Machine Learning Algorithms Summary

肖庭富
tingfuxiao@hotmail.com
15001021061

2017.05.27

Contents

Classical Algorithms based on statistics

- Generate Learning Algorithm

Step-1: model $p\{x|y\}, p(y)$

$Training\ set(X_i, Y_i), learning\ join\ probability\ distribution\ p(x,y)$

$$p\{y|x\} = \frac{p(x,y)}{p(x)}$$

$$p(x,y) = p\{x|y\} \times p(y)$$

$$p\{y|x\} = \frac{p\{x|y\} \times p(y)}{p(x)}$$

Step-2: learn parameters of the models by maximizing joint likelihood

For new data x , predict which class x belongs to. It belongs to the class which make $p\{y|x\}\ max$

$$y = \operatorname*{argmax}_i(p\{y=i|x\}) = \operatorname*{argmax}_i\left(\frac{p\{x|y=i\} \times p\{y=i\}}{p(x)}\right) = \operatorname*{argmax}_i(p\{x|y=i\} \times p\{y=i\})$$

Step-3: predict which class the new data will belong to

Classical Algorithms based on statistics

- Gaussian Discriminant Analysis – GDA Model

Assumption:

$$y \sim Bernoulli(\varphi)$$
$$x|y = 0 \sim N(u_0, \sum)$$
$$x|y = 1 \sim N(u_1, \sum)$$

$$p(y) = \varphi^y (1-\varphi)^{1-y}$$
$$p\{x \mid y=0\} = \frac{1}{(2\pi)^{m/2} |\Sigma|^{1/2}} \exp\left(\frac{1}{2} (x-u_0)^T \Sigma^{-1} (x-u_0) \right)$$
$$p\{x \mid y=1\} = \frac{1}{(2\pi)^{m/2} |\Sigma|^{1/2}} \exp\left(\frac{1}{2} (x-u_1)^T \Sigma^{-1} (x-u_1) \right)$$

Joint likelihood: $$l(\varphi, u_0, u_1, \Sigma) = \prod_i^n p(x_i, y_i) = \prod_i^n p\{x_i \mid y_i\} p(y_i)$$

$$\varphi = \frac{\sum_i^n 1\{y_i = 1\}}{n}$$

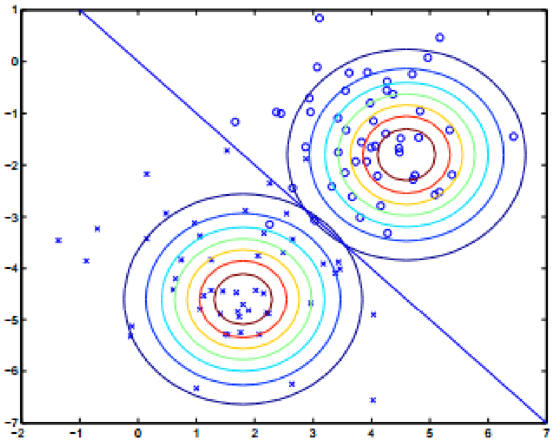
$$u_0 = \frac{\sum_i^n 1\{y_i = 0\} x_i}{\sum_i^n 1\{y_i = 0\}}$$

$$\Sigma = \frac{1}{n} \sum_i^n (x_i - u_{y_i})(x_i - u_{y_i})^T$$

$$u_1 = \frac{\sum_i^n 1\{y_i = 1\} x_i}{\sum_i^n 1\{y_i = 1\}}$$

Classical Algorithms based on statistics

- Gaussian Discriminant Analysis – GDA Model



The figure are training set. (The contours of the 2 Gaussian distribution)

2 Gaussian contours are same shape and orientation due to same Σ

3. The straight line shown in the figure is the decision boundary at which

$$p\{y=1|x\} = p\{y=0|x\} = 0.5$$

4. On one side of the boundary, predict y=1, and on another side, predict y=0

Classical Algorithms based on statistics

- Naive Bayes Classifier

Training set:

$$\begin{cases} (X_i, Y_i) \quad \text{Every training point, contains } m \text{ features: } X_i: [x_1, x_2, \ldots, x_m] \\ 0 \le i < n \\ Lable\ variable:\ Y_i \in \{1,2,3, \ldots, k\} \end{cases}$$

Target: Given a new data X with feature X: $[x_1, x_2, \ldots, x_m]$, computer $p\{Y = i | X(x_1, x_2, \ldots, x_m)\}$

Finally to get: $y = arg\max_i (p\{y = i | x_1, x_2, \ldots, x_m\})$

Bayes Formula:

$$p\{Y = i | X(x_1, x_2, \ldots, x_m)\} = \frac{p\{X(x_1, x_2, \ldots, x_m) | Y = i\} \times p\{Y = i\}}{p\{X(x_1, x_2, \ldots, x_m)\}}$$

$$= \frac{p\{x, x, \ldots, x | y = i\} * p\{y = i\}}{p\{x_1, x_2, \ldots, x_m\}}$$

Classical Algorithms based on statistics

- Naive Bayes Classifier

Assumption: independence between all features

$$p\{x_1, x_2, \ldots, x_m | y = i\} = \prod_k^m p\{x_k | y = i\}$$

Estimate $p\{y = i\}$ through the frequency of $y = i$ in training set

$$p\{y = i\} = \frac{\sum_k^n 1\{y_k = i\}}{m} \qquad \frac{The\ number\ of\ Label\ Y_k = i\ in\ training\ set}{Total\ number\ of\ Training\ set}$$

For a specific new data $X(x_1, x_2, \ldots, x_m)$ $p\{x_1, x_2, \ldots, x_m\}$ will be a constant for all $Y = i$
Since it can be computed through Total Probability Formula, so it can be ignored when deciding which type it is.

$$p\{Y = i | X(x_1, x_2, \ldots, x_m)\}$$

$$p\{Y = i | X(x_1, x_2, \ldots, x_m)\} \infty \frac{\sum_k^n 1\{y_k = i\}}{n} \prod_k^m p\{x_k | y = i\}$$

Different naive Bayes classifier differ mainly by the assumptions of the distribution of $p\{x_k | y = i\}$

Classical Algorithms based on statistics

- Naive Bayes Classifier

1. $compute\ p\{x_k|y=i\}\ through\ frequency\ count$ $p\{x_k \mid y=i\} = \dfrac{\sum_j^n 1\{x_{jk}=x_k, y_j=i\}}{\sum_j^n 1\{y_j=i\}}$ Number of x=xk, y=i / Number of y=i

2. $p\{x_k|y=i\} \sim N(u_i, \sigma_i)$ $u_i = \dfrac{\sum_j^n 1\{y=i\} \times x_{jk}}{\sum_j^n 1\{x_{jk}=x_k, y=i\}}$ $\sigma_i^2 = \dfrac{\sum_j^n (x_{jk}-u_i)^2}{\sum_j^n 1\{y_j=i\}}$

3. $p\{x_k|y=i\} \sim Bernoulli(x_k, p\{y=i\})\ \ if\ k=2$

Laplace Smooth: $p\{y=i\}$ $p\{x_k|y=i\}$ maybe is zero(0), so will add a constant to Numerator and Denominator

$$p\{y=i\} = \frac{\sum_k^n 1\{y_k=i\}+1}{m+k}$$

$$p\{x_k \mid y=i\} = \frac{\sum_j^n 1\{x_{jk}=x_k, y_j=i\}+1}{\sum_j^n 1\{y_j=i\}+m}$$

Linear Regression and Ridge Regression

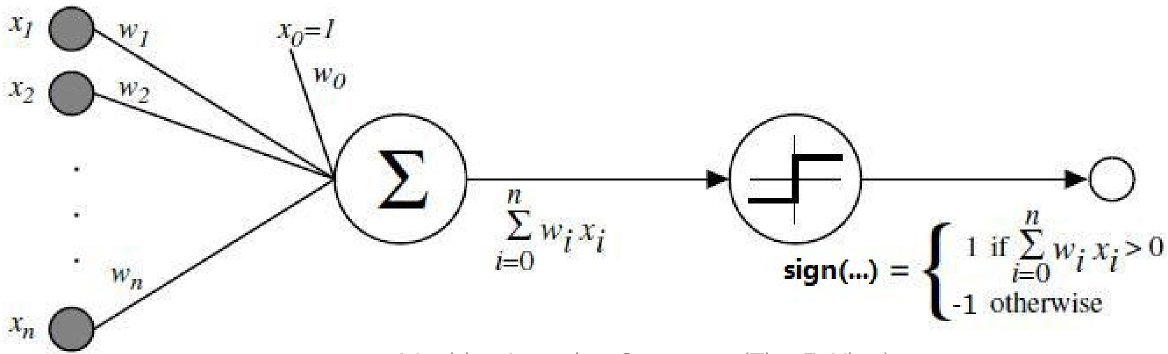
Training set:
- (X_i, d_i) Every training point, contains m features: $X_i: [x_1, x_2, \ldots, x_m]$
- $1 \le i \le n$
- $Lable\ variable$: $d_i \in \{-1,1\}$

$$regression: y_i = w_i^T x_i + b = \sum_{j=1}^{n} w_j(i) x_j(i) + b$$

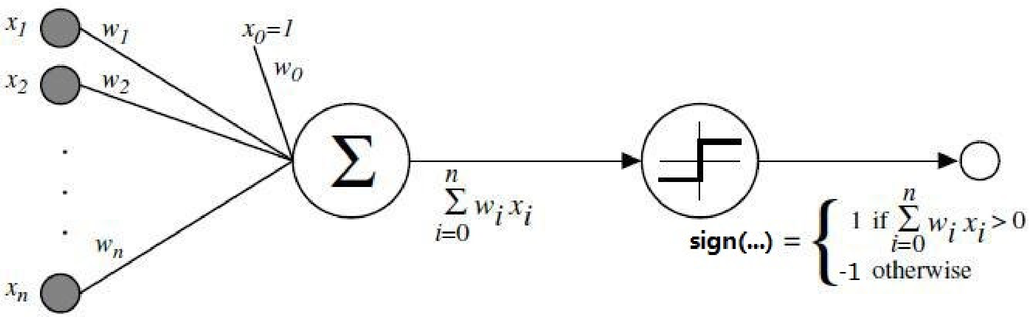
Weight vector: $w_1(i), w_2(i), \ldots, w_m(i)$

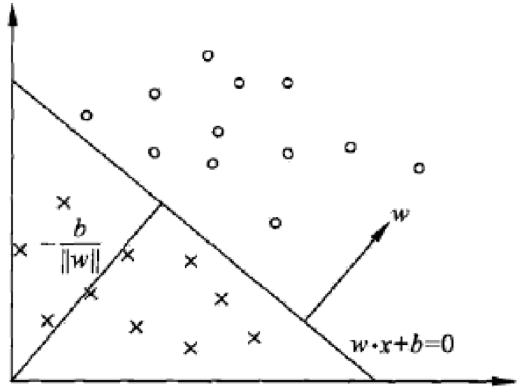
$$classification: y_i = sign(w_i^T x_i + b) = sign(\sum_{j=0}^{n} w_j(i) x_j(i) + b)$$

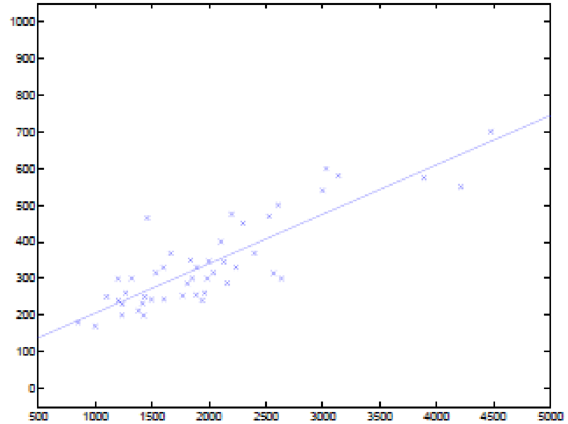
$$sign(x) = \begin{cases} 1\ if\ \ x \ge 0 \\ -1\ if\ \ x < 0 \end{cases}$$



Linear Regression and Ridge Regression







$$Error\ signal: e(i) = d(i) - y(i) \qquad Error\ sum\ of\ squares: \xi = \frac{1}{2}\sum_{i}^{n} e^2(i) = \frac{1}{2}\sum_{i}^{n}(d(i) - y(i))^2$$

Target: Try to find a group weight vector to **minimize** the error sum of square – LMS algorithm for regression or **minimize** the error of classification

Actually, to find a super plain $WX + b = 0$ to separate the data

Linear Regression and Ridge Regression

- Minimize the error of classification

$error\ classification:\ d_i(wx_i + b) < 0$

$Lossfunction: L(w,b) = -\sum_i d_i(wx_i + b)$

Target: Try to find a group weight vector to **minimize** the error sum of square

or **minimize** the error of classification

Actually, to find a super plain $WX + b = 0$ to separate the data

Minimize: $L(w,b) = -\sum d_i(wx_i + b)$

Stochastic gradient descent:

$$\nabla_w L(w,b) = \frac{\partial L(w,b)}{\partial w} = -\sum_i d_i x_i$$

$$\nabla_b L(w,b) = \frac{\partial L(w,b)}{\partial b} = -\sum_i d_i$$

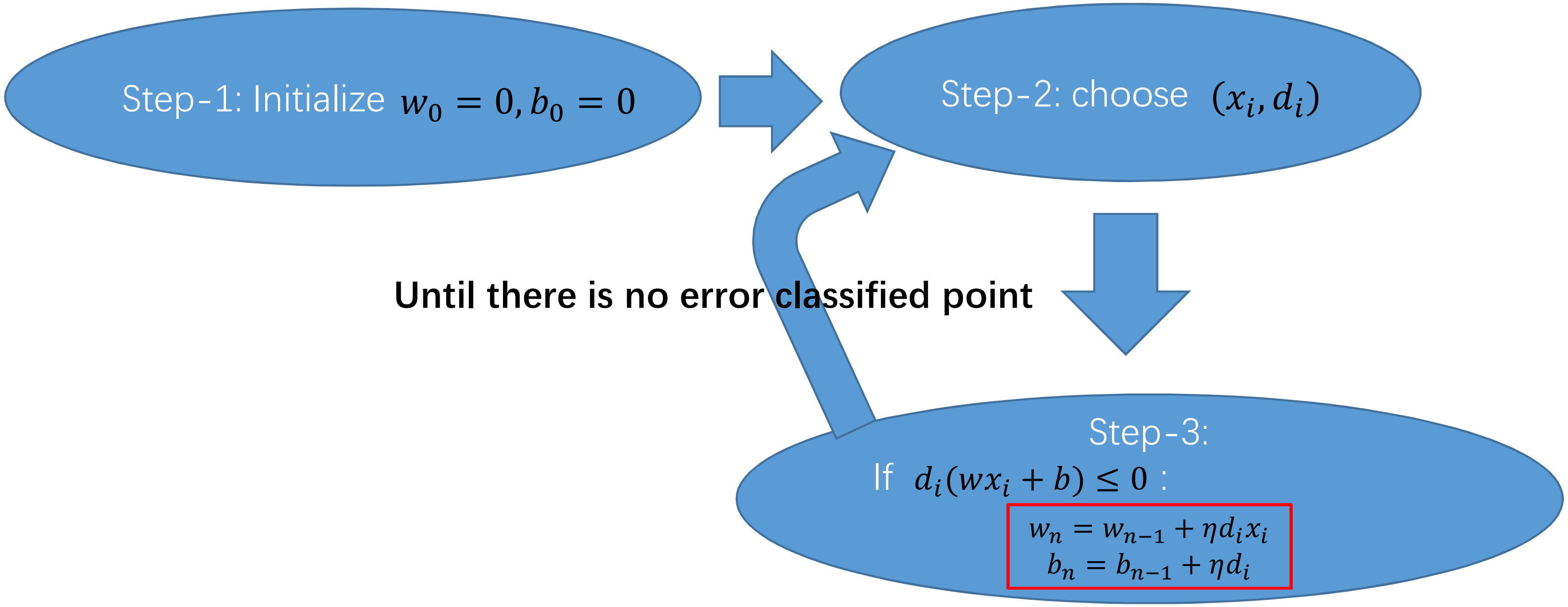
Weight updating:

$$w_n = w_{n-1} + \eta d_i x_i$$

$$b_n = b_{n-1} + \eta d_i$$

Linear Regression and Ridge Regeression

- Classification: Learning algorithm



Linear Regression and Ridge Regression

- Regression: LMS algorithm

Batch gradient descent:

Error sum of squares (Loss function) : $l(w,b) = \xi = \frac{1}{2}\sum_i^n e^2(i) = \frac{1}{2}\sum_i^n (d(i)-y(i))^2 = \frac{1}{2}\sum_i^n (d(i)-y(i))^2$

$$\nabla_w l(w,b) = \frac{\partial l(w,b)}{\partial w} = -\sum_i^n (d_i - y_i)\frac{\partial y_i}{\partial w_{ij}} = \sum_i^n (y_i - d_i)x_{ij}$$

$$\nabla_b l(w,b) = \frac{\partial l(w,b)}{\partial b} = -\sum_i^n (d_i - y_i)\frac{\partial y_i}{\partial b} = \sum_i^n (y_i - d_i)$$

$$w_{ij} = w_{ij} - \eta \nabla_w l(w,b) = w_{ij} + \sum_i^n (d_i - y_i)x_{ij}$$

$$b = b - \eta \nabla_b l(w,b) = b + \sum_i^n (d_i - y_i)$$

$l(w,b) = \xi = e^2(i) = (d(i)-y(i))^2$

Stochastic gradient descent:

$$\nabla_w l(w,b) = \frac{\partial l(w,b)}{\partial w} = -(d_i - y_i)\frac{\partial y_i}{\partial w_{ij}} = (y_i - d_i)x_{ij}$$

$$\nabla_b l(w,b) = \frac{\partial l(w,b)}{\partial b} = -(d_i - y_i)\frac{\partial y_i}{\partial b} = (y_i - d_i)$$

$$w_{ij} = w_{ij} - \eta \nabla_w l(w,b) = w_{ij} + (d_i - y_i)x_{ij}$$

$$b = b - \eta \nabla_b l(w,b) = b + (d_i - y_i)$$

Linear Regression and Ridge Regression

- Matrix representation for LMS algorithm

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \vec{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \qquad XW - \vec{d} = \begin{bmatrix} x_1 w \\ x_2 w \\ \vdots \\ x_n w \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} x_1 w - d_1 \\ x_2 w - d_2 \\ \vdots \\ x_n w - d_n \end{bmatrix}$$

Loss function: $l(W) = \frac{1}{2}\sum_{i=0}^{n}(y_i - d_i)^2 = \frac{1}{2}(XW - \vec{d})^T(XW - \vec{d})$

$$= \frac{1}{2}(W^T X^T XW - W^T X^T \vec{d} - \vec{d}^T XW + \vec{d}^T \vec{d})$$

$$\nabla_W l(W) = \frac{1}{2}\nabla_W (W^T X^T XW - W^T X^T \vec{d} - \vec{d}^T XW + \vec{d}^T \vec{d})$$

$$= \frac{1}{2}\nabla_W Trace(W^T X^T XW - W^T X^T \vec{d} - \vec{d}^T XW + \vec{d}^T \vec{d})$$

$$= \frac{1}{2}\nabla_W Trace(W^T X^T XW - 2\vec{d}^T XW)$$

$$= \frac{1}{2}(X^T XW + X^T XW - 2X^T \vec{d})$$

$$= X^T XW - X^T \vec{d}$$

$$\begin{aligned} \nabla_A \mathrm{tr} AB &= B^T \\ \nabla_{A^T} f(A) &= (\nabla_A f(A))^T \\ \nabla_A \mathrm{tr} ABA^T C &= CAB + C^T AB^T \\ \nabla_{A^T} \mathrm{tr} ABA^T C &= B^T A^T C^T + BA^T C \end{aligned}$$

$$\boxed{\begin{aligned} &\nabla_W l(W) = 0 \\ &\Rightarrow X^T XW - X^T \vec{d} = 0 \\ &\Rightarrow W = (X^T X)^{-1} X^T \vec{d} \end{aligned}}$$

Linear Regression and Ridge Regression

- Ridge Regression

Regularization

Loss function: $l(W)=\frac{1}{2}\sum_{i=0}^{n}(y_i-d_i)^2+\frac{\lambda}{2}\|W\|^2=\frac{1}{2}(XW-\vec{d})^T(XW-\vec{d})+\frac{\lambda}{2}W^TW$

$$=\frac{1}{2}\left(W^TX^TXW-W^TX^T\vec{d}-\vec{d}^TXW+\vec{d}^T\vec{d}+\lambda W^TW\right)$$

$$\nabla_W l(W)=\frac{1}{2}\nabla_W\left(W^TX^TXW-W^TX^T\vec{d}-\vec{d}^TXW+\vec{d}^T\vec{d}+\lambda W^TW\right)$$

$$=\frac{1}{2}\nabla_W Trace\left(W^TX^TXW-W^TX^T\vec{d}-\vec{d}^TXW+\vec{d}^T\vec{d}+\lambda W^TW\right)$$

$$=\frac{1}{2}\nabla_W Trace\left(W^TX^TXW-2\vec{d}^TXW+\lambda W^TW\right)$$

$$=\frac{1}{2}\left(X^TXW+X^TXW-2X^T\vec{d}+2\lambda W\right)$$

$$=X^TXW-X^T\vec{d}+\lambda W$$

$$\boxed{\begin{aligned}&\nabla_W l(W)=0\\&\Rightarrow X^TXW-X^T\vec{d}+\lambda W=0\\&\Rightarrow W=(X^TX+\lambda E)^{-1}X^T\vec{d}\end{aligned}}$$

Logistic Regression

- Logistic function(Sigmod function)

$$F(x)=\frac{1}{1+e^{-\frac{x-u}{\gamma}}}=\frac{1}{1+e^{-x}}$$

$$f(x)=\frac{e^{-x}}{\left(1+e^{-x}\right)^2}=\frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}})=F(x)(1-F(x))$$

Training set: (X_i, d_i), $d_i = 0,1; 1 \le i \le n$

$$\varphi(x)=w^T x+b$$

$$y=F(\varphi(x))=F(w^T x+b)=\frac{1}{1+e^{-(w^T x+b)}}$$

Assumption:

$$p\{y=1\,|\,x\}=F(\varphi(x))$$

$$p\{y=0\,|\,x\}=1-F(\varphi(x))$$

$$\boxed{p\{y\,|\,x;w,b\}=\left(F\left(\varphi(x)\right)\right)^y\left(1-F\left(\varphi(x)\right)\right)^{1-y}}$$

Logistic Regression

- Max likelihood

$$L(w,b)=\prod_{i=0}^{n} p\{y_i \mid x_i; w,b\} = \prod_{i=0}^{n}\left(F\left(\varphi\left(x_i\right)\right)\right)^{y_i}\left(1-F\left(\varphi\left(x_i\right)\right)\right)^{1-y_i}$$

$$\ell(w,b)=\log L(w,b)=\sum_{i=0}^{n} y_i \log F\left(\varphi\left(x_i\right)\right)+(1-y_i)\log\left(1-F\left(\varphi\left(x_i\right)\right)\right)$$

$$\nabla_w \ell(w,b)=\frac{\partial \ell(w,b)}{\partial w_j}$$

$$f(x)=F(x)(1-F(x))$$

$$=\left(\frac{y_i}{F(\varphi(x_i))}-\frac{1-y_i}{1-F(\varphi(x_i))}\right)\frac{\partial F(\varphi(x_i))}{\partial \varphi(x_i)}\frac{\partial \varphi(x_i)}{\partial w_j}$$

$$=\left(\frac{y_i}{F(\varphi(x_i))}-\frac{1-y_i}{1-F(\varphi(x_i))}\right)F(\varphi(x_i))\left(1-F(\varphi(x_i))\right)\frac{\partial(wx_i+b)}{\partial w_j}$$

$$=\left(y_i\left(1-F(\varphi(x_i))\right)-F(\varphi(x_i))\left(1-y_i\right)\right)x_{ij}$$

$$=\left(y_i-F(\varphi(x_i))\right)x_{ij}$$

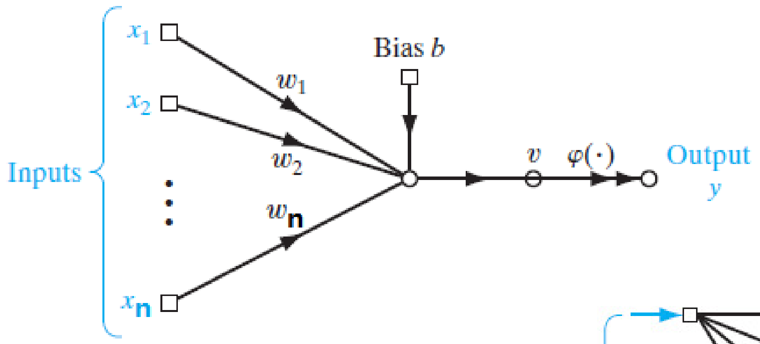
$$\nabla_b \ell(w,b)=y_i-F(\varphi(x_i))$$

Stochastic gradient ascent:

$$\boxed{\begin{aligned} w_n &= w_{n-1}+\eta\nabla_w \ell(w,b)=w_{n-1}+\eta\left(y_i-F(\varphi(x_i))\right)x_{ij} \\ b_n &= b_{n-1}+\eta\nabla_b \ell(w,b)=b_{n-1}+\eta\left(y_i-F(\varphi(x_i))\right) \end{aligned}}$$

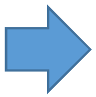
Multilayer perceptron

- Network structure



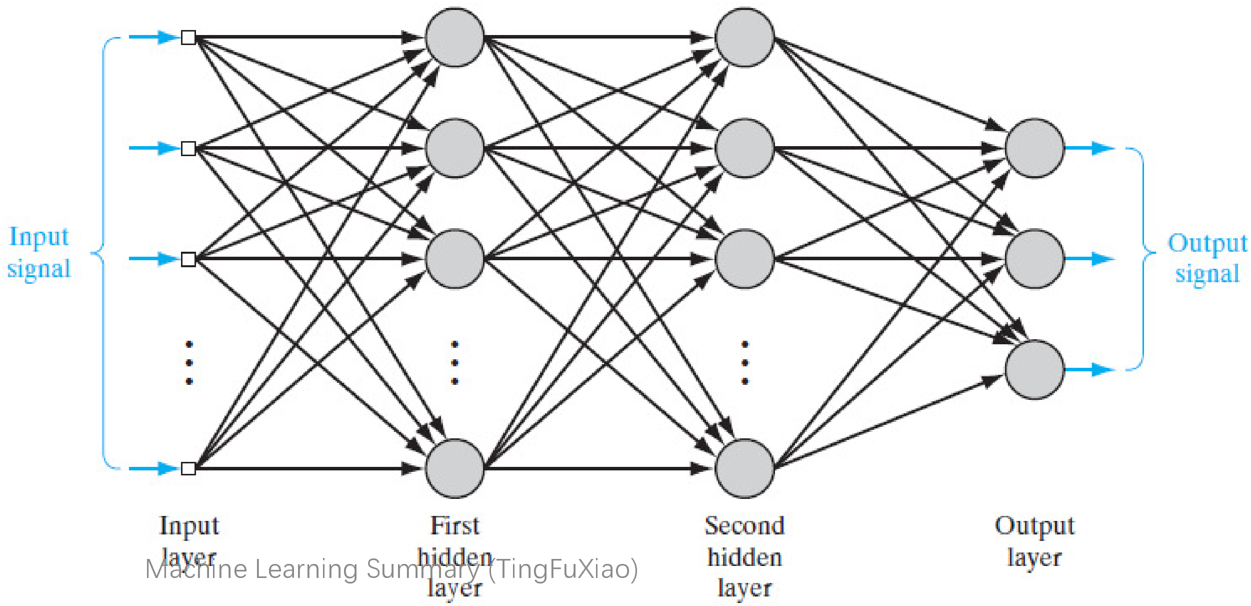
$$v=\sum_{i=1}^{n} w_i x_i + b = \sum_{i=0}^{n} w_i x_i$$

$$y=\varphi(v)=\varphi(\sum_{i=0}^{n} w_i x_i)$$



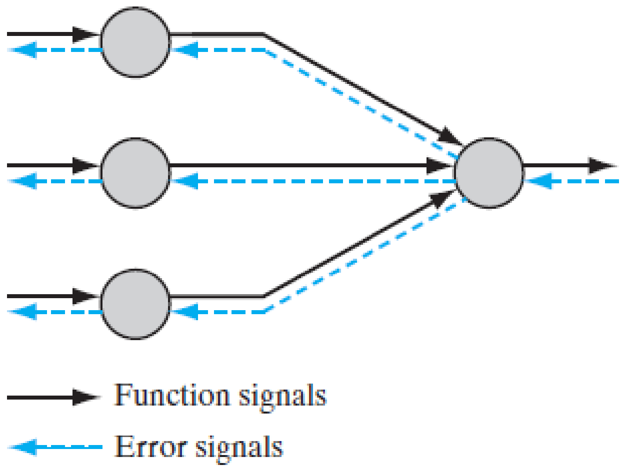
$$\boxed{\begin{aligned} w_n &= w_{n-1} + \eta d_i x_i \\ b_n &= b_{n-1} + \eta d_i \end{aligned}}$$

Multilayer perceptron:



Multilayer perceptron

- Computation on Output layer and Hidden layer



1. Function signals: computation of the function signal appearing at the output of each neuron

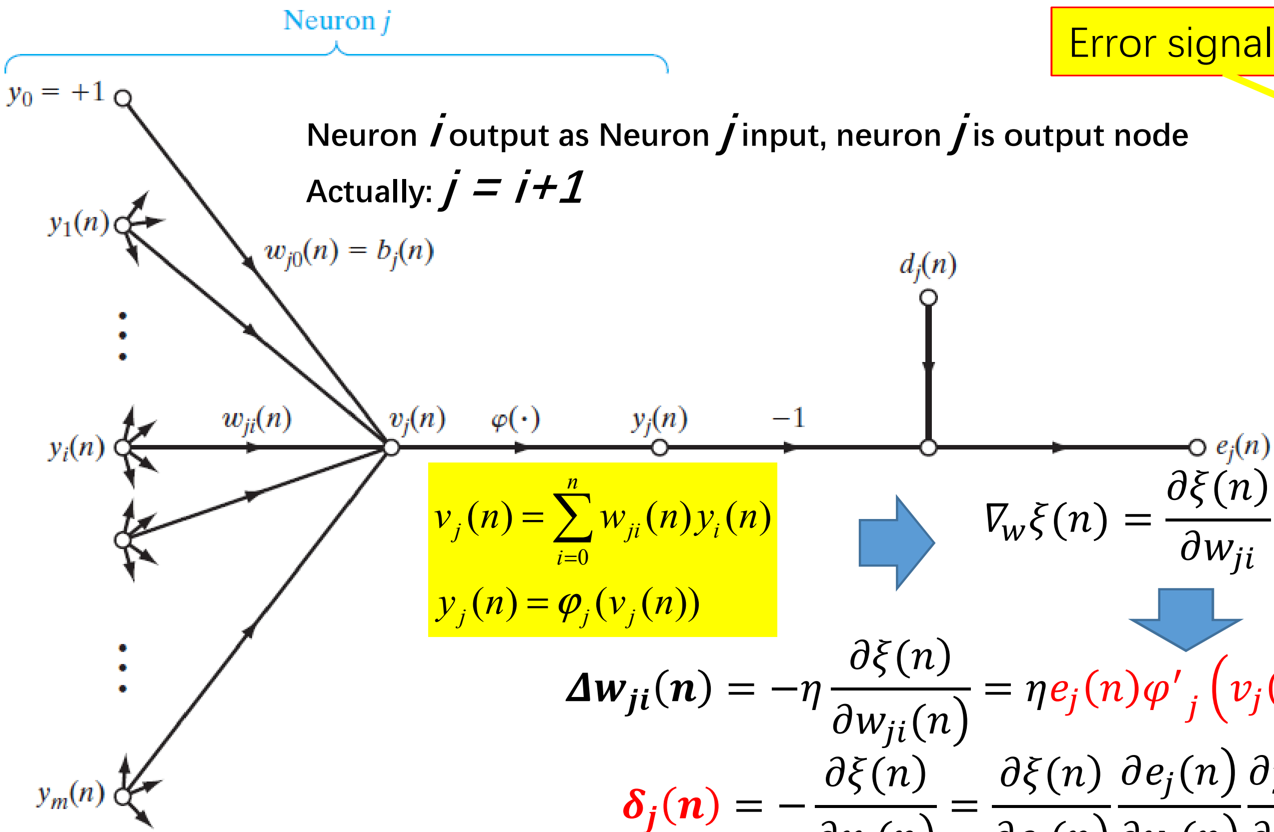
continuous nonlinear function of the input signal and synaptic weights associated with that neuron

2. Error signals: computation of an estimate of the gradient vector

Backward propagation algorithm

Multilayer perceptron

- Back propagation algorithm – Forward propagation



Neuron i output as Neuron j input, neuron j is output node

Actually: $j = i+1$

$$v_j(n) = \sum_{i=0}^{n} w_{ji}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v_j(n))$$

Error signal produced at the output neuron j

$$e_j(n) = d_j(n) - y_j(n)$$

$$\xi_j(n) = \frac{1}{2} e_j^2(n)$$

$$\xi(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

$$\nabla_w \xi(n) = \frac{\partial \xi(n)}{\partial w_{ji}} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}}$$

$$= -e_j(n) \varphi'_j\left(v_j(n)\right) y_i(n)$$

$$\boldsymbol{\Delta w_{ji}(n)} = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} = \eta e_j(n) \varphi'_j\left(v_j(n)\right) y_j(n) = \eta \delta_j(n) y_j(n)$$

delt rule on output Neuron j

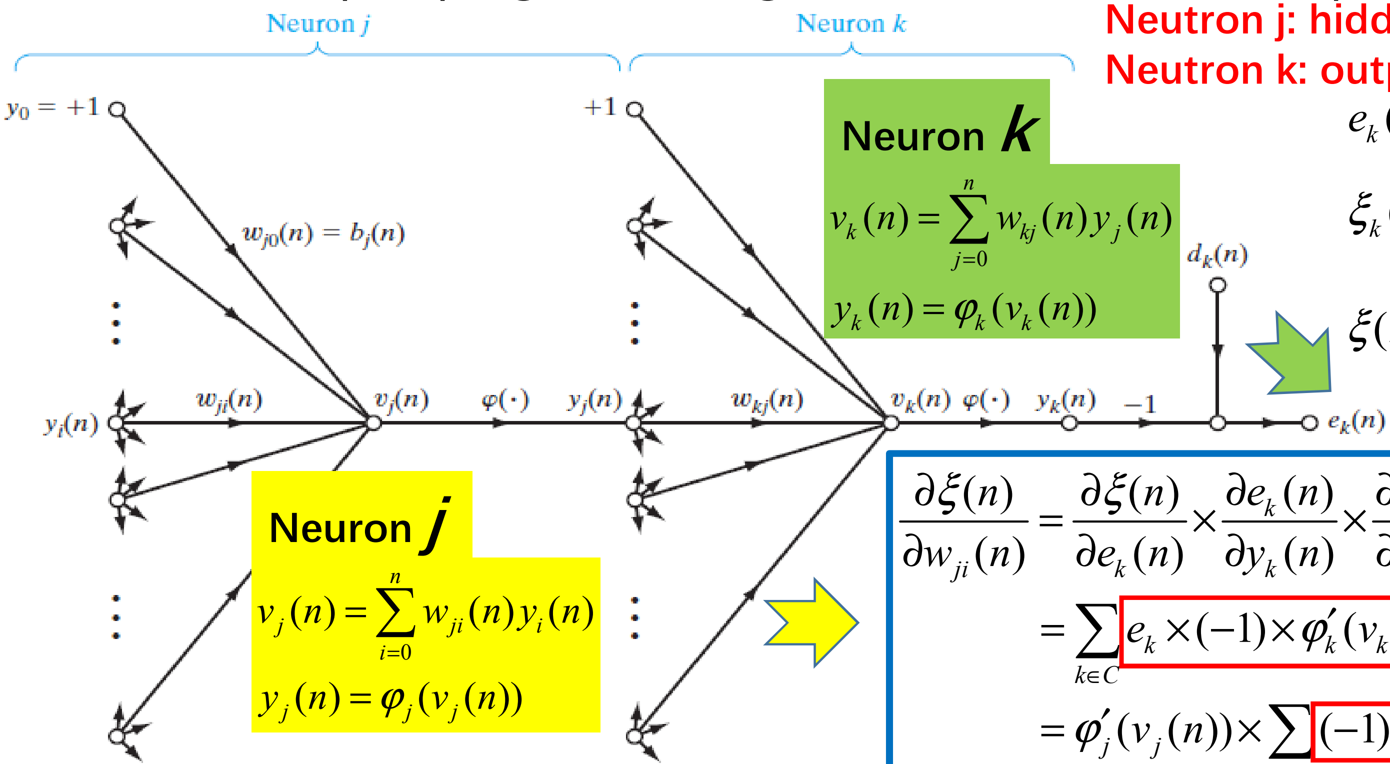
$$\boldsymbol{\delta_j(n)} = -\frac{\partial \xi(n)}{\partial v_j(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi'_j\left(v_j(n)\right)$$

Local gradient

Multilayer perceptron

- Back propagation algorithm – backward propagation

Neutron j: hidden node, can not get error signal directly
Neutron k: output node (k= j+1)



Neuron *k*

$$v_k(n)=\sum_{j=0}^{n} w_{kj}(n)y_j(n)$$

$$y_k(n)=\varphi_k(v_k(n))$$

Neuron *j*

$$v_j(n)=\sum_{i=0}^{n} w_{ji}(n)y_i(n)$$

$$y_j(n)=\varphi_j(v_j(n))$$

$$e_k(n)=d_k(n)-y_k(n)$$

$$\xi_k(n)=\frac{1}{2}e_k^2(n)$$

$$\xi(n)=\frac{1}{2}\sum_{j\in C}e_k^2(n)$$

$$\frac{\partial\xi(n)}{\partial w_{ji}(n)}=\frac{\partial\xi(n)}{\partial e_k(n)}\times\frac{\partial e_k(n)}{\partial y_k(n)}\times\frac{\partial y_k(n)}{\partial v_k(n)}\times\frac{\partial v_k(n)}{\partial y_j(n)}\times\frac{\partial y_j(n)}{\partial v_j(n)}\times\frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$=\sum_{k\in C}\boxed{e_k\times(-1)\times\varphi_k'(v_k(n))}\times w_{kj}(n)\times\varphi_j'(v_j(n))\times y_i(n)$$

$$=\varphi_j'(v_j(n))\times\sum_{k\in C}\boxed{(-1)\times\delta_k(n)}\times w_{kj}(n)\times y_i(n)$$

Multilayer perceptron

- Back propagation algorithm – backward propagation

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_k(n)} \times \frac{\partial e_k(n)}{\partial y_k(n)} \times \frac{\partial y_k(n)}{\partial v_k(n)} \times \frac{\partial v_k(n)}{\partial y_j(n)} \times \frac{\partial y_j(n)}{\partial v_j(n)} \times \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$= \sum_{k \in C} \boxed{e_k \times (-1) \times \varphi_k'(v_k(n))} \times w_{kj}(n) \times \varphi_j'(v_j(n)) \times y_i(n)$$

$$= \varphi_j'(v_j(n)) \times \sum_{k \in C} \boxed{(-1) \times \delta_k(n)} \times w_{kj}(n) \times y_i(n)$$

Local gradient in hidden node:

$$\boxed{\begin{aligned} \boldsymbol{\delta_j(n)} &= -\frac{\partial \xi(n)}{\partial v_j(n)} = \frac{\partial \xi(n)}{\partial e_k(n)} \times \frac{\partial e_k(n)}{\partial y_k(n)} \times \frac{\partial y_k(n)}{\partial v_k(n)} \times \frac{\partial v_k(n)}{\partial y_j(n)} \times \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= \varphi'_j(v_j(n)) \times \sum_{k \in C} \delta_k(n) \times w_{kj}(n) \end{aligned}}$$

delt rule on Hidden Neuron *j*:

$$\boxed{\Delta w_{ji} = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} = \eta \delta_j(n) y_j(n)}$$

Multilayer perceptron

- Back propagation algorithm – delta rules for updating weight

The correction Δw_{ji} applied to the synaptic connecting **neuron *i*** to **neuron *j*** is defined:

$$\boxed{\begin{pmatrix} \textit{Weight} \\ \textit{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \textit{learning-} \\ \textit{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \textit{local} \\ \textit{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \textit{input signal} \\ \textit{of neuron } j, \\ y_i(n) \end{pmatrix}}$$

$$\boldsymbol{Output\ node}: \boldsymbol{\delta_j(n)} = -\frac{\partial \xi(n)}{\partial v_j(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j\left(v_j(n)\right)$$

$$\boldsymbol{Hidden\ node}: \boldsymbol{\delta_j(n)} = -\frac{\partial \xi(n)}{\partial v_j(n)} = \frac{\partial \xi(n)}{\partial e_k(n)} \times \frac{\partial e_k(n)}{\partial y_k(n)} \times \frac{\partial y_k(n)}{\partial v_k(n)} \times \frac{\partial v_k(n)}{\partial y_j(n)} \times \frac{\partial y_j(n)}{\partial v_j(n)}$$

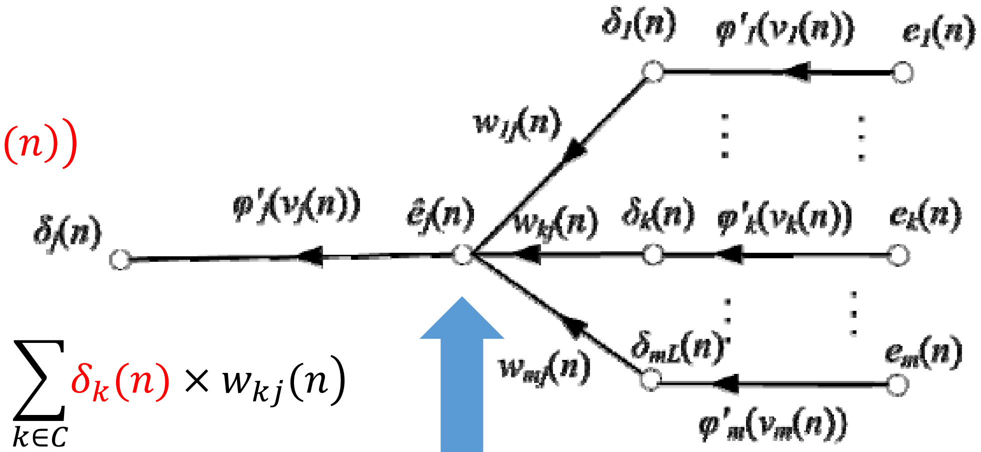
$$= \varphi'_j(v_j(n)) \times \sum_{k \in C} \delta_k(n) \times w_{kj}(n)$$

Multilayer perceptron

- Back propagation algorithm – propagation process of error signal

$$\boldsymbol{Output\ node\ k}: \boldsymbol{\delta_k(n)} = e_k(n)\varphi'_k(v_k(n))$$

$$\boldsymbol{Hidden\ node\ j}: \boldsymbol{\delta_j(n)} = \varphi'_j(v_j(n)) \times \sum_{k \in C} \delta_k(n) \times w_{kj}(n)$$

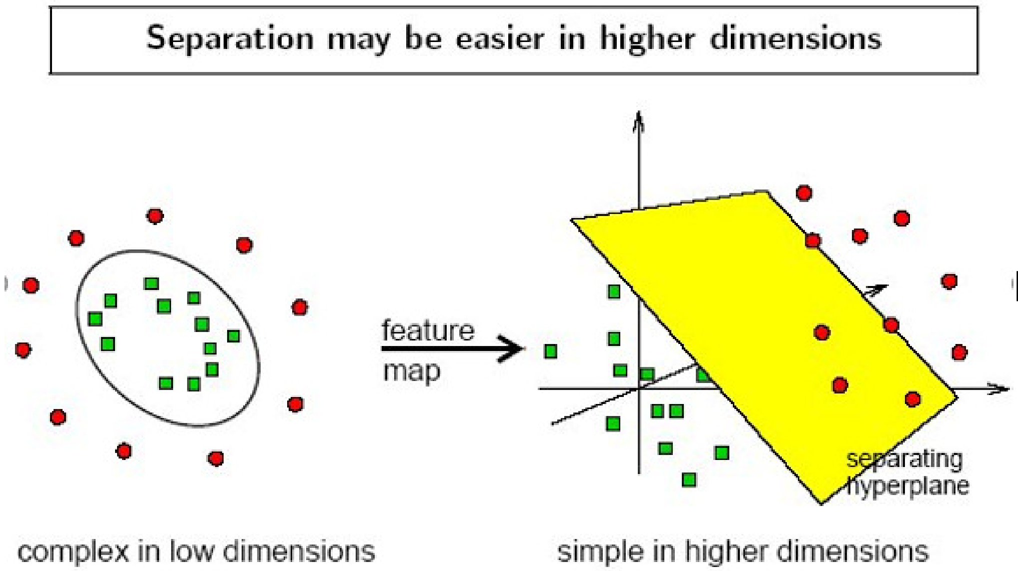


$$\tilde{e}_j(n) = \sum_{k \in C} \delta_k(n) \times w_{kj}(n)$$

Kernel Method and RBF

- Cover theorem

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.



low-dimensional: non-separable linearly

high-dimensional, more likely to be linearly separable.

Kernel Method and RBF

- Inner product and Kernel Method

$$X_1 = (x_{11}, x_{12}, \cdots, x_{1m_0})^T$$
$$X_2 = (x_{21}, x_{22}, \cdots, x_{2m_0})^T$$

$$\Rightarrow \langle X_1, X_2 \rangle = X^T{}_1 \cdot X_2 = [x_{11}, x_{12}, \cdots, x_{1m_0}] \cdot \begin{bmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2m_0} \end{bmatrix} = \sum_i^{m_0} x_{1i} x_{2i}$$

$$X : (x_1, x_2, \cdots, x_{m_0}) \in \mathbb{R}^{m_0}$$

Input Space

m1>>m0

$$\phi_i(X): \quad \{\varphi_i(X) | i = 1, 2, \cdots, m_1\}: \quad \{\varphi_1(X), \varphi_2(X), \varphi_3(X), \cdots, \varphi_{m_1}(X)\} \in \mathbb{R}^{m1}$$

Feature Space

$$\boxed{\text{kernel} : k(x, z) = \varphi^T(x) \bullet \varphi(z) = \langle \varphi(x), \varphi(z) \rangle}$$

Kernel method and RBF

- linearly separable in high dimensional

$$input\ space\ linearly \quad VS \quad feature\ space\ linearly$$

$$y_i = \begin{cases} 1 & W^T X_i \geq 0 \\ -1 & W^T X_i < 0 \end{cases} \quad VS \quad y_i = \begin{cases} 1 & W^T \phi_i(X) \geq 0 \\ -1 & W^T \phi_i(X) < 0 \end{cases} \qquad hyperplane: \quad W^T \varphi_i(X) = 0$$

Kernel method and RBF

- Solution: Gram For Training set

$$Training\ set(X_i, y_i): 0 \le i \le n$$

$$y_i = W^T X_i \to \begin{cases} y_i = W^T \phi_i(X_i) & \phi_i(X_i): \quad \{\varphi_1(X_i), \varphi_2(X_i), \varphi_3(X_i), \cdots, \varphi_{m_1}(X_i)\}^T \in \mathbb{R}^{m1} \\ \vec{y} = \Phi(X) W & \Phi(X): \quad \{\phi_1(X_1), \phi_2(X_2), \phi_3(X_3), \cdots, \phi_n(X_n)\} \in \mathbb{R}^{n \times m1} \end{cases}$$

$$\boxed{l(W) = \sum_{i=0}^{n} (y_i - d_i)^2 + \frac{\lambda}{2} W^T W = \frac{1}{2}\left(\Phi(X)W - \vec{d}\right)^T \left(\Phi(X)W - \vec{d}\right) + \frac{\lambda}{2} W^T W}$$

$$\boxed{\begin{aligned} \nabla_W l(W) &= \frac{1}{2} \nabla_W \left(W^T \Phi(X)^T \Phi(X) W - W^T \Phi(X)^T \vec{d} - \vec{d}^T \Phi(X) W + \vec{d}^T \vec{d} + \lambda W^T W\right) \\ &= \frac{1}{2} \nabla_W Trace\left(W^T \Phi(X)^T \Phi(X) W - 2\vec{d}^T \Phi(X) W + \lambda W^T W\right) \\ &= \Phi(X)^T \Phi(X) W - \Phi(X)^T \vec{d} + \lambda W \end{aligned}}$$

Kernel method and Radial Basis Function

- Solution: Gram For Training set

$$\nabla_W l(W) = 0 \quad \Rightarrow \quad \Phi(X)^T \Phi(X) W - \Phi(X)^T \vec{d} + \lambda W = 0$$

$$\Rightarrow \quad W = \lambda^{-1}\Phi(X)^T \left(\vec{d} - \Phi(X) W\right) = \Phi(X)^T \boldsymbol{\alpha}$$

$$\Rightarrow \quad \boldsymbol{\alpha} = \lambda^{-1}\left(\vec{d} - \Phi(X) W\right)$$

W : *Linear combination of training points*

$\boldsymbol{\alpha}$: *dual variable*

$$\boldsymbol{\alpha} = \lambda^{-1}\left(\vec{d} - \Phi(X)\Phi(X)^T W\right) \quad \Rightarrow \quad \boldsymbol{\alpha} = \left(\Phi(X)\Phi(X)^T + \lambda E\right)^{-1} \vec{d}$$

$$\Rightarrow \quad \boldsymbol{\alpha} = (G + \lambda E)^{-1} \vec{d} \in \mathbb{R}^{n\times 1}, \quad G = \Phi(X)\Phi(X)^T \in \mathbb{R}^{n\times n}$$

Gram Matrix

$G_{ij} = \left\langle \phi_i(X_i), \phi_j(X_j) \right\rangle_{n\times n}$

Inner product between training points

$$y_i = W^T \phi_i(X_i) = \boldsymbol{\alpha}^T \Phi(X) \phi_i(X_i) = \sum_{j=1}^{n} \alpha_j \phi_j(X_j) \phi_i(X_i) = \sum_{j=1}^{n} \alpha_j \left(\phi_j(X_j) \phi_i(X_i)\right)$$

$$K\left\langle \phi_j(X_j)_{1\times m1}, \phi_i(X_i)_{1\times m1} \right\rangle$$

$$\boxed{y_i = \sum_{j=1}^{n} \alpha_j \left(K\left\langle \phi_j(X_j), \phi_i(X_i) \right\rangle\right)}$$

$\alpha^T : 1\times n$

$\Phi(X) : n\times m1 \rightarrow \phi_j(X_j) : 1\times m1$

$\phi_i(X_i) : m1\times 1$

Kernel method and RBF

- Property of Kernel method

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') k_b(\mathbf{x}_b, \mathbf{x}_b')
\end{aligned}
$$

Kernel method and Radial Basis Function

- Common Kernel method

Polynomial kernel :

$$k(x,x')=(x^T x')^2$$

$$k(x,x')=(x^T x'+c)^2$$

$$k(x,x')=(x^T x')^M$$

$$k(x,x')=(x^T x'+c)^M$$

高斯核(Gaussian kernel)

$$k(\mathbf{x},\mathbf{x}')=\exp\left(-\|\mathbf{x}-\mathbf{x}'\|^2/2\sigma^2\right)$$

Note: can substitute $\boldsymbol{x}^T\boldsymbol{x}'$ with a nonlinear kernel $\kappa(\boldsymbol{x},\boldsymbol{x}')$

$$k(\mathbf{x},\mathbf{x}')=\exp\left(-\mathbf{x}^{\mathrm{T}}\mathbf{x}/2\sigma^2\right)\exp\left(\mathbf{x}^{\mathrm{T}}\mathbf{x}'/\sigma^2\right)\exp\left(-(\mathbf{x}')^{\mathrm{T}}\mathbf{x}'/2\sigma^2\right)$$

Sigmoid kernel: $k(\boldsymbol{x},\boldsymbol{x}')=\tanh(a\boldsymbol{x}^T\boldsymbol{x}'+b)$

定义在非向量类型数据的Kernel:

$$k(A_1,A_2)=2^{|A_1\cap A_2|}$$

Kernel method and Radial Basis Function

- Common Kernel method

基于生成模型的核

$$
\begin{aligned}
k(\boldsymbol{x}, \boldsymbol{x}') &= p(\boldsymbol{x})p(\boldsymbol{x}') \\
k(\boldsymbol{x}, \boldsymbol{x}') &= \sum_i p(\boldsymbol{x}|i)p(\boldsymbol{x}'|i)p(i) \\
k(\boldsymbol{x}, \boldsymbol{x}') &= \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{x}'|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}
\end{aligned}
$$

基于隐马尔科夫模型(HMM)的核

$$
k(\mathbf{X}, \mathbf{X}') = \sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z})p(\mathbf{X}'|\mathbf{Z})p(\mathbf{Z})
$$

$\mathbf{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_L\}$ - observations
$\mathbf{Z} = \{\boldsymbol{z}_1, ..., \boldsymbol{z}_L\}$ - hidden states

Kernel Method and Radial Basis Function

- Network for kernel method when using regularization

$$\boldsymbol{\alpha} = \left(G + \lambda E\right)^{-1} \vec{d} \in \mathbb{R}^{n\times 1}$$

$$G = \Phi\left(X\right)\Phi\left(X\right)^{T} \in \mathbb{R}^{n\times n}$$

$\alpha^{T}: 1\times n$

$\Phi(X): n\times m1 \rightarrow \phi_j(X_j): 1\times m1$

$\phi_i(X_i): m1\times 1$

$$y_i = \sum_{j=1}^{n} \alpha_j \left(K\left\langle \phi_j\left(X_j\right), \phi_i\left(X_i\right)\right\rangle\right), \quad n: \textit{size of training set}$$

Shortcoming: Huge compute, since n (size of training set) will be huge.
1. computer the inverse matrix of nxn matrix. $o(n^3)$
2. increase possibility of ill-matrix when increase n.

- Radial Basis Function - RBF

Kernel Method and Radial Basis Function

- RBF Interpolation

Interpolation is the process of estimating unknown values that fall between known values

Multivariable Interpolation:

Given a set of N different points $\{\mathbf{x}_i \in \mathbb{R}^{m_0} | i = 1, 2, ..., N\}$ *and a corresponding set of N real numbers* $\{d_i \in \mathbb{R}^1 | i = 1, 2, ..., N\}$, *find a function* $F: \mathbb{R}^N \to \mathbb{R}^1$ *that satisfies the interpolation condition:*

$$F(\mathbf{x}_i) = d_i, \qquad i = 1, 2, ..., N$$

- Radial Basis Function

$$F(x) = \sum_{i=1}^{n} w_i \varphi\left(\|x - x_i\|\right)$$

Radial Basis Function

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}$$

$$\varphi_{ij} = \varphi\left(\|x_i - x_j\|\right), \quad i, j = 1, 2, \cdots, N$$

Kernel Method and Radial Basis Function

- Common Radial Basis Function

1. *Multiquadrics:*

$$\varphi(r) = (r^2 + c^2)^{1/2} \quad \text{for some } c > 0 \text{ and } r \in \mathbb{R}$$

2. *Inverse multiquadrics:*

$$\varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \quad \text{for some } c > 0 \text{ and } r \in \mathbb{R}$$

3. *Gaussian functions:*

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \text{for some } \sigma > 0 \text{ and } r \in \mathbb{R}$$

Kernel Method and Radial Basis Function

- RBF Network

1. Input Layer: $X_i = (x_{i1}, x_{i2}, \cdots, x_{im_0})^T$

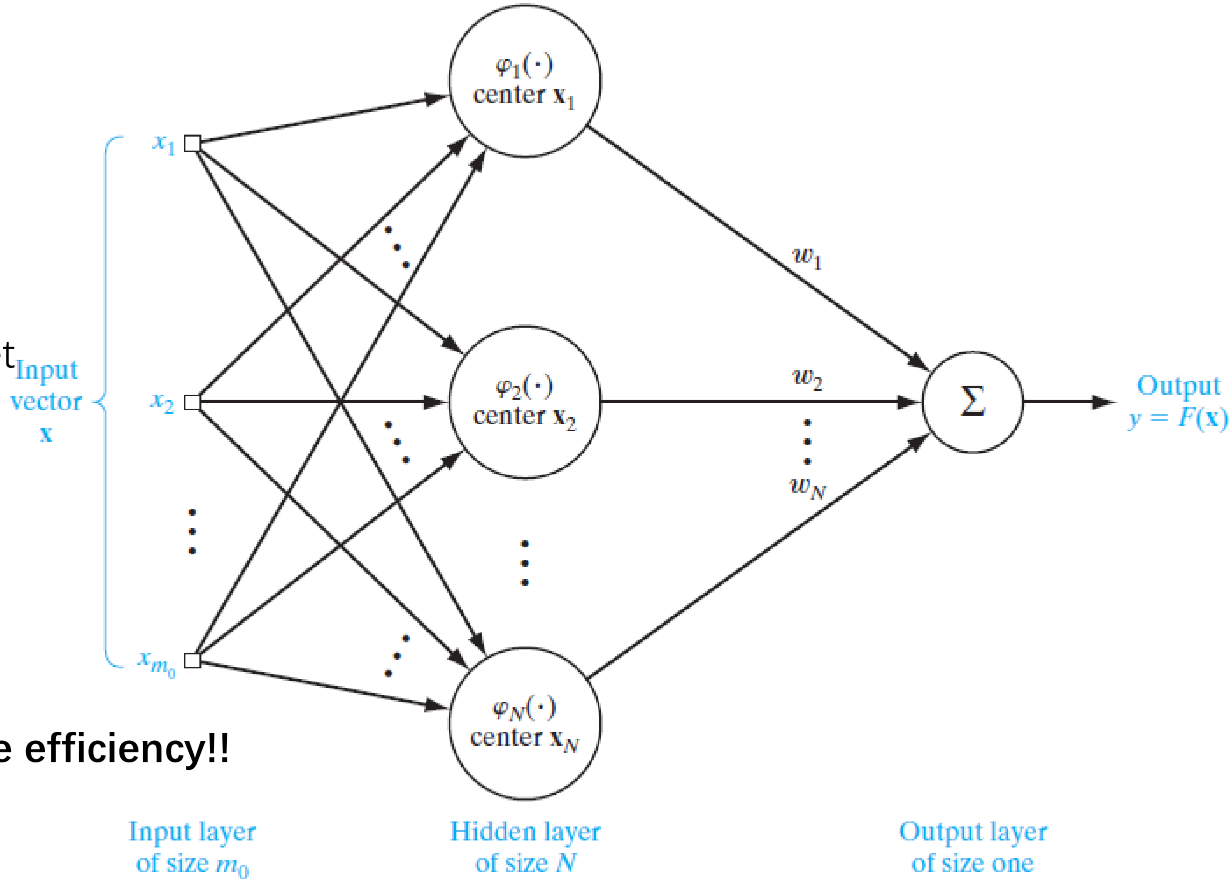
2. Hidden Layer: the size of the training set

$$\varphi_j(X_i) = \varphi\left(\left\|X_i - X_j\right\|\right), j = 1, 2, \cdots, N$$

3. Output Layer:

$$y_i = F(X_i) = \sum_{j=1}^{N} w_j \varphi\left(\left\|X_i - X_j\right\|\right)$$

Note: Once N is too huge, will affect the efficiency!!

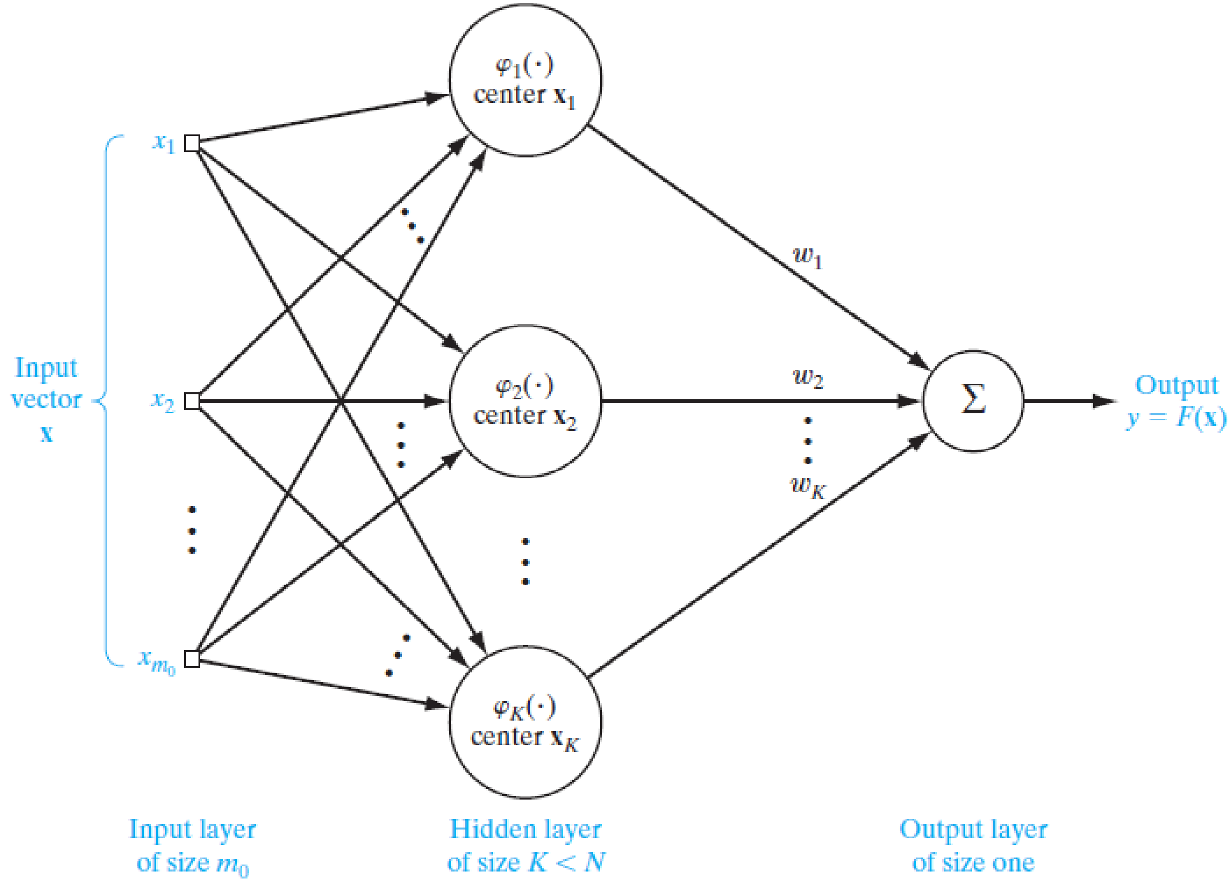


Kernel Method and Radial Basis Function

- RBF Network Modification

Redundancy of neurons in hidden layer

$$N \quad \rightarrow \quad K \ll N$$



Kernel Method and Radial Basis Function

- RBF Network for K-Means Clustering

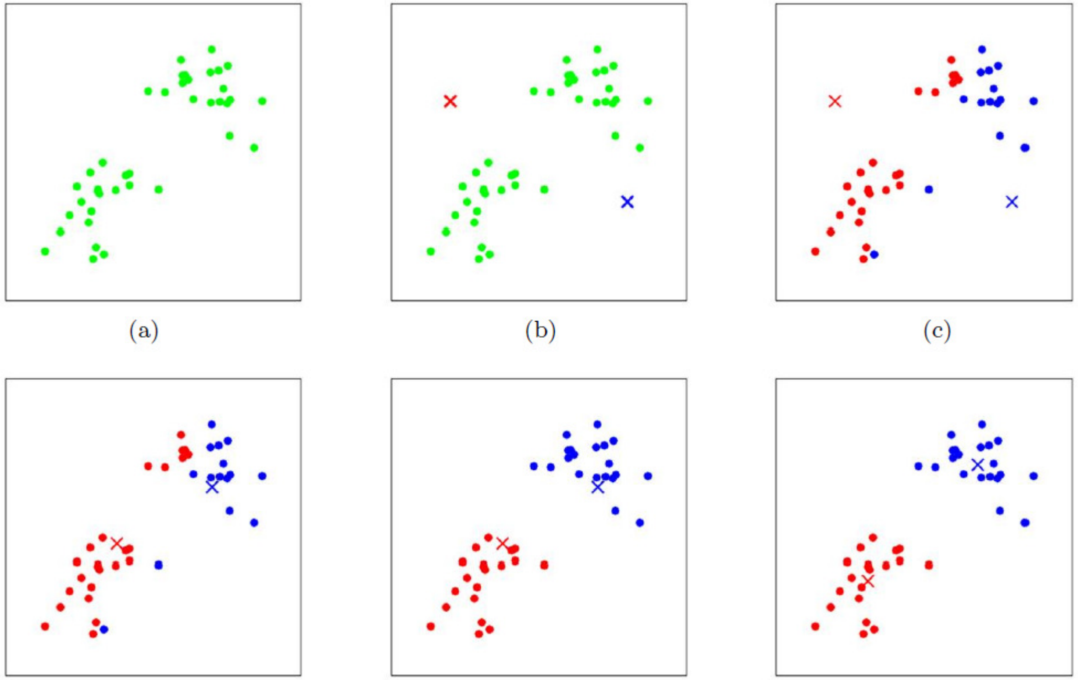
The k-means clustering algorithm is as follows:

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

    For every i, set
    $$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

    For each j, set
    $$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$$

    }



Kernel Method and Radial Basis Function

- RLS algorithm for RBF

$$Training\ set(X_i, y_i): 0 \le i \le n$$

$$y_i = W^T X_i \rightarrow \begin{cases} y_i = W^T \phi_i(X_i) \quad \phi_i(X_i): \quad \{\varphi_1(X_i), \varphi_2(X_i), \varphi_3(X_i), \cdots, \varphi_{m_1}(X_i)\}^T \in \mathbb{R}^{m1\times 1} \\ \qquad \varphi_j(X_i) = \varphi\left(\left\|X_i - X_j\right\|\right), j = 1, 2, \ldots, m_1 \\ \vec{y} = \Phi(X)W \qquad \Phi(X): \quad \{\phi_1(X_1), \phi_2(X_2), \phi_3(X_3), \cdots, \phi_n(X_n)\} \in \mathbb{R}^{m1\times n} \end{cases}$$

$$l(W) = \sum_{i=0}^{n}(y_i - d_i)^2 = \frac{1}{2}\left(\Phi(X)W - \vec{d}\right)^T\left(\Phi(X)W - \vec{d}\right)$$

$$\nabla_W l(W) = \frac{1}{2}\nabla_W\left(W^T\Phi(X)^T\Phi(X)W - W^T\Phi(X)^T\vec{d} - \vec{d}^T\Phi(X)W + \vec{d}^T\vec{d}\right)$$

$$= \frac{1}{2}\nabla_W Trace\left(W^T\Phi(X)^T\Phi(X)W - 2\vec{d}^T\Phi(X)W\right)$$

$$= \underbrace{\Phi(X)^T\Phi(X)}_{R(n)}W - \underbrace{\Phi(X)^T\vec{d}}_{r(n)}$$

Kernel Method and Radial Basis Function

- RLS algorithm for RBF

$$\Phi(X)=\left(\phi_1(X_1),\phi_2(X_2),\phi_3(X_3),\cdots,\phi_n(X_n)\right)$$
$$=\begin{pmatrix} \varphi_1(X_1) & \varphi_1(X_2) & \cdots & \varphi_1(X_n) \\ \varphi_2(X_1) & \varphi_2(X_2) & \cdots & \varphi_2(X_n) \\ \vdots & \vdots & \cdots & \vdots \\ \varphi_{m1}(X_1) & \varphi_{m1}(X_2) & \cdots & \varphi_{m1}(X_n) \end{pmatrix}$$

$$\Phi(X)^T=\begin{pmatrix} \varphi_1(X_1) & \varphi_2(X_1) & \cdots & \varphi_n(X_1) \\ \varphi_1(X_2) & \varphi_2(X_2) & \cdots & \varphi_n(X_2) \\ \vdots & \vdots & \cdots & \vdots \\ \varphi_1(X_n) & \varphi_2(X_n) & \cdots & \varphi_n(X_n) \end{pmatrix}$$

$$\Phi(X)^T\Phi(X)=\begin{pmatrix} \sum_{i=1}^n \varphi_i^2(X_1) & \sum_{i=1}^n \varphi_i(X_1)\varphi_i(X_2) & \cdots & \sum_{i=1}^n \varphi_i(X_1)\varphi_i(X_n) \\ \sum_{i=1}^n \varphi_i(X_1)\varphi_i(X_2) & \sum_{i=1}^n \varphi_i^2(X_2) & \cdots & \sum_{i=1}^n \varphi_i(X_2)\varphi_i(X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n \varphi_i(X_1)\varphi_i(X_n) & \sum_{i=1}^n \varphi_i(X_2)\varphi_i(X_n) & \cdots & \sum_{i=1}^n \varphi_i^2(X_n) \end{pmatrix}$$

$$R(n)=\Phi(X)^T\Phi(X)=\sum_{i=1}^n \phi_i(X_i)\phi_i^T(X_i)$$
$$=R(n-1)+\phi_n(X_n)\phi_n^T(X_n)$$

$$r(n)=\Phi(X)^T\vec{d}=\sum_{i=1}^n \phi_i(X_i)d_i$$
$$=r(n-1)+\phi_n(X_n)d_n$$
$$=R(n-1)W(n-1)+\phi_n(X_n)d_n$$
$$=\left(R(n-1)+\phi_n(X_n)\phi_n^T(X_n)\right)W(n-1)+\phi_n(X_n)\left(d_n-\phi_n^T(X_n)W(n-1)\right)$$
$$=R(n)W(n-1)+\phi_n(X_n)\alpha(n)$$

$$\alpha(n)=d_n-\phi_n^T(X_n)W(n-1)$$

$$R(n)W(n)=R(n)W(n-1)+\phi_n(X_n)\alpha(n)$$
$$W(n)=W(n-1)+R^{-1}(n)\phi_n(X_n)\alpha(n)$$

Kernel Method and Radial Basis Function

- RSL algorithm - $R^{-1}(n)$

$$W(n) = W(n-1) + R^{-1}(n)\phi_n(X_n)\alpha(n)$$

$$R(n) = \Phi(X)^T\Phi(X) = \sum_{i=1}^{n}\phi_i(X_i)\phi_i^T(X_i)$$

$$= R(n-1) + \phi_n(X_n)\phi_n^T(X_n)$$

$$A = B^{-1} + CDC^T$$

$$A^{-1} = B - BC(D + C^T BC)^{-1}C^T B$$

$$A = R(n)$$

$$B^{-1} = R(n-1)$$

$$C = \phi_n(X_n)$$

$$D = 1$$

$$R^{-1}(n) = R^{-1}(n-1) - \frac{R^{-1}(n-1)\phi_n(X_n)\phi_n^T(X_n)R^{-1}(n-1)}{1 + \phi_n(X_n)R^{-1}(n-1)\phi_n(X_n)}$$

$$R^{-1}(n) = p(n)$$

$$\boxed{p(n) = p(n-1) = \frac{p(n-1)\phi_n(X_n)\phi_n^T(X_n)p(n-1)}{1 + \phi_n(X_n)p(n-1)\phi_n(X_n)}}$$

Kernel Method and Radial Basis Function

- RSL algorithm for learning

Step-1: Map primary data to a new space using Radial Basis Function

$$\phi_j(X_i)=\left\{\varphi_j(X_i)\mid j=1,2,\cdots,m1\right\}=\left\{\varphi\left(\left\|X_i-X_j\right\|\right)\mid j=1,2,\cdots,m1\right\}$$

$$Training\ set:(X_i,d_i)\quad\Rightarrow\quad\left(\phi_i(X_i),d_i\right)$$

$$\phi_i(X_i):\quad\{\varphi_1(X_i),\varphi_2(X_i),\varphi_3(X_i),\cdots,\varphi_{m_1}(X_i)\}^T\in\mathbb{R}^{m1\times1}$$

Step-2: Initialization for W(0), and P(0)

$$W(0)=0,P(0)=\lambda^{-1}E$$

Regularization Item

$$\frac{1}{2}\lambda\|W\|^2=\frac{1}{2}\lambda W^TW$$

Step-3: Recursive iteration

$$\boxed{W(n)=W(n-1)+p(n)\phi_n(X_n)\alpha(n)=g(n)\alpha(n)}$$

$$p(n)=p(n-1)=\frac{p(n-1)\phi_n(X_n)\phi_n^T(X_n)p(n-1)}{1+\phi_n(X_n)p(n-1)\phi_n(X_n)}\qquad g(n)=p(n)\phi_n(X_n)\qquad\alpha(n)=d_n-\phi_n^T(X_n)W(n-1)$$

Supported Vector Machine

- Introduction

Supported Vector Machine (SVM): a machine learning algorithm that is perhaps the most elegant of all kernel-learning methods

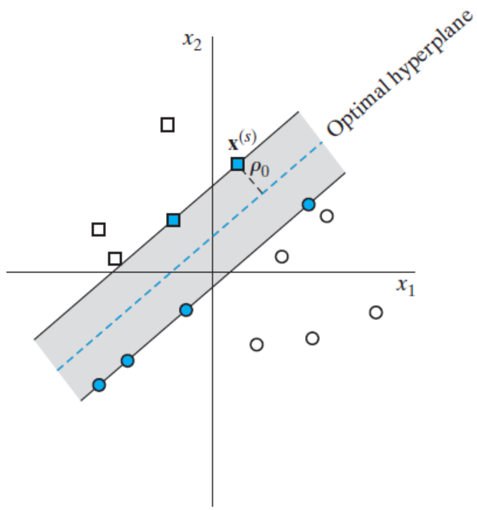
Given a training set, the support vector machine constructs a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized

Inner-product kernel between a support vector X(i) and a vector X drawn from the input data space

Supported Vector Machine

- Maximum margin Hyperplane for Linearly separable

$$y_i = W^T X_i + b \quad sign(y_i) = \begin{cases} 1 & W^T X_i + b \geq 0 \\ -1 & W^T X_i + b < 0 \end{cases} \quad Hyperplane: W^T X + b = 0$$



Distance from X_i to the optimal hyperplane: $r = \dfrac{W^T X_i + b}{\|W\|}$

Supported Vector: $W^T X_i + b = \pm 1$

Target:

$$y_i = W^T X_i + b \quad sign(y_i) = \begin{cases} 1 & W^T X_i + b \geq 1 \\ -1 & W^T X_i + b < -1 \end{cases} \quad Hyperplane: W^T X + b = 0$$

Supported Vector Machine

- Maximum margin Hyperplane for Linearly separable

For Supported Vector: $r = \dfrac{W^T X_i + b}{\|W\|} = \begin{cases} \dfrac{1}{\|W\|} & d_i = 1 \\ -\dfrac{1}{\|W\|} & d_i = -1 \end{cases}$

Target:

$$\boxed{\begin{aligned} &maximize\left(\frac{1}{\|W\|}\right) \Rightarrow min\left(\frac{1}{2}W^T W\right) \\ &st: d_i\left(W^T X + b\right) \geq 1, \quad for\ i = 1, 2, \cdots, n \end{aligned}}$$

Given the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^N$, *find the optimum values of the weight vector* $\mathbf{w}$ *and bias* b *such that they satisfy the constraints*

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \text{for } i = 1, 2, ..., N$$

and the weight vector $\mathbf{w}$ *minimizes the cost function*

$$\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

Supported Vector Machine

- Solution for Maximum margin Hyperplane

$$maximize\left(\frac{1}{\|W\|}\right) \Rightarrow min\left(\frac{1}{2}W^T W\right)$$

$$s.t. \quad d_i\left(W^T X + b\right) \geq 1, \quad for\ i = 1, 2, \cdots, n$$

Important Note:

$$\alpha_i = \begin{cases} !0 & if\ \alpha_i\left[d_i\left(W^T X_i + b\right) - 1\right] = 0 \\ 0 & otherwise \end{cases}$$

$$\Rightarrow \alpha_i \neq 0\ only\ for\ supported\ vectors$$

Lagrangian function:

$$L(w, b, \alpha) = \frac{1}{2}W^T W - \sum_{i=1}^{n} \alpha_i\left[d_i\left(W^T X_i + b\right) - 1\right]$$

α_i : *lagrangian multipliers*, $\alpha_i \geq 0$

$$\frac{\partial L(w, b, \alpha)}{\partial w} = 0$$

$$\frac{\partial L(w, b, \alpha)}{\partial b} = 0$$

$$W = \sum_{i=1}^{n} \alpha_i d_i X_i$$

$$\sum_{i=1}^{n} \alpha_i d_i = 0$$

$$Dual\ problem: L(w, b, \alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j X_i^T X_j$$

$$D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j X_i^T X_j$$

$$min\left(L(w, b, \alpha)\right) \Leftrightarrow max\left(D(\alpha)\right)$$

Supported Vector Machine

- Solution for Maximum margin Hyperplane

$Important\ Note:$

$$\alpha_i = \begin{cases} !0 & if\ \alpha_i\left[d_i\left(W^T X_i + b\right) - 1\right] = 0 \\ 0 & otherwise \end{cases}$$

$$\Rightarrow \alpha_i \neq 0\ only\ \ for\ supported\ vectors \quad \Rightarrow \quad \alpha_i^s$$

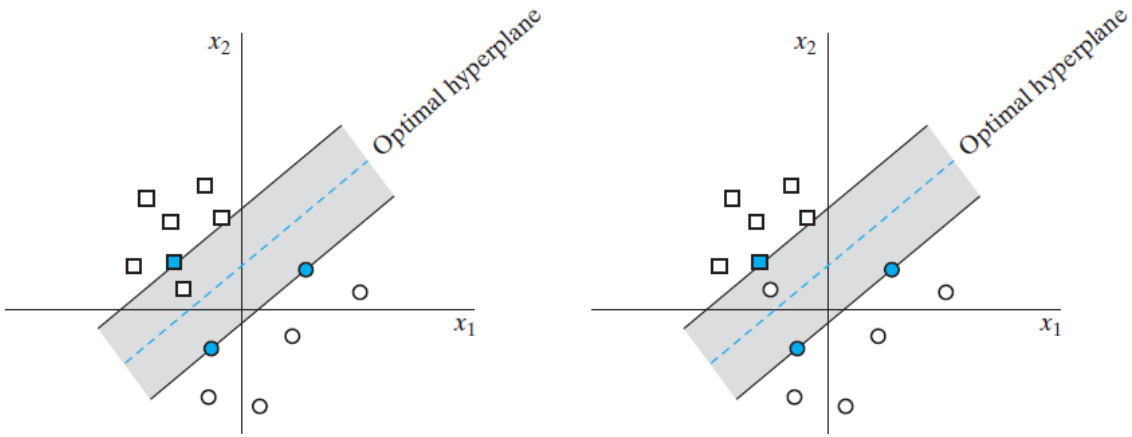
$$\boxed{\begin{aligned} & W^s = \sum_{i=1}^{n} \alpha_i d_i X_i = \sum_{i=1}^{ns} \alpha_i^s d_i X_i^s \\ & b^s = 1 - \left(W^s\right)^T X^s \end{aligned}}$$

Supported Vector Machine

- Maximum margin Hyperplane for Non-separable

It is not possible to construct a separating hyperplane without encountering classification errors.

Find a optimal hyperplane that minimizes the probability of classification error



Supported Vector Machine

- Maximum margin Hyperplane for Non-separable

$$d_i\left(W^T X + b\right) \ge 1, \quad for\ i = 1, 2, \cdots, n$$

Scalar variables

$$d_i\left(W^T X + b\right) \ge 1 - \xi_i, \quad for\ i = 1, 2, \cdots, n; \quad 0 < \xi_i \le 1$$

$$\boxed{\begin{aligned} &min\left(\frac{1}{2}W^T W + C\sum_{i=0}^{n}\xi_i\right) \\ &s.t. \quad d_i\left(W^T X + b\right) \ge 1 - \xi_i, \quad for\ i = 1, 2, \cdots, n \\ &\qquad 0 \le \xi_i < 1 \end{aligned}}$$

Given the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, *find the Lagrange multipliers* $\{\alpha_i\}_{i=1}^{N}$ *that maximize the objective function*

$$Q(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to the constraints

$$(1) \quad \sum_{i=1}^{N}\alpha_i d_i = 0$$

$$(2) \quad 0 \le \alpha_i \le C \qquad \text{for } i = 1, 2, ..., N$$

where C is a user-specified positive parameter.

1. **Nether the slack variable nor the Lagrange multipliers appear in the dual problem**

$2.\ \alpha_i \ge 0\ is\ replaced\ by\ 0 \le \alpha_i \le C$

Supported Vector Machine

- SVM based on Kernel method

$$\mathbf{w}^T\mathbf{x}+b=0 \xrightarrow{K(\mathbf{x},\mathbf{x}_i)=\Phi(\mathbf{x})^T\Phi(\mathbf{x}_i)} \mathbf{w}^T\Phi(\mathbf{x})+b=0$$

$$Q(\alpha)=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$$

$$\sum_{i=1}^{N}\alpha_i y_i=0$$

$$0\le\alpha_i\le C,\ \text{for } i=1,\dots,N$$

Supported Vector Machine

- SVM based on Kernel method – SMO algorithm

SMO: Sequential Minimal Optimization

每次求解仅涉及两个优化变量的二次规划问题

- 在每次迭代时，选中两个优化变量α_{i^*}和α_{j^*}，同时保持其它变量固定，求解关于α_{i^*}和α_{j^*}的二次规划问题
  - 在LibSVM中被采用

Principal Components Analysis

- Project Data to unit vector

Problem *Does there exist an invertible linear transformation* $\mathbf{T}$ *such that the truncation of* $\mathbf{Tx}$ *is optimum in the mean-square-error sense?*

$Assumption:$

$X_i : (x_{i1}, x_{i2}, \cdots, x_{im}), m-dimensional\ vector$

$X : \{X_i \mid i = 1, 2, \cdots, n\}$

$E(X) = 0$

Project ***X***

$Try\ to\ find\ unit\ vector\ \mathbf{q}, \|\mathbf{q}\| = (\mathbf{q}^T\mathbf{q})^{1/2} = 1:$

$Projected\ X\ to\ \mathbf{q} : A = X^T q$

$$\boxed{\begin{aligned} &E(X) = 0 \Rightarrow E(A) = 0 \\ &\Rightarrow \sigma^2 = E(A^2) = E(q^T X X^T q) = q^T E(XX^T) q = q^T R_{XX} q \end{aligned}}$$

$The\ variance\ \sigma^2\ of\ the\ projection\ A\ is\ a\ function\ of\ the\ unit\ vector\ q:$

$$Variance\ Probe : \varphi_A(q) = \sigma = q^T R_{XX} q$$

Principal Components Analysis

- Eigenstructure of Principal-Components Analysis

  - Finding those unit vectors **q** along $\varphi_A(q)$ which has extremal or stationary values

$$\varphi_A(q+\Delta q)=\varphi_A(q), s.t. \|q+\Delta q\|=1 \Rightarrow \Delta q^T q=0$$

$$\begin{aligned}\varphi_A(q+\Delta q) &= (q+\Delta q)^T R_{XX}(q+\Delta q)\\ &= q^T R_{XX} q + q^T R_{XX}\Delta q + \Delta q^T R_{XX} q + \Delta q^T R_{XX}\Delta q\\ &= q^T R_{XX} q + 2\Delta q^T R_{XX} q + \Delta q^T R_{XX}\Delta q \quad \leftarrow \quad Note: a^T R_{XX} b = b^T R_{XX} a\end{aligned}$$

$$\Delta q^T R_{XX}\Delta q = 0 \quad \Rightarrow \quad \Delta q^T R_{XX} q = 0$$

$$\Delta q^T R_{XX} q - \lambda \Delta q^T q = 0$$

$$\Delta q^T\left(R_{XX} q - \lambda q\right) = 0$$

$$R_{XX} q = \lambda q$$

$$\boxed{\begin{aligned}&eigenvalue\ : \lambda_1 > \lambda_2 > \cdots > \lambda_m\\ &eigenvector: q_1, q_2, \cdots, q_m\\ &R_{XX} q_i = \lambda q_i\end{aligned}}$$

$$\boxed{\begin{aligned}&Q=(q_1, q_2, \cdots, q_m)\\ &R_{XX} Q = Q\Lambda\\ &\Lambda = diag(\lambda_1, \lambda_2, \cdots, \lambda_m)\\ &Note: Q^T Q = E \Rightarrow Q^T = Q^{-1}\end{aligned}}$$

Principal Components Analysis

- Eigenstructure of Principal-Components Analysis

$$Q=(q_1,q_2,\cdots,q_m)$$
$$R_{XX}Q=Q\Lambda$$
$$\Lambda=diag(\lambda_1,\lambda_2,\cdots,\lambda_m)$$
$$Note: Q^TQ=E\Rightarrow Q^T=Q^{-1}$$

$$\Rightarrow Q^{-1}R_{XX}Q=\Lambda \Rightarrow \boxed{Q^TR_{XX}Q=\Lambda} \Rightarrow \boxed{Q^TR_{XX}Q=Q\Lambda Q^T}$$

Spectral theorem

- The eigenvectors of the correlation matrix $\mathbf{R}$ pertaining to the zero-mean random vector $\mathbf{X}$ define the unit vectors $\mathbf{q}_j$, representing the principal directions along which the variance probes $\psi(\mathbf{q}_j)$ have their extremal values.
- The associated eigenvalues define the extremal values of the variance probes $\psi(\mathbf{u}_j)$.

➢ The projection of X onto the principal directions represented by the unit vector

$$principal\ components: a_j=q_j^TX=X^Tq_j$$

$$\begin{aligned}\mathbf{a} &= [a_1, a_2, \ldots, a_m]^T \\ &= [\mathbf{x}^T\mathbf{q}_1, \mathbf{x}^T\mathbf{q}_2, \ldots, \mathbf{x}^T\mathbf{q}_m]^T \\ &= \mathbf{Q}^T\mathbf{x}\end{aligned} \quad \Leftrightarrow \quad \begin{aligned}\mathbf{x} &= \mathbf{Q}\mathbf{a} \\ &= \sum_{j=1}^{m} a_j\mathbf{q}_j\end{aligned}$$

Principal Components Analysis

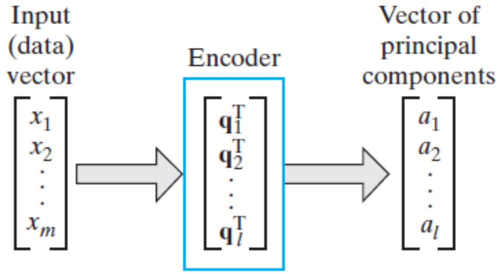
- Dimensionality Reduction

searches for directions in the data that have largest variance and subsequently project the data onto it.

In this way, obtain a lower dimensional representation of the data, that removes some of the noisy directions

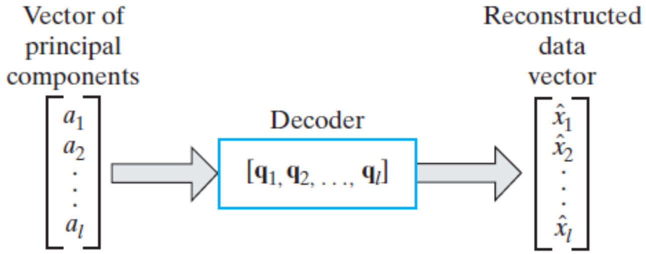
$$\hat{\mathbf{x}} = \sum_{j=1}^{l} a_j \mathbf{q}_j$$

$$= [\mathbf{q}_1, \mathbf{q}_1, \ldots, \mathbf{q}_j] \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_l \end{bmatrix}, \qquad l \le m$$



(a)

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_l \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_l^T \end{bmatrix} \mathbf{x}, \qquad l \le m$$



➢ ***In the sense of least squares, search the low dimensional subspaces of the data sets, and build up a new frame of axes***

Principal Components Analysis

- PCA Process

Step-1: Computer the Covariance matrix:

$$R_{XX} = XX^T \quad or:$$

$$R_{XX} = \frac{1}{n}\sum_{i=1}^{n}(X_i - u)(X_i - u)^T$$

Step-2: Eigenvalue decomposition

$$R_{XX}q = \lambda q$$

Step-3: Extract main directions, construct projection matrix using eigenvectors

$$Q = (q_1, q_2, \cdots, q_l)$$

➢ **The projection matrix Q is constructed by taking the characteristic vectors corresponding to the largest L eigenvalues, and the low dimensional features can be obtained by computational projection**

Principal Components Analysis

- Kernel PCA

Un-separable linearly

Low dimensional → High dimensional
Input space → feature space

PCA under feature space

$$X:(x_1,x_2,\cdots,x_{m_0})\in\mathbb{R}^{m_0}$$

m1>>m0

$$\phi_i(X):\ \ \{\varphi_i(X)\,|\,i=1,2,\cdots,m_1\}:\ \ \{\varphi_1(X),\varphi_2(X),\varphi_3(X),\cdots,\varphi_{m_1}(X)\}\in\mathbb{R}^{m1}$$

$$R_{XX}=XX^T$$

$$R_{\Phi\Phi}=\Phi(X)\Phi(X)^T=K(X,X)$$

Multidimensional scaling (MDS)

- Introduction

$$X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m0} \\ x_{21} & x_{22} & \cdots & x_{2m0} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm0} \end{pmatrix} \in R^{n \times m0}$$

$$\Downarrow PCA$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1m1} \\ y_{21} & y_{22} & \cdots & y_{2m1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm1} \end{pmatrix} \in R^{n \times m1} \quad (m1 \le m0)$$

A proximity matrix **D**=[d_{ij}] represents the "distance" (similarity/dissimilarity) between rows or columns of **X**

- **Q**: Suppose that **X** is not observed. Given **D**, how do we find **X** (or **Y**)?

- Note: a proximity matrix is invariant to (1) change in location, (2) rotation, (3) reflections
  $\Rightarrow$ cannot expect to recover **X** completely

Multidimensional scaling (MDS)

- Principle of classical multidimensional scaling

➢ assume that the *observed* $n \times n$ proximity matrix **D** is a matrix of Euclidean distances derived from a raw $n \times m0$ data matrix, **X**, which is *not observed*.

➢ define an $n \times n$ matrix **B** $\quad B = (XM)(XM)^T = (XM)(M^T X^T) \quad M$: an orthogonal matrix

- the elements of **B** are given by: $b_{ij} = \sum_{b=1}^{m1} x_{ik} x_{jk}$

➢ the squared Euclidean distances between the rows of **X** can be written in terms of the elements of **B** as:

$$d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}$$

- idea: If b_{ij} could be found in terms of d_{ij} in the equation above, then we can derive **X** from **B** by factoring **B**
- to obtain **B** from **D**, no unique solution exists unless a location constraint is introduced. Usually, the center of the columns of **X** are set at origin, i.e., $\sum_{i=1}^{n} x_{ik} = 0, \quad for\ all\ k$
- these constraints imply that sum of the terms in any row of **B** must be 0, i.e.,

$$\sum_{j=1}^{n} b_{ij} = \sum_{j=1}^{n} \sum_{k=1}^{q} x_{ik} x_{jk} = \sum_{k=1}^{q} x_{ik} \left(\sum_{j=1}^{n} x_{jk} \right)$$

Multidimensional scaling (MDS)

- Principle of classic MDS

➢ Let T be the trace of **B**. To obtain **B** from **D**, notice that

$$\sum_{i=1}^{n} d_{ij}^2 = T + nb_{jj} \qquad \sum_{j=1} d_{ij}^2 = nb_{ii} + T$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}^2 = 2nT$$

➢ the elements of **B** can be found from **D** as $b_{ij} = -\frac{1}{2}\left(d_{ij}^2 - d_{\bullet j}^2 - d_{i\bullet}^2 + d_{\bullet\bullet}^2\right)$

where $d_{i\cdot}^2 = (\sum_{j=1}^{n} d_{ij}^2)/n$, $d_{\cdot j}^2 = (\sum_{i=1}^{n} d_{ij}^2)/n$, $d_{\cdot\cdot}^2 = (\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}^2)/n^2$

➢ **B** can be written as

$$\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}',$$

where $\mathbf{\Lambda} = diag[\lambda_1, \cdots, \lambda_n]$ $(\lambda_1 \geq \cdots \geq \lambda_n)$ is the diagonal matrix of eigenvalues of **B** and $\mathbf{V} = [\mathbf{V_1}, \cdots, \mathbf{V_n}]$ is the corresponding matrix of normalized eigenvectors (i.e., $\mathbf{V_i}'\mathbf{V_i} = 1$)

- Note: when **D** arises from an $n \times m1$ data matrix, the rank of **B** is $m1$ (i.e, the last n-$m1$ eigenvalues should be zero)
- So, **B** can be chosen as

$$\mathbf{B} = \mathbf{V}^*\mathbf{\Lambda}^*\mathbf{V}^{*\prime},$$

where $\mathbf{V}^*$ contains the first q eigenvectors and $\mathbf{\Lambda}^*$ the first q eigenvalues

- Thus, a solution of **X** is $\mathbf{X} = \mathbf{V}^*\mathbf{\Lambda}^{*1/2}$

Multidimensional scaling (MDS)

- Example

Linear Discriminant Analysis (LDA)

- LDA for 2 classes

➢ **The objective of LDA is to perform dimensionality reduction while preserving as much of the class discriminatory information as possible**

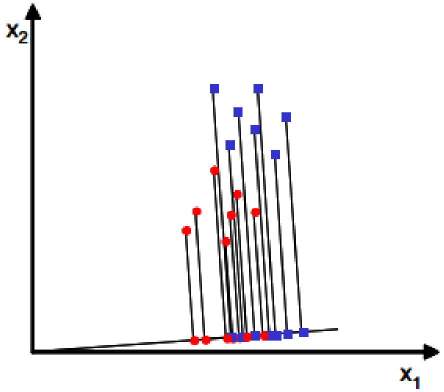
$Training\ set: X:\{X_1, X_2, X_3, \cdots, X_n\}\ with\ m-dimensional\ \ for\ each\ sample\ X_i:\{x_1, x_2, x_3, \cdots, x_m\}$

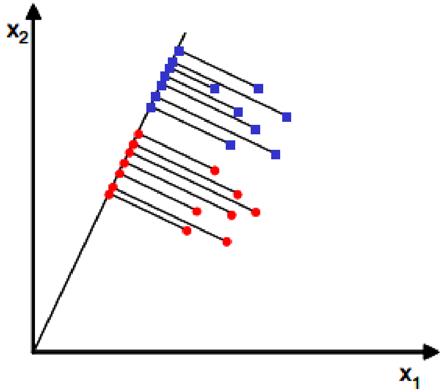
$Number\ \ for\ class-1: n_1 \quad Number\ \ for\ class-2: n_2 \leftarrow n_1 + n_2 = n$

- We seek to obtain a scalar y by projecting the samples x onto a line

$$y = W^T X$$

- Of all the possible lines we would like to select the one that maximizes the separability of the scalars





Linear Discriminant Analysis (LDA)

- LDA for 2 classes

➢ **In order to find a good projection vector, need define a measure of separation between projections**

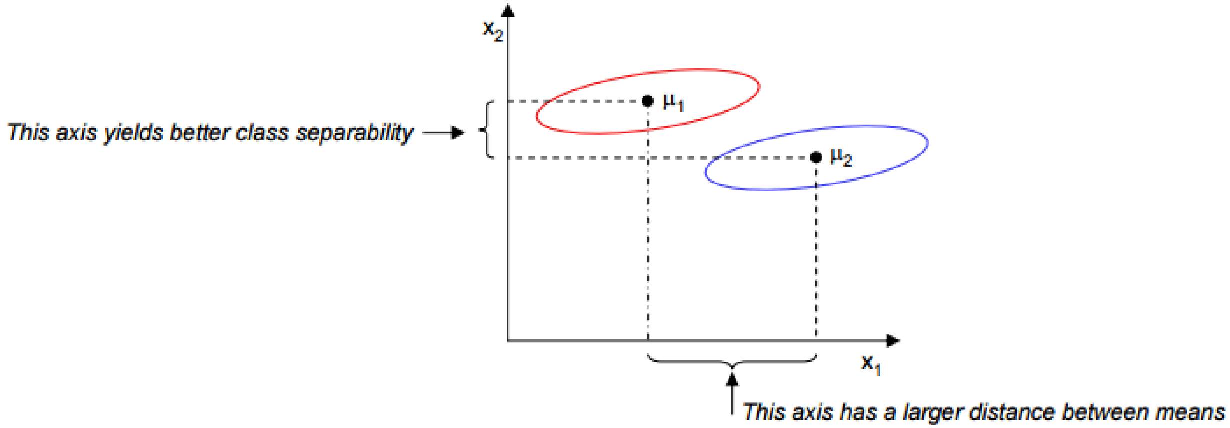
- The mean vector of each class in x and y feature space is

$$u_i = \frac{1}{n_i} \sum_{X \in \omega_1} X \quad i = 1,2 \quad u_{y_i} = \frac{1}{n_i} \sum_{y \in \omega_1} y \quad i = 1,2$$

- choose the distance between the projected means as our objective function

$$L(W) = \left| u_{y_0} - u_{y_1} \right| = \left| W^T u_0 - W^T u_1 \right| = \left| W^T (u_0 - u_1) \right|$$

- ***However***, the distance between the projected means is not a very good measure since it does not take into account the standard deviation within the classes, expect that same-class data are as close as possible.



Linear Discriminant Analysis (LDA)

- LDA for 2 classes

➢ **The solution proposed by Fisher is to maximize a function that represents the difference between the means, normalized by a measure of the within-class scatter**

- For each class we define the scatter, an equivalent of the variance, as

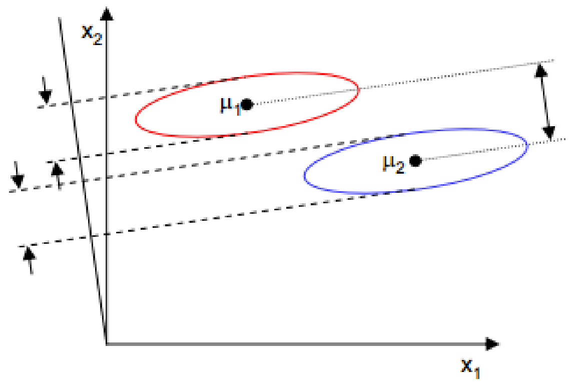
$$S_{y_i} = \sum_{y \in \omega_1} \left(y - u_{y_i} \right)^2 \quad i = 1,2$$

  - Within-class scatter(variance) of the projected samples : $S_{y_0} + S_{y_1}$

- The Fisher linear discriminant is defined as the linear function $W^T X$ that maximizes the criterion function

$$L(W) = \frac{\left| u_{y_0} - u_{y_1} \right|^2}{S_{y_0} + S_{y_1}}$$

- **Therefore, *Target* is: looking for a projection where samples from the same class are projected very close to each other and, the projected means are as farther apart as possible**



Linear Discriminant Analysis (LDA)

- LDA for 2 classes
  - define a measure of the scatter in multivariate feature space

$$S_i = \sum_{X \in \omega_1} (X - u_i)^2 \quad i = 1,2 \quad \textit{scatter matrix under input space}$$

$$\textit{within} - \textit{class scatter matrix} : S_w = S_1 + S_2$$

  - The scatter of the projection y can then be expressed as a function of the scatter matrix in original feature space x

$$S_{y_i} = \sum_{y \in \omega_1} \left(y - u_{y_i} \right)^2 = \sum_{y \in \omega_1} \left(W^T X - W^T u_i \right)^2 \sum_{y \in \omega_1} W^T \left(X - u_i \right) \left(X - u_i \right)^T W = W^T S_i W$$

$$S_{y_0} + S_{y_1} = W^T S_w W$$

  - Similarly, the difference between the projected means can be expressed in terms of the means in the original feature space

$$\left(u_{y_1} - u_{y_2} \right)^2 = \left(W^T u_1 - W^T u_2 \right)^2 = W^T \left(u_1 - u_2 \right) \left(u_1 - u_2 \right)^T W = W^T S_B W$$

Between-class scatter:
Rank: at most 1

  - **finally express the Fisher criterion in terms of SW and SB as**

$$\boxed{L(W) = \frac{W^T S_B W}{W^T S_W W}}$$

Linear Discriminant Analysis (LDA)

- LDA for 2 classes
  - To find the maximum of L(w) using Lagrange, and constraint $\left|W^T S_w W\right| = 1$

$$L(W) = W^T S_B W - \lambda \left[W^T S_W W - 1 \right]$$
$$\frac{\partial L(W)}{\partial W} = 2 S_B W - 2\lambda S_W W = 0$$
$$\Rightarrow S_W^{-1} S_B W = \lambda W$$

  - Solving the generalized eigenvalue problem (SW-1SBw=Jw) yields

$$W = \arg\max \{ \frac{W^T S_B W}{W^T S_W W} \} = S_W^{-1} (u_1 - u_2)$$

Fisher Linear Discriminant

Linear Discriminant Analysis (LDA)

- LDA for 2 classes-Example

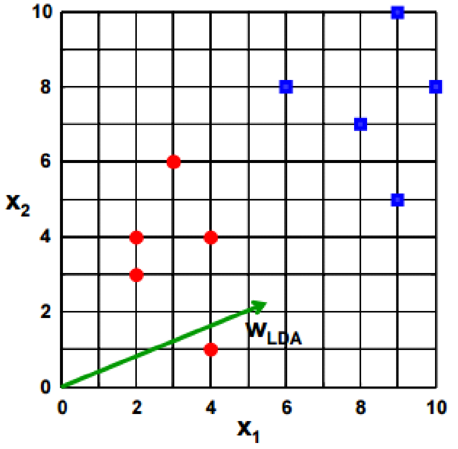
  - **Compute the Linear Discriminant projection for the following two-dimensional dataset**
    - $X1=(x_1,x_2)=\{(4,1),(2,4),(2,3),(3,6),(4,4)\}$
    - $X2=(x_1,x_2)=\{(9,10),(6,8),(9,5),(8,7),(10,8)\}$
  - **SOLUTION (by hand)**
    - The class statistics are:

$$S_1=\begin{bmatrix}0.80 & -0.40\\ -0.40 & 2.60\end{bmatrix};\ S_2=\begin{bmatrix}1.84 & -0.04\\ -0.04 & 2.64\end{bmatrix}$$

$$\mu_1=[3.00\quad 3.60];\qquad \mu_2=[8.40\quad 7.60]$$

    - The within- and between-class scatter are

$$S_B=\begin{bmatrix}29.16 & 21.60\\ 21.60 & 16.00\end{bmatrix};\ S_W=\begin{bmatrix}2.64 & -0.44\\ -0.44 & 5.28\end{bmatrix}$$



    - The LDA projection is then obtained as the solution of the generalized eigenvalue problem

$$S_W^{-1}S_Bv=\lambda v\Rightarrow\left|S_W^{-1}S_B-\lambda I\right|=0\Rightarrow\begin{vmatrix}11.89-\lambda & 8.81\\ 5.08 & 3.76-\lambda\end{vmatrix}=0\Rightarrow\lambda=15.65$$

$$\begin{bmatrix}11.89 & 8.81\\ 5.08 & 3.76\end{bmatrix}\begin{bmatrix}v_1\\ v_2\end{bmatrix}=15.65\begin{bmatrix}v_1\\ v_2\end{bmatrix}\Rightarrow\begin{bmatrix}v_1\\ v_2\end{bmatrix}=\begin{bmatrix}0.91\\ 0.39\end{bmatrix}$$

    - Or directly by

$$w^*=S_W^{-1}(\mu_1-\mu_2)=[-0.91\quad -0.39]^T$$

Linear Discriminant Analysis (LDA)

- LDA for C classes

➢ **Fisher LDA generalizes very gracefully for C-class problems**

- Instead of one projection y, we will now seek (C-1) projections [y1,y2,···,yC-1] by means of (C-1) projection vectors wi, which can be arranged by columns into a projection matrix W=[w1|w2|···|wC-1]:

$$y = W^T X$$

➢ **Derivation**

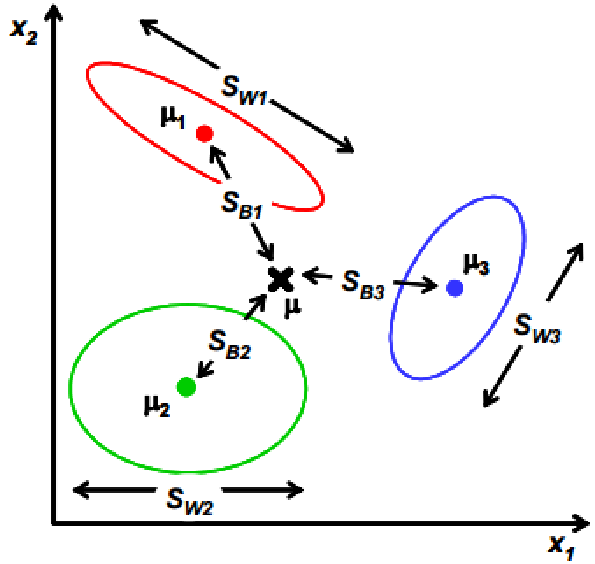
- The generalization of the within-class scatter is

$$S_w = \sum_{i=1}^{C} S_i \qquad where\ S_i = \sum_{X \in \omega_I} (X - u_i)(X - u_i)^T, and\ u_i = \frac{1}{n_i} \sum_{X \in \omega_I} X$$

- The generalization for the between-class scatter is

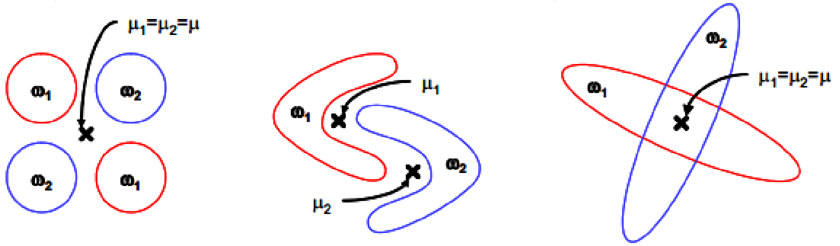
$$S_B = \sum_{i=1}^{C} n_i (u_i - u)(u_i - u)^T, \quad where\ u = \frac{1}{n} \sum_{\forall X} X = \frac{1}{n} \sum_{\forall X} n_i u_i$$

$$W = \arg\max\{\frac{W^T S_B W}{W^T S_W W}\} = \max\left(eigenvector\ matched\ with\ eigenvalue\ of\ S_W^{-1} S_B\right)$$

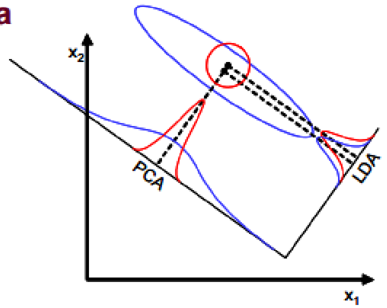


Linear Discriminant Analysis (LDA)

- Limitation of LDA
  - **LDA produces at most C-1 feature projections**
    - If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features
  - **LDA is a parametric method since it assumes unimodal Gaussian likelihoods**
    - If the distributions are significantly non-Gaussian, the LDA projections will not be able to preserve any complex structure of the data, which may be needed for classification



  - **LDA will fail when the discriminatory information is not in the mean but rather in the variance of the data**



Laplacian Eigenmaps (LE)

- Introduction

➢ *Question:* **How to reconstruct data points based on the similarity of given samples?**

➢ ***The basic idea:***

- **project the data points into low-dimensional Euclidean space, as much as possible to maintain the similarity between data points before and after projection**
- **so that similar data points are adjacent to each other after projection, and non-similar data points are distant from each other after projection**

➢ **Undirected weighted graph:**

- Graph G, vertex set: V(G), Edge set: E(G), Adjacency Matrix: W (edge weights)

$$w_{ij} = w_{ji} = \begin{cases} > 0 & if\ vertex\ i\ is\ connected\ with\ vertex\ j \\ = 0 & if\ vertex\ i\ is\ NOT\ connected\ with\ vertex\ j \end{cases}$$

- **Degree Matrix D(dij): The degree of vertex i is defined as the sum of the weights of all edges connected to it**

$$d_i = \sum_{j=1}^{n} w_{ij} \qquad D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix}$$

Laplacian Eigenmaps (LE)

- How to construct adjacency matrix – Similar matrix

➢ ***The basic idea:***

- The weight value of the edge between the two points far away is lower
- The weight value of the edge between the two points closer to the distance is higher

➢ **Method to construct similar matrix**

$1.\ \varepsilon - Nearest\ Neighbors$

$$w_{ij} = \begin{cases} 0 & |x_i - x_j|^2 > \varepsilon \\ \varepsilon & |x_i - x_j|^2 \le \varepsilon \end{cases}$$

$problem: not\ precise$

$2.\ K - Nearest\ Neighbors$

$$w_{ij} = w_{ji} = \begin{cases} 0 & x_i \notin KNN(x_j)\ and\ x_j \notin KNN(x_i) \\ e^{-\frac{\|x_i - x_j\|}{2\sigma^2}} & x_i \in KNN(x_j)\ or\ x_j \in KNN(x_i) \\ & x_i \in KNN(x_j)\ and\ x_j \in KNN(x_i) \end{cases}$$

$problem: W\ will\ be\ non-symetric$

$3.\ Complete\ connection:$

$$w_{ij} = w_{ji} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

Laplacian Eigenmaps (LE)

- Graph Laplacian - Laplacian matrix

➢ ***Definiation:***

$$L = D - W$$

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix} \qquad d_i = \sum_{j=1}^{n} w_{ij} \qquad w_{ij} = w_{ji} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

➢ **Laplacian matrix properties:**

- L is symmetric matrix, Its all eigenvalue are real number
- For all vectors f:

$$\begin{aligned} f^T L f = f^T D f - f^T W f &= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} w_{ij} f_i f_j \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{j=1}^{n} d_j f_j^2 \right) \\ &= \frac{1}{2}\sum_{i,j=1}^{n} w_{ij} \left(f_i - f_j \right)^2 \end{aligned}$$

- Eigenvalue: $0 = \lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$

Laplacian Eigenmaps (LE)

- Objective function and solution

➢ ***Definiation:***

$$X_i : (x_{i1}, x_{i2}, \cdots, x_{im0}) \in \mathbb{R}^{m0} \quad \rightarrow \quad y_i : (y_{i1}, y_{i2}, \cdots, y_{im1}) \in \mathbb{R}^{m1}, where\ m1 < m0$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

$$W(w_{ij}) \qquad \Rightarrow \qquad \min\left(\sum_{i,j}^{n} w_{ij}\left(y_i - y_j\right)^2\right) = \min\left(Y^T L Y\right)$$

➢ **Solution:**

$$\min\left(Y^T L Y\right) = \min_{Y^T DY = E}\left(Y^T L Y\right)$$

$$L(Y) = Y^T L Y - \lambda\left(Y^T D Y - E\right)$$

$$\frac{\partial L(Y)}{\partial Y} = 2LY - 2\lambda DY = 0$$

$$\Rightarrow LY = \lambda DY \in Generalized\ eigenvalue\ problem$$

$$\lambda_1 \le \lambda_2 \le \cdots \lambda_{m1} \le \cdots \le \lambda_n$$

$$\underbrace{Y_1 \quad Y_2 \cdots\cdots Y_{m1}}_{Solution} \cdots\cdots\cdots Y_n$$

Laplacian Eigenmaps (LE)

- Example

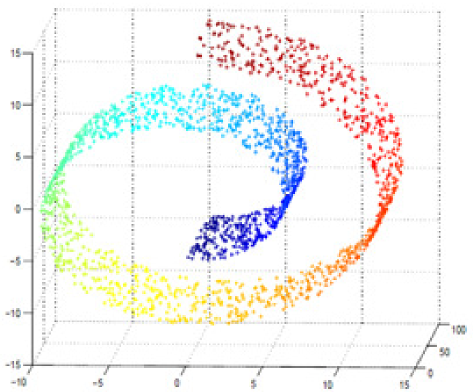
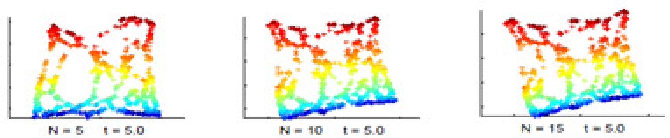
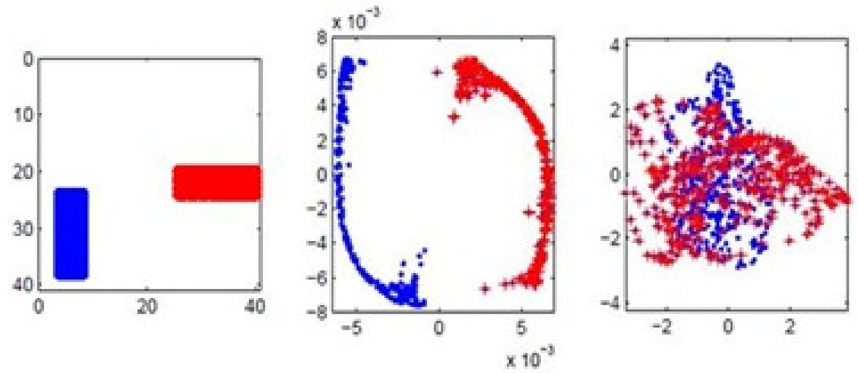


Figure 1: 2000 random data points on the "swiss roll".



LE: 3D → 2D



LE VS PCA

Spectral clustering

- Introduction

➢ **Target: Given data points X1, ⋯, Xn, and Given data points X1, ⋯, Xn and similarities w(Xi,Xj), partition the data into groups so that points in a group are similar and points in different groups are dissimilar.**

➢ **Basic idea:**

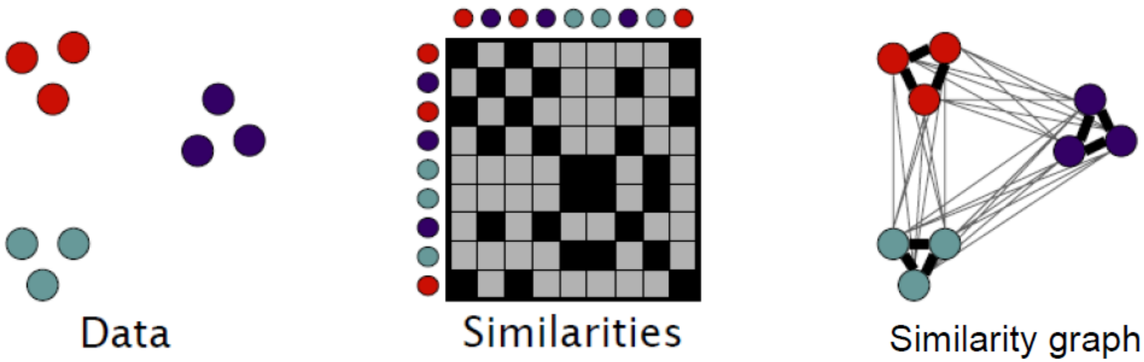
- Partition the graph so that edges within a group have large weights and edges across groups have small weights
- Give subsets A and B, whose vertex are disjoint, define the cutting between A and B is:

$$cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$$

Target: find smallest graph cutting

Similarity Graph: G(V,E,W)

V – Vertices (Data points)
E – Edge if similarity > 0
W - Edge weights (similarities)



Partition the graph so that edges within a group have large weights and edges across groups have small weights.

Spectral clustering

- Undirected graph cutting

➢ Undirected graph G cutting:

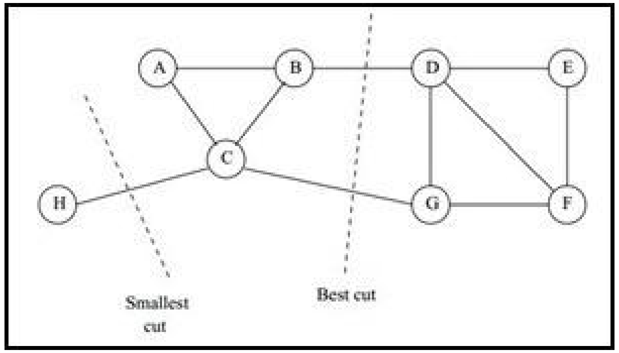
$$k\ sub-graph: A_1, A_2, \cdots, A_k$$
$$s.t. \quad A_i \cap A_j = \phi$$
$$A_1 \cup A_2 \cup \cdots \cup A_k = Vertex(G)$$

➢ The weights between set A and B:

$$W(A,B) = \sum_{i \in A, j \in B} w_{ij}$$

➢ Minimize cutting (**MinCut**):

$$MinCut(A_1, A_2, \cdots, A_k) = \min\left(\frac{1}{2}\sum_{i=1}^{k} W(A_i, \bar{A}_i)\right)$$



基本步骤 (MinCut):

- 1. 定义相似度矩阵W
- 2. 由W构造图Laplacian矩阵
  - **L = D – W ，D为对角阵, W要求对称**
- 3. 对L进行奇异值分解[U, S, V] = svd(L)
  - **其中特征值按由小到大排列**
- 4. 通过零特征值数目估计聚类的个数k，使用k个最小特征值对应的k个右奇异值向量V作为新的特征，运行k-means

un-optimal: Select the smallest weight edge to cut the graph

Spectral clustering

- Undirected graph cutting

➢ **RatioCut:**

- **MinCut only consider the smallest weights between the cutting graphs. RatioCut consider the maximum vetex numbers in a group.**

$$RatioCut\left(A_1, A_2, \cdots, A_k\right) = \min\left(\frac{1}{2}\sum_{i=1}^{k}\frac{W(A_i, \bar{A}_i)}{|A_i|}\right)$$

$$Matrix\ H: \{H_1, H_2, \cdots, H_k\}, \quad k\ vectors$$

$$Vector\ H_i: \{h_{i1}, h_{i2}, \cdots, h_{in}\}, \quad n: training\ set\ size$$

$$h_{ij} = \begin{cases} 0 & vetex\ j\ v_j \notin A_j \\ \frac{1}{\sqrt{|A_i|}} & vetex\ j\ v_j \in A_j \end{cases}$$

$$\begin{aligned} H_i^T L H_i &= \frac{1}{2}\sum_{m=1}^{k}\sum_{n=1}^{k} w_{mn}\left(h_{im} - h_{in}\right)^2 \\ &= \frac{1}{2}\left(\sum_{m\in A_i, n\notin A_i} w_{mn}\left(\frac{1}{\sqrt{|A_i|}} - 0\right)^2 + \sum_{m\notin A_i, n\in A_i} w_{mn}\left(0 - \frac{1}{\sqrt{|A_i|}}\right)^2\right) \\ &= \frac{1}{2}\left(\sum_{m\in A_i, n\notin A_i} w_{mn}\frac{1}{|A_i|} + \sum_{m\notin A_i, n\in A_i} w_{mn}\frac{1}{|A_i|}\right) \\ &= \frac{1}{2}\left(cut(A_i, \bar{A}_i)\frac{1}{|A_i|} + cut(\bar{A}_i, A_i)\frac{1}{|A_i|}\right) \\ &= \frac{cut(A_i, \bar{A}_i)}{|A_i|} \end{aligned}$$

$$RatioCut\left(A_1, A_2, \cdots, A_k\right) = \min\left(H_i^T L H_i\right) \quad s.t.\ H^T H = 1$$

the smallest K eigenvalues of matrix L
the corresponding K eigenvectors

Spectral clustering

- Undirected graph cutting

➢ **NormalizedCut (NCut):**

$$NCut(A_1, A_2, \cdots, A_k) = \min\left(\frac{1}{2}\sum_{i=1}^{k}\frac{W(A_i, \bar{A}_i)}{vol(A_i)}\right)$$

$$Matrix\ H : \{H_1, H_2, \cdots, H_k\}, \quad k\ vectors$$

$$Vector\ H_i : \{h_{i1}, h_{i2}, \cdots, h_{in}\}, \quad n : training\ set\ size$$

$$h_{ij} = \begin{cases} 0 & vetex\ j\ v_j \notin A_j \\ \dfrac{1}{\sqrt{vol(A_i)}} & vetex\ j\ v_j \in A_j \end{cases}$$

$$\begin{aligned} H_i^T L H_i &= \frac{1}{2}\sum_{m=1}^{k}\sum_{n=1}^{k} w_{mn}(h_{im} - h_{in})^2 \\ &= \frac{1}{2}\left(\sum_{m\in A_i, n\notin A_i} w_{mn}\left(\frac{1}{\sqrt{vol(A_i)}} - 0\right)^2 + \sum_{m\notin A_i, n\in A_i} w_{mn}\left(0 - \frac{1}{\sqrt{vol(A_i)}}\right)^2\right) \\ &= \frac{1}{2}\left(\sum_{m\in A_i, n\notin A_i} w_{mn}\frac{1}{vol(A_i)} + \sum_{m\notin A_i, n\in A_i} w_{mn}\frac{1}{vol(A_i)}\right) \\ &= \frac{1}{2}\left(cut(A_i, \bar{A}_i)\frac{1}{vol(A_i)} + cut(\bar{A}_i, A_i)\frac{1}{vol(A_i)}\right) \\ &= \frac{cut(A_i, \bar{A}_i)}{vol(A_i)} \end{aligned}$$

$$RatioCut(A_1, A_2, \cdots, A_k) = \min\left(H_i^T L H_i\right) \quad s.t.\ H^T D H = 1$$

$$H^T D H = \sum_{i=1}^{n} h_{ij}^2 d_j = \frac{1}{vol(A_i)}\sum_{v_j\in A_i} w_{v_j} = \frac{1}{vol(A_i)} vol(A_i) = 1$$

the smallest K eigenvalues of matrix $D^{-1/2} L D^{-1/2}$
the corresponding K eigenvectors

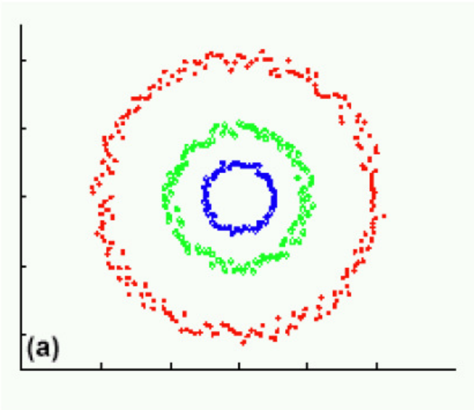
$$Normalized : D^{-1/2} L D^{-1/2} \Leftrightarrow \frac{L_{ij}}{\sqrt{vol(A_i)\, vol(A_j)}}$$

Spectral clustering

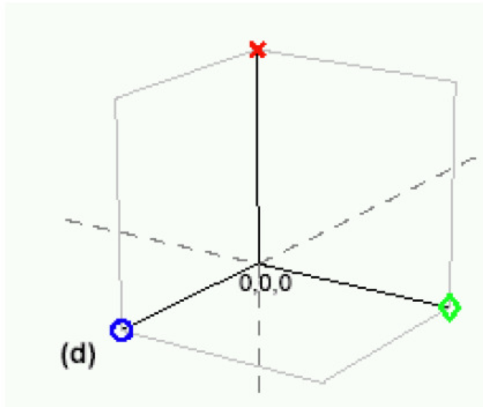
- Why does it work?

Data are projected into a lower-dimensional space (the spectral/eigenvector domain) where they are easily separable, say using k-means.

Original data



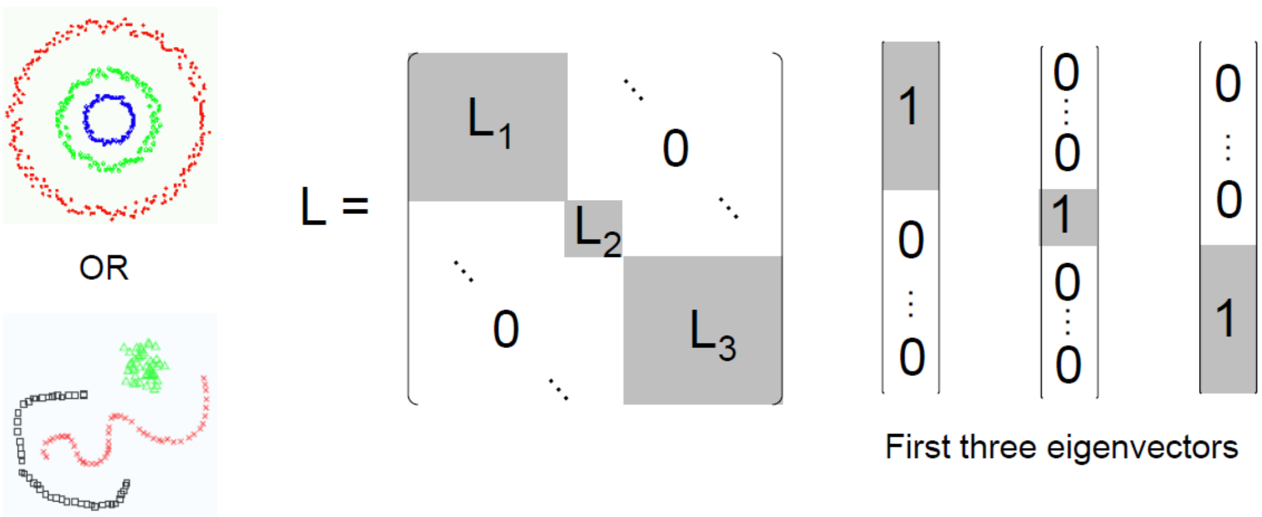
Projected data



Graph has 3 connected components – first three eigenvectors are constant (all ones) on each component.

Spectral clustering

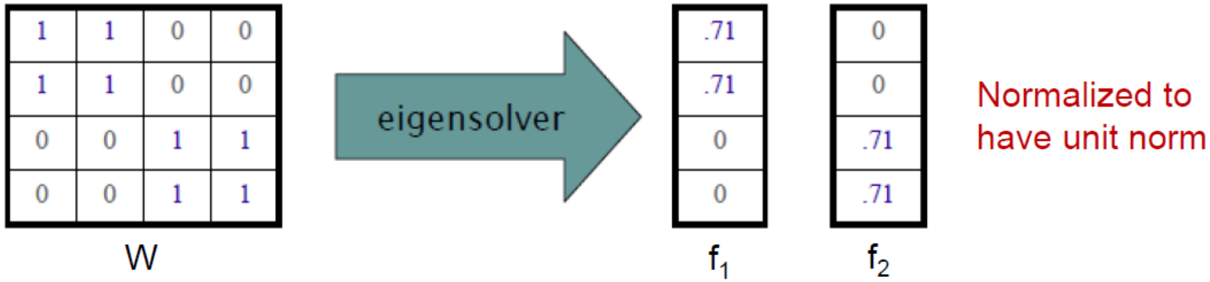
- Understanding Spectral clustering
  - If graph is connected, first Laplacian evec is constant (all 1s)
  - If graph is disconnected (k connected components), Laplacian is block diagonal and first k Laplacian evecs are:



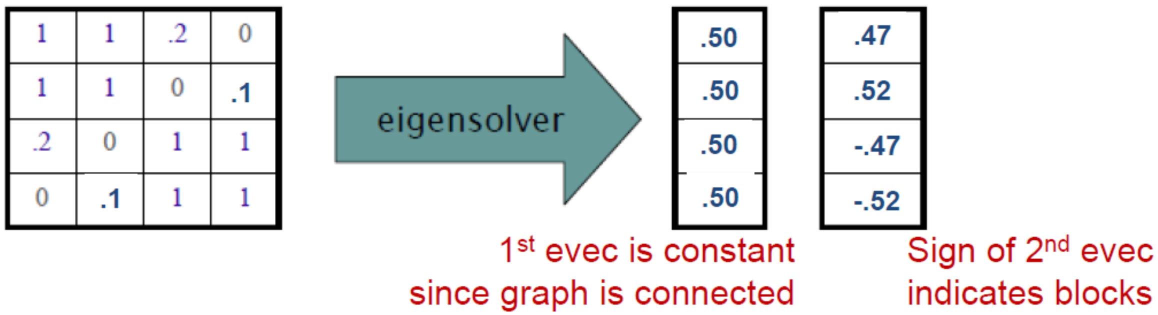
Spectral clustering

- Understanding Spectral clustering

Block weight matrix (disconnected graph) results in block eigenvectors:



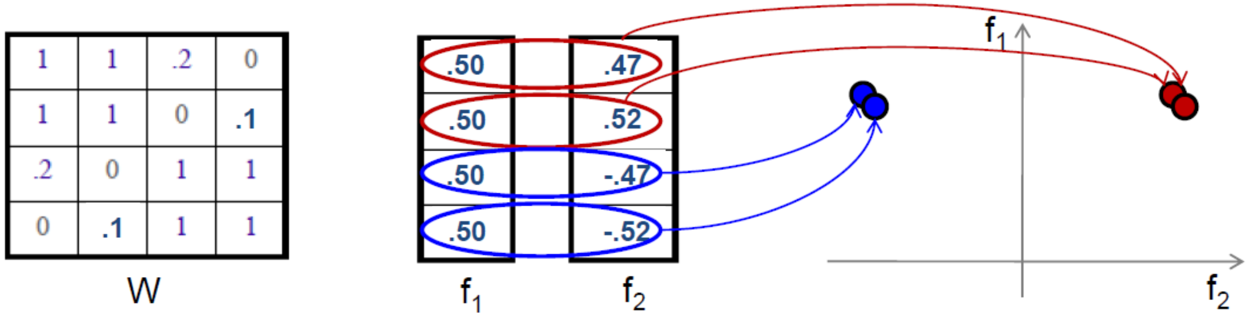
Slight perturbation does not change span of eigenvectors significantly:



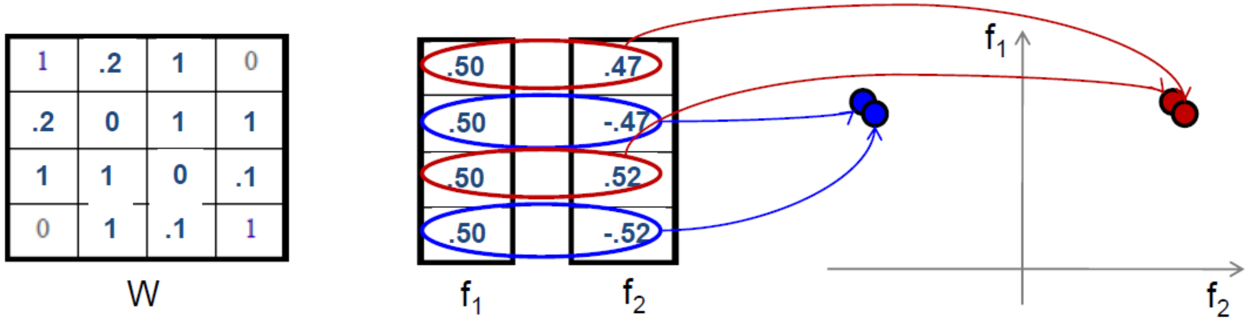
Spectral clustering

- Understanding Spectral clustering

Can put data points into blocks using eigenvectors:



Embedding is same regardless of data ordering:



This summary which I spend 1 week to work for, is the first stage during my learning ML. Well, It covers most algorithms I learned from April 2017, except some statistical algorithms such as EM, decision tree, Maximum entropy, and other Density estimation algorithms, and Local Linear Embedding (LLE) and ISOMAP. That is unfortunately since It will spend many time …… if I add these missing algorithms to this summary. I will continue to summarize these algorithms in future.

The most important things for me in ML area is to *PRACTICE*, *PRACTICE*!!!!

So next stage work for me is practice in computer vision using CNN!!!! My first idea is OCR!!!!

Let me start!