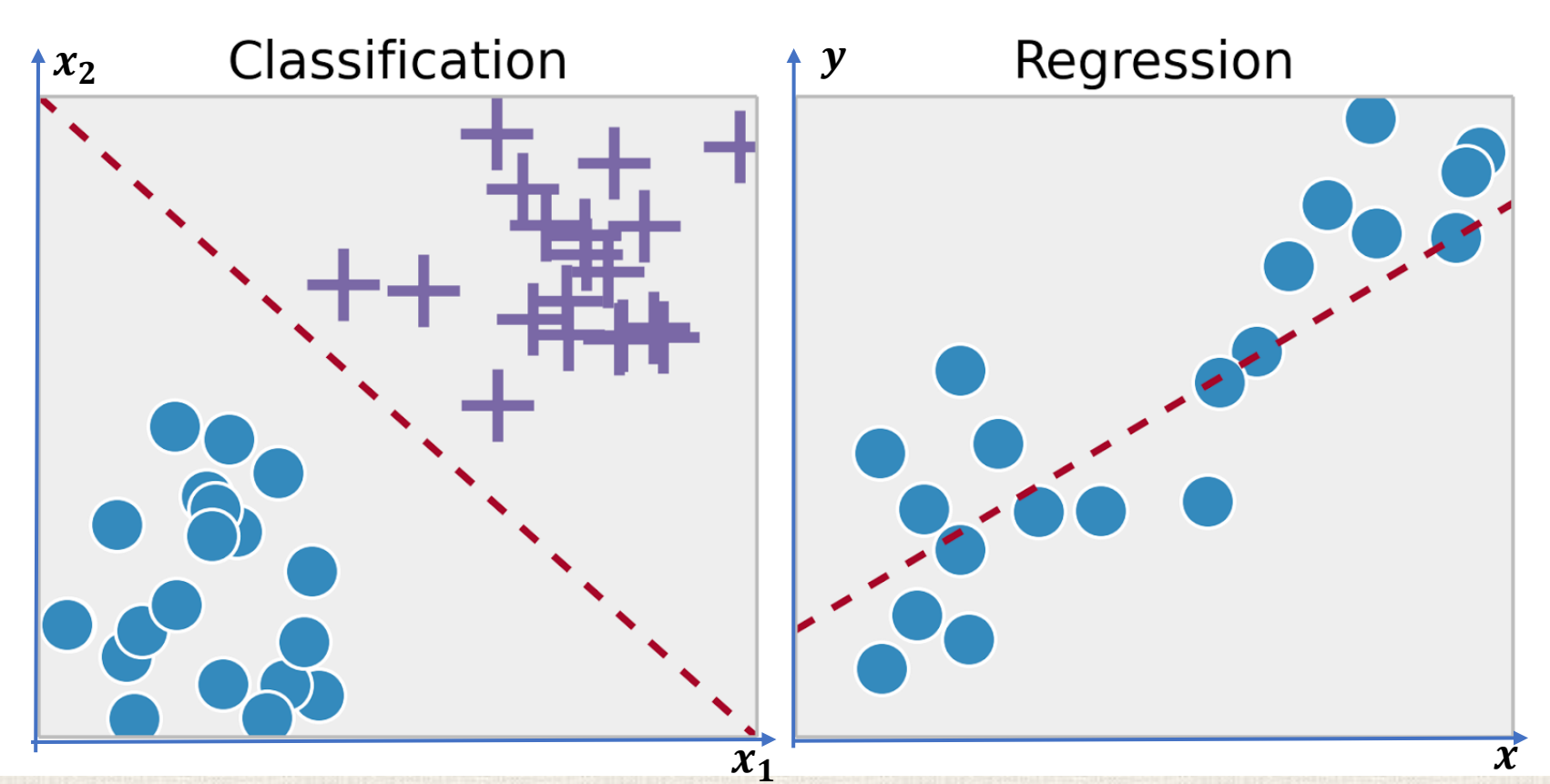
Regression vs Classification



Predict a nominal class label

Predict a continuous quantity

The Classification Problems

- ***The classification problems:*** given a training data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ of n points, where each data points $\boldsymbol{x}_i \in \mathcal{R}^d$ is paired with a known discrete class label y_i which represent one of K discrete classes $\mathcal{C}_k$, where $k = 1,2, \dots, K$. The goal is to train a classifier which, when fed any arbitrary d-dimensional data point, classifies that data point as one of the K discrete classes.

- **Target values in classification problems:**

  - In a binary classification problem, $y \in \{0,1\}\ or\ \{-1,1\}$, such that $y = 1$ represents class $\mathcal{C}_1$ and $y = 0,\ or\ (y = -1)$ represents class $\mathcal{C}_2$

  - In multi-class classification, it is convenient to use a ***1-of-K (one-hot) coding scheme*** where $\boldsymbol{y}$ becomes a vector of length K such that if the class is $\mathcal{C}_j$ then all the element of $\boldsymbol{y}$ are zero except the element y_j, which takes the value of 1. For instance, let $K = 4$ classes, the pattern from class 2 would be given the target vector

$$\boldsymbol{y} = [0 \quad 1 \quad 0 \quad 0]^T$$

  - The value of y_j can be interpreted as the probability that the class is $\mathcal{C}_j$.

❑ **Target values in classification problems:**

- We prefer one-hot encoding over representing $y = 1,2,3,4$ (***ordinal values***) for $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\ and\ \mathcal{C}_4$ (***nominal values***) is that the later implies order as $4 > 3 > 2 > 1$.
  - Let the target value of an input $\boldsymbol{x}$ be $y = 4$. Classifier#1 classifies the input as $y_1 = 1$, classifier#2 classifies the same input as $y_2 = 3$.
  - Since $y - y_1 = 3 > y - y_2 = 1$, this implies that classifier#2 is better than classifier#1. This is not true. Both classifiers are equally bad in the classification problem.
- ***Mean squared error*** is no longer a popular loss function in classification problem.

- ***Binary classification (dichotomizer)*** ($K = 2$) and ***multi-class classification*** ($K > 2$)
  - Many applications are binary classification problems
  - Some classification methods can only handle binary classification problems (Perceptron, SVM, etc.)
  - Some classification methods can naturally handle multi-class classification problems (nearest neighbor, ANN, decision trees, etc.)
  - In many cases, a multi-class classification problem can be decomposed as multiple binary classification problems.

- ***Binary classification for multi-class classification***
  - Multi-class classification problem can be turned into multiple binary classification problems.
  - One-vs-Rest (OvR)

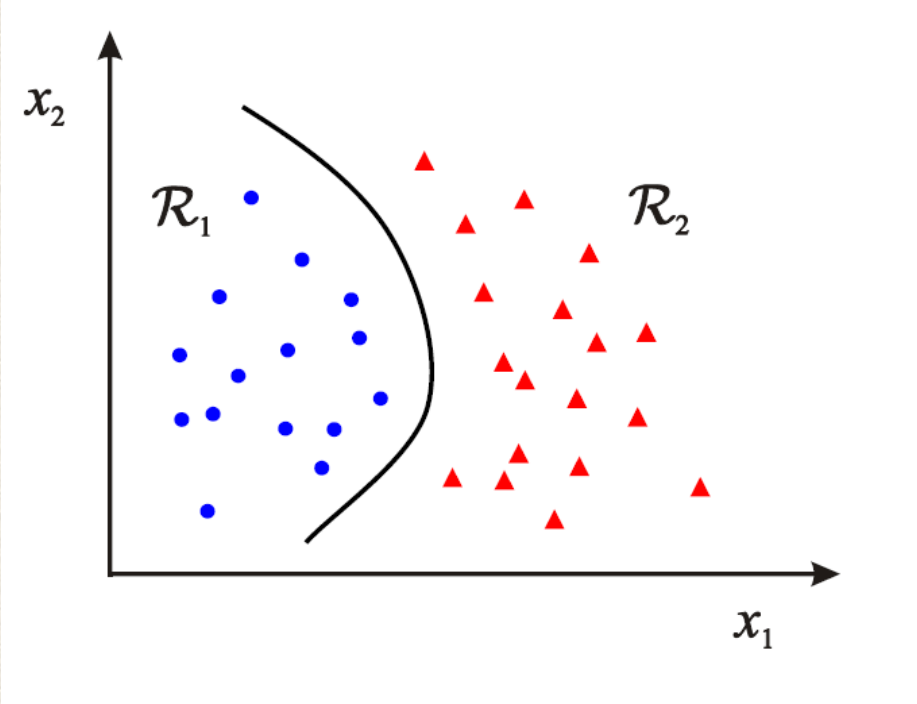
    For example, a multi-class classification problem with three classes: “red”, ”blue” and “green”. This could be divided into three binary classification problems as follows:
    Binary classifier #1: red vs (blue and green)
    Binary classifier #2: blue vs (red and green)
    Binary classifier #3: green vs (red and blue)

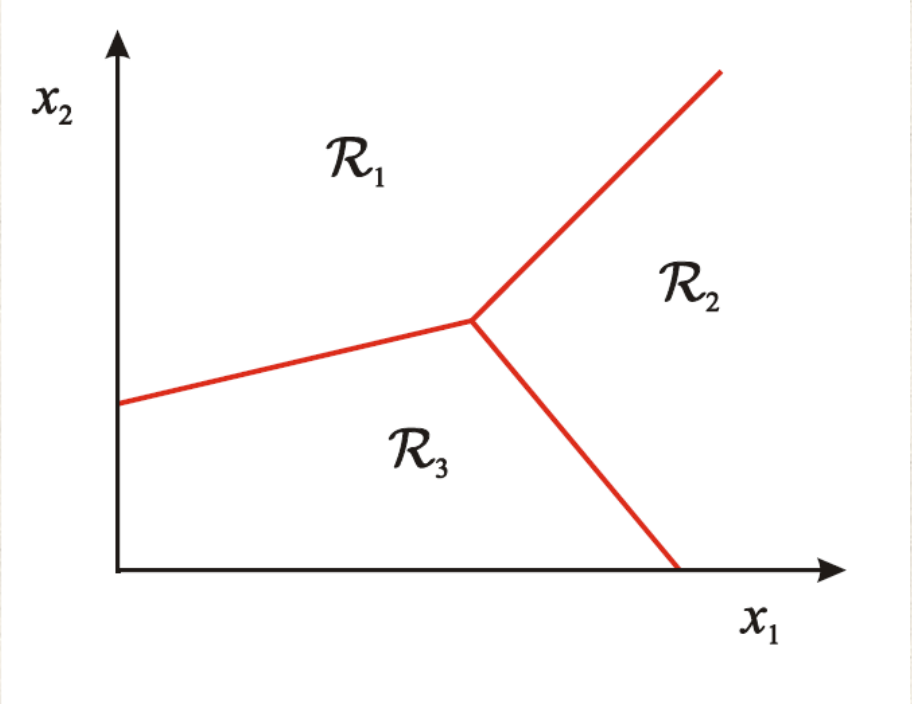
  - One-vs-One (OvO)

    Using the same example, the problem is split into multiple binary classification problems as follows:
    Binary classifier #1: red vs blue
    Binary classifier #2: red vs green
    Binary classifier#3: blue vs green

Decision regions and Decision boundaries

- In most common scenario, the classes are taken to be disjoint so that ***each input is assigned to one and only one class (no class label ambiguity!)***
- The input space is thereby divided into ***decision regions*** R_k such that a point falling in R_k is assigned to class $\mathcal{C}_k$. The boundaries between these decision regions are called ***decision boundaries***.





Discriminant Functions

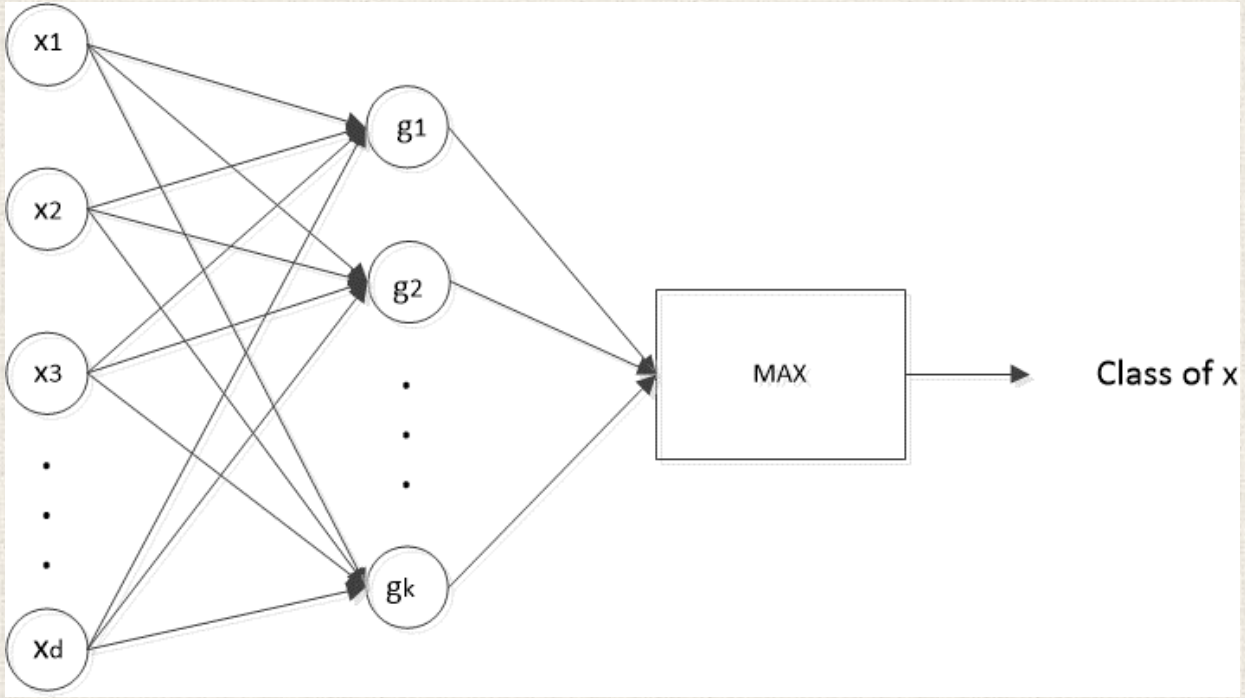
- One of the most useful way to represent classifiers is in terms of a set of ***discriminant functions*** $g_i(\boldsymbol{x}), i = 1, \ldots, K$.

- The classifier is said to assign a feature vector $\boldsymbol{x}$ to class $\mathcal{C}_i$ if

$$g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \qquad for\ all\ j \neq i$$

Discriminant Functions

- The classifier is viewed as a network that computes K discriminant functions and selects the category corresponding to the largest discriminant.



- One popular example:

$$g_i(\boldsymbol{x}) = P(\mathcal{C}_i|\boldsymbol{x})$$

The maximum discriminant function becomes the maximum posterior probability

- ***Discriminant function is not unique!***

$g_i(\boldsymbol{x})$ can be replaced by $f(g_i(\boldsymbol{x}))$ if $f(\cdot)$ is ***monotonically increasing function!***

$$g_i(\boldsymbol{x}) = P(\mathcal{C}_i|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i)}{\sum_{i=1}^{K} P(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i)}$$

$$g_i(\boldsymbol{x}) = P(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i)$$

$$g_i(\boldsymbol{x}) = \ln P(\boldsymbol{x}|\mathcal{C}_i) + \ln P(\mathcal{C}_i)$$

- ***Discriminant function for two-class case:***

  Instead of using two discriminant functions $g_1\ and\ g_2$ and assigning $\boldsymbol{x}$ to $\mathcal{C}_1$ if $g_1(\boldsymbol{x}) > g_2(\boldsymbol{x})$, it is more common to define a single discriminant function

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$

  And to use the following decision rule:

  Assign $\boldsymbol{x}$ to $\mathcal{C}_1$ if $g(\boldsymbol{x}) > 0$, otherwise assign $\boldsymbol{x}$ to $\mathcal{C}_2$

  Examples:

$$g(\boldsymbol{x}) = P(\mathcal{C}_1|\boldsymbol{x}) - P(\mathcal{C}_2|\boldsymbol{x})$$

$$g(\boldsymbol{x}) = ln\left(\frac{P(\boldsymbol{x}|\mathcal{C}_1)}{P(\boldsymbol{x}|\mathcal{C}_2)}\right) + ln\left(\frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)}\right)$$

Performance Metrics for Classifiers

- ***Confusion Matrix***: a natural way to represent classification results.

  A binary classifier case:

classifier		Actual class	
		positive	negative
Predicted class	positive	10	2
	negative	3	5

- 13 positive instances and 7 negative instances
- 10 positive instances are classified as positive
- 3 positive instances are classified as negative
- 5 negative instances are classified as negative
- 2 negative instances are classified as positive
- Diagonal numbers represent number of instance classified correctly
- Off-diagonal numbers represent classification errors

- A multi-class classifier case: (iris data set)

classifier		Actual class		
		setosa	versicolor	virginica
Predicted class	setosa	13	0	0
	versicolor	0	10	6
	virginica	0	0	9

A binary classifier case:

Classifier		Actual class	
		positive	negative
Predicted class	Positive	TP (hit)	FP (false alarm)
	negative	FN (miss)	TN (correct rejection)

- **Condition positive (P):** the number of real positive instances in the data.
- **Condition negative (N):** the number of real negative instances in the data.
- **True positive (TP) (hit):** the number of real positive instances that are classified as positive.
- **True negative (TN) (correct rejection):** the number of negative instances that are classified as negative.
- **False positive (FP) (false alarm) (type I error):** number of real negative instances that are classified as positive.
- **False negative (FN) (miss) (type II error):** number of positive instances that are classified as negative.

Accuracy and Error Rate:

Classifier		Actual class	
		positive	negative
Predicted class	Positive	TP	FP
	negative	FN	TN

- ***Accuracy*** $= \frac{\#\ of\ instances\ classified\ correctly}{\#\ of\ instances\ tested}$

$$= \frac{TP + TN}{TP + FP + FN + TN}$$

- ***Error rate*** $= 1 - Accuracy$

$$= \frac{FP + FN}{TP + FP + FN + TN}$$

Example: Find the accuracy of the following two classifiers

Classifier#1		Actual class	
		positive	negative
Predicted class	Positive	90	10
	negative	10	90

Accuracy = $\frac{90+90}{200} = 90\%$

Classifier#2		Actual class	
		positive	negative
Predicted class	Positive	10	10
	negative	10	170

Accuracy = $\frac{10+170}{200} = 90\%$

Problems with Accuracy and Error rate

- ***The problem with class imbalance:***

  Accuracy simply treat all instances the same. Accuracy maybe fine if you deal with balanced (or approximated balanced) data sets. It can be misleading when deal with imbalanced data sets.

- In real applications domain, imbalanced data sets are the norm!
  For example, cancer diagnose, intrusion detection, defects detection, etc.
  The positive examples to negative examples ratio can be 1:10 or less!

- Need more metrics for classifier performance evaluation!!!

Precision

Classifier		Actual class	
		positive	negative
Predicted class	Positive	TP	FP
	negative	FN	TN

- ***Precision*** $= \frac{TP}{TP+FP}$
- Precision measures what portion of instances that are classified as positive are actually positive.
- Precision gives us information about its performance with respect to False Positive (false alarm).
- If the focus of the application is on ***minimizing False Positive (false alarm),*** we should make precision close to 100%.
- For example, spam email detector.

Classifier#1		Actual class	
		positive	negative
Predicted class	Positive	90	10
	negative	10	90

$$\text{Accuracy} = \frac{90+90}{200} = 90\%$$

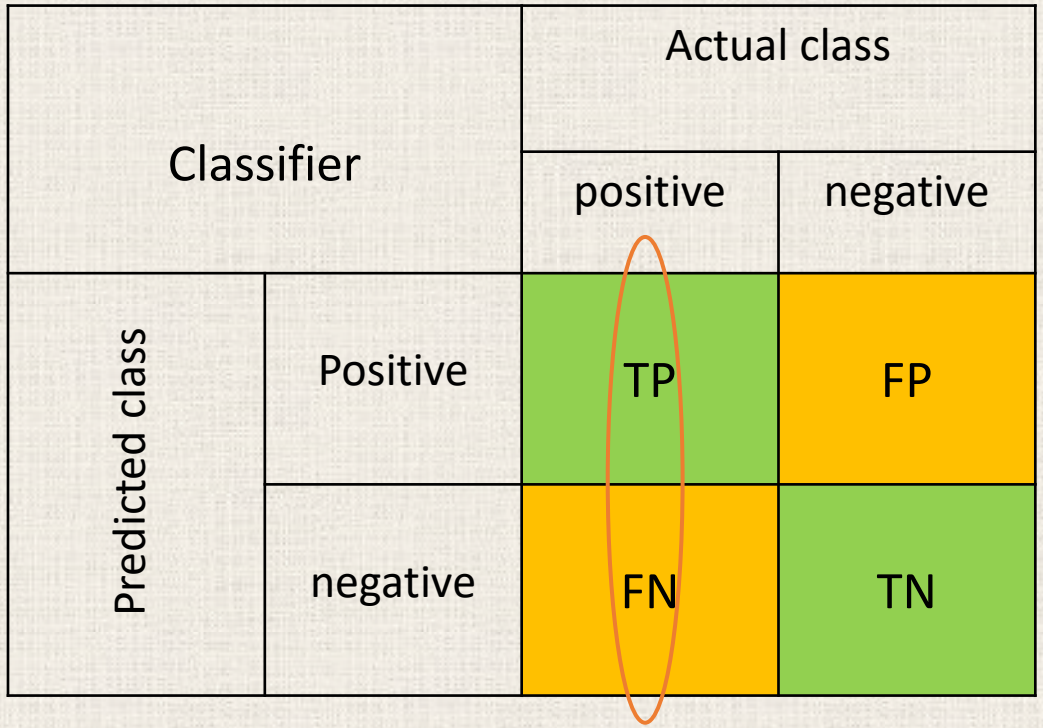
$$\text{Precision} = \frac{90}{90+10} = 90\%$$

Classifier#2		Actual class	
		positive	negative
Predicted class	Positive	10	10
	negative	10	170

$$\text{Accuracy} = \frac{10+170}{200} = 90\%$$

$$\text{Precision} = \frac{10}{10+10} = 50\%$$

Recall or Sensitivity



- ***Recall (sensitivity)*** $= \frac{TP}{TP+FN}$
- **Recall** measures what portion of positive instances that are classified as positive
- **Recall** gives us information about a classifier performance with respect to ***False Negative (miss)***
- ***If the application focus more on minimizing False Negative (miss), we should make recall close to 100% and without precision being too bad.***
- *For example, cancer diagnose, information retrieval*

Performance comparison of two classifiers

Classifier#2		Actual class	
		positive	negative
Predicted class	Positive	10	10
	negative	10	170

$$\text{Recall} = \frac{10}{10+10} = 50\%$$

$$\text{precision} = \frac{10}{10+10} = 50\%$$

$$\text{Accuracy} = \frac{10+170}{200} = 90\%$$

Classifier#2		Actual class	
		positive	negative
Predicted class	Positive	20	20
	negative	0	160

$$\text{Recall} = \frac{20}{15+5} = 100\%$$

$$\text{precision} = \frac{20}{20+20} = 50\%$$

$$\text{Accuracy} = \frac{20+160}{200} = 90\%$$

F-score: a combination of precision and recall

- F_1 score (balanced F score) is the ***harmonic mean*** of precision and recall

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = \frac{2 * precision * recall}{precision + recall}$$

- F_β score: $\beta > 0$ is chosen such that recall is considered β times as important as precision

$$F_\beta = (1 + \beta^2) \frac{precision * recall}{(\beta^2) * precision + recall}$$

Other metrics

- ***True positive rate (TPR)(hit rate)*** = $\frac{TP}{TP+FN} = \frac{TP}{total\ positives}$ = ***recall*** = ***sensitivity***

  = 1- ***false negative rate (FNR)(miss rate)***

- ***False positive rate (FPR)(false alarm rate)*** = $\frac{FP}{FP+TN} = \frac{FP}{total\ negatives}$

  = 1- ***true negative rate (TNR)***

- ***Specificity*** = ***true negative rate (TNR)*** = $\frac{TN}{FP+TN}$

Generative vs. Discriminative Classifier

- There are two main types of classification models: ***generative models and discriminative models***
- Posterior probability plays critical roles in classification problems
- ***Generative models*** have strong roots in probabilistic modeling. As in Bayes Theorem, generative models **explicitly** use prior probability and likelihood to calculate the posterior probabilities

$$\hat{y} = \underset{i}{argmax}\, P(\mathcal{C}_i|\boldsymbol{x}) = \underset{i}{argmax}\, P(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i)$$

e.g., Naïve Bayes, Hidden Markov Models (HMMs), Bayesian Networks, etc.

Generative vs. Discriminative Classifier

- On the other hand, in ***discriminative models***, we bypass learning a generative model altogether and directly learn a decision boundary.

- **Discriminative models** are parameterized by weights that either form a posterior distribution without considering the prior or conditional distribution or directly form a hard decision boundary without considering any probabilities in the first place.

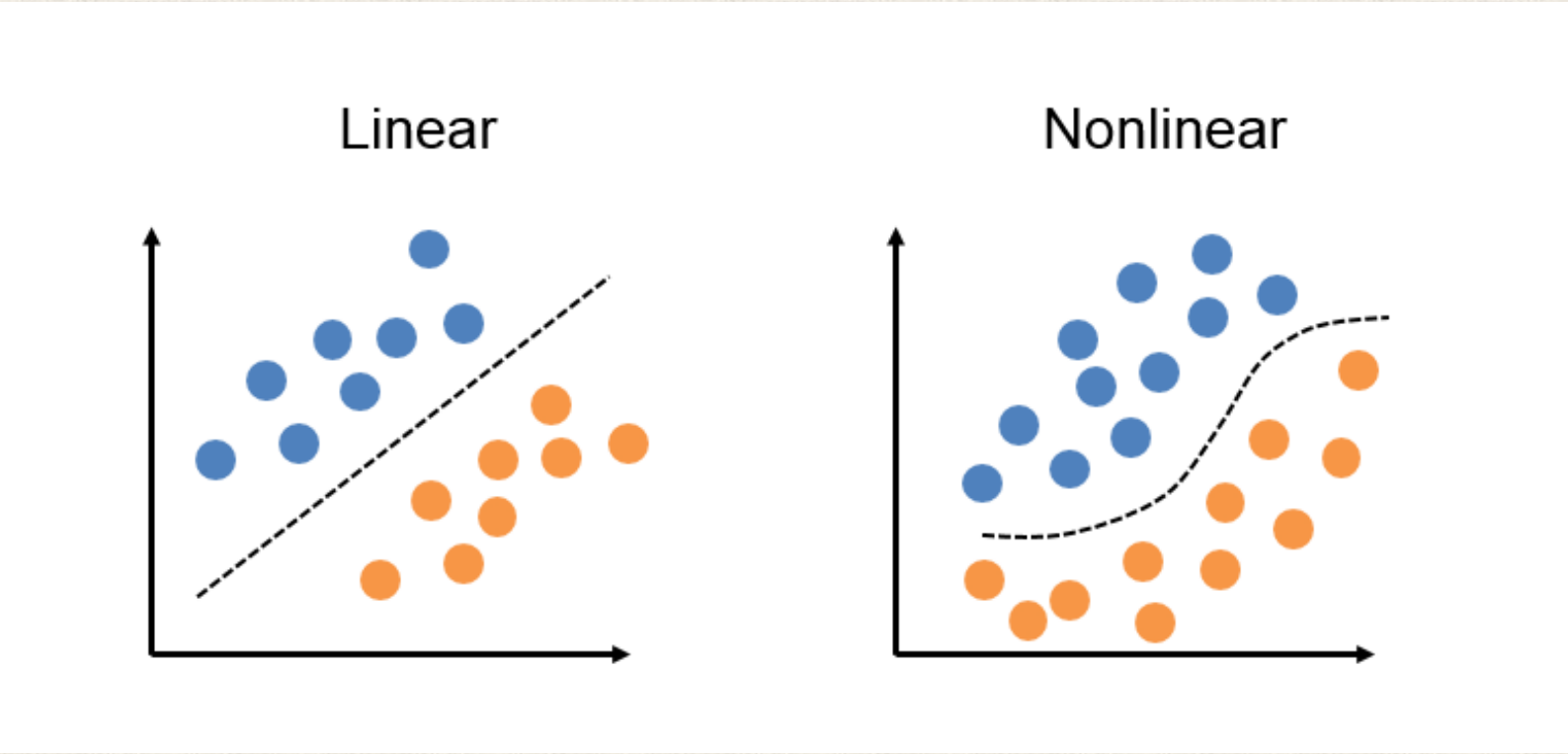
e.g., Perceptron, logistic regression, SVM, artificial neural networks (ANN), nearest neighbor, etc.

Linearly separable data

Binary linearly separable case:

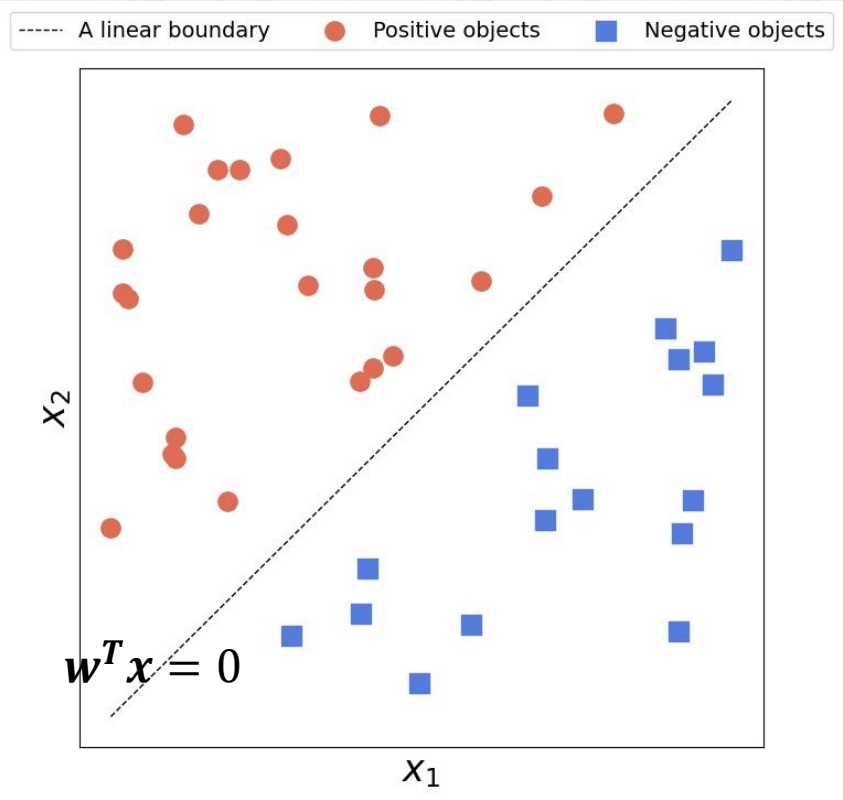
- Let's assume that we have two sets of points on ***2D Euclidian plane***, one set of points are colored blue and other colored red.
- These two sets are ***linearly separable*** if there exists at least one ***line*** in the plane with all the blue points on one side and all the red points on the other side.
- This definition generalizes to *higher-dimensional Euclidian spaces* if ***line*** is replaced by ***hyperplane***

Binary Linearly separable data sets



Equation of the linear boundary

- The equation of the hyperplane: $\boldsymbol{w^T x = 0}$
  - Any point $\boldsymbol{x}$ on the hyperplane: $\boldsymbol{w^T x = 0}$
  - Any point $\boldsymbol{x}$ above the hyperplane: $\boldsymbol{w^T x > 0}$
  - Any point $\boldsymbol{x}$ below the hyperplane: $\boldsymbol{w^T x < 0}$



The Perceptron Model (a discriminative model)

Binary linearly separable case:

- The ***perceptron classifier*** classifies an input $\boldsymbol{x}$ into one of two classes: $C_1\ and\ C_2$. The target values used in this problem is $y = 1$ for C_1 and $y = -1$ for C_2.
- The ***discriminant function*** is in a ***linear form***: $g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$,

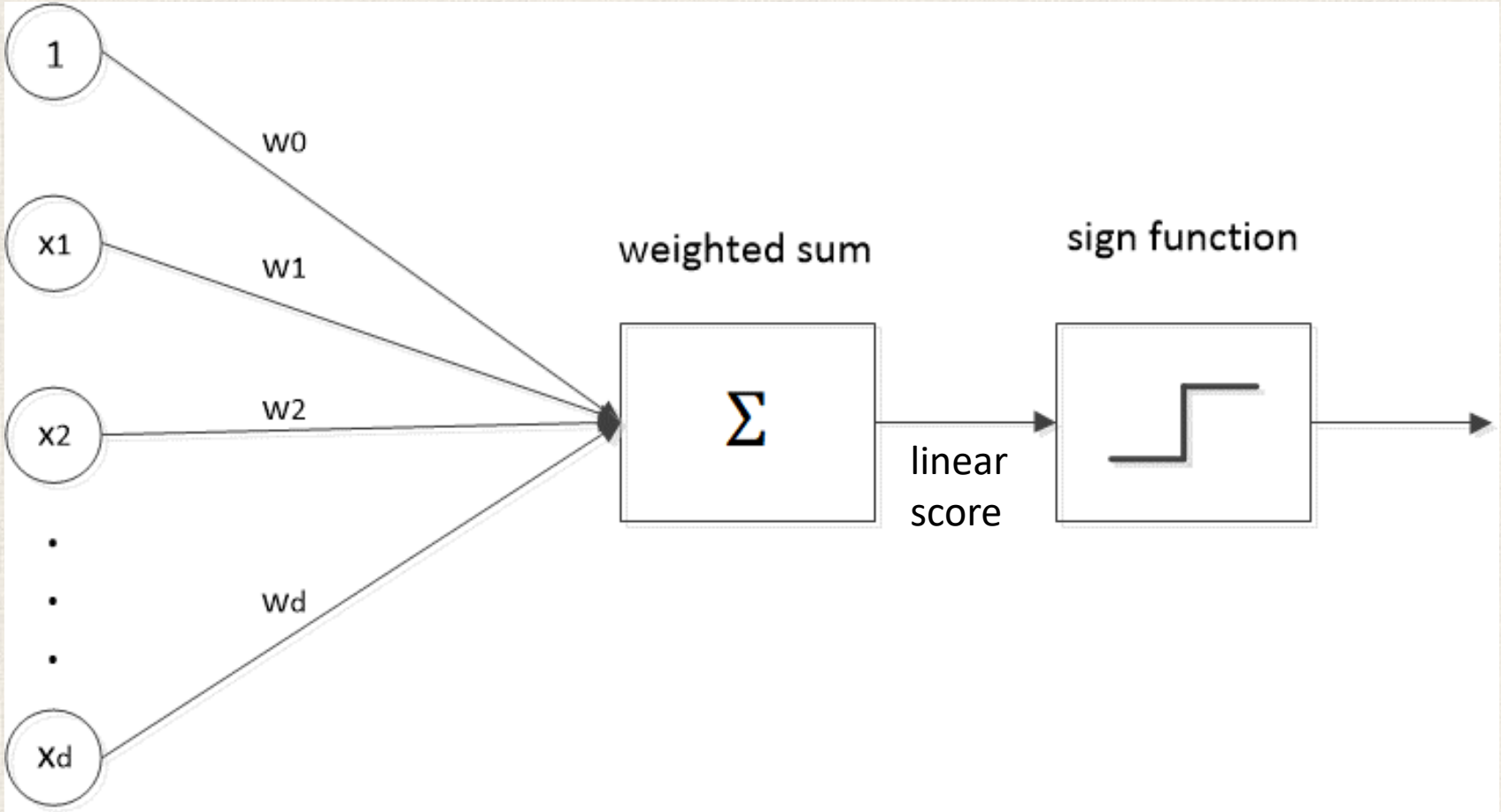
$$\boldsymbol{w}^T\boldsymbol{x}_i > 0, \qquad \boldsymbol{x}_i\ in\ C_1$$

$$\boldsymbol{w}^T\boldsymbol{x}_i \leq 0, \qquad \boldsymbol{x}_i\ in\ C_2$$

  - $\boldsymbol{w}$ is the weight vector to be determined.
  - $\boldsymbol{w}^T\boldsymbol{x}$ is called the ***linear score*** of the input $\boldsymbol{x}$

The Perceptron Model - a graphical explanation

$$y(\boldsymbol{x}) = f(\boldsymbol{w}^T \boldsymbol{x})$$



Binary linearly separable case:

- Notice that in this algorithm, we are seeking the weight vector $\boldsymbol{w}$ such that for any input $\boldsymbol{x}_i$ in the given data set,

$$\boldsymbol{w}^T \boldsymbol{x}_i > 0, \qquad \boldsymbol{x}_i \; in \; C_1 \;\; (y_i = 1)$$

$$\boldsymbol{w}^T \boldsymbol{x}_i \leq 0, \qquad \boldsymbol{x}_i \; in \; C_2 \;\; (y_i = -1)$$

This is equivalent to

$$\boldsymbol{w}^T \boldsymbol{x}_i y_i \geq 0, \qquad any \; \boldsymbol{x}_i \in D$$

Where $y_i \in \{1, -1\}$ is the target value of the input $\boldsymbol{x}_i$. $\boldsymbol{w}^T \boldsymbol{x}_i y_i$ is the ***normalized linear score*** of input $\boldsymbol{x}_i$.

The training algorithm of Perceptron Model

How to find the weight vector that can completely classifies all the data points in the training data set.

The Perceptron criterion : the error (loss) function:

Hence, we have the following perceptron criterion error function:

$$E_p(\boldsymbol{w}) = -\sum_{i \in M} \boldsymbol{w}^T \boldsymbol{x}_i y_i$$

Where, M denote the ***set of all misclassified instances***.

The perceptron criterion associates zero error with any instance that is correctly classified; for a misclassified instance $\boldsymbol{x}_i$, it tries to minimize the quantity $-\boldsymbol{w}^T \boldsymbol{x}_i y_i$

Show that for any **misclassified** instance, $-\boldsymbol{w}^T \boldsymbol{x}_i y_i \geq 0$

- An instance $\boldsymbol{x}_i$ is misclassified using the decision rule:

$$\boldsymbol{w}^T \boldsymbol{x}_i > 0, \qquad \boldsymbol{x}_i \; in \; C_1$$

$$\boldsymbol{w}^T \boldsymbol{x}_i \leq 0, \qquad \boldsymbol{x}_i \; in \; C_2$$

- Case 1: $\boldsymbol{x}_i$ is a positive instance ($y_i = +1$) but classified as a negative instance. $\boldsymbol{w}^T \boldsymbol{x}_i \leq 0$, hence, $\boldsymbol{w}^T \boldsymbol{x}_i y_i \leq 0$
- Case 2: $\boldsymbol{x}_i$ is a negative instance ($y_i = -1$) but classified as a positive instance. $\boldsymbol{w}^T \boldsymbol{x}_i > 0$, hence, $\boldsymbol{w}^T \boldsymbol{x}_i y_i < 0$

The Gradient Descent Procedure

To find the weight vector $\boldsymbol{w}$ that minimize the *perceptron criterion error function* $E_p(\boldsymbol{w})$, we use the following ***gradient descent procedure:***

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta * \nabla E_p(\boldsymbol{w}^k)$$

Where,

- k is an integer that indexes the steps of the algorithm;
- η is a positive scale factor or learning rate that set the step size;
- $\nabla E_p(\boldsymbol{w})$ is the ***gradient*** vector of the *perceptron criterion error function*.

- To find the weight vector $\boldsymbol{w}$ that minimize the *perceptron criterion error function* $E_p(\boldsymbol{w})$, we use the following ***gradient descent procedure:***

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta \nabla E_p(\boldsymbol{w}^k)$$

- Gradient descent is based on the observation that if a multi-variable function $E_p(\boldsymbol{w})$ is defined and differentiable in a neighborhood of a point $\boldsymbol{w}$**,** then $E_p(\boldsymbol{w})$ decreases fastest if one goes from $\boldsymbol{w}$ in the direction of the negative gradient of E_p at $\boldsymbol{w}$**,** $-\nabla E_p(\boldsymbol{w})$.

- It follows that if $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta \nabla E_p(\boldsymbol{w}^k)$, and for η ***small enough***, one starts with a guess $\boldsymbol{w}^0$ for the local minimum of E_p, and consider the sequence $\boldsymbol{w}^0, \boldsymbol{w}^1, \boldsymbol{w}^2$, …, we will have a monotonic sequence

$$E_p(\boldsymbol{w}^0) \geq E_p(\boldsymbol{w}^1) \geq E_p(\boldsymbol{w}^2) \geq \cdots,$$

so the sequence $\boldsymbol{w}^k, k \geq 0$ converges to the desired local minimum.

- To find the weight vector $\boldsymbol{w}$ that minimize the *perceptron criterion error function* $E_p(\boldsymbol{w})$, we use the following ***gradient descent procedure:***

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta \nabla E_p(\boldsymbol{w}^k)$$

Where, k is an integer that indexes the steps of the algorithm; η^k is a positive scale factor or learning rate that set the step size; $\nabla E_p(\boldsymbol{w})$ is the gradient vector of the *perceptron criterion error function*.

$$\nabla E_p(\boldsymbol{w}) = -\sum_{i \in M} \boldsymbol{x}_i y_i$$

- Substitute this into weight update expression, we obtain:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \eta \sum_{i \in M} \boldsymbol{x}_i y_i$$

The Perceptron training Algorithms

When should the weight vector be updated?

- ***Batch Training Algorithm***
- ***Sequential Training Algorithm***

Some terminologies about gradient descent algorithm

- ***Batch size*** is the number of training instances to work through before updating the weight vector
- Training dataset can be divided into one or more batches
- An ***epoch*** is one cycle through the entire training dataset
  - It usually takes more than a few epochs for a gradient descent algorithm to converge
- ***Batch gradient descent:*** batch size == N
- ***Sequential (stochastic) gradient descent:*** batch size == 1 + shuffling the training data set every epoch

Batch Training Algorithm:

Batch size = N

Begin initialize $\boldsymbol{w}$ with random values, and k(# of epoch) $\leftarrow 0$

do $k \leftarrow k + 1$

$M_k = \{\}$, $j = 0$ (index of data points)

do $j \leftarrow j + 1$

Classify $\boldsymbol{x}_j$ using current $\boldsymbol{w}$

if $\boldsymbol{x}_j$ is misclassified then append $\boldsymbol{x}_j, y_j$ to M_k

until $j = N$

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \sum_{i \in M_k} \boldsymbol{x}_i y_i$

until $M_k == \{\}$

return $\boldsymbol{w}$

End

Sequential Training Algorithm:

Begin initialize $\boldsymbol{w}$ with random values, and $k \leftarrow 0$ (index of points in the training dataset)

do $k \leftarrow k \; mod \; N$

Classify $\boldsymbol{x}_k$ using current $\boldsymbol{w}$

if $\boldsymbol{x}_k$ is misclassified, $\boldsymbol{w} = \boldsymbol{w} + \eta \boldsymbol{x}_k y_k$

$k \leftarrow k + 1$

until all instances in the training data set are properly classified

return $\boldsymbol{w}$

End

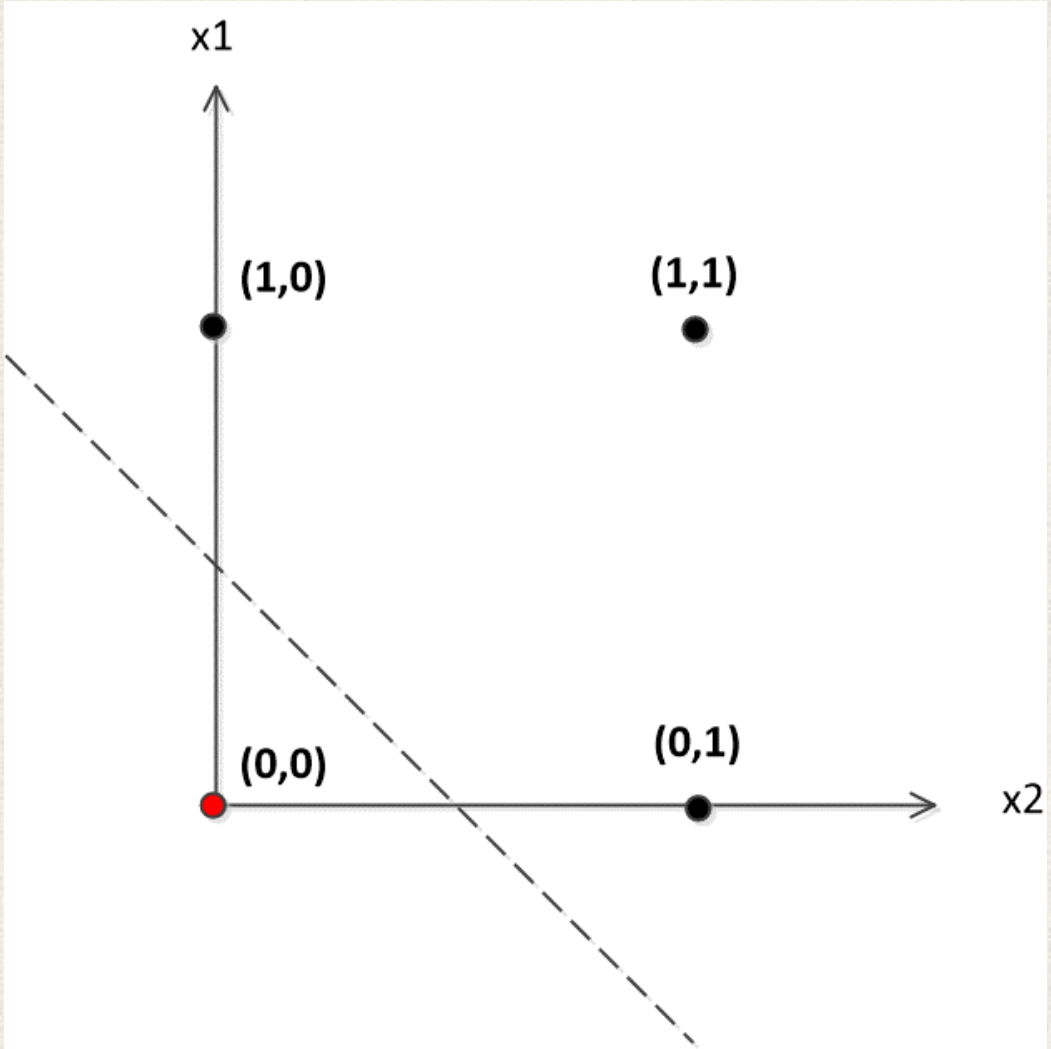
N is the size of the training dataset

The logic ***OR*** problem:

$$y = x_1 \; or \; x_2$$

$$x_1, x_2, y \in \{0,1\}$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

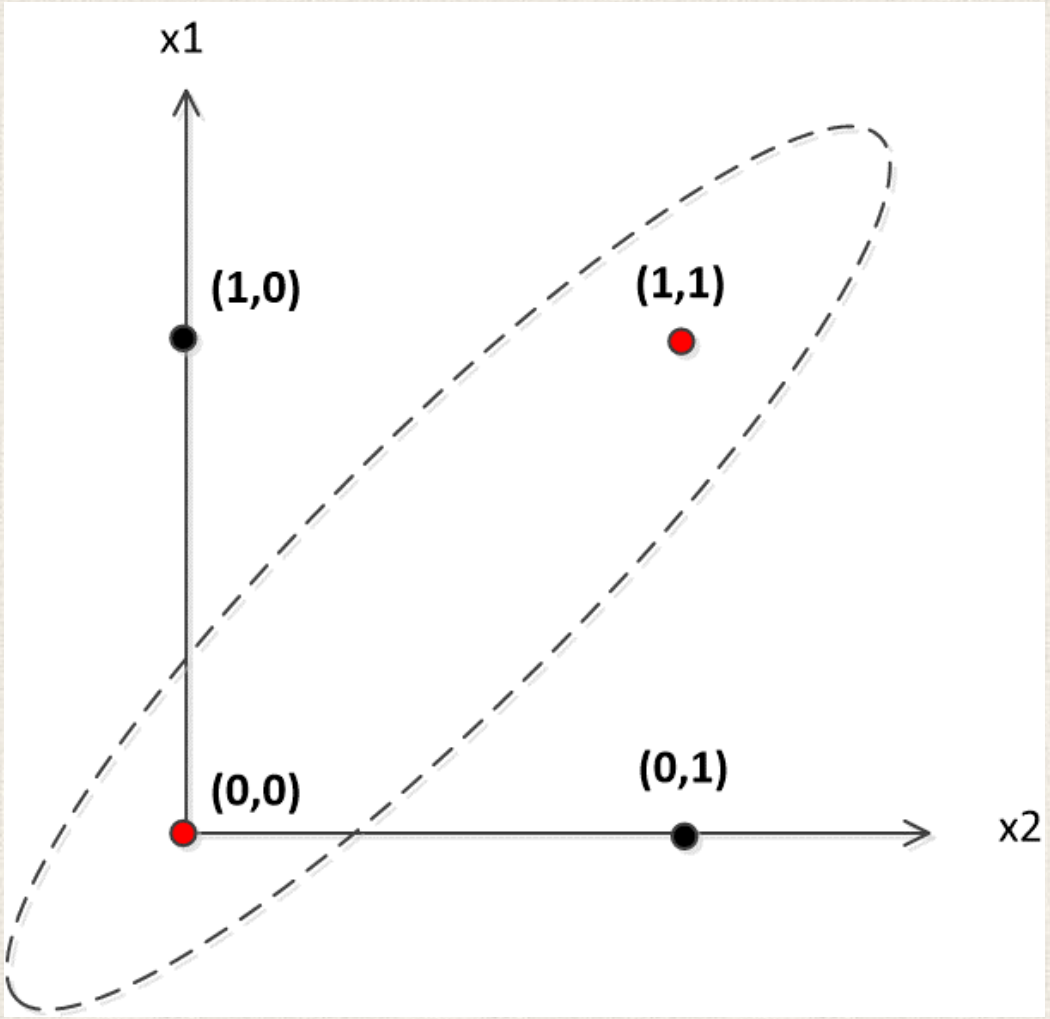


The logic ***XOR*** (exclusive ***OR***) problem:

$$y = x_1 \oplus x_2$$

$$x_1, x_2, y \in \{0,1\}$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Training a Perceptron for the logic OR problem (Sequential training)

The weight vector is $\boldsymbol{w} = [w_0 \quad w_1 \quad w_2]^T$

the extended feature vector is $\boldsymbol{x} = [1 \quad x_1 \quad x_2]^T$

Let's set the initial value of $\boldsymbol{w}$ as $\boldsymbol{w}^0 = [0 \quad 0 \quad 0]^T$

The learning rate is chosen as $\eta = 1$

Sequential weight update rule:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \boldsymbol{x}_i y_i \quad \text{if } \boldsymbol{x}_i \text{ is misclassified}$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step 1: feed the first training instance $\boldsymbol{x}_1 = [1 \quad 0 \quad 0]^T, \ \ y_1 = -1$

$$(\boldsymbol{w}^0)^T \boldsymbol{x}_1 = [0 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0, \qquad \hat{y}_1 = sign\big((\boldsymbol{w}^0)^T \boldsymbol{x}_1\big) = -1, \qquad correct$$

Then, no weight update

$$\boldsymbol{w}^1 = \boldsymbol{w}^0$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step2: feed the second training instance $\boldsymbol{x}_2 = [1 \quad 0 \quad 1]^T, \ \ y_2 = 1$

$$(\boldsymbol{w}^1)^T \boldsymbol{x}_2 = [0 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 0, \qquad \hat{y}_2 = sign\big((\boldsymbol{w}^1)^T \boldsymbol{x}_2\big) = -1, \qquad missclassified$$

Then,

$$\boldsymbol{w}^2 = \boldsymbol{w}^1 + \boldsymbol{x}_2 y_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step3: feed the third training instance $\boldsymbol{x}_3 = [1 \quad 1 \quad 0]^T,\ y_3 = 1$

$$(\boldsymbol{w}^2)^T \boldsymbol{x}_3 = [1 \quad 0 \quad 1] \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 1, \qquad \hat{y}_3 = sign\left((\boldsymbol{w}^2)^T \boldsymbol{x}_3\right) = 1, \qquad correct$$

Then, no weight update

$$\boldsymbol{w}^3 = \boldsymbol{w}^2$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step4: feed the fourth training instance $\boldsymbol{x}_4 = [1 \quad 1 \quad 1]^T, \ \ y_4 = 1$

$$\hat{y}_4(\boldsymbol{w}^3)^T\boldsymbol{x}_4 = [1 \quad 0 \quad 1]\begin{bmatrix}1\\1\\1\end{bmatrix} = 2, \qquad \hat{y}_4 = sign\left((\boldsymbol{w}^3)^T\boldsymbol{x}_4\right) = 1, \qquad correct$$

Then,

$$\boldsymbol{w}^4 = \boldsymbol{w}^3$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step5: feed the first training instance $\boldsymbol{x}_1 = [1 \quad 0 \quad 0]^T, \ \ y_1 = -1$

$$(\boldsymbol{w}^4)^T \boldsymbol{x}_1 = [1 \quad 0 \quad 1] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 1, \qquad \hat{y}_1 = sign\big((\boldsymbol{w}^4)^T \boldsymbol{x}_1\big) = 1, \qquad missclassified$$

Then,

$$\boldsymbol{w}^5 = \boldsymbol{w}^4 + \boldsymbol{x}_1 y_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step6: feed the second training instance $\boldsymbol{x}_2 = [1 \quad 0 \quad 1]^T, \quad y_2 = 1$

$$(\boldsymbol{w}^5)^T \boldsymbol{x}_2 = [0 \quad 0 \quad 1] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 1, \qquad \hat{y}_2 = sign\left((\boldsymbol{w}^5)^T \boldsymbol{x}_2\right) = 1, \qquad correct$$

Then, no weight update

$$\boldsymbol{w}^6 = \boldsymbol{w}^5$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step7: feed the third training instance $\boldsymbol{x}_3 = [1 \quad 1 \quad 0]^T, \ \ y_3 = 1$

$$(\boldsymbol{w}^6)^T \boldsymbol{x}_3 = [0 \quad 0 \quad 1] \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 0, \qquad \hat{y}_3 = sign\big((\boldsymbol{w}^6)^T \boldsymbol{x}_3\big) = -1, \qquad missclassified$$

Then,

$$\boldsymbol{w}^7 = \boldsymbol{w}^6 + \boldsymbol{x}_3 y_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step8: feed the fourth training instance $\boldsymbol{x}_4 = [1 \quad 1 \quad 1]^T, \quad y_4 = 1$

$$(\boldsymbol{w}^7)^T \boldsymbol{x}_4 = [1 \quad 1 \quad 1] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 3, \qquad \hat{y}_4 = sign\left((\boldsymbol{w}^7)^T \boldsymbol{x}_4\right) = 1, \qquad correct$$

Then, no weight update

$$\boldsymbol{w}^8 = \boldsymbol{w}^7$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step9: feed the first training instance $\boldsymbol{x}_1 = [1 \quad 0 \quad 0]^T, \ \ y_1 = -1$

$$(\boldsymbol{w}^8)^T \boldsymbol{x}_1 = [1 \quad 1 \quad 1] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 1, \qquad \hat{y}_1 = sign\big((\boldsymbol{w}^8)^T \boldsymbol{x}_1\big) = 1, \qquad missclassified$$

Then,

$$\boldsymbol{w}^9 = \boldsymbol{w}^8 + \boldsymbol{x}_1 y_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step10: feed the second training instance $\boldsymbol{x}_2 = [1 \quad 0 \quad 1]^T, y_2 = 1$

$$(\boldsymbol{w}^9)^T \boldsymbol{x}_2 = [0 \quad 1 \quad 1] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 1, \qquad \hat{y}_2 = sign\left((\boldsymbol{w}^9)^T \boldsymbol{x}_2\right) = 1, \qquad correct$$

Then, no weight update

$$\boldsymbol{w}^{10} = \boldsymbol{w}^9$$

x_1	x_2	yy
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step11: feed the third training instance $\boldsymbol{x}_3 = [1 \quad 1 \quad 0]^T, \ \ y_3 = 1$

$$(\boldsymbol{w}^{10})^T \boldsymbol{x}_3 = [0 \quad 1 \quad 1] \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 1, \qquad \hat{y}_3 = sign\left((\boldsymbol{w}^{10})^T \boldsymbol{x}_3\right) = 1, \qquad correct$$

Then, no weight update

$$\boldsymbol{w}^{11} = \boldsymbol{w}^{10}$$

x_1	x_2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step12: feed the fourth training instance $\boldsymbol{x}_4 = [1 \quad 1 \quad 1]^T, y_4 = 1$

$$(\boldsymbol{w}^{11})^T \boldsymbol{x}_4 = [0 \quad 1 \quad 1] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 2, \qquad \hat{y}_4 = sign\left((\boldsymbol{w}^{11})^T \boldsymbol{x}_4\right) = 1, \qquad correct$$

Then, no weight update

$$\boldsymbol{w}^{12} = \boldsymbol{w}^{11}$$

x1	x2	y
0	0	0 (-1)
0	1	1
1	0	1
1	1	1

Step13: feed the first training instance $\boldsymbol{x}_1 = [1 \quad 0 \quad 0]^T, \ \ y_1 = -1$

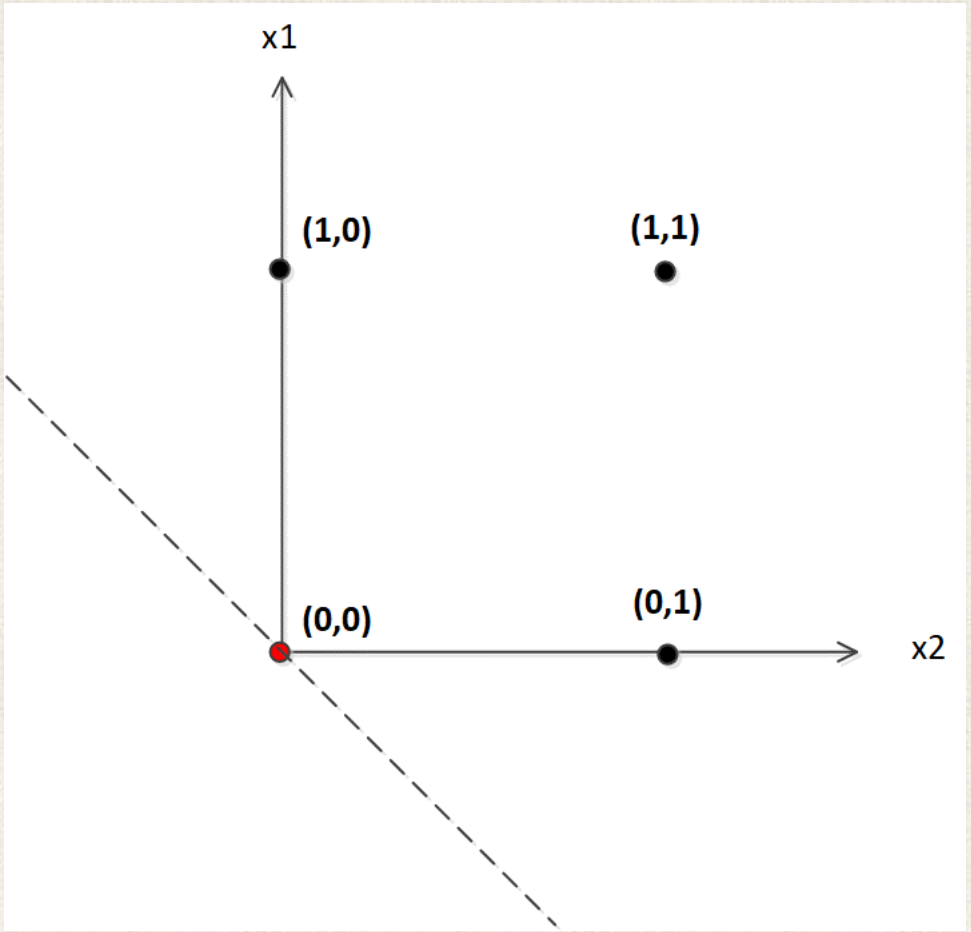
$$(\boldsymbol{w}^{12})^T \boldsymbol{x}_1 = [0 \quad 1 \quad 1] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0, \qquad \hat{y}_1 = sign\left((\boldsymbol{w}^{12})^T \boldsymbol{x}_1\right) = -1, \qquad correct$$

Now, We have all the instances classified correctly using the weight:

$$\boldsymbol{w} = [0 \quad 1 \quad 1]^T$$

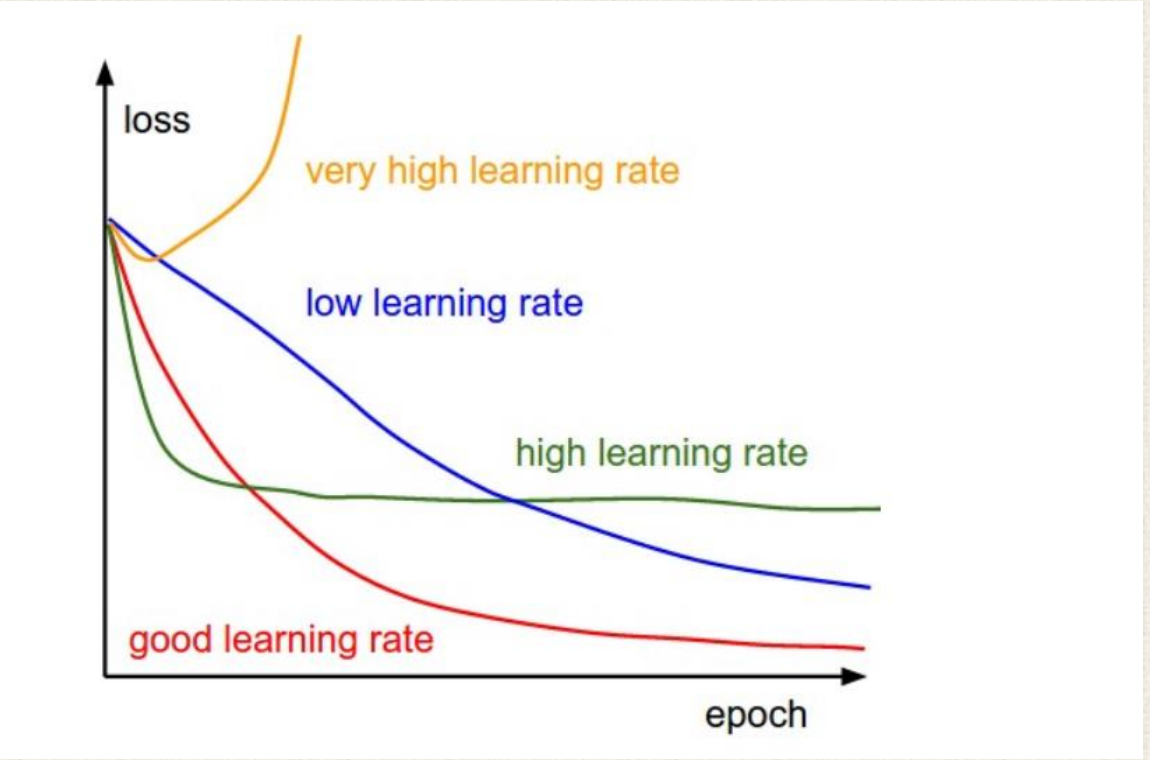
i.e.,

$$\boldsymbol{w}^T \boldsymbol{x} = x_1 + x_2$$


x1
(1,0)
(1,1)
(0,0)
(0,1)
x2

Learning rate in the gradient descent algorithm

- if the learning rate η is too small, it would take long time for the weights to converge, and the algorithm becomes computationally expensive
- If the learning rate η is too large, the weights may fail to converge and overshoot the minimum
- The most commonly used learning rates are: 0.001, 0.01, 0.1
- Fixed learning rate vs changing learning rate



Binary linearly separable case:

- Notice that in this algorithm, we are seeking the weight vector $\boldsymbol{w}$ such that for an instance $\boldsymbol{x}_i$

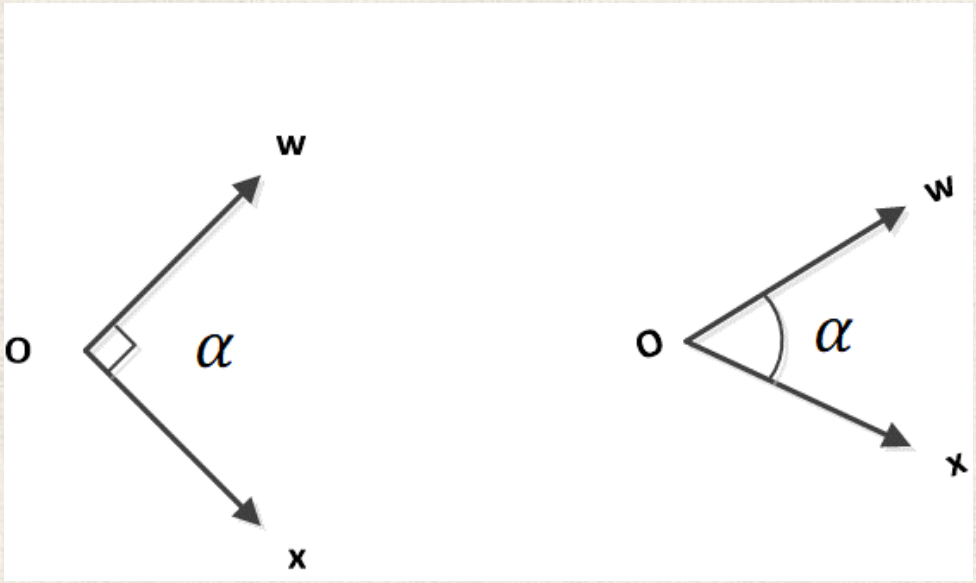
$$\boldsymbol{w}^T \boldsymbol{x}_i > 0, \qquad \boldsymbol{x}_i \; in \; C_1 \;\; (y_i = 1)$$

$$\boldsymbol{w}^T \boldsymbol{x}_i \leq 0, \qquad \boldsymbol{x}_i \; in \; C_2 \;\; (y_i = -1)$$

This is equivalent to

$$\boldsymbol{w}^T \boldsymbol{x}_i y_i \geq 0, \qquad any \; \boldsymbol{x}_i$$

Where $y_i \in \{1, -1\}$ is the target value of the input $\boldsymbol{x}_i$. $\boldsymbol{w}^T \boldsymbol{x}_i y_i$ is the ***normalized linear score*** of input $\boldsymbol{x}_i$.



- Notice that

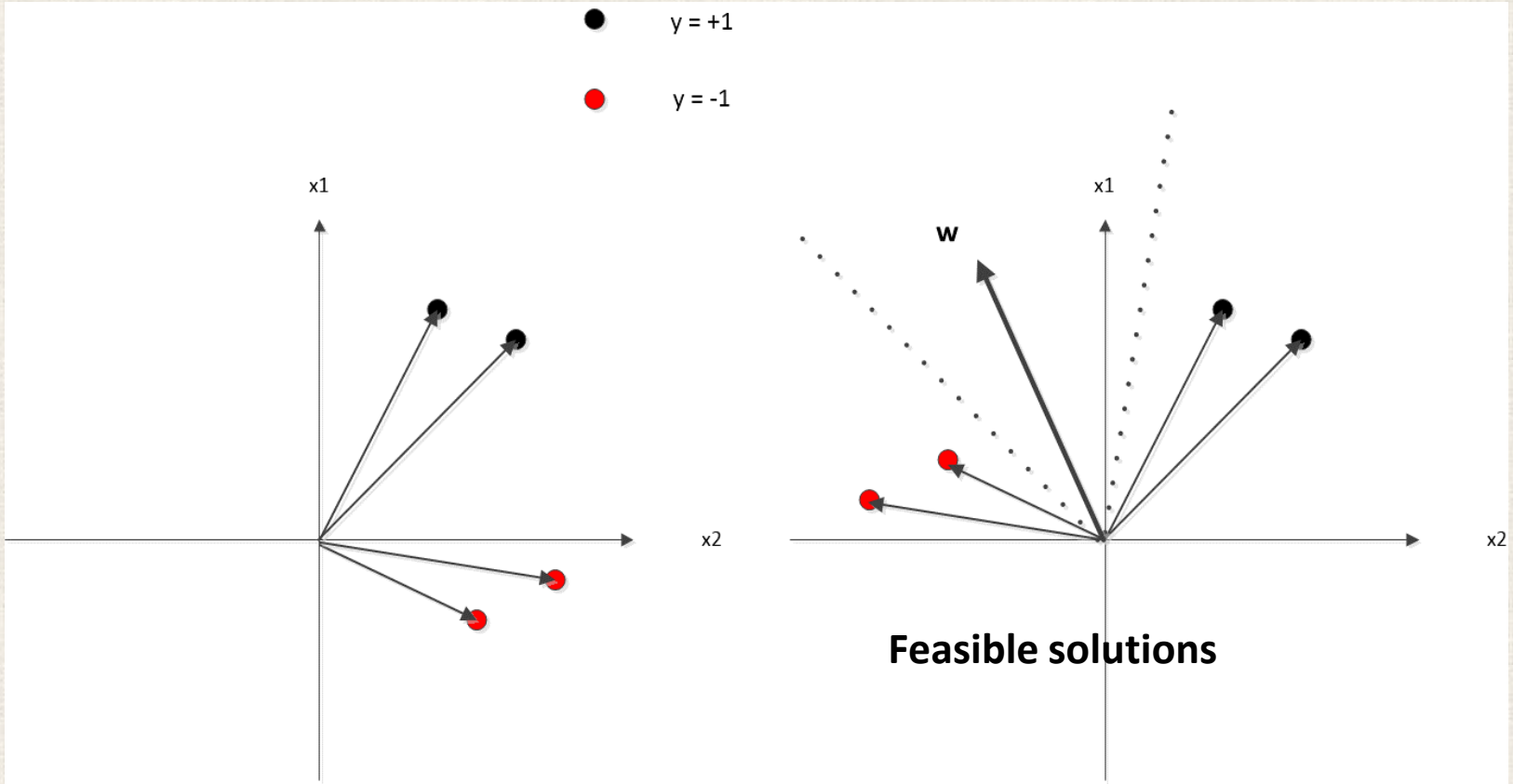
$$\boldsymbol{w}^T\boldsymbol{x} = \|\boldsymbol{w}\|\|\boldsymbol{x}\| \cos\alpha$$

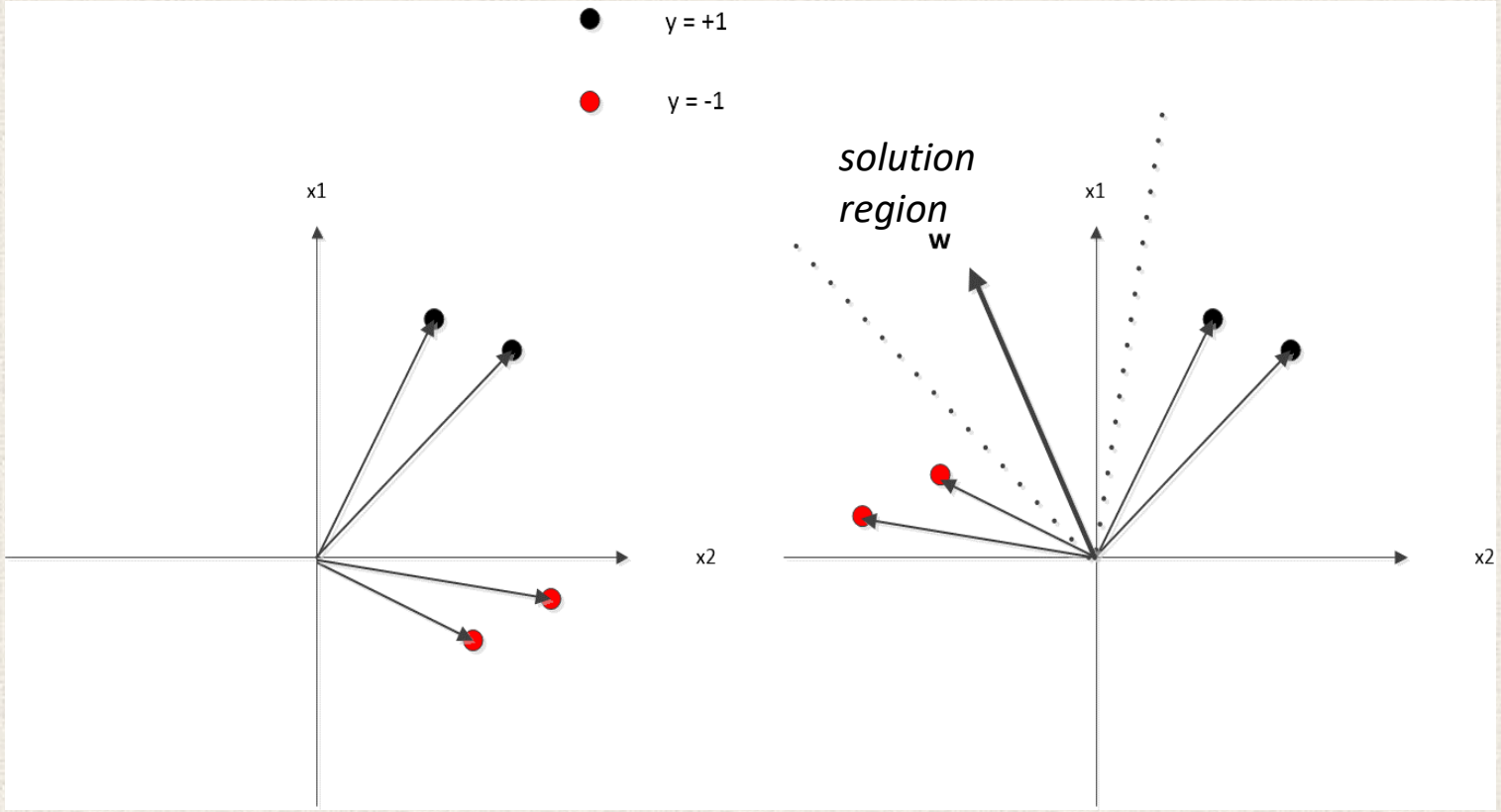
  - When $\alpha = \frac{\pi}{2}$, we have $\boldsymbol{w}^T\boldsymbol{x} = 0,\ \boldsymbol{w}\ and\ \boldsymbol{x}$ are orthogonal to each other.
  - When $|\alpha| < \frac{\pi}{2}$, we have $\boldsymbol{w}^T\boldsymbol{x} > \boldsymbol{0}$

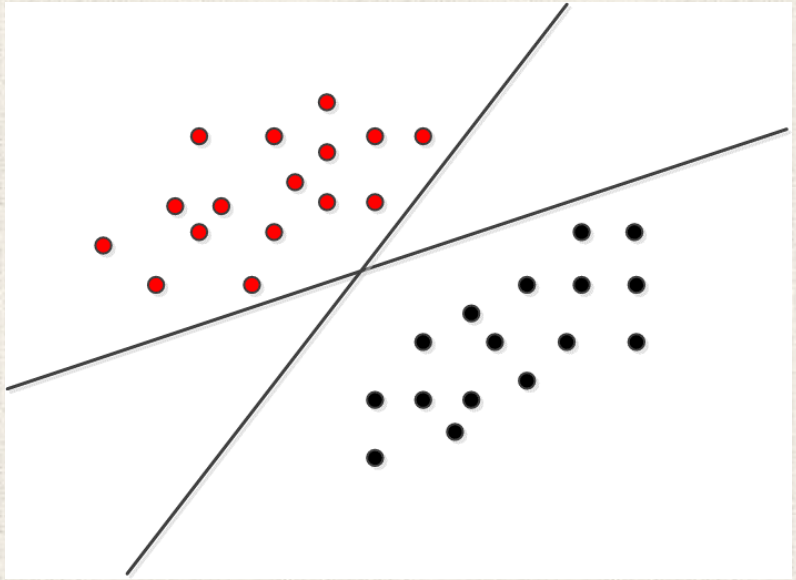
- we are seeking the weight vector $\boldsymbol{w}$ such that for an instance $\boldsymbol{x}_i$

$$\boldsymbol{w}^T \boldsymbol{x}_i y_i \geq 0, \qquad any\ \boldsymbol{x}_i, i = 1, \ldots, N$$

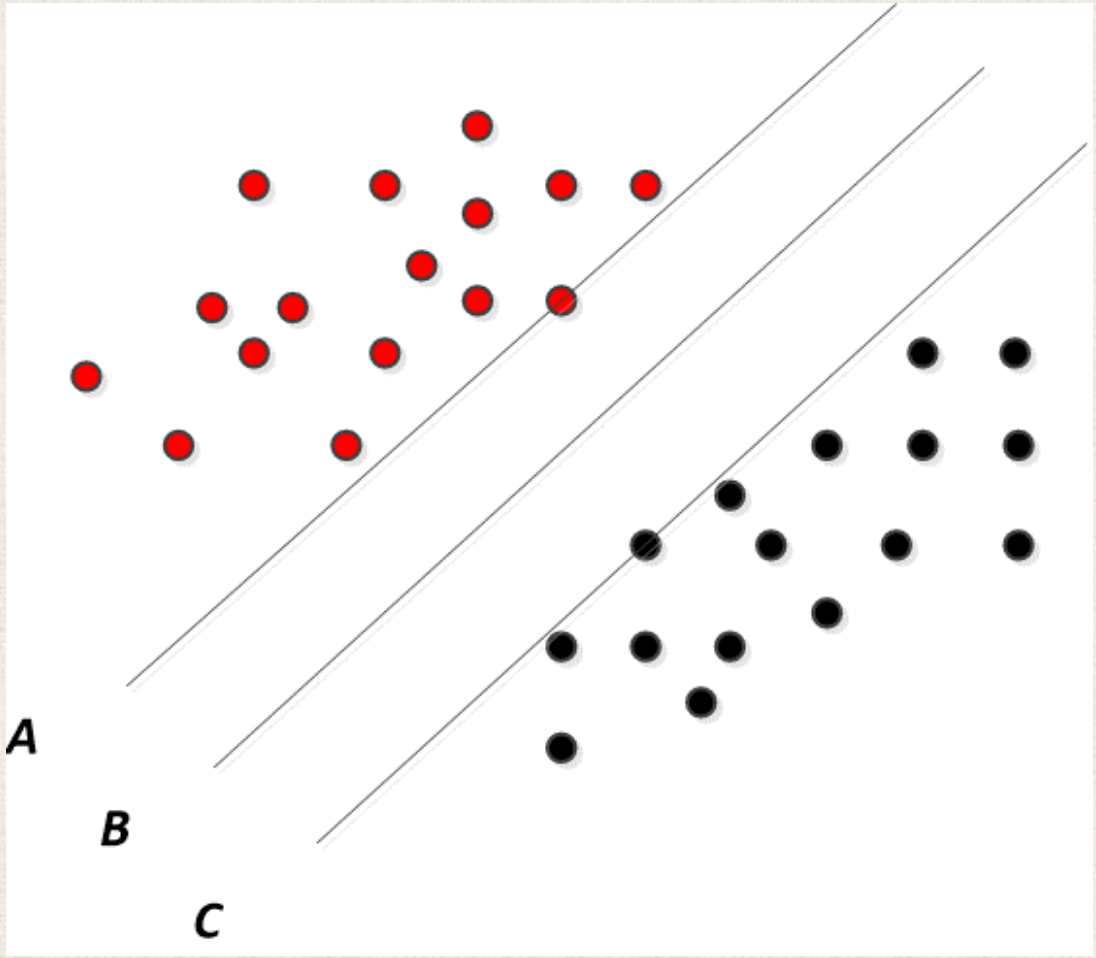
- Any weight vector $\boldsymbol{w}$ within the ***solution region*** can classify all the instances correctly. $\boldsymbol{w}$ is called a ***solution vector***.







- Any weight vector ***w*** within the ***solution region*** can classify all the instances correctly. ***w*** is called a ***solution vector***.
- Solution vectors are not unique!

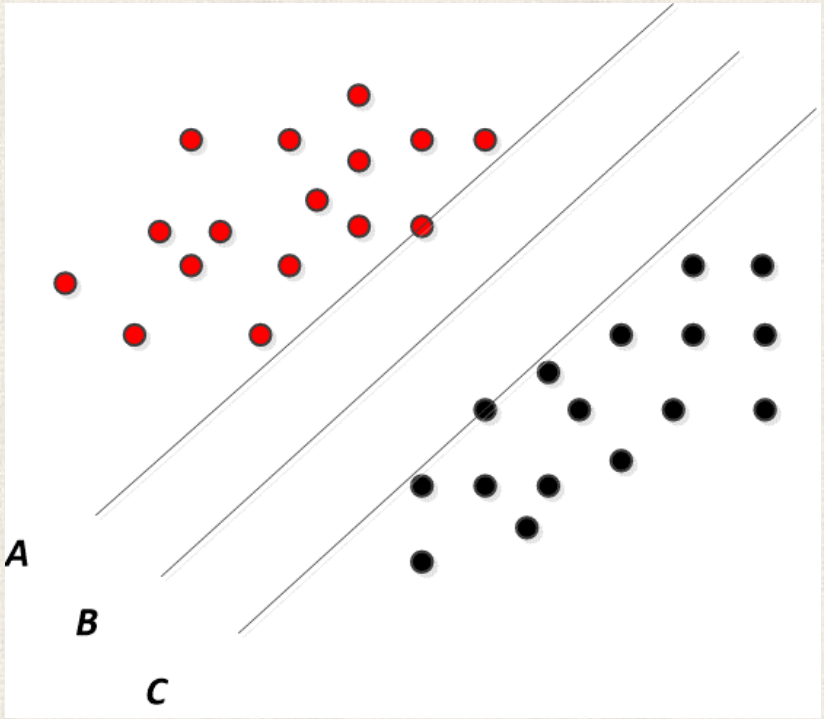

A
B
C

- There are several ways to impose additional requirements to constrain the solution vector.
- One possibility is to seek minimum length weight vector satisfying

$$\boldsymbol{w}^T \boldsymbol{x}_i y_i \geq b, \qquad for\ all\ i$$

Where $b > 0$ is a constant called ***margin***.

- ***The result solution is more likely to classify new test instances correctly***

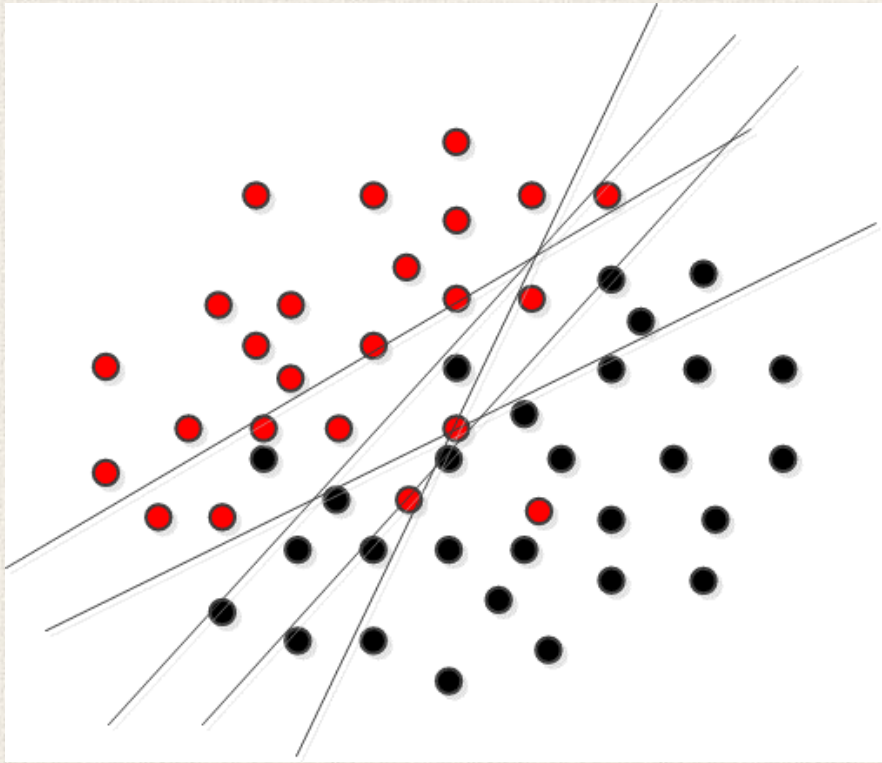


Non-linearly separable Behavior

When apply the Perceptron model to a not linearly separable data set, what is going to happen?

- No guarantee of the convergence
- No guarantee of the performance of the trained model

What should we do?



The Voted and Averaged Perceptron

Training:

Input: a labeled training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$
number of epochs T
Output: a list of weighted Perceptrons $\{(\boldsymbol{w}_1, c_1), \ldots, (\boldsymbol{w}_k, c_k)\}$
Initialize: $k \leftarrow 0, w_1 = 0, c_1 = 0$

Repeat T times: { %training stops after T epochs

for $i = 1, \ldots, m$

compute prediction $\hat{y} = sign(\boldsymbol{w}_k^T \boldsymbol{x}_i)$
if $y_i == \hat{y}$
$c_k = c_k + 1$ (times of "survive")
else
$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \boldsymbol{x}_i y_i$
$c_{k+1} = 1$
$k = k + 1$

}
Return $\{(\boldsymbol{w}_1, c_1), \ldots, (\boldsymbol{w}_k, c_k)\}$

Test:

Given: a list of weighted Perceptrons $\{(\boldsymbol{w}_1, c_1), \ldots, (\boldsymbol{w}_k, c_k)\}$
an unseen test instance $\boldsymbol{x}$

The ***voted perceptron*** computes a predicted label $\hat{y}$ as follows:

$$s = \sum_{i=1}^{k} c_i sign(\boldsymbol{w}_i^T \boldsymbol{x}) \; ; \;\; \hat{y} = sign(s)$$

The ***averaged perceptron*** computes a predicted label $\hat{y}$ as:

$$\hat{y} = sign\left(\sum_{i=1}^{k} c_i (\boldsymbol{w}_i^T \boldsymbol{x}) \right)$$

How to enhance the Perceptron model to handle non-linearly separable data sets?

Can nonlinear feature mapping be used in Perceptron model?

Logistic Regression

A discriminative classification model

Let's consider the ***binary case***: no need to be linearly separable

- Given a training data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, where $y_i \in \{0,1\}, i = 1, \dots, N$.
- *Logistic regression method directly construct the* ***posterior probability*** *that a data point is in class* C_1 *or* C_2.

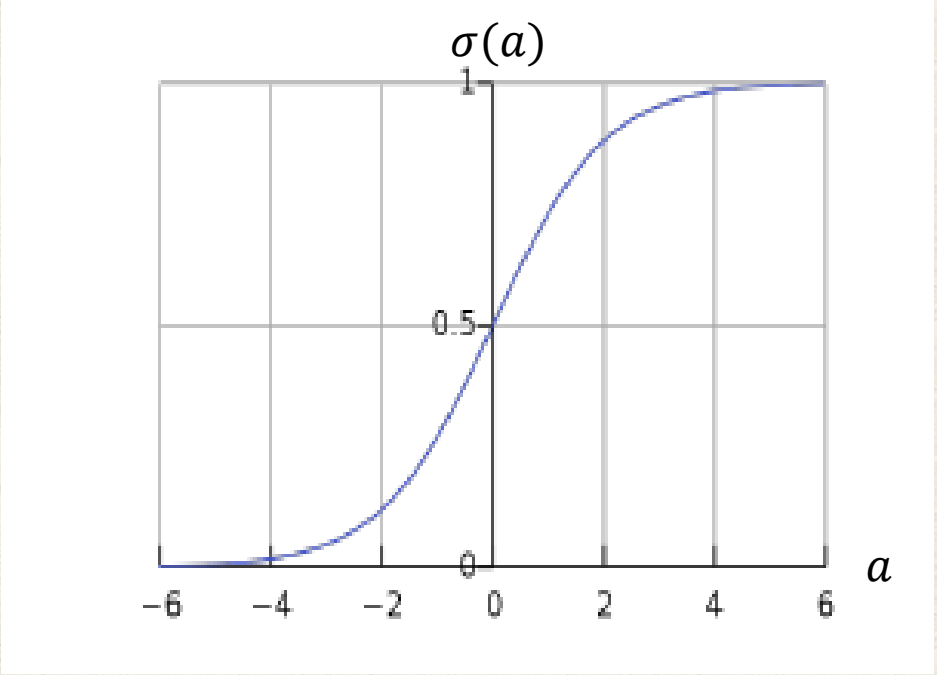
- *Logistic regression method directly construct the* ***posterior probability*** *that a data point is in class* C_1 *or* C_2.

- It converts the raw linear "*score*" $\boldsymbol{w}^T\boldsymbol{x}$ to a probability between 0 and 1 by applying a ***sigmoid (logistic)*** function $\sigma(\boldsymbol{w}^T\boldsymbol{x})$, where,

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

- A nice property of the ***sigmoid*** function

$$\frac{d\sigma(a)}{da} = \sigma(a)(1 - \sigma(a))$$



- To classify an arbitrary point $\boldsymbol{x}$, we use the **sigmoid (logistic)** function to output a posterior probability over the class C_1 and C_2:

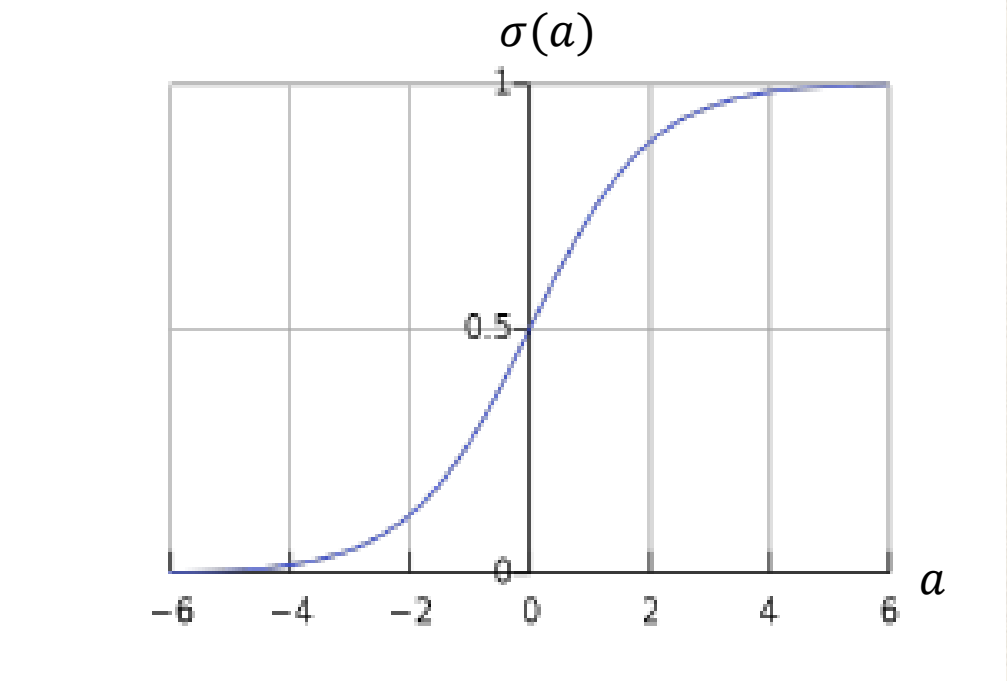
$$P(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}), \qquad P(C_2|\boldsymbol{x}) = 1 - \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

- We classify $\boldsymbol{x}$ as the class with the maximum ***posterior probability***:

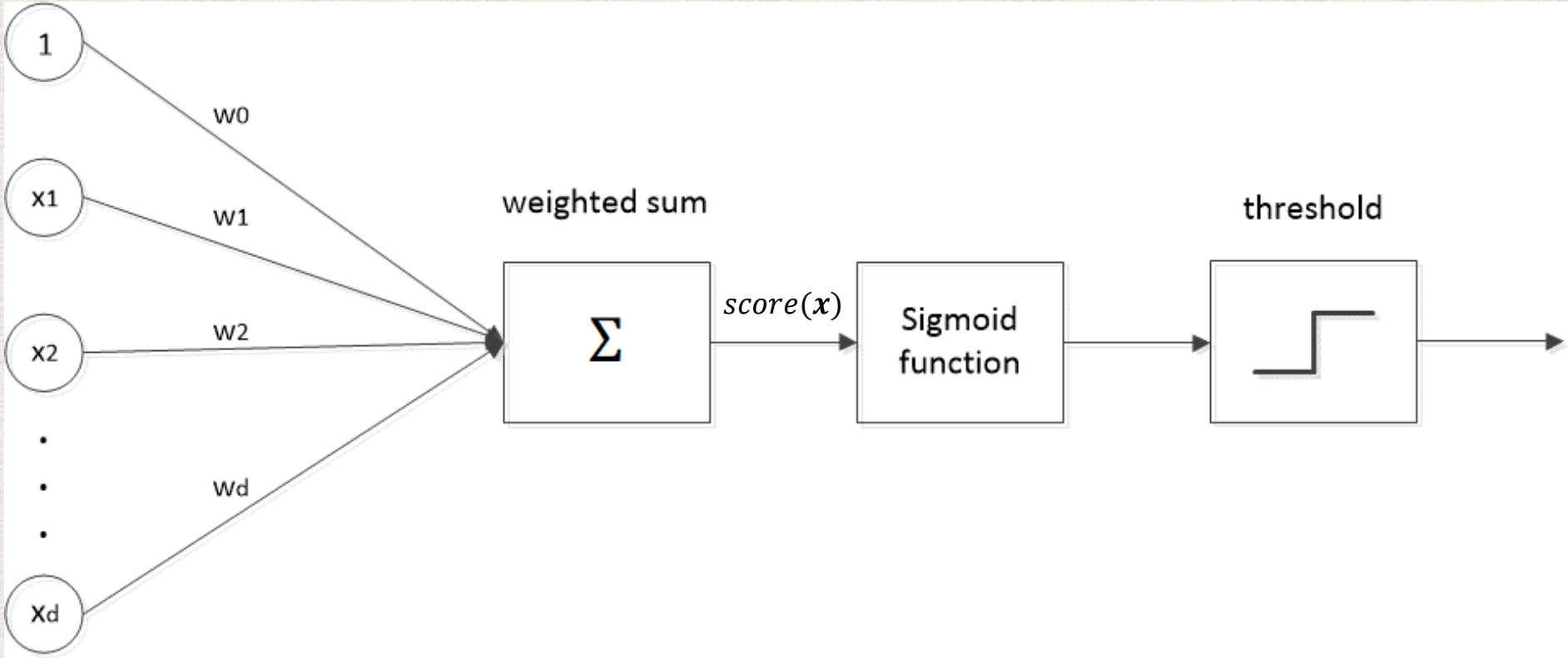
$$\hat{y} = \max_{k=1,2} P(C_k|\boldsymbol{x}) = \begin{cases} 1, & if\ \sigma(\boldsymbol{w}^T\boldsymbol{x}) \geq 0.5 \\ 0, & otherwise \end{cases}$$

- Equivalently, we classify $\boldsymbol{x}$ as (the decision rule):

$$\hat{y} = \begin{cases} 1, & if\ \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ 0, & otherwise \end{cases}$$



Graphical model of Binary Logistic Regression



Logistic Regression - A discriminative method

- Perceptron and Logistic regression are both discriminative method
- Perceptron directly work on the discriminant function to find the decision boundary
- Logistic Regression model use posterior probability $P(C_1|\boldsymbol{x})$ to make classification decision
- Logistic Regression does not calculate the posterior probability using likelihood and prior like in Bayes Theorem. Instead, it constructs the posterior probability by learning from the training data.
- Logistic Regression is a ***discriminative method***

$$P(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}),\ P(C_2|\boldsymbol{x}) = 1 - \sigma(\boldsymbol{w}^T\boldsymbol{x})\ , \sigma(a) = \frac{1}{1+e^{-a}}$$

The error (loss) function: (with a maximum likelihood explanation)

Given a training data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, where $y_i \in \{0,1\}, i = 1, \dots, N$. We view each target value y_i as an independent sample from a ***Bernoulli distribution*** $Y_i \sim Bern(p_i)$, where p_i is a function of $\boldsymbol{x_i}$. Thus, we have the following probability:

$$P(Y_i = y_i) = \begin{cases} p_i, & if\ y_i = 1 \\ 1 - p_i, & if\ y_i = 0 \end{cases}$$

Which can also be written as follows:

$$P(Y_i = y_i) = p_i^{y_i}(1 - p_i)^{1-y_i}$$

Then the ***likelihood function*** can be written as: the probability that the specific sample (the training data set) happens given the hypothesis model (represented by the weight vector $\boldsymbol{w}$).

$$p(\boldsymbol{y}|\boldsymbol{w}) = P[(Y_1 = y_1)\ and\ (Y_2 = y_2)\ and\ \ldots and\ (Y_N = y_N)] = \prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{1-y_i}$$

Where, $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ and $p_i = P(C_1|\boldsymbol{x}_i) = \sigma(\boldsymbol{w}^T\boldsymbol{x}_i)$.

- ***The error function:***

  We can define an error function by taking the **negative** logarithm of the likelihood, which gives the ***cross-entropy*** error function in the form:

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{y}|\boldsymbol{w}) = -\sum_{i=1}^{N} \{y_i \ln p_i + (1 - y_i) \ln(1 - p_i)\}$$

- ***This is the maximum likelihood explanation of the error function!***
- ***Minimizing*** $E(\boldsymbol{w})$ ***is equivalent to Maximizing the likelihood function*** $p(\boldsymbol{y}|\boldsymbol{w})$

Logistic Regression – the training method

The error function:

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{y}|\boldsymbol{w}) = -\sum_{i=1}^{N}\{y_i \ln p_i + (1-y_i)\ln(1-p_i)\}$$

Where,

$$p_i = P(C_1|\boldsymbol{x}_i) = \sigma(\boldsymbol{w}^T\boldsymbol{x}_i) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i}}$$

The gradient is:

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \left(-\sum_{i=1}^{N} \{ y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i) \} \right)$$

$$= -\sum_{i=1}^{N} \{ y_i \nabla_{\boldsymbol{w}} \, ln \, p_i + (1 - y_i) \, \nabla_{\boldsymbol{w}} ln(1 - p_i) \}$$

$$= -\sum_{i=1}^{N} \left\{ \frac{y_i}{p_i} \nabla_{\boldsymbol{w}} p_i - \frac{(1 - y_i)}{1 - p_i} \nabla_{\boldsymbol{w}} p_i \right\} = -\sum_{i=1}^{N} \left\{ \frac{y_i}{p_i} - \frac{(1 - y_i)}{1 - p_i} \right\} \nabla_{\boldsymbol{w}} p_i$$

Let's find out $\nabla_{\boldsymbol{w}} p_i$, with $p_i = P(C_1|\boldsymbol{x}_i) = \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) = \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}_i}}$.

Note that $\nabla_a \sigma(a) = \sigma(a)\big(1 - \sigma(a)\big)$, and from the chain rule of derivative, we have,

$$\nabla_{\boldsymbol{w}} p_i = \nabla_{\boldsymbol{w}} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)$$

$$= \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)\big(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)\big)\boldsymbol{x}_i$$

$$= p_i(1 - p_i)\boldsymbol{x}_i$$

Hence, we have,

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = -\sum_{i=1}^{N} \left\{ \frac{y_i}{p_i} - \frac{(1-y_i)}{1-p_i} \right\} p_i(1-p_i)\boldsymbol{x}_i$$

$$= -\sum_{i=1}^{N} \{y_i(1-p_i) - (1-y_i)p_i\}\boldsymbol{x}_i$$

$$= -\sum_{i=1}^{N} (y_i - p_i)\boldsymbol{x}_i$$

The gradient descent update is:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla_{\boldsymbol{w}} E(\boldsymbol{w}_k) = \boldsymbol{w}_k + \eta \sum_{i=1}^{N} (y_i - p_i)\boldsymbol{x}_i$$

Where, $p_i = \sigma(\boldsymbol{w}_k^T \boldsymbol{x}_i) = \frac{1}{1+e^{-\boldsymbol{w}_k^T \boldsymbol{x}_i}}$

Q1: What are the differences between Perceptron training and Logistic regression training algorithms?

Q2: Do we have batch training and stochastic training for Logistic regression?

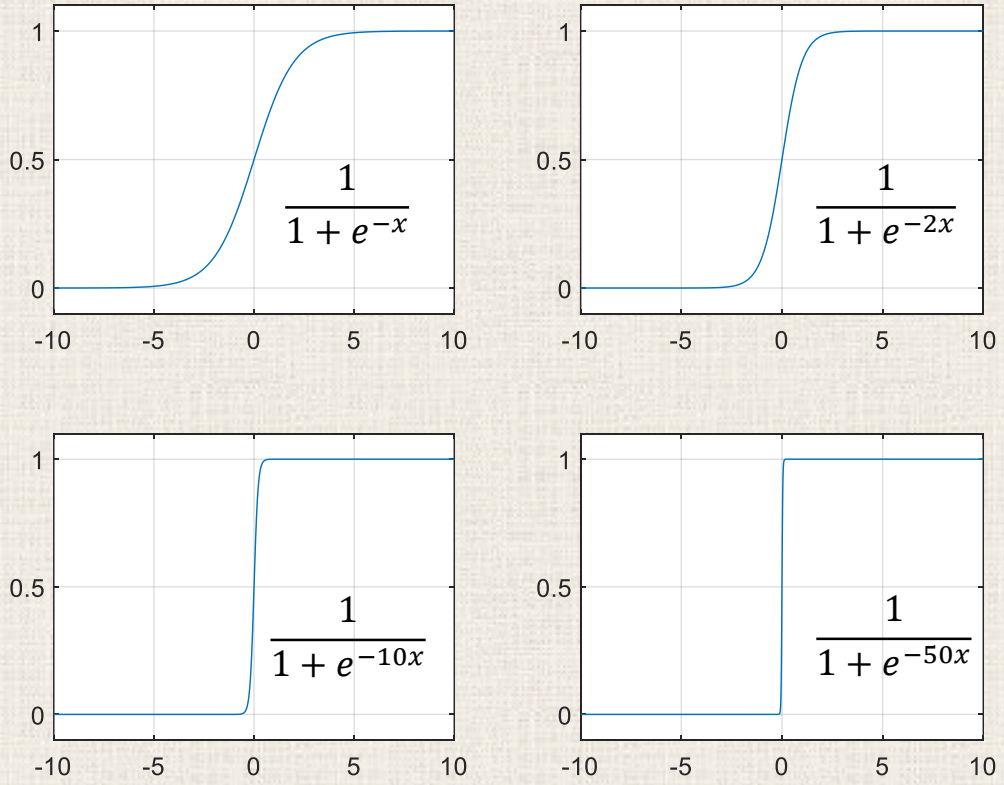
Q3: Can we use feature mapping $\phi(\boldsymbol{x})$ in Logistic Regression?

Regularized Logistic Regression

- ***When data set is linearly separable, logistic regression weights go to infinite.***

when data set is linearly separable, the conditional likelihood function is perfectly $0\ and\ 1$, the error function $E(\boldsymbol{w})$ goes to zero.

This is corresponding to the steepest sigmoid function $\frac{1}{1+e^{\boldsymbol{w}^T\boldsymbol{x}}}$ where the weights go to infinite.



Regularized Logistic Regression

- *Also, using feature mapping $\phi(x)$ may cause overfitting*
- Large number of features in the model trained using a small data set may cause overfitting
- To avoid overfitting, a regularization penalty term can be included into the error function:

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{y}|\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = -\sum_{i=1}^{N}\{y_i \ln p_i + (1-y_i)\ln(1-p_i)\} + \lambda\|\boldsymbol{w}\|_2^2$$

Regularized Logistic Regression

This new error function is equivalent to:

$$E(\boldsymbol{w}) = 1/\lambda \left\{ -\sum_{i=1}^{N} \{y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)\} \right\} + \|\boldsymbol{w}\|_2^2$$

$$E(\boldsymbol{w}) = C \left\{ -\sum_{i=1}^{N} \{y_i \, ln \, p_i + (1 - y_i) \, ln(1 - p_i)\} \right\} + \|\boldsymbol{w}\|_2^2$$

Where, $C = 1/\lambda$ is the inverse of the regularization strength (*sk-learn use regularized LR by default*).

Multinomial Logistic Regression

Multiclass Logistic Regression:

- Let's generalize logistic regression to the case where there are K classes
  - The training data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$
  - The one-hot vector encoding:
- If the i^{th} observation belongs to class C_k, instead of $y_i = k$, we use the representation $\boldsymbol{y}_i = \boldsymbol{e}_k$, the k^{th} canonical basis vector. For example, if an observation belong to the class C_2, its label representation would be: $(K = 3, k = 2)$

$$\boldsymbol{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Now, the target value becomes a K dimensional vector.

Multiclass Logistic Regression:

- Instead of a weight vector, now we have a weight matrix to be determined:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1^{\boldsymbol{T}} \\ \vdots \\ \boldsymbol{w}_K^{\boldsymbol{T}} \end{bmatrix}$$

  Where $\boldsymbol{w}_k \in R^{d+1}$ is the weight vector for the k^{th} class.

- For each input $\boldsymbol{x}_i \in R^{d+1}$, each class k is given a "score" $z_k = \boldsymbol{w}_k^T \boldsymbol{x}_i$

- In total there are K raw linear scores for an arbitrary input $\boldsymbol{x}$:

$$\boldsymbol{W}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{w}_1^T \boldsymbol{x} \\ \vdots \\ \boldsymbol{w}_K^T \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_K \end{bmatrix}$$

Multinomial Logistic Regression

Multiclass Logistic Regression:

$$\boldsymbol{W}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{w}_1^T\boldsymbol{x} \\ \vdots \\ \boldsymbol{w}_K^T\boldsymbol{x} \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_K \end{bmatrix}$$

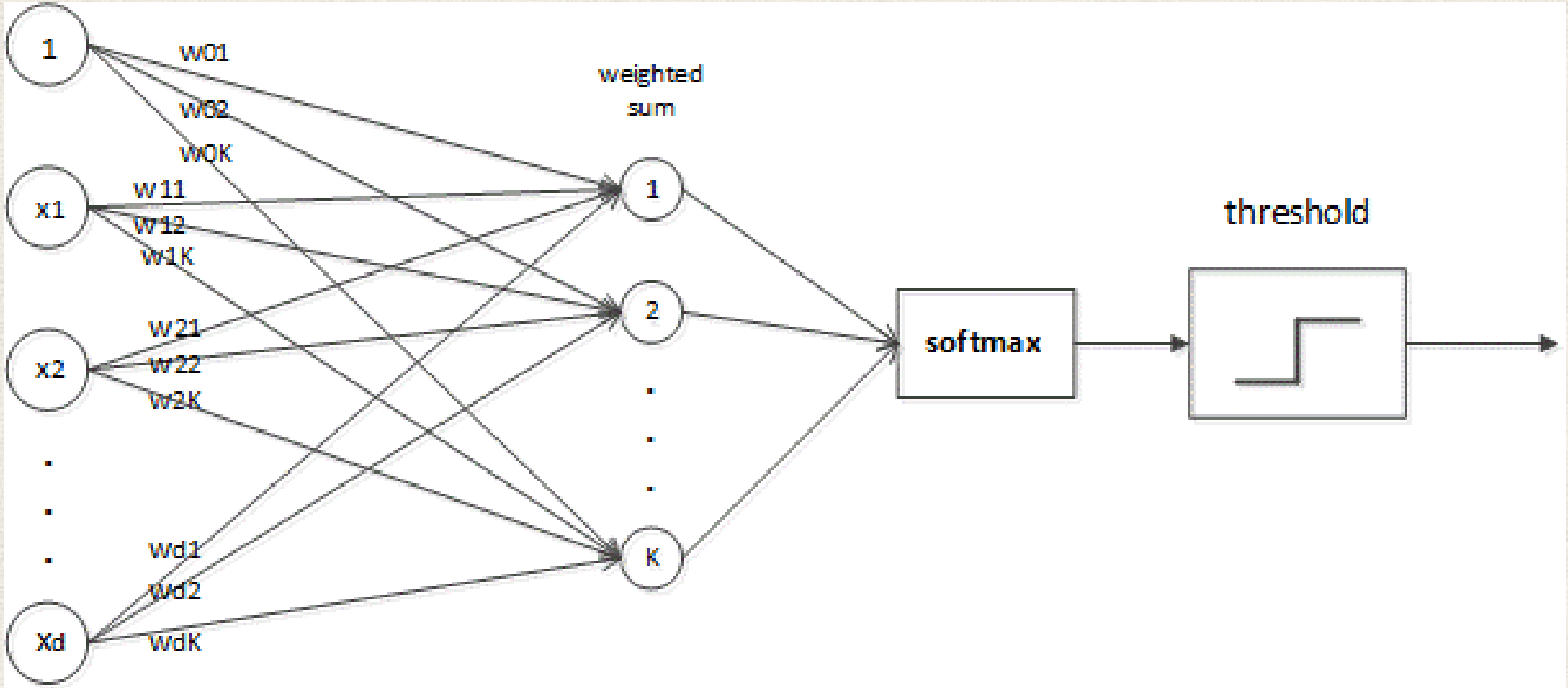
- We now must transform all of these scores into a posterior probability distribution $P(C_k|\boldsymbol{x})$. For the multiclass case, We use ***softmax*** function:

$$P(C_k|\boldsymbol{x}) = \sigma(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} = \frac{e^{\boldsymbol{w}_k^T\boldsymbol{x}}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T\boldsymbol{x}}}, \qquad k = 1,2,\dots,K$$

- We then predict the class with the maximum probability:

$$\hat{y} = \max_k P(C_k|\boldsymbol{x})$$

Graphical model of multinomial Logistic Regression



The error function for multi-class logistic regression:

- The *likelihood function* is given by:

$$p(\boldsymbol{y}|\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_k) = \prod_{i=1}^{N}\prod_{k=1}^{K}[P(C_k|\boldsymbol{x_i})]^{y_{ik}} = \prod_{i=1}^{N}\prod_{k=1}^{K}[p_{ik}]^{y_{ik}}$$

Where $\boldsymbol{y}$ is an $K \times N$ matrix of target values with elements y_{ik} and

$$p_{ik} = P(C_k|\boldsymbol{x_i}) = \frac{e^{\boldsymbol{w}_k^T \boldsymbol{x_i}}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T \boldsymbol{x_i}}}$$

- Taking the negative logarithm then gives:

$$E(\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_K) = -\ln p(\boldsymbol{y}|\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_k) = -\sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} \ln p_{ik}) = -\sum_{i=1}^{N}\sum_{k=1}^{K}\left(y_{ik} \ln\left(\frac{e^{\boldsymbol{w}_k^T \boldsymbol{x_i}}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T \boldsymbol{x_i}}}\right)\right)$$

Which is know as the ***cross-entropy*** error function for the multiclass logistic regression problem.

- Instead of finding the gradient with respect to all of the parameters of the matrix $\boldsymbol{W}$, let's find the gradient with respect to one row of $\boldsymbol{W}$ at a time:

$$\nabla_{\boldsymbol{w}_l} E(\boldsymbol{w}_1, \dots, \boldsymbol{w}_K) = \nabla_{\boldsymbol{w}_l} \left\{ -\sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_{ik} \ln \left(\frac{e^{\boldsymbol{w}_k^T \boldsymbol{x}_i}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T \boldsymbol{x}_i}} \right) \right) \right\}$$

$$= -\sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_{ik} \nabla_{\boldsymbol{w}_l} \left\{ \ln \left(\frac{e^{\boldsymbol{w}_k^T \boldsymbol{x}_i}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T \boldsymbol{x}_i}} \right) \right\} \right)$$

$$= -\sum_{i=1}^{N} \sum_{k=1}^{K} \left(y_{ik} \nabla_{\boldsymbol{w}_l} \left\{ \ln \left(\frac{e^{\boldsymbol{w}_k^T \boldsymbol{x}_i}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T \boldsymbol{x}_i}} \right) \right\} \right)$$

$$= -\sum_{i=1}^{N} (y_{il} - p_{il}) \boldsymbol{x}_i$$

Multinomial Logistic Regression

- The gradient descent update for $\boldsymbol{w}_l$ is then, $(l = 1, \ldots, K)$

$$\boldsymbol{w}_l^{t+1} = \boldsymbol{w}_l^t + \eta \sum_{i=1}^{N} (y_{il} - p_{il}) \boldsymbol{x}_i$$