

Computer Vision

CS308

Feng Zheng

SUSTech CS Vision Intelligence and Perception



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



Content

- Brief Review
- Pattern recognition - classification
 - The perception
 - Support vector machine

Brief Review

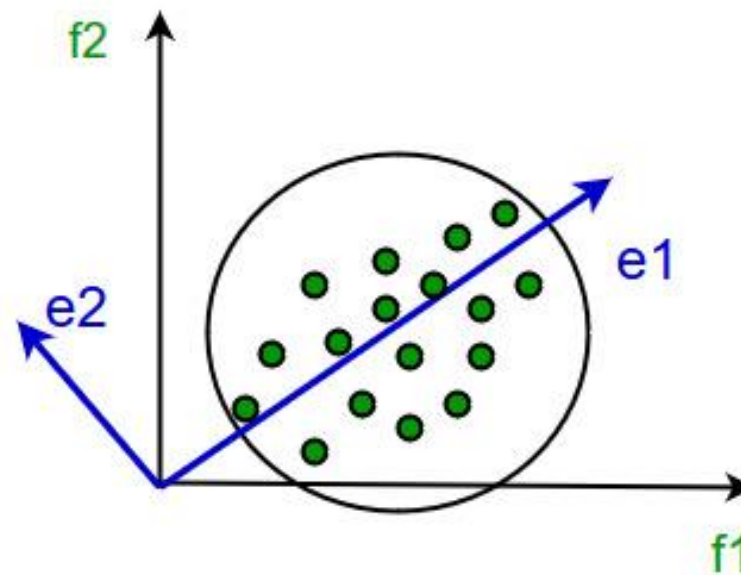


Review

- Principal component analysis (PCA)
 - Why?
 - How?
- Two matrices
 - Covariance matrix
 - Similarity matrix

(b): $\alpha_i = Xv_i^T = \sqrt{\lambda_i}\mu_i$; **(c):** $n\lambda_i = \tau_i$;

(d): $v_ix^T = \sum_{j=1}^n \alpha_i(j)x_jx^T$ (x is a new sample)





Machine Learning Problems

- Taxonomy

		<i>Supervised Learning</i>	<i>Unsupervised Learning</i>
<i>Discrete</i>	<i>Continuous</i>	classification or categorization	clustering
		regression	dimensionality reduction

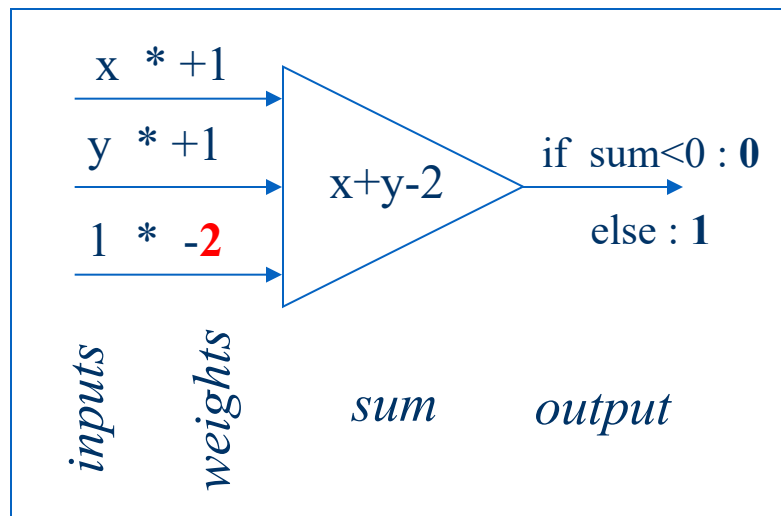
The Perception



Prehistory

- Seminal work

- W.S. McCulloch & W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, 5, 115-137
- Point out that simple artificial "neurons" could be made to perform basic logical operations such as **AND, OR and NOT**.



**Truth Table for Logical
AND**

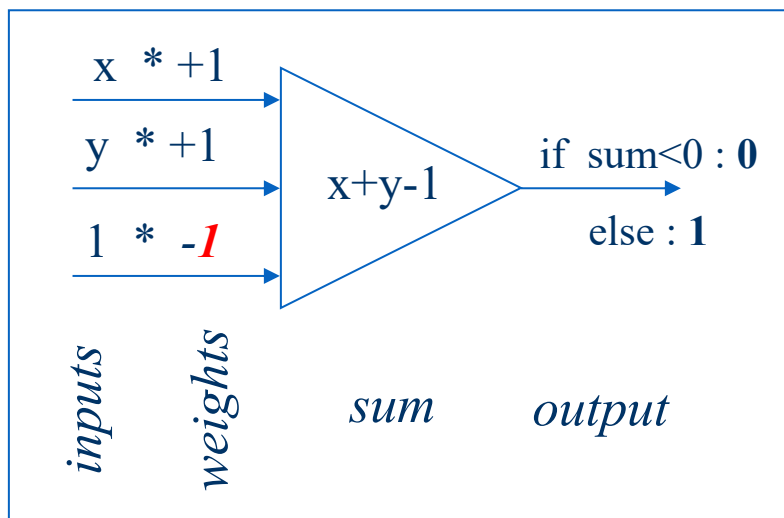
x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

inputs *output*



Nervous Systems as Logical Circuits

- Groups of these “neuronal” logic gates could carry out **any computation**, even though each neuron was very **limited**.
 - Could **computers** built from these simple units reproduce the computational power of biological brains?
 - Were biological neurons performing logical operations?



**Truth Table for Logical
OR**

x	y	$x \mid y$
0	0	0
0	1	1
1	0	1
1	1	1

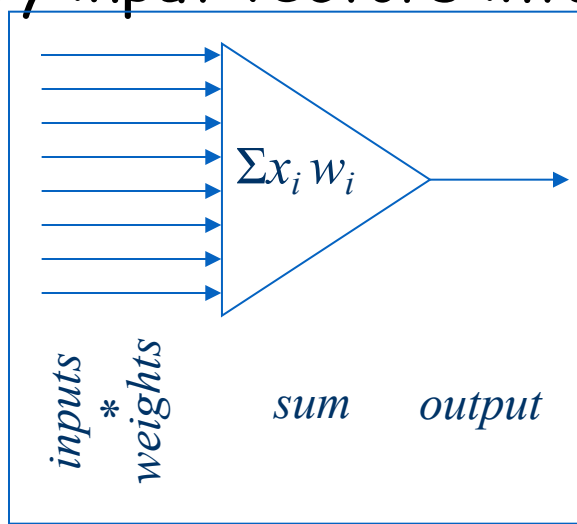
inputs *output*



The Perceptron

- Pioneer work

- Frank Rosenblatt (1962). Principles of Neurodynamics, Spartan, New York, NY.
- Subsequent progress was inspired by the **invention of learning rules** inspired by ideas from neuroscience...
- Rosenblatt's Perceptron could automatically learn to categorise or classify input vectors into types.



It obeyed the following rule:

If the sum of the weighted inputs exceeds a threshold, output 1, else output -1.

1 if $\sum input_i * weight_i > threshold$

-1 if $\sum input_i * weight_i < threshold$



Linear Neurons

- The neuron has a real-valued output which is a weighted sum of its inputs

$$\hat{y} = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

Neuron's estimate of the desired output

Weight vector

Input vector

- The aim of learning is to **minimize the discrepancy** between the desired output and the actual output
 - How do we **measure** the discrepancies? Loss
 - Do we **update** the weights after every training case? Optimization
 - Why don't we solve it **analytically**? Noisy



A Motivating Example (Analytically)

- Each day you get lunch at the cafeteria.
 - Your diet consists of fish, chips, and beer
 - You get several portions of each
- The cashier only tells you the total price of the meal
 - After several days, you should be able to figure out the price of each portion
- Each meal price gives a **linear constraint** on the prices of the portions:

$$price = x_{fish}w_{fish} + x_{chips}w_{chips} + x_{beer}w_{beer}$$



Two Ways to Solve the Equations

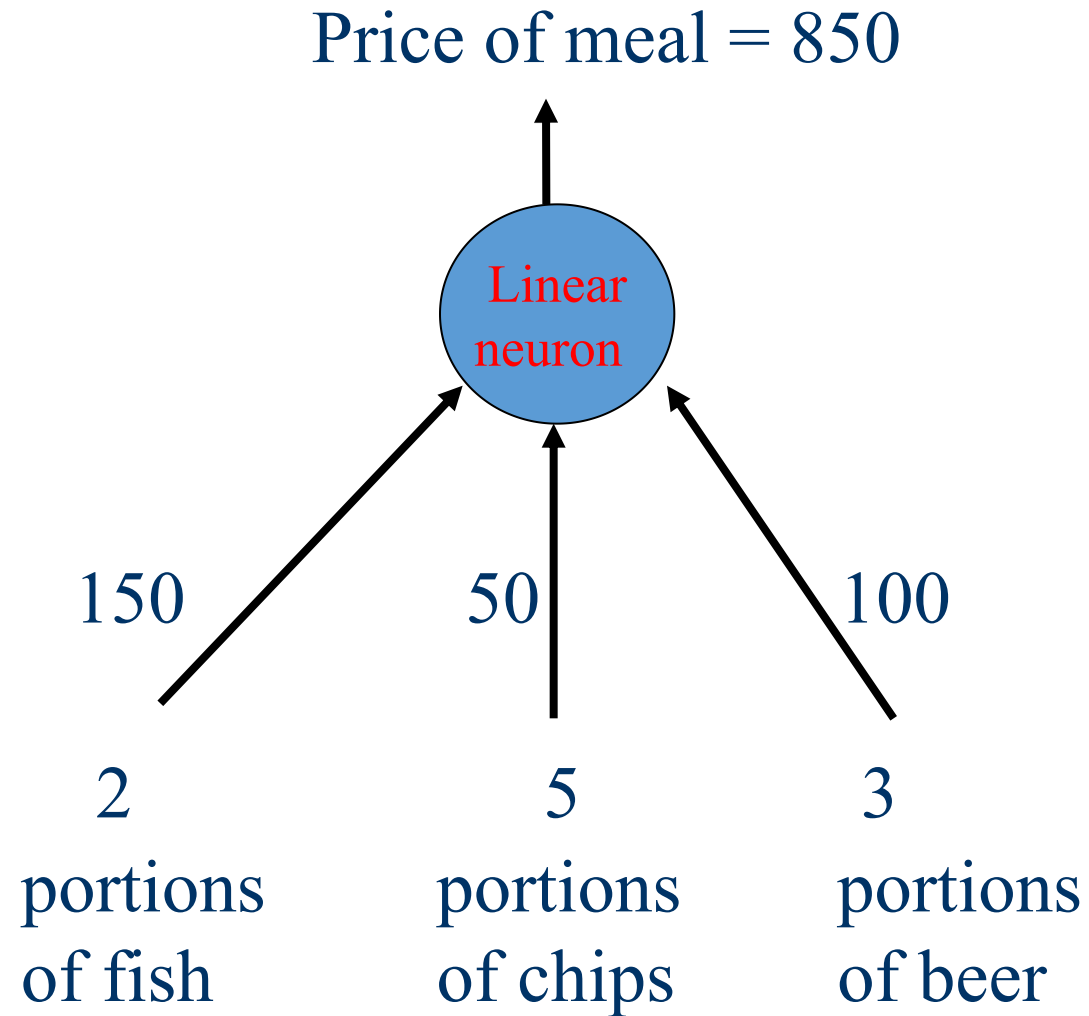
- 1. The obvious approach is just to solve a set of simultaneous **linear equations**, one per meal
- 2. But we want a method that could be implemented in a **neural network**
 - The prices of the portions are like the weights in of a linear neuron

$$\mathbf{w} = (w_{fish}, w_{chips}, w_{beer})$$

- We will start with **guesses** for the **weights** and then **adjust** the guesses to give a **better fit** to the prices given by the cashier



The Cashier's Brain



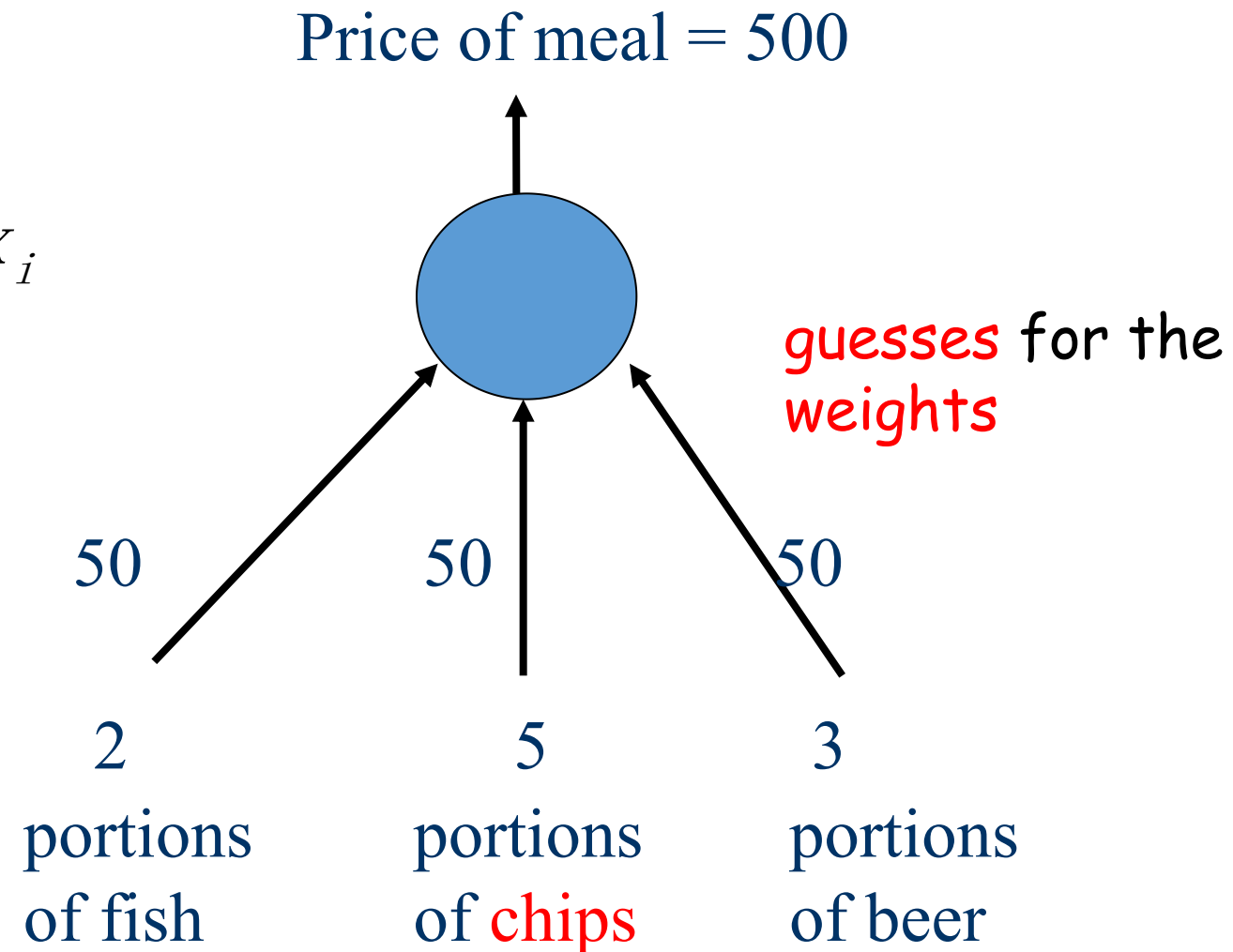


A model of the cashier's brain with arbitrary initial weights

- Residual error = 350
- The learning rule is:

$$\Delta W_i = \varepsilon x_i (y - \hat{y}) = 10x_i$$

- With a learning rate ε of 1/35, the weight changes are +20, +50, +30
- This gives new weights of
- 70, 100, 80 \leftarrow 50, 50, 50
- Notice that the weight for chips got worse!
150, 50, 100





Behavior of the iterative learning procedure

- Do the updates to the weights **always** make them get closer to their correct values? **No!**
- Does the online version of the learning procedure **eventually** get the **right** answer? **Yes**, if the **learning rate** gradually decreases in the appropriate way. 只要学习率逐渐减小到合适的值，最终会收敛到正确答案。
- How **quickly** do the weights converge to their correct values? It can be very slow if two input dimensions are highly correlated (e.g. ketchup and chips). 如果输入维度之间高度相关（例如薯条和番茄酱），则收敛会非常慢。
- Can the iterative procedure be **generalized** to much more complicated, multi-layer, non-linear nets? **YES!**



Deriving the delta rule

- Define the **error** as the squared residuals summed over all training cases:
- Now **differentiate** to get error derivatives for weights
- The batch delta rule changes the weights in proportion to their error derivatives summed over **all training cases**

$$\rightarrow E = \frac{1}{2} \sum_n (y_n - \hat{y}_n)^2$$

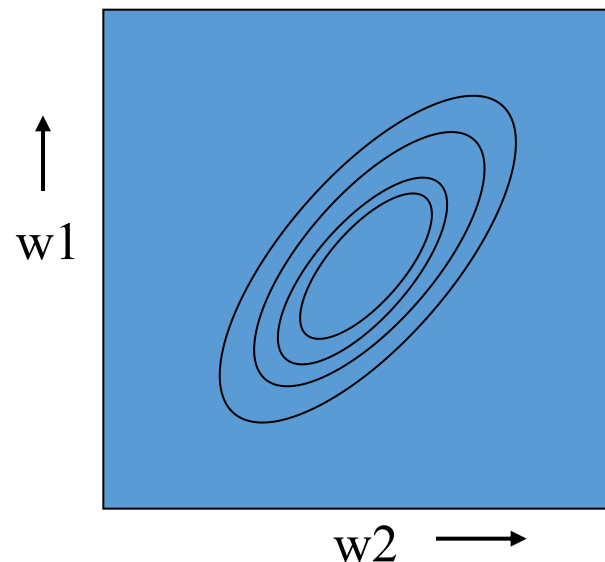
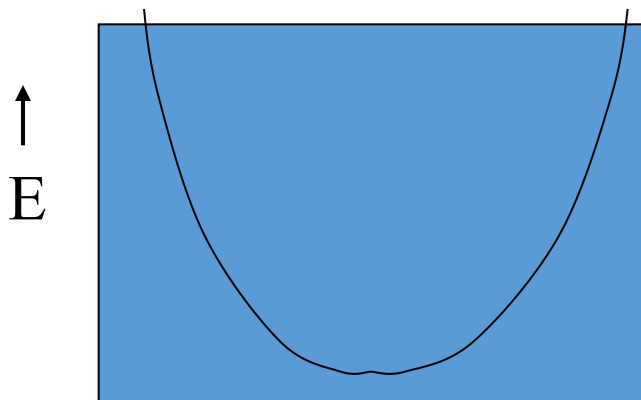
$$\begin{aligned} \rightarrow \frac{\partial E}{\partial w_i} &= \frac{1}{2} \sum_n \frac{\partial \hat{y}_n}{\partial w_i} \frac{\partial E_n}{\partial \hat{y}_n} \\ &= - \sum_n x_{i,n} (y_n - \hat{y}_n) \end{aligned}$$

$$\rightarrow \Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i}$$



The Error Surface

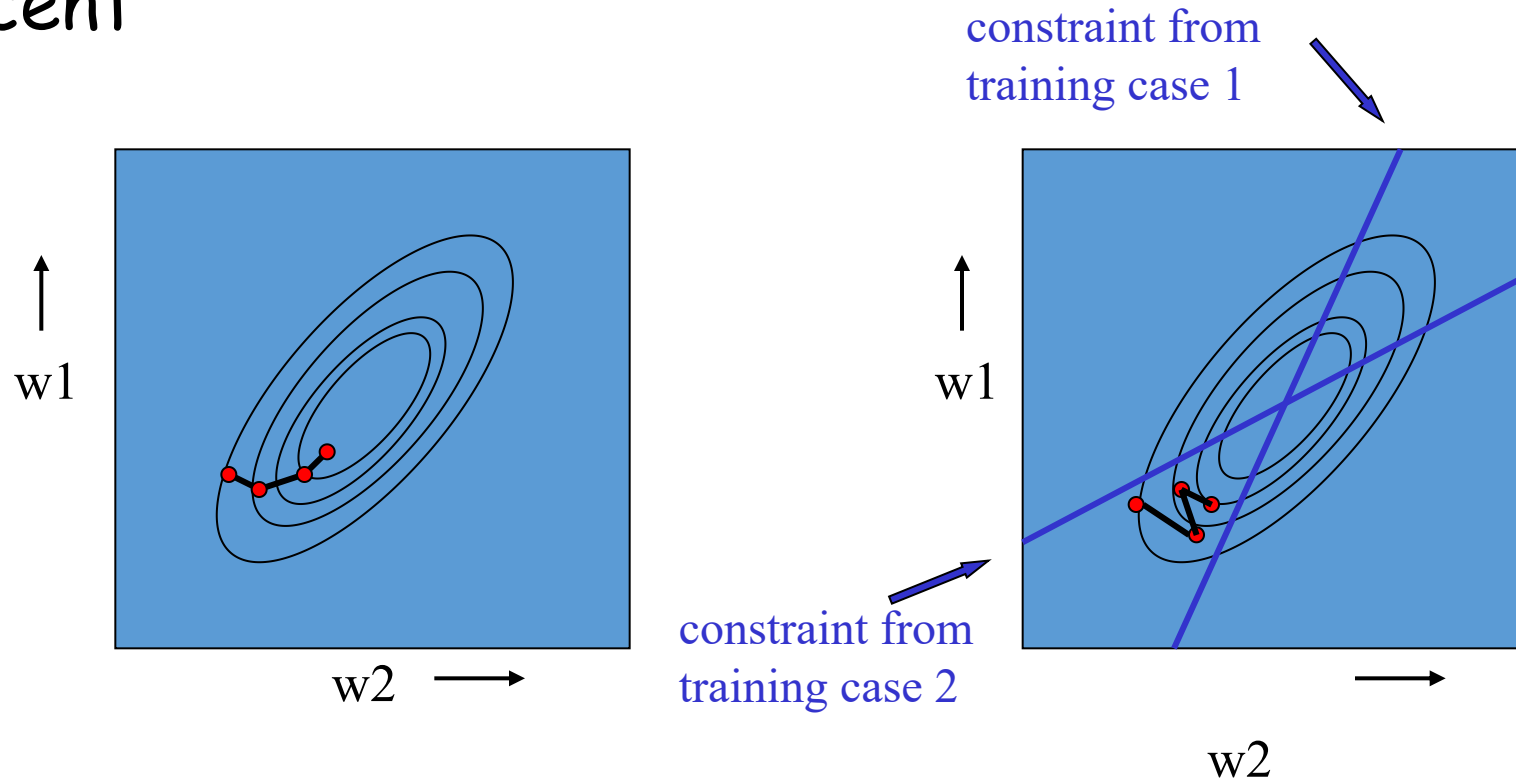
- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
 - For a linear neuron, it is a quadratic bowl.
 - Vertical cross-sections are parabolas.
 - Horizontal cross-sections are ellipses.





Online versus batch learning

- **Batch** learning does steepest descent on the error surface
- **Online** learning zig-zags around the direction of steepest descent

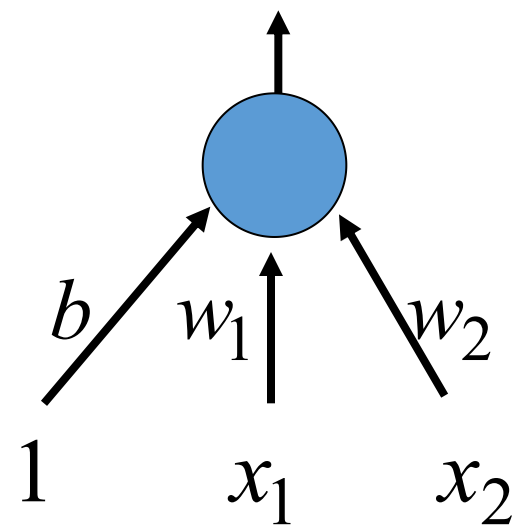




Adding biases

- A linear neuron is a more **flexible** model if we include a bias
- We can avoid having to figure out a separate learning rule for the bias by using a trick:
 - A bias is exactly equivalent to a weight on an extra input line that always has an activity of **1**

$$\hat{y} = b + \sum_i x_i w_i$$





Preprocessing the Input Vectors

- Instead of trying to predict the answer directly from the raw inputs we could start by **extracting** a layer of "features"
 - Sensible if we already know that certain combinations of input values would be useful
 - The features are equivalent to a layer of **hand-coded** non-linear neurons.
- So far as the learning algorithm is concerned, the hand-coded features are the input



Is Preprocessing Cheating?

- It seems like cheating if the aim to show how powerful learning is. The really hard bit is done by the preprocessing.
- Its not cheating if we learn the non-linear preprocessing
 - This makes learning much more difficult and much more interesting
- Its not cheating if we use a very big set of non-linear features that is task-independent
 - Support Vector Machines make it possible to use a huge number of features without much computation or data.



Statistical and ANN Terminology

人工神经网络

- A perceptron model with a linear transfer function is equivalent to a possibly multiple or multivariate linear regression model [Weisberg 1985; Myers 1986].
- A perceptron model with a logistic transfer function is a logistic regression model [Hosmer and Lemeshow 1989].
- A perceptron model with a threshold transfer function is a linear discriminant function [Hand 1981; McLachlan 1992; Weiss and Kulikowski 1991]. An ADALINE is a linear two-group discriminant.



Transfer functions

传递函数决定了神经元的输出，输出来自输入的加权总和

- Determines the output from a summation of the weighted inputs of a neuron.

$$O_j = f_j \left(\sum_i w_{ij} x_i \right)$$

- Maps any real numbers into a domain normally bounded by 0 to 1 or -1 to 1, i.e. squashing functions. Most common functions are **sigmoid** functions:

logistic: $f(x) = \frac{1}{1 + e^{-x}}$

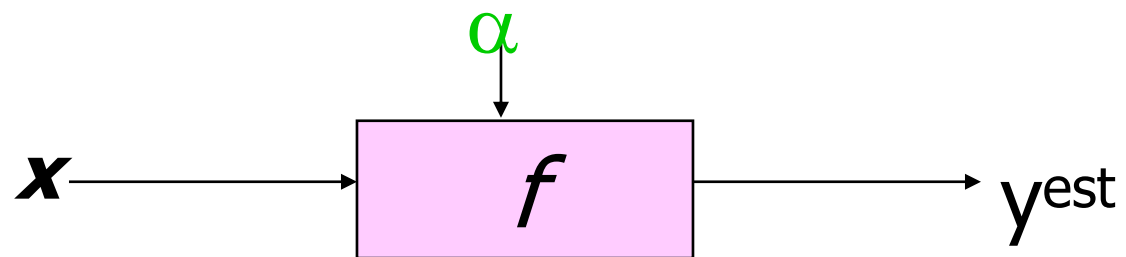
hyperbolic tangent: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Support Vector Machine



Linear Classifiers

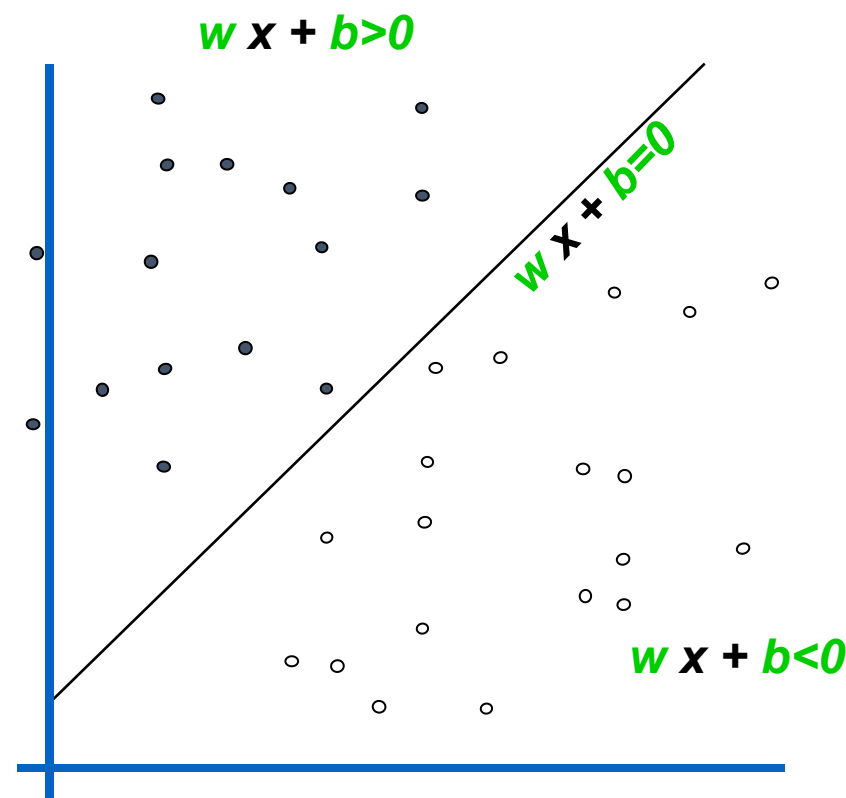
- How would you build the classifier?



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

denotes +1

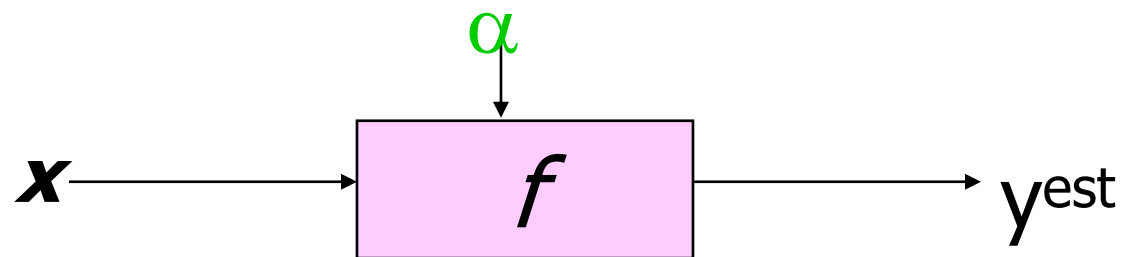
denotes -1





Linear Classifiers

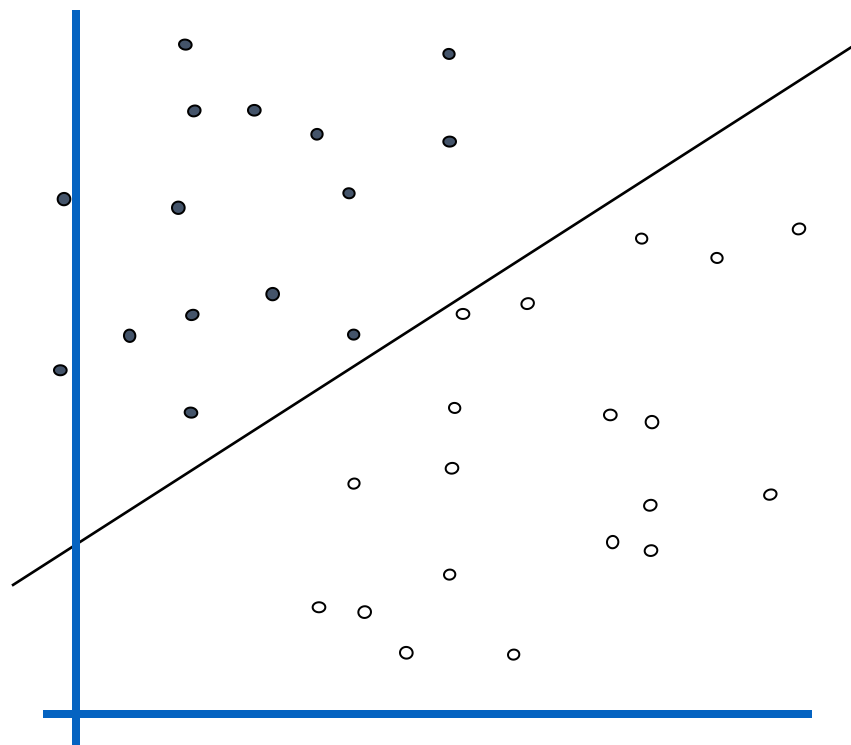
- How would you build the classifier?



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

denotes +1

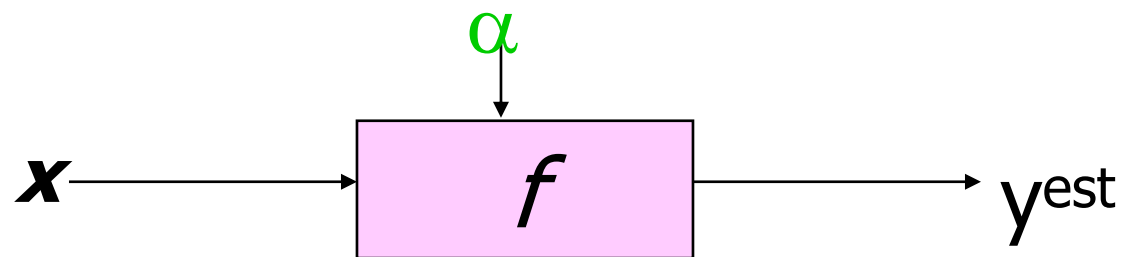
denotes -1





Linear Classifiers

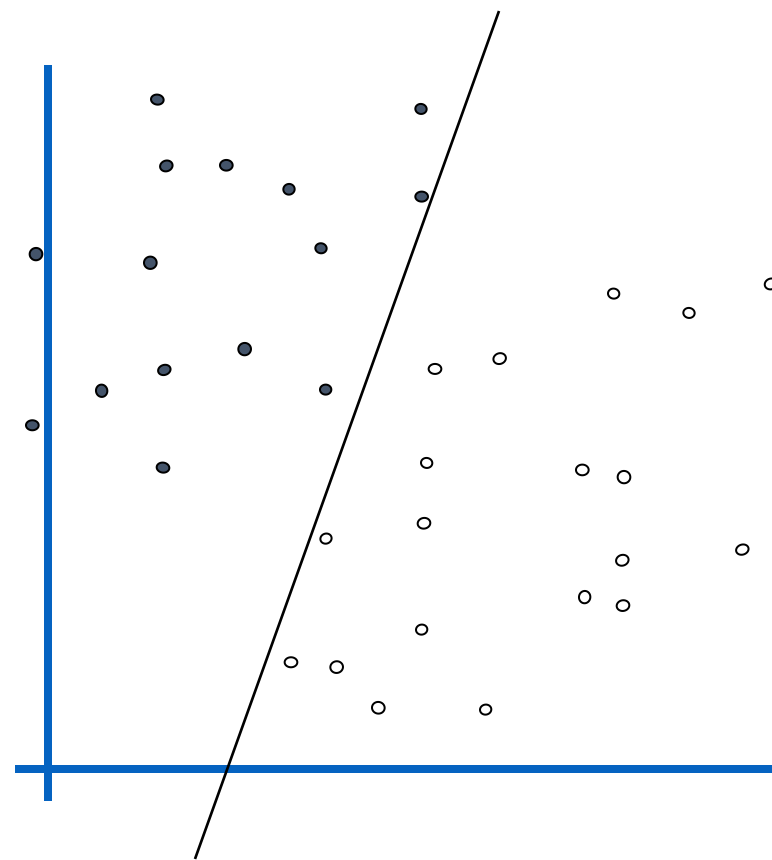
- How would you build the classifier?



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

denotes +1

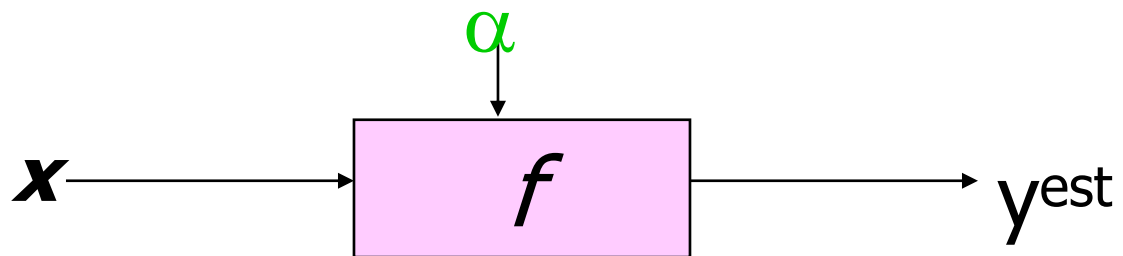
denotes -1





Linear Classifiers

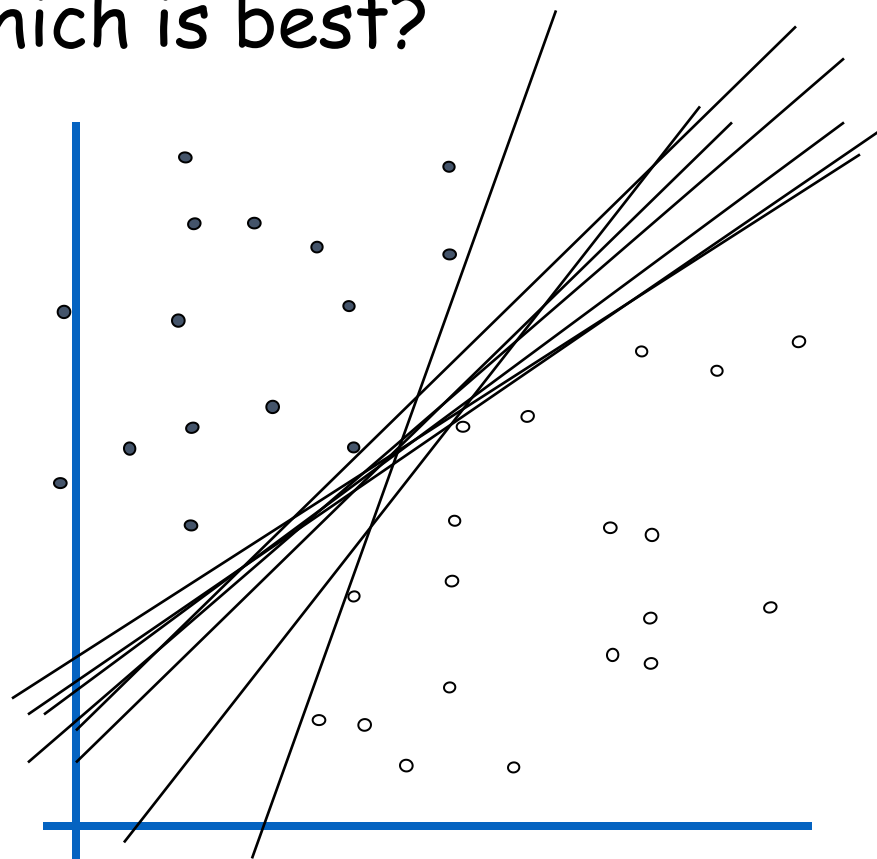
- Any of these would be fine, but which is best?



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

denotes +1

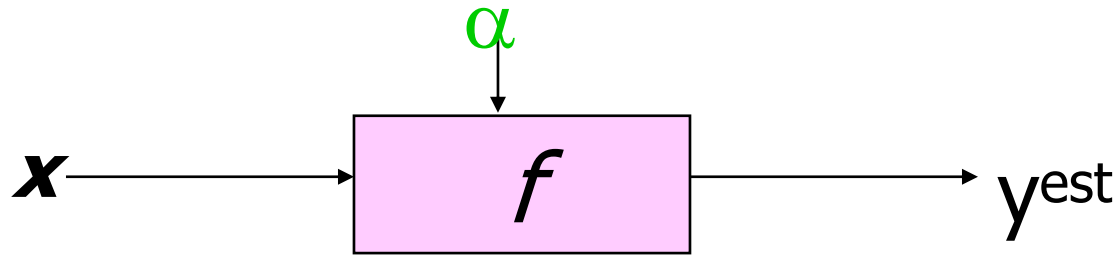
denotes -1





Linear Classifiers

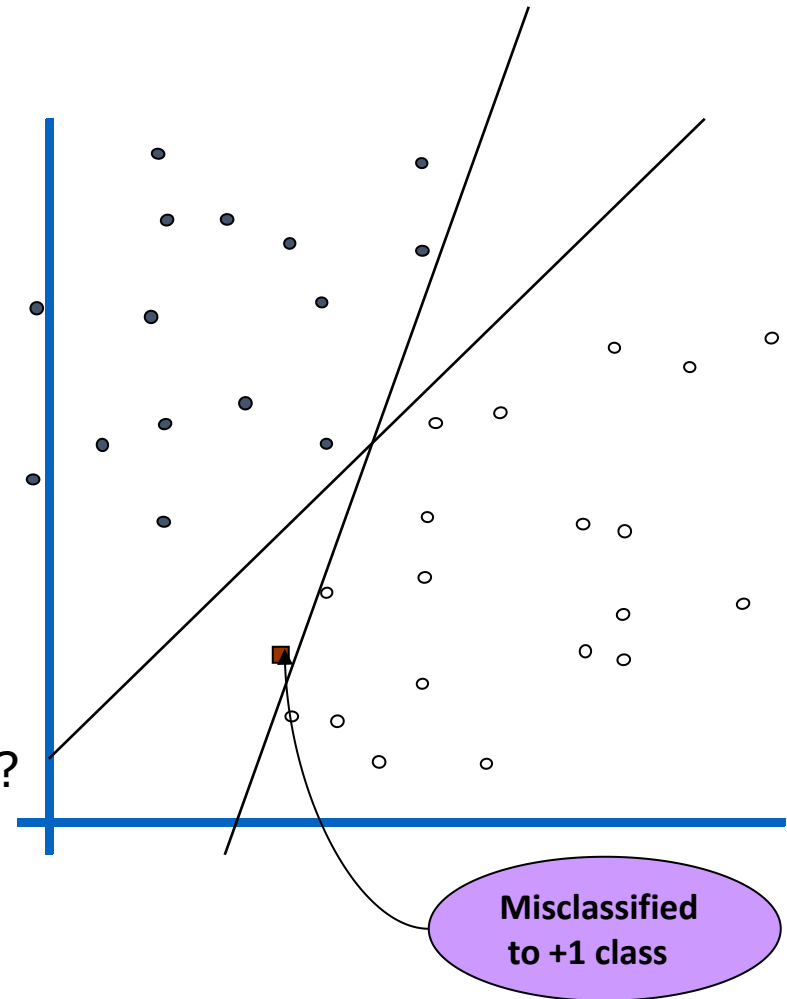
- How would you classify this data?



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

denotes +1
denotes -1

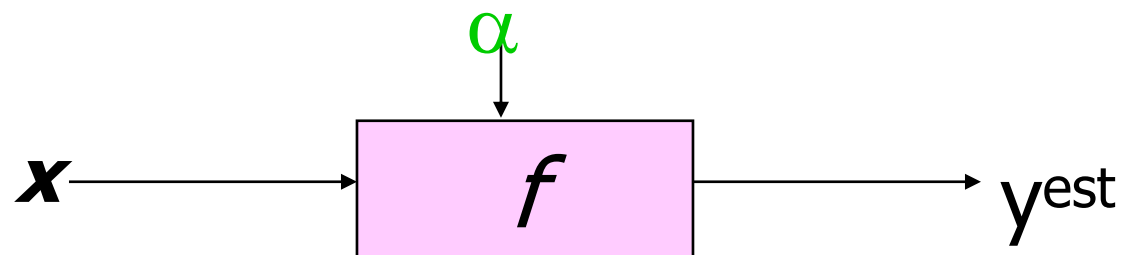
How would you classify this new data?





Classifier Margin

- How would you classify this data?

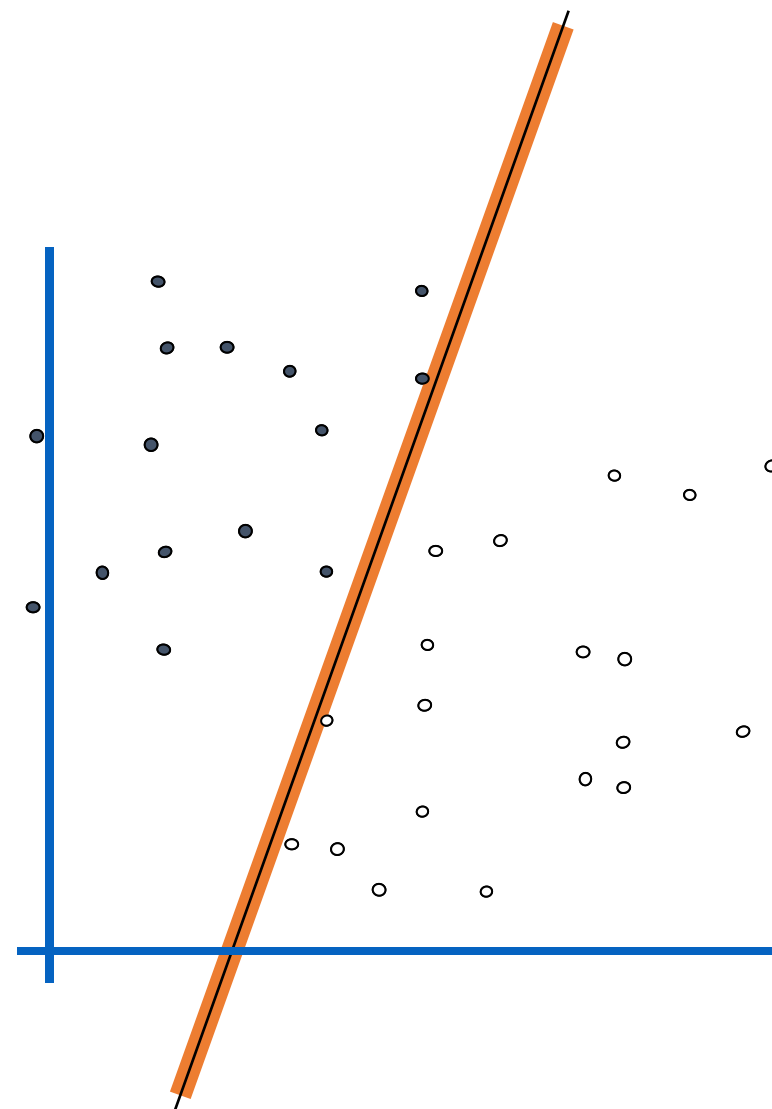


$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

denotes +1

denotes -1

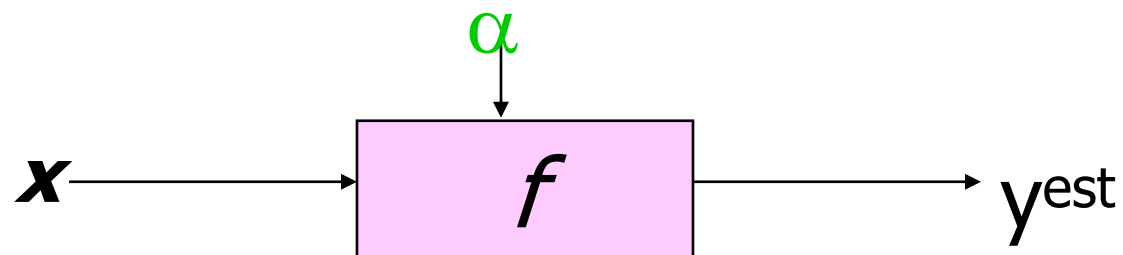
Define the **margin** of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.





Maximum Margin

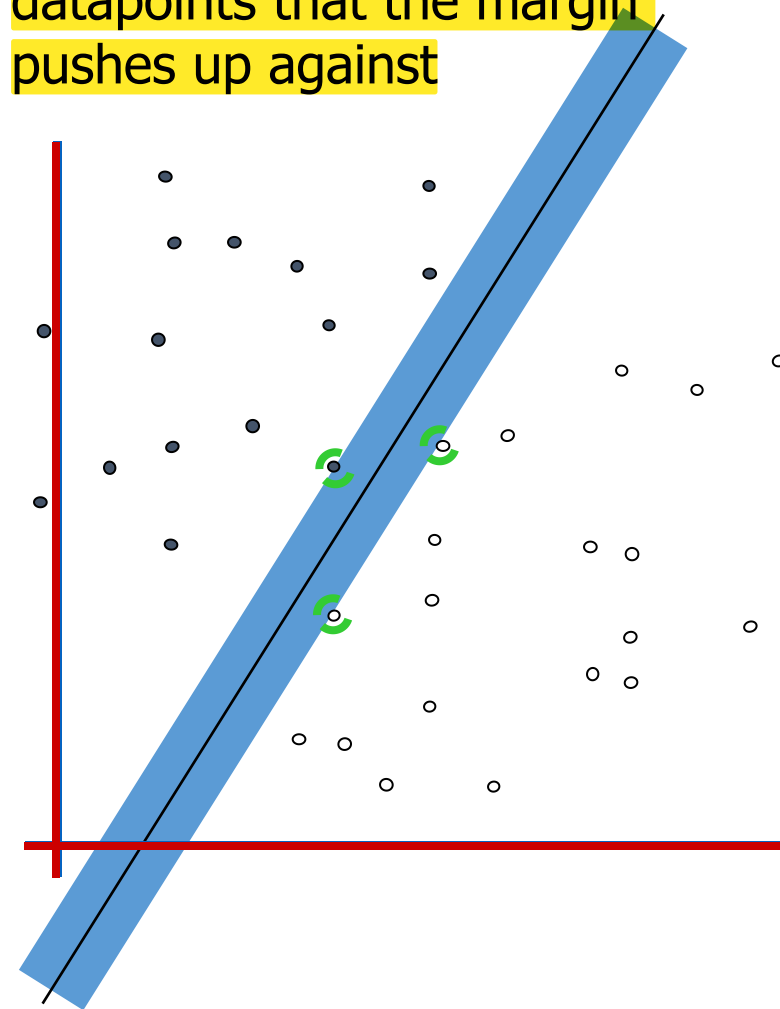
- This is the simplest kind of SVM



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

1. Maximizing the **margin** is good according to intuition and PAC theory
2. Implies that only **support vectors** are **important**; other training examples are ignorable.
3. Empirically it works **very very** well.

Support Vectors are those datapoints that the margin pushes up against



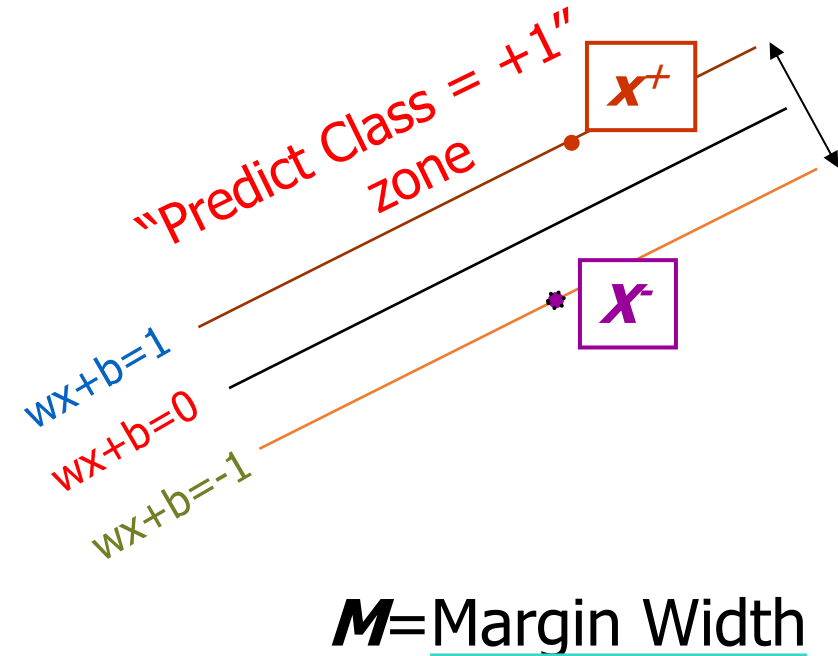


Linear SVM Mathematically

What we know:

- $w \cdot x^+ + b = +1$
- $w \cdot x^- + b = -1$
- $w \cdot (x^+ - x^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$





Linear SVM Mathematically

- Goal:

1) Correctly classify all training data

$$\begin{aligned} wX_i + b &\geq 1 && \text{if } y_i = +1 \\ wX_i + b &\leq -1 && \text{if } y_i = -1 \\ y_i(wX_i + b) &\geq 1 && \text{for all } i \end{aligned}$$

2) Maximize the margin

$$M = \frac{2}{|w|}$$

3) Same to minimize

$$\frac{1}{2} w^t w$$

- We can formulate a Quadratic Optimization Problem and solve for w and b

Minimize $\Phi(w) = \frac{1}{2} w^t w$

Subject to $y_i(wX_i + b) \geq 1 \quad \forall i$



Solving the Optimization Problem

- Solving

- Need to optimize a *quadratic* function subject to *linear* constraints
- Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them
- The solution involves constructing a dual problem where a Lagrange multiplier α_i is associated with every constraint in the primary problem:

Find \mathbf{w} and b such that
 $\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ is minimized;
and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

Find $\alpha_1 \dots \alpha_N$ such that
 $Q(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and
(1) $\sum \alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i



The Optimization Problem Solution

- The solution has the form:

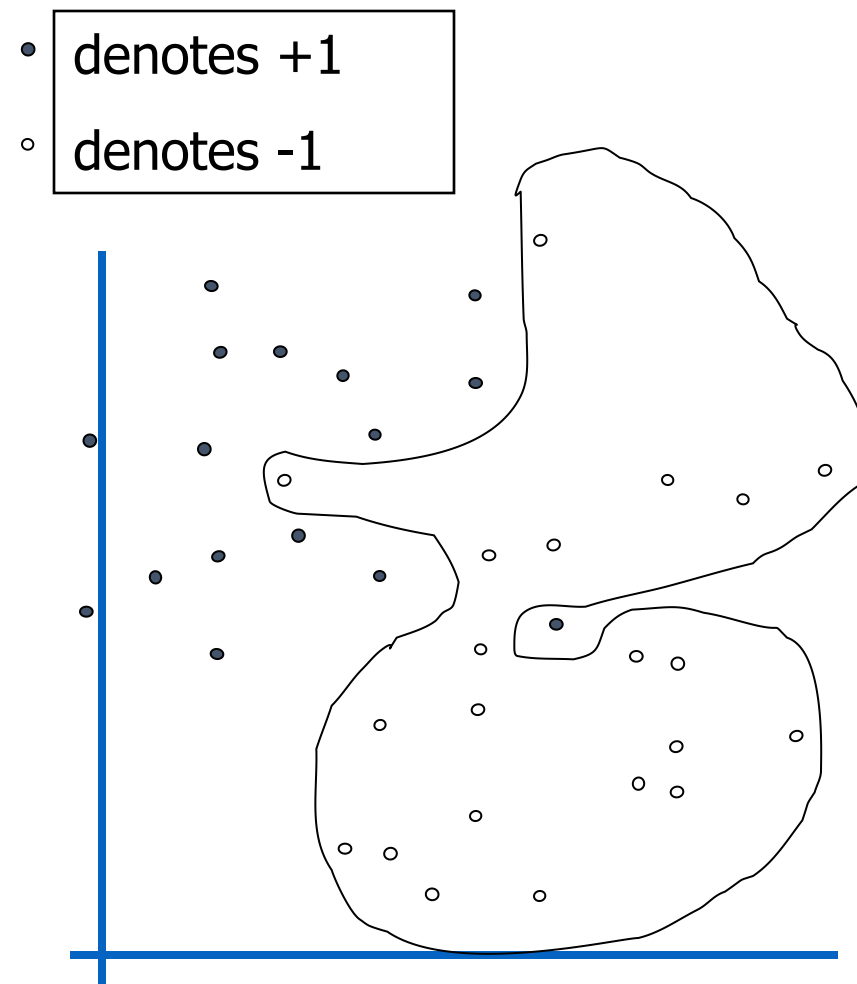
$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \quad b = y_k - \mathbf{w}^T \mathbf{x}_k \text{ for any } \mathbf{x}_k \text{ such that } \alpha_k \neq 0$$

- Each **non-zero** α_i indicates that corresponding \mathbf{x}_i is a **support vector**
- Then the classifying function will have the form:
$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$
- Notice that it relies on an **inner product** between the test point \mathbf{x} and the support vectors \mathbf{x}_i - we will return to this later
- Also keep in mind that solving the optimization problem involved computing the **inner products** $\mathbf{x}_i^T \mathbf{x}_j$ between all pairs of training points



Dataset with noise

- Hard Margin: So far we require **all** data points be classified **correctly**
 - No training error
- What if the training set is **noisy**?
 - Solution 1: use very powerful kernels





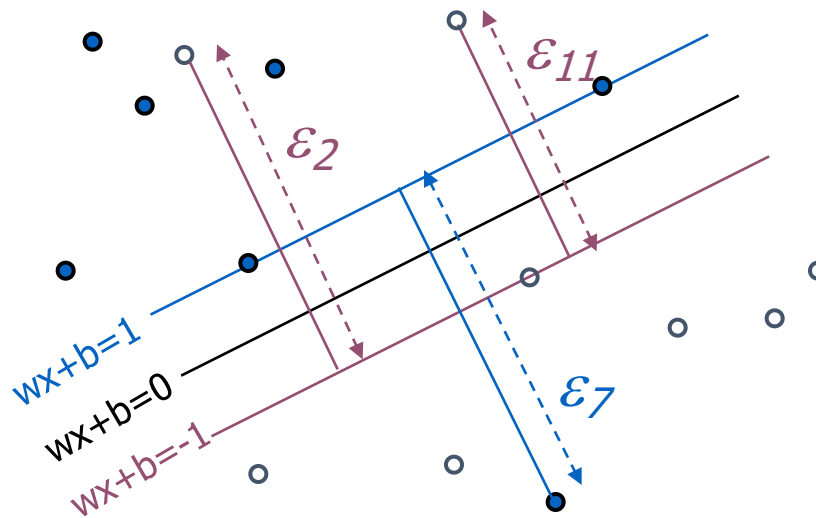
Soft Margin Classification

松弛变量

- **Slack variables** ξ_i can be added to allow misclassification of difficult or noisy examples.
- What should our quadratic optimization criterion be?

$$\frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{k=1}^R \varepsilon_k$$

加上惩罚项





Hard Margin v.s. Soft Margin

- The old formulation:

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

- The new formulation incorporating slack variables:

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum \xi_i$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \text{ for all } i$$

- Parameter C can be viewed as a way to control **overfitting**.



Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most "important" training points are support vectors; they define the hyperplane.
- QP can identify which training points \mathbf{x}_i are support vectors with non-zero Lagrangian multipliers α_i .
- Both in the dual formulation of the problem and in the solution training points appear only inside dot products:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

(1) $\sum \alpha_i y_i = 0$

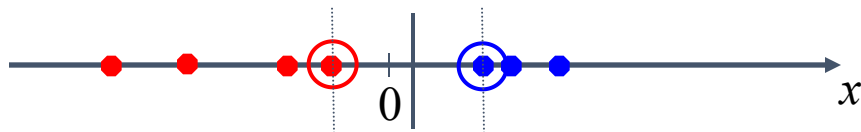
(2) $0 \leq \alpha_i \leq C$ for all α_i

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$



Non-linear SVMs

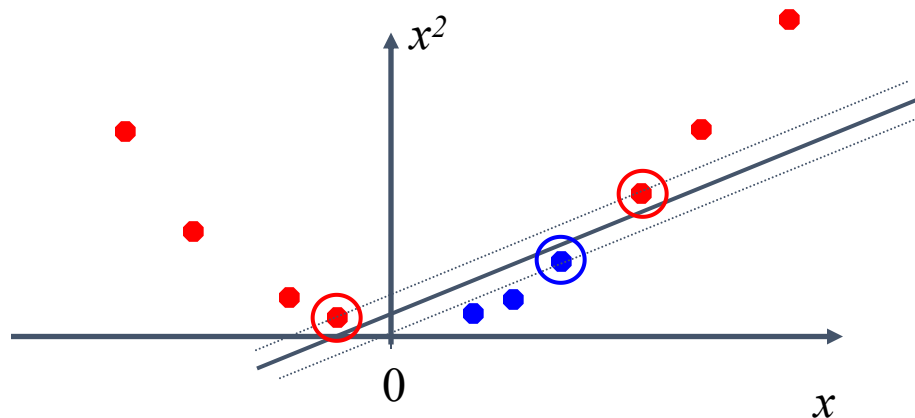
- Datasets that are **linearly separable** with some noise work out great:



- But what are we going to do if the dataset is just too hard?



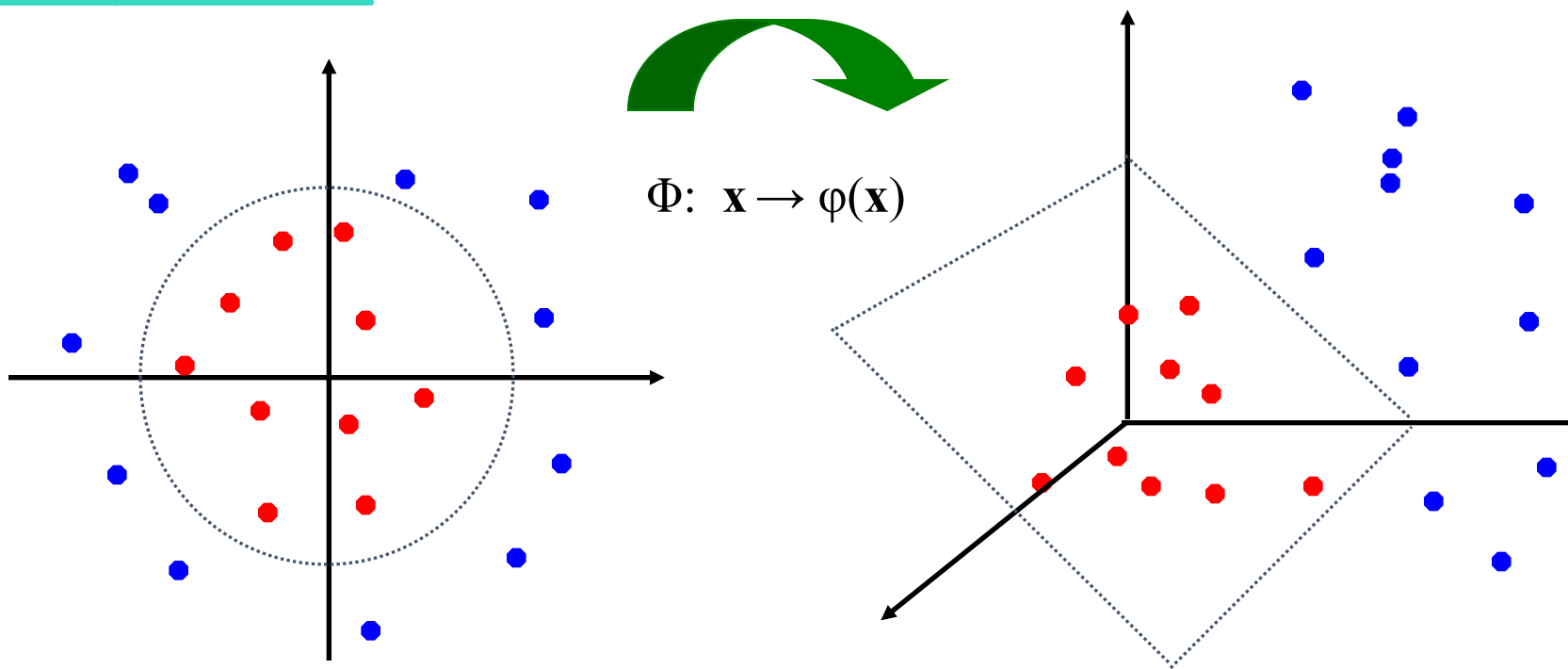
- How about **mapping data to a higher-dimensional space?**





Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:





The “Kernel Trick”

- The linear classifier relies on dot product between vectors

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

- If every data point is mapped into high-dimensional space via some transformation

$$\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x}),$$

- The dot product becomes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- A kernel function is some function that corresponds to an inner product in some expanded feature space.

核函数是一个函数，它对应于扩展特征空间中的内积，这使得在高维空间中计算点积变得更加高效而无需显式地计算映射。



The “Kernel Trick”

- Example:

- 2-dimensional vectors

$$\mathbf{x}=[x_1 \ x_2]; \text{ let } K(\mathbf{x}_i, \mathbf{x}_j)=(1 + \mathbf{x}_i^T \mathbf{x}_j)^2,$$

- Need to show that $K(\mathbf{x}_i, \mathbf{x}_j)=\varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$:

$$K(\mathbf{x}_i, \mathbf{x}_j)=(1 + \mathbf{x}_i^T \mathbf{x}_j)^2,$$

$$= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$$

$$= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}]$$

$$= \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j), \quad \text{可以写成点乘}$$

$$\text{where } \varphi(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2]$$



What Functions are Kernels?

- For some functions $K(\mathbf{x}_i, \mathbf{x}_j)$ checking that $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$ can be cumbersome.
- Mercer's theorem:
 - Every *semi-positive definite symmetric function* is a kernel
- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

表示数据点之间的相似度 $K =$

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_3)$...	$K(\mathbf{x}_1, \mathbf{x}_N)$
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_3)$		$K(\mathbf{x}_2, \mathbf{x}_N)$
	
$K(\mathbf{x}_N, \mathbf{x}_1)$	$K(\mathbf{x}_N, \mathbf{x}_2)$	$K(\mathbf{x}_N, \mathbf{x}_3)$...	$K(\mathbf{x}_N, \mathbf{x}_N)$

在支持向量机中，核函数就是用来计算样本之间相似度的，这对于高维数据特别有用。



Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Polynomial of power p : $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
- Gaussian (radial-basis function network):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$



Non-linear SVMs Mathematically

- Dual problem formulation:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is maximized and

(1) $\sum \alpha_i y_i = 0$

代替原来的点乘

(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- Optimization techniques for finding α_i 's remain the same!



Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the **feature space** and classify points in that space
- It does **not** need to represent the space **explicitly**, simply by defining a **kernel** function
- The **kernel** function plays the role of the **dot product** in the feature space.



Properties of SVM

SVM允许根据具体问题选择不同的相似度量函数，以帮助分类任务

- Flexibility in choosing a **similarity** function
- **Sparseness** of solution when dealing with large data sets
 - Only support vectors are used to specify the separating hyperplane
- Ability to handle **large** feature spaces
 - Complexity does not depend on the dimensionality of the feature space
- **Overfitting** can be **controlled** by **soft margin approach**
- **Nice math** property: a simple **convex** optimization problem which is guaranteed to converge to a single **global** solution
- Feature selection



Weakness of SVM

- It is sensitive to noise
 - A relatively **small number** of mislabeled examples can dramatically decrease the performance
- It only considers **two** classes
 - How to do multi-class classification with SVM?
 - Answer:

1) With **m** output, learn **m** SVM's

- ✓ SVM 1 learns "Output==1" vs "Output != 1"
- ✓ SVM 2 learns "Output==2" vs "Output != 2"
- ✓ SVM m learns "Output==m" vs "Output != m"

2) To predict the output for a **new** input, just predict with each SVM and find out which one puts the prediction the **furthest into the positive region**

对于每个SVM分类器，都会计算一个分数或距离，表示该样本距离该分类器的分类边界有多远。SVM的输出是一个符号（+1或-1），但实际上它也可以输出一个距离值（即样本到分类超平面的距离）。你需要找出哪个SVM的距离值最远（也就是分类器的决策函数输出值最大），这个分类器的决策将是最终的预测结果。

选择其中一个最远离决策边界的



Some Issues

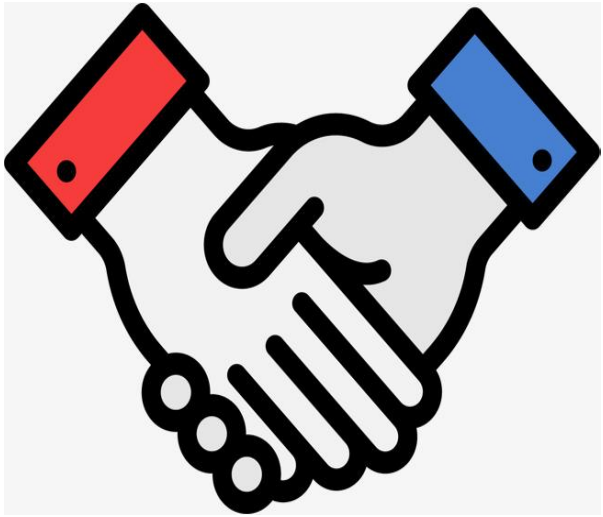
- Choice of **kernel**
 - Gaussian or polynomial kernel is default
 - If ineffective, more elaborate kernels are needed
 - Domain experts can give assistance in formulating appropriate similarity measures
- Choice of kernel **parameters**
 - e.g. σ in Gaussian kernel
 - σ is the distance between closest points with different classifications
 - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.
- Optimization **criterion** - Hard margin v.s. Soft margin
 - A lengthy series of experiments in which various parameters are tested

Conclusions



Conclusion

- The perception
 - Online learning
 - One layer
 - Multiple classifiers
- Support vector machine
 - Maximum margin
 - Support vector
 - Kernel trick



Thanks



zhengf@sustc.edu.cn