_0$	$x_1$	$x_2$	$y$
1	1	-1	1
1	1	1	1
1	-1	1	1
1	-1	-1	-1



# How to Plot the Decision Boundary?

- The learnt  $\mathbf{w}$  by using perceptron algorithm is  $[1, 1, 1]^T$ , then the corresponding decision boundary is  $w_0 * x_0 + w_1 * x_1 + w_2 * x_2 = 0$ .
- We know that  $w_0$  is 1 and  $x_0$  is an artificial feature which always equal to 1. Also, we know  $w_1 = 1$  and  $w_2 = 1$ . Therefore, the decision boundary is:  
$$1 + x_1 + x_2 = 0 \rightarrow x_1 + x_2 = -1$$
- The line (i.e. decision boundary) can be plotted by connecting two data points lies on the line.
  - Suppose  $x_1 = 0$ , then  $x_2 = -1$
  - Suppose  $x_2 = 0$  then  $x_1 = -1$



# Why perceptron works?

- Some preliminary on vectors

- The magnitude  $\|\mathbf{a}\|$  ( norm ) of a vector  $\mathbf{a} = (a_1, a_2)$  is

$$\|\mathbf{a}\| = \sqrt{a_1^2 + a_2^2}$$

- Vector addition:  $\mathbf{a} = (a_1, a_2)$ ,  $\mathbf{b} = (b_1, b_2)$

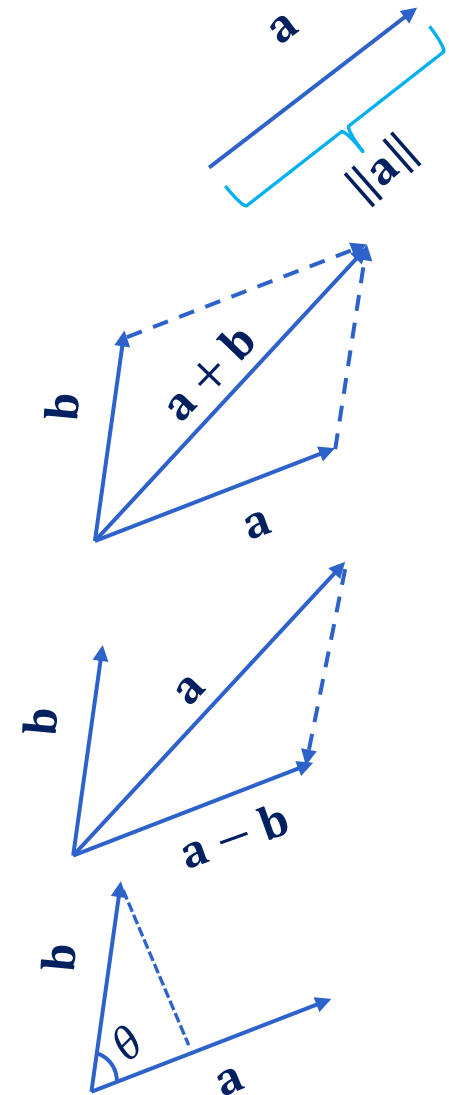
- $\mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2)$

- Vector subtraction:  $\mathbf{a} = (a_1, a_2)$ ,  $\mathbf{b} = (b_1, b_2)$

- $\mathbf{a} - \mathbf{b} = (a_1 - b_1, a_2 - b_2)$

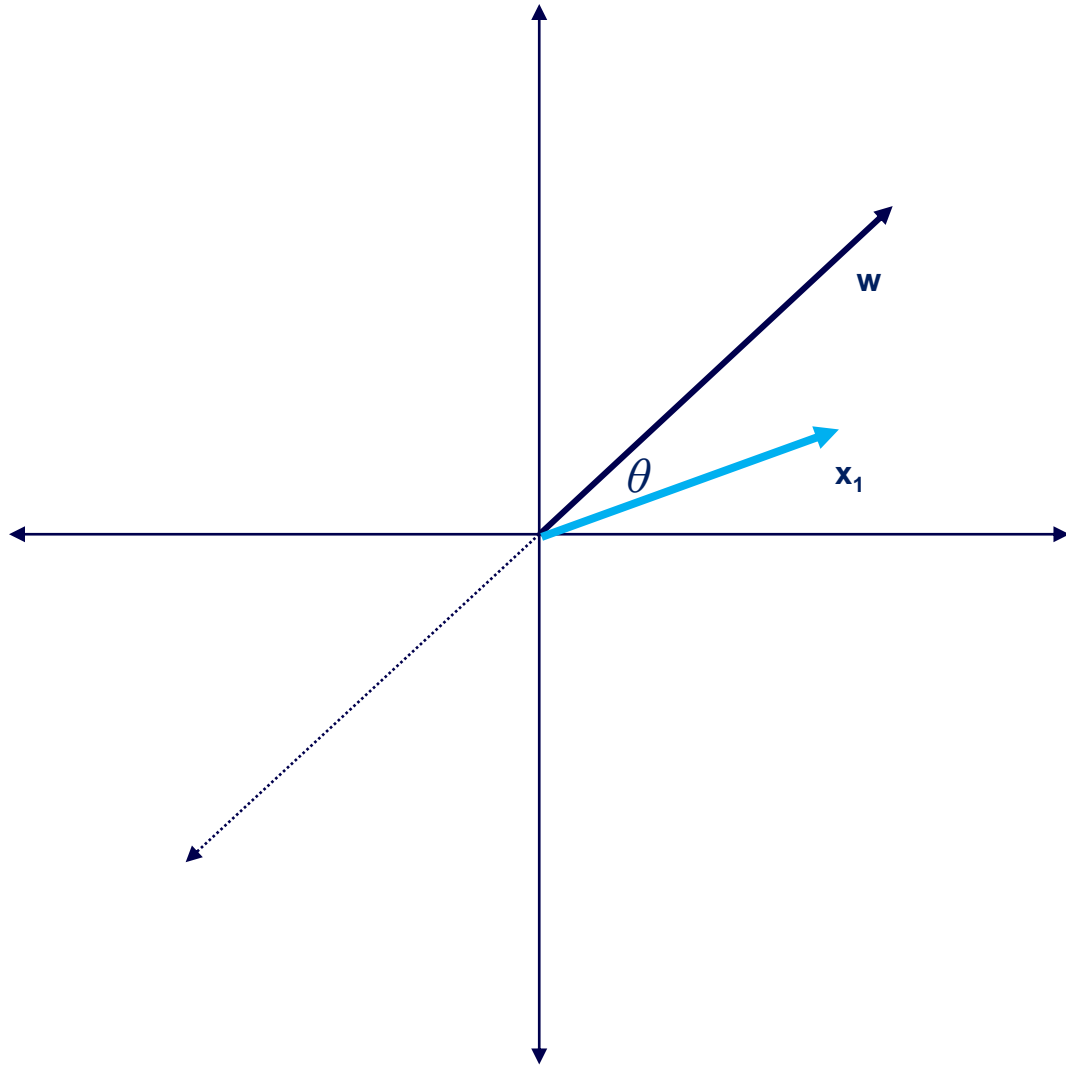
- Dot product:  $\mathbf{a} = (a_1, a_2)$ ,  $\mathbf{b} = (b_1, b_2)$

- $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}\mathbf{b} = a_1b_1 + a_2b_2 = \|\mathbf{a}\|\|\mathbf{b}\| \cos \theta$

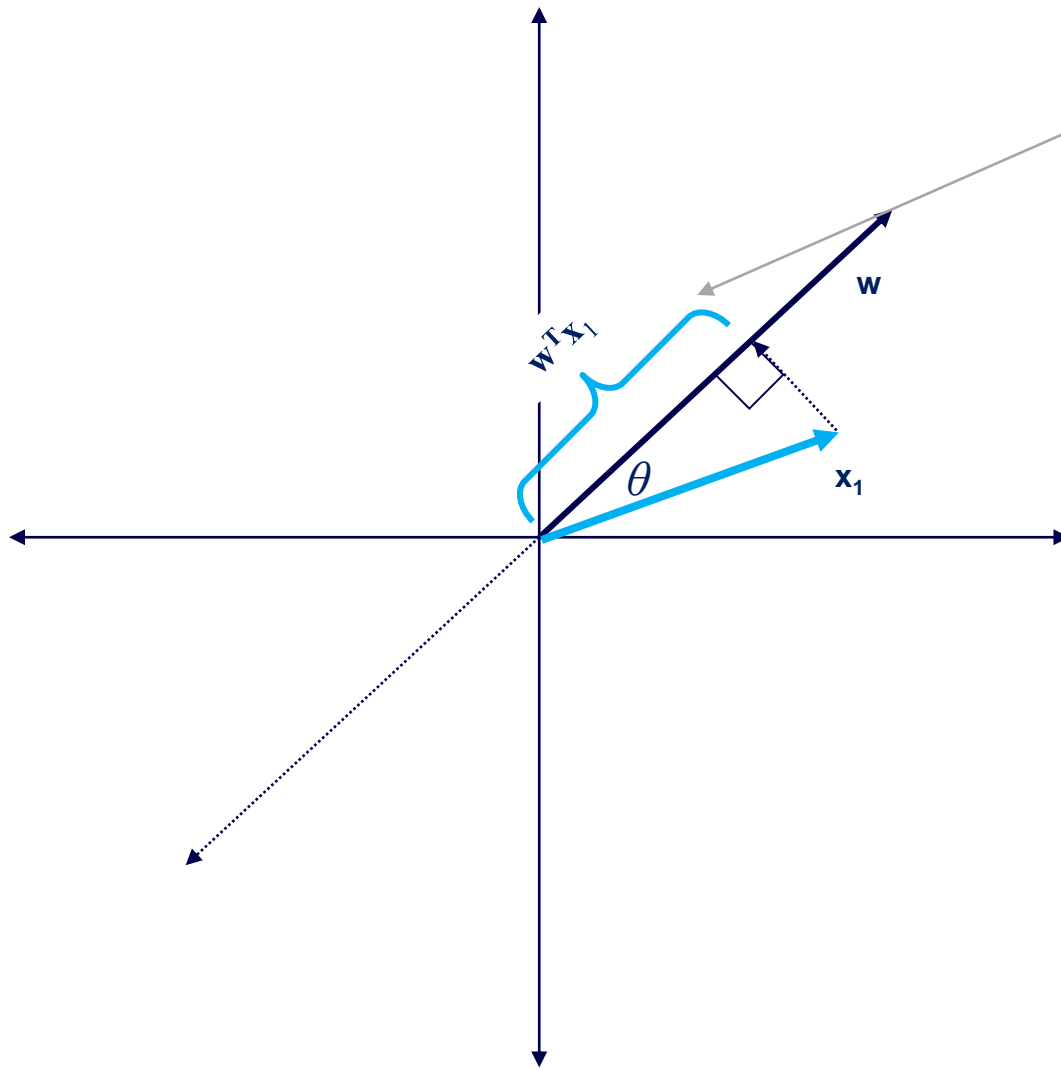




# Why Perceptron Works?

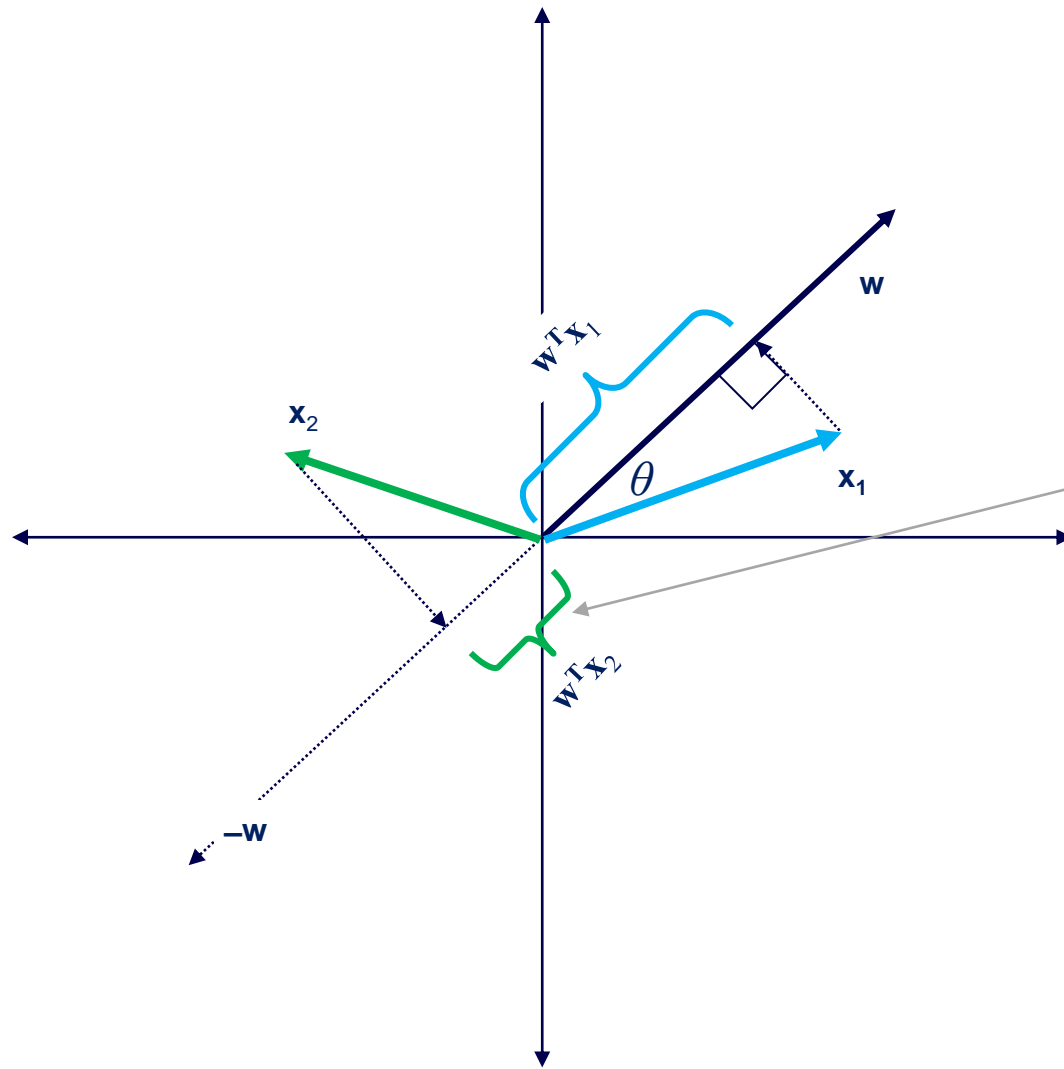


# Why Perceptron Works?



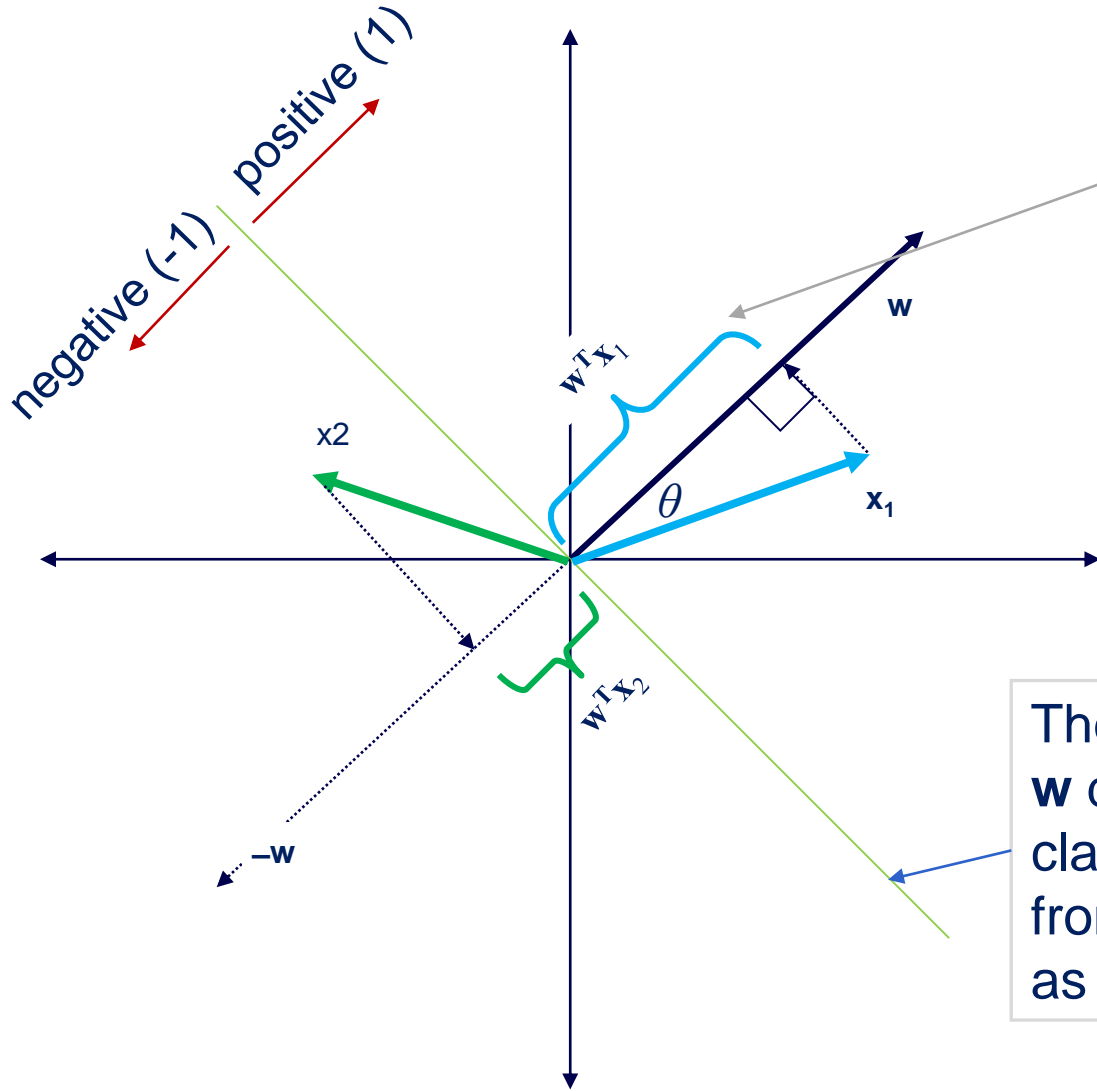
Without loss of generality, we assume  $\|w\| = 1$ .  
Visually,  $w^T x = \|w\| \|x\| \cos \theta$  is the scalar value you get if you “project  $x$  onto  $w$ ”.  
If the value is positive, predict it as positive.

# Why Perceptron Works?



Without loss of generality, we assume  $\|\mathbf{w}\| = 1$ .  
Visually,  $\mathbf{w}^T \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta$  is the scalar value you get if you “project  $\mathbf{x}$  onto  $\mathbf{w}$ ”.  
If the value is positive, predict it as positive.  
If the value is negative, predict it as negative.

# Why Perceptron Works?



Without loss of generality, we assume  $\|\mathbf{w}\| = 1$ .  
Visually,  $\mathbf{w}^T \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta$  is the scalar value you get if you “project  $\mathbf{x}$  onto  $\mathbf{w}$ ”.  
If the value is positive, predict it as positive.  
If the value is negative, predict it as negative.

The line perpendicular to  $\mathbf{w}$  divides the vectors classified as positive (1) from the vectors classified as negative -1.

# Why Perceptron Works?

- Consider a positive data point ( $y_i = 1$ ) was wrongly classified as negative
  - Perceptron predict as  $-1$  since  $\mathbf{w}^T \mathbf{x}_i < 0$

- Model Update

$$\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + y_i \mathbf{x}_i$$

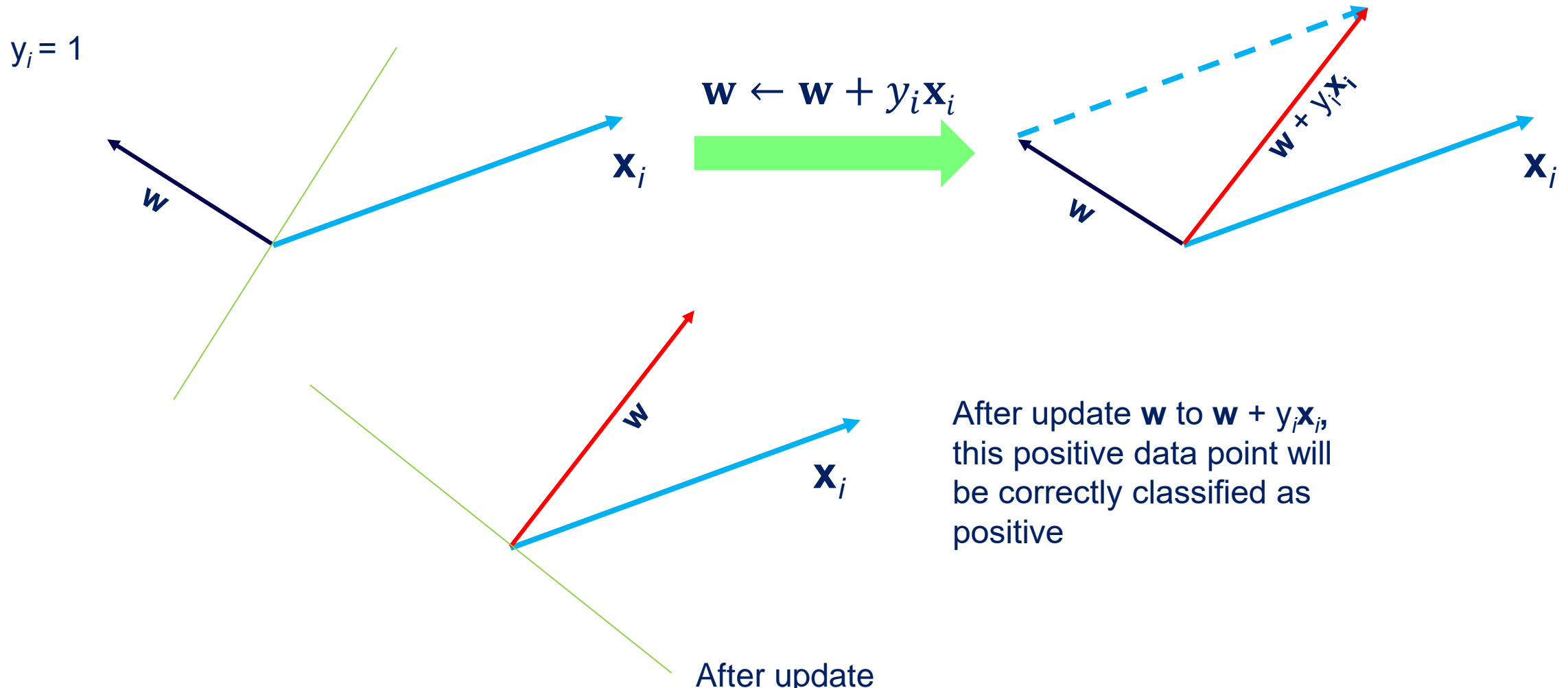
Note that,

$$\begin{aligned} \mathbf{w}_{new}^T \mathbf{x}_i &= (\mathbf{w}_{old} + y_i \mathbf{x}_i)^T \mathbf{x}_i \\ &= \mathbf{w}_{old}^T \mathbf{x}_i + \mathbf{x}_i^T \mathbf{x}_i \end{aligned} \quad \rightarrow \quad \|\mathbf{x}_i\|_2^2 > 0$$

- Therefore,  $\mathbf{w}_{new}^T \mathbf{x}_i$  is **less negative** than  $\mathbf{w}_{old}^T \mathbf{x}_i$ 
  - The update makes the classifier more correct on this data point  $(\mathbf{x}_i, y_i)$

# Why Perceptron Works? (Visually)

- Consider a positive data point was wrongly classified as negative



# Why Perceptron Works?

- Consider a negative data point ( $y_i = -1$ ) was wrongly classified as positive
  - Perceptron predict as 1 since  $\mathbf{w}^T \mathbf{x}_i > 0$

- Model Update

$$\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + y_i \mathbf{x}_i$$

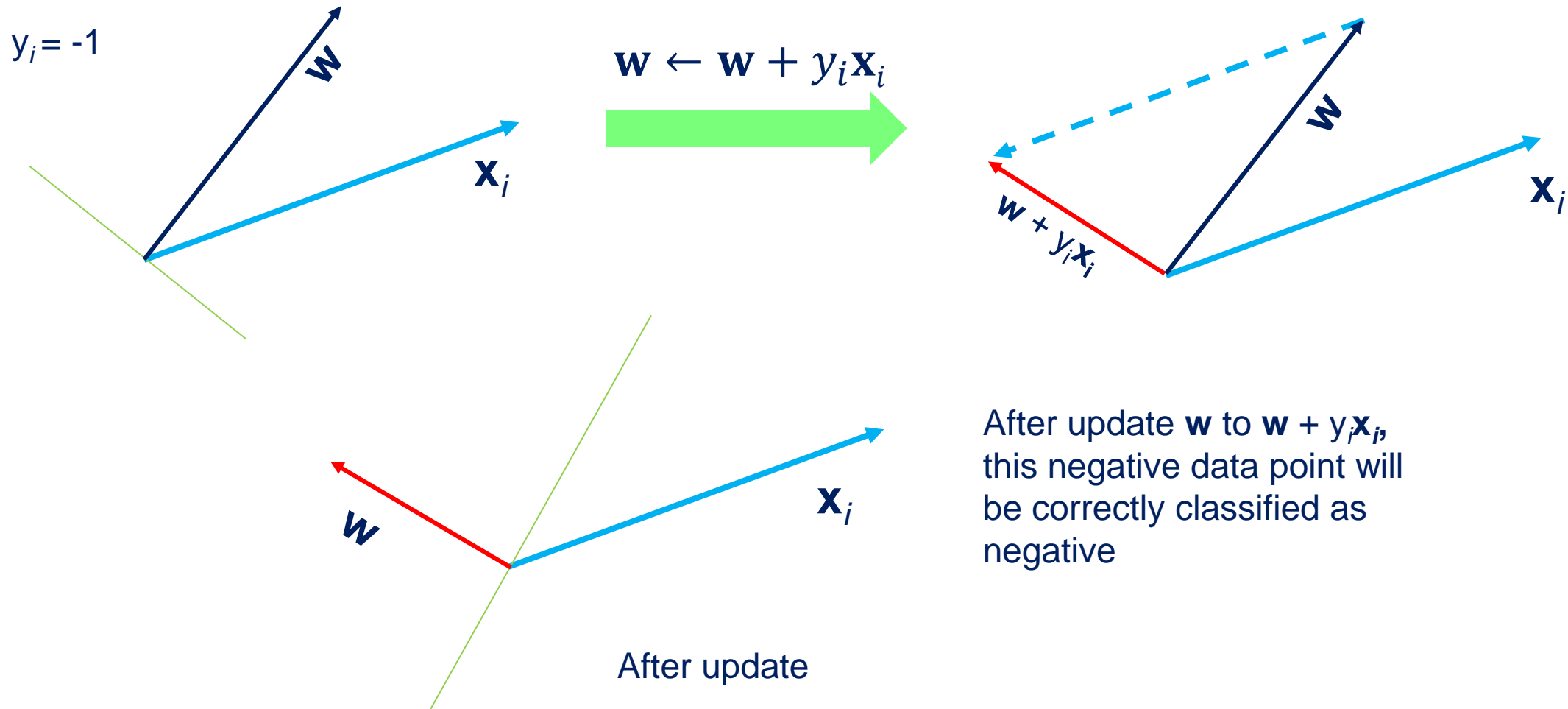
Note that,

$$\begin{aligned} \mathbf{w}_{new}^T \mathbf{x}_i &= (\mathbf{w}_{old} + y_i \mathbf{x}_i)^T \mathbf{x}_i \\ &= \mathbf{w}_{old}^T \mathbf{x}_i - \boxed{\mathbf{x}_i^T \mathbf{x}_i} \end{aligned} \quad \rightarrow \quad \|\mathbf{x}_i\|_2^2 > 0$$

- Therefore,  $\mathbf{w}_{new}^T \mathbf{x}_i$  is **less positive** than  $\mathbf{w}_{old}^T \mathbf{x}_i$ 
  - The update makes the classifier more correct on this data point  $(\mathbf{x}_i, y_i)$

# Why Perceptron Works? (Visually)

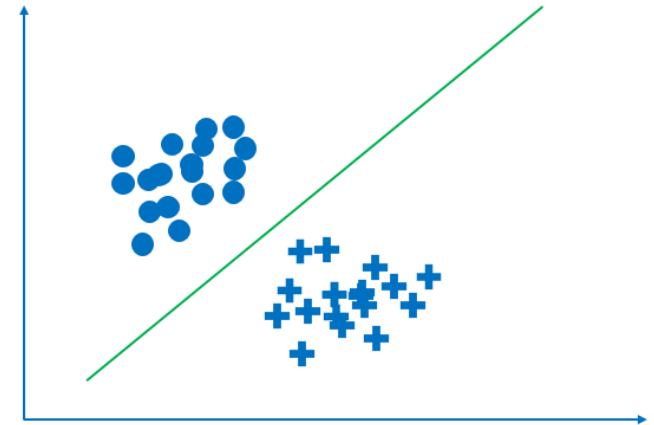
- Consider a negative data point was wrongly classified as positive



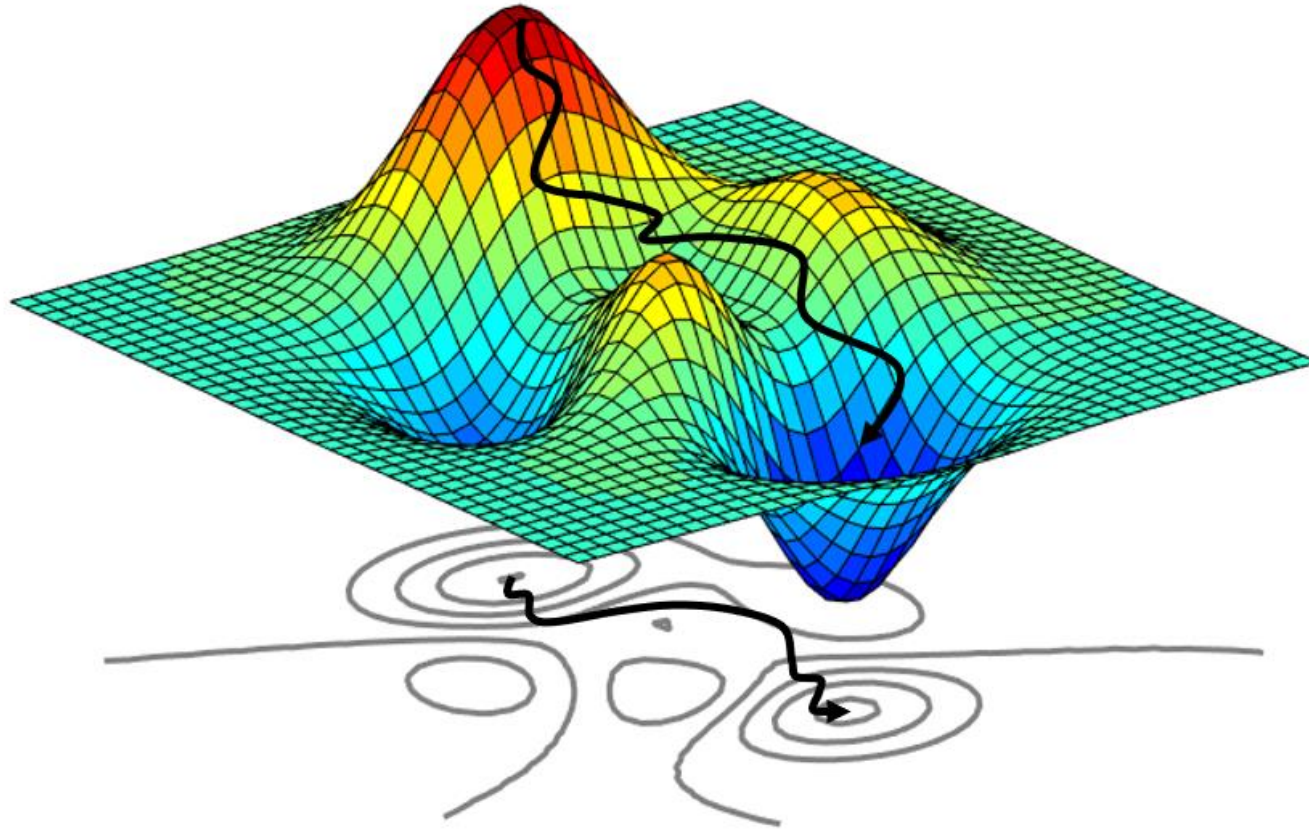


# Perceptron: Some Additional Notes

- Given a linearly separable training set  $\{\mathbf{x}_i, y_i\}_{i=1}^m$ , perceptron algorithm finds a classification model with “zero training error”.
- We can formulate the goal of learning of the separating hyperplane as an optimization problem (*minimizing a cost (loss) function related to the training error*).



# Learning as Optimization

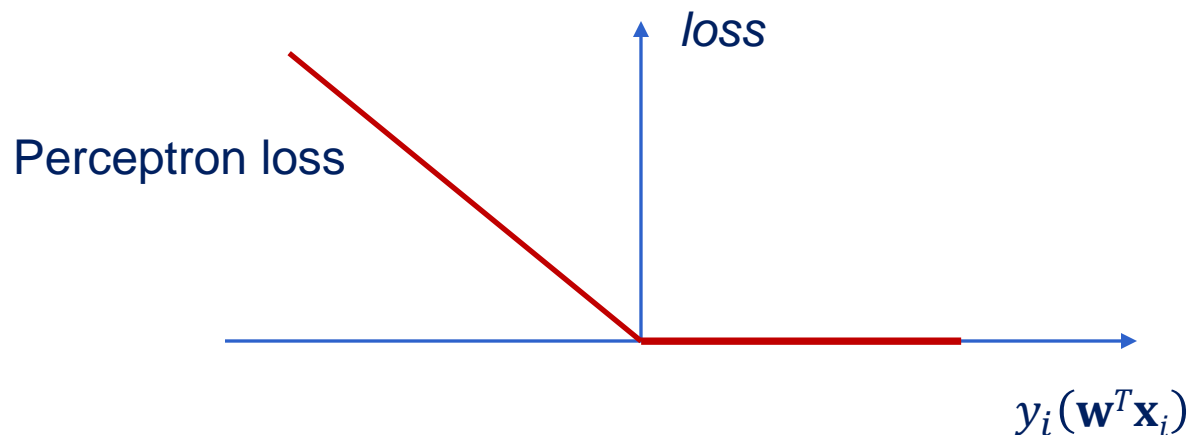


# Learning as Optimization

- The cost (loss) function of perceptron:

$$l(\mathbf{w}) = \sum_{i=1}^m \max\{0, -y_i(\mathbf{w}^T \mathbf{x}_i)\}$$

- Loss = 0 on samples where Perceptron prediction is **correct**, i.e.,  $y_i(\mathbf{w}^T \mathbf{x}_i) > 0$
- Loss > 0 on samples where Perceptron prediction is **wrong**, i.e.,  $y_i(\mathbf{w}^T \mathbf{x}_i) < 0$



# Learning as Optimization

- Therefore, the classification model  $\mathbf{w}$  is by *minimizing the perceptron cost function*:

$$\min_{\mathbf{w}} l(\mathbf{w}) = \sum_{i=1}^m \max\{0, -y_i(\mathbf{w}^T \mathbf{x}_i)\}$$

learning rate

- Gradient Descent:  $\mathbf{w} = \mathbf{w} - \eta \frac{\partial l(\mathbf{w})}{\partial \mathbf{w}}$  gradient

- Gradient is computed based on the entire training data

- Stochastic Gradient Descent

- Works like gradient descent, now the ('stochastic') gradient is computed based on only one data sample

$$\mathbf{w} = \mathbf{w} - \eta \frac{\partial \max\{0, -y_i(\mathbf{w}^T \mathbf{x}_i)\}}{\partial \mathbf{w}} \rightarrow \begin{cases} 0 & \text{if } y_i(\mathbf{w}^T \mathbf{x}_i) > 0 \\ -y_i \mathbf{x}_i & \text{if } y_i(\mathbf{w}^T \mathbf{x}_i) \leq 0 \end{cases}$$

By setting  $\eta = 1$ , it returns the perceptron updating rule:

$$\begin{aligned} &\text{If } y_i(\mathbf{w}^T \mathbf{x}_i) \leq 0 \\ &\quad \mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i \end{aligned}$$

# Learning as Optimization

- **Optimization** is one of the core components of machine learning algorithms.
- Most machine learning algorithms is to learn the parameters by minimizing the cost function based on training data.

- Linear Regression:

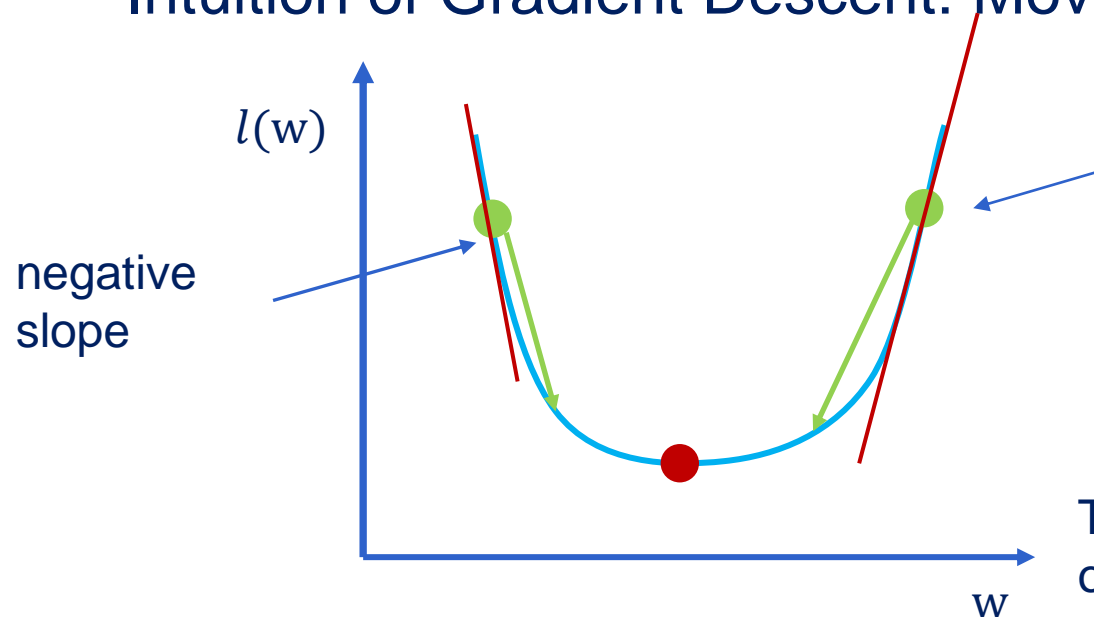
$$\min_{\mathbf{w}} l(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^m ((\mathbf{w}^T \mathbf{x}_i + b) - y_i)^2$$

- Support Vector Machine:

$$\begin{aligned} \min \quad & \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^m \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, \dots, m \\ & \xi_i \geq 0 \end{aligned}$$

# Recap: Optimization using Gradient Descent

- **Gradient Descent** is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the *negative* of the gradient. It is a general algorithm that can be applied to many machine learning model.
- Intuition of Gradient Descent: Moving in the direction of steepest descent



What is the gradient of  $l(w)$  at this point?

- The gradient of the function at this point is the slope of the tangent line (i.e., the straight line that "just touches" the curve at this point)
- This is a positive slope (increasing)
- Therefore, we need to go to opposite direction.

The gradient determines the direction of steepest increase of  $l(w)$ . We need to go to opposite direction of gradient to minimize the  $l(w)$ .

# Gradient Descent Algorithm

## Gradient Descent

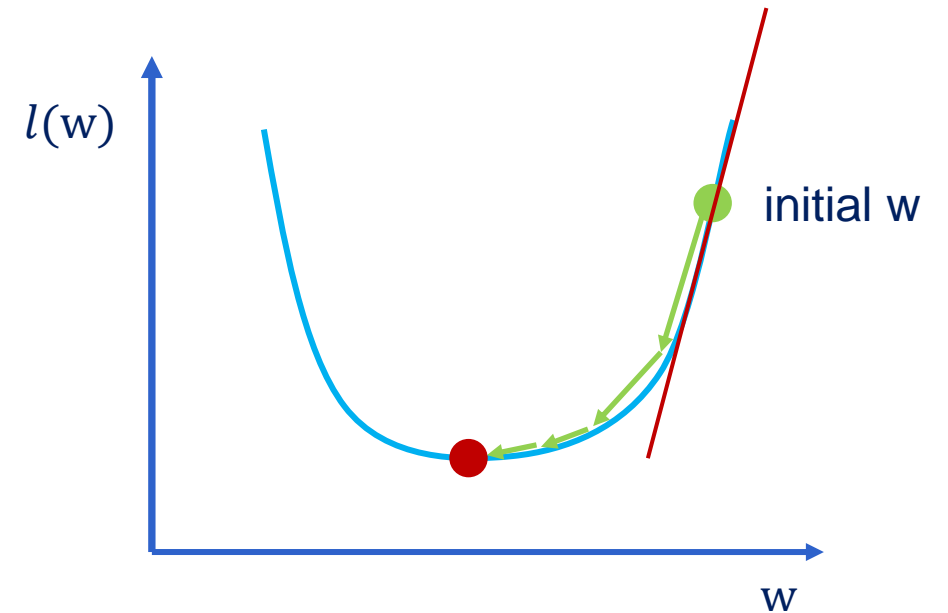
Initialize  $\mathbf{w}$

Repeat  $T$  times

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial l(\mathbf{w})}{\partial \mathbf{w}}$$

Output:  $\mathbf{w}$

Learning rate (e.g., 0.1)      gradient



# Stochastic Gradient Descent

- Works like gradient descent, now the ('stochastic') gradient is computed based on only one data sample

## Stochastic Gradient Descent

Initialize  $\mathbf{w}$

Repeat  $T$  times

For a data sample  $(\mathbf{x}_i, y_i)$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial l(\mathbf{w}, \mathbf{x}_i, y_i)}{\partial \mathbf{w}}$$

Output:  $\mathbf{w}$

Learning rate (e.g., 0.1)

stochastic gradient based on one data sample



# Perceptron: additional notes



**Frank Rosenblatt**  
1928–1969

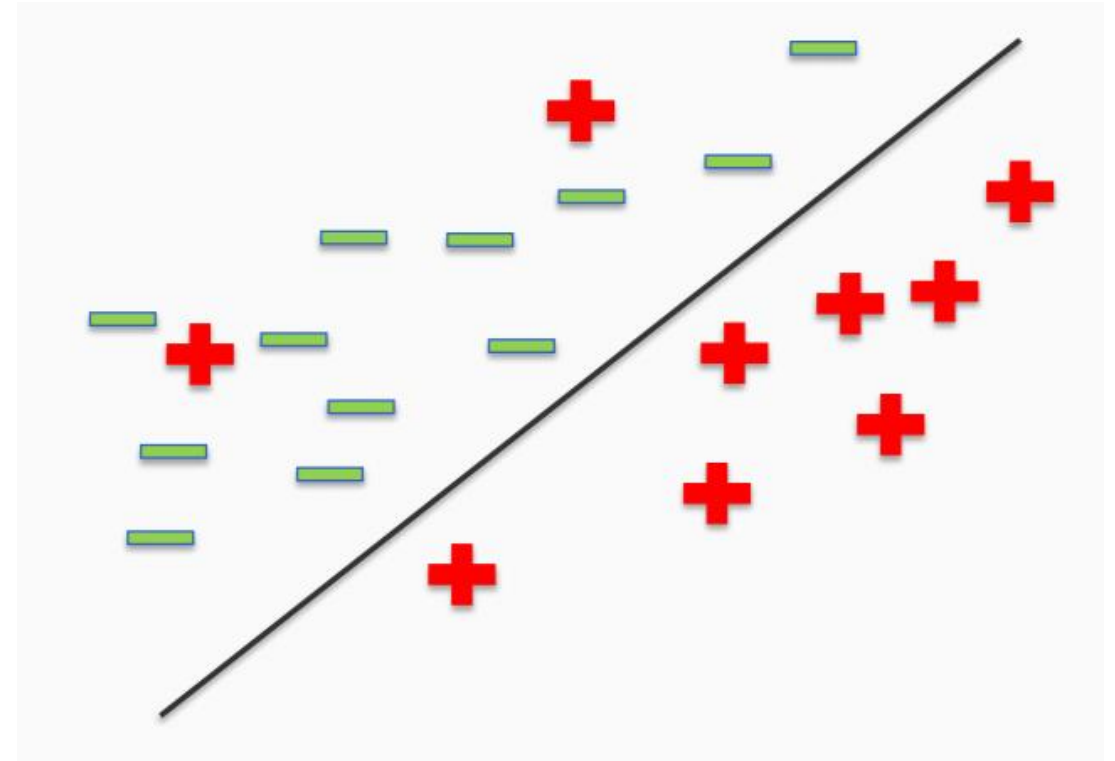
Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minsky, whose objections were published in the book "Perceptrons", co-authored with

Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.

Perceptron was criticized for its inability to handle nonlinearly separable problem (e.g. XOR function).

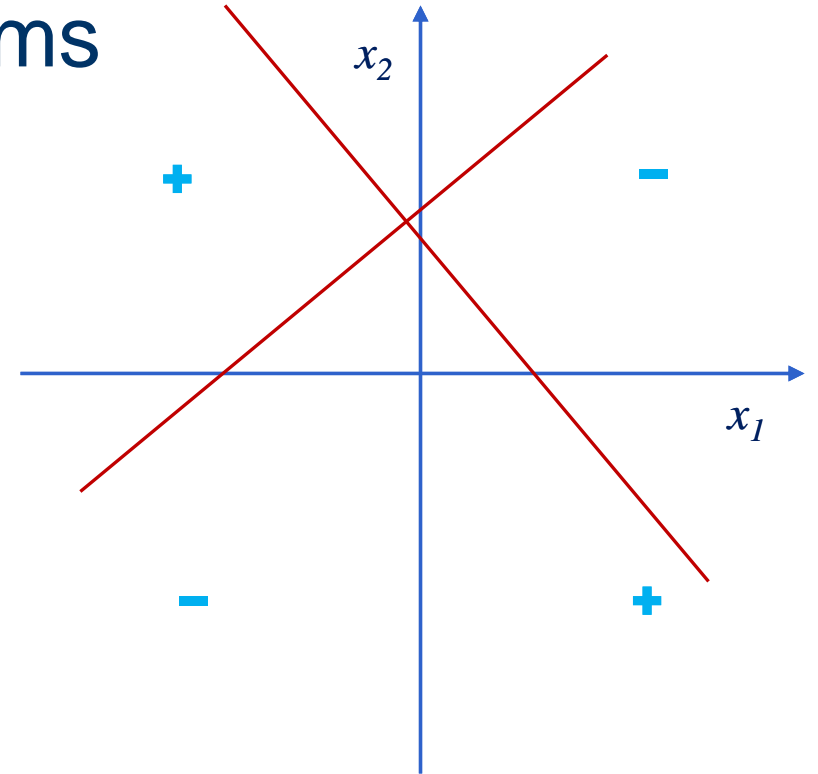
# If the data is not linearly separable

- The perceptron algorithm would not converge if the training data is not linearly separable.



# Perceptron: Nonlinear separable problems

$x_1$	$x_2$	$y$
1	-1	1
1	1	-1
-1	1	1
-1	-1	-1



- Perceptron can not solve nonlinear separable problems
- Nonlinear separable problems can be solved by
  - Using multiple layers perceptron (**Artificial Neural Networks**)
  - Making it linearly separable using kernel (**Support Vector Machine**)