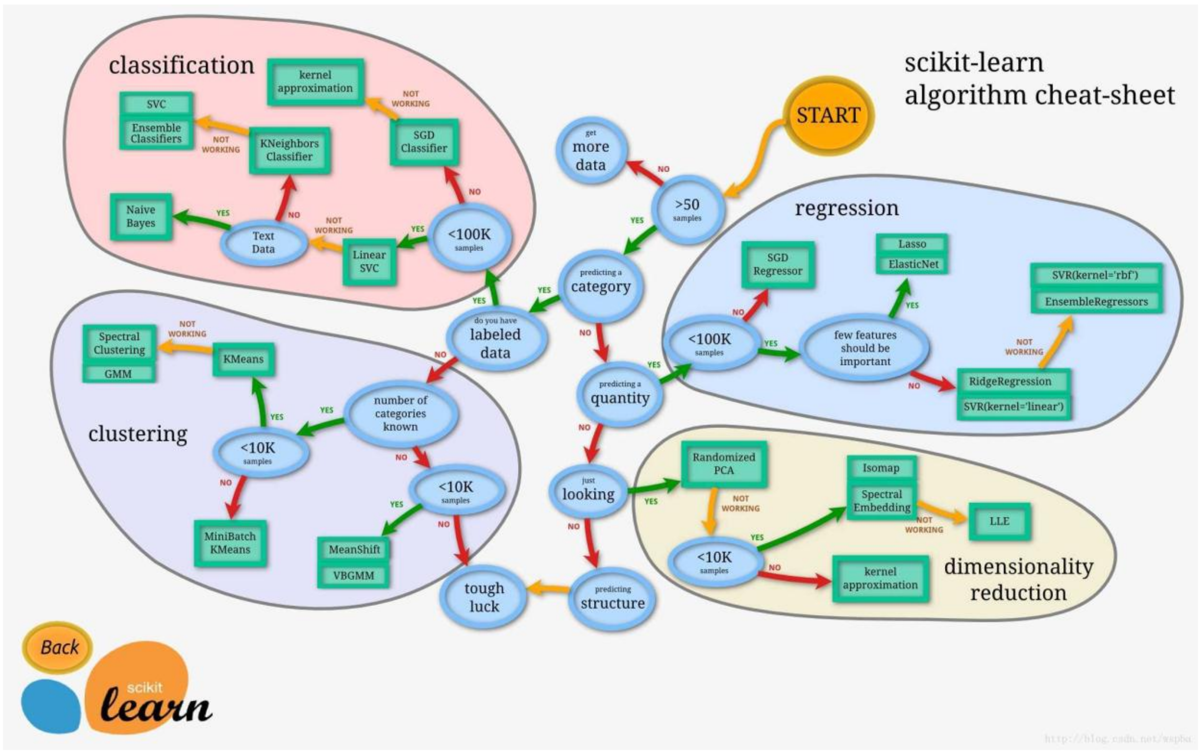
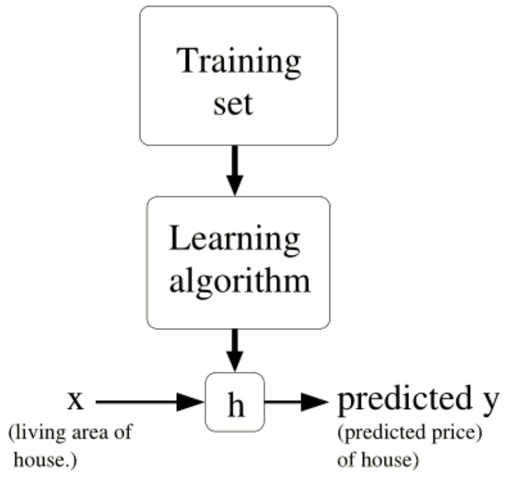
ML



Lesson 1

① Supervised Learning (SL)



Continuous → Regression Question (to predict)

Discrete → Classification Question (to 'chose' whether or not)

② Unsupervised Learning (UL)

No comparative sets or no certain answer → Clustering

③ Reinforcement Learning (RL)

'Good Dog' & 'Bad Dog' Question → Maximize 'Reward Signal' (decision making, like SLAM)

Lesson 2

① Optimization Question → least square method → ***Gradient Descent Algorithm***

② GDA associated

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta)$$

α called 'learning rate', need to set by experience because too large can lead to go over the minimum, accordingly, too small can lead to convergent slowly.

(1) Batch gradient descent (e.g. linear regression)

assume

$$h(x) = h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_i x_i + \cdots + \theta_n x_n$$

for conciseness, define $x_0 = 1$.

$h(x) = \sum_{i=0}^{n} \vartheta_i x_i = \boldsymbol{\theta}^T \boldsymbol{x}$ *(matrix type)*

$n = \#features$

purpose:

$$\min_\theta \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

(minimize the sum of squares of residuals)

$x^{(i)}$ $y^{(i)}$: i^{th} *of m training sets inputs&outputs*

→*structure* $J(\theta)$:

$$J(\theta) = \frac{1}{2} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$\underset{\theta}{minimize}\ J(\theta)$$

→→*adjust* θ to reduce J(θ) *(how to?)*:

for i^{th} *parameter* θ_i:

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta)$$

as linear assume:

$$\frac{\partial}{\partial\theta_i}J(\theta)=\sum_{j=1}^{m}\left(h_\theta\left(x^{(j)}\right)-y^{(j)}\right)\cdot\frac{\partial}{\partial\theta_i}\left(h_\theta\left(x^{(j)}\right)-y^{(j)}\right)$$

$$=\sum_{j=1}^{m}\left(h_\theta\left(x^{(j)}\right)-y^{(j)}\right)\cdot\frac{\partial}{\partial\theta_i}\left(x_0^{(j)}\theta_0+\cdots+x_i^{(j)}\theta_i+\cdots+x_n^{(j)}\theta_n-y^{(j)}\right)$$

$$=\sum_{j=1}^{m}\left(h_\theta\left(x^{(j)}\right)-y^{(j)}\right)\cdot x_i^{(j)}$$

$\therefore$ *Batch Gradient Descent called*

$repeat\ till\ convergence:$

$$\theta_i:=\theta_i-\alpha\cdot\sum_{j=1}^{m}\left(h_\theta\left(x^{(j)}\right)-y^{(j)}\right)\cdot x_i^{(j)}\ (for\ all\ i)$$

$(\boldsymbol{\theta}\ can\ be\ initialized\ as\ [0,0,\dots,0])$

(2) Stochastic gradient descent (suit large quantities of training sets)

```
repeat
    for $j = 1$ to $m$
        $\theta_i := \theta_i - \alpha \cdot \left(h_\theta\left(x^{(j)}\right) - y^{(j)}\right) \cdot x_i^{(j)}$ (for all i)
```

'accelerate' algorithm, local minimize feature not as good as (1), just swing around it.

(Q: How to check convergence? Any optimized solution?)

③ Normal Equation

GDA iteration complex → Matrix-related presentation (trace properties & equation deductions)

$\boldsymbol{\nabla}\ defination:$

$assume\ \boldsymbol{A}\in\mathcal{R}^{m\times n},\ f(\boldsymbol{A}):\mathcal{R}^{m\times n}\longmapsto\mathcal{R}$

$define$

$$\boldsymbol{\nabla}_{\boldsymbol{A}}f(\boldsymbol{A})=\begin{bmatrix}\frac{\partial f}{\partial a_{11}} & \cdots & \frac{\partial f}{\partial a_{1n}}\\ \vdots & \ddots & \vdots\\ \frac{\partial f}{\partial a_{m1}} & \cdots & \frac{\partial f}{\partial a_{mn}}\end{bmatrix}$$

$especially$

$$if\ \boldsymbol{A} \in \mathcal{R}^{n\times n},$$

$define$ $tr\ \boldsymbol{A}\big(or\ tr(\boldsymbol{A})\big) = \sum_{i=1}^{n} a_{ii}$

$some\ useful\ properties:$

① $tr\ \boldsymbol{AB} = tr\ \boldsymbol{BA}$

→ $tr\ \boldsymbol{ABC} = tr\ \boldsymbol{CAB} = tr\ \boldsymbol{BCA}$

② $tr\ \boldsymbol{A} = tr\ \boldsymbol{A}^T$

→ $tr\ a = tr\ a^T = a \qquad a \in \mathcal{R}$

③ $\boldsymbol{\nabla}_{\boldsymbol{A}} tr\ \boldsymbol{AB} = \boldsymbol{B}^T$

→ $\boldsymbol{\nabla}_{\boldsymbol{A}} tr\ \boldsymbol{B}^T\boldsymbol{A} = \boldsymbol{B},\ \boldsymbol{\nabla}_{\boldsymbol{A}} tr\ \boldsymbol{BA}^T = \boldsymbol{B}$

④ $\boldsymbol{\nabla}_{\boldsymbol{A}} tr\ \boldsymbol{ABA}^T\boldsymbol{C} = \boldsymbol{CAB} + \boldsymbol{C}^T\boldsymbol{AB}^T \quad (uv)' = u'v + uv'$

$*\ brief\ proof$

$\boldsymbol{\nabla}_{\boldsymbol{A}} tr\ \boldsymbol{ABA}^T\boldsymbol{C} = \boldsymbol{\nabla}_{\boldsymbol{A}} tr\ \boldsymbol{ABA}^T\boldsymbol{C}|to\ \boldsymbol{A} + \boldsymbol{\nabla}_{\boldsymbol{A}} tr\ \boldsymbol{CABA}^T|to\ \boldsymbol{A}^T = \boldsymbol{C}^T\boldsymbol{AB}^T + \boldsymbol{CAB}$

try to concise iteration by upwards definition & properties

$define$

$$\boldsymbol{x}(\vec{x}) = \begin{bmatrix} x_0 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} \qquad \boldsymbol{\Theta}(\vec{\theta}) = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_i \\ \vdots \\ \theta_n \end{bmatrix} \qquad n = \#features/inputs$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(1)T} \\ \vdots \\ \boldsymbol{x}^{(j)T} \\ \vdots \\ \boldsymbol{x}^{(m)T} \end{bmatrix} \qquad \boldsymbol{Y}(\vec{y}) = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(j)} \\ \vdots \\ y^{(m)} \end{bmatrix} \qquad m = \#training\ examples$$

$therefore,$

$$J(\theta) = \frac{1}{2} \cdot (\boldsymbol{X\Theta} - \boldsymbol{Y})^T(\boldsymbol{X\Theta} - \boldsymbol{Y})$$

$$minimize: \quad \boldsymbol{\nabla_\theta} J(\theta) \xrightarrow{set} \vec{0}$$

$$\boldsymbol{\nabla_\theta} J(\theta) = \boldsymbol{\nabla_\theta} \frac{1}{2} \cdot (\boldsymbol{X\Theta} - \boldsymbol{Y})^T (\boldsymbol{X\Theta} - \boldsymbol{Y})$$

$$= \frac{1}{2} \boldsymbol{\nabla_\theta} tr(\boldsymbol{\Theta}^T \boldsymbol{X}^T \boldsymbol{X\Theta} - \boldsymbol{\Theta}^T \boldsymbol{X}^T \boldsymbol{Y} - \boldsymbol{Y}^T \boldsymbol{X\Theta} + \boldsymbol{Y}^T \boldsymbol{Y})$$

$$= \frac{1}{2} (\boldsymbol{X}^T \boldsymbol{X\Theta} + \boldsymbol{X}^T \boldsymbol{X\Theta} - \boldsymbol{X}^T \boldsymbol{Y} - \boldsymbol{X}^T \boldsymbol{Y})$$

$$= \boldsymbol{X}^T \boldsymbol{X\Theta} - \boldsymbol{X}^T \boldsymbol{Y} \xrightarrow{set} \vec{0}$$

$$\therefore \boldsymbol{X}^T \boldsymbol{X\Theta} = \boldsymbol{X}^T \boldsymbol{Y}$$

$$\rightarrow \; \boldsymbol{\Theta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{Y}$$

cannot inverse?

1. features have extra linear relation →delete!
2. m<n →regularize training examples!

Lesson 3

Outline this Lesson:

1. Linear regression→locally weighted regression

   ↓

2. Probabilistic interpretation ($J(\theta)$ probabilities' expression)

   ↓

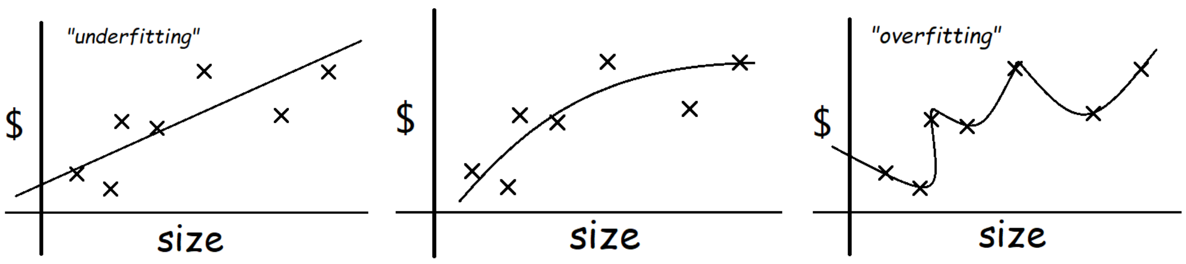
3. Logistic regression →perceptron algorithm

① linear regression extension (polynomial regression)

linear regression $h_\theta(x)$ parameter x_i can be represent like this:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 \quad where\ x_2 = x_1^2$$

→*underfitting & overfitting*



$\theta_0 + \theta_1 x_1$ $\qquad$ $\theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ $\qquad$ $\theta_0 + \theta_1 x_1 + \cdots + \theta_6 x_1^6$

"Parametric" learning algorithm: training examples - get θs - fixed set of parameters

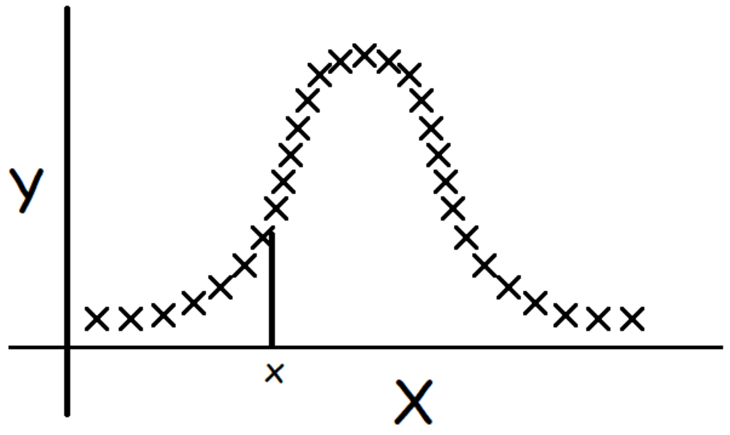
→"Non-parametric" learning algorithm:

$(definition) - number\ of\ parameters\ grows\ linearly\ with\ M (M\ is\ \#sets)$

In another word, parameters rely the whole training sets, even after learning.

→ *locally weighted regression*(loess/lowess)

e.g.



$To\ evaluate\ \boldsymbol{h}\ at\ a\ certain\ x$

$LR: Fit\ \theta\ to\ minimize \sum_i \left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2, then\ return\ \theta^T x (the\ estimated\ \boldsymbol{h}).$

LWR: *Fit* θ *to minimize*

$$\sum_i \omega^{(i)}\left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2$$

where

$$\omega^{(i)} = e^{-\frac{\left(x^{(i)}-x\right)^2}{2\tau^2}} \quad \left(\omega^{(i)} \in (0,1) \;\; \tau \text{ is called "bandwith"}\right)$$

$If\ \left|x^{(i)} - x\right|\ smail, then\ \omega^{(i)} \approx 1 \qquad If\ \left|x^{(i)} - x\right|\ large, then\ \omega^{(i)} \approx 0$

② Probabilistic interpretation

assume $$y^{(i)} = \boldsymbol{\theta}^T x^{(i)} + \varepsilon^{(i)}$$

where $$\varepsilon^{(i)} = error \sim \mathcal{N}(0, \sigma^2) \quad (by\ central\ limit\ theorem)$$

$$P\left(\varepsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(\varepsilon^{(i)}\right)^2}{2\sigma^2}}$$

therefore, $$P\left(y^{(i)} | x^{(i)}; \boldsymbol{\theta}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)}-\boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}}$$

$\rightarrow$ $$y^{(i)} | x^{(i)}; \boldsymbol{\theta} \sim \mathcal{N}\left(\boldsymbol{\theta}^T x^{(i)}, \sigma^2\right)$$

PS: $P\left(y^{(i)} | x^{(i)}, \boldsymbol{\theta}\right)$ means the probability of $y^{(i)}$ given $x^{(i)}, \theta$ (independent variables);

$P\left(y^{(i)} | x^{(i)}; \boldsymbol{\theta}\right)$ means the probability of $y^{(i)}$ given $x^{(i)}$ and parameterized by θ

assume $\varepsilon^{(i)}s$ *are IID (independently, identically distributed),*

$$L(\boldsymbol{\theta}) = P(\boldsymbol{Y} | \boldsymbol{X}; \boldsymbol{\theta}) = \prod_{i=1}^{m} P\left(y^{(i)} | x^{(i)}; \boldsymbol{\theta}\right) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)}-\boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}}$$

purpose:

$$\underset{\boldsymbol{\theta}}{maximize}\, L(\theta)$$

(choose $\boldsymbol{\theta}$ *to maximize the likelihood)*

$\rightarrow$*structure* $\ell(\boldsymbol{\theta})$:

$$\ell(\boldsymbol{\theta}) = log\, L(\boldsymbol{\theta}) \quad (log\ means\ ln)$$

$$= log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)}-\boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}}$$

$$= \sum_{i=1}^{m} log \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)}-\boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}}\right]$$

$$= m\, log \frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{m} - \frac{\left(y^{(i)}-\boldsymbol{\theta}^T x^{(i)}\right)^2}{2\sigma^2}$$

$\therefore$ *maximize* $\ell(\boldsymbol{\theta})$ *is the same as minimizing*

$$J(\theta) = \frac{1}{2} \cdot \sum_{i=1}^{m} \left(y^{(i)} - \boldsymbol{\theta}^T x^{(i)}\right)^2$$

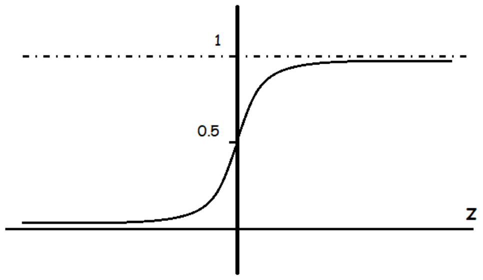
③ Logistic regression (Classification)

$$y \in \{0,1\} \rightarrow h_\theta(x) \in [0,1]$$

Choose cost function:

$$h_\theta(x) = g(\boldsymbol{\theta}^T x) = \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}$$

PS: $g(z) = \frac{1}{1+e^{-z}}$ (*so called Sigmoid function / logistic function*)



$$\rightarrow P(y=1|x;\boldsymbol{\theta}) = h_\theta(x) \ , \ P(y=0|x;\boldsymbol{\theta}) = 1 - h_\theta(x)$$

$$\therefore \quad P(y|x;\boldsymbol{\theta}) = h_\theta(x)^y \cdot \left(1-h_\theta(x)\right)^{1-y} \qquad y \in \{0,1\}$$

therefore,

$$L(\boldsymbol{\theta}) = P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) = \prod_i P\left(y^{(i)}|x^{(i)};\boldsymbol{\theta}\right)$$

$$= \prod_i h\left(x^{(i)}\right)^{y^{(i)}} \cdot \left(1 - h\left(x^{(i)}\right)\right)^{1-y^{(i)}}$$

$$\rightarrow \ \ell(\boldsymbol{\theta}) = \log L(\boldsymbol{\theta}) \quad (\log \text{ means } \ln)$$

$$= \sum_{i=1}^{m} \left[y^{(i)} \log h_{\boldsymbol{\theta}}\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right)\right]$$

$\therefore$ *Logistic regression called*

$$\boldsymbol{\Theta} := \boldsymbol{\Theta} + \alpha \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$$

(like gradient "ascent", to maximize likelihood)

$\rightarrow$ *for all j in* $\boldsymbol{\Theta}$,

$$\frac{\partial}{\partial \boldsymbol{\theta}_j} \ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

$$\boldsymbol{\theta}_j := \boldsymbol{\theta}_j + \alpha \cdot \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

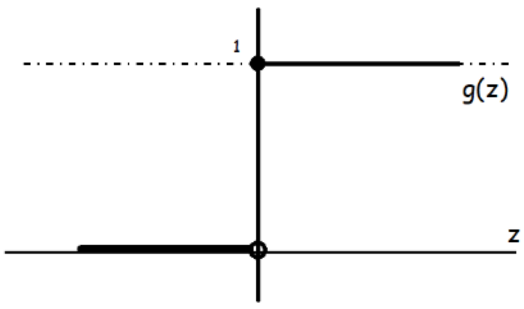
perceptron algorithm

Choose cost function:

$$h_\theta(x) = g(\boldsymbol{\theta}^T x) = \begin{cases} 1 & \text{if } \boldsymbol{\theta}^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

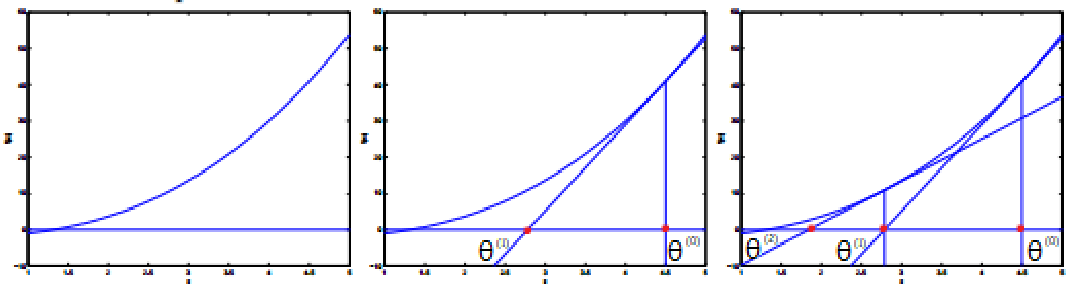
repeat till convergence:

$$\boldsymbol{\theta}_j := \boldsymbol{\theta}_j + \alpha \cdot \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) \cdot x_j^{(i)}$$



Lesson 4

① Logistic regression – Newton's Method



$f(\boldsymbol{\theta})$ $\quad$ $find\ \boldsymbol{\theta} \quad s.t.\ f(\boldsymbol{\theta}) = 0$

$$let\ \Delta = \theta^{(0)} - \theta^{(1)} \quad then \quad \Delta = \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$$

$$\therefore\ \theta^{(1)} = \theta^{(0)} - \frac{f(\theta^{(0)})}{f'(\theta^{(0)})} \Rightarrow \theta^{(t+1)} = \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}$$

$$\rightarrow \quad \ell(\boldsymbol{\theta})\ want\ \boldsymbol{\theta}\ s.t.\ \ell'(\boldsymbol{\theta}) = 0 \Rightarrow \theta^{(t+1)} = \theta^{(t)} - \frac{\ell'(\theta^{(t)})}{\ell''(\theta^{(t)})}$$

$$\therefore \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{H}^{-1}\boldsymbol{\nabla}_{\boldsymbol{\theta}}\ell$$

*where **H** is the Hessian matrix*

$$\boldsymbol{H}^{-1} = \begin{bmatrix} H_{11} & \cdots & H_{1n} \\ \vdots & \ddots & \vdots \\ H_{n1} & \cdots & H_{nn} \end{bmatrix}^{-1}, \qquad H_{ij} = \frac{\partial^2 \ell}{\partial\theta_i \partial\theta_j}$$

Newton's Method convergence rate: quadratic conversions

② Exponential Family

$$P(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) \begin{cases} \boldsymbol{y} \in \mathcal{R}: Gaussian \rightarrow least\ squares & \mathcal{N}(\mu,\sigma^2) \\ \boldsymbol{y} \in \{0,1\}: Bernoulli \rightarrow logistic\ regression & P(y=1;\phi) = \phi \end{cases}$$

if a class of distributions can be written in the form

$$P(\boldsymbol{y};\boldsymbol{\eta}) = b(\boldsymbol{y}) \cdot e^{\boldsymbol{\eta}^T T(\boldsymbol{y}) - a(\boldsymbol{\eta})}$$

$\boldsymbol{\eta}$ – *natural(or canonical) parameter*

$T(\boldsymbol{y})$ – *sufficient statistic(usually, $T(\boldsymbol{y}) = \boldsymbol{y}$)*

$a(\boldsymbol{\eta})$ – *log partition function*

it belongs to Exponential Family, represented by

$$\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta} \sim ExpFamily(\boldsymbol{\eta})$$

e.g.

$\rightarrow Ber(\phi): (y \in \{0,1\})$

$$P(y;\phi) = \phi^y(1-\phi)^{1-y}$$
$$= e^{\log \phi^y (1-\phi)^{1-y}}$$
$$= exp(y \log \phi + (1-y)\log(1-\phi))$$
$$= exp\left(y \log \frac{\phi}{1-\phi} + \log(1-\phi)\right)$$
$$= 1 \cdot e^{\log\frac{\phi}{1-\phi}\cdot y - \log\frac{1}{1-\phi}}$$
$$\therefore \ \boldsymbol{\eta}^T = \eta = \log\frac{\phi}{1-\phi} \quad \Rightarrow \quad \phi = \frac{1}{1+e^{-\eta}}$$

→ *Gaussian*:

$$\mathcal{N}(\mu, \sigma^2) \qquad set \ \ \sigma^2 = 1$$

$$P(y;\mu) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-\mu)^2} = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2} e^{\mu\cdot y - \frac{1}{2}\mu^2}$$

→ $b(y) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2}, \ \eta = \mu, \ T(y) = y, \ a(\eta) = \frac{1}{2}\eta^2$

③ Generalized Linear Models (GLMs)

assume:

(1) $\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta} \ \sim \ ExpFamily(\eta)$

(2) *given* $\boldsymbol{x}$, *goal* → *output* $\boldsymbol{E}\left[T(\boldsymbol{y})|\boldsymbol{x}\right]$ *want* $h(\boldsymbol{x}) = \boldsymbol{E}\left[T(\boldsymbol{y})|\boldsymbol{x}\right]$

(3) $\eta = \boldsymbol{\theta}^T\boldsymbol{x}$ (η *is a real number*) *or* $\eta_i = \boldsymbol{\theta}_i^T\boldsymbol{x}, \ if \ \boldsymbol{\eta} \in \mathcal{R}^\ell$

e.g.

→*Bernoulli*:

$$\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta} \ \sim \ ExpFamily(\eta) \quad (from\ above) \tag{1}$$

For fixed $\boldsymbol{x},\boldsymbol{\theta}$ *algorithm output*

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{E}[\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}] = P(y=1|x;\theta) = \phi \tag{2}$$

$$= \frac{1}{1+e^{-\eta}} = \frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}} \tag{3}$$

$$\left(\begin{matrix} g(\eta) = \boldsymbol{E}[y;\eta] = \dfrac{1}{1+e^{-\eta}} \ \ \text{“canonical response function”} \\ g^{-1}(\eta) = 1 + e^{-\eta} \qquad \text{“canonical link function”} \end{matrix}\right)$$

→$Multinomial$:

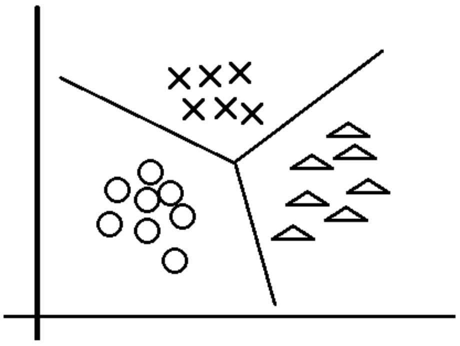
$$y \in \{1, \dots, k\}$$

$Parameters$: $\phi_1, \phi_2, \dots, \phi_k$

$$P(y = i) = \phi_i \qquad i = 1, \dots, k$$

$$\therefore \qquad \phi_k = 1 - (\phi_1 + \cdots + \phi_{k-1})$$

$Parameters$: $\phi_1, \phi_2, \dots, \phi_{k-1}$



$define$:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \in \mathcal{R}^{k-1}$$

$$T(k-1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad T(k) = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\mathbb{1}\{true\} = 1 \qquad \mathbb{1}\{false\} = 0$$

$$\therefore \quad T(y)_i = \mathbb{1}\{y = i\} \qquad i = 1, \dots, k$$

$$\therefore \quad P(y) = \phi_1^{\mathbb{1}\{y=1\}} \cdot \phi_2^{\mathbb{1}\{y=2\}} \cdot \dots \cdot \phi_k^{\mathbb{1}\{y=k\}}$$

$$= \phi_1^{T(y)_1} \cdot \phi_2^{T(y)_2} \cdot \dots \cdot \phi_{k-1}^{T(y)_{k-1}} \cdot \phi_k^{1-\sum_{j=1}^{k-1} T(y)_j}$$

$$= 1 \cdot e^{\sum_{i=1}^{k-1} log \frac{\phi_i}{\phi_k} \cdot T(y)_i - (-\,log\,\phi_k)}$$

$$\sim \; ExpFamily(\eta) \tag{1}$$

$$\rightarrow \eta = \begin{bmatrix} log\frac{\phi_1}{\phi_k} \\ \vdots \\ log\frac{\phi_{k-1}}{\phi_k} \end{bmatrix}, T(y) = \begin{bmatrix} \mathbb{1}\{y = 1\} \\ \vdots \\ \mathbb{1}\{y = k-1\} \end{bmatrix} \in \mathcal{R}^{k-1} \quad b(y) = 1, a(\eta) = -\,log\,\phi_k$$

$$\therefore \quad \phi_i = \frac{e^{\eta_i}}{1+\sum_{j=1}^{k-1} e^{\eta_j}} \left(= \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}} \right) \ (i = 1, \dots, k)$$

$$= \frac{e^{\boldsymbol{\theta}_i^T \boldsymbol{x}}}{1+\sum_{j=1}^{k-1} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}}} \left(= \frac{e^{\boldsymbol{\theta}_i^T \boldsymbol{x}}}{\sum_{j=1}^{k} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}}} \right) \quad [\eta_i = \boldsymbol{\theta}_i^T \boldsymbol{x}] \tag{3}$$

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{E}[T(y)|\boldsymbol{x};\boldsymbol{\theta}]$$

$$= \boldsymbol{E}\left[\begin{matrix} \mathbb{1}\{y=1\} \\ \vdots \\ \mathbb{1}\{y=k-1\} \end{matrix} \middle| \boldsymbol{x};\boldsymbol{\theta} \right] = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix}$$

$$= \begin{bmatrix} \dfrac{e^{\boldsymbol{\theta}_1^T \boldsymbol{x}}}{\left(1+\sum_{j=1}^{k-1} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}}\right)} \\ \vdots \\ \dfrac{e^{\boldsymbol{\theta}_{k-1}^T \boldsymbol{x}}}{\left(1+\sum_{j=1}^{k-1} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}}\right)} \end{bmatrix} \tag{2}$$

Softmax regression:

$y \in \{1, \dots, k\}$ $\qquad$ *samples are* $\left(x^{(1)}, y^{(1)}\right), \dots, \left(x^{(m)}, y^{(m)}\right)$

purpose: $\underset{\boldsymbol{\theta}}{maximize}\, L(\theta) \rightarrow \underset{\boldsymbol{\theta}}{maximize}\, \ell(\theta) \ (= \log L(\theta))$

$$L(\boldsymbol{\theta}) = P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) = \prod_{i=1}^{m} P\left(y^{(i)}|x^{(i)};\boldsymbol{\theta}\right)$$

$$= \prod_{i=1}^{m} \left(\phi_1^{\mathbb{1}\{y^{(i)}=1\}} \cdot \phi_2^{\mathbb{1}\{y^{(i)}=2\}} \cdot \dots \cdot \phi_k^{\mathbb{1}\{y^{(i)}=k\}} \right)$$

$$\phi_l = \frac{e^{\boldsymbol{\theta}_l^T \boldsymbol{x}}}{1+\sum_{j=1}^{k-1} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}}} \quad (l = 1, \dots, k \quad \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_{k-1} \in \mathcal{R}^{n+1})$$

$$\therefore \qquad \ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} log \prod_{l=1}^{k} \left(\frac{e^{\boldsymbol{\theta}_l^T \boldsymbol{x}^{(i)}}}{\sum_{j=1}^{k} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}^{(i)}}} \right)^{\mathbb{1}\{y^{(i)}=l\}}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{k} \mathbb{1}\{y^{(i)} = j\} \cdot log \frac{e^{\boldsymbol{\theta}_j^T \boldsymbol{x}^{(i)}}}{\sum_{l=1}^{k} e^{\boldsymbol{\theta}_l^T \boldsymbol{x}^{(i)}}}$$

$$\boldsymbol{\Theta} := \boldsymbol{\Theta} + \alpha \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$$

$\rightarrow$ *for all j in* $\boldsymbol{\Theta}$,

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}_j} \ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left[\boldsymbol{x}^{(i)} \left(\mathbb{1}\{y^{(i)} = j\} - \frac{e^{\boldsymbol{\theta}_j^T \boldsymbol{x}^{(i)}}}{\sum_{l=1}^{k} e^{\boldsymbol{\theta}_l^T \boldsymbol{x}^{(i)}}} \right) \right]$$

$$\boldsymbol{\theta}_j := \boldsymbol{\theta}_j + \alpha \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}_j} \ell(\boldsymbol{\theta})$$

Lesson 5

Outline this Lesson:

1. Generative learning algorithms
2. GDA (Gaussian discriminant analysis)→digression: Gaussians
3. Generative & Discriminative comparison (GDA & logistic regression comparison)
4. Naive Bayes
5. Laplace Smoothing

① generative learning algorithms (e.g. logistic regression)

→*Discriminative*:

- *learns* $p(y|x)$

- *or learns* $h_{\boldsymbol{\theta}}(x) \in \{0,1\}$ *directly*

→*Generative*:

- *models* $p(x|y)$ (*and* $p(y)$) (*where* x: *features* y: *class label*)

$$p(y|x) = \frac{p(x|y) \cdot p(y)}{p(x)} \rightarrow P(y = 1|x) = \frac{P(x|y = 1) \cdot p(y = 1)}{p(x)}$$

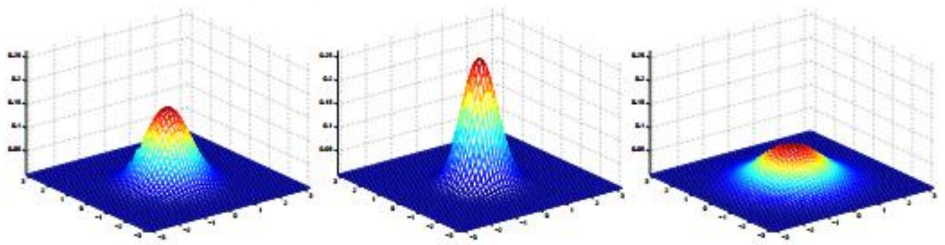
$p(x) = P(x|y = 1) \cdot p(y = 1) + P(x|y = 0) \cdot p(y = 0)$

② GDA (Gaussian Discriminant Analysis)

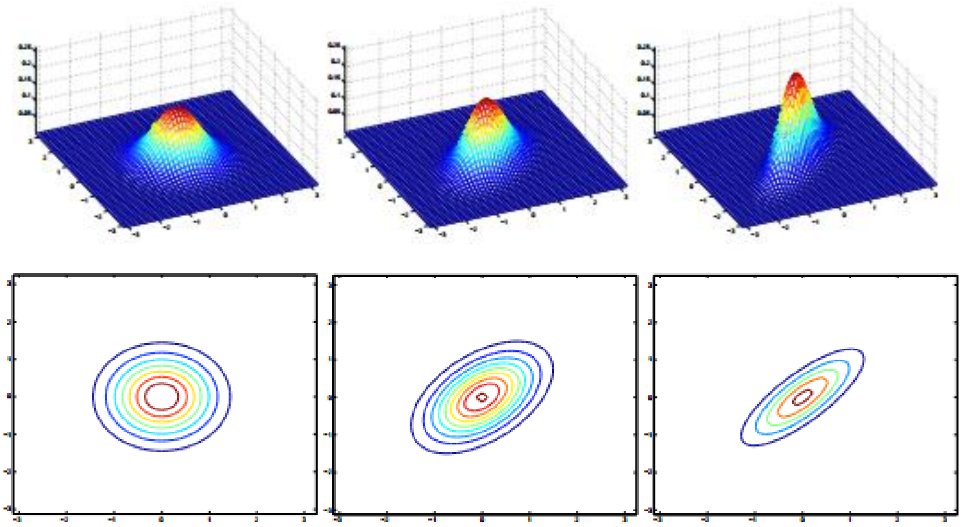
assume $x \in \mathcal{R}^n$, *continuous-valued* $p(x|y)$ *is Gaussian*

$x \sim \mathcal{N}(\vec{\mu}, \Sigma)$ *where* $\vec{\mu}$ *is the mean of* x, Σ $(\in \mathcal{R}^{n\times n})$ *is the covariance matrix*

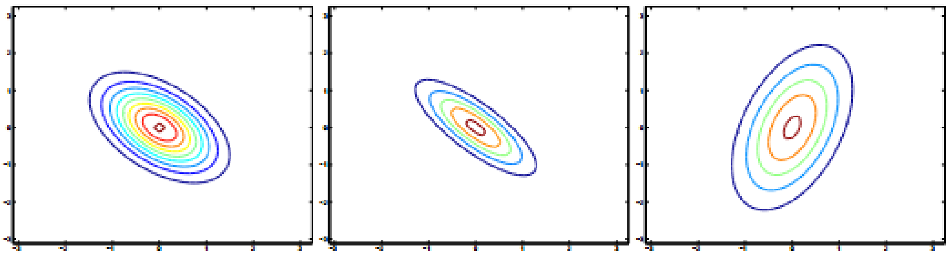
$$p(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$



$\Sigma = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$; $\Sigma = \begin{bmatrix}0.6 & 0\\0 & 0.6\end{bmatrix}$; $\Sigma = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}$

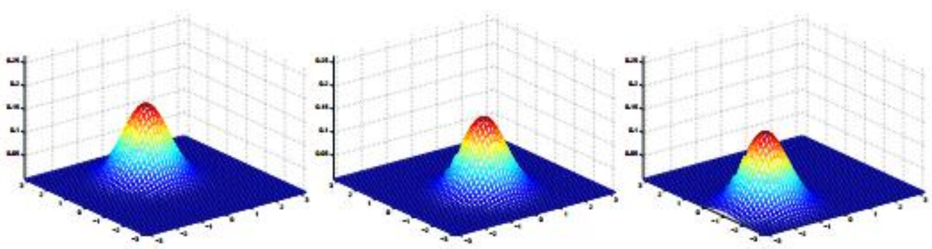


$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$; $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$; $\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$



$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$; $\Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$; $\Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}$

figure: different covariance of Gaussian functions (2-D)



$\mu = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$; $\mu = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}$; $\mu = \begin{bmatrix} -1 \\ -1.5 \end{bmatrix}$

figure: different means of Gaussian functions (2-D)

$GDA\ model(e.g.\ 2D\ of\ \boldsymbol{Y})$:

$$y \sim Bernoulli(\phi)$$

$$x|y = 0 \sim \mathcal{N}(\mu_0, \Sigma)$$

$$x|y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$$

→ $p(y) = \phi^y(1-\phi)^{1-y}$

$$p(x|y=0) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)$$
$$p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)$$

$\therefore$ *the log Joint likelihood*:

$$\ell(\phi,\mu_0,\mu_1,\Sigma) = log\prod_{i=1}^{m} p\left(x^{(i)},y^{(i)};\phi,\mu_0,\mu_1,\Sigma\right)$$
$$= log\prod_{i=1}^{m} p\left(x^{(i)}\middle|y^{(i)};\mu_0,\mu_1,\Sigma\right)\cdot p\left(y^{(i)};\phi\right)$$

$\leftrightarrow$ *different from "the conditional likelihood" (e.g. logistic regression)*

$$\ell(\theta) = log\prod_{i=1}^{m} p\left(y^{(i)}\middle|x^{(i)};\theta\right)$$

purpose:

Maximize ℓ *with respected to* ϕ,μ_0,μ_1,Σ

$$\rightarrow \quad \phi = \frac{1}{m}\sum_i y^{(i)} = \frac{1}{m}\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=0\}x^{(i)}}{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=0\}}$$

$$\left(\frac{\text{sum of } x^{(i)} \text{ for which } y^{(i)}=0}{\#\text{ examples with label } 0}\right)$$

$$\mu_1 = \frac{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=1\}x^{(i)}}{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=1\}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)}-\mu_{y^{(i)}}\right)\left(x^{(i)}-\mu_{y^{(i)}}\right)^T$$

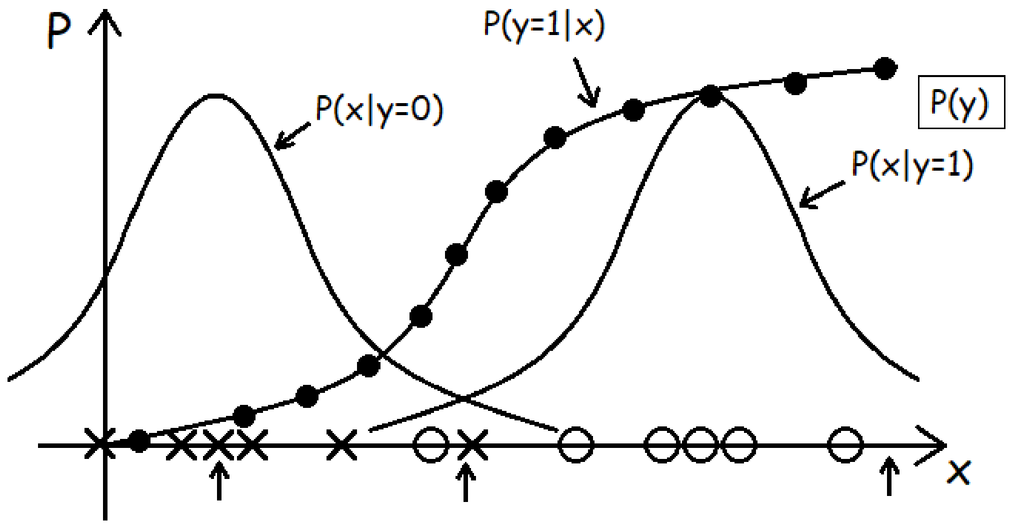
Predict:

$$\arg\max_y P(y|x) = \arg\max_y \frac{P(x|y)P(y)}{P(x)}$$

$$= \arg\max_y P(x|y)P(y) \quad (P(x)\ not\ depend\ on\ y)$$

$$(especially, if\ P(y)\ is\ uniform:\ \arg\max_y P(y|x) = \arg\max_y P(x|y))$$

$$\arg\max_X (min)\cdots \quad means\ the\ argument\ X\ of\ \max_X(min)\cdots$$



$$\therefore P(y=1|x) = \frac{P(x|y=1)\cdot p(y=1)}{p(x)}$$

$$\left(\begin{array}{c} p(y=1)\ can\ be\ fit\ by\ \dfrac{\#\ x\ which\ y=1}{\#\ total\ x} \\ here, p(y) = p(y=0) + p(y=1) \end{array} \right)$$

$$p(x) = P(x|y=1)\cdot p(y=1) + P(x|y=0)\cdot p(y=0)$$

③ GDA & logistic regression comparison

$$x|y \sim Gaussian$$
$$\Downarrow \qquad \not\Uparrow$$
$$logistic\ posterior\ for\ p(y=1|x)$$

accordingly,

$$\begin{cases} x|y=1 \sim Poisson(\lambda_1) \\ x|y=0 \sim Poisson(\lambda_0) \end{cases} \quad \begin{array}{c}\Rightarrow \\ \not\Leftarrow\end{array} \quad p(y=1|x)\ is\ logistic$$

more generally,

$$\begin{cases} x|y=1 \sim ExpFamily(\eta_1) \\ x|y=0 \sim ExpFamily(\eta_0) \end{cases} \quad \begin{array}{c}\Rightarrow \\ \not\Leftarrow\end{array} \quad p(y=1|x)\ is\ logistic$$

It shows that logistic regression is robust for different classification questions.

GDA makes stronger modeling assumptions, and is more data efficient (i.e., requires less training data to learn "well") when the modeling assumptions are (approximately) correct.

Logistic regression makes weaker assumptions, and is significantly more robust to deviations from modeling assumptions.

④ Naive Bayes (e.g. text classification: anti-spam)

$$y \in \{0,1\} \quad (1 \textit{ represent spam email})$$

$$x = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} a \\ aardvark \\ \vdots \\ buy \\ cs229 \\ \vdots \\ zygmurgy \end{matrix} \qquad \left| \begin{matrix} p(x|y) \\ x \in \{0,1\}^n \\ n = 50{,}000 \\ 2^{50{,}000} \ results \\ 2^{50{,}000} - 1 \ parameters \end{matrix} \right.$$

assume $x_i's$ *are conditinally independent given* y

$$p(x_1, \dots, x_{50000}|y)$$

$$= p(x_1|y)p(x_2|y,x_1)p(x_3|y,x_1,x_2)\cdots p(x_{50000}|y,x_1,\dots,x_{49999})$$

$$= p(x_1|y)p(x_2|y)p(x_3|y)\cdots p(x_{50000}|y) \quad (conditinally\ independent)$$

$$= \prod_{i=1}^{n} p(x_i|y)$$

Parameterize:

$$\phi_{i|y=1} = p(x_i = 1|y = 1) \quad \rightarrow \quad build\ p(x|y)$$

$$\phi_{i|y=0} = p(x_i = 1|y = 0) \quad \nearrow$$

$$\phi_y = p(y = 1) \quad \rightarrow \quad build\ p(y)$$

∴ *Joint likelihood*:

$$\mathcal{L}(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)})$$

→
$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} \mathbb{1}\{x_j^{(i)}=1, y^{(i)}=1\}}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)}=1\}}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} \mathbb{1}\{x_j^{(i)}=1, y^{(i)}=0\}}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)}=0\}}$$

$$\phi_y = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)}=1\}}{m}$$

$\therefore$ *When predict a new sample*:

$$P(y=1|x) = \frac{P(x|y=1)p(y=1)}{p(x)}$$

$$= \frac{(\prod_{i=1}^{n} p(x_i|y=1))p(y=1)}{(\prod_{i=1}^{n} p(x_i|y=1))p(y=1) + (\prod_{i=1}^{n} p(x_i|y=0))p(y=0)}$$

⑤ Laplace Smoothing

When a brand new word (e.g. “NIPS”, which is the 30,000th word in dictionary) inputs, the prior probabilities are as follows:

$$P(x_{30000} = 1|y=1) = 0$$

$$P(x_{30000} = 1|y=0) = 0$$

and then,

$$P(y=1|x) = \frac{(\prod_{i=1}^{n} p(x_i|y=1))p(y=1)}{(\prod_{i=1}^{n} p(x_i|y=1))p(y=1) + (\prod_{i=1}^{n} p(x_i|y=0))p(y=0)} = \frac{0}{0}$$

To avoid this problem, we use Laplace Smoothing to replace estimate with

$$P(y=j) = \phi_j = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = j\} + 1}{m+k}$$

$(k \text{ is } \#class, \text{the all } y\text{'s possible values})$

Lesson 6

Outline this Lesson:

1. Naive Bayes – event models
2. Neural Networks (Intro.)
3. Support Vector Machines

① event models

$primal$

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} \quad x_i \in \{0,1\}$$

$Generative\ Learning\ algorithm:$

$1. P(x|y) = \prod_{i=1}^{n} P(x_i|y)$

$2. P(y)$

$3. \arg\max_y P(y|x) = \arg\max_y P(x|y)P(y)$

$n = \#\ words\ in\ dictionary (e.g.\ 50{,}000)$

called *"Multi-variate Bernoulli event model"*

(here x represents one of samples, x_i is one of words in dictionary whether in x or not)

without using number information? → $x_i \in \{1,2,\dots,k\}$

Multinomial event model

$$P(x|y) = \prod_{i=1}^{n} P(x_i|y) \qquad here\ x_i\ is\ multinomial$$

e.g.

Living area (sq. feet)	< 400	400-800	800-1200	1200-1600	>1600
x_i	1	2	3	4	5

back to primal example, we can change the model with

$$x^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \dots, x_{n_i}^{(i)}\right) \quad n_i = \#\ words\ in\ email\ x^{(i)}, x_j^{(i)} \in \{1,2,\dots,50000\}$$

($x^{(i)}$ is a vector with different length in emails, x_i represents the number where in dictionary)

$$P(x,y) = \left(\prod_{i=1}^{n} P(x_i|y)\right) P(y)$$

$$\therefore\ \mathcal{L}(\phi_y, \phi_{k|y=0}, \phi_{k|y=1}) = \prod_{i=1}^{m} p\left(x^{(i)}, y^{(i)}\right)$$

$$= \prod_{i=1}^{m} \left(\prod_{j=1}^{n_i} P\left(x_j^{(i)} \middle| y; \phi_{k|y=0}, \phi_{k|y=1}\right)\right) P(y; \phi_y)$$

$$\rightarrow \quad \phi_{k|y=1} = P(x_j = k|y=1) = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)}=1\}\sum_{j=1}^{n_i}\mathbb{1}\{x_j^{(i)}=k\} + 1}{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=1\}\cdot n_i + |V|(50000)}$$

$$\left(\frac{\#\ k\ in\ all\ spams}{\#\ all\ words\ in\ all\ spams}\right)$$

$$\phi_{k|y=0} = P(x_j = k|y=0) = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)}=0\}\sum_{j=1}^{n_i}\mathbb{1}\{x_j^{(i)}=k\} + 1}{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=0\}\cdot n_i + |V|(50000)}$$

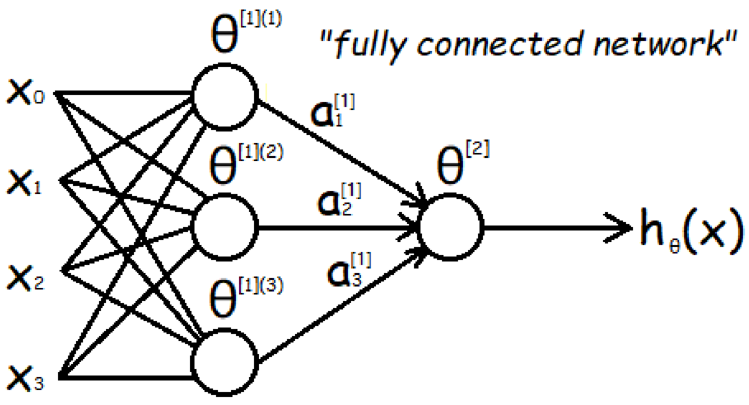
$$\phi_y = P(y=1) = \frac{\sum_{i=1}^{m}\mathbb{1}\{y^{(i)}=1\}}{m}$$

in other words,

$$x_i \in \{1,2,\dots,\ell\} \quad \rightarrow \quad P(x=k) = \frac{\#\ observations\ of\ "x{=}k" + 1}{\#\ observations\ of\ all\ x + \ell}$$

Because Naive Bayes algorithm is built by Bernoulli or Multinomial (i.e. or other distributions), it still belongs to exponential family (GLMs), so finally, it is still a *linear classifier*.

② Neural Networks



$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \vec{a}^{[2]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}$$

$$g(z) = \frac{1}{1+e^{-z}} \; (or\ others, nonlinear)$$
$$a_i^{[1]} = g\left(\boldsymbol{x}^T\theta^{[1](i)}\right) \quad i = 1, \dots, 3$$
$$h_\theta(\boldsymbol{x}) = g\left(\vec{a}^{[2]T}\theta^{[2]}\right)$$
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(\boldsymbol{x}^{(i)}\right)\right)^2$$

Q: How to build a basic neural network and compute all the parameters?

- references: cs229-notes-deep_learning & cs229-notes-backprop

③ Support Vector Machines

(1) Functional margins (e.g. logistic regression)

$Compute\ \theta^T x$

$Predicts\ "1" \quad \Leftrightarrow \quad \theta^T x \geq 0$

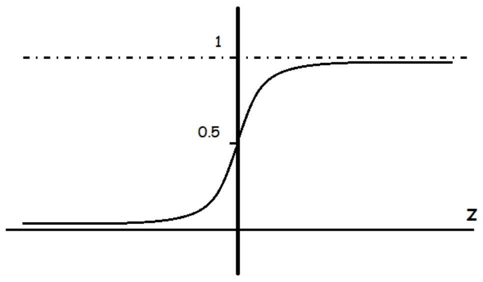
$Predicts\ "0" \quad \Leftrightarrow \quad \theta^T x < 0$

$If\ \theta^T x \gg 0, very\ "confident"\ \ that\ y = 1$

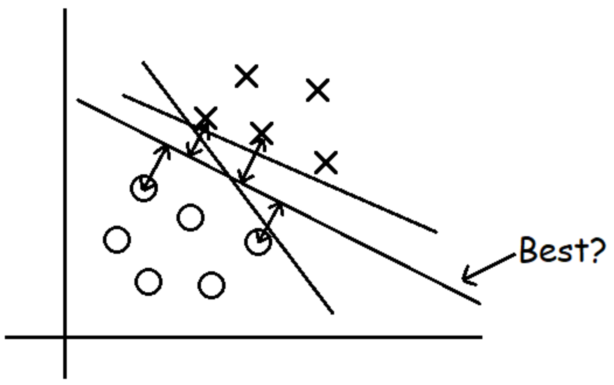
$If\ \theta^T x \ll 0, very\ "confident"\ \ that\ y = 0$

$Nice\ if\ \forall i\ \ s.t.\ \ y^{(i)} = 1, have\ \theta^T x^{(i)} \gg 0$

$\qquad \forall i\ \ s.t.\ \ y^{(i)} = 0, have\ \theta^T x^{(i)} \ll 0$



(2) Geometric margins



(3) notation

$$y \in \{-1, +1\} \qquad g(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

$have\ (function)\ h\ output\ values\ in\ \{-1, +1\}$

$$h_\theta(\boldsymbol{x}) = g(\boldsymbol{\theta}^T\boldsymbol{x}) \; (x_0 = 1) \quad \boldsymbol{x} \in \mathcal{R}^{n+1}$$
$$\downarrow$$
$$h_{\boldsymbol{w},b}(\boldsymbol{x}) = g(\boldsymbol{w}^T\boldsymbol{x} + b) \; where\ \boldsymbol{w} = (\theta_1, \dots, \theta_n)^T, b = \theta_0$$

(4) definition & deduction

define

Functional margin of a hyperplane $(\boldsymbol{w}, b)$ *w.r.t.* $\left(x^{(i)}, y^{(i)}\right)$ *is:*

$$\hat{\gamma}^{(i)} = y^{(i)}\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b\right)$$

$If\ y^{(i)} = 1, want\ \boldsymbol{w}^T x^{(i)} + b \gg 0$

$If\ y^{(i)} = -1, want\ \boldsymbol{w}^T x^{(i)} + b \ll 0$

$If\ y^{(i)}(\boldsymbol{w}^T x^{(i)} + b) > 0, then\ classified\ \left(x^{(i)}, y^{(i)}\right)\ correctly$

Functional margin of all samples $\left(x^{(i)}, y^{(i)}\right)(i = 1, \dots, m)$ *is:*

$$\hat{\gamma} = \min_i \hat{\gamma}^{(i)}$$

if we change

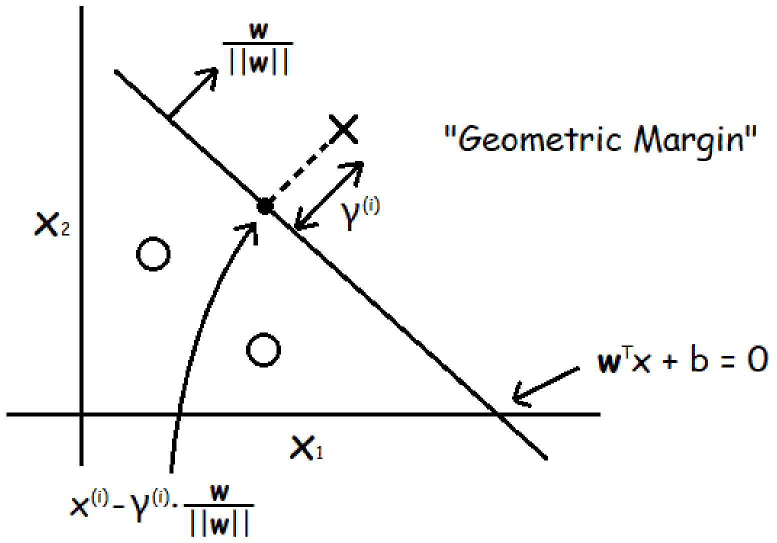
$$\boldsymbol{w} \rightarrow 2\boldsymbol{w}$$
$$b \rightarrow 2b$$

the hyperplane is still the same one, but the functional margin becomes double

→ *want*

$$\|\boldsymbol{w}\| = 1$$

and then, we can deduct the "geometric margin" shown as follows:



$$\boldsymbol{w}^T \left(\boldsymbol{x}^{(i)} - \gamma^{(i)} \cdot \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\right) + b = 0$$

→ $$\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b = \gamma^{(i)} \cdot \frac{\boldsymbol{w}^T \boldsymbol{w}}{\|\boldsymbol{w}\|} = \gamma^{(i)} \|\boldsymbol{w}\|$$

$$\gamma^{(i)} = \left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\right)^T \boldsymbol{x}^{(i)} + \frac{b}{\|\boldsymbol{w}\|}$$

$More\ generally, geometric\ margin$:

$$\gamma^{(i)} = y^{(i)}\left[\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\right)^T \boldsymbol{x}^{(i)} + \frac{b}{\|\boldsymbol{w}\|}\right]$$

$$(If\ \|\boldsymbol{w}\| = 1, \hat{\gamma}^{(i)} = \gamma^{(i)}.\ \gamma^{(i)} = \frac{\hat{\gamma}^{(i)}}{\|\boldsymbol{w}\|})$$

So to the whole samples, we have the definition of "geometric margin" as follows:

$Geometric\ margin$:

$$\gamma = \min_i \gamma^{(i)}$$

$Max\ margin\ classifier$:

$$\max_{\gamma,\boldsymbol{w},b}\ \gamma$$

$$s.t.\quad y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) \geq \gamma \qquad i = 1, \dots, m$$

$$\|\boldsymbol{w}\| = 1$$

Lesson 7

Outline this Lesson:

1. Optimal Margin Classifier
2. Primal/Dual Optimization problem (KKT)
3. SVM dual
4. Kernels

① Optimal margin classifier

from last lesson, we know that change the constraints like:

$$\|\boldsymbol{w}\| = 1, |\boldsymbol{w}_1| = 1, \boldsymbol{w}_1^2 + |\boldsymbol{w}_1| = 1, \ldots$$

will not change the original optimization problem. so,

#1

$$\max_{\gamma,\boldsymbol{w},b} \gamma$$

$$s.t. \quad y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) \geq \gamma \qquad i = 1, \ldots, m$$

$$\|\boldsymbol{w}\| = 1 \leftarrow change!$$

#2

$$\max_{\hat{\gamma},\boldsymbol{w},b} \frac{\hat{\gamma}}{\|\boldsymbol{w}\|} \qquad \left(\frac{\hat{\gamma}}{\|\boldsymbol{w}\|} = \gamma\right)$$

$$s.t. \quad y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) \geq \hat{\gamma} \qquad i = 1, \ldots, m$$

→ $set \quad \hat{\gamma} = 1$

$$\Rightarrow \qquad \max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|}$$

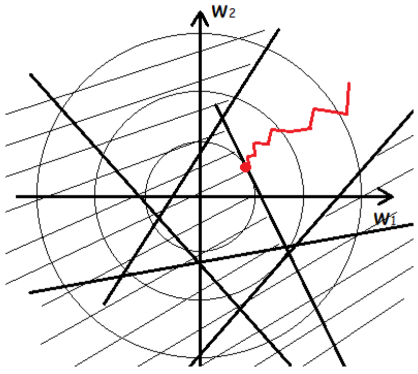
$$s.t. \quad \min_i y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) = 1 \qquad i = 1, \ldots, m$$

so, original problem can re-describe into:

#3

$$\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|^2$$

$$s.t. \quad y^{(i)}\left(\boldsymbol{w}^T x^{(i)} + b\right) \geq 1 \qquad i = 1, \ldots, m$$



② Primal/Dual Optimization problem

here's a function of $\boldsymbol{w}$, $f(\boldsymbol{w})$, that

$$\min_{\boldsymbol{w}} f(\boldsymbol{w})$$

$$s.t. \quad h_i(\boldsymbol{w}) = 0 \qquad i = 1, \ldots, l \quad or \quad "h(\boldsymbol{w}) = \begin{bmatrix} h_1(\boldsymbol{w}) \\ h_2(\boldsymbol{w}) \\ \vdots \\ h_l(\boldsymbol{w}) \end{bmatrix} = \vec{\boldsymbol{0}}"$$

we can build a *"Lagrange equation"* to re-describe it as:

$$\mathcal{L}(\boldsymbol{w},\boldsymbol{\beta}) = f(\boldsymbol{w}) + \sum_i \beta_i h_i(\boldsymbol{w})$$

then,

$$\frac{\partial \mathcal{L}}{\partial w_i} \xrightarrow{set} 0 \ , \quad \frac{\partial \mathcal{L}}{\partial \beta_i} \xrightarrow{set} 0$$

$for\ \boldsymbol{w}^*\ to\ be\ a\ solution, necessary\ that$

$$\exists \boldsymbol{\beta}^* \quad s.t.\ \frac{\partial \mathcal{L}(\boldsymbol{w}^*,\boldsymbol{\beta}^*)}{\partial w_i} = 0, \frac{\partial \mathcal{L}(\boldsymbol{w}^*,\boldsymbol{\beta}^*)}{\partial \beta_i} = 0$$

more generally,

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) \qquad (primal\ problem, p^*)$$

$$\begin{aligned} s.t.\quad & g_i(\boldsymbol{w}) \le 0 \qquad i = 1, \dots, k \quad or \quad "g(\boldsymbol{w}) \le \vec{\boldsymbol{0}}" \\ & h_j(\boldsymbol{w}) = 0 \qquad j = 1, \dots, l \quad or \quad "h(\boldsymbol{w}) = \vec{\boldsymbol{0}}" \end{aligned}$$

$Lagrangian:$

$$\mathcal{L}(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta}) = f(\boldsymbol{w}) + \sum_{i=1}^{k} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{l} \beta_j h_j(\boldsymbol{w})$$

$define:$ $$\theta_{\mathcal{P}}(\boldsymbol{w}) = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \ge 0} \mathcal{L}(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta})$$

$$\rightarrow \qquad p^* = \min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \ge 0} \mathcal{L}(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta}) = \min_{\boldsymbol{w}} \theta_{\mathcal{P}}(\boldsymbol{w})$$

- *why?*

$$\begin{aligned} & If\ g_i(\boldsymbol{w}) > 0\ then\ \ \theta_{\mathcal{P}}(\boldsymbol{w}) = \infty \\ & If\ h_i(\boldsymbol{w}) \ne 0\ then\ \ \theta_{\mathcal{P}}(\boldsymbol{w}) = \infty \\ & Otherwise, \theta_{\mathcal{P}}(\boldsymbol{w}) = f(\boldsymbol{w}) \end{aligned} \Rightarrow \theta_{\mathcal{P}}(\boldsymbol{w}) = \begin{cases} f(\boldsymbol{w}) & f.f.\ constraint\ satisfied \\ \infty & otherwise \end{cases}$$

$Dual\ problem:$

$$\theta_{\mathcal{D}}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta})$$

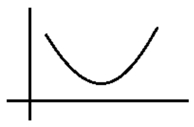
$$d^* = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \ge 0} \min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\beta}) = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i \ge 0} \theta_{\mathcal{D}}(\boldsymbol{\alpha},\boldsymbol{\beta})$$

$$\rightarrow \qquad d^* \le p^* \qquad \max\min(\cdots) \le \min\max(\cdots)$$

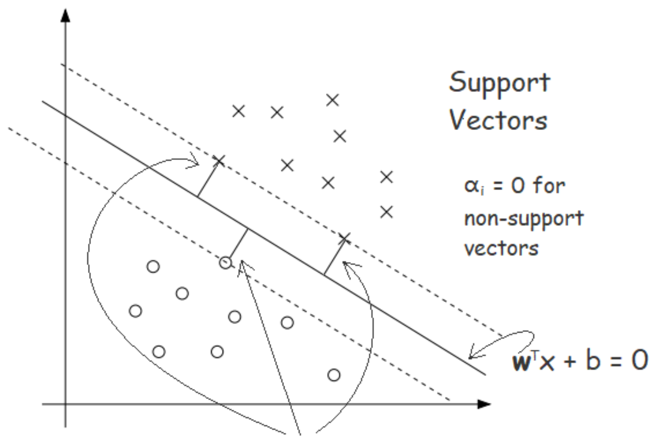
$e.g.$ $$\max_{y \in \{0,1\}} \min_{x \in \{0,1\}} 1\{x = y\} \le \min_{x \in \{0,1\}} \max_{y \in \{0,1\}} 1\{x = y\}$$

$Sometimes,$ $$d^* = p^*$$

Let function "f" be convex. $(Hessian\ H \geq 0)$
Suppose h_i *is affine* $[h_i(w) = a_i^T w + b_i]$
and constraints g_i *are (strictly) feasible.* $[\exists w,\ st.\ \forall i\ g_i(w) < 0]$
Then $\exists w^*, \alpha^*, \beta^*$ *st.* w^* *solve primal, and* α^*, β^* *solve dual, and to solve*
$p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$, *we still need KKT conditions.*



③ SVM dual (& KKT conditions)



SVM cost function as:

$$J(\boldsymbol{w}, b) = \min_{\boldsymbol{w}, b} \frac{1}{2} \|\boldsymbol{w}\|^2$$

$$s.t. \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 \quad (i = 1, \dots, m)$$

Lagrange duality

primal equation

$$\max_{\alpha_i \geq 0} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \max_{\alpha_i \geq 0} \left(\frac{1}{2} \|\boldsymbol{w}\|^2 - \sum_{i=1}^{m} \alpha_i \cdot \left[y^{(i)} \left(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b \right) - 1 \right] \right)$$

$$p^* = \min_{\boldsymbol{w}, b} \max_{\alpha_i \geq 0} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})$$

duality equation

$$d^* = \max_{\alpha_i \geq 0} \min_{\boldsymbol{w}, b} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})$$

KKT conditions $\left(make\ p^* = d^* = \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)\right)$:

$$\frac{\partial}{\partial w_i} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = 0,\ i = 1, \dots, n \qquad (1)$$

$$\frac{\partial}{\partial \beta_i} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = 0,\ i = 1, \dots, l \qquad (2)$$

$$\alpha_i^* g_i(\boldsymbol{w}^*) = 0,\ i = 1, \dots, k \quad (KKT\ complementary\ condition) \qquad (3)$$

$$g_i(\boldsymbol{w}^*) \leq 0,\ i = 1, \dots, k \qquad (4)$$

$$\alpha_i^* \geq 0,\ i = 1, \dots, k \qquad (5)$$

$$\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{m}\alpha_i \cdot \left[y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) - 1\right] \tag{6}$$

$Dual\ problem:$

$$\theta_{\mathcal{D}}(\boldsymbol{\alpha}) = \min_{\boldsymbol{w},b} \mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha})$$

$by\ KKT\ condition\ (1)(2):$

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha}) = \boldsymbol{w} - \sum_{i=1}^{m}\alpha_i y^{(i)}\boldsymbol{x}^{(i)} \overset{set}{\longrightarrow} 0$$

→
$$\boldsymbol{w} = \sum_{i=1}^{m}\alpha_i y^{(i)}\boldsymbol{x}^{(i)} \tag{7}$$

$$\frac{\partial}{\partial b}\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha}) = -\sum_{i=1}^{m}\alpha_i y^{(i)} \overset{set}{\longrightarrow} 0 \tag{8}$$

$from\ Equation\ (6)(7)(8):$

$$\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\left(\boldsymbol{x}^{(i)}\right)^T\boldsymbol{x}^{(j)}$$

→
$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\boldsymbol{y}^{(i)}\boldsymbol{y}^{(j)}\alpha_i\alpha_j\langle\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\rangle$$

From above, we know that $\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha})$ just associate with $\boldsymbol{\alpha}$**.** So,

$Dual\ problem:$

$$\max_{\boldsymbol{\alpha}}\ W(\boldsymbol{\alpha})$$

$$s.t. \quad \alpha_i \geq 0 \quad (i = 1,\dots,m)$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

Once we solve the dual problem, we can easily solve for $\boldsymbol{w}, b$:

$Solve\ for\ \boldsymbol{\alpha}, then$

$$\boldsymbol{w}^* = \sum_{i=1}^{m}\alpha_i y^{(i)}\boldsymbol{x}^{(i)}$$

$$b^* = -\frac{1}{2}\left(\max_{i:y^{(i)}=-1}\boldsymbol{w}^{*T}\boldsymbol{x}^{(i)} + \min_{i:y^{(i)}=1}\boldsymbol{w}^{*T}\boldsymbol{x}^{(i)}\right)$$

$$(\alpha_i \neq 0 \ \textit{only for support vectors!!!})$$

*Finally, to predict new sample(s), we just need calculate:

$$y^p = \begin{cases} +1 & \textit{if } \boldsymbol{w}^{*T} x^p + b^* \geq 0 \\ -1 & \textit{otherwise} \end{cases}$$

④ Kernels

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}$$

$$\rightarrow \qquad \boldsymbol{w}^T \boldsymbol{x} + b = \left(\sum_{i=1}^{m} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \right)^T \boldsymbol{x} + b$$

$$= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle \boldsymbol{x}^{(i)}, \boldsymbol{x} \rangle + b$$

Assume $$h_{\boldsymbol{w},b}(\boldsymbol{x}) = g(\boldsymbol{w}^T \boldsymbol{x} + b)$$

then h *can be describe by those inner products* $\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle$.

If $\boldsymbol{x}^{(i)}$ *belongs to very high dimension* (*e.g.* $\boldsymbol{x}^{(i)} \in \mathcal{R}^{\infty}$), h *almost can't be computed easily. However, if* h *can be transfered like* $\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle$, *it can be computed very efficiently.* (*not only in SVM algorithm* !!!)

Kernel:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$$

**Lagrange duality problem must be a convex optimization problem!*

proof as follows:

original problem:

$$\min_{\boldsymbol{x}\in\mathcal{R}^n} f(\boldsymbol{x})$$

$$s.t.\ c_i(\boldsymbol{x}) \le 0, i = 1,2,\dots,k$$
$$h_j(\boldsymbol{x}) = 0, j = 1,2,\dots,l$$

generalized Lagrange function:

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{\beta}) = f(\boldsymbol{x}) + \sum_{i=1}^{k} \alpha_i c_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_j h_j(\boldsymbol{x})$$

$$\therefore p^* = \min_{\boldsymbol{x}} F(\boldsymbol{x}) = \min_{\boldsymbol{x}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0} \mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{\beta})$$

Lagrange duality:

$$d^* = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0} \min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{\beta})$$

$$\therefore \theta_D(\boldsymbol{\alpha},\boldsymbol{\beta}) = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0} \min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{\beta}) = \boldsymbol{H}(\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{x})$$

$$\boldsymbol{H}(\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{x}) = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0} \boldsymbol{H}'(\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{x})$$

here $\boldsymbol{x}$ *just a parameter vector in function* $\boldsymbol{H}'$*, so it just a linear function*:

$$\boldsymbol{H}'(\boldsymbol{\alpha},\boldsymbol{\beta};\boldsymbol{x}) = \boldsymbol{c}^T \cdot \boldsymbol{\alpha} + \boldsymbol{h}^T \cdot \boldsymbol{\beta} + \boldsymbol{f}$$

from defination, linear function must both be concave and convex!

$$\boldsymbol{H}(\lambda x + (1-\lambda)y)$$

$$= \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0}\left(\boldsymbol{H}'_1(\lambda x + (1-\lambda)y),\dots,\boldsymbol{H}'_n(\lambda x + (1-\lambda)y)\right)$$

$$\le \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0}\left(\boldsymbol{H}'_1(\lambda x) + \boldsymbol{H}'_1((1-\lambda)y),\dots,\boldsymbol{H}'_n(\lambda x) + \boldsymbol{H}'_n((1-\lambda)y)\right)$$

$$\le \lambda \cdot \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0}\left(\boldsymbol{H}'_1(x),\dots,\boldsymbol{H}'_n(x)\right) + (1-\lambda)\cdot \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0}\left(\boldsymbol{H}'_1(y),\dots,\boldsymbol{H}'_n(y)\right)$$

$$= \lambda \cdot \boldsymbol{H}(x) + (1-\lambda)\cdot \boldsymbol{H}(y)$$

$$\therefore \boldsymbol{H} = \max_{\boldsymbol{\alpha},\boldsymbol{\beta}:\alpha_i\ge 0}(\boldsymbol{H}'_1,\boldsymbol{H}'_2,\dots,\boldsymbol{H}'_n)\ \textit{must be convex}$$

(https://www.cnblogs.com/xubing-613/p/5941549.html *Convex Optimization associated*)

Lesson 8

Outline this Lesson:

1. Kernels
2. Soft margin
3. SMO algorithm

① Kernels

$$K(\boldsymbol{x},\boldsymbol{z}) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{z})$$

e.g. $Have\ \boldsymbol{X} \in \mathcal{R}\ (\boldsymbol{X}\ is\ living\ area)$

$$\boldsymbol{x} \overset{\phi}{\rightarrow} \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix} = \phi(\boldsymbol{x})$$

$$Replace\ \langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle\ with\ \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)}) \rangle\ \ (\phi(\boldsymbol{x})\text{ - }can\ be\ very\ high\ dim.)$$

$$K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x}^{(j)}) \rangle$$

e.g. 2 $(here\ n = 3)$

$$\boldsymbol{x}, \boldsymbol{z} \in \mathcal{R}^n, \quad K(\boldsymbol{x},\boldsymbol{z}) = (\boldsymbol{x}^T\boldsymbol{z})^2$$

$$\rightarrow \quad K(\boldsymbol{x},\boldsymbol{z}) = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) = \sum_{i=1}^{n}\sum_{j=1}^{n}(x_i x_j)(z_i z_j) = (\phi(\boldsymbol{x}))^T(\phi(\boldsymbol{z}))$$

$$\phi(\boldsymbol{x}) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

$Need\ O(n^2)\ time\ to\ compute\ \phi(\boldsymbol{x})$

$Only\ need\ O(\mathrm{n})\ time\ to\ compute\ K(\boldsymbol{x},\boldsymbol{z})$

e.g. 3 $(here\ n = 3)$

$$K(\boldsymbol{x},\boldsymbol{z}) = (\boldsymbol{x}^T\boldsymbol{z} + c)^2$$

$$\phi(\boldsymbol{x}) = \begin{bmatrix} x_1x_1 \\ x_2x_2 \\ x_3x_3 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \sqrt{2}x_2x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix}$$

e.g. 4 (polynomial kernel)

$$K(\boldsymbol{x},\boldsymbol{z}) = (\boldsymbol{x}^T\boldsymbol{z} + c)^d \mapsto \binom{n+d}{d} \quad \textit{features of all monomials up to degree "d"} (d \le n)$$

e.g. 5 (Gaussian kernel/radial basis function)

$$\boldsymbol{x} \mapsto \phi(\boldsymbol{x}), \qquad \boldsymbol{z} \mapsto \phi(\boldsymbol{z})$$

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle \left| \begin{array}{l} \textit{large if } \boldsymbol{x},\boldsymbol{z} \textit{ similar} \\ \textit{small if } \boldsymbol{x},\boldsymbol{z} \textit{ "dissimilar"} \end{array} \right.$$

$$\rightarrow \qquad K(\boldsymbol{x},\boldsymbol{z}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2}} \; or \; e^{-\gamma\|\boldsymbol{x}-\boldsymbol{z}\|^2} (\gamma > 0)$$

- $\exists\phi \quad st. K(\boldsymbol{x},\boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle$?

Suppose K *is a kernel. Let* $\{x^{(1)}, \dots, x^{(m)}\}$ *be given, let* $\boldsymbol{K} \in \mathcal{R}^{m\times m}$

$$\boldsymbol{K}_{ij} = K\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$$

Then, for $\forall$ *vector* $\boldsymbol{z} \in \mathcal{R}^m \quad (\boldsymbol{a}^T\boldsymbol{b} = \sum_k \boldsymbol{a}_k\boldsymbol{b}_k)$

$$\begin{aligned} \boldsymbol{z}^T\boldsymbol{K}\boldsymbol{z} &= \sum_i\sum_j \boldsymbol{z}_i\boldsymbol{K}_{ij}\boldsymbol{z}_j \\ &= \sum_i\sum_j \boldsymbol{z}_i\phi\left(\boldsymbol{x}^{(i)}\right)^T\phi\left(\boldsymbol{x}^{(j)}\right)\boldsymbol{z}_j \\ &= \sum_i\sum_j \boldsymbol{z}_i\sum_k \phi_k\left(\boldsymbol{x}^{(i)}\right)\phi_k\left(\boldsymbol{x}^{(j)}\right)\boldsymbol{z}_j \\ &= \sum_k\sum_i\sum_j \boldsymbol{z}_i\phi_k\left(\boldsymbol{x}^{(i)}\right)\phi_k\left(\boldsymbol{x}^{(j)}\right)\boldsymbol{z}_j \\ &= \sum_k\left(\sum_i \boldsymbol{z}_i\phi_k\left(\boldsymbol{x}^{(i)}\right)\right)^2 \\ &\ge 0 \end{aligned}$$

Mercer Theorem:

Let $K(\boldsymbol{x},\boldsymbol{z})$ *be given. Then* K *is a valid (Mercer) kernel (i.e.* $\exists\phi \quad st. \; K(\boldsymbol{x},\boldsymbol{z}) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{z})$*) if and only if for all* $\{x^{(1)}, \dots, x^{(m)}\}(m < \infty)$ *the kernel matrix* $\boldsymbol{K} \in \mathcal{R}^{m\times m}$ *is symmetric positive semi-definite.*

e.g. $$K(\boldsymbol{x},\boldsymbol{x}) = -1 \neq \phi(\boldsymbol{x})^T\phi(\boldsymbol{x})$$

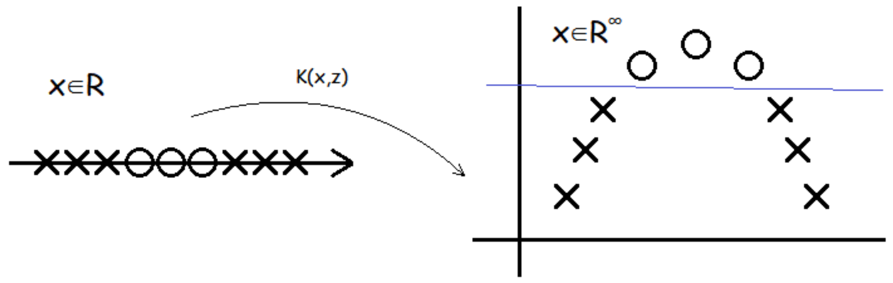
- How to use kernels?

$$Choose \quad K(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{x}^T\boldsymbol{z} + b \ \ or\ (\boldsymbol{x}^T\boldsymbol{z}+c)^d \ or\ e^{-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2}} \ or \ldots$$

$$Replace \quad \langle \boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\rangle \ with\ K\left(\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\right)$$

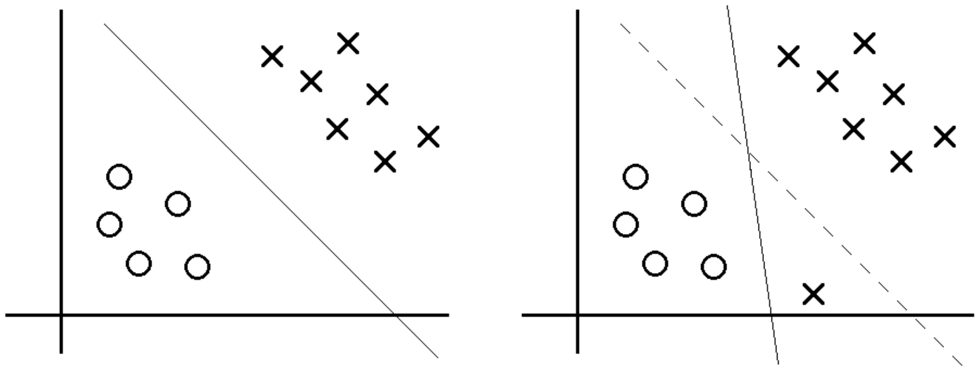
$(\text{"hiddenly"}\ \boldsymbol{x}^{(i)} \to \phi(\boldsymbol{x}^{(i)}), rather\ than\ just\ in\ SVM!)$

$$Compute\ \ result$$



② Soft margin (Regularization)

"L₁ (or L₂) norm soft margin SVM"



$$\min_{\boldsymbol{w},b,\xi}\ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i (or\ \xi_i^2)$$

$$s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \quad (i = 1,\ldots,m)$$

→ $\mathcal{L}(\boldsymbol{w},b,\xi,\boldsymbol{\alpha},\boldsymbol{r}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i f(\boldsymbol{w}) - \sum_{i=1}^{m}\alpha_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b) - 1 + \xi_i] - \sum_{i=1}^{m} r_j\xi_j$

$Accordingly, we\ obtain\ the\ dual\ problem:$

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle \boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\rangle$$

$$s.t. \quad 0 \le \alpha_i \le C \quad (i = 1, \dots, m)$$
$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

$$\rightarrow \quad \begin{aligned} \alpha_i = 0 &\Rightarrow y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b) \ge 1 \\ \alpha_i = C &\Rightarrow y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b) \le 1 \\ 0 < \alpha_i < C &\Rightarrow y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b) = 1 \end{aligned}$$

③ SMO algorithm

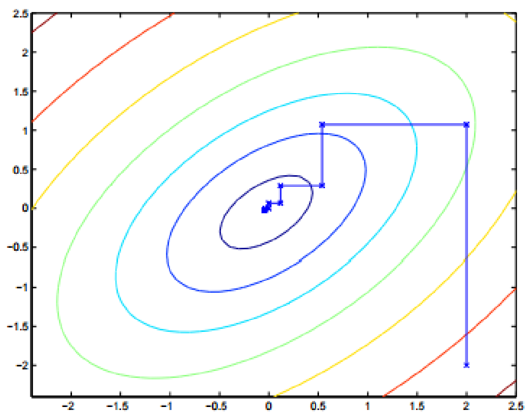
- Digression: Coordinate ascent

Considering an unconstrained optimization problem:

$$\max_{\boldsymbol{\alpha}} W(\alpha_1, \alpha_2, \dots, \alpha_m)$$

Coordinate ascent:

```
repeat till convergence:
    for $i = 1$ to $m$
        $\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_m)$
            (Hold every parameter fixed except $\alpha_i$)
```



- Sequential Minimal Optimization (SMO)

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \quad \Rightarrow \quad at\ least\ change\ 2\ \alpha_i's\ at\ a\ time$$

algorithm outline:

1. *Select* α_i, α_j $\left(\begin{matrix} heuristically\ use\ those\ can\ make\ the\ biggest \\ progress\ towards\ the\ global\ maximum \end{matrix}\right)$

2. *Hold all* $\alpha_i's$ *fixed except* α_i, α_j

3. *Reoptimize* $W(\boldsymbol{\alpha})$ *with respect to* α_i, α_j $(s.t.\ constraints)$

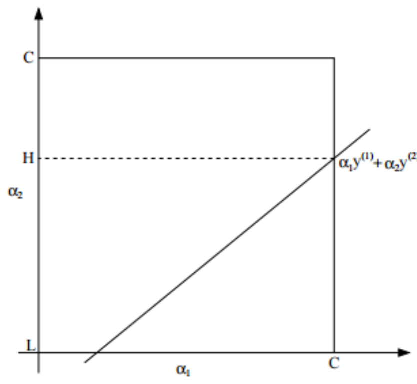
$e.g.$

$$Update\ \alpha_1, \alpha_2. \quad (Know \sum_{i=1}^{m} \alpha_i y^{(i)} = 0)$$

$$\rightarrow\ \alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)} = \zeta, \quad 0 \le \alpha_i \le C$$

$$\because\ W(\alpha_1, \alpha_2, \ldots, \alpha_m) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle$$

$$\therefore\ W(\alpha_1, \alpha_2, \ldots, \alpha_m) = W\left(\frac{\zeta - \alpha_2 y^{(2)}}{y^{(1)}}, \alpha_2, \ldots, \alpha_m \right) = A\alpha_2^2 + B\alpha_2 + C$$



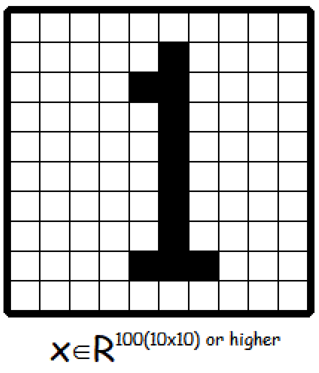
$finally, original\ problem\ changes\ into$:

$$\max_{\alpha_2}\ A\alpha_2^2 + B\alpha_2 + C$$
$$s.t. \quad L \le \alpha_2 \le H\ (in\ picture)$$

reference of how to choose α_i, α_j & update b: *John Platt's paper of SMO*

- two brief examples of using SVM

#1 Handler's Integer Recognition



$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y})^d\ or\ exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2} \right)$$

In this case, using SVM with kernels like above, even much better than the best Neural Network designed for many years!

#2 classify protein sequences

represent amino acids by alphabet (even though only 20 to human's protein XD)

$$A, \dots, Z$$

here's a protein sequence

$$BAJTIKAIBAJTAU$$

- *how to represent $\phi(\boldsymbol{x})$?*

$$\phi(\boldsymbol{x}) \in \mathcal{R}^{20^4} \to \mathcal{R}^{160000}$$

$$DP:\ \phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$$

$$\phi(\boldsymbol{x}) = \begin{array}{c} \left[\begin{array}{c} 0 \\ \vdots \\ 2 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{array}\right] \end{array} \begin{array}{c} AAAA \\ \vdots \\ BAJT \\ \vdots \\ IKAI \\ \vdots \\ ZZZZ \end{array}$$

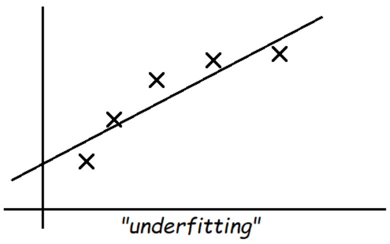
Lesson 9

Outline this Lesson:

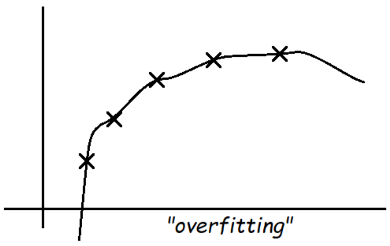
Learning Theory

1. Bias/Variance
2. Empirical Risk Minimization (ERM)
3. Union Bound & Hoeffding inequality
4. The case of finite $\mathcal{H}$(Uniform convergence)

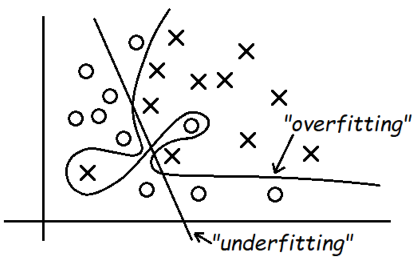
① Bias/Variance



$\theta_0 + \theta_1 x$
$"bias"$



$\theta_0 + \theta_1 x + \cdots + \theta_4 x^4$
$"variance"$



$h_{\boldsymbol{\theta}}(x) = g\left(\sum_{i=0}^{n} \theta_i x^i\right)$

$Linear(Binary)\ Classification$:

$$h_{\boldsymbol{\theta}}(x) = g(\boldsymbol{\theta}^T \boldsymbol{x}),\ g(z) = \mathbb{1}\{z \geq 0\}\ here\ y \in \{0,1\}$$

② Empirical Risk Minimization (ERM)

$assume$

$$S = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{m},\quad \left(x^{(i)}, y^{(i)}\right) \sim iid\mathcal{D}$$

$define$

$training\ error(empirical\ risk/error)\ of\ h_{\boldsymbol{\theta}}$:

$$\hat{\varepsilon}(h_{\boldsymbol{\theta}}) = \hat{\varepsilon}_S(h_{\boldsymbol{\theta}}) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{h_{\boldsymbol{\theta}}(x^{(i)}) \neq y^{(i)}\}$$

$ERM(empirical\ risk\ minimization)$:

$$\hat{\theta} = arg\min_{\boldsymbol{\theta}} \hat{\varepsilon}(h_{\boldsymbol{\theta}})$$

$more\ generally(in\ learning\ theory), assume$

$hypothesis\ function\ class\ \mathcal{H} = \{h_{\boldsymbol{\theta}} \cdot \boldsymbol{\theta} \in \mathcal{R}^{n+1}\}$

ERM: $$\hat{h} = arg\min_{h\in\mathcal{H}} \hat{\varepsilon}(h)$$

All above is just curious about training with training sets, but our ultimate goal is the prediction, which means the accuracy of what we want to classify or seize. So we define *generalization error*:

$$\varepsilon(h) = P_{(x,y)\sim\mathcal{D}}(h(x) \neq y)$$

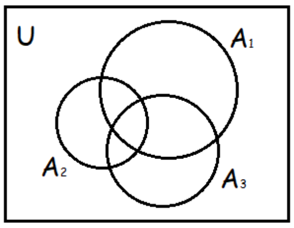
③ Union Bound & Hoeffding inequality

(Union Bound)

Let $A_1, A_2, \dots, A_k$ *be* k *events(not necessarily independent). Then*

$$P(A_1 \cup A_2 \cup \cdots \cup A_k)$$

$$\leq P(A_1) + P(A_2) + \cdots + P(A_k)$$



e.g.

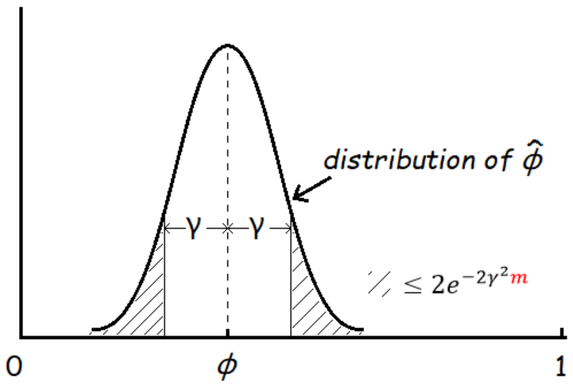
$$P(A_1 \cup A_2 \cup A_3) \leq$$
$$P(A_1) + P(A_2) + P(A_3)$$

(Hoeffding inequality)

Let $Z_1, \dots, Z_m$ *be* m *IID Bernoulli*(ϕ) *random variables(i.e.* $P(Z_i = 1) = \phi$),

$\hat{\phi} = \frac{1}{m}\sum_{i=1}^{m} Z_i$ *, and let any* $\gamma > 0$ *be fixed. Then*

$$P\left(\left|\hat{\phi} - \phi\right| > \gamma\right) \leq 2e^{-2\gamma^2 m}$$



④ The case of finite $\mathcal{H}$(Uniform convergence)

$\mathcal{H} = \{h_1, \dots, h_k\}$ *k hypotheses* $\hat{h} = \arg\min_{h_i \in \mathcal{H}} \hat{\varepsilon}(h_i)$

Strategy of proof:

(1) $\hat{\mathcal{E}} \approx \mathcal{E}$

(2) *show upper-bound on* $\mathcal{E}(\hat{h})$

assume $h_j \in \mathcal{H}$

define $Z_i = \mathbb{1}\{h_j(x^{(i)}) \neq y^{(i)}\} \in \{0,1\}$

$\therefore\ P(Z_i = 1) = \varepsilon(h_j)$ $Z_i s$ *are IID (definition)*

$$\hat{\varepsilon}(h_j) = \frac{1}{m}\sum_{i=1}^{m} Z_{i\,(mean\ \varepsilon(h_j))} = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{h_j(x^{(i)}) \neq y^{(i)}\}$$

$let\ A_j = event\ that\ |\varepsilon(h_j) - \hat{\varepsilon}(h_j)| > \gamma, by\ Hoeffding\ inequality$:

$$P(A_j) \leq 2e^{-2\gamma^2 m}$$

$$P(\exists h_j \in \mathcal{H}, |\varepsilon(h_j) - \hat{\varepsilon}(h_j)| > \gamma)\ (just\ logistical\ meaning)$$

$$= P(A_1 \cup A_2 \cup \cdots \cup A_k) \leq \sum_{i=1}^{k} P(A_i)\ \ (union\ bound)$$

$$\leq \sum_{i=1}^{k} 2e^{-2\gamma^2 m} = 2ke^{-2\gamma^2 m}\ (Hoeffding\ inequality)$$

$in\ the\ other\ words,$

$$P(\forall h_j \in \mathcal{H}, |\varepsilon(h_j) - \hat{\varepsilon}(h_j)| \leq \gamma) \geq 1 - 2ke^{-2\gamma^2 m}$$

So we have the *uniform convergence*:

$$with\ probability\ (at\ least)\ 1 - 2ke^{-2\gamma^2 m},\ \ \hat{\varepsilon}(h)\ will\ be\ within\ \gamma$$

$$of\ \varepsilon(h)\ for\ all\ h \in \mathcal{H}$$

From above, we can know that there are 3 quantities of interest: #samples m, bound γ and the probability of error. Here's some other ways to re-describe this solution where we can bound either one in terms of the other two!

(1) $Given\ \gamma, \delta\ \ what\ is\ (the\ least)\ \ m?$

→ $$\delta = 2ke^{-2\gamma^2 m}, \quad solve\ for\ m?$$

Transform to solve m with this equation, we have the *sample complexity* bound:

$$So\ long\ as\ \ m \geq \frac{1}{2\gamma^2} log\frac{2k}{\delta}, then\ with\ probability\ \ 1 - \delta, we$$

$$have\ that\ \forall h \in \mathcal{H}, |\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma.$$

In practical, almost for finite $\mathcal{H}$'s length k in Computer Science, it has "$\forall k, log\, k \leq 30$", which means we don't worry too much about m of vary kinds of hypothesis functions.

(2) $Fixed\ \delta, m\ \ solve\ for\ \gamma$

Similarly, we have the *error bound*:

With probability $1-\delta$*, we have that* $\forall h \in \mathcal{H}$,

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \le \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}} \; (\gamma)$$

→ *assume* $$|\varepsilon(h) - \hat{\varepsilon}(h)| \le \gamma \qquad (1)$$

let $$\hat{h} = \arg\min_{h\in\mathcal{H}} \hat{\varepsilon}(h), \ (training\ error) \qquad (2)$$

$$h^* = \arg\min_{h\in\mathcal{H}} \varepsilon(h) \ (generalization\ error) \qquad (3)$$

$$\varepsilon(\hat{h}) \le \hat{\varepsilon}(\hat{h}) + \gamma \qquad by\ (1)$$

$$\le \hat{\varepsilon}(h^*) + \gamma \qquad by\ (2)$$

$$\le \varepsilon(h^*) + 2\gamma \qquad by\ (1)$$

From above, we have proved a so-called *bias variance tradeoff* theorem:

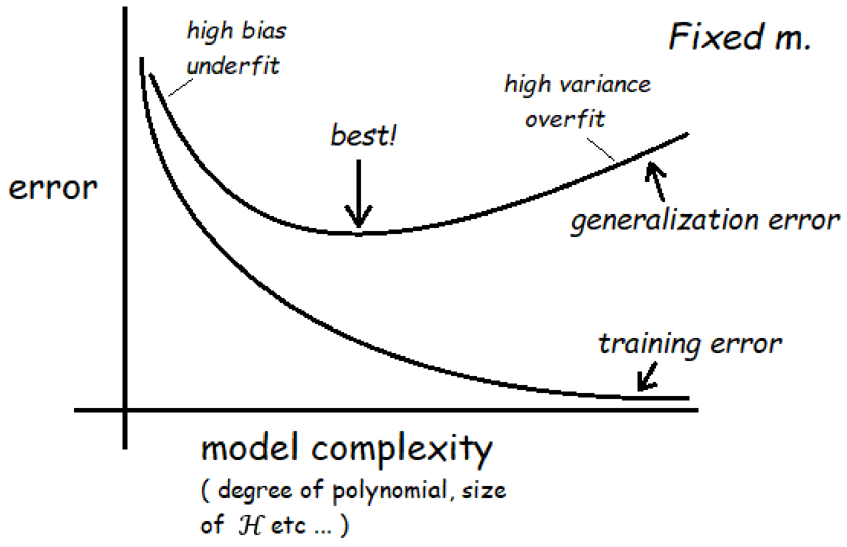
Let $|\mathcal{H}| = k$*, and let any* m, δ *be fixed. Then w.p. (at least)* $1-\delta$,

we have that

$$\varepsilon(\hat{h}) \le \left(\min_{h\in\mathcal{H}} \varepsilon(h)\right) + 2\sqrt{\frac{1}{2m}\log\frac{2k}{\delta}}$$

$\varepsilon(h^*)$ *"bias"* *"variance"*

E.g., we have some larger hypothesis class $\mathcal{H}' \supseteq \mathcal{H}$, switching to $\mathcal{H}'$ means $\min_h \varepsilon(h)$ (bias) can only decrease, meanwhile, the $2\sqrt{\cdot}$ term will increase (variance) by increasing k.



$$Corollary: Let\ |\mathcal{H}| = k, and\ let\ any\ \delta, \gamma\ be\ fixed.\ Then\ for$$

$$\varepsilon(\hat{h}) \le \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma\ \ w.p.\ (at\ least)\ 1 - \delta, it\ suffices\ that$$

$$m \ge \frac{1}{2\gamma^2} log \frac{2k}{\delta}$$

$$= O\left(\frac{1}{\gamma^2} log \frac{k}{\delta}\right)$$

PS: m is the *size* of training set, γ is the *gap* between training error & generalization error,

δ is the *probability* of this condition isn't satisfied.

Lesson 10

Outline this Lesson:

Learning Theory

1. The case of infinite $\mathcal{H}$(VC dimension)→digression: tie up some loose facts
2. Model selection: Cross validation & Feature Selection

① The case of infinite $\mathcal{H}$(VC dimension)

$assume\ \mathcal{H}\ is\ parameterized\ by\ d\ real\ numbers$

→ $e.g.\ 64\ d\ bits(in\ computer\ system)$

$\therefore\ k = |\mathcal{H}| = 2^{64d},\ suffices\ that$

$$m \geq O\left(\frac{1}{\gamma^2} log \frac{2^{64d}}{\delta}\right) = O\left(\frac{d}{\gamma^2} log \frac{1}{\delta}\right) = O_{\gamma,\delta}(d)$$

In this case, the number of training set almost linear with the parameter d, but not all the hypothesis class own this linear feature (generally, only suit for linear hypothesis).

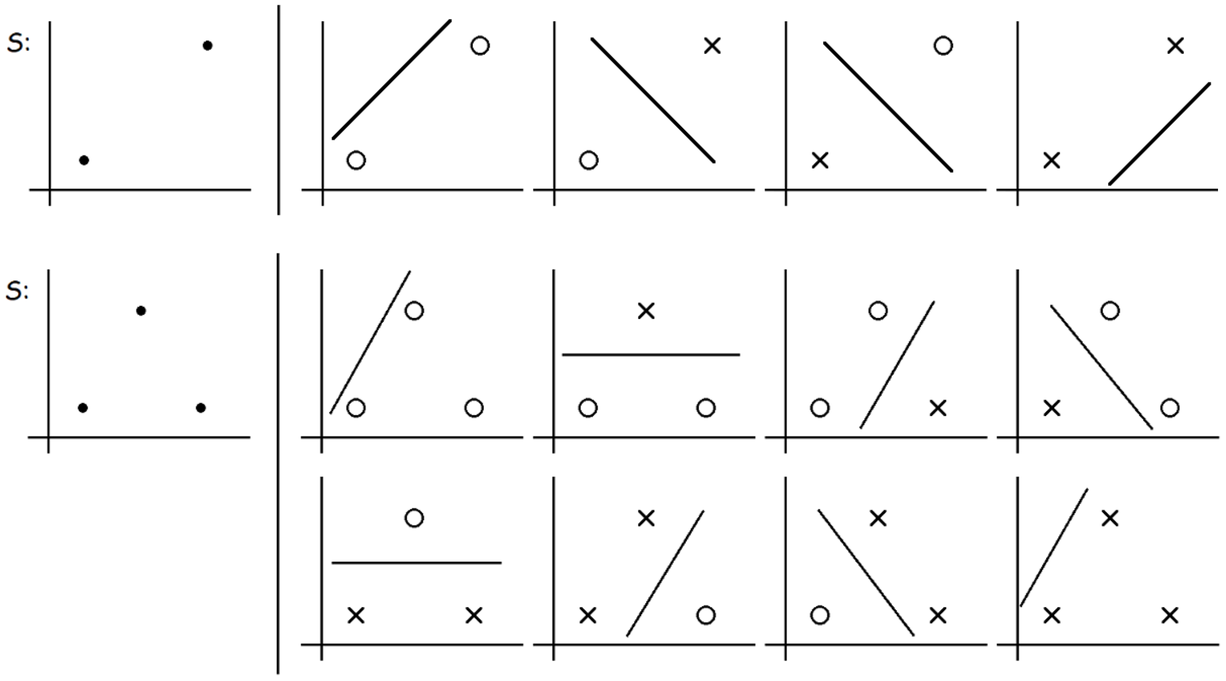
$assume$

$$Given\ a\ set\ S = \{x^{(i)}, \dots, x^{(d)}\},\quad x^{(i)} \in \mathcal{X}$$

$define$

$$\mathcal{H}\ \boldsymbol{shatters}\ S\ \ if\ \mathcal{H}\ can\ realize\ any\ labelling\ on\ it.$$

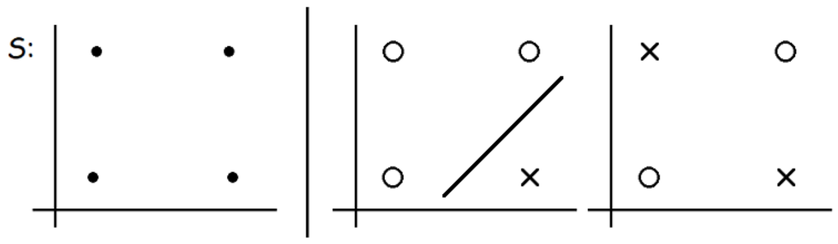
→ $e.g.\quad \mathcal{H} = \{linear\ classifiers\ in\ 2\text{-}D\}$ ($|S| = 2,3$)



$define$

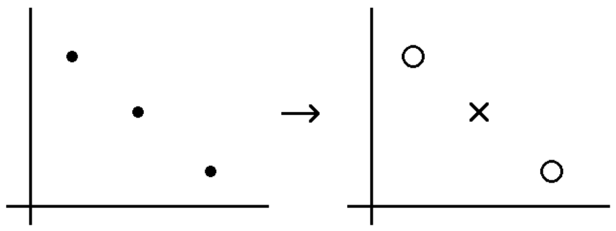
The *Vapnik-Chervonenkis* *dimension of* $\mathcal{H}$ $(\boldsymbol{VC}(\mathcal{H}))$ *is the size of the largest set shattered by* $\mathcal{H}$.

→ $e.g.\ \ \mathcal{H} = \{linear\ classifiers\ in\ 2\text{-}D\},\ VC(\mathcal{H}) = 3$



$(because\ when\ |S| \geq 4, it\ can\ never\ be\ shattered\ by\ \mathcal{H}!)$

Q: this cannot shattered by $\mathcal{H}$ *situation?*



It doesn't matter. Under the definition of the VC dimension, in order to prove that $VC(\mathcal{H})$ is at least d, we need to show only that there's *just exist* one set of size d that $\mathcal{H}$ can shatter.

$More\ generally, in\ n\text{-}dimentions,$

$$VC(\{linear\ classifiers\ in\ n\text{-}D\}) = n + 1$$

With the definition of VC dimension, we can prove a theorem of the infinite $\mathcal{H}$(the progress of proof quite complicated, here just conclusion).

$Let\ \mathcal{H}\ be\ given\ and\ let\ VC(\mathcal{H}) = d. Then\ w.p.\ (at\ least)\ 1 - \delta, we\ have\ that\ \forall h \in \mathcal{H},$

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq O\left(\sqrt{\frac{d}{m} log \frac{m}{d} + \frac{1}{m} log \frac{1}{\delta}}\right)$$

$Thus, w.p.\ (at\ least)\ 1 - \delta, we\ also\ have\ that$

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + {}_{(2)}O\left(\sqrt{\frac{d}{m} log \frac{m}{d} + \frac{1}{m} log \frac{1}{\delta}}\right)$$

$Corollary: To\ guarantee\ \varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma,\ \ w.p.\ (at\ least) 1 - \delta, it$

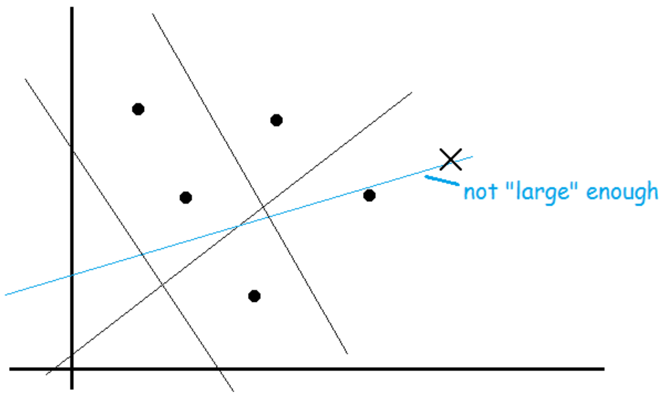
suffices that

$$m = O_{\gamma,\delta}(d)$$

It means that, if we want to minimize the gap between training error with generalization error whether the hypothesis is finite or infinite, the least number of training set must be the *same order* with (i.e. change with the same rate, cause it's just a very loose bound that has implicitly considered worse or even the worst condition) the number of hypothetical features (VC dimension).

- digression: tie up some loose facts

#1 Large margin linear classifier (e.g. SVM), i.e.,



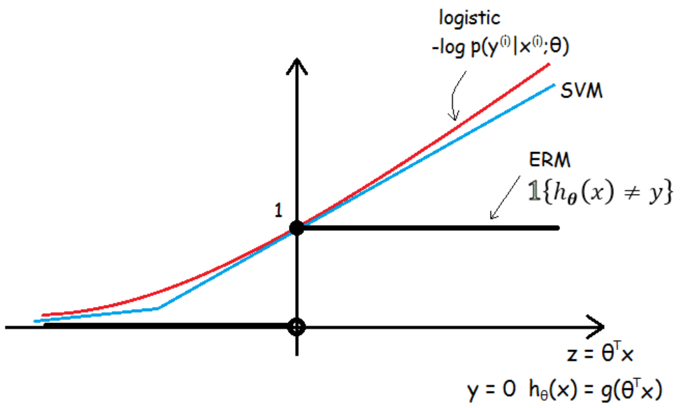
$$If \left\|x^{(i)}\right\|_2 \leq R, margin\ at\ least\ \gamma,$$

$$VC(\mathcal{H}) \leq \left\lceil \frac{R^2}{4\gamma^2} \right\rceil + 1\ (\lceil 4.23 \rceil = 5, \lceil -2.18 \rceil = -2. Upper.)$$

$$\|x\|_2^2 = \sum_{i=1}^{n} x_i^2\ \ (finite\ \#features)$$

$$\|x\|_2^2 = \sum_{i=1}^{\infty} x_i^2\ \ (infinite\ \#features)$$

#2 ERM refers to logistic regression and SVM, i.e.,

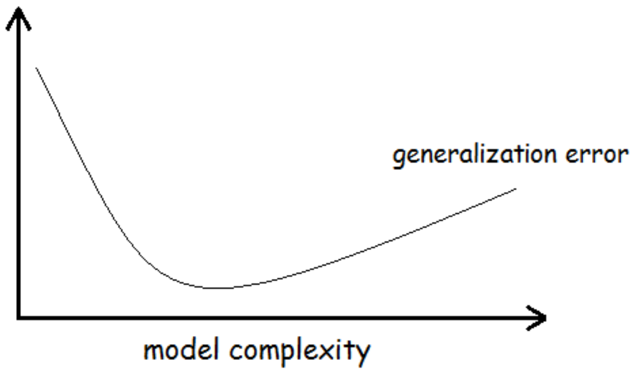

logistic
$-\log p(y^{(i)}|x^{(i)};\theta)$
SVM
ERM
$\mathbb{1}\{h_\theta(x) \neq y\}$
1
$z = \theta^T x$
$y = 0 \quad h_\theta(x) = g(\theta^T x)$

Considering only one parameter's condition, logistic regression and SVM can be both viewed as the convex approximation of ERM(can also generalize), because the linear classifiers minimizing the training error is an NP-hard problem(not convex).

② Model selection: Cross validation & Feature Selection

(1) cross validation

Relationship between generalization error and model complexity is re-drawn as:



Consider 3 kinds of problems of selecting among several models for a learning problem:

$\#1\ \ polynomial$

$$\begin{gathered}\theta_0+\theta_1 x\\ \theta_0+\theta_1 x+\theta_2 x^2\\ \vdots\\ \theta_0+\theta_1 x+\cdots+\theta_n x^n\end{gathered}$$

$\#2\ \ LWLR\ (locally\ weighted\ linear\ regression)$

$$\omega^{(i)}=e^{-\frac{\left(x^{(i)}-x\right)^2}{2\tau^2}} \qquad \tau\text{ - }bandwidth\ parameter$$

$\#3\ \ soft\ margin\ SVM\ (\ell_1\text{-}regularized\ SVM)$

$$min\ \frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{m}\xi_i$$

→ Practically, these kinds of problems we assume we have some finite set of models

$$\mathcal{M}=\{M_1,M_2,\dots,M_d\}$$

Here $\mathcal{M}$ can consist of SVM, neural network, logistic regression etcetera.

$define \qquad\qquad S\text{ - }a\ given\ training\ set$

→ *Hold-out cross validation* (or *simple cross validation*):

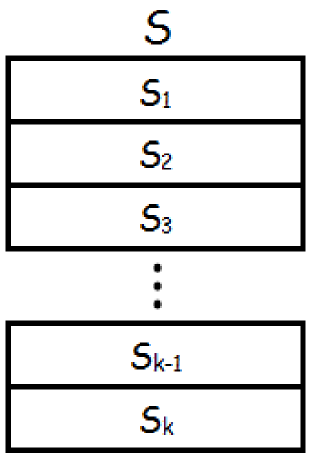
$(1)\ Split\ S\ into\ S_{train}(\sim 70\%)\ and\ S_{cv}(\sim 30\%)$

$(2)\ Train\ each\ model\ on\ S_{train}\ only, test\ on\ S_{cv}$

$(3)\ Pick\ model\ with\ lowest\ error\ on\ S_{cv}$

- Easy and fast to use especially in quite large sets, but not sufficiently enough.

→ *K-fold cross validation*:



(1) $Randomly\ split\ S\ into\ k\ pieces\ that\ each\ has\ \#m/k\ set, get\ \{S_1, S_2, \ldots, S_k\}$
(2) $For\ j = 1, \ldots, k$
$train\ each\ model\ \mathcal{M}_i\ on\ k-1\ pieces, test\ on\ the\ rest\ one\ S_j\ to\ get\ \hat{\varepsilon}_{S_j}.$
$Then\ average\ over\ the\ k\ results\ to\ get\ \hat{\varepsilon}_i$
(3) $Pick\ model\ \mathcal{M}_i\ with\ lowest\ error\ \hat{\varepsilon}_i, then$ *retrain* $on\ S\ to\ get\ the\ final\ \mathcal{M}_i$
$(PS: k = 10\ is\ common)$

- Suit for data-scarce sets (e.g. $m = 20$) and be more sufficient in using data, but inevitably accompany with more computational expense.

→ *Leave-one-out cross validation:* when $k = m$, with the extremely rare data.

(2) feature selection

Some special learning problems could be described as: the number of features n is (extremely) larger than the number of training set m ($n \gg m$), but there is only a small number of features that are "(strongly) relevant" to the learning task. It means that

$$With\ n\ features, there\ are\ 2^n\ possible\ subsets.$$

However we almost cannot search for the very features through comparing all 2^n models, here we have got some heuristic search procedure as follows.

→ *Forward search*:

```
(1) $Start\ with\ \mathcal{F} = \emptyset$
(2) $Repeat\ \{$
    a) $For\ i = 1, \ldots, n\ if\ i \notin \mathcal{F}, try\ adding\ feature\ i\ to\ \mathcal{F}, evaluate\ using$
       $cross\text{-}validation.$
    b) $Set\ \mathcal{F} = \mathcal{F} \cup \{\ best\ feature\ found\ in\ a)\ step\ \}$
    $\}\ (one\ feature\ in\ a\ loop)$
(3) $Select\ and\ output\ the\ best\ hypothesis\ found$
```

- The algorithm can be finally terminated in 2 situations: all features are in $\mathcal{F}$, or $|\mathcal{F}|$ exceeds some pre-set threshold (e.g. pre-set $\#output\text{-}features \le 10$).

PS: It's a kind of *"wrapper" model feature selection*.

Backward search:

$Start\ with\ \mathcal{F} = \{1,2, \ldots, n\},\ delete\ features\ one\ at\ a\ time.$

→ *"Filter" feature selection*:

(1) $Compute\ some\ simple\ score\ S(i)\ that\ measures\ how\ informative\ each\ feature\ x_i\ is\ about\ the\ class\ labels\ y.$

(2) $Pick\ the\ k\ features\ with\ the\ largest\ scores\ S(i)\ as\ the\ final\ output.$

E.g. $corr(x_i, y)$

Here we can define *mutual information* (mainly in text problems):

$$MI(x_i, y) = \sum_{x_i \in \mathcal{D}} \sum_{y \in \mathcal{O}} p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)p(y)}$$

$\mathcal{D}$ is all the results that x_i can get (usually $\{0,1\}$), $\mathcal{O}$ is the same. The probabilities $p(x_i, y), p(x_i), p(y)$ can be estimated from training set.

Through information theory, we can also re-write the mutual information $MI(x_i, y)$ as a *KL(Kullback-Leibler)-divergence*:

$$MI(x_i, y) = KL(p(x_i, y) \parallel p(x_i)p(y))$$

Informally, this equation gives a measure of how different the probability distributions are. It is clearly that if x_i and y are independent random variables, the KL-divergence between will be 0.

Finally when we pick top k features, we can also choose them using cross validation as measure!

Lesson 11

Outline this Lesson:

Learning Theory

1. Bayesian statistics & regularization
2. Digression: Online learning
3. Advice for applying ML algorithms

① Bayesian statistics & regularization

"Frequentist" view (e.g. Linear regression):

maximum likelihood

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

($\boldsymbol{\theta}$ *is a constant-valued but unknown parameter*)

*"*Bayesian*" view:

$p(\boldsymbol{\theta})$ - *prior distribution* $\quad$ $e.g.\ \boldsymbol{\theta} \sim \mathcal{N}(0, \tau^2 \boldsymbol{I})$

$Get\ S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

$$Calculate\ p(\boldsymbol{\theta}|S) = \frac{p(S|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(S)}$$ - *posterior distribution*

To make a new prediction on x,

$$p(y|x,S) = \int_{\boldsymbol{\theta}} p(y|x,\boldsymbol{\theta})p(\boldsymbol{\theta}|S)\, d\boldsymbol{\theta} \quad (computationally\ difficult\ to\ get)$$

$$E[y|x,S] = \int_{y} y p(y|x,S)\, dy$$

($\boldsymbol{\theta}$ *is a random variable*)

Practically,

$$\widehat{\boldsymbol{\theta}}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|S) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) p(\boldsymbol{\theta})$$

(*here we view* $\boldsymbol{\theta}$ *as a random variable, so* $p(y^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta})$ *means* $p(y^{(i)}|\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$)

To make a new prediction,

$$\hat{y} = h_{\widehat{\boldsymbol{\theta}}_{MAP}}(x) = \widehat{\boldsymbol{\theta}}_{MAP}{}^{T} x$$

e.g. linear regression problem can be regularized like:

$$\widehat{\boldsymbol{\theta}}_{lr} = \arg\min_{\boldsymbol{\theta}} \underbrace{\sum_{i=1}^{m} \left\| y^{(i)} - \boldsymbol{\theta}^T x^{(i)} \right\|^2}_{loss} + \underbrace{\lambda \|\boldsymbol{\theta}\|^2}_{prior}$$

**Regularization can always be transformed by "loss + prior" type!*

proof as follows:

$$\hat{\boldsymbol{\theta}}_{MAP} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) p(\boldsymbol{\theta})$$

$p(\boldsymbol{\theta})$ - *prior distribution, straightly associated*

$p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right)$ - *conditional pdf of set, same as* $p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$

through **Lesson 3**, we can accordingly know that

assume $\quad y^{(i)} = f\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) + \varepsilon^{(i)} \quad \left(\textit{here } f\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) \textit{ means } h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)$

where $\quad \varepsilon^{(i)} = error \sim \mathcal{N}(0, \sigma^2) \quad (\textit{by central limit theorem})$

$$P\left(\varepsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(\varepsilon^{(i)}\right)^2}{2\sigma^2}}$$

therefore, $$P\left(y^{(i)} | x^{(i)}, \boldsymbol{\theta}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)} - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right)\right)^2}{2\sigma^2}}$$

$\rightarrow$ $$y^{(i)} | x^{(i)}, \boldsymbol{\theta} \sim \mathcal{N}\left(f\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right), \sigma^2\right)$$

$$\therefore \quad \hat{\boldsymbol{\theta}}_{MAP} = \arg\max_{\boldsymbol{\theta}} \log \prod_{i=1}^{m} p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) p(\boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) + \log p(\boldsymbol{\theta})$$

$$= \arg\min_{\boldsymbol{\theta}} -\sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)} - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right)\right)^2}{2\sigma^2}} - \log p(\boldsymbol{\theta})$$

$$= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \left(y^{(i)} - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right)\right)^2 - \log p(\boldsymbol{\theta})$$

if $p(\boldsymbol{\theta}): \boldsymbol{\theta} \sim \mathcal{N}(0, \tau^2 \boldsymbol{I})$ *then* $-\log p(\boldsymbol{\theta}) \propto \frac{\|\boldsymbol{\theta}\|^2}{2\tau^2 \boldsymbol{I}} = \lambda \|\boldsymbol{\theta}\|^2 \rightarrow \ell_2$

if $p(\boldsymbol{\theta}): \boldsymbol{\theta} \sim \mathcal{La}\left(0, \frac{1}{\lambda}\right)$ *then* $-\log p(\boldsymbol{\theta}) \propto \lambda \sum_{j=0}^{n} \left|\boldsymbol{\theta}_j\right| \rightarrow \ell_1$

② Digression: Online learning

$$\begin{tikzcd}
x^{(1)} \arrow[dr] & & y^{(1)} \arrow[r] & x^{(2)} \arrow[dr] & & y^{(2)} \arrow[r] & x^{(3)} \arrow[dr] & & \cdots\cdots & x^{(m)} \arrow[dr] & & y^{(m)} \\
& \hat{y}^{(1)} \arrow[ur] & & & \hat{y}^{(2)} \arrow[ur] & & & \hat{y}^{(3)} \arrow[ur] & & & \hat{y}^{(m)} \arrow[ur] &
\end{tikzcd}$$

$Total\ online\ error\ (at\ m\ moment)$:

$$\varepsilon_{ol} = \sum_{i=1}^{m} \mathbb{1}\{\hat{y}^{(i)} \neq y^{(i)}\}$$

$e.g.\ Perceptron\ algorithm$:

$Initialize\ \boldsymbol{\theta} = \vec{0}.$

$After\ i\#\ example, update$

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \alpha \cdot (y^{(i)} - h_{\boldsymbol{\theta}}(x^{(i)}))x^{(i)}$$

Online error can also solve some incredible problem, such as when we use perceptron algorithm but $x^{(i)} \in \mathcal{R}^{\infty}$, if it is proved that positive and negative examples are separated by a margin, perceptron algorithm can still converge to digital dimensional space. Total online error is at most D^2/γ^2 $(\forall i, \|x^{(i)}\| \leq D;\ \gamma$ - $the\ geometric\ margin)$.

$(PS: proof\ of\ this\ conclusion\ are\ in$ cs229-notes6.pdf, P.g. 2~3)

*③ Advice for applying ML algorithms

Notes here are experiential for applying, might not suitable for researching. Some of these advice is debatable, but still make a dent in parts of problems.

(1) diagnostics for debugging learning algorithms

#1 Someone used Bayesian logistic regression to build up an anti-spam, which contains a small set of words as features, but got unacceptable 20% (or more) test error.

$Bayesian\ logistic\ regression$:

$$\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p\left(y^{(i)} \middle| x^{(i)}, \boldsymbol{\theta}\right) - \lambda \|\boldsymbol{\theta}\|^2$$

a) Common approach: Just try in different ways (quite randomly)

- Try getting more training examples.
- Try a smaller set of features.
- Try a larger set of features.
- Try changing the features: Email header vs. email body features.
- Run GD for more iterations.
- Try Newton's method.
- Use a different value for λ.
- Try using an SVM.

This approach is time-consuming, gambly, sometimes it might work though.

b) Better approach: Diagnose and repair

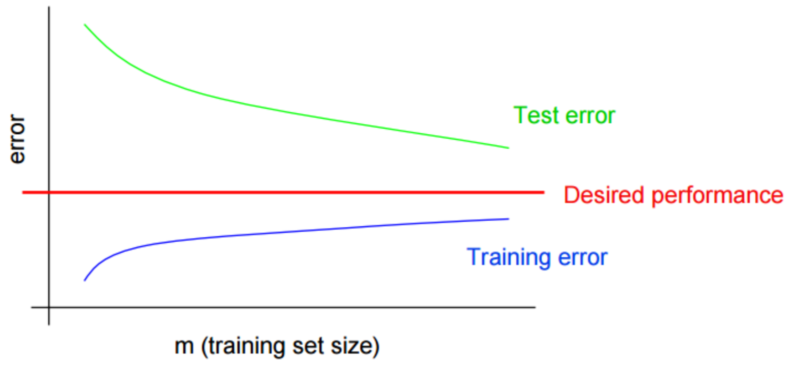
- Run diagnostics to figure out what the problem truly is.
- Fix (or precisely optimize) whatever the problem is.

→ 1) Suppose the problem is either (*bias vs. variance diagnostics*):

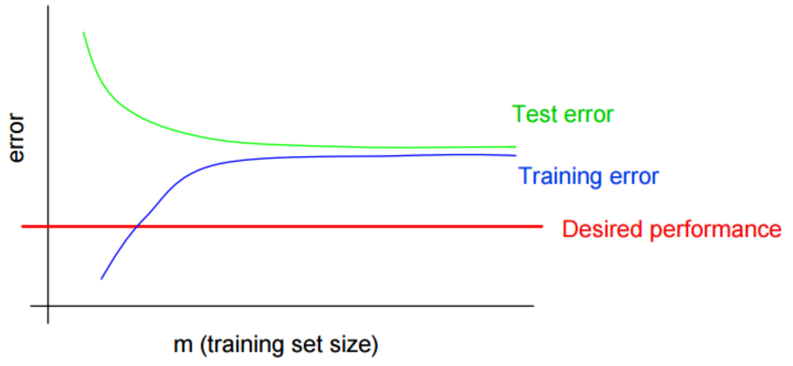
- Overfitting (high variance).
- Too few features to classify spam (high bias).

2) Diagnostic:

- Variance: Training error will be much lower than test error.



- Bias: Training error will also be high (although both are very closed of each other).



here, we can know what fixes to try:

- Try getting more training examples.	Fixes high variance.
- Try a smaller set of features.	Fixes high variance.
- Try a larger set of features.	Fixes high bias.
- Try email header features.	Fixes high bias.

For other problems, it’s usually up to ingenuity to design effective and unique diagnostics to figure out what’s wrong.

#2 Someone uses Bayesian logistic regression to build up an anti-spam, which gets 2% error on spam, and 2% error on non-spam. (Unacceptable error on non-spam.) He also applies SVM using a linear kernel, which gets 10% error on spam, and 0.01% error on non-spam (acceptable). Because of some reasons (like computational efficiency etc.), he chooses to use the former algorithm.

We can figure out lots of potential improvements (or questions) like:

- Is the algorithm converging?
- Optimize the right function?
- If using weight, need weights higher for non-spam than spam?

- Correct value for λ in Bayesian logistic regression?
- Correct value for C in SVM? …

And whatever reason, we really want to deploy Bayesian logistic regression, even though SVM does much better for this application.

Q: What to do next?

Say

$\boldsymbol{\theta}_{SVM}$ - *the parameters learned by an SVM*

$\boldsymbol{\theta}_{BLR}$ - *the parameters learned by Bayesian logistic regression*

$a(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \sum_i w^{(i)} \cdot \mathbb{1}\{h_{\boldsymbol{\theta}}(x^{(i)}) = y^{(i)}\}$ - *the weighted accuracy*

$\boldsymbol{\theta}_{SVM}$ *outperforms* $\boldsymbol{\theta}_{BLR}$*. So*

$$a(\boldsymbol{\theta}_{SVM}) > a(\boldsymbol{\theta}_{BLR})$$

BLR tries to maximize:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \log p(y^{(i)} | x^{(i)}, \boldsymbol{\theta}) - \lambda \|\boldsymbol{\theta}\|^2$$

Diagnostics:

$$J(\boldsymbol{\theta}_{SVM}) > J(\boldsymbol{\theta}_{BLR})?$$

- Problem is with optimization algorithm.

$$\begin{cases} a(\boldsymbol{\theta}_{SVM}) > a(\boldsymbol{\theta}_{BLR}) \\ J(\boldsymbol{\theta}_{SVM}) > J(\boldsymbol{\theta}_{BLR}) \end{cases}$$

It means that BLR tries to maximize J but fails, because using SVM's parameters J can be larger than BLR's. Obviously algorithm doesn't converge well.

- Problem is with optimization objective function.

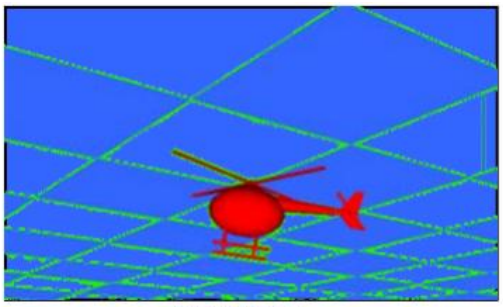
$$\begin{cases} a(\boldsymbol{\theta}_{SVM}) > a(\boldsymbol{\theta}_{BLR}) \\ J(\boldsymbol{\theta}_{SVM}) \le J(\boldsymbol{\theta}_{BLR}) \end{cases}$$

It shows that the SVM, which does worse on $J(\boldsymbol{\theta})$, actually does better on weighted accuracy $a(\boldsymbol{\theta})$. In other words, maximizing $J(\boldsymbol{\theta})$ doesn't really correspond that well to maximizing $a(\boldsymbol{\theta})$. This confirms that if you care about $a(\boldsymbol{\theta})$, $J(\boldsymbol{\theta})$ is the wrong function to be maximizing.

here, we can know what fixes to try:

- Run GD for more iterations.	Fixes optimization algorithm.
- Try Newton's method.	Fixes optimization algorithm.
- Use a different value for λ.	Fixes optimization objective.
- Try using an SVM.	Fixes optimization objective.

#3 Andrew Ng's project on reinforcement learning: The Stanford Autonomous Helicopter.



Simulator

Build a simulator of helicopter → Choose a cost function $J(\boldsymbol{\theta})$ → minimize $J(\boldsymbol{\theta})$ to get $\boldsymbol{\theta}_{RL}$

Say

$$J(\boldsymbol{\theta}) = \|\boldsymbol{x} - \boldsymbol{x}_{desired}\|^2 \quad \boldsymbol{x} \text{ - } helicopter\ position$$

$$\boldsymbol{\theta}_{RL} = arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Suppose the resulting controller parameters $\boldsymbol{\theta}_{RL}$ gives much worse performance than human pilot. *What to do next?*

- Improve simulator?
- Modify cost function $J(\boldsymbol{\theta})$?
- Modify RL algorithm?

$Suppose\ that$:

1. $The\ helicopter\ simulator\ is\ accurate.$
2. $The\ RL\ algorithm\ correctly\ controls\ the\ helicopter\ (in\ simulation)\ so\ as\ to\ minimize\ J(\boldsymbol{\theta}).$
3. $Minimizing\ J(\boldsymbol{\theta})\ corresponds\ to\ correct\ autonomous\ flight.$

$Then: The\ learned\ parameters\ \boldsymbol{\theta}_{RL}\ should\ fly\ well\ on\ the\ actual\ helicopter.$

Diagnostics:

- If $\boldsymbol{\theta}_{RL}$ flies well in simulation, but not in reality. → Problem in simulation.

Let $\boldsymbol{\theta}_{human}$ be the human control policy.

- If $J(\boldsymbol{\theta}_{human}) < J(\boldsymbol{\theta}_{RL})$, cost function is failed to minimize. → Problem in RL algorithm.
- If $J(\boldsymbol{\theta}_{human}) \geq J(\boldsymbol{\theta}_{RL})$, it means that minimizing J doesn't correspond to good autonomous flight. → Problem in cost function.

Conclusions:

We've got 3 different diagnostics of different learning problems. Practically, it's just a little bit useful in some ways because it is quite often to come up with our own diagnostics (via ingenuity) to figure out what's happening in different learning problems.

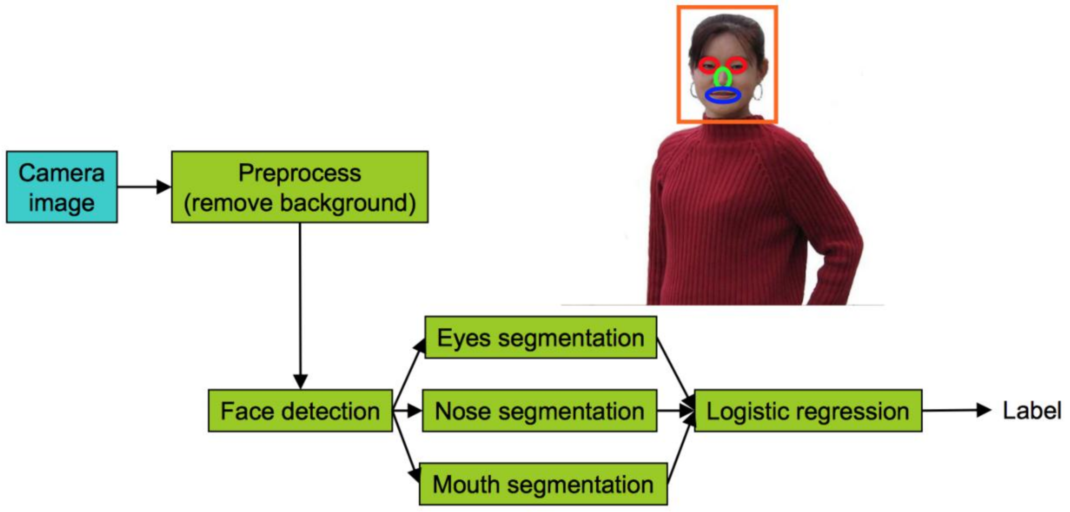
Moreover, even if a learning algorithm is working well, we can also run diagnostics to help us understand the algorithm further. This is useful for:

- Understanding application problems further, even getting an intuitive understand of what works and what doesn't work.
- Writing research papers: enrich insight about the problem and justify research claims.
- Making sense and being convincible when explain the core algorithm to others.

→ So, we need *error analysis* to understand what sources of error are!

(2) error analysis & ablative analysis

a) error analysis



E.g. A pipeline of Face recognition from images (not quite formal)

Suppose we have a pipeline like above picture. Now the recognition accuracy of overall system is only 85%.

What we can do in brief, plug in *ground-truth* for each component, which means the perfect output of each component however it’s got (like using PS, coding by hand etc.), see how accuracy changes. Perhaps the changes can be noted as:

Component	Accuracy
Overall system	85%
Preprocess (remove background)	85.1%
Face detection	91%
Eyes segmentation	95%
Nose segmentation	96%
Mouth segmentation	97%
Logistic regression	100%

And then, find the several maximum gaps between and improve them first. In this case, we know that we have most room for improvement in face detection and eyes segmentation.

b) ablative analysis

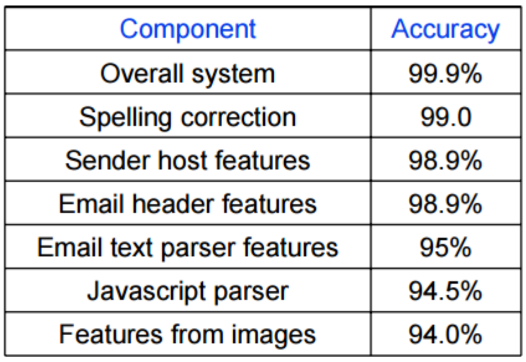
Compared with error analysis, ablative analysis is the opposite strategy which tries to explain the difference between some baseline and current performance.

E.g. A good anti-spam classifier by adding lots of clever features to logistic regression:

- Spelling correction.
- Sender host features.
- Email header features.
- Email text parser features.
- Javascript parser.
- Features from images.

Q: How much did each of these components really help?

Suppose we apply a simple logistic regression without any clever features get 94% performance. In ablative analysis, just remove components from the system **one at a time** (not only one-by-one, one-out-others-in if suitable is ok) to see how it breaks. Perhaps the changes can be noted as:



Component	Accuracy
Overall system	99.9%
Spelling correction	99.0
Sender host features	98.9%
Email header features	98.9%
Email text parser features	95%
Javascript parser	94.5%
Features from images	94.0%

[baseline]

Accordingly, we can also find out the several maximum gaps between and figure out what is the core component of the system. In this case, it shows that email text parser features contribute for the most of the improvement.

From the discussion above, we can reorganize the both analysis briefly as:

Type	**Error Analysis**	**Ablative Analysis**
How	1. Select suitable components 2. Make each "perfect" (*ground-truth*), record the changes 3. Find out the most improving parts, focus on those	1. Select suitable components 2.Remove each in some rules, record the changes 3. Find out the most decreasing parts, pay attention on those
Situation	Improve the algorithm or debug it.	Analysis improved algorithm, find out the core change(s).

(3) how to get started on a learning problem

There are two typical approaches to applying learning algorithms:

#1 Design very carefully, then implement it.

Benefit: Nicer, perhaps more scalable algorithms. May come up with new, elegant, learning algorithms; contribute to basic research in machine learning.

Risk: *Premature* (*statistical*) *optimization*.

#2 Build a quick-and-dirty prototype, diagnose and fix it.

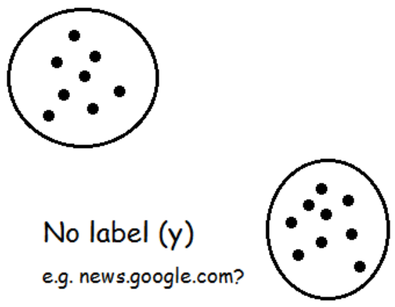
Benefit: Will often get application problem working more quickly. Faster time to market.

Risk: Time-consuming debugging and testing (if not quite experienced in some part).

Lesson 12

Outline this Lesson:
Unsupervised Learning
1. Clustering (k-means)
2. Mixture of Gaussians
3. Jensen's inequality
4. EM (Expectation-Maximization) algorithm

① Clustering (k-means)



K-means algorithm:

$Input\ training\ set\ \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}, \quad x^{(i)} \in \mathcal{R}^n.$
1. $Initialize$ $\boldsymbol{cluster\ centriods}$ $\mu_1, \mu_2, \dots, \mu_k \in \mathcal{R}^n\ randomly.$
2. $Repeat\ until\ convergence$:

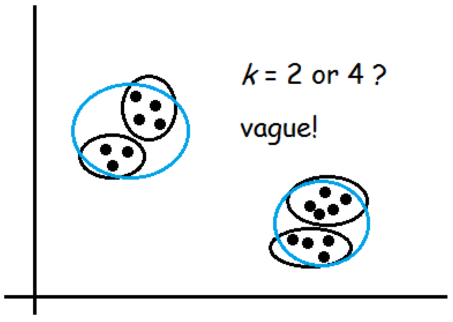
$$i)\ Set\ c^{(i)} := \arg\min_j \left\|x^{(i)} - \mu_j\right\| \ (or\ \left\|x^{(i)} - \mu_j\right\|^{(2)}).$$

$$ii)\ \mu_j := \frac{\sum_{i=1}^{m} \mathbb{1}\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} \mathbb{1}\{c^{(i)} = j\}}.$$

This algorithm guarantees convergence by *distortion function*:

$$J(c, \mu) = \sum_{i=1}^{m} \left\|x^{(i)} - \mu_{c^{(i)}}\right\|^2$$

Here $c^{(i)}$ is a label of $x^{(i)}$ to decide one example $x^{(i)}$ belongs to which cluster centroid related. Moreover, k (# clusters) is assumed already known, even though it is quite a problem to choose practically. If we are just curious about the minimum of $J(c, \mu)$, we can choose this parameter "randomly" and choose that can minimize J. (I.e. the exact k may vague as below.)

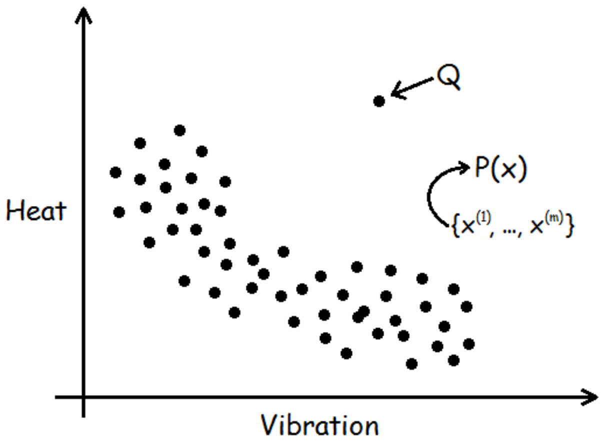


Algorithm above do those things in two steps: (i) "Assigning" each training example $x^{(i)}$ to the closest cluster centroid μ_j. (ii) Moving each μ_j to the mean of the points assigned to it.

As a matter of fact, *k*-means is exactly coordinate descent on $J(c,\mu)$. Because $J(c,\mu)$ is a non-convex function, *k*-means can always converge, but not guarantee to converge to the global minimum. (Initialize cluster centroids randomly and choose the minimum J may help.)

→ Density estimation

Here's a practical example, we need to detect whether an aeroengine is broken or not (briefly) in such two potential factors: vibration and heat. The data all we have and the detected one, point *Q*, are shown as:

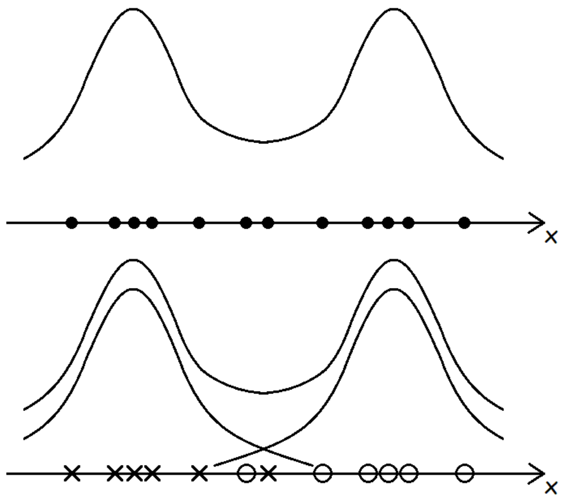


We can divide all training set into several clusters by *k*-means, then check *Q* belongs to none of them to figure out *Q* is the one need further inspected. Some problem like this are called *anormaly detection*. For brevity, we can describe the algorithm to solve those as:

1. $Do\ density\ estimation\ to\ data\ set\ \{x^{(1)},\dots,x^{(m)}\}\ to\ build\ a\ model, get\ P(x).$
2. $Calculate\ P(Q).$
3. $Confirm\ the\ Q\ is\ anormal\ or\ not.$

In this part, we are curious about *step 1*: how to get $P(x)$, so called *density estimation*. Considering this situation, we can hardly build a model with all the basic distributions we have already known like Gaussian or Poisson etc. So a usual way to build complex models like this is to apply the mixture of Gaussians.

② Mixture of Gaussians



$There's\ a\ latent\ (hidden/unobserved)\ random\ variable\ z\ in\ training\ data. And$ $x^{(i)}, z^{(i)}\ have\ a\ joint\ distribution\ (PS{:}\ k\ is\ \#\ z^{(i)}s\ can\ take\ on){:}$

$$p\left(x^{(i)}, z^{(i)}\right) = p\left(x^{(i)} \middle| z^{(i)}\right) p\left(z^{(i)}\right)$$

$$Here\ z^{(i)} \sim Multinomial(\phi) \quad \left(\phi_j \geq 0, \textstyle\sum_{j=1}^{k} \phi_j = 1\right)$$

$$x^{(i)} | z^{(i)} = j \sim \mathcal{N}(\mu_j, \Sigma_j)$$

$If\ we\ knew\ z^{(i)}s,$

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{m} \log p\left(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma\right)$$

$Maximizing\ it, we\ have{:}$

$$\phi_j = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\{z^{(i)} = j\}$$

$$\mu_j = \frac{\sum_{i=1}^{m} \mathbb{1}\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} \mathbb{1}\{z^{(i)} = j\}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} \mathbb{1}\{z^{(i)} = j\} \left(x^{(i)} - \mu_j\right)\left(x^{(i)} - \mu_j\right)^T}{\sum_{i=1}^{m} \mathbb{1}\{z^{(i)} = j\}}$$

As it's shown above, it is quite similar with GDA in **Lesson 5** (here each Gaussian may vary because of the difference of Σ_j). However, the fact is that we actually do *not* know $z^{(i)}s$. In order to satisfy this condition, we need special type of EM algorithm to help.

→ *EM algorithm (with mixture of Gaussians model)*:

$repeat\ until\ convergence{:}$

$(E\text{-}step)\ For\ each\ i, j, let$

$$w_j^{(i)} := p\left(z^{(i)} = j \middle| x^{(i)}; \phi, \mu, \Sigma\right)$$

$(M\text{-}step)\ Update\ the\ parameters$

$$\phi_j := \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}$$

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} \left(x^{(i)} - \mu_j\right)\left(x^{(i)} - \mu_j\right)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

We can apply Bayes' Rule to expand *E-step* as:

$$w_j^{(i)} := p\left(z^{(i)} = j \middle| x^{(i)}; \phi, \mu, \Sigma\right)$$
$$= \frac{p\left(x^{(i)} \middle| z^{(i)} = j; \mu, \Sigma\right) p\left(z^{(i)} = j; \phi\right)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$
$$= \frac{\frac{1}{(2\pi)^{n/2} \left|\Sigma_j\right|^{1/2}} exp\left(-\frac{1}{2}\left(x^{(i)} - \mu_j\right)^T \Sigma_j^{-1}\left(x^{(i)} - \mu_j\right)\right) \cdot \phi_j}{\sum_{l=1}^{k} \frac{1}{(2\pi)^{n/2} |\Sigma_l|^{1/2}} exp\left(-\frac{1}{2}(x^{(i)} - \mu_l)^T \Sigma_l^{-1}(x^{(i)} - \mu_l)\right) \cdot \phi_l}$$
$$\left(n = \#features\ in\ x^{(i)}, k = \#\ z^{(i)}s\ can\ take\ on\right)$$

Despite of computational difficulty, we can update $w^{(i)}$ easily. Practically, *k*-means can also be described as EM algorithm. Moreover, EM algorithm converges to the local minimum instead of the global minimum.

Q: Does EM algorithm can always converge like k-means?

- Need some other factors first, like Jensen's inequality.

③ Jensen's inequality

Jensen's inequality actually is the property of convex functions or distributions, and it has lots of different forms in different ways. Here's one useful form of them:

$Let\ f\ be\ a\ convex\ function\ (\text{i.e. } f''(x) \geq 0), and\ let\ X\ be\ a\ random\ variable. Then$

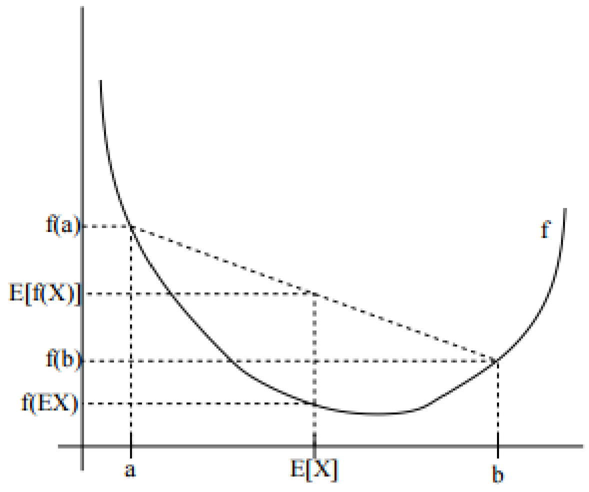
$$f(EX) \leq E[f(X)] \quad (f(EX) = f(E[X]))$$

$Moreover, if\ f\ is\ strictly\ convex\ (\text{i.e. } f''(x) > 0), then$

$$f(EX) = E[f(X)] \iff X = E[X] \quad (i.e.\ X\ is\ a\ constant)$$

$(On\ the\ contrary, all\ the\ inequalities\ just\ need\ reverse\ if\ f\ is\ concave!)$

We can draw a picture to help us understand and remember it:



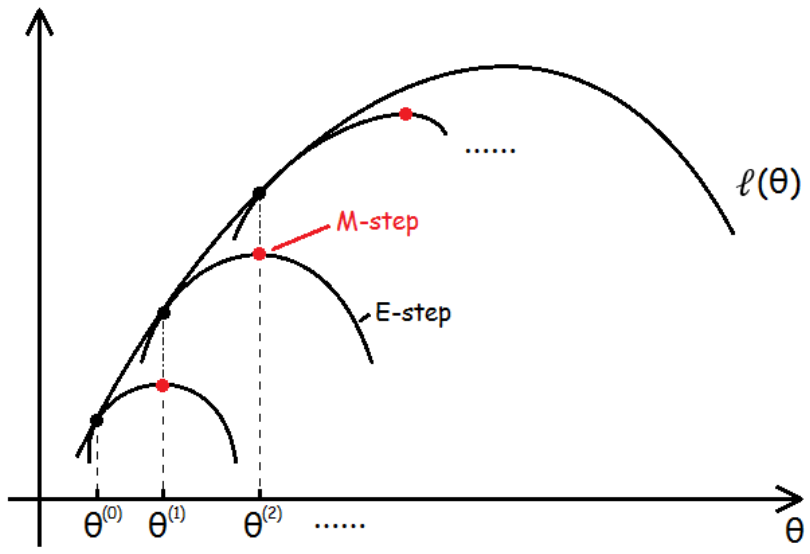
④ EM (Expectation-Maximization) algorithm

From the mixture of Gaussians part, we can conclude it and generalize such problem as follows:

1. $Have\ training\ set\ \{x^{(1)}, \dots, x^{(m)}\}\ with\ some\ latent\ variables\ z^{(i)}.$
2. $Build\ model\ and\ want\ to\ find\ parameters\ for\ P(x, z; \theta).$
3. $Want\ to\ maximize\ the\ likelihood:$

$$\ell(\theta) = \sum_{i=1}^{m} \log p(x^{(i)}; \theta)$$
$$= \sum_{i=1}^{m} \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)$$

Now the problem is transferred to find suitable θ to maximize the likelihood. The general EM algorithm gives an efficient method instead of maximizing $\ell(\theta)$ straightly: (i) repeatedly construct a lower-bound on $\ell(\theta)$ (*E-step*); (ii) optimize that lower-bound (*M-step*). The overall process can also be drawn as a picture:

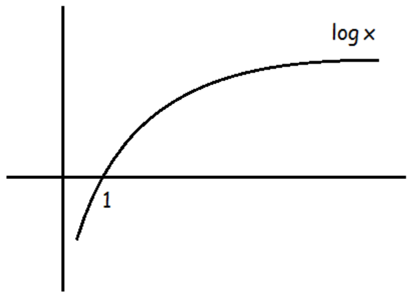


For each i, let Q_i be some distribution over the $z^{(i)}s$ ($\sum_z Q_i(z) = 1, Q_i(z) \geq 0$), we get:

$$\ell(\theta) = \sum_i \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)$$
$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$
$$\left(\sum_i \log \underset{z^{(i)} \sim Q_i}{E} \left[\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \atop use\ Jensen's\ quality\ (concave) \right)$$
$$\geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

Here we use some little tricks:

1. $If\ z \sim p, we\ have\ g(z). Then$
$$E[g(z)] = \sum_z p(z)g(z)$$
2. $log\, E[x] \geq E[log\, x]$



According to the above equations, we know that for ***any*** Q_i satisfied those given conditions, we have a lower-bound on $\ell(\theta)$.

Q: How to choose the very Q_i*?*

- Want fit the inequation's equality sign, in other words, make the lower-bound **tight**!

$$\rightarrow \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = constant \ \ (for\ all\ values\ of\ z^{(i)})$$

$$\therefore\ Q_i(z^{(i)}) \propto p(x^{(i)}, z^{(i)}; \theta) \ \ \rightarrow \ \ \Sigma_z Q_i(z^{(i)}) \propto \Sigma_z p(x^{(i)}, z; \theta)$$

$$\rightarrow\rightarrow \frac{Q_i(z^{(i)})}{\Sigma_z Q_i(z^{(i)})} = \frac{p(x^{(i)}, z^{(i)}; \theta)}{\Sigma_z p(x^{(i)}, z; \theta)}$$

Since we know $\Sigma_z Q_i(z) = 1$,

$$\therefore \qquad Q_i(z^{(i)}) = \frac{p(x^{(i)}, z^{(i)}; \theta)}{\Sigma_z p(x^{(i)}, z; \theta)}$$
$$= \frac{p(x^{(i)}, z^{(i)}; \theta)}{p(x^{(i)}; \theta)}$$
$$= p(z^{(i)} | x^{(i)}; \theta) \ \ (conditional\ probablility)$$

Thus, we simply set the $Q_i s$ to be the posterior distribution of the $z^{(i)}s$, given $x^{(i)}$ and parameterized by θ.

Finally, we can give out the *generalized EM (Expectation-Maximization) algorithm*:

```
$repeat\ until\ convergence:$
    $(E\text{-}step)\ For\ each\ i, set$
        $Q_i(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta)$
    $(M\text{-}step)\ Update\ the\ parameters$
        $\theta := arg\max_\theta \sum_i \sum_{z^{(i)}} Q_i(z^{(i)})\, log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$
```

Lesson 13

Outline this Lesson:

EM

1. Mixture of Gaussians
2. Mixture of naive Bayes
3. Digression: Gaussians
4. Factor analysis

① EM: Mixture of Gaussians

- Another perspective of EM (in optimization)

define

$$J(\theta, Q) = \sum_i \sum_{z^{(i)}} Q_i\left(z^{(i)}\right) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i\left(z^{(i)}\right)}$$

then we have $\ell(\theta) \geq J(\theta, Q)$

EM algorithm does co-ordinate ascent on J via:

$E\text{-}step$: $Maximize$ w.r.t. Q

$M\text{-}step$: $Maximize$ w.r.t. θ

- Mixture of Gaussians

According to the process above in **Lesson 12**, we know that in *E-step*:

$$w_j^{(i)} = Q_i\left(z^{(i)} = j\right) = p\left(z^{(i)} = j \middle| x^{(i)}; \phi, \mu, \Sigma\right)$$

$$= \frac{p\left(x^{(i)} \middle| z^{(i)} = j; \mu, \Sigma\right) p\left(z^{(i)} = j; \phi\right)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

$$x^{(i)} | z^{(i)} = j \sim \mathcal{N}(\mu_j, \Sigma_j)$$

$$z^{(i)} \sim Multinomial(\phi) \quad \left(\phi_j \geq 0, \sum_{j=1}^{k} \phi_j = 1\right)$$

So, *M-step*:

$$\max_{\phi,\mu,\Sigma} \sum_i \sum_{z^{(i)}} Q_i\left(z^{(i)}\right) \log \frac{p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma)}{Q_i\left(z^{(i)}\right)}$$

$$= \max_{\phi,\mu,\Sigma} \sum_i \sum_j w_j^{(i)} \log \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p\left(z^{(i)} = j; \phi\right)}{w_j^{(i)}}$$

$$= \max_{\phi,\mu,\Sigma} \sum_i \sum_j w_j^{(i)} \log \frac{exp(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)) \cdot \phi_j}{(2\pi)^{n/2} \left|\Sigma_j\right|^{1/2} \cdot w_j^{(i)}}$$

define

$$G(\phi, \mu, \Sigma) = \sum_i \sum_j w_j^{(i)} \log \frac{exp(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)) \cdot \phi_j}{(2\pi)^{n/2} \left|\Sigma_j\right|^{1/2} \cdot w_j^{(i)}}$$

#1 maximize $G(\phi, \mu, \Sigma)$ with respect to μ_l

$$\boldsymbol{\nabla}_{\mu_l} G(\phi, \mu, \Sigma) = \boldsymbol{\nabla}_{\mu_l} \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} (-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j))$$

$$= \sum_{i=1}^{m} w_l^{(i)} \boldsymbol{\nabla}_{\mu_l} (-\frac{1}{2}(-2\mu_l^T \Sigma_l^{-1} x^{(i)} + \mu_l^T \Sigma_l^{-1} \mu_l))$$

$$= \sum_{i=1}^{m} w_l^{(i)} (\Sigma_l^{-1} x^{(i)} - \Sigma_l^{-1} \mu_l)) \overset{set}{\longrightarrow} \vec{0}$$

→ Compute the update rule of μ_l:

$$\mu_l := \frac{\sum_{i=1}^{m} w_l^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_l^{(i)}}$$

#2 maximize $G(\phi, \mu, \Sigma)$ with respect to ϕ_j

$$\because \boldsymbol{\nabla}_{\phi_j} G(\phi, \mu, \Sigma) = \boldsymbol{\nabla}_{\phi_j} \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} (\text{"}constant\ to\ \phi_j\text{"} + log\ \phi_j)$$

$$= \boldsymbol{\nabla}_{\phi_j} \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)}\ log\ \phi_j$$

$$\left(\phi_j \geq 0, \textstyle\sum_{j=1}^{k} \phi_j = 1\right)$$

Considering the constraint of ϕ_j, we can construct the Lagrangian ($\phi_j \geq 0$ is naturally satisfied because of log function):

$$\mathcal{L}(\phi) = \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)}\ log\ \phi_j + \beta(\sum_{j=1}^{k} \phi_j - 1)$$

→ $$\frac{\partial}{\partial \phi_j} \mathcal{L}(\phi) = \frac{1}{\phi_j} \sum_{i=1}^{m} w_j^{(i)} + \beta \overset{set}{\longrightarrow} \vec{0}$$

$$\phi_j = \frac{\sum_{i=1}^{m} w_j^{(i)}}{-\beta} \quad (\phi_j \propto \sum_{i=1}^{m} w_j^{(i)})$$

Similar to the equation in **Lesson 12**, we have $\sum_{j=1}^{k} \phi_j = 1$ so it's easily to figure out the Lagrange multiplier β

$$-\beta = \sum_{j=1}^{k} \sum_{i=1}^{m} w_j^{(i)} = \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} = \sum_{i=1}^{m} 1 = m$$

→ Compute the update rule of ϕ_j:

$$\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)}$$

#3 maximize $G(\phi,\mu,\Sigma)$ with respect to Σ_j

$$\nabla_{\Sigma_j} G(\phi,\mu,\Sigma) = -\frac{1}{2}\nabla_{\Sigma_j}\sum_{i=1}^{m} w_j^{(i)}((x^{(i)}-\mu_j)^T\Sigma_j^{-1}(x^{(i)}-\mu_j) + log|\Sigma_j| + \text{"C to }\Sigma_j\text{"})$$

$$= \frac{1}{2}\Sigma_j^{-T}\sum_{i=1}^{m} w_j^{(i)}((x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T\Sigma_j^{-T} - I)$$

$$(\frac{\partial tr\, AX^{-1}B}{\partial X} = -X^{-T}A^TB^TX^{-T}, \frac{\partial|X|}{\partial X} = X^{*T} = |X|X^{-T})$$

→ $$\nabla_{\Sigma_j} G(\phi,\mu,\Sigma) \xrightarrow{set} \vec{0}$$

$$\sum_{i=1}^{m} w_j^{(i)}(x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T = \Sigma_j(\sum_{i=1}^{m} w_j^{(i)}) \quad (right\ multiply\ by\ \Sigma_j^T)$$

→ Figure out the update rule of Σ_j:

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)}\left(x^{(i)}-\mu_j\right)\left(x^{(i)}-\mu_j\right)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

② EM: Mixture of naive Bayes

Here's a Text Clustering example (Multi-variant Bayes event model)

Assume

$$training\ set\ \{x^{(1)},\dots,x^{(m)}\}$$

$$x^{(i)} \in \{0,1\}^n, \qquad x_j^{(i)} = \mathbb{1}\{word\ j\ appears\ in\ document\ i\}$$

$$z^{(i)} \in \{0,1\}\ (2\ clusters), \quad z^{(i)} \sim Bernoulli(\phi)$$

thus

$$P\left(x^{(i)}\middle|z^{(i)}\right) = \prod_{i=1}^{n} P\left(x_j^{(i)}\middle|z^{(i)}\right)$$

$$P\left(x_j^{(i)} = 1\middle|z^{(i)} = 0\right) = \phi_{j|z=0}$$

So, EM algorithm of this model

$repeat\ until\ convergence$:

$E\text{-}step$:

$$w^{(i)} := p\left(z^{(i)} = 1\middle|x^{(i)}; \phi_{j|z}, \phi\right)$$

$M\text{-}step$:

$$\phi_{j|z=1} := \frac{\sum_{i=1}^{m} w^{(i)} \mathbb{1}\{x_j^{(i)} = 1\}}{\sum_{i=1}^{m} w^{(i)}}$$

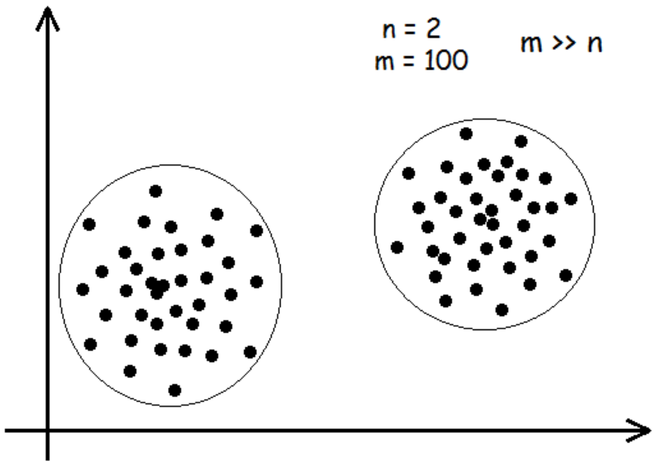
$$\phi_{j|z=0} := \frac{\sum_{i=1}^{m} (1 - w^{(i)}) \mathbb{1}\{x_j^{(i)} = 1\}}{\sum_{i=1}^{m} (1 - w^{(i)})}$$

$$\phi_z := \frac{\sum_{i=1}^{m} w^{(i)}}{m}$$

The parameter $w^{(i)}$s here are almost similar to 1 or 0 (e.g. 0.999/0.00001). Obviously, in *M-step* we treat z as GDA's y and keep updating all the parameters till convergence.

③ Digression: Gaussians

Most models we have discussed above in unsupervised learning satisfy $m \gg n$. For instance, here's a simplified GMM data:



And we have

$$\{x^{(1)}, \dots, x^{(m)}\} \qquad build\ p(x)$$

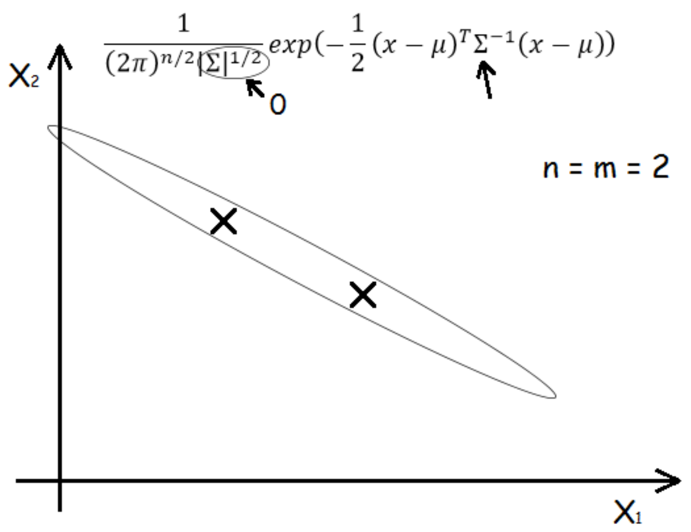
$$x \sim \mathcal{N}(\mu, \Sigma), \qquad \Sigma \in \mathcal{R}^{n \times n}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \qquad \Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

When $n \approx m$ or even $n \gg m$, we'll find that the matrix Σ is singular. Because of this problem, Σ^{-1} and $|\Sigma|$ do not exist, which means the model itself cannot describe this situation to solve the core problem we really concern in this way.

Here's a quite impressive example to describe such a problem, the numbers of training set and features are both 2, the potential Gaussian model would be a very narrow ellipse (infinite width and very thin height).

$$\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$



Q: How to deal with it?

- Use some restrictions of Σ.

$Make\ \Sigma\ diagonal.$

$$\Sigma = diag(\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2) = \begin{pmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n^2 \end{pmatrix} \in \mathcal{R}^{n\times n} \quad \Bigg| \quad \Sigma = \sigma^2 I = \begin{pmatrix} \sigma^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma^2 \end{pmatrix}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)} - \mu_j\right)^2 \quad \Bigg| \quad \sigma^2 = \frac{1}{mn}\sum_{j=1}^{n}\sum_{i=1}^{m}\left(x_j^{(i)} - \mu_j\right)^2$$

The right side restriction is stricter than left side if necessary. Even both method can rebuild model to fit, however, they will model the orthogonal features of data which means each feature is uncorrelated and independent. Often, we are curious about the correlation structure in the data. We need to build a *factor analysis model*, which use parameters than the diagonal and captures some correlations in the data, without having to fit a full covariance matrix.

- Marginals and conditionals of Gaussians

$Joint\ multivariate\ Gaussian\ distribution$

$assume$

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \quad \boldsymbol{x} \sim \mathcal{N}(\mu, \Sigma)\ \ where$$

$$(\boldsymbol{x}_1 \in \mathcal{R}^r, \boldsymbol{x}_2 \in \mathcal{R}^s, \boldsymbol{x} \in \mathcal{R}^{r+s})$$

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

$$(\mu_1 \in \mathcal{R}^r, \mu_2 \in \mathcal{R}^s, \Sigma_{11} \in \mathcal{R}^{r\times r}, \Sigma_{12} = \Sigma_{21}^T \in \mathcal{R}^{r\times s}, \Sigma_{22} \in \mathcal{R}^{s\times s})$$

$thus$

$$E[\boldsymbol{x}] = \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

$$
\begin{aligned}
Cov(\boldsymbol{x}) &= \Sigma \\
&= \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \\
&= E[(\boldsymbol{x}-\mu)(\boldsymbol{x}-\mu)^T] \\
&= E\begin{bmatrix} (\boldsymbol{x}_1-\mu_1)(\boldsymbol{x}_1-\mu_1)^T & (\boldsymbol{x}_1-\mu_1)(\boldsymbol{x}_2-\mu_2)^T \\ (\boldsymbol{x}_2-\mu_2)(\boldsymbol{x}_1-\mu_1)^T & (\boldsymbol{x}_2-\mu_2)(\boldsymbol{x}_2-\mu_2)^T \end{bmatrix}
\end{aligned}
$$

Since marginal distributions of Gaussians are themselves Gaussian and above shows that $\boldsymbol{x}_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$, we can also figure out the conditional distribution of $\boldsymbol{x}_1$ given $\boldsymbol{x}_2$ as $\boldsymbol{x}_1|\boldsymbol{x}_2 \sim \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$, where

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\boldsymbol{x}_2 - \mu_2) \tag{1}$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \tag{2}$$

It'll be helpful in the next part. (the details of conditional distribution parameters' proof at *references: more_on_gaussians Pg. 8*. PS: pay very attention to get Σ^{-1}, partitioned matrix!)

④ Factor analysis

Posit a joint distribution on (x, z) as follows, here z is a latent random variable:

$$z \sim \mathcal{N}(0, I) \qquad z \in \mathcal{R}^d \ (d < n)$$

$$x|z \sim \mathcal{N}(\mu + \Lambda z, \psi)$$

Equivalently, we can also define that

$$\varepsilon \sim \mathcal{N}(0, \psi)$$

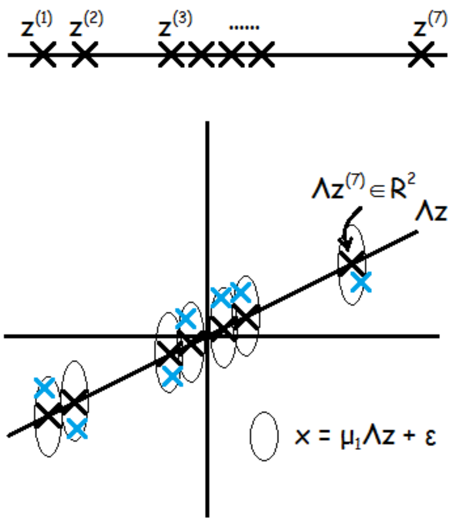
$$x = \mu + \Lambda z + \varepsilon$$

$$(\mu \in \mathcal{R}^n, \Lambda \in \mathcal{R}^{n\times d}, diagonal\ \psi \in \mathcal{R}^{n\times n}. z, \varepsilon\ are\ independent.)$$

E.g.

$$z \in \mathcal{R}^1, x \in \mathcal{R}^2. \quad z^{(i)} \sim \mathcal{N}(0,1)$$

$$\Lambda = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \psi = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$


$z^{(1)}$
$z^{(2)}$
$z^{(3)}$
......
$z^{(7)}$
$\Lambda z^{(7)} \in R^2$
Λz
$x = \mu_1 \Lambda z + \varepsilon$

So, z and x have a joint Gaussian distribution

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim \mathcal{N}(\mu_{zx}, \Sigma)$$

$$z \sim \mathcal{N}(0, I), \varepsilon \sim \mathcal{N}(0, \psi), x = \mu + \Lambda z + \varepsilon$$

$$\rightarrow \qquad E[z] = 0, \; E[x] = E[\mu + \Lambda z + \varepsilon] = \mu$$

$$\mu_{zx} = E\begin{bmatrix} z \\ x \end{bmatrix} = \begin{bmatrix} \vec{0} \\ \mu \end{bmatrix} \begin{matrix} \updownarrow d \\ \updownarrow n \end{matrix}$$

$$E[z] = 0, E[x] = E[\mu + \Lambda z + \varepsilon] = \mu$$

$$\rightarrow \; \Sigma = \begin{bmatrix} \Sigma_{zz} & \Sigma_{zx} \\ \Sigma_{xz} & \Sigma_{xx} \end{bmatrix} = \begin{bmatrix} E[(z - E[z])(z - E[z])^T] & E[(z - E[z])(x - E[x])^T] \\ E[(x - E[x])(z - E[z])^T] & E[(x - E[x])(x - E[x])^T] \end{bmatrix}$$

$$\begin{aligned}
\Sigma_{zz} &= Cov(z) = I \\
\Sigma_{zx} &= E[z(\mu + \Lambda z + \varepsilon - \mu)^T] \\
&= E[zz^T]\Lambda^T + E[z\varepsilon^T] \\
&= I\Lambda^T + 0 = \Lambda^T \\
\Sigma_{xx} &= E[(\mu + \Lambda z + \varepsilon - \mu)(\Lambda z + \varepsilon)^T] \\
&= E[\Lambda z z^T \Lambda^T + \varepsilon z^T \Lambda^T + \Lambda z \varepsilon^T + \varepsilon\varepsilon^T] \\
&= \Lambda E[zz^T]\Lambda^T + 0 + 0 + E[\varepsilon\varepsilon^T] \\
&= \Lambda I \Lambda^T + \psi = \Lambda\Lambda^T + \psi
\end{aligned}$$

$$\therefore \qquad \begin{bmatrix} z \\ x \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}, \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \psi \end{bmatrix}\right) \qquad (3)$$

Moreover, we can also figure out the marginal distribution of z is given by

$$x \sim \mathcal{N}(\mu, \Lambda\Lambda^T + \psi)$$

So, we can write down the log likelihood through a given training set $\{x^{(i)}; i = 1, \dots, m\}$

$$\ell(\mu, \Lambda, \psi) = log \prod_{i=1}^{m} p(x^{(i)}; \mu, \Lambda, \psi)$$

$$= log \prod_{i=1}^{m} \frac{1}{(2\pi)^{n/2}|\Lambda\Lambda^T + \psi|^{1/2}} exp\left(-\frac{1}{2}(x^{(i)} - \mu)^T(\Lambda\Lambda^T + \psi)^{-1}(x^{(i)} - \mu)\right)$$

Accordingly, *EM algorithm of factor analysis model* is described as follows:

$repeat\ until\ convergence:$

$E\text{-}step:$

$$Q_i\left(z^{(i)}\right) \coloneqq p\left(z^{(i)} \middle| x^{(i)}; \mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}}\right)$$

$$\left.\begin{aligned} \mu_{z^{(i)}|x^{(i)}} &= \Lambda^T(\Lambda\Lambda^T + \psi)^{-1}\left(x^{(i)} - \mu\right) \\ \Sigma_{z^{(i)}|x^{(i)}} &= I - \Lambda^T(\Lambda\Lambda^T + \psi)^{-1}\Lambda \end{aligned}\right., use\ (1)(2)(3)$$

$M\text{-}step:$

$$\begin{aligned} \boldsymbol{\Theta} &\coloneqq \arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} \int_{z^{(i)}} Q_i\left(z^{(i)}\right) \log \frac{p\left(x^{(i)}, z^{(i)}; \mu, \Lambda, \psi\right)}{Q_i\left(z^{(i)}\right)} dz^{(i)} \\ &= \arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[\log \frac{p\left(x^{(i)} \middle| z^{(i)}; \mu, \Lambda, \psi\right) p\left(z^{(i)}\right)}{Q_i\left(z^{(i)}\right)}\right] \\ &= \arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[\log p\left(x^{(i)} \middle| z^{(i)}; \mu, \Lambda, \psi\right)\right] \end{aligned}$$

$(here\ \boldsymbol{\Theta}\ represents\ \mu, \Lambda, \psi)$

**Here $Q_i\left(z^{(i)}\right)$ in *M-step* of one loop is a *fixed* part just after *E-step* finished.

Lesson 14

Outline this Lesson:

1. Factor analysis: EM steps
2. Principal Component Analysis (PCA)

① Factor analysis: EM steps

From **Lesson 13**, we can figure out that *EM algorithm of factor analysis model* can be described as follows:

$$repeat\ until\ convergence:$$

$$E\text{-}step:$$

$$Q_i\left(z^{(i)}\right) := p\left(z^{(i)} \middle| x^{(i)}; \mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}}\right)$$

$$\mu_{z^{(i)}|x^{(i)}} = \Lambda^T(\Lambda\Lambda^T + \psi)^{-1}\left(x^{(i)} - \mu\right)$$

$$\Sigma_{z^{(i)}|x^{(i)}} = I - \Lambda^T(\Lambda\Lambda^T + \psi)^{-1}\Lambda$$

$$M\text{-}step:$$

$$\boldsymbol{\Theta} := arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} \int_{z^{(i)}} Q_i\left(z^{(i)}\right) log \frac{p\left(x^{(i)}, z^{(i)}; \mu, \Lambda, \psi\right)}{Q_i\left(z^{(i)}\right)} dz^{(i)}$$

$$= arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[log\, p\left(x^{(i)} \middle| z^{(i)}; \mu, \Lambda, \psi\right)\right]$$

$$(here\ \boldsymbol{\Theta}\ represents\ \mu, \Lambda, \psi)$$

Here, we want to get the exact *M-step*, the update form of exact parameters μ, Λ, ψ instead of $\boldsymbol{\Theta}$ step by step. First, we write down the complete form of "max" goal.

$$\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[log\, p\left(x^{(i)} \middle| z^{(i)}; \mu, \Lambda, \psi\right)\right]$$

$$= \max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[log \frac{1}{(2\pi)^{n/2}|\psi|^{1/2}} exp\left(-\frac{1}{2}\left(x^{(i)} - \mu - \Lambda z^{(i)}\right)^T \psi^{-1}\left(x^{(i)} - \mu - \Lambda z^{(i)}\right)\right)\right]$$

$$= \max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[-\frac{1}{2}\left(x^{(i)} - \mu - \Lambda z^{(i)}\right)^T \psi^{-1}\left(x^{(i)} - \mu - \Lambda z^{(i)}\right) - \frac{1}{2} log|\psi| - \frac{n}{2} log\, 2\pi\right]$$

define

$$G(\mu, \Lambda, \psi) = \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[-\frac{1}{2}\left(x^{(i)} - \mu - \Lambda z^{(i)}\right)^T \psi^{-1}\left(x^{(i)} - \mu - \Lambda z^{(i)}\right) - \frac{1}{2} log|\psi| - \frac{n}{2} log\, 2\pi\right]$$

#1 get μ:

$$\boldsymbol{\nabla}_\mu G(\mu, \Lambda, \psi) = \boldsymbol{\nabla}_\mu \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[-tr\frac{1}{2}\mu^T\psi^{-1}\mu + tr\,\mu^T\psi^{-1}\left(x^{(i)} - \Lambda z^{(i)}\right)\right]$$

$$= \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[\psi^{-1}\left(x^{(i)} - \Lambda z^{(i)} - \mu\right)\right] \xrightarrow{set} \vec{0}$$

$$\therefore \qquad \mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

Here μ is a fixed parameter when update.

#2 get Λ:

$$\nabla_\Lambda G(\mu,\Lambda,\psi) = \nabla_\Lambda \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[-tr\frac{1}{2}\Lambda^T\psi^{-1}\Lambda z^{(i)}z^{(i)^T} + tr\,\Lambda^T\psi^{-1}\left(x^{(i)}-\mu\right)z^{(i)^T}\right]$$

$$= \sum_{i=1}^{m} E_{z^{(i)}\sim Q_i}\left[-\psi^{-1}\left(\Lambda z^{(i)}z^{(i)^T} - \left(x^{(i)}-\mu\right)z^{(i)^T}\right)\right] \xrightarrow{set} \vec{0}$$

$$\therefore \qquad \sum_{i=1}^{m} \Lambda E_{z^{(i)}\sim Q_i}\left[z^{(i)}z^{(i)^T}\right] = \sum_{i=1}^{m}\left(x^{(i)}-\mu\right)E_{z^{(i)}\sim Q_i}\left[z^{(i)^T}\right]$$

$$\Lambda = \left(\sum_{i=1}^{m}\left(x^{(i)}-\mu\right)E_{z^{(i)}\sim Q_i}\left[z^{(i)^T}\right]\right)\left(\sum_{i=1}^{m}E_{z^{(i)}\sim Q_i}\left[z^{(i)}z^{(i)^T}\right]\right)^{-1}$$

Meanwhile, we have

$$E_{z^{(i)}\sim Q_i}\left[z^{(i)^T}\right] = \mu_{z^{(i)}|x^{(i)}}^T$$

$$E_{z^{(i)}\sim Q_i}\left[z^{(i)}z^{(i)^T}\right] = \mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}$$

$$\left(z^{(i)}\ is\ not\ independent\ with\ z^{(i)^T}\right)$$

$$\therefore \qquad \Lambda = \left(\sum_{i=1}^{m}\left(x^{(i)}-\mu\right)\mu_{z^{(i)}|x^{(i)}}^T\right)\left(\sum_{i=1}^{m}\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}\right)^{-1}$$

#3 get ψ:

Similarly, we have (here should use some ***tricks*** to get $\nabla_\psi A\psi^{-1}B$ and $\nabla_\psi|\psi|$):

$$\Phi := \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T} - x^{(i)}\mu_{z^{(i)}|x^{(i)}}^T\Lambda^T - \Lambda\mu_{z^{(i)}|x^{(i)}}x^{(i)^T} + \Lambda\left(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}\right)\Lambda^T$$

$$\psi = diag\,\Phi$$

→ Final *EM algorithm of factor analysis model*:

$$repeat\ until\ convergence:$$

$$E\text{-}step:$$

$$Q_i\left(z^{(i)}\right) := p\left(z^{(i)}\middle|x^{(i)};\mu_{z^{(i)}|x^{(i)}},\Sigma_{z^{(i)}|x^{(i)}}\right)$$

$$\mu_{z^{(i)}|x^{(i)}} = \Lambda^T\left(\Lambda\Lambda^T+\psi\right)^{-1}\left(x^{(i)}-\mu\right)$$

$$\Sigma_{z^{(i)}|x^{(i)}} = I - \Lambda^T\left(\Lambda\Lambda^T+\psi\right)^{-1}\Lambda$$

$M\text{-}step$:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Lambda := \left(\sum_{i=1}^{m}(x^{(i)}-\mu)\mu_{z^{(i)}|x^{(i)}}^{T}\right)\left(\sum_{i=1}^{m}\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^{T}+\Sigma_{z^{(i)}|x^{(i)}}\right)^{-1}$$

$$\Phi := \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T} - x^{(i)}\mu_{z^{(i)}|x^{(i)}}^{T}\Lambda^{T} - \Lambda\mu_{z^{(i)}|x^{(i)}}x^{(i)^T} + \Lambda\left(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^{T}+\Sigma_{z^{(i)}|x^{(i)}}\right)\Lambda^{T}$$

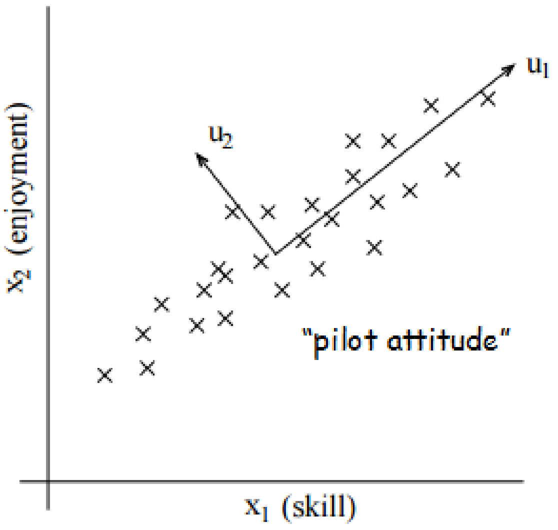
$$\psi = diag\,\Phi$$

② Principal Component Analysis (PCA)

PCA is another dimensionality reduction algorithm just with a little bit "lack" of original information. This algorithm directly do eigenvector calculations instead of EM. The main progress can be described as follows:

$$Given\ \{x^{(1)},\ldots,x^{(m)}\}, x^{(i)} \in \mathcal{R}^{n}.\ \ Reduce\ it\ to\ k\text{-}dim\ data. (k < n\ \ or\ even\ \ k \ll n)$$

E.g. A case of helicopter pilots' skill level associated with their enjoyment. As the picture below, maybe either skill or enjoyment is our concern. As a matter of fact, the inner relation of both features, named "pilot attitude" (vector u_1), is the very feature we are truly interested in. And the other quadrature component u_2 just means the noise or other nonsense part.



Before we develop the PCA algorithm, we need to do pre-processing to normalize its mean and variance first.

$Pre\text{-}processing$:

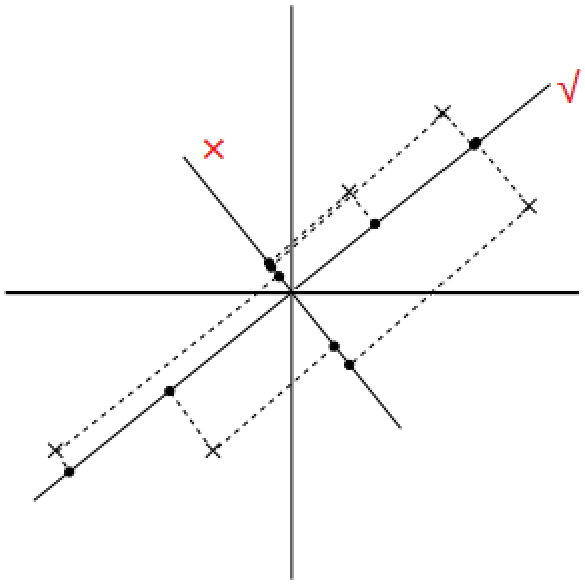
$$\left.\begin{array}{l} 1. Set\ \mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \\ 2. Replace\ x^{(i)}\ with\ x^{(i)}-\mu \end{array}\right\}\ zero\ out\ mean$$

$$\left.\begin{array}{l} 3. Set\ \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)}\right)^2 \\ 4. Replace\ x_j^{(i)}\ with\ x_j^{(i)}/\sigma_j \end{array}\right\} normalize\ to\ unit\ variance\ of\ features$$

In some case, e.g. a grayscale image, which has the same scale of each feature don't need to do *steps 3-4*.

Q: How to compute the "major axis of variation" u?

- Finding a unit vector u so that when data is projected onto the direction corresponding to u, the variance of the projected points is maximized.



We can describe it in a specific way as follows:

$If\ \|u\|_2 = 1, vector\ x^{(i)}\ projected\ on\ u\ has\ length\ {x^{(i)}}^T u.$

$Choose\ u$:

$$u = arg \max_{u:\ \|u\|_2=1} \frac{1}{m}\sum_{i=1}^{m}\left({x^{(i)}}^T u\right)^2$$

$$= arg \max_{u:\ \|u\|_2=1} \frac{1}{m}\sum_{i=1}^{m}(u^T x^{(i)})({x^{(i)}}^T u)$$

$$= arg \max_{u:\ \|u\|_2=1} u^T\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T\right)u$$

$\Rightarrow \quad u\ is\ the\ principal\ eigenvector\ of\ \Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T$

$$\begin{array}{c}\max_{u}\ u^T\Sigma u \\ s.t.\ u^T u = 1\end{array} \quad \Rightarrow \quad \mathcal{L}(u,\lambda) = u^T\Sigma u - \lambda(u^T u - 1)$$

$$\nabla_u \mathcal{L} = 2\Sigma u - 2\lambda u \overset{set}{\longrightarrow} \vec{0}$$

$\therefore \qquad \lambda u = \Sigma u, \quad u \text{ is the principal eigenvector of } \Sigma$

Generalize:

If want k*-dim subspace* $(k < n)$,

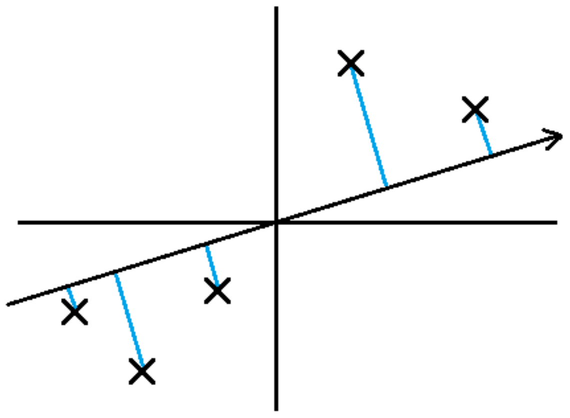
$$\text{choose } u_1, \ldots, u_k \text{ to be } k \text{ top eigenvectors of } \Sigma.$$

then have $x^{(i)} \in \mathcal{R}^n$, *new representation of data in* $\{u_1, \ldots, u_k\}$ *basis*:

$$y^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \in \mathcal{R}^k$$

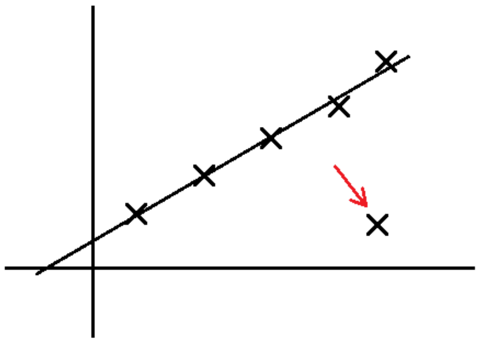
Sometimes, the several eigenvalues may be very closed to themselves. And it's quite dangerous to choose only one or two top eigenvectors to represent it instead of subspace. Because the several "similar" eigenvalues may lead to those associated eigenvectors u_i rotate freely within subspaces. Choosing top k eigenvectors (usually contains $\geq 90\%$ original info) is usually about same.

Here's another way to explain PCA, trying to minimize the sum of the distance' squares between each origin data point and accordingly projected point (sum of squares of blue parts).



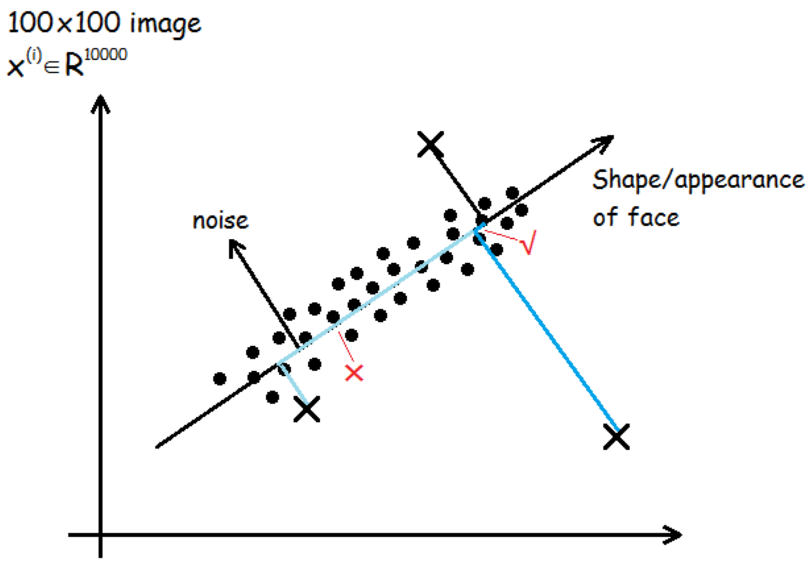
Some applications by PCA:
- Visualization.
  To draw pictures of high dimensional data, use PCA to reduce it to 2-d or 3-d.
- Compressor.
  Use much lower dimensions to save original main information, without too much loss.
- Learning problems.
  1. avoid overfitting, e.g. linear regression; 2. reduce dimensions to simplify the model.
- Anomaly detection.

When a new prediction is quite far away from the subspace in some case.



- Matching/distance calculations.

E.g. face detection. When positions of faces vary in different pictures or the origin features almost have 10,000-d.



Original faces

Eigenfaces

Lesson 15

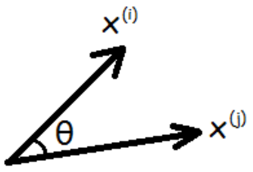
Outline this Lesson:
PCA
1. Latent Semantic Indexing (LSI)
2. Singular Value Decomposition (SVD) implementation
ICA
3. Independent Component Analysis (ICA)

① Latent Semantic Indexing (LSI)

Here's an example. We want to compare lots of text article to find the most similar pairs of them or just scatter them into different clusters by their themes. The first thing we can do is represent all the data set as the vector of a solid vocabulary. Such as:

$$x^{(i)} = \begin{array}{c} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \end{bmatrix} \end{array} \begin{array}{c} a \\ aardvark \\ \vdots \\ learn \\ \vdots \\ study \\ \vdots \end{array} \qquad (usually\ skip\ normalization)$$

$$x^{(i)}, x^{(j)} \text{ - } want\ measure\ "similarity"$$



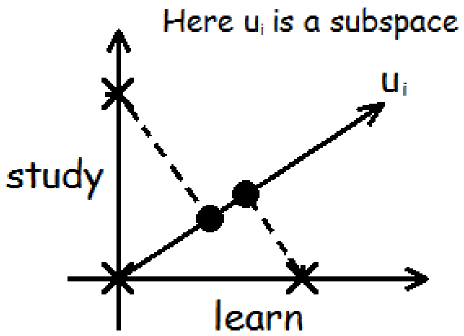
$$\rightarrow sim(x^{(i)}, x^{(j)}) = \cos\theta = \frac{{x^{(i)}}^T x^{(j)}}{\|x^{(i)}\|\|x^{(j)}\|}$$

Observe the equation above, the numerator of $sim(x^{(i)}, x^{(j)})$ actually is

$${x^{(i)}}^T x^{(j)} = \sum_k x_k^{(i)} x_k^{(j)} = \sum_k \mathbb{1}\{Documents\ i\ and\ j\ both\ contain\ word\ k\}$$

And here comes a problem. When two documents have different words with the same meaning, the $sim(x^{(i)}, x^{(j)})$ here must become 0. To avoid this situation, we can apply PCA to project such features into u_i, and then calculate $sim(x^{(i)}, x^{(j)})$. That's LSI really matters in brief. (E.g. the words "learn" and "study")

$$x^{(i)} \rightarrow "study" \qquad x^{(j)} \rightarrow "learn"$$



② Singular Value Decomposition (SVD) implementation

When the number of training set features is very large (e.g. $x^{(i)} \in \mathcal{R}^{50000}$), the covariance of it ($\Sigma \in \mathcal{R}^{50000\times 50000}$) may become a very high dimensional matrix, not easily figured out the principle components. One way to solve such a problem is applying SVD to decompose this very large matrix into three matrix product.

$SVD\ is\ known\ as$:

$$\forall A \in \mathcal{R}^{m\times n},$$

$$\underset{m\times n}{A} = \underset{m\times n}{U}\ \underset{n\times n}{D}\ \underset{n\times n}{V^T}\ \ (SVD)$$

$especially,$

$$D = diag\ \sigma_i = \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n \end{bmatrix}$$

$$\sigma_i = singular\ values\ of\ A$$

$$U's\ columns: eigenvetors\ of\ AA^T$$

$$V's\ columns: eigenvetors\ of\ A^TA$$

In PCA, we have two ways to implement SVD.

#1 Implement SVD directly to training set (when $n \gg m\ or\ n \approx m$).

$define$

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_j^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \qquad X = \begin{bmatrix} {x^{(1)}}^T \\ \vdots \\ {x^{(i)}}^T \\ \vdots \\ {x^{(m)}}^T \end{bmatrix}$$

$$\rightarrow \qquad \Sigma = \sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T = \left[x^{(1)} \cdots x^{(i)} \cdots x^{(m)}\right] \begin{bmatrix} {x^{(1)}}^T \\ \vdots \\ {x^{(i)}}^T \\ \vdots \\ {x^{(m)}}^T \end{bmatrix} = X^TX$$

$To\ get\ top\ k\ eigenvectors\ of\ \Sigma$:

$$X = UDV^T$$

$Top\ k\ columns\ of\ V\ are\ top\ k\ eigenvectors\ of\ X^TX = \Sigma.$

#2 Implement SVD to Σ (when $m \gg n$).

$To\ get\ top\ k\ eigenvectors\ of\ \Sigma$:

$(use\ the\ property: \Sigma\ is\ symmetric, U, V\ are\ orthogonal\ matrix)$

$$\Sigma = UDV^T = UDU^T = UDU^{-1}$$

$Top\ k\ columns\ of\ U\ are\ top\ k\ eigenvectors\ of\ \Sigma.$

The last part is some advice and comparisons, when training set satisfies as:

$$\left[\underset{m\times n}{X}\right] = \left[\underset{m\times r}{U}\middle|\underset{r\times n}{\mathbf{0}}\right]\begin{bmatrix}\underset{r\times s}{D} & \underset{r\times(n-s)}{\mathbf{0}} \\ \underset{(n-r)\times s}{\mathbf{0}} & \underset{(n-s)\times(n-s)}{\mathbf{0}}\end{bmatrix}\left[\underset{n\times n}{V^T}\right]$$

$$\rightarrow \quad = \left[\underset{m\times r}{U}\right]\left[\underset{r\times s}{D}\middle|\underset{r\times(n-s)}{\mathbf{0}}\right]\left[\underset{n\times n}{V^T}\right]$$

Here's a conclusion table of when to use such unsupervised algorithms so far:

	Model $P(x)$	**Not probabilistic**
"Subspace"	Factor Analysis	PCA
"Clumps"/ "Groups"	Mixture of Gaussians	K-means

③ Independent Component Analysis (ICA)

Cumulative distribution functions (Cdf):

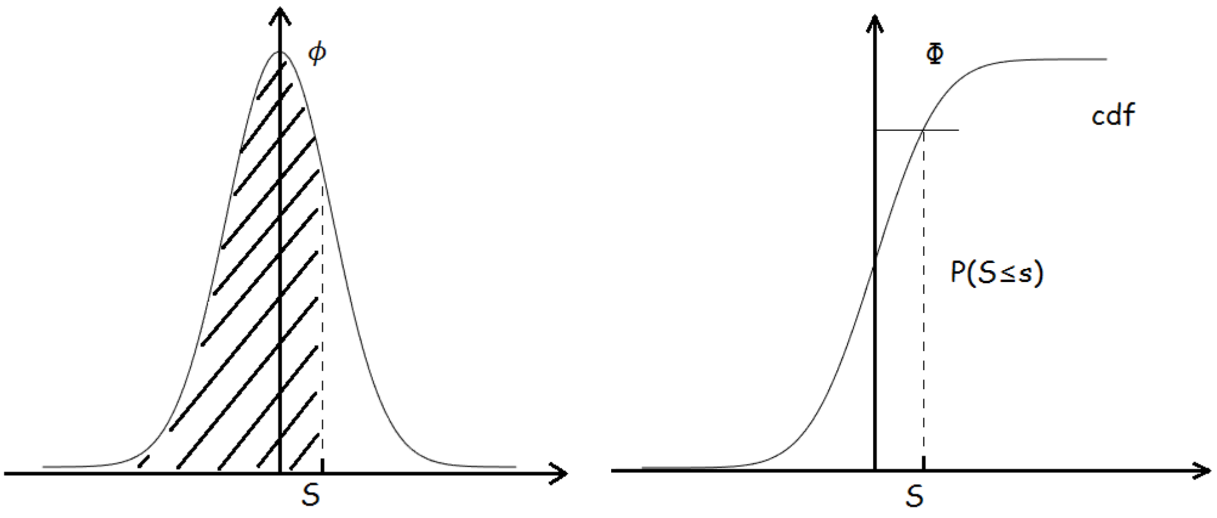
$random\ variable\ s, has\ density\ p_s(s). Cdf$

$$F(s) = P(S \le s) = \int_{-\infty}^{s} p_s(t)\,dt$$

It means when we specify $p_s(s)$ or $F(s)$, we can get one with another because

$$p_s(s) = F'(s)$$

E.g. Gaussian



Considering the "cocktail party problem", here we have n microphones placed in the room to record several n speakers different combinitions of voices. The goal is separating out the original speakers' speech signals. We can observe:

$$x^{(i)} = As^{(i)} \quad x^{(i)} \in \mathcal{R}^n \ (n\, microphones)$$

$I.e.$ $$x_j^{(i)} = \sum_{k=1}^{n} A_{jk} s_k^{(i)}$$

$purpose$:

$$Find \quad W = \begin{bmatrix} — w_1^T — \\ \vdots \\ — w_n^T — \end{bmatrix} = A^{-1} \ \ so\ that \ \ s^{(i)} = Wx^{(i)}$$

Here's some so-called "ICA ambiguities". E.g. assume $s_j^{(i)} \sim Uniform[-1,1]$. Because of the symmetry and $s_j^{(i)} - non\text{-}Gaussian$, the scaling (voices volume), permutation (speakers sequence) and the sign of speakers' voices may vary but finally don't matter. (If Gaussian, we cannot decompose even one of them because the density is rotationally symmetric.)

$Let \ \ s \in \mathcal{R}^n.$

$Density\ of\ s: p_s(s)$

$$x = As = W^{-1}s, \qquad s = Wx$$

→ $$p_x(x) = p_s(Wx) \cdot |W|$$

ICA model:

$$p(s) = \prod_{i=1}^{n} p_s(s_i) \quad choose\ p_s(s_i)$$

→ $$p(x) = \prod_{i=1}^{n} p_s(w_i^T x) \cdot |W|$$

$$(where\ W = A^{-1} = \begin{bmatrix} — w_1^T — \\ \vdots \\ — w_n^T — \end{bmatrix}, s_i = w_i^T x)$$

Q: How to choose $p_s(s_i)$*?*

- E.g. $F(s) = \frac{1}{1+e^{-s}}$, $p_s(s_i) = F'(s_i)$. We can also use Laplacian $\frac{1}{2}e^{-|s|}$ or others.

$Then, given\ \{x^{(1)}, \dots, x^{(m)}\}$:

$$\begin{aligned}\ell(W) &= \sum_i log \prod_j p_s(w_j^T x^{(i)}) \cdot |W| \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} log\, p_s(w_j^T x^{(i)}) + log|W|\end{aligned}$$

We can implement $g(s) = \frac{1}{1+e^{-s}}$, $p_s(s_i) = g'(s_i)$, and maximize it via stochastic gradient ascent (or can be transferred to SGD):

$$W \coloneqq W + \alpha \left(\begin{bmatrix} 1 - 2g(w_1^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{bmatrix} {x^{(i)}}^T + (W^T)^{-1} \right)$$

Finally, we can get the parameter matrix W (though it's not very accurate practically, because the truth is $x^{(i)}$s are dependent) to rebuild the origin unmixed $s^{(i)}$.

$$s^{(i)} = Wx^{(i)}$$

PS: Some applications of ICA:
- EEG (Electroencephalogram) data analysis preprocess.
- Small natural image patches (Humanlike recognition).

Lesson 16

Outline this Lesson:

Reinforcement Learning

1. MDPs (Markov Decision Processes)
2. Value Function
3. Value iteration
4. Policy iteration

① MDPs (Markov Decision Processes)

"*Credit assignment problem*" (not formal):

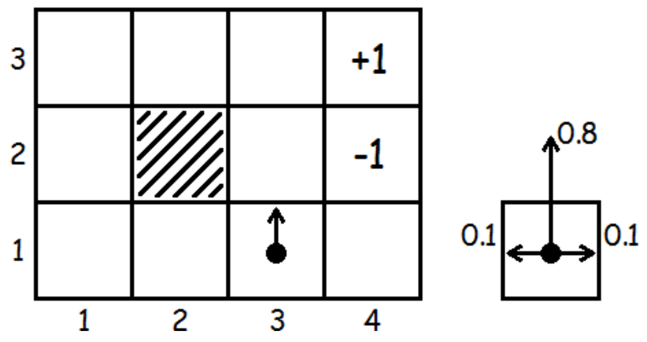
Whether we get a positive or negative reward, figure out what we *actually* did right or did wrong to cause the reward, so we can do more of the right things and less of the wrong things.

MDP tuple

$$(S, A, \{P_{sa}\}, \gamma, R) \ (\mathit{must\ all\text{-}known})$$

$$S \text{ - } \mathit{set\ of\ states}$$
$$A \text{ - } \mathit{set\ of\ actions}$$
$$P_{sa} \text{ - } \mathit{state\ transition\ distributions}$$
$$\textstyle\sum_{s'} P_{sa}(s') = 1, P_{sa}(s') \geq 0$$
$$\gamma \text{ - } \mathit{discount\ factor} \quad 0 \leq \gamma < 1$$
$$R \text{ - } \mathit{reward\ function} \quad R: S \mapsto \mathcal{R}$$

Here is an example to describe how to model with simplified MDP, not considering γ.



We can know that

$$S: 11\ \mathit{status} \qquad A = \{N, S, E, W\}$$

Suppose that when we let the android go north, the probability of going north is just 0.8 along with 0.1 going east or west (assume not stand still), and when it doesn't move to the final sections, it will take some consumption by -0.02. Then we can write down every element in this step tuple just like

$$\left.\begin{array}{l} P_{(3,1),N}\big((3,2)\big) = 0.8 \\ P_{(3,1),N}\big((4,1)\big) = 0.1 \\ P_{(3,1),N}\big((2,1)\big) = 0.1 \\ P_{(3,1),N}\big((3,3)\big) = 0 \\ \quad\vdots \end{array}\right| \begin{array}{c} R\big((4,3)\big) = +1 \\ R\big((4,2)\big) = -1 \\ R(S) = -0.02 \\ \mathit{for\ all\ other\ status} \end{array}$$

The dynamics of an MDP proceeds as follows:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \cdots$$

$$\begin{aligned}&\text{At state } s_0\\&\text{Choose } a_1\\&\text{Get to } s_1 \sim P_{s_0 a_0}\\&\text{Choose } a_1\\&\text{Get to } s_2 \sim P_{s_1 a_1}\\&\vdots\end{aligned}$$

$\therefore$ *Total payoff:*

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \quad (0 \le \gamma < 1)$$

purpose:

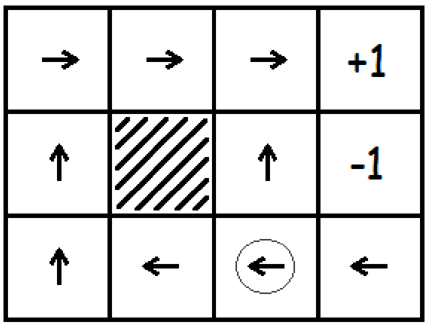
Choose actions over time $(a_0, a_1, \ldots)$ *to maximize*

$$E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

Policy $\pi : S \mapsto A$

Q: How to understand the meaning of "policy"?

- E.g. *Optional policy:*



Policy here means a function mapping from the states to the actions. Concretely, we execute some policy π if, whenever we are in state s, we take action $a = \pi(s)$.

② Value Function

$\left(\textit{Define } V^\pi, V^*, \pi^* \textit{ this part.}\right)$

value function$(V^\pi : S \mapsto \mathcal{R})$:

$$V^\pi(s) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$

$$\rightarrow \qquad V^\pi(s) = E\left[R(s_0) + \gamma\left(V^\pi(s_1)\right) \middle| s_0 = s, \pi\right]$$

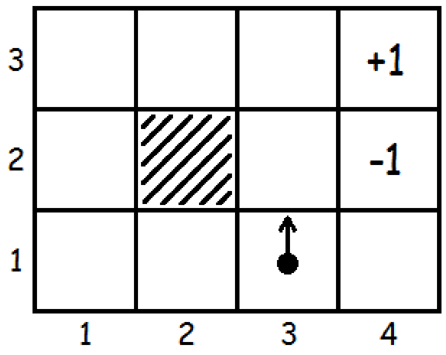
→*Bellman equations*:

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

→→ $$V^{\pi}(s) = R(s) + \gamma E_{s' \sim P_{s\pi(s)}}[V^{\pi}(s')] \quad (s_0 \rightarrow s, s_1 \rightarrow s')$$

e.g.

$\pi((3,1)) = N, V^{\pi}((3,1))$?



→ $V^{\pi}((3,1)) = R((3,1)) + \gamma[0.8V^{\pi}((3,2)) + 0.1V^{\pi}((4,1)) + 0.1V^{\pi}((2,1))]$

optimal value function:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^{*}(s')$$

So, we can also have the "*optimal policy*" π^{*}:

$$\pi^{*}(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{*}(s')$$

Obviously, we can get the relationships among those 3 functions that

$$V^{*}(s) = V^{\pi^{*}}(s) \geq V^{\pi}(s)$$

There is one meaningful property that we can use the same policy π^{*} no matter what the initial state of our MDP is!

③ Value iteration

Value iteration algorithm:

```
Assume finite-state MDPs: $|S| < \infty, |A| < \infty$.
1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence:
    For every state, update
        $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$.
```

The inner-loop of this algorithm can be updated both in synchronous and asynchronous ways. No matter what the inner loop is actually realized, value iteration will cause V to converge to V^*, and we can finally find the optimal policy π^*.

④ Policy iteration

Instead of calculating the optimal value, we can also straightly find an optimal policy for an MDP.

Policy iteration algorithm:

```
$Assume\ finite\text{-}state\ MDPs: |S| < \infty, |A| < \infty.$
$1. Initialize\ \pi\ randomly.$
$2. Repeat\ until\ convergence:$
  $i)\ Let\ V := V^{\pi}\ (solve\ Bellman\ equations).$
  $ii)\ For\ each\ state\ s, let\ \pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s').$
```

Both value iteration and policy iteration are standard algorithms for solving MDPs, and there isn't currently universal agreement over which algorithm is better. However, for MDPs with large state spaces, value iteration maybe preferred. For this reason practically, value iteration seems to be used more often than policy iteration.

Q: What if we don't know P_{sa}?

- We can estimate them from data via:

$$P_{sa}(s') = \frac{\#times\ took\ action\ a\ in\ s, got\ to\ s'}{\#times\ took\ action\ a\ in\ s}$$
$$\left(or\ \frac{1}{|S|}\ \ if\ "0/0"\right)$$

Reinforcement learning problems may have different policies to reach to the final goal. E.g. using value iteration with unknown P_{sa}:

```
$1. Initialize\ \pi\ randomly.$
$2. Repeat:$
$(a)\ Execute\ \pi\ in\ the\ MDP\ for\ some\ number\ of\ trials.$
$(b)\ Update\ estimates\ of\ P_{sa}.$
$(c)\ Solve\ Bellman's\ equation\ using\ value\ interation\ to\ get\ V.$
$(d)\ Update\ \pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s').$
```

Lesson 17

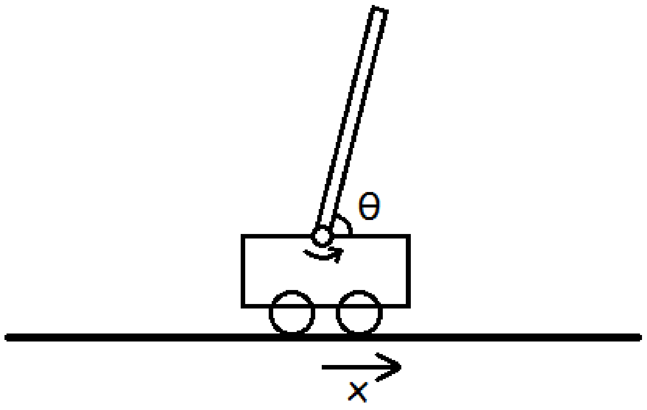
Outline this Lesson:
Continuous state MDPs
1. Discretization
2. Models/Simulators
3. Fitted value iteration

① Continuous state MDPs: Discretization

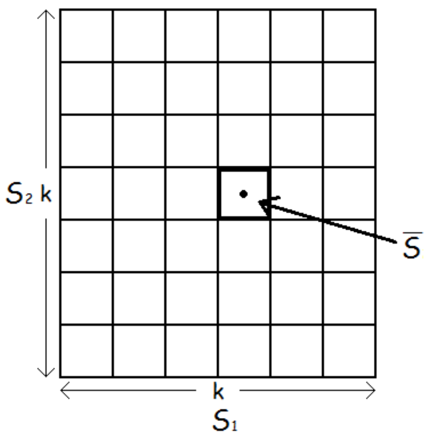
In **Lesson 16**, we have focused on the MDPs with a finite number of states. We now need to discuss some algorithms for solving MDPs with an infinite number of states.
E.g.

$$Car(in\ 2\text{-}d): x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}$$
$$Helicopter(in\ 3\text{-}d): x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}$$
$$Inverted\ pendulum: x, \theta, \dot{x}, \dot{\theta}$$

PS: "*Inverted pendulum*" problem can be shown as picture below, what we're care about is how to keep dynamic balance of the pole with the little car's moving controls.



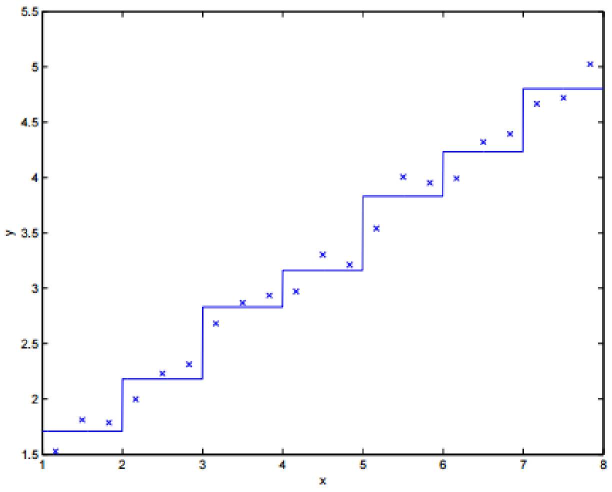
One easy way to solve this kind of problems is to discretize those state space. E.g. we have 2-d states (s_1, s_2), we can use a grid to discretize the state space like:



$$Discretize\ into\ discrete\ states\ \bar{s}$$
$$Solve\ for\ V^*(\bar{s}), \pi^*(\bar{s})$$

Here's two downsides when using this discretization algorithm: V^* and π^* are assumed fairly naive which means the value function (policy function) is piecewise constant in each of the gridcells.

To better understand such a limitation, we can take a supervised learning problem as an example. Obviously, linear regression would do fine on this problem. However, if we use a representation of discretization intervals, then it'll look like this:



When it happens, we would also need a very fine discretization, which means discretize into very small grid cells, to get a good approximation. Meanwhile, if we do discretize quite much smaller, we can cause another problem called the *curse of dimensionality*.

$The\ curse\ of\ dimensionality$:

$$Suppose\ S = \mathcal{R}^n, discretize\ each\ of\ the\ n\text{-}dim\ states\ into\ k\ values.$$

$$Then\ the\ total\ \#\ discrete\ states\ we\ have\ is\ k^n.$$

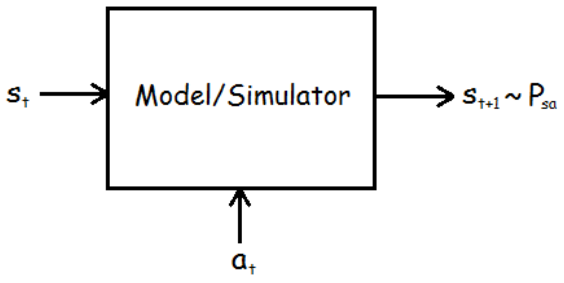
Owing to this exponential growth, discretization does not scale well to large problems. E.g., with a $10\text{-}d$ continuous state, if we discretize each state variable into 100 values, we would have $100^{10} = 10^{20}$ discrete states, pretty high computational expense!

As a rule of thumb, discretization usually works extremely well for $1\text{-}d$ & $2\text{-}d$ problems, along with being simple and quick to implement. $4\text{-}d$ & $6\text{-}d$ problems may also works with some extremely careful designs. When $dim(states) > 6$, it will very rarely works to solve the problem.

② Models/Simulators

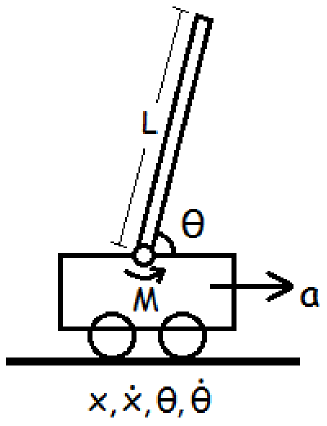
$$"Continuous\ S, discrete\ A."$$

$Assume\ we\ have\ a\ model/simulator\ of\ MDP\ as$:



We have several ways to get such a model. One is to use *physics simulation*. Here's an

example of the inverted pendulum mentioned above.
E.g. *physics simulator*



$$s = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}. \ \dot{s} = \begin{bmatrix} \dot{x} \\ \alpha - L \cdot \beta \cos\theta / M \\ \dot{\theta} \\ \beta \end{bmatrix}, where \quad \begin{matrix} \alpha = \dfrac{a + L \cdot \dot{\theta}^2 \sin\theta}{M} \\ \beta = \dfrac{a + L \cdot \dot{\theta}^2 \cos\theta}{M} \end{matrix}$$

$$\rightarrow \quad s_{t+1} = s_t + \Delta t \cdot \begin{bmatrix} \dot{x} \\ \alpha - L \cdot \beta \cos\theta / M \\ \dot{\theta} \\ \beta \end{bmatrix}, \Delta t = 0.1 \ seconds$$

We can easily get P_{sa}, then the other parameters can come out naturally.

An alternative way to get a model is to *learn a model* from data straightly. Suppose we execute m **trials** in which we repeatedly take actions in an MDP, each trial for T timesteps. The m state sequences like the following:

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} \cdots \xrightarrow{a_{T-1}^{(1)}} s_T^{(1)}$$
$$s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} \cdots \xrightarrow{a_{T-1}^{(2)}} s_T^{(2)}$$
$$\cdots$$
$$s_0^{(m)} \xrightarrow{a_0^{(m)}} s_1^{(m)} \xrightarrow{a_1^{(m)}} s_2^{(m)} \xrightarrow{a_2^{(m)}} \cdots \xrightarrow{a_{T-1}^{(m)}} s_T^{(m)}$$

We can use learning algorithm to estimate s_{t+1} as a **function** of s_t, a_t.
E.g.
$Assume$

$$s_{t+1} = \boldsymbol{A} s_t + \boldsymbol{B} a_t \quad (\boldsymbol{A} \in \mathcal{R}^{n \times n}, \boldsymbol{B} \in \mathcal{R}^{n})$$

$purpose$:

$$\arg\min_{\boldsymbol{A},\boldsymbol{B}} \sum_{i=1}^{m} \sum_{t=0}^{T-1} \left\| s_{t+1}^{(i)} - \left(\boldsymbol{A} s_t^{(i)} + \boldsymbol{B} a_t^{(i)}\right) \right\|^2$$

Here we can use both *deterministic* model and *stochastic* model as follows:

$$s_{t+1} = \boldsymbol{A} s_t + \boldsymbol{B} a_t \qquad (deterministic)$$

or

$$s_{t+1} = \boldsymbol{A} s_t + \boldsymbol{B} a_t + \varepsilon_t \qquad (stochastic)$$

$$\varepsilon_t \sim \mathcal{N}(0, \Sigma)$$

Q: Non-linear model?

- "$Choose\ feature\ \phi(s)\ of\ state\ s$".

$Approximate$:

$$V(s) = \boldsymbol{\theta}^T \phi(s) \quad \leftrightarrow \quad "h_{\boldsymbol{\theta}}(x) = \boldsymbol{\theta}^T \phi(x)"$$

③ Fitted value iteration

Recall the discrete value iteration, we would like to perform the update here as:

$$V(s) \coloneqq R(s) + \gamma \max_a \int_{s'} P_{sa}(s') V(s')\, ds'$$

$$= R(s) + \gamma \max_a E_{s' \sim P_{sa}}[V(s')]$$

Fitted value iteration algorithm (linear regression type):

```
1. Randomly sample m states $\{s^{(1)}, s^{(2)}, \dots, s^{(m)}\} \subseteq S$.
2. Initialize $\boldsymbol{\theta} \coloneqq \boldsymbol{0}$.
3. Repeat:
       For $i = 1, \dots, m$
           For each action $a \in A$
               Sample $s'_1, \dots, s'_k \sim P_{s^{(i)}a}$ (using a model of the MDP).
               Let $q(a) = \frac{1}{k}\sum_{j=1}^{k} R(s^{(i)}) + \gamma V(s'_j)$
                   //estimate for $R(s^{(i)}) + \gamma E_{s' \sim P_{s^{(i)}a}}[V(s')]$.
           Set $y^{(i)} = \max_a q(a)$.
               //estimate for $R(s^{(i)}) + \gamma \max_a E_{s' \sim P_{s^{(i)}a}}[V(s')]$.
       //Want: $V(s^{(i)}) \approx y^{(i)}$.
       $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \frac{1}{2}\sum_{i=1}^{m}(\boldsymbol{\theta}^T \phi(s^{(i)}) - y^{(i)})^2$
```

In this algorithm, we just assume s satisfies stochastic model. When using deterministic model, we can simplify it by setting the parameter k to be equal to 1.

Q: How to get the action π^* / a *?*

- Because of the continuous states and the discrete actions, we just take action at some specific states by

$$\pi^*(s) = a = arg\max_a E_{s'\sim P_{sa}}[V(s')]$$

$E_{s'\sim P_{sa}}[V(s')]$ may hardly come out in some case when we have to sample state S.

We can apply the approximation when the simulator satisfies some deterministic type, i.e. $s_{t+1} = f(s_t, a_t) + \varepsilon_t$, f is a deterministic function such as $f(s_t, a_t) = As_t + Ba_t$ and $\varepsilon_t \sim \mathcal{N}(0, \Sigma)$ or $\varepsilon_t = 0$ using the approximation

$$\begin{aligned} E_{s'}[V(s')] &\approx V(E_{s'}[s']) \\ &= V(f(s,a)) \end{aligned}$$

then choose action

$$arg\max_a V(f(s,a))$$

Lesson 18

Outline this Lesson:

1. State-action rewards
2. Finite horizon MDPs
3. Linear dynamical systems: Models
4. Linear Quadratic Regulation (LQR)
5. Riccati equation

① State-action rewards

General (discrete) MDPs:

$$MDP: (S, A, \{P_{sa}\}, \gamma, R)$$

$$0 \leq \gamma < 1, \qquad R: S \mapsto \mathcal{R}$$

$$V(s) := R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s') \ (converge\ to\ V^*)$$

$$= R(s) + \gamma \max_{a \in A} E_{s' \sim P_{sa}}[V(s')]$$

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

$$= \arg\max_{a \in A} E_{s' \sim P_{sa}}[V^*(s')]$$

Here the expectation part means estimating the next state by the distribution P_{sa}. And we have a more general setting to fit continuous MDPs precisely, we need to apply some variation of previous assumptions.

State-action rewards (*Q-learning*):

$$R: S \times A \mapsto \mathcal{R}$$

→ *Total payoff*:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$$

→→ *Value & policy*:

$$V^*(s) = \max_{a \in A} R(s, a) + \gamma E_{s' \sim P_{sa}}[V^*(s')]$$

$$\rightarrow V(s) := \max_{a \in A} R(s, a) + \gamma E_{s' \sim P_{sa}}[V(s')]$$

$$\pi^*(s) = \arg\max_{a \in A} R(s, a) + \gamma E_{s' \sim P_{sa}}[V^*(s')]$$

② Finite horizon MDPs

Finite horizon MDPs: $(S, A, \{P_{sa}^{(t)}\}, T, R)$ $(T = horizon\ time)$

$\rightarrow$ *Total payoff*:

$$R^{(0)}(s_0, a_0) + R^{(1)}(s_1, a_1) + \cdots + R^{(T)}(s_T, a_T)$$

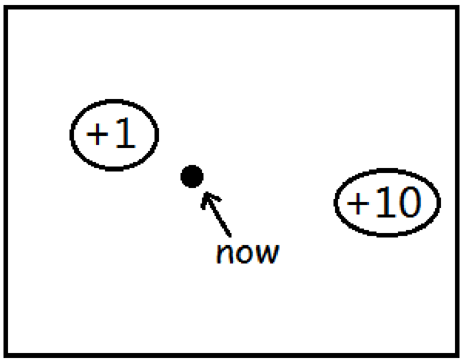
Optimal policy $\quad \pi^{(t)} : S \mapsto A \quad$ (*non-stationary*)

Time dependent dynamics:

$$s_{t+1} \sim P^{(t)}_{s_t, a_t}$$

Q: Why here the optimal policy happen to be non-stationary?

- E.g.



As the cartoon shown above, we want to take actions to aim for the +10 goal at the very beginning. However, when we don't have enough time steps left to reach the +10 goal along with a closer +1 goal, choose the latter is better.

③ Linear dynamical systems: Models

$$V_t^*(s) = E\left[R^{(t)}(s_t, a_t) + \cdots + R^{(T)}(s_T, a_T) \middle| s_t = s, \pi^*\right]$$

$$\forall s \in S : \quad V_T^*(s) := \max_{a \in A} R^{(T)}(s, a) \quad (1)$$

$$\forall t < T : \quad V_t^*(s) := \max_{a \in A} R^{(t)}(s, a) + E_{s' \sim P_{sa}^{(t)}}[V_{t+1}^*(s')] \quad (2)$$

$$\rightarrow \quad \pi_t^*(s) := \arg\max_{a \in A} R^{(t)}(s, a) + E_{s' \sim P_{sa}^{(t)}}[V_{t+1}^*(s')]$$

We can calculate every value and policy sequence with *dynamic programming* like:

$$V_T^*, V_{T-1}^*, V_{T-2}^*, \dots, V_0^*$$
$$\pi_T^*, \quad \dots \dots \dots \dots \quad, \pi_0^*$$

Dynamic Programming algorithm:

```
1. Compute $V_T^*$ using eq. (1).
2. For $t = T-1, \dots, 0$:
       compute $V_t^*$ with $V_{t+1}^*$ using eq. (2)
```

Linear model

$$s_{t+1} \sim P_{s_t, a_t} = A s_t + B a_t$$

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} \cdots \xrightarrow{a_{T-1}^{(1)}} s_T^{(1)}$$

$$\vdots$$

$$s_0^{(m)} \xrightarrow{a_0^{(m)}} s_1^{(m)} \xrightarrow{a_1^{(m)}} \cdots \xrightarrow{a_{T-1}^{(m)}} s_T^{(m)}$$

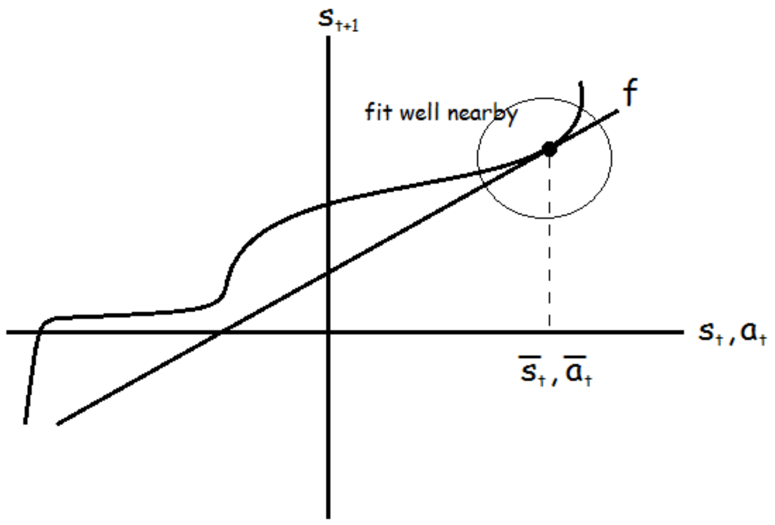
$\rightarrow$ $$\arg\min_{A,B} \frac{1}{2} \sum_{i=1}^{m} \sum_{t=1}^{T-1} \| s_{t+1} - (A s_t + B a_t) \|^2$$

Linearize a non-linear model:

$$s_{t+1} = f(s_t, a_t)$$

E.g. inverted pendulum

$$\begin{pmatrix} x_{t+1} \\ \dot{x}_{t+1} \\ \theta_{t+1} \\ \dot{\theta}_{t+1} \end{pmatrix} = f\left(\begin{pmatrix} x_t \\ \dot{x}_t \\ \theta_t \\ \dot{\theta}_t \end{pmatrix}, a_t \right)$$



$$s_{t+1} \approx f(\bar{s}_t, \bar{a}_t) + (\nabla_s f)^T (s_t - \bar{s}_t) + (\nabla_a f)^T (a_t - \bar{a}_t)$$

Q: How to solve the problem if s_{t+1} *has a constant?*

- Let $s_t' = (1, s_t^T)^T$.

④ Linear Quadratic Regulation (LQR)

$$(S, A, \{P_{sa}^{(t)}\}, T, R) \qquad S = \mathcal{R}^n, A = \mathcal{R}^d$$

$$P_{sa}^{(t)}: \quad s_{t+1} = A_t s_t + B_t a_t + w_t$$

$$A_t \in \mathcal{R}^{n \times n}, B_t \in \mathcal{R}^{n \times d}$$

$$w_t \sim \mathcal{N}(0, \Sigma_t), \textit{ not very important}$$

$Quadratic\ rewards$:

$$R^{(t)}(s_t, a_t) = -s_t^T U_t s_t - a_t^T W_t a_t$$

$$U_t \in \mathcal{R}^{n\times n}, W_t \in \mathcal{R}^{d\times d}$$

$$U_t \geq 0, W_t \geq 0\ (semi\text{-}pd)$$

$\therefore \qquad s_t^T U_t s_t \geq 0,\ a_t^T W_t a_t \geq 0 \ \Rightarrow\ R^{(t)}(s_t, a_t) \leq 0$

E.g. Helicopter control

$$Want\ \ s_t \approx 0.\ Choose\ \ U_t = I, W_t = I.$$

$\therefore \qquad R^{(t)}(s_t, a_t) = -s_t^T s_t - a_t^T a_t = -\|s_t\|^2 - \|a_t\|^2$

To simplify this model, we can also suppose that

$$A = A_1 = A_2 = \cdots$$
$$B = B_1 = B_2 = \cdots$$

$purpose$:

$$max\ \ R^{(0)}(s_0, a_0) + R^{(1)}(s_1, a_1) + \cdots + R^{(T)}(s_T, a_T)$$

$\rightarrow \qquad V_T^*(s_T) = \max_{a_T} R^{(T)}(s_T, a_T)$

$$= \max_{a_T} -s_T^T U_T s_T - a_T^T W_T a_T$$

$$= -s_T^T U_T s_T \quad (a_T^T W_T a_T \geq 0)$$

$$\pi_T^*(s_T) = arg \max_{a_T} R^{(T)}(s_T, a_T) = 0$$

$V_{t+1}^* \rightsquigarrow V_t^*$:

$Suppose$

$$V_{t+1}^*(s_{t+1}) = s_{t+1}^T \phi_{t+1} s_{t+1} + \psi_{t+1}$$

$$(\phi_{t+1} \in \mathcal{R}^{m\times n}, \psi_{t+1} \in \mathcal{R})$$

$Then\ for\ some\ \phi_t, \psi_t$

$$V_t^*(s_t) = s_t^T \phi_t s_t + \psi_t$$

$\rightarrow \quad V_T^*(s_T) = -s_T^T U_T s_T = s_T^T \phi_T s_T + \psi_T\ (\phi_T(1) = -U_T, \psi_T = 0)$

$$V_t^*(s_t) = s_t^T \phi_t s_t + \psi_t \tag{3}$$
$$= \max_{a_t} R^{(t)}(s_t, a_t) + E_{s_{t+1} \sim P_{s_t,a_t}^{(t)}}[V_{t+1}^*(s_{t+1})]$$
$$= \max_{a_t} -s_t^T U_t s_t - a_t^T W_t a_t + E_{s_{t+1} \sim \mathcal{N}(A_t s_t + B_t a_t, \Sigma_t)}[s_{t+1}^T \phi_{t+1} s_{t+1} + \psi_{t+1}]$$
$$= \max_{a_t} -s_t^T U_t s_t - a_t^T W_t a_t + \psi_{t+1} + tr(\Sigma_t \phi_{t+1}) + (A_t s_t + B_t a_t)^T \phi_{t+1} (A_t s_t + B_t a_t)$$
$$= \max_{a_t} -a_t^T W_t a_t + (A_t s_t + B_t a_t)^T \phi_{t+1} (A_t s_t + B_t a_t)$$
$$\therefore \quad a_t = \pi_t^*(s_t) = -(B_t^T \phi_{t+1} B_t - W_t)^{-1} B_t^T \phi_{t+1} A_t \cdot s_t = L_t s_t \tag{4}$$

⑤ Riccati equation

Using $eq.(3)\ (4)$, we can finally get the *Discrete Ricatti equations*:

$$\phi_t = A_t^T[\phi_{t+1} - \phi_{t+1} B_t (B_t^T \phi_{t+1} B_t - W_t)^{-1} B_t^T \phi_{t+1}] A_t - U_t$$
$$\psi_t = tr(\Sigma_t \phi_{t+1}) + \psi_{t+1}$$

To summarize, the algorithm work as follows:

1. *(If necessary) Estimate parameters* A_t, B_t, Σ_t.
2. *Initialize* $\phi_T \coloneqq -U_T, \psi_T \coloneqq 0$.
3. *Iterate from* $t = T-1, \dots, 0$ *to update* ϕ_t, ψ_t *using* ϕ_{t+1}, ψ_{t+1}.
   Compute L_t *using* ϕ_{t+1}, ψ_{t+1} *using eq.* (4)
   $\pi_t^*(s_t) = L_t s_t$.

If we just need to figure out the optimal policy, we can run this algorithm just to update only ϕ_t!

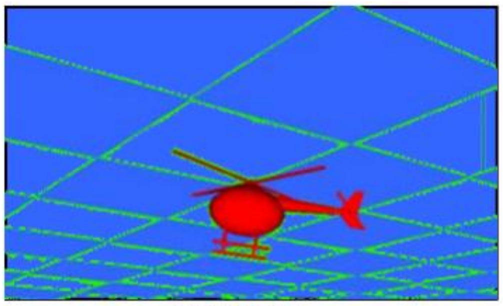
Lesson 19

Outline this Lesson:

1. Debugging RL algorithms
2. LQR - Differential dynamic programming (DDP)
3. Kalman filter & Linear Quadratic Gaussian (LQG)

① Debugging RL algorithms

This example we have talked about in **Lesson 11**, here we pay attention to the analysis of such a problem rather than diagnostics before.



Simulator

Build a simulator of helicopter → Choose a cost function $J(\boldsymbol{\theta})$ → minimize $J(\boldsymbol{\theta})$ to get $\boldsymbol{\theta}_{RL}$

Say

$$J(\boldsymbol{\theta}) = \|\boldsymbol{x} - \boldsymbol{x}_{desired}\|^2 \quad \boldsymbol{x}\text{ - } \textit{helicopter position}$$

$$\boldsymbol{\theta}_{RL} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Suppose the resulting controller parameters $\boldsymbol{\theta}_{RL}$ gives much worse performance than human pilot. *What to do next?*

- Improve simulator?
- Modify cost function $J(\boldsymbol{\theta})$?
- Modify RL algorithm?

**Suppose that*:

1. *The helicopter simulator is accurate.*
2. *The RL algorithm correctly controls the helicopter (in simulation) so as to minimize* $J(\boldsymbol{\theta})$.
3. *Minimizing* $J(\boldsymbol{\theta})$ *corresponds to correct autonomous flight.*

Then: The learned parameters $\boldsymbol{\theta}_{RL}$ *should fly well on the actual helicopter.*

Diagnostics:

- If $\boldsymbol{\theta}_{RL}$ flies well in simulation, but not in reality. → Problem in simulation.

Let $\boldsymbol{\theta}_{human}$ be the human control policy.

- If $J(\boldsymbol{\theta}_{human}) < J(\boldsymbol{\theta}_{RL})$, cost function is failed to minimize. → Problem in RL algorithm.
- If $J(\boldsymbol{\theta}_{human}) \geq J(\boldsymbol{\theta}_{RL})$, it means that minimizing J doesn't correspond to good autonomous flight. → Problem in cost function.

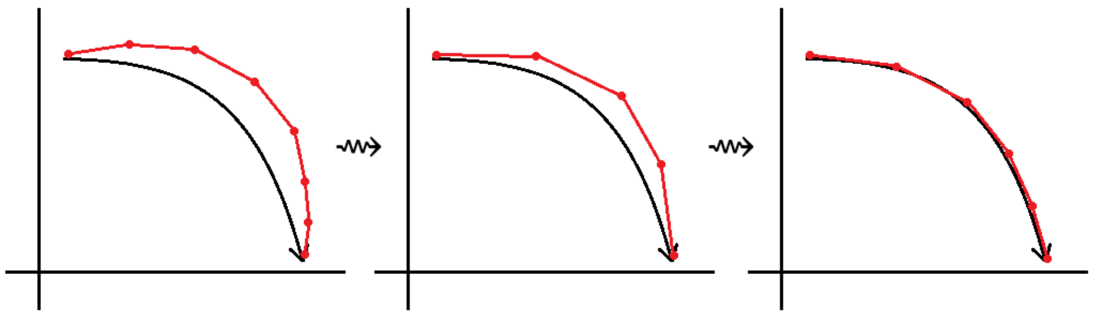
② LQR - Differential dynamic programming (DDP)

DDP algorithm:

$$s_{t+1} = f(s_t, a_t),\ \ non\text{-}linear, deterministic.$$
$$(must\ given\ first!)$$

1. *Come up with nomial trajectory with*
   $\bar{s}_0, \bar{a}_0 \to \bar{s}_1, \bar{a}_1 \to \cdots \to \bar{s}_T, \bar{a}_T.$
2. *Linearize f around each trajectory point,*
   $$i.e.\ \ s_{t+1} \approx f(\bar{s}_t, \bar{a}_t) + (\nabla_s f(\bar{s}_t, \bar{a}_t))(s_t - \bar{s}_t) + (\nabla_a f(\bar{s}_t, \bar{a}_t))(a_t - \bar{a}_t)$$
   $$= A_t s_t + B_t a_t$$
   Expect $(s_t, a_t) \approx (\bar{s}_t, \bar{a}_t)$.
3. *Use LQR to get* π_t.
4. *Use new policy* π_t *to get new nomial trajectory,*
   $i.e.\ \ \bar{s}_0 = initial\ state$
   $\bar{a}_t = \pi_t(\bar{s}_t)$
   $\bar{s}_{t+1} = f(\bar{s}_t, \bar{a}_t)$
   get $\bar{s}_0, \pi_0(\bar{s}_0) \to \bar{s}_1, \pi_1(\bar{s}_1) \to \cdots \to \bar{s}_T, \pi_T(\bar{s}_T)$
   linearize around new trajectory and repeat.

The generate steps can be described as:



$s_{t+1} = f(s_t, a_t)$

Here we just give out the steps of the algorithm, and we will discuss the exact example in the next lesson.

③ Kalman filter & Linear Quadratic Gaussian (LQG)

In the real world, we can hardly observe the full state s_t, we just observe the partial state. To solve such a problem, we have a new model called *POMDP* (Partially Observable MDPs). In this case, we cannot straightly get current state s_t, but with the observation o_t, we can indirectly get it by:

$$o_t | s_t \sim O(o|s)$$

Meanwhile, the tuple of a finite-horizon POMDP is

$$(S, Y, \{O_s\}, A, \{P_{sa}\}, T, R)$$

In this section, we can get the LQG, which is a special type of LQR through the following assumptions (y_t is the observation of the *belief state* s_t, $y_t \in \mathcal{R}^m$, $m < n$):

$$\begin{cases} s_{t+1} = As_t + w_t, & w_t \sim \mathcal{N}(0, \Sigma_w) \\ y_t = Cs_t + v_t, & v_t \sim \mathcal{N}(0, \Sigma_v) \end{cases}$$

$Want:$ $$P(s_t | y_1, \dots, y_t)$$

$s_0, s_1, \dots, s_t, y_1, \dots, y_t$ *have a joint Gaussian distribution*

$$\begin{bmatrix} s_0 \\ \vdots \\ s_t \\ y_1 \\ \vdots \\ y_t \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma) \qquad for\ some\ \ \mu, \Sigma$$

Using the marginal formulas of Gaussians, we would get (theoretically)

$$s_t | y_1, \dots, y_t \sim \mathcal{N}\left(s_{t|t}, \Sigma_{t|t}\right)$$

Unfortunately, the matrices of the marginal distribution always change in time! When time goes, the final matrices will be computationally expensive, about a cost in $O(t^3)$! To avoid this situation, we use the *Kalman filter* algorithm.

$Outline:$

$$\begin{aligned} & P(s_t | y_1, \dots, y_t) \\ & \downarrow \quad predict \\ & P(s_{t+1} | y_1, \dots, y_t) \\ & \downarrow \quad update \\ & P(s_{t+1} | y_1, \dots, y_{t+1}) \end{aligned}$$

$Predict\ step:$

$$s_t | y_1, \dots, y_t \sim \mathcal{N}\left(s_{t|t}, \Sigma_{t|t}\right)$$

then

$$s_{t+1} | y_1, \dots, y_t \sim \mathcal{N}\left(s_{t+1|t}, \Sigma_{t+1|t}\right)$$

where

$$\begin{aligned} s_{t+1|t} &= As_{t|t} \\ \Sigma_{t+1|t} &= A\Sigma_{t|t}A^T + \Sigma_s \end{aligned}$$

$Update\ step$:

get

$$s_t | y_1, \dots, y_t \sim \mathcal{N}\left(s_{t+1|t+1}, \Sigma_{t+1|t+1}\right)$$

$where$

$$s_{t+1|t+1} = s_{t+1|t} + K_{t+1}\left(y_{t+1} - C s_{t+1|t}\right)$$
$$\Sigma_{t+1|t+1} = \Sigma_{t+1|t} - K_{t+1} C \Sigma_{t+1|t}$$
$$K_{t+1} \coloneqq \Sigma_{t+1|t} C^T \left(C \Sigma_{t+1|t} C^T + \Sigma_y\right)^{-1}$$

So, $s_{t+1|t+1}$ is the “best” estimate for s_{t+1}.

Through this algorithm, we can obviously figure out that we don’t need to calculate the former t time steps! The update steps only depends on the previous distribution.

To put it all together, algorithm first runs a *forward pass* to compute the $K_t, \Sigma_{t|t}, s_{t|t}$. Then it runs a *backward pass* (LQR updates) to compute Φ_t, Ψ_t, L_t. Finally, we can get the optimal policy $a_t = L_t s_{t|t}$.

Lesson 20

Outline this Lesson:

1. POMDPs (Partially Observable MDPs)
2. Policy search
3. Conclusion

① POMDPs (Partially Observable MDPs)

The tuple of POMDP:

$$(S, A, Y, \{P_{sa}\}, \{O_s\}, T, R)$$

$$Y - \text{set of possible observations}$$
$$O_s - \text{observation distributions}$$

$$\text{At each step, observe} \quad y_t \sim O_{s_t} \quad (\text{if in state } s_t)$$

② Policy search

Define a set Π *of policies, search for a good* $\pi \in \Pi$. (*like define a set* $\mathcal{H}$ *of hypothesis, search for a good* $h \in \mathcal{H}$)

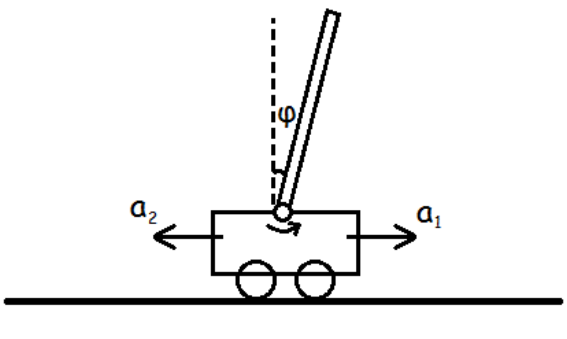
(1) Reinforced

New definition:

A stochastic policy is a function $\pi: S \times A \mapsto \mathcal{R}$ *where* $\pi(s, a)$ (*now* $Q(s, a)$) *is probability of taking action* a *in state* s.

$$\left(\sum_a \pi(s, a) = 1, \pi(s, a) \geq 0\right)$$

E.g. An inverted pendulum system just like



Assume

$$\pi_{\boldsymbol{\theta}}(s, a_1) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T s}}$$

$$\pi_{\boldsymbol{\theta}}(s, a_2) = 1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^T s}}$$

$$s=\begin{bmatrix}1\\x\\\dot{x}\\\phi\\\dot{\phi}\end{bmatrix},\qquad \boldsymbol{\theta}=\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}$$

$$P(\text{"}a=right\text{"})=\frac{1}{1+e^{-\boldsymbol{\theta}^T s}}-\frac{1}{1+e^{-\phi}}$$

$purpose$:

$$\max_{\boldsymbol{\theta}}\ E[R(s_0,a_0)+\cdots+R(s_T,a_T)|\pi_{\boldsymbol{\theta}},s_0]$$

$$E[R(s_0,a_0)+\cdots+R(s_T,a_T)|\pi_{\boldsymbol{\theta}},s_0]$$

$$=\sum P(s_0,a_0,s_1,a_1,\dots,s_T,a_T)[R(s_0,a_0)+\cdots+R(s_T,a_T)]$$
$$=\sum P(s_0)\pi_{\boldsymbol{\theta}}(s_0,a_0)P_{s_0a_0}(s_1)\pi_{\boldsymbol{\theta}}(s_1,a_1)\cdot\cdots\cdot\pi_{\boldsymbol{\theta}}(s_T,a_T)\times\text{"}payoff\text{"}$$

$$(payoff=\ R(s_0,a_0)+\cdots+R(s_T,a_T))$$

So the *reinforced algorithm* (gradient ascend):

$Loop\ until\ convergence$:

```
    1. $Sample\quad s_0,a_0,s_1,a_1,\dots,s_T,a_T.$
    2. $Compute\ \ payoff=R(s_0,a_0)+\cdots+R(s_T,a_T).$
    3. $Update$
        $\boldsymbol{\theta}:=\boldsymbol{\theta}+\alpha\cdot\left[\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s_0,a_0)}{\pi_{\boldsymbol{\theta}}(s_0,a_0)}+\cdots+\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s_T,a_T)}{\pi_{\boldsymbol{\theta}}(s_T,a_T)}\right]\times payoff.$
```

If we only have an approximation $\hat{s}$ of s (could be $\hat{s}=s_{t|t}$ from Kalman Filter), we can also use it with just a little variation:

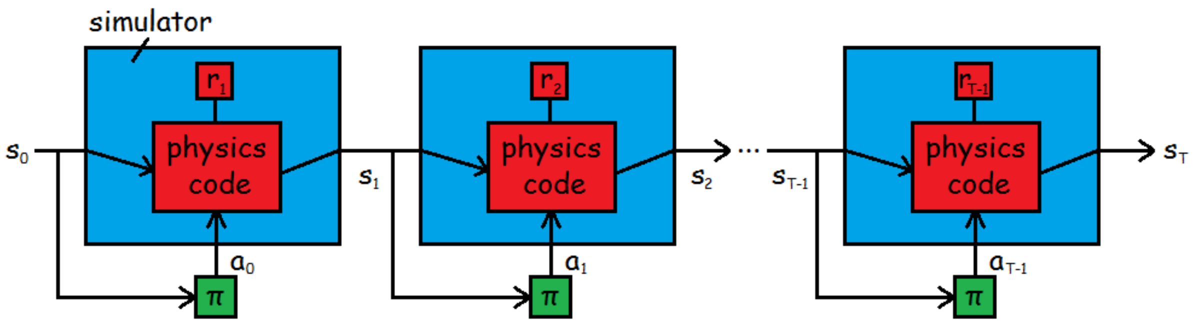
$$\pi_{\boldsymbol{\theta}}(\hat{s},a)=\frac{1}{1+e^{-\boldsymbol{\theta}^T\hat{s}}}\ (or\ other\ model)$$

As a gradient ascend algorithm, the reinforced also has the negative property such as: Because of the observed noise, the sampled sequence in essentially a sort of random direction, only satisfy the expectation; As the parameter T grows and the noise, it may take a lot of time to sample the different sequence (see the pseudocode), converge with million iteration step etc.

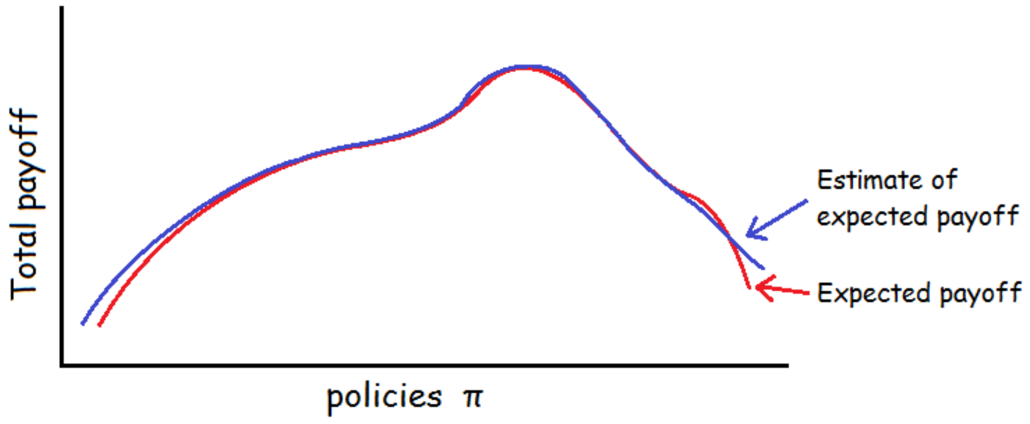
This algorithm is more useful in just simulator rather than the actual physical device. So, we have another algorithm called *PEGASUS* to simplify it.

(2) PEGASUS (Policy Evaluation of Gradient And Search Using Scenarios)

This algorithm totally based on the inspiration of simulator way to build models (rather than sample the physical data). The core method is using a random number generator ***only once*** to generate the stochastic sequence. Then fix it in advance and always use the same sequence of random numbers.



Solution: Using PEGASUS method, we can turn problem into an ordinary deterministic optimization problem.



③ Conclusion

Policy search algorithms are easier to the low level control tasks (just like reflexes), such as helicopter flight, driving or controlling various robots.

Problems of long path plannings (like chess, Tetris etc.) prefer to apply a *value function* method instead.

The key of RL problems is really *sequential decision making*, which means we need to make decisions step by step, and each step may have long-term consequences.

Associated course finally:

CS221	*- an overview of AI via Andrew Ng*
CS223A CS225A CS225B	*- robotics class*
CS222 CS227	*- knowledge representation and reasoning class*
CS228	*- Probabilistic Graphical Models*
EE364A	*- Convex Optimization*