Neural Networks (CSC372)

Unit 2: Rosenblatt's Perceptron

(3 Hrs.)

Reference: Simon Haykin (3rd Edition)

By Kiran Bagale,

Department of IT, 2025

Learning Objectives

- Understand the structure and working of Rosenblatt's Perceptron
- Explain the Perceptron Convergence Theorem
- Compare Perceptron with Bayes Classifier
- Apply the Batch Perceptron Algorithm for classification tasks

Before going to Rosenblatt's Perceptron

- In the formative years of neural networks (1943–1958), several researchers stand out for their pioneering contributions:
  - McCulloch and Pitts (1943) for introducing the idea of neural networks as computing machines.
  - Hebb(1949) for postulating the first rule for self-organized learning.
  - Rosenblatt (1958) for proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).

Introduction to Perceptron

- Developed by Frank Rosenblatt (1958)
- Inspired by the biological neuron
- Simplest form of neural Network
- Used for binary classification tasks
- Processes input via weighted sum and activation function

Assumptions

1. Binary classification $(i.e. y_i \in \{-1, +1\})$
2. Data is linearly separable

Perceptron Model

- Rosenblatt's perceptron is built around a nonlinear neuron, namely, the *McCulloch–Pitts model* of a neuron.

Input vector: $x = [x_1, x_2, \ldots, x_m]$
Weight vector: $w = [w_1, w_2, \ldots, w_m]$
Output: $y = \varphi(w \cdot x + b)$

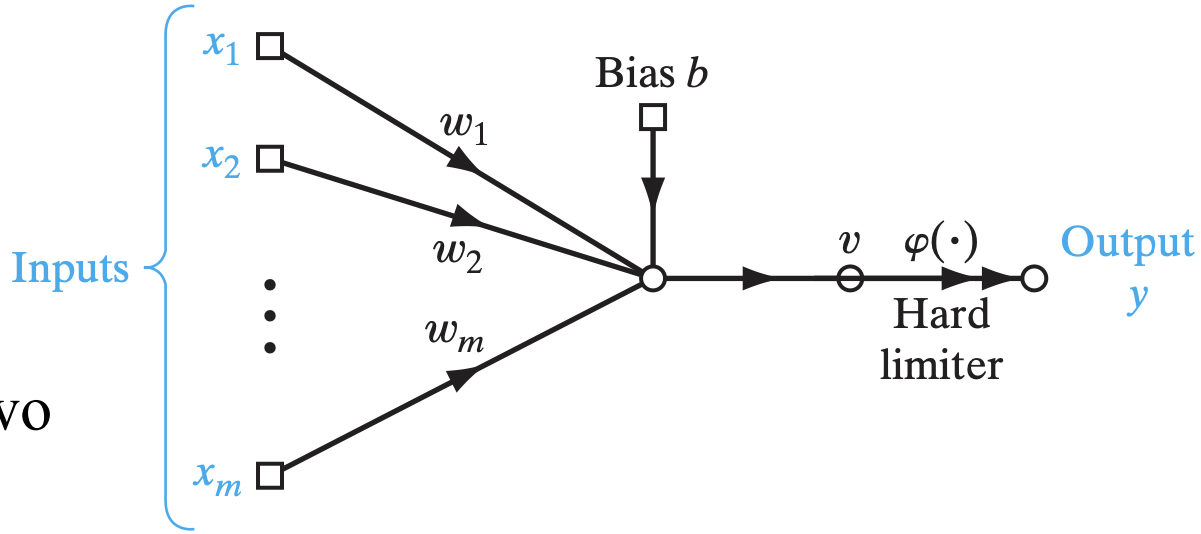


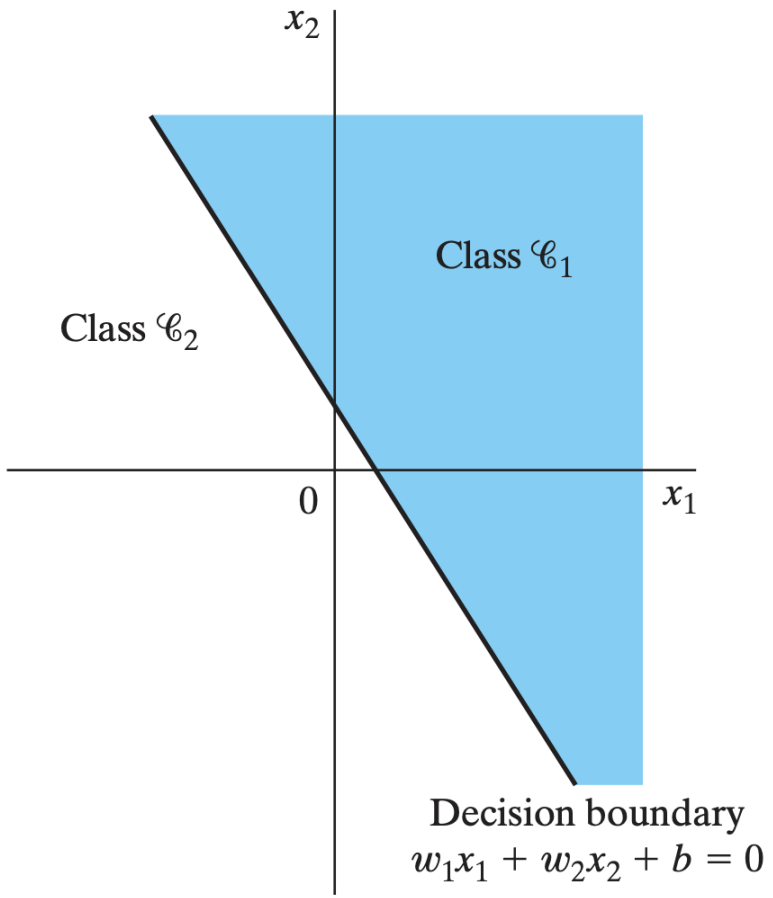
FIG 2.1: Signal-flow graph of the perceptron.

Perceptron Goal :
Classify input vector (x1,x2,...,xm) into two classes:

- If output y=+1, assign to class **c1**
- If output y=−1, assign to class **c2**

Learning rule: *Adjust weights to minimize classification error*

Classifier



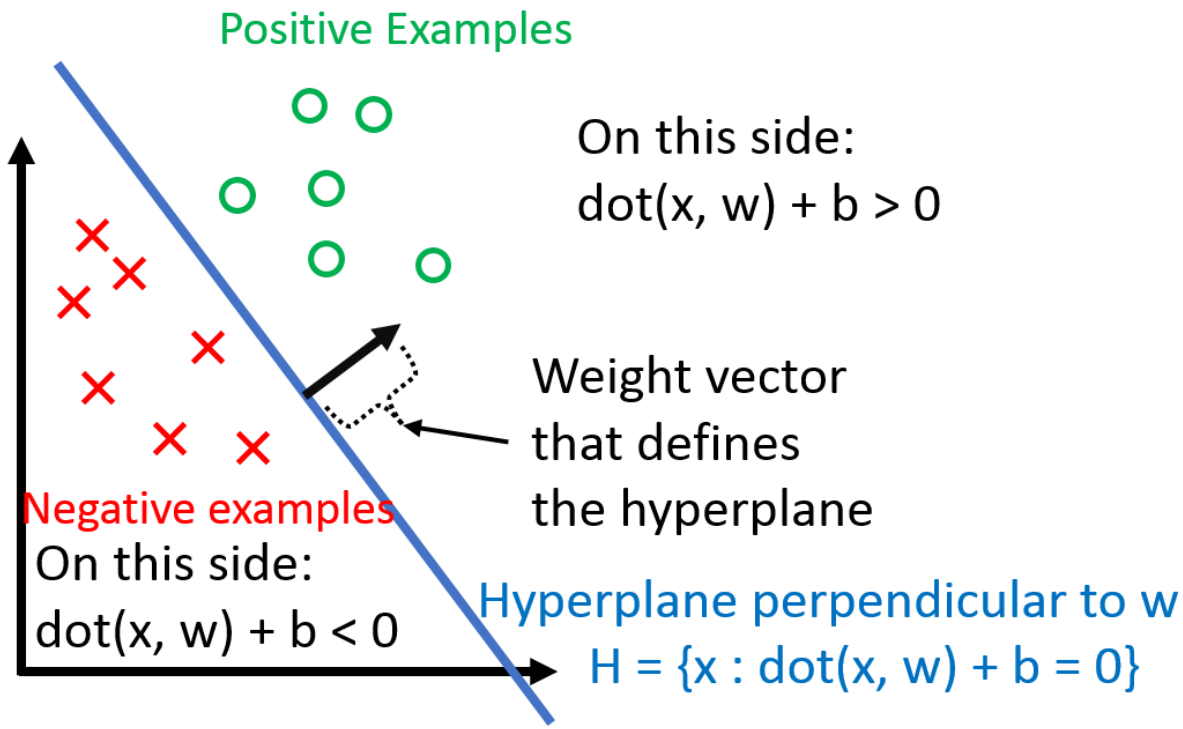


FIG 2.2: Illustration of the hyperplane (in this example, a straight line) as decision boundary for a two-dimensional, two-class pattern-classification problem.

- From the model, we find that the hard limiter input, or induced local field, of the neuron is,

$$y_i = v = \sum_{i=1}^{m} w_i x_i + b$$

$\boldsymbol{b}$ is the bias term (without the bias term, the hyperplane that $\boldsymbol{w}$ defines would always have to go through the origin). Dealing with $\boldsymbol{b}$ can be a pain, so we 'absorb' it into the feature vector $\boldsymbol{w}$ by adding one additional constant dimension.

- Under this convention,

$$\boldsymbol{x_i}\ \boldsymbol{becomes}\ \begin{bmatrix} \boldsymbol{x_i} \\ \mathbf{1} \end{bmatrix} \text{ and}$$

$$w\ becomes \begin{bmatrix} w \\ b \end{bmatrix}$$

- We can verify that

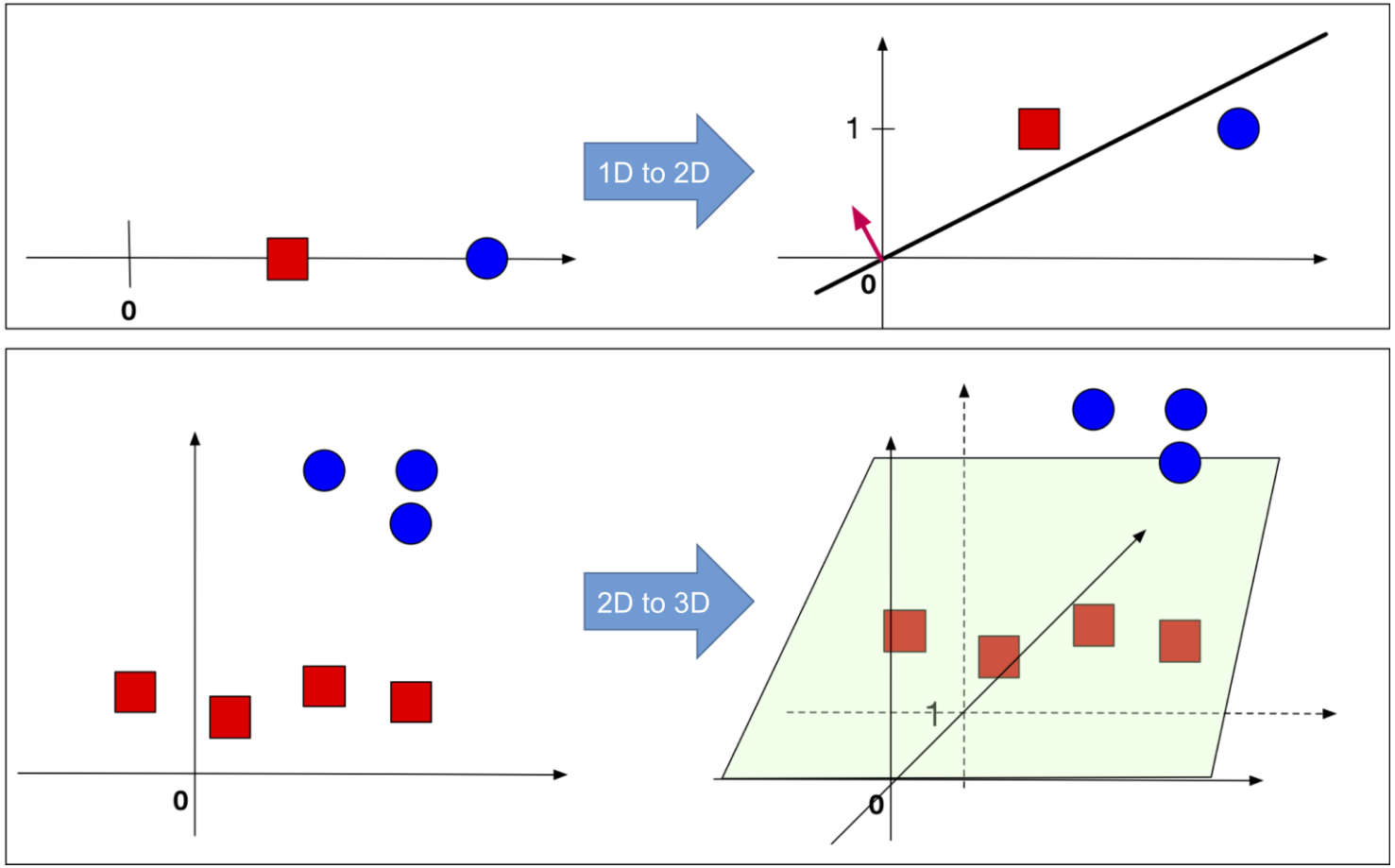
$$\begin{bmatrix} \boldsymbol{x_i} \\ \mathbf{1} \end{bmatrix}^\top \begin{bmatrix} w \\ b \end{bmatrix} = w^\top x_i + b$$

Using this, we can simplify the above formulation of $f(xi)$ to

$$\mathrm{f}(xi) = \varphi(w^\top x)$$

.

<u>Observation:</u> Note that $\boldsymbol{y_i}(\boldsymbol{w}^\top \boldsymbol{x_i}) > \mathbf{0} \Leftrightarrow \boldsymbol{x_i}$ is classified correctly
where 'classified correctly' means that $\boldsymbol{x^i}$ is on the correct side of the hyperplane defined by $\boldsymbol{w}$. Also, note that the left side depends on $y_i \in \{-1, +1\}$ (it wouldn't work if, for example $y_i \in \{0, +1\}$).



(Left:) The original data is 1-dimensional (top row) or 2-dimensional (bottom row). There is no hyper-plane that passes through the origin and separates the red and blue points. (Right:) After a constant dimension was added to all data points such a hyperplane exists.

Perceptron Algorithm

i. $Initalize\ \vec{w} = \vec{0}$

ii. $while\ \boldsymbol{True}\ do$

$m\ =\ 0$ *//count the miss classifications, m*

$\text{for}\ (x_i, yi) \in D\ do$ *//loop over each (data, label) pair in the dataset, D*

$if\ \boldsymbol{yi}(\vec{w}^{\boldsymbol{T}}.\overrightarrow{\boldsymbol{x_i}}) \geq 0\ then$ *//if the pair (xi, yi)is misclassified*

$\vec{\boldsymbol{w}} \leftarrow \vec{\boldsymbol{w}} + y\vec{\boldsymbol{x}}$ *//update the weight vector*

$\boldsymbol{m}\ \leftarrow \boldsymbol{m} + \boldsymbol{1}$

$end\ if$

$end\ for$

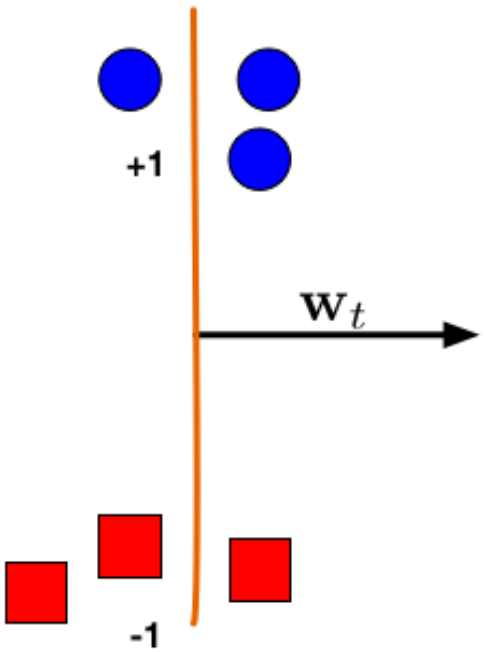
$\text{if}\ m = 0\ then$ *//if the recent w gives 0 misclassifications*

break *//break out the while loop*

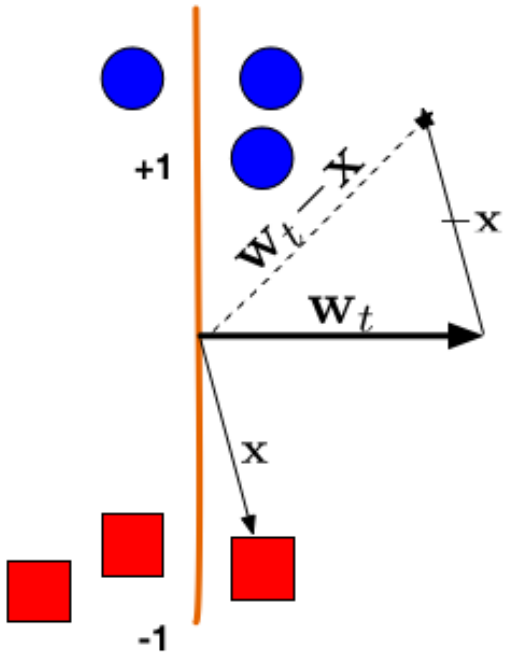
$end\ if$

$end\ while$

Geometric Intuition

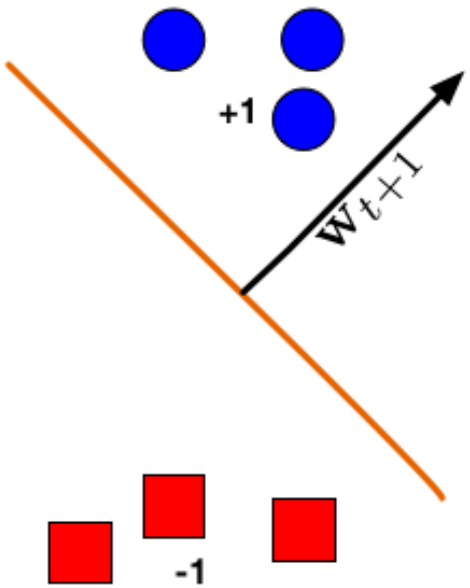


The hyperplane defined by $\mathbf{w_t}$ *misclassifies one red (-1) and one blue (+1) point.*



The red point $\boldsymbol{x}$ *is chosen and used for an update. Because its label is -1 we need to* ***subtract x*** *from* $\mathbf{w_t}$.

Fig: lustration of a Perceptron update.



The updated hyperplane $\boldsymbol{w_t + 1 = w_t - x}$ *separates the two classes and the Perceptron algorithm has converged.*

Perceptron Convergence Theorem

- Proves that perceptron learning algorithm converges in finite steps
- Assumes data is linearly separable
- Guarantees finding a solution (if it exists)
- Let us assume, $\exists \boldsymbol{w}^{*}$ such that $\boldsymbol{y_i}(\boldsymbol{w}^{*T}\boldsymbol{x}) > \boldsymbol{0} \ \forall(\boldsymbol{xi}, \boldsymbol{yi}) \in \boldsymbol{D}$
- The bias $b(n)$ is treated as a synaptic weight driven by a fixed input equal to 1. Thus, Input dimension is (m+1) by 1 given as,

$$\mathbf{x}(n) = [+1, x_1(n), x_2(n), ..., x_m(n)]^T$$

where n denotes the time step in applying the algorithm. Correspondingly, we define the $(m + 1) - by - 1$ *weight vector as,*

$$\mathbf{w}(n) = [b, w_1(n), w_2(n), ..., w_m(n)]^T$$

Accordingly, the linear combiner output is written in the compact form

$$v(n) = \sum_{i=0}^{m} w_i(n)x_i(n)$$
$$= \mathbf{w}^T(n)\mathbf{x}(n)$$

Adjust the weight vector **w** in such a way that the two classes $\mathcal{C}1$ and $\mathcal{C}2$ are linearly separable. i.e.

$\boldsymbol{w^T X} > \mathbf{0}$ for every input vector $\boldsymbol{x}$ belonging to class $\mathcal{C}1$
$\boldsymbol{w^T X} < \mathbf{0}$ for every input vector $\boldsymbol{x}$ belonging to class $\mathcal{C}2$

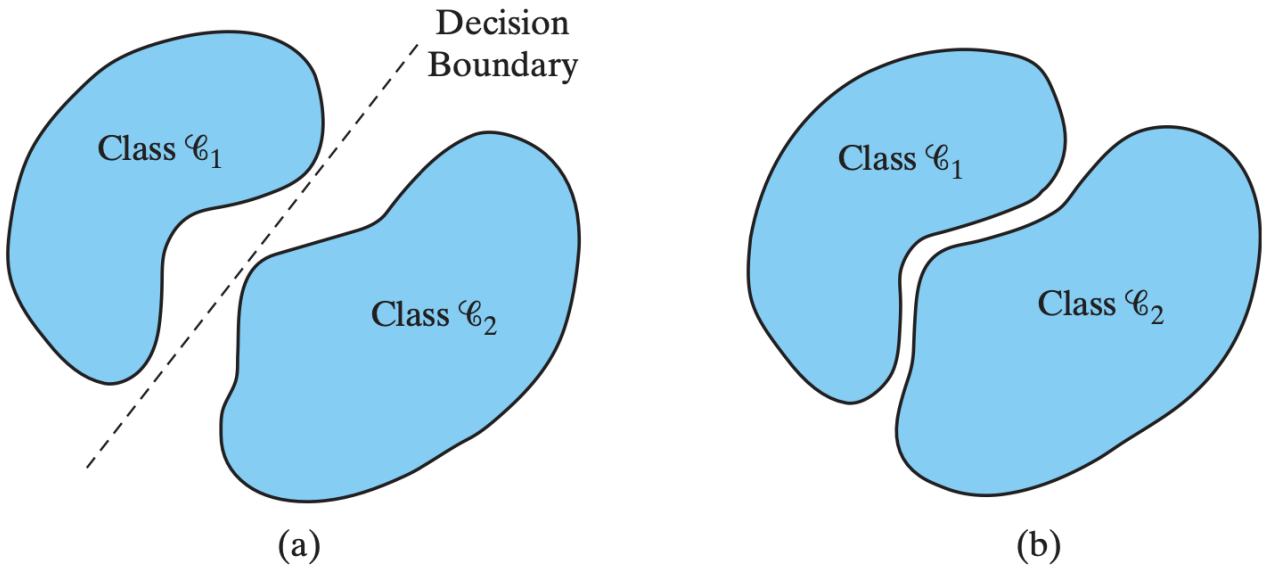
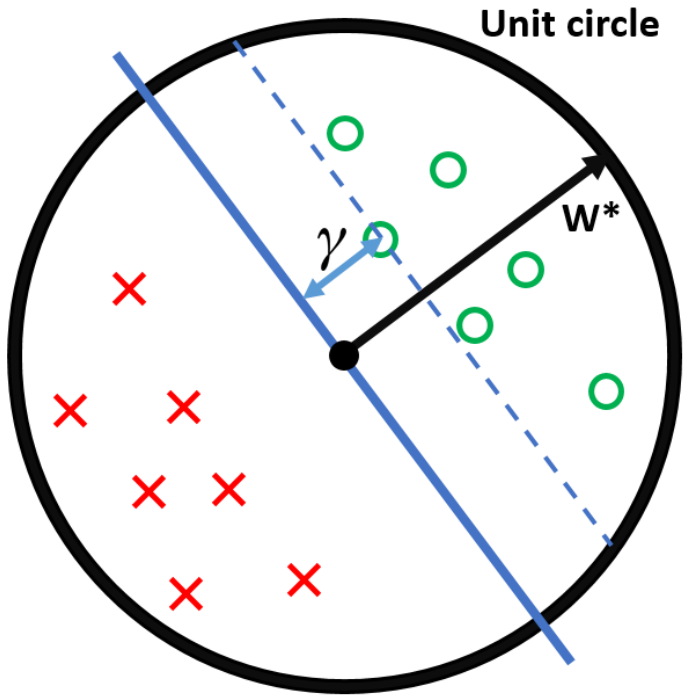


Fig: (a) A pair of linearly separable patterns. (b) A pair of non-linearly separable patterns.

Now, suppose that we rescale each data point and the **w*** such that
$\|w^*\| = 1$ and $\|x_i\| \leq 1 \;\; \forall xi \in D$
Let us define the <u>*margin*</u> $\boldsymbol{\gamma}$ <u>*of the hyperplane*</u> ***w**** *as*

$$\boldsymbol{\gamma = min(xi, yi) \in D|w^{*Tx}|}$$



To summarize our setup:

- All inputs $\boldsymbol{x}_i$ live within the unit sphere.
- There exists a separating hyperplane defined by **w∗**, with $\|w\|^*=1$ (i.e. **w∗** lies exactly on the unit sphere).
- γ is the distance from this hyperplane (blue) to the closest data point.
- ***Theorem:*** *If all of the above holds, then the Perceptron algorithm makes at most* ***1/γ2*** *mistakes.*

Or "For any finite set of linearly separable labeled examples, the Perceptron Learning Algorithm (PLA) will terminate after a finite number of iterations."

Proof:

Keeping what we defined above, consider the effect of an update ($\boldsymbol{w}$ becomes $\boldsymbol{w} + y\boldsymbol{x}$) on the two terms $\boldsymbol{w}^T\boldsymbol{w}^*$ and $\boldsymbol{w}^T\boldsymbol{w}$. We will use two facts:

- $y(\boldsymbol{x}^\top\boldsymbol{w}) \leq 0$: This holds because $\boldsymbol{x}$ is misclassified by $\boldsymbol{w}$ - otherwise we wouldn't make the update.
- $\boldsymbol{y}(\boldsymbol{x}^\top\boldsymbol{w}^*) > \boldsymbol{0}$: This holds because $\boldsymbol{w}^*$ is a separating hyperplane and classifies all points correctly.

1. Consider the effect of an update on $\boldsymbol{w}^\top\boldsymbol{w}^*$:

$$(\boldsymbol{w} + \boldsymbol{y}\boldsymbol{x})^\top\boldsymbol{w}^* = \boldsymbol{w}^\top\boldsymbol{w}^* + \boldsymbol{y}(\boldsymbol{x}^\top\boldsymbol{w}^*) \geq \boldsymbol{w}^\top\boldsymbol{w}^* + \boldsymbol{\gamma}$$

The inequality follows from the fact that, for $\boldsymbol{w}^*$, the distance from the hyperplane defined by $\boldsymbol{w}^*$ to $\boldsymbol{x}$ must be at least $\boldsymbol{\gamma}$ $(\boldsymbol{i.e.}\, \boldsymbol{y}(\boldsymbol{x}^\top\boldsymbol{w}^*) = |\boldsymbol{x}^\top\boldsymbol{w}^*| \geq \boldsymbol{\gamma})$.

<u>This means that for each update,</u> $\boldsymbol{w}^T\boldsymbol{w}^*$ <u>grows by **at least** γ</u>.

2. Consider the effect of an update on $\boldsymbol{w}^T\boldsymbol{w}$:

$$(\boldsymbol{w} + \boldsymbol{y}\boldsymbol{x})^\top(\boldsymbol{w} + \boldsymbol{y}\boldsymbol{x}) = \boldsymbol{w}^\top\boldsymbol{w} + \mathbf{2}\boldsymbol{y}(\boldsymbol{w}^\top\boldsymbol{x}) + \boldsymbol{y}^{\mathbf{2}}(\boldsymbol{x}^\top\boldsymbol{x}) \leq \boldsymbol{w}^\top\boldsymbol{w} + \mathbf{1}$$

The inequality follows from the fact that

- $\mathbf{2}\boldsymbol{y}(\boldsymbol{w}^\top\boldsymbol{x}) < \mathbf{0}$ as we had to make an update, meaning $\boldsymbol{x}$ was misclassified
- $\mathbf{0} \leq \boldsymbol{y}^{\mathbf{2}}(\boldsymbol{x}^\top\boldsymbol{x}) \leq 1$ as $y^2=1$ and all $x^Tx \leq 1$ (because $\|x\| \leq 1$).

<u>This means that for each update, $\boldsymbol{w}^T\boldsymbol{w}$ grows by **at most** 1.</u>

3. Now we know that after M updates the following two inequalities must hold:

   1) $\boldsymbol{w}^\top \boldsymbol{w}^* \geq \boldsymbol{M\gamma}$
   2) $\boldsymbol{w}^\top \boldsymbol{w} \leq \boldsymbol{M}$.

   We can then complete the proof:

$$\boldsymbol{M\gamma} \leq \boldsymbol{w}^\top \boldsymbol{w}^*$$ By (1)

$$= \| \boldsymbol{w} \| \cos(\theta)$$ by definition of inner-product, where θ is the angle between $\boldsymbol{w}$ and $\boldsymbol{w}^*$.

$$\leq \left|\left|\boldsymbol{w}\right|\right|$$ by definition of cos, we must have cos(θ)≤1.

$$= \sqrt{\boldsymbol{w}^\top \boldsymbol{w}}$$ by definition of ‖w‖

$$\leq \sqrt{\boldsymbol{M}}$$ By (2)

$$\Rightarrow \boldsymbol{M\gamma} \leq \sqrt{\boldsymbol{M}}$$
$$\Rightarrow \boldsymbol{M^2\gamma^2} \leq \boldsymbol{M}$$
$$\Rightarrow \boldsymbol{M} \leq \boldsymbol{1/\gamma^2}$$

And hence, the number of updates M is bounded from above by a constant.

Limitations of Perceptron

- Fails to solve non-linearly separable problems (e.g., XOR)
- Can only classify linearly separable datasets
- Motivated development of multilayer perceptron

RELATION BETWEEN THE PERCEPTRON AND BAYES CLASSIFIER FOR A GAUSSIAN ENVIRONMENT

- The perceptron bears a certain relationship to a classical pattern classifier known as the Bayes classifier.
- When the environment is Gaussian, the Bayes classifier reduces to a linear classifier.

$\mathscr{X} = \mathcal{H}$

$\mathscr{C}_j = Ci = \mathcal{C}_j$

Bayes Classifier

- *Bayes classifier*, or *Bayes hypothesis testing procedure*
  - minimize the *average risk*, denoted by $\mathfrak{R}$.

- For a two-class problem, represented by classes $\mathcal{C}1$ and $\mathcal{C}2$, the average risk is defined by Van Trees (1968) as,

$$\mathcal{R} = c_{11}p_1 \int_{\mathcal{H}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)d\mathbf{x} + c_{22}p_2 \int_{\mathcal{H}_2} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2)d\mathbf{x} \quad \text{Correctly classified}$$

$$+ c_{21}p_1 \int_{\mathcal{H}_2} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)d\mathbf{x} + c_{12}p_2 \int_{\mathcal{H}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2)d\mathbf{x} \quad \text{missclassified} \qquad (1.23)$$

Where,

p_i - *prior probability* that the observation vector corresponds to an object in class $\mathcal{C}1$, with i=1,2, and

$$p_1 + p_2 = 1$$

$\mathcal{C}_{ij}$ cost of deciding in favor of class $\mathcal{C}_i$ represented by subspace H_i when class$\mathcal{C}_i$ is true (i.e., observation vector **x** corresponds to an object in class $\mathcal{C}_1$), with $i, j = 1, 2$

$p_x(\mathbf{x}|\ \mathcal{C}_i)$ conditional probability density function (pdf) of the random vector **X**, given that the observation vector **x** corresponds to an object in class $\mathcal{C}_1$, with $i = 1, 2$.

- To minimize the risk,
  - each observation vector $\mathbf{x}$ must be assigned in the overall observation space $\mathcal{H}$ to either $\mathcal{H}1$ *or* $\mathcal{H}2$. Thus,

$$\mathcal{H} = \mathcal{H}1 + \mathcal{H}2 \quad \ldots\ldots (1.24)$$

we may rewrite Eq. (1.23) in the equivalent form

$$\begin{aligned}\mathcal{R} = {} & c_{11}p_1 \int_{\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)d\mathbf{x} + c_{22}p_2 \int_{\mathcal{X}-\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2)d\mathbf{x} \\ & + c_{21}p_1 \int_{\mathcal{X}-\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)\mathbf{x} + c_{12}p_2 \int_{\mathcal{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2)d\mathbf{x}\end{aligned} \tag{1.25}$$

where $c_{11} < c_{21}$ and $c_{22} < c_{12}$. We now observe the fact that

$$\int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)d\mathbf{x} = \int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2)d\mathbf{x} = 1 \tag{1.26}$$

Hence, Eq. (1.25) reduces to

$$\mathcal{R} = c_{21}p_1 + c_{22}p_2 + \int_{\mathcal{X}_1} [p_2(c_{12} - c_{22})\, p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_2) - p_1(c_{21} - c_{11})\, p_{\mathbf{X}}(\mathbf{x}|\mathcal{C}_1)]d\mathbf{x} \tag{1.27}$$

The first two terms on the right-hand side of Eq. (1.27) represent a fixed cost. Since the requirement is to minimize the average risk $\mathfrak{R}$, we may therefore deduce the following strategy from Eq.(1.27) for optimum classification:

1. Assign any value of **x** to class $\boldsymbol{\mathcal{C}1}$ if the expression inside the square brackets is **-ve**, because this will reduce the overall risk $\mathfrak{R}$.
2. Assign any value of **x** to class $\boldsymbol{\mathcal{C}2}$ if the expression is **+ve**, because this will increase the overall risk if included in class $\boldsymbol{\mathcal{C}1}$.
3. If the expression is **0**, it doesn't affect the risk, so those values of **x** can be assigned to either class. Here, we choose to assign them to **class** $\boldsymbol{\mathcal{C}2}$.

On this basis, we may now formulate the Bayes classifier as follows:

If the condition

$$p_1(c_{21} - c_{11})\, p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1) > p_2(c_{12} - c_{22})\, p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)$$

holds, assign the observation vector $\mathbf{x}$ *to subspace* $\mathscr{X}_1$ *(i.e., class* $\mathscr{C}_1$*). Otherwise assign* $\mathbf{x}$ *to* $\mathscr{X}_2$ *(i.e., class* $\mathscr{C}_2$*).*

To simplify matters, define

$$\Lambda(\mathbf{x}) = \frac{p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)}{p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)} \tag{1.28}$$

and

$$\xi = \frac{p_2(c_{12} - c_{22})}{p_1(c_{21} - c_{11})} \tag{1.29}$$

The quantity $\Lambda(\mathbf{x})$, the ratio of two conditional probability density functions, is called the ***likelihood ratio.***

The quantity ξ is called the *threshold* of the test.

Note that both $\xi(\boldsymbol{x})$ *and* $\Lambda(\boldsymbol{x})$ *are always positive.*

Bayes classifier by stating the following

If, for an observation vector **x**, *the likelihood ratio* $\Lambda(x)$ *is greater than the threshold* ξ, *assign* **x** *to class* $\mathcal{C}1$. *Otherwise, assign it to class* $\mathcal{C}2$.

The data processing involved in designing the Bayes classifier is confined entirely to the computation of the likelihood ratio $\Lambda(x)$.

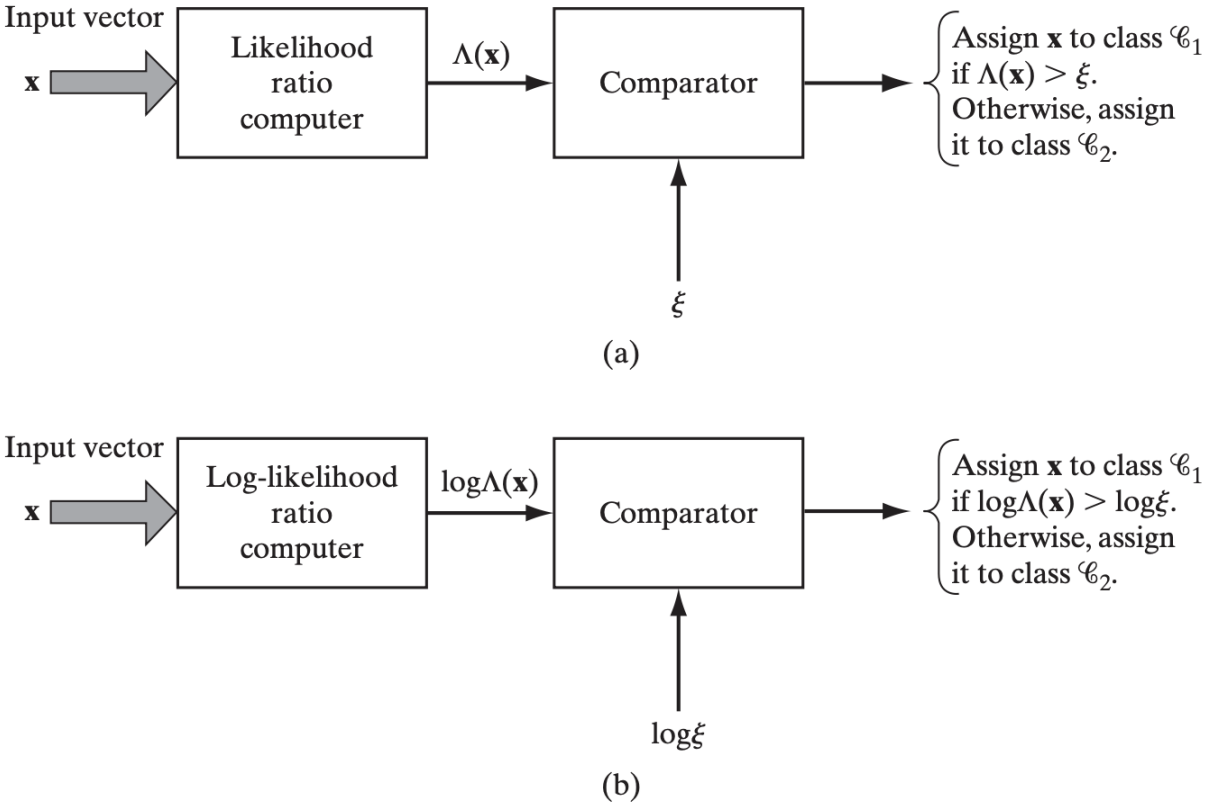


Fig: Two equivalent implementations of the Bayes classifier: (a) Likelihood ratio test, (b) Log-likelihood ratio test.

Bayes Classifier for a Gaussian Distribution

- Consider a case of a two-class problem, for which the underlying distribution is Gaussian.
- The random vector **X** has a mean value that depends on whether it belongs to class c1 or class c2, but the covariance matrix of **X** is the same for both classes.

$$
\begin{aligned}
\text{Class } \mathscr{C}_1: \quad & \mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}_1 \\
& \mathbb{E}[(\mathbf{X} - \boldsymbol{\mu}_1)(\mathbf{X} - \boldsymbol{\mu}_1)^T] = \mathbf{C} \\
\text{Class } \mathscr{C}_2: \quad & \mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}_2 \\
& \mathbb{E}[(\mathbf{X} - \boldsymbol{\mu}_2)(\mathbf{X} - \boldsymbol{\mu}_2)^T] = \mathbf{C}
\end{aligned}
$$

The covariance matrix **C** is nondiagonal, which means that the samples drawn from classes c1 and c2 are *correlated.* It is assumed that **C** is nonsingular, so that its inverse matrix $\boldsymbol{C}^{-1}$ exists.

- The conditional probability density function of **X** as the multivariate Gaussian distribution

$$p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_i) = \frac{1}{(2\pi)^{m/2}(\det(\mathbf{C}))^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\right), \quad i = 1, 2 \quad (1.30)$$

where m is the dimensionality of the observation vector $\mathbf{x}$.

The two classes C1 and C2 are equiprobable:

$$p_1 = p_2 = \frac{1}{2} \quad (1.31)$$

Misclassifications carry the same cost, and no cost is incurred on correct classifications:

$$c_{21} = c_{12} \quad \text{and} \quad c_{11} = c_{22} = 0 \quad (1.32)$$

- By substituting Eq. (1.30) into (1.28) and taking the natural logarithm, we get (after simplifications)

$$\begin{aligned}\log\Lambda(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) \\ &= (\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^T\mathbf{C}^{-1}\mathbf{x} + \frac{1}{2}(\boldsymbol{\mu}_2^T\mathbf{C}^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T\mathbf{C}^{-1}\boldsymbol{\mu}_1)\end{aligned} \tag{1.33}$$

- By substituting Eqs. (1.31) and (1.32) into Eq. (1.29) and taking the natural logarithm, we get

$$\log\xi = 0 \tag{1.34}$$

Equations (1.33) and (1.34) state that the Bayes classifier for the problem at hand is a *linear classifier*, as described by the relation

$$y = \mathbf{w}^T\mathbf{x} + b \tag{1.35}$$

Where,

$$\mathbf{y} = \log\Lambda(\mathbf{x}) \tag{1.36}$$

$$\mathbf{w} = \mathbf{C}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{1.37}$$

$$b = \frac{1}{2}(\boldsymbol{\mu}_2^T\mathbf{C}^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T\mathbf{C}^{-1}\boldsymbol{\mu}_1) \tag{1.38}$$

On the basis of Eq. (1.35), we may now describe the log-likelihood ratio test for our two-class problem as follows:

If the output y of the linear combiner (including the bias b) is positive, assign the observation vector **x** *to class* $C1$. *Otherwise, assign it to class* $C2$.

$\boldsymbol{\mu 1}$and $\boldsymbol{\mu 2}$ the covariance matrix **C**.

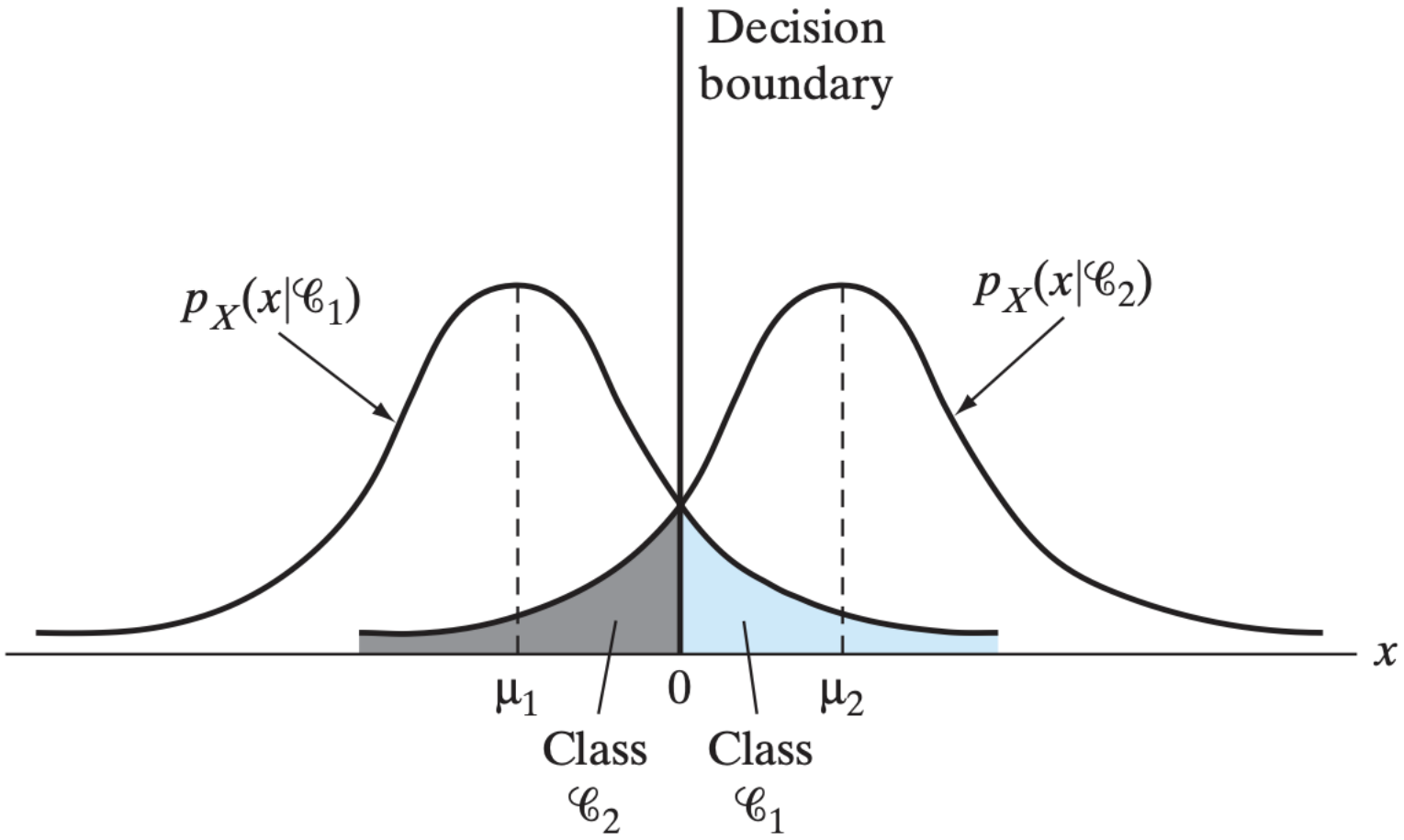
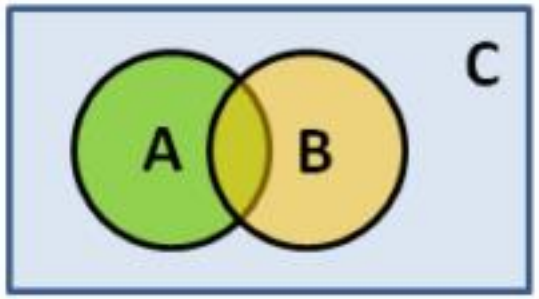
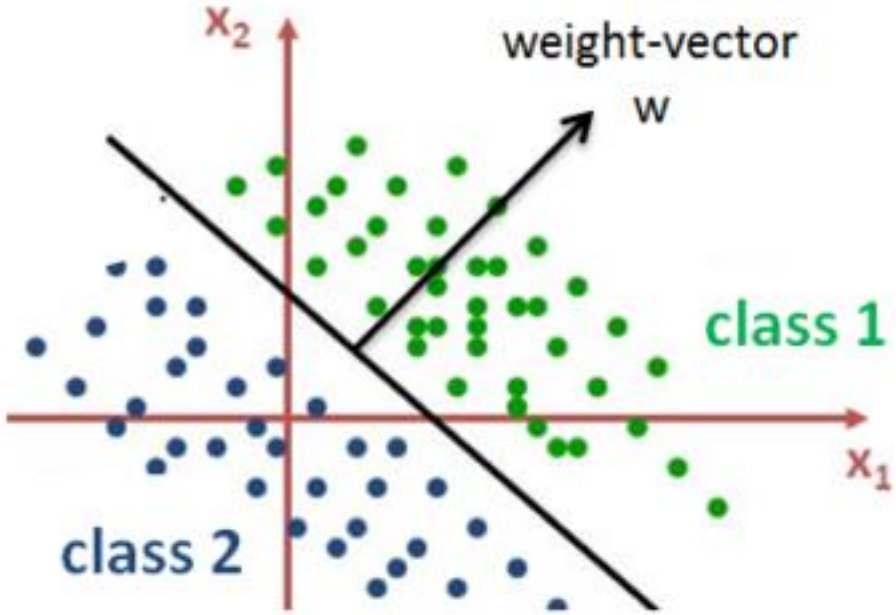


Fig: Two overlapping, one-dimensional Gaussian distributions.

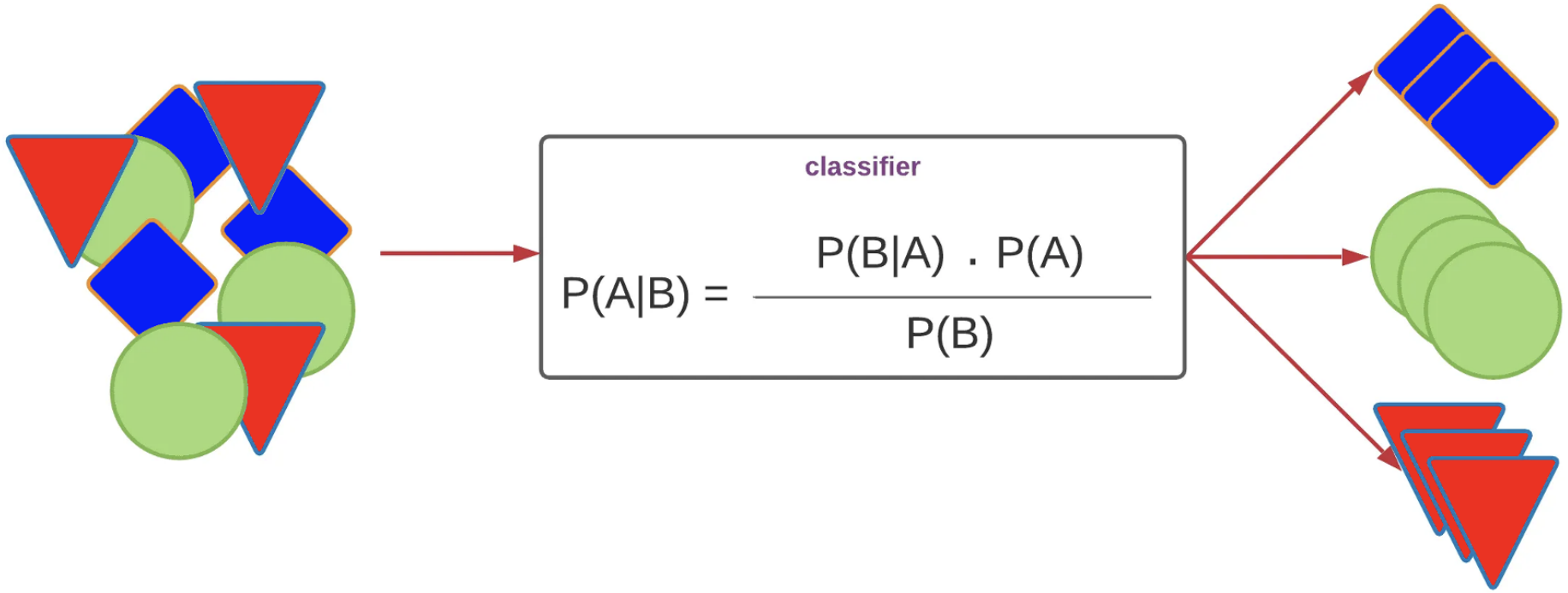
Perceptron vs. Bayes Classifier

- **Bayes Classifier**
  - Minimizes classification error probability, regardless of overlap between class distributions.
  - Decision boundary lies where class distributions intersect (e.g., Gaussians).
  - **Parametric**: Assumes specific distribution form (e.g., Gaussian), limiting applicability.
  - Fixed design; can be made adaptive but requires more storage and computation.

- **Perceptron Convergence Algorithm**:
  - **Nonparametric**: Makes no assumptions about input distribution.
  - Focuses on classification errors, especially in overlapping regions.
  - Performs well with nonlinear or non-Gaussian input distributions.
  - **Adaptive and simple**: Requires only storage for weights and bias.

Statistical Approach (Naive Bayes)	Geometrical Approach (Perceptron, SVM)
$$P(A\|B) = \frac{P(B\|A)P(A)}{P(B)}$$ 	

Naive Bayes Classifier



THE BATCH PERCEPTRON ALGORITHM

1. ***Introduces the generalized form of a perceptron cost function.***
2. ***Uses the cost function to formulate a batch version of the perceptron convergence algorithm.***

we define the *perceptron cost function* as

$$J(\mathbf{w}) = \sum_{\mathbf{x}(n)\in\mathcal{X}} (-\mathbf{w}^T\mathbf{x}(n)d(n)) \tag{1.39}$$

where $\mathcal{H}$ is the set of samples $\mathbf{x}$ *misclassified* by a perceptron using $\mathbf{w}$ as its weight vector.

- If all the samples are classified correctly, then the set $\mathcal{H}$ is empty, in which case the cost function $J(\boldsymbol{w})$ is zero.
- In any event, the nice feature of the cost function $J(\boldsymbol{w})$ is that it is *differentiable* with respect to the weight vector $\mathbf{w}$.

Differentiating $J(\boldsymbol{w})$ with respect to $\mathbf{w}$ yields the *gradient vector,*

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x}(n)\in\mathscr{X}} (-\mathbf{x}(n)d(n)) \tag{1.40}$$

where the *gradient operator*

$$\boldsymbol{\nabla} = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T \tag{1.41}$$

In the *method of steepest descent,* the adjustment to the weight vector $\mathbf{w}$ at each time-step of the algorithm is applied in a direction *opposite* to the gradient vector $\nabla J(\boldsymbol{w})$. Accordingly, the algorithm takes the form

$$\begin{aligned}\mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\nabla J(\mathbf{w}) \\ &= \mathbf{w}(n) + \eta(n)\sum_{\mathbf{x}(n)\in\mathscr{X}} \mathbf{x}(n)d(n)\end{aligned} \tag{1.42}$$

which includes the single-sample correction version of the perceptron convergence algorithm as a special case.

Applications and Experiment

- Simple pattern classification
- Handwritten digit recognition (basic)
- Basis for further neural network models

Summary

- Perceptron is a foundational model for neural networks
- Perceptron convergence theorem ensures learning under linear separability
- Limitations led to the evolution of deep networks

Thank You!!!