Gradient Descent

Wish to fit the model $h(x)=\sum_{i=0}^{n}\theta_i x_i$ (let $x_0=1$) (i: feature number)

- Define cost function $J(\theta)=\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$ [least-square cost function] [Reason to choose: $y^{(i)}=\theta^T x^{(i)}+\varepsilon^{(i)}$, (IID) $\varepsilon^{(i)}\sim\mathcal{N}(0,\sigma^2)$, $y^{(i)}\,|\,x^{(i)};\theta\sim\mathcal{N}(\theta^T x^{(i)},\sigma^2)$; ($h_\theta(x^{(i)})$: ith evaluation; minimunize $J\Leftrightarrow$ MLE to find $\hat{\theta}$]

- Gradient Descent. $\theta_j:=\theta_j-\alpha\frac{\partial}{\partial\theta_j}J(\theta)$ [take a step in the steepest direction of J] (α: learning rate)

  - where $\frac{\partial}{\partial\theta_j}\frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)})-y^{(i)})^2=\sum_{i=1}^{n}(h_\theta(x^{(i)})-y^{(i)})\cdot x_j^{(i)}\Rightarrow\theta_j:=\theta_j-\alpha\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$, for every j (m: number of data)

  $\Rightarrow\theta:=\theta+\alpha\sum_{i=1}^{n}(y^{(i)}-h_\theta(x^{(i)}))x^{(i)}$ [batch gradient-descent]

  - choice of learning rate α: too small — takes too long for convergence
    too big — takes too large step and run past minimum
    (strong signal: $J(\theta)$ increasing instead of decreasing)

  - Stochastic Gradient Descent (for large data-set)

    for $i=1$ to n:

    $\theta_j:=\theta_j+\alpha(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}$ (for every j) [for one sample do improvement]

① Note: this algorithm will not self-converge like batch-gradient descent (i.e. not using parameters from the global optimum)

② when using stochastic gradient descent, decrease the learning rate

Normal Equation (for gradient descent to reach global optimum in "one" step)

Solve normal equation $\nabla_\theta J(\theta)=\vec{0}$. (Eq. 2.1)

Define Design Matrix X (of samples) $=\begin{pmatrix}-(x^{(1)})^T-\\ \vdots\\ -(x^{(m)})^T-\end{pmatrix}$ → train sample vector transposed.

then $X\theta=\begin{pmatrix}x^{(0)T}\\ \vdots\\ x^{(m)T}\end{pmatrix}_{m\times n}\begin{pmatrix}\theta_0\\ \theta_1\\ \vdots\\ \theta_n\end{pmatrix}_{n+1}=\begin{pmatrix}h_\theta^{(1)}(x)\\ \vdots\\ h_\theta^{(m)}(x)\end{pmatrix}\Rightarrow X\theta-\vec{y}=\begin{pmatrix}h_\theta^{(1)}(x)-y^{(1)}\\ \vdots\\ h_\theta^{(m)}(x)-y^{(m)}\end{pmatrix}$

then $\frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y})=\frac{1}{2}\sum_{i=1}^{m}(h_\theta^{(i)}(x)-y^{(i)})^2=J(\theta)$

$\nabla_\theta J(\theta)=\nabla_\theta\frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y})=\frac{1}{2}\nabla_\theta(\theta^T X^T X\theta-(\vec{y})^T(X\theta)-\vec{y}^T(X\theta)^T+\vec{y}^T\vec{y})$ (Eq. 2.2)

Now Notice: $J:\mathbb{R}^{n+1}\to\mathbb{R}$, generalize: for $f:\mathbb{R}^{m\times n}\to\mathbb{R}$, define $\nabla_A f(A)=\begin{pmatrix}\frac{\partial f}{\partial A_{11}}&\cdots&\frac{\partial f}{\partial A_{1n}}\\ \vdots&&\vdots\\ \frac{\partial f}{\partial A_{m1}}&\cdots&\frac{\partial f}{\partial A_{mn}}\end{pmatrix}$ (A)

where $A=\begin{pmatrix}A_{11}&\cdots&A_{1n}\\ \vdots&&\vdots\\ A_{m1}&\cdots&A_{mn}\end{pmatrix}$.

Claim 1. $f(A)=\mathrm{tr}AB$, where $A,B\in\mathbb{R}^{n\times n}$, $f(A)=\sum_{i=1}^{n}\sum_{j=1}^{n}A_{ij}B_{ji}\Rightarrow \nabla_A f(A)=B^T$

Claim 2. $\mathrm{tr}ABC=\mathrm{tr}CAB$.

Claim 3. $\nabla_A \mathrm{tr}AA^TC=CA+C^TA$.

Note: $x^Ty=\mathrm{tr}(yx^T)$ Hence $x^TAx=\mathrm{tr}(Axx^T)$

$\nabla_x x^TAx=2Ax$

(Eq 2.2) $\Rightarrow \frac{1}{2}\nabla_\theta\left(\theta^TX^TX\theta-(\vec{y})^T(X\theta)-(X\theta)^T(\vec{y})\right)$

$=\frac{1}{2}\nabla_\theta\left(\theta^TX^TX\theta-(\vec{y})^T(X\theta)-(\vec{y})^T(X\theta)\right)$

$=\frac{1}{2}\nabla_\theta\left(\theta^TX^TX\theta-2(X^T\vec{y})^T\theta\right)$

$=\frac{1}{2}\nabla_\theta\left(\mathrm{tr}\left((X^TX)\theta\theta^T\right)-2\mathrm{tr}\left(\theta(X^T\vec{y})^T\right)\right)$

$\nabla_\theta\mathrm{tr}(\theta\theta^TX^TX)\leftarrow$ Claim 2+3 ; $\downarrow$ Claim 2 $\rightarrow \nabla_\theta\mathrm{tr}(\theta(X^T\vec{y})^T)$

$=\frac{1}{2}\left[X^TX\theta+(X^TX)^T\theta-2X^T\vec{y}\right]$

$=X^TX\theta-X^T\vec{y}\overset{\text{set}}{=}\vec{0}\Rightarrow\theta=(X^TX)^{-1}(X^T\vec{y})$ (if X^TX is inversable, then calc θ directly)

no repeated features

Locally Weighted Regression

Adjustions as compared to linear regression:

Fit θ to minimize $\sum_{i=1}^{m}w^{(i)}(y^{(i)}-\theta^Tx^{(i)})^2$, $w^{(i)}$ is a weighting function defaulted by: $w^{(i)}=\exp\left(-\frac{(x^{(i)}-x)^2}{2\tau^2}\right)$

[intuitively, if $|x^{(i)}-x|$ is small, $w^{(i)}\approx 1$; if $|x^{(i)}-x|$ is big, $w^{(i)}\approx 0$]

- Sensitivity test for the selection of τ
- Use locally weighted regression when feature numbers are small (i.e. 2,3,4)
- Can choose τ according to the impression held by Gaussian Distribution (i.e. $\frac{1}{\sqrt{2\pi}\tau}e^{-\frac{(x-\mu)^2}{2\tau^2}}$)

τ's effect in the distribution.

Logistic Regression

want $h_\theta(x)\in[0,1]\Rightarrow$ choose $g(z)=\frac{1}{1+e^{-z}}$ s.t. to a hypothesis $h_\theta(x)$, $h_\theta(x)=g(\theta^Tx)=\frac{1}{1+e^{-\theta^Tx}}$

("sigmoid"/"logistic" function)

here $y\in\{0,1\}$, for conventional reasons, choose (assume)

$P(y=1\mid x;\theta)=h_\theta(x)\Rightarrow P(y\mid x;\theta)=h_\theta(x)^y[1-h_\theta(x)]^{1-y}$

$P(y=0\mid x;\theta)=1-h_\theta(x)$

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

Assuming that n training samples were generated independently, the likelihood of the parameters:

$L(\theta) = p(\vec{y}|X;\theta) = \prod_{i=1}^{n} (h_\theta(x^{(i)}))^{y^{(i)}} (1-h_\theta(x^{(i)}))^{(1-y^{(i)})}$

$l(\theta) = \log L(\theta) = \sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$ [Choose θ to maximize $l(\theta)$]

- gradient ascent: $\theta_j := \theta_j + \alpha \frac{\partial}{\partial \theta_j} l(\theta)$ (⇐ when derivating this part remember $h_\theta(x)$ is NONE-LINEAR)

where $\frac{\partial}{\partial \theta_j} l(\theta) = \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)}\right) \frac{\partial}{\partial \theta_j} g(\theta^T x)$

$= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)}\right) g(\theta^T x)(1-g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x$ ($g'(z) = g(z)(1-g(z))$)

$= (y(1-g(\theta^T x)) - (1-y) g(\theta^T x)) x_j = (y - h_\theta(x)) x_j$

- Stochastic gradient ascent updating: $\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$ [$L(\theta)$ guanteed concave ⇒ only global optimal]

? (Q: How to set the boundary since it won't self-converge) ↓ decrease learning rate until θ doesn't change

* Consider Newton's method for solving $l'(\theta) = 0$ (i.e. $\theta^{(t+1)} := \theta^{(t)} - \frac{l'(\theta^{(t)})}{l''(\theta^{(t)})}$ if $\theta^{(t)} \in \mathbb{R}$)

↑ (quadratic convergence)

- When θ is a vector (say $\theta \in \mathbb{R}^{n+1}$) $\theta^{(t+1)} := \theta^{(t)} + H^{-1} \nabla_\theta l$

where: - $\nabla_\theta l$ is a vector of derivative $\in \mathbb{R}^{n+1}$

- $H^{-1} \in \mathbb{R}^{(n+1)\times(n+1)}$ and $H_{ij} = \frac{\partial^2 l}{\partial \theta_i \partial \theta_j}$ (Hessian)

but Newton's method quickly converges (every iteration of Newton can be expensive). when calculating H^{-1}.

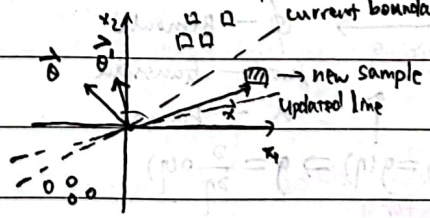
Perceptron Learning

Similar to logistic regression, we now choose $g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$

and $h_\theta(x) = g(\theta^T x)$, with an updating rule $\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$ [h is different from logistic]

Notice: - $y^{(i)} - h_\theta(x^{(i)}) = 0 \Rightarrow$ right prediction

- $|y^{(i)} - h_\theta(x^{(i)})| > 1 \Rightarrow$ wrong prediction, a wrong "prediction" contributes to update;



$\Rightarrow \vec{\theta}$ rotates toward $\vec{x}$

(only geometric interpretation, no probabilistic interpretation).

Generalized Linear Models

- Exponential Family $p(y;\eta) = b(y) \exp(\eta^T T(y) - a(\eta))$ [given canonical parameters]

where - often the case that $T(y) = y$

- $a(\eta)$ plays a role in normalization (i.e. $\int b(y) \frac{\exp(\eta^T T(y))}{\exp(a(\eta))} = 1$)

- η, $T(y)$ must match in dimension

given a fixed choice of $T(y)$, $a(\eta)$ and $b(y)$ $\Rightarrow$ vary η to get different distributions in this exponential family.

- can also include dispersion parameter s.t. $p(y;\eta,\tau) = b(a,\tau)\exp\left(\eta^T T(y) - \frac{a(\eta)}{c(\tau)}\right)$

  in Gaussian, $c(\tau)=\sigma^2$ (negative log likelihood)

- Properties ① MLE with respect to η $\Rightarrow$ concave; NLL wrt η $\Rightarrow$ convex.

  ② $E(y;\eta) = \frac{\partial}{\partial \eta} a(\eta)$

  ③ $Var(y;\eta) = \frac{\partial^2}{\partial^2 \eta} a(\eta)$ [if η is a vector, then operator would be a Hessian]

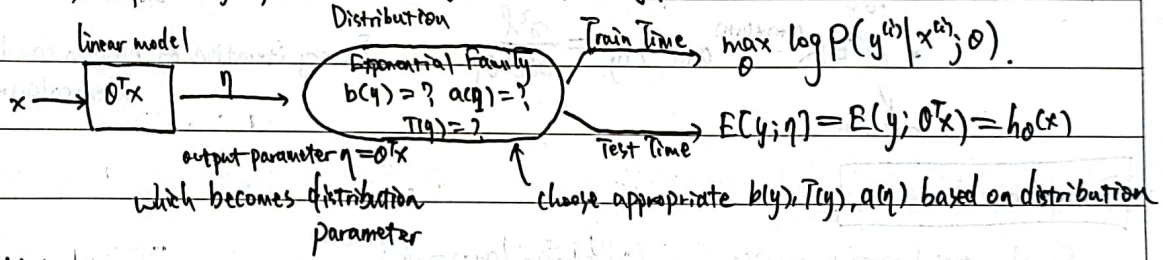
[---] Choices: Real (numbers) — Gaussian
Binary — Bernoulli
Count (positive int) — Poisson
R^+ (positive real) — Gamma, Exponential]---

GLM design choices / construction

1) $y|x;\theta \sim$ Exponential Family(η) [given x and θ, $y \sim EF(\eta)$]

2) Given x, $\eta = \theta^T x$ [if η is vector then $\eta_i = \theta_i^T x$]

3) At Test time, output $E[y|x;\theta]$ [$h_\theta(x) = E(y|x;\theta)$]



Incidental Benefits:

1) Learning update rule: $\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$ [only difference is the distribution $h_\theta(x)$].

2) Three (kinds) of Parameterizations

Model Parameter θ $\xrightarrow{\theta^T x}$ Natural Parameter η $\xrightarrow{g}$ Canonical Parameter

θ: the only thing that is learned

$\theta^T x$: design choice (choose linear)

g: algebraic

Canonical Parameter: ϕ — Bernoulli; μ, σ^2 — Gaussian; λ — Poisson

$E(y;\eta) = g(\eta) \Rightarrow g = \frac{\partial}{\partial \eta} a(\eta)$

① Choose distribution for y

3) Ex. in logistic regression, ** The choice distribution of y is Bernoulli **

② Calculate Expection of distribution ①

$\Rightarrow h_\theta(x) = g(\theta^T x) = E(y|x;\theta) = \phi$, here in Bernoulli, $P(y;\phi) = \exp\left(\left(\log\frac{\phi}{1-\phi}\right)y + \log(1-\phi)\right)$

③ Write in terms of Exp Family to calculate $g = \frac{\partial}{\partial \eta} a(\eta)$

$\Rightarrow a(\eta) = -\log(1-\phi) \Rightarrow g = \frac{\partial}{\partial \eta} a(\eta) = \frac{e^\eta}{1+e^\eta} = \frac{1}{1+e^{-\eta}}$

$= \log(1+e^\eta)$

$\Rightarrow \phi = \frac{1}{1+e^{-\eta}} = \frac{1}{1+e^{-\theta^T x}}$ ④ Equation $E(y|x;\theta) = g(\eta)$ to solve for canonical parameters

SoftMax Regression

$\phi_K = 1 - \sum_{i=1}^{K-1} \phi_i$ make $(K-1)$ independent variables

Classification Problem with $y \in \{1,2,\cdots K\}$, use $\phi_1, \phi_2, \cdots \phi_K$ to denote possibility of belonging to class i.

define $T(y) \in \mathbb{R}^{K-1}$ as one-hot vector, where:

$$T(1) = \begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix},\ T(2) = \begin{pmatrix}0\\1\\\vdots\\0\end{pmatrix},\ \cdots\ T(K-1) = \begin{pmatrix}0\\0\\\vdots\\1\end{pmatrix} \text{ and } T(K) = \begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix}$$

obtain $E(T(y)_i) = P(y=i) = \phi_i$ [$T(y)_i$ denotes i-th element in $T(y)$]

- then $P(y;\phi) = \phi_1^{T(y)_1}\phi_2^{T(y)_2}\cdots\phi_{K-1}^{T(y)_{K-1}}\phi_K^{1-\sum_{i=1}^{K-1}T(y)_i}$

ExpF. Interpretation

$$= \exp\left(T(y)_1\log\phi_1 + \cdots + T(y)_{K-1}\log\phi_{K-1} + \left(1-\sum_{i=1}^{K-1}T(y)_i\right)\log\phi_K\right)$$

$$= \exp\left(\begin{pmatrix}\log\phi_1/\phi_K\\\vdots\\\log\phi_{K-1}/\phi_K\end{pmatrix}^T\begin{pmatrix}T(y)_1\\T(y)_2\\\vdots\\T(y)_{K-1}\end{pmatrix} + \log\left(1-\sum_{i=1}^{K-1}\phi_i\right)\right)$$ [belongs to Exp Family]

the link function is given by $\eta_i = \log\phi_i/\phi_K \Rightarrow \phi_K = \frac{1}{\sum_{i=1}^{K} e^{\eta_i}} \Rightarrow \phi_i = \frac{e^{\eta_i}}{\sum_{i=1}^{K} e^{\eta_i}}$ [softmax function]

given η's and x's are linearly linked, $\eta_i = \theta_i^T x$ $(i = 1,2,\cdots K-1)$ and $\theta_i \in \mathbb{R}^{d+1}$, then

$$p(y|x;\theta) = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}}$$

output is also a distribution

$$h_\theta(x) = E[T(y)|x;\theta] = (\phi_1,\phi_2,\cdots\phi_{K-1})^T = \left(\frac{e^{\theta_1^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}},\cdots,\frac{e^{\theta_{K-1}^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x}}\right)$$ [output probability $p(y=i|x;\theta)$]

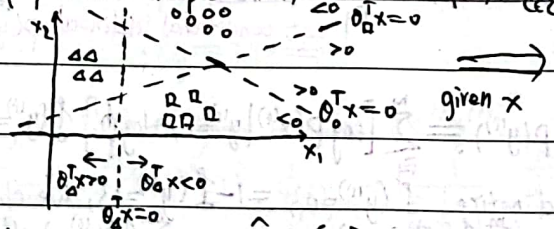
log-likelihood given n samples $\{x^{(i)}, y^{(i)}\}$ $(i=1,2,\cdots n)$

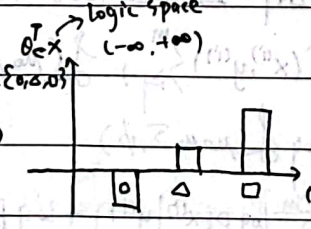
$$\ell(\theta) = \sum_{i=1}^{n}\log p(y^{(i)}|x^{(i)};\theta) = \sum_{i=1}^{n}\log\prod_{l=1}^{K}\left(\frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^{K} e^{\theta_j^T x^{(i)}}}\right)^{1\{y^{(i)}=l\}}$$ [$1\{\cdot\}=1$ if $\cdot$ else 0]

maximize $\ell(\theta)$ ⇐ gradient ascent or Newton's method.

- Cross Entropy Interpretation

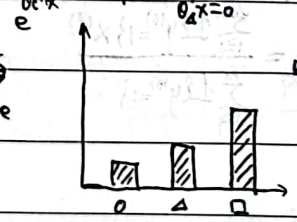
one set of parameters per class that satisfies:



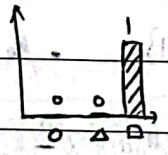


make them similar

[minimize the cross entropy of the two distribution]



$\hat{P}(y) \longleftrightarrow P(y)$ (true labeling)



$\hat{P}(y)$ for given x ⇒ a distribution

(likelihood of given x belonging to which class)

Cross Entropy $(\hat{p}(y), p(y)) = -\sum_{y \in \{\text{label}\}} p(y) \log \hat{p}(y) \overset{p(y) \text{ being ideal/true label}}{=\!=\!=} -\sum_{y \in [0,1,0]} \begin{pmatrix} p(y)=1 \text{ for } 1 \\ p(y)=0 \text{ for } 0 \\ p(y)=0 \text{ for } 0 \end{pmatrix} \log \hat{p}(y)$

$= -\log \hat{p}(y) = -\log \frac{e^{\theta_y^T x}}{\sum_{c \in [0,1,0]} e^{\theta_c^T x}}$ $\Rightarrow$ do gradient descent to minimize cross entropy.

Generative Learning Algorithm

Idea: given y, what do features look like? (namely learn $p(x|y)$ as opposed to discrimitive learning which learns $p(y|x)$)

say that $y \in \{0,1\}$, according to Bayes: $p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1) + p(x|y=0)p(y=0)}$

learned — determined by distribution of y

- Gaussian Discriminant Analysis (GDA)
  - * Assume: given y, features x is distributed according to multivariate normal distribution.
  - * $z \sim \mathcal{N}(\vec{\mu}, \Sigma)$, where $z \in \mathbb{R}^n$, $\vec{\mu} \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ being the covariance matrix.

    where
    - $E(z) = \mu$
    - $\text{Cov}(z) = E(z-\mu)(z-\mu)^T = E[zz^T] - (E(z))(E(z))^T = \Sigma$
    - $p(z) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(z-\mu)^T \Sigma^{-1}(z-\mu)\right)$.

Based on the assumed distribution, GDA model:

$p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right)$ [parameters: μ_0, μ_1 (different means for each classes $\in \mathbb{R}^n$), Σ (same cov for each $\in \mathbb{R}^{n \times n}$), ϕ]

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

Can make the covariance matrix different for each class, but this would give a non-linear boundary as opposed to a linear one (with same Σ)

with y considering Bernolli distributed: $p(y) = \phi^y (1-\phi)^{1-y}$

* Maximize JOINT-LIKELIHOOD. (train x,y together)

given training set $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$, $L(\mu_0, \mu_1, \Sigma, \phi) = \prod_{i=1}^m p(x^{(i)}, y^{(i)}; \mu_0, \mu_1, \Sigma, \phi) = \prod_{i=1}^m p(x^{(i)}|y^{(i)}) p(y^{(i)})$.

not conditional likelihood $p(y^{(i)}|x^{(i)}; \theta)$.

$\ell(\mu_0, \mu_1, \Sigma, \phi) = \log L(\mu_0, \mu_1, \Sigma, \phi)$

$= \sum_{i=1}^m \log p(x^{(i)}|y^{(i)}) + \log p(y^{(i)}) = \sum_{i=1}^m \left[\log p(x^{(i)}|y^{(i)}=1) + \log \phi\right] 1\{y^{(i)}=1\} + \left[\log p(x^{(i)}|y^{(i)}=0) + \log(1-\phi)\right] 1\{y^{(i)}=0\}$

take partial derivative for each parameter and notice $1\{y^{(i)}=0\} = 1 - 1\{y^{(i)}=1\}$, we obtain:

- $\phi = \frac{1}{m}\sum_{i=1}^m 1\{y^{(i)}=1\}$
- $\mu_0 = \frac{\sum_{i=1}^m 1\{y^{(i)}=0\} x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)}=0\}}$
- $\mu_1 = \frac{\sum_{i=1}^m 1\{y^{(i)}=1\} x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)}=1\}}$
- $\Sigma = \frac{1}{m}\sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$

when making predictions, model output $\arg\max_y p(y|x) = \arg\max_y p(y)p(x|y)$.

- Linkage with Logistic Regression

If plotting $p(y=1|x;\phi,\mu_0,\mu_1,\Sigma)$ as a function of x, we'll find the form

$$p(y=1|x;\phi,\mu_0,\mu_1,\Sigma)=\frac{1}{1+e^{-\theta^T x}} \quad \text{for some appropriate } \theta.$$

GDA assumptions (generative) Logistic Regression assumptions (discriminative)

$x|y=0 \sim \mathcal{N}(\mu_0,\Sigma)$
$x|y=1 \sim \mathcal{N}(\mu_1,\Sigma)$
$y \sim \text{Bernoulli}(\phi)$

indicates $\Longrightarrow$ $p(y=1|x)=\frac{1}{1+e^{-\theta^T x}}$ $x|y \sim$ Exp Family and for $y=k$ and $y=j$, if $k\neq j$ then $x|y$ only vary in natural parameters $\Rightarrow p(y|x)$ is logistic

- GDA — stronger set of assumptions; LR — weaker set of assumptions
  - if assumptions are roughly correct, then model produced will do better (even with a small dataset)
  - weaker assumptions make the model more robust, a safe choice if don't know the distribution of $x|y$.

- Naive Bayes

* Turn occurrence of word into feature vectors $\vec{x}=\{0,1\}^n$, where n is the total number of feature words and $x_i=1\{\text{word } i \text{ appears in current piece of text}\}$
* want to model $p(x|y)$ and $p(y)$, direct modeling using 2^n-1 parameters won't work
* Assume x_i's are conditionally independent given y, then

$$p(x_1,x_2,\cdots x_n|y)=p(x_1|y)p(x_2|x_1,y)p(x_3|x_1,x_2,y)\cdots p(x_n|x_1,\cdots x_{n-1},y)$$
$$=p(x_1|y)p(x_2|y)\cdots p(x_n|y)$$

extremely strong ↓ [Naive Bayes Assumption (not exactly true)] ↓ but works well on problems

Parametrized by
- $\phi_{j|y=1}=P(x_j=1|y=1)$
- $\phi_{j|y=0}=P(x_j=1|y=0)$
- $\phi_y=P(y=1)$

Then the joint likelihood given samples $\{x^{(i)},y^{(i)}\}_{i=1}^m$ is $\mathcal{L}(\phi_y,\phi_{j|y=0},\phi_{j|y=1})=\prod_{j=1}^m p(x^{(i)},y^{(i)})$

MLE: $\phi_y=\frac{1}{m}\sum_{i=1}^m 1\{y^{(i)}=1\}$, $\phi_{j|y=1}=\frac{1}{\sum_{i=1}^m 1\{y^{(i)}=1\}}\sum_{i=1}^m 1\{x_j^{(i)}=1 \wedge y_j^{(i)}=1\}$, $\phi_{j|y=0}=\frac{1}{\sum_{i=1}^m 1\{y^{(i)}=0\}}\sum_{i=1}^m 1\{x_j^{(i)}=0 \wedge y^{(i)}=1\}$

when making predictions,

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

[Note:] When x_j can take values of $\{1,2,\cdots k\}$, can model $p(x_j|y)$ as multinomial instead of Bernoulli. Moreover, can discretize continous values to use Naive Bayes when GDA doesn't turn out well.

e.g. x_i: <400 → 1, $400-800$ → 2, $800-1200$ → 3, >1200 → 4. [conventionally, discretize into 10 values]

↓ Extension: Say maybe a word didn't exist in the trainning set, then with MLE some of ϕ_j might end up as zero.

Apply Laplace Smoothing: $\phi_j = \frac{1}{k+n}\left(\sum_{i=1}^{n} 1\{z^{(i)}=j\} + 1\right)$, where $\sum_{j=1}^{k}\phi_j = 1$ still holds. [Given based on Bayesian-Estimate].

Then the estimate for parameters becomes:

$$\phi_{j|y=1} = \frac{1+\sum_{i=1}^{m} 1\{x_j^{(i)}=1, y^{(i)}=1\}}{2+\sum_{i=1}^{m} 1\{y^{(i)}=1\}}, \quad \phi_{j|y=0} = \frac{1+\sum_{i=1}^{m} 1\{x_j^{(i)}=1, y^{(i)}=0\}}{2+\sum_{i=1}^{m} 1\{y^{(i)}=0\}}, \quad \phi_y = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}{m}$$

↑ whether or not applying LS to ϕ_y doesn't matter toomuch.

Support Vector Machine

• Functional Margin and Geometric Margin

consider binary classification problem with labels $y \in \{-1, 1\}$, and rewrite classifier as

$$h_{w,b}(x) = g(w^T x + b), \text{ where } g(z) = \begin{cases} +1, & \text{if } z \geq 0 \\ -1, & \text{o/w.} \end{cases} \quad [\underbrace{\theta_0}_{b}, \underbrace{\theta_1, \theta_2, \cdots, \theta_d}_{w}]^T$$

($w \in \mathbb{R}^n$, $b \in \mathbb{R}$; drop $x_0 = 1$ convention)

* Functional Margin.

- Given a training example $(x^{(i)}, y^{(i)})$, define functional margin of w, b as

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

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

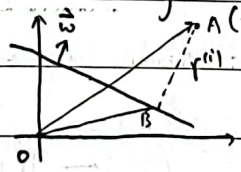
↓ if $\hat{\gamma}^{(i)} > 0$, then right prediction. (correctness)

[Notice] if scaling $(w,b) \to (kw, kb)$ $(k>1)$ we can make $\hat{\gamma}^{(i)}$ arbitrarily large without effecting the sign of $\hat{\gamma}^{(i)}$

⇒ impose normalization condition such as $\|w\|_2 = 1$ or $(w,b) \to \left(\frac{w}{\|w\|_2}, \frac{b}{\|b\|_2}\right)$

- Given a training set, want the functional margin $\hat{\gamma} = \min_i \hat{\gamma}^{(i)}$ to be big

* Geometric Margin



$$\vec{OB} = \vec{OA} - \gamma^{(i)} \cdot \frac{\vec{w}}{\|\vec{w}\|_2} = x^{(i)} - \gamma^{(i)} \frac{w}{\|w\|}$$

where $w^T(x_B) + b = 0 \Rightarrow \gamma^{(i)} = \frac{w^T x^{(i)} + b}{\|w\|}$

more generally, define gm wrt $(x^{(i)}, y^{(i)})$ $\gamma^{(i)} = y^{(i)}\left(\left(\frac{w}{\|w\|}\right)^T x^{(i)} + \frac{b}{\|w\|}\right) = \frac{\hat{\gamma}^{(i)}}{\|w\|}$

can impose arbitrary scale on (w,b) such as $|w_1 + b| + |w_2| = 2$. ← [invariance to scaling of parameters]

Define geometric margin of $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ to be

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

* Optimal Margin Classifier

Assume the training set is linearly separable, pose optimization problem:

$$\max_{\gamma, w, b} \gamma \quad \text{[non-convex optimization]} \quad (1)$$

$$\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \; i = 1, 2, \cdots n.$$

$$\|w\| = 1$$

Notice we can add arbitrary scaling constraint to w, b, and $(kw, kb) \to k\hat{\gamma}$, we can assume $\hat{\gamma} = 1$

wrt training set, and farther notice $\max_{\hat{\gamma}, w, b} \frac{\hat{\gamma}}{\|w\|}$ s.t. $y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}$ $\iff$ $\min_{w,b} \frac{1}{2}\|w\|^2$ ($\|w\|$) s.t. $y^{(i)}(w^T x^{(i)} + b) \geq 1$, $i = 1, 2, \cdots n$ (2).

[quadratic objective and linear constraints]

which is, switch γ in Eq(1) to $\hat{\gamma}$, and choose $\|w\| = \frac{1}{\hat{\gamma}}$ to get Eq(2).

↓ Extension: Lagrange Duality

Consider primal optimization: $\min_w f(w)$

s.t. $g_i(w) \leq 0$, $i = 1, 2, \cdots k$ (3)

$h_i(w) = 0$, $i = 1, 2, \cdots l$

Define generalized Lagrangian $\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$ [α_i, β_i being lagrange multipliers]

PRIMAL $\theta_P(w) = \max_{\alpha, \beta; \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = \begin{cases} f(w), & \text{if } w \text{ satisfies primal constraints} \\ \infty, & \text{o/w.} \end{cases}$

then $p^* \triangleq \min_w \theta_P(w) = \min_w \max_{\alpha, \beta; \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$ is the same optimal problem as (3)

DUAL $d^* \triangleq \max_{\alpha, \beta; \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) = \max_{\alpha, \beta; \alpha_i \geq 0} \theta_D(\alpha, \beta)$

We obtain that

$$d^* = \max_{\alpha, \beta; \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) \leq \min_w \max_{\alpha, \beta; \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = p^*$$

Under Conditions of: • f and g_i's are convex, and h_i's are affine (linear with intercept)

↓ if f has a Hessian, then f is convex $\iff$ H_f is positive + semidefinite

• constraints g_i are strictly feasible $\Rightarrow \exists w \in \mathbb{R}^n$ s.t. $g_i(w) < 0$ for $\forall i \in \{1, 2, \cdots k\}$.

Then there must exist w^*, α^*, β^* s.t. w^* is the solution to the primal problem; α^*, β^* are solutions to the dual problem, and moreover $p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$.

Moreover, w^*, α^*, β^* satisfy the KKT conditions:

• $\frac{\partial}{\partial w_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0$, $\frac{\partial}{\partial \beta_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0$

• $g_i(w^*) \leq 0$, $\alpha^* \geq 0$

(dual complementarity) $\alpha_i^* g_i(w^*) = 0$, $i = 1, 2, \cdots k$.

$\Rightarrow$ if $\alpha_i^* > 0$ then $g_i(w^*) = 0$

[Note: ① w^*, α^*, β^* satisfy KKT $\Rightarrow$ they are solutions for optimality ② if $p^* = d^*$ and w^*, α^*, β^* is solution $\Rightarrow$ they must satisfy KKT].

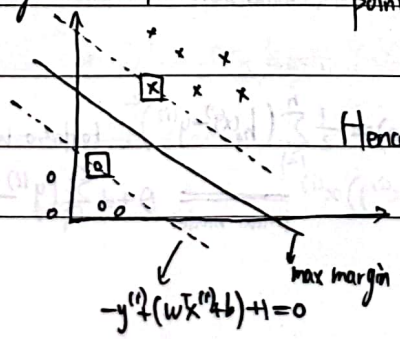
[also: if some w^*, α^*, β^* satisfy KKT $\Rightarrow$ solution to primal & dual

↑ Optimal Margin Classifier

points closest to the separating hyperplane have

$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 = 0$ [rather than ≤ 0]

Hence, only these points hold $\alpha_i > 0$ [support vectors]



Construct Lagrangian $\mathcal{L}(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i[y^{(i)}(wx^{(i)}+b)-1]$ [solve the dual problem].

- Find $\min_{w,b}\mathcal{L}(w,b,\alpha)$ which is $\theta_D(\alpha)$. $\overset{w}{\Rightarrow} \nabla_w\mathcal{L}(w,b,\alpha) = w - \sum_{i=1}^{n}\alpha_i y^{(i)}x^{(i)} = 0 \Rightarrow w = \sum_{i=1}^{n}\alpha_i y^{(i)}x^{(i)}$.

$\overset{b}{\Rightarrow} \frac{\partial}{\partial b}\mathcal{L}(w,b,\alpha) = \sum_{i=1}^{n}\alpha_i y^{(i)} = 0$ [w lies in the span of training sets].

- Then $\mathcal{L}(w,b,\alpha)$ simplifies to $\mathcal{L}(w,b,\alpha) = \sum_{i=1}^{n}\alpha_i - \sum_{i,j=1}^{n}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)})^T x^{(j)} - b\sum_{i=1}^{n}\alpha_i y^{(i)} + \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y^{(i)}y^{(j)}\langle x^{(i)},x^{(j)}\rangle$.

$$= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^T x^{(j)}$$

- Get the following optimization problem:

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

s.t. $\alpha_i \geq 0,\ i=1,2,\cdots n$

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

[satisfy conditions for $p^*=d^*$ and holds KKT conditions]

- if we can solve for optimal α in this case, then

$w^* = \sum_{i=1}^{n}\alpha_i^* y^{(i)}x^{(i)}$ will give us w^*

- Consider the primal problem $\min_{w,b}\max_{\alpha}\mathcal{L}(w,b,\alpha) = \min_{w,b}\theta_p(w,b) = \min\left\{\frac{1}{2}\|w^*\|^2 - \sum_{i=1}^{n}\alpha_i^*[y^{(i)}((w^*)^T x^{(i)}+b)-1]\right\}$.

plug in solutions of dual: w^*, α^*

$b = w^T x_k - \vec{y}_k$ for $\alpha_k > 0$ (Q: mathematical instead?)

[consider $g_i(w) = -y^{(i)}(w^T x^{(i)}+b)+1$]

with $y^{(i)}=1 \Rightarrow b \geq -(w^T x^{(i)})+1$; $y^{(i)}=-1 \Rightarrow b \leq -w^T x^{(i)} - 1$, geometricly maximize geometric margin of both side.

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

Notice when making a new prediction, $w^T x + b = \left(\sum_{i=1}^{n}\alpha_i y^{(i)}x^{(i)}\right)^T x + b = \boxed{\sum_{i=1}^{n}\alpha_i y^{(i)}\langle x^{(i)},x\rangle + b}$ [calculate quantity of inner product between x_{test} and x_{train}].

Moreover, based on KKT conditions, α_i would be non-zero for support vectors

$\Rightarrow$ only need to find inner product between x and the support vectors. [of which there is often only a small number]

- Kernel Methods

* Define Kernel corresponding to feature map $\phi: \mathbb{R}^d \to \mathbb{R}^p$ as a function that maps $\mathcal{X}\times\mathcal{X} \to \mathbb{R}$ satisfying

$$K(x,z) \triangleq \langle \phi(x), \phi(z)\rangle \quad \text{[in this case, } \mathcal{X} = \mathbb{R}^d\text{]}$$

Logic behind kernels: if algorithm can be expressed in terms of $\langle x^{(i)},x^{(j)}\rangle$, but $x^{(i)} \in \mathbb{R}^d$ where d is too large thus calculating $\langle x^{(i)},x^{(j)}\rangle$ in $O(d)$ time can be costly (and need to be done repeatedly), we can map $x^{(i)}$ to higher dimensional feature $\phi(x^{(i)})$ where $K(x,z)$ is precomputed and saves time in calculation and then work implicitly in the mapping space $\mathbb{R}^p$ without explicitly refering to original $x^{(i)}$ in $\mathbb{R}^d$ dimension. for some $K(\cdot,\cdot)$

[Example] Least mean square with kernels.

In section of linear regression, we need to minimize $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$, to do so we refer to batch gradient descent update: $\theta := \theta + \alpha\sum_{i=1}^{n}(y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$ $\underset{\text{linear model}}{=\!=\!=}$ $\theta + \alpha\sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})x^{(i)}$.

Let $\phi: \mathbb{R}^d \to \mathbb{R}^p$, now our goal is to fit $\theta^T\phi(x)$ with θ now a vector in $\mathbb{R}^p$, with the new update rule:

$$\theta := \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - \theta^T\phi(x^{(i)})\right)\phi(x^{(i)})$$

In original update rule, we need to update every entry of θ and store it, but with kernel trick, we will not need to store θ explicitly due to the following fact:

$$\theta = \sum_{i=1}^{n} \beta_i \phi(x^{(i)}). \quad [\theta \text{ can be represented by linear combination of } (\phi(x^{(1)}), \cdots, \phi(x^{(n)}))]$$

and with induction we can obtain update rule for β_i's.

$$\theta = \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - \theta^T\phi(x^{(i)})\right)\phi(x^{(i)})$$

$$= \sum_{i=1}^{n} \left(\beta_i + \alpha\left(y^{(i)} - \theta^T\phi(x^{(i)})\right)\right)\phi(x^{(i)}) \Rightarrow \beta_i := \beta_i + \alpha y^{(i)} - \alpha\theta^T\phi(x^{(i)}).$$

Moreover as we substitute θ: $\beta_i := \beta_i + \alpha\left(y^{(i)} - \sum_{j=1}^{n}\beta_j \phi^T(x^{(j)})\phi(x^{(i)})\right) = \beta_i + \alpha\left(y^{(i)} - \sum_{j=1}^{n}\beta_j K(x^{(j)}, x^{(i)})\right)$ $\forall i \in \{1, 2, \cdots n\}$.

Kernel trick become interesting as
- we can pre-compute $K(x^{(i)}, x^{(j)})$ before loop starts
- for some feature maps, computing $K(x^{(i)}, x^{(j)})$ is efficient.

~~without the necessity to explicitly refer to $\phi(x^{(i)})$~~

Algorithm (LMS with kernels)

1. Compute all $K(x^{(i)}, x^{(j)}) \triangleq \langle\phi(x^{(i)}), \phi(x^{(j)})\rangle$ and obtain $K_{n\times n}$ with $K_{ij} = K(x^{(i)}, x^{(j)})$
2. Loop: $\vec{\beta} := \vec{\beta} + \alpha(\vec{y} - K\vec{\beta})$.

When making predictions, $\theta^T\phi(x) = \sum_{i=1}^{n}\beta_i\phi^T(x^{(i)})\phi(x) = \sum_{i=1}^{n}\beta_i K(x^{(i)}, x)$.]

consider $K(\cdot,\cdot)$ defined by $K(x,z) = (x^Tz + c)^2 = \sum_{i,j=1}^{n}(x_ix_j)(z_iz_j) + \sum_{i=1}^{d}(\sqrt{2c}x_i)(\sqrt{2c}z_j) + c^2$

the corresponding feature mapping $\phi(x)$ contains all feature combination up to order 2.

[Example for $d=3$, $\phi(x) = [x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3, \sqrt{2c}x_1, \sqrt{2c}x_2, \sqrt{2c}x_3, c]^T$]

More broadly, kernel $K(x,z) = (x^Tz + c)^k$ corresponds to an C_{k+d}^{d} feature space, corresponding to all monomials of form $x_{i_1}x_{i_2}\cdots x_{i_k}$ that are up to order k, notice in this $O(d^k)$-dimensional space, computing $K(x,z)$ still only takes $O(d)$ time.

[Notice $C_{k+d}^{k} \underset{\text{(very roughly)}}{\approx} (k+d)^k \to e$ when $k \to \infty$, think this could give an intuition of why the Gaussian kernel $K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$ can be considered as infinite dimensional]

* Necessary conditions for valid kernels

Let $K \in \mathbb{R}^{n\times n}$ be defined s.t. $K_{ij} = K(x^{(i)}, x^{(j)})$, then the following theorem gives sufficient condition for K to be a valid kernel.

Theorem (Mercer) K is a valid (Mercer) kernel $\Leftrightarrow$ for $\forall \{x^{(1)}, x^{(2)} \cdots x^{(n)}\}$, the corresponding kernel matrix is symmetric positive semi-definite.

Proof ($\Rightarrow$) For any $z \in \mathbb{R}^d$, we have $z^T K z = \sum_i \sum_j z_i k_{ij} z_j = \sum_i \sum_j z_i \phi^T(x^{(i)}) \phi(x^{(j)}) z_j$

$= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j = \sum_k \sum_j \sum_i z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j = \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \geq 0$

- L_1-norm soft margin SVM

To make SVM algorithm work for non-linearly separable datasets as well as less susceptible to outliers, we can reformulate the optimization as:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$
$$\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, 2, \cdots n \qquad \text{[relax boundary and add penalty]}$$
$$\xi_i \geq 0, \quad i = 1, 2, \cdots n$$

The dual form of problem then becomes:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$
$$\text{s.t. } 0 \leq \alpha_i \leq C$$
$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0, \quad i = 1, 2, \cdots n$$

and prediction formulas remain the same, the KKT dual-complementarity conditions in this case are

$$\begin{cases} \alpha_i = 0 \Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C \Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C \Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1 \end{cases}$$

Cross Validation

- simple (hold-out) cross validation — when don't have massive dataset

1. randomly split S into S_{train} (say 70% of data), S_{dev} (S_{cv}, the remaining 30%) [(7-3)/(6-2-2)]
2. train each model (M_i, where $\mathcal{M} = \{M_1, M_2 \cdots M_n\}$ is a finite set of models we are selecting among) on S_{train} to get some hypothesis h_i. [choose S_{train}/S_{test} in a way that let meaningful comparison happen between models]
3. Select the h_i that has the smallest error $\hat{\varepsilon}_{S_{cv}}(h_i)$ on the development set S_{dev}.

Optionally, Step 3 can be carried out as selecting M_i according to $\arg\min_i \hat{\varepsilon}_{S_{cv}}(h_i)$ and then retrain M_i on the * ENTIRE * training set again. [exceptions being algorithms sensitive to perturbations of initial data]

- K-fold cross validation [on relatively small training set]

1. randomly split S into k disjoint subsets each containing m/k training samples.
2. for each model M_i, do

   For $j = 1, 2, \cdots k$, train the model M_i on $S_1 \cup S_2 \cup \cdots \cup S_{j-1} \cup S_{j+1} \cup \cdots \cup S_k$ to get hypothesis h_{ij}.

Test the hypothesis h_{ij} on S_j and calculate $\hat{\varepsilon}_{S_j}(h_{ij})$, and let error of M_i being the averge of $\hat{\varepsilon}_{S_j}(h_{ij})$s'

3. Pick the Model M_i with $i = \arg\min_i \frac{1}{k}\sum_{j=1}^{k} \hat{\varepsilon}_{S_j}(h_{ij})$.

$k=10$ is a typical choice of k-fold cross validation, when data is really scarce, consider

Leave-one cross validation with $k=m$ (the size of training set).

[Note:] Here we denote the training model set M as finite, but if we were to select from an infinite model set (e.g. choose bandwidth parameter $\tau \in \mathbb{R}^+$), we can 1. discretize $\tau \in \mathbb{R}^+$ to form finite selection space; 2. perform search over infinite model classes (optimization space $=\infty$).

[Note:] Cross validation can not only be used on selection of the model, but evaluation as well.

• Feature Selection

Apply feature selection algorithm to reduce the number of features.

* Algorithm Forward Search (Wrapper model feature selection)

1. Initialize $F = \phi$

2. Repeat {

(a) For $i = 1, 2, \cdots d$ (d is the number of features), if $i \notin F$, let $F_i = F \cup \{i\}$, train the learning model using only the features in F_i and estimate the generalization error [can be done via cross-validation]

(b) Set $F := F_{\arg\min_i \hat{\varepsilon}_{F_i}(h_i)}$, which is the set that perfams best in step (a).

}. Select the best feature subset that was evaluated during the entire search procedure.

A: given the number of features each iteration, the model is decided, both are not decided?

Q: Can forward-search carry out if # features & model

[Main idea: Adding feature greedily, and the outer loop can terminate with a pre-set $|F|$.]

* Filter Feature Selection

1. For discrete-valued features x_i, choose the informative score $S(i)$ to be mutual information:

$$S(i) := MI(x_i, y) = \sum_{x_i \in \{a,i\}} \sum_{y \in \{a,i\}} p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)p(y)}$$ [extropolation to finite set is trivial]

2. More generally, notice mutual information can be expressed as Kullback-Leibler (KL) divergence:

$$MI(x_i, y) = KL\left(p(x_i, y) \,\|\, p(x_i)p(y)\right)$$

where $KL(p\|Q) = \sum_{x \in X} p(x) \log \frac{p(x)}{Q(x)} = -\sum_{x \in X} p(x) \log \frac{Q(x)}{p(x)}$

in case where $Q(x)$ is unsuitable for numerator

For distributions P and Q of a continuous random variable, relative entropy is defined as

$$KL(P\|Q) = \int_{-\infty}^{+\infty} p(x) \log\left(\frac{P(x)}{Q(x)}\right) dx.$$

in terms of Bayesian inference, $KL(P||Q)$ is a measure of information gained by revising prior probability distribution Q to posterior probability distribution P (i.e. the info lost when Q is used to approximate P).

Note: the Neyman-Pearson Lemma states that the most powerful way to distinguish between distributions P and Q based on observation Y (drawn from one of them) is through $\log \frac{P(Y)}{Q(Y)}$, KL divergence appears to be the expectation if Y is actually drawn from P, namely, $E[Y$ taken from $P]$.

Logic chain: if x_i and y is independant, then KL divergence between $p(x_i, y)$ and $p(x_i)p(y)$ will be zero, thus $S(i)$ will be small, so x_i will be unlikely selected.

3. After ranking the scores, do a cross-validation to decide on the number of features to select.

Bayesian Statistics

Frequentist statistics view θ as being constant valued but unknown, and it's our job to come up with statistical procedures (e.g. maximum likelihood) to estimate θ.

Bayesian view θ as being a random variable, and our "prior belief" of θ is reflected on our choice of θ prior distribution $p(\theta)$. When giving a data set $S = \{x^{(i)}, y^{(i)}\}_{i=1}^{n}$, we can compute the posterior distribution as: ($p(S|\theta)$ depend on the model using)

$$p(\theta|S) = \frac{\prod_{i=1}^{n} p(y^{(i)}|x^{(i)}, \theta)\, p(\theta)}{\int_{\theta} \prod_{i=1}^{n} p(y^{(i)}|x^{(i)}, \theta)\, p(\theta)\, d\theta}$$

→ computationally expensive when integrating over high-dimensional θ.

When given a new test sample x, compute $p(y|x,S) = \int_{\theta} p(y|x,\theta)\, p(\theta|S)\, d\theta$.

and $E(y|x,S) = \int_{y} y\, p(y|x,S)\, dy$. [sum if y is discrete]. ("maximum a posteriori")

Instead of computing for posterior distribution, we can do an approximation, epitome being MAP:

$$\theta_{MAP} = \arg\max_{\theta} \prod_{i=1}^{n} p(y^{(i)}|x^{(i)}, \theta)\, p(\theta)$$ [single-point estimation]

Common choice for prior $p(\theta)$ is to assume that $\theta \sim \mathcal{N}(0, \tau^2 I)$, using this choice, $\|\theta_{MAP}\|_2 < \|\theta_{MLE}\|_2$.

In practice, this cause Bayesian MAP being less susceptible to overfitting than ML estimate.

[due to regularization, and why Bayesian logistic regression is effective for text classificati

Bias-Variance Analysis

Say our goal is to construct an estimator for the unknown parameter θ^* given observed data set $\{(x^{(i)}, y^{(i)})\}$. The distribution of our model output $\hat{\theta}_n$ (random variable given noise typically in label $\vec{y}$) is called *sampling distribution*, and bias and variance are first and second moments of its sampling distribution, namely $\text{Bias}(\hat{\theta}_n) \equiv E[\hat{\theta}_n - \theta^*]$, $\text{Var}(\hat{\theta}_n) \equiv \text{Cov}(\hat{\theta}_n)$

Bias and Variance of an estimator are not necessarily directly related, under squared error, they can be related as: $\text{MSE}(\hat{\theta}_n) = E[\|\hat{\theta}_n - \theta^*\|^2]$

$$= E[\|\hat{\theta}_n - E[\hat{\theta}_n] + E[\hat{\theta}_n] - \theta^*\|^2]$$

$$= E[\|\hat{\theta}_n - E[\hat{\theta}_n]\|^2 + \|E[\hat{\theta}_n] - \theta^*\|^2 + 2(\hat{\theta}_n - E[\hat{\theta}_n])^T(E[\hat{\theta}_n] - \theta^*)]$$

$$= \|E[\hat{\theta}_n] - \theta^*\|^2 + E[\text{tr}[(\hat{\theta}_n - E[\hat{\theta}_n])(\hat{\theta}_n - E[\hat{\theta}_n])^T]]$$

$$= \text{tr}[\text{Var}(\hat{\theta}_n)] + \|\text{Bias}(\hat{\theta}_n)\|^2$$

[quite often the case: techniques employed to reduce variance results in increased bias and vice versa]

- **Bias-Variance Tradeoff in Linear Regression with L_2-regularization.** [L_2-norm]

In linear regression, when taking a probabilistic view, we assume $y^{(i)} = \theta^{*T}x^{(i)} + \epsilon^{(i)}$, with each $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ (i.i.d). Then L_2-regularization led to minimizing the cost function

$$J(\theta) = \frac{\lambda}{2}\|\theta\|_2^2 + \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})^2 \quad [\text{regularization parameter } \lambda > 0]$$

and enjoys a closed-form solution $\hat{\theta}_n = \underset{\theta \in \mathbb{R}^d}{\text{argmin}}\, J(\theta) = \underset{\theta \in \mathbb{R}^d}{\text{argmin}}\left[\frac{\lambda}{2}\|\theta\|_2^2 + \frac{1}{2}\|X\theta - \vec{y}\|_2^2\right]$

$\hat{\theta}_n$ s.t. $\nabla_\theta J(\hat{\theta}_n) = 0 \Rightarrow (\lambda I + X^TX)\hat{\theta}_n = X^T\vec{y} \Rightarrow \hat{\theta}_n = \underbrace{(\lambda I + X^TX)^{-1}}_{\theta \in \mathbb{R}^d}(X^T\vec{y})$

Consider eigendecomposition of PSD matrix X^TX: $X^TX = U\begin{bmatrix}\sigma_1^2 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \sigma_n^2\end{bmatrix}U^T$ with $UU^T = I_n$

even though X (hence X^TX) is not full rank, $(X^TX + \lambda I)$ is always symmetric and PD.

$\Rightarrow (X^TX + \lambda I)^{-1} = U\begin{bmatrix}\frac{1}{\sigma_1^2+\lambda} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{1}{\sigma_n^2+\lambda}\end{bmatrix}U^T \Rightarrow \hat{\theta}_n$ always exists and is unique.

* Bias-Variance analysis: $\hat{\theta}_n = (X^TX + \lambda I)^{-1}X^T(X\theta^* + \vec{\varepsilon}) = (X^TX + \lambda I)^{-1}X^TX\theta^* + (X^TX + \lambda I)^{-1}X^T\vec{\varepsilon}$

$$E[\hat{\theta}_n] \underset{E[\vec{\varepsilon}]=0}{=} U\begin{bmatrix}\frac{1}{\sigma_1^2+\lambda} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{1}{\sigma_n^2+\lambda}\end{bmatrix}U^TX^TX\theta^* = U\begin{bmatrix}\frac{\sigma_1^2}{\sigma_1^2+\lambda} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{\sigma_n^2}{\sigma_n^2+\lambda}\end{bmatrix}U^T\theta^*$$

Observations: 1. when $\lambda = 0$, then $E[\hat{\theta}_n] = \theta^* \Rightarrow$ unbiased but biggest variance.

2. the larger λ, the smaller the eigenvalues will be, shrink towards $0 \Rightarrow$ more bias

$\Rightarrow$ the estimator $\hat{\theta}_n$ of L_2-regularization Linear Regression is Biased.

$$Cov[\hat{\theta}_n] = Cov\left[(X^TX+\lambda I)^{-1}X^TX\theta^* + (X^TX+\lambda I)^{-1}X^T\vec{\varepsilon}\right]$$

$$= Cov\left[(X^TX+\lambda I)^{-1}X^T\vec{\varepsilon}\right] \quad [\text{if } \vec{\varepsilon}\sim\mathcal{N}(0,\tau^2 I) \text{ then } A\vec{\varepsilon}\sim\mathcal{N}(0, A(\tau^2 I)A^T)]$$

$$= (X^TX+\lambda I)^{-1}X^T Cov[\vec{\varepsilon}]\, X(X^TX+\lambda I)^{-1}$$

$$= \tau^2 (X^TX+\lambda I)^{-1}X^TX(X^TX+\lambda I)^{-1}$$

$$= \tau^2 U \begin{bmatrix} \frac{\sigma_1^2}{(\sigma_1^2+\lambda)^2} & \cdots & 0 \\ & \ddots & \\ 0 & \cdots & \frac{\sigma_n^2}{(\sigma_n^2+\lambda)^2} \end{bmatrix} U^T$$

Observations: 1. if larger λ, then smaller $tr(Cov[\hat{\theta}_n]) \Rightarrow$ less variance (but more bias)

• Bias and Variance in Prediction

Given a training set $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, under the assumption $y = f(x) + \varepsilon$ where $E[\varepsilon]=0$, $Var[\varepsilon]=\tau^2$.

Further, we define the true f as $f(x) \equiv E[y|x=x']$.

* Construct $\hat{f}_n$ to mimic f on unseen examples $\Rightarrow$ small generalization error (expected square error loss) for $\hat{f}_n$.

[Note: $\hat{f}_n$ is random due to the randomness embedded in $\varepsilon^{(i)}$s.]

Consider a new unseen example pair (y^*, x^*), we obtain:

$$MSE[\hat{f}_n] = E\left[(y_* - \hat{f}_n(x_*))^2\right] = E\left[(f(x_*) - \hat{f}_n(x_*))^2\right] + \varepsilon^2 - 2\varepsilon(f(x_*) - \hat{f}_n(x_*))$$

$$= \tau^2 + \underbrace{E(f(x_*) - \hat{f}_n(x_*))^2}_{Bias^2} + \underbrace{Var(f(x_*) - \hat{f}_n(x_*))}_{Variance} = \underbrace{I^2}_{\text{irreducible error}} + Bias^2 + Variance.$$

* relation with bias and variance in inference:

$$Bias[\hat{f}_n] = E[\hat{f}_n(x) - f(x)] = E[\hat{\theta}_n^T x - \theta^{*T}x] = E[\hat{\theta}_n - \theta^*]^T x = Bias(\hat{\theta}_n)^T x.$$

and similarly, $Variance(\hat{f}_n) = x^T[Var(\hat{\theta}_n)]x$.

~~the irreducible error only exists in test setting~~, since it is an artifact of noise in test sample.

[noise in the training data contributes to Variance term, noise in test sample manifests as irreducible error].

+ ~~in inference setting~~, observed bias-variance tradeoff

+ in prediction setting, choose λ to be the value that minimizes the squared error in cross-validation. (L_2-regularization LG)

* relation with overfitting and underfitting:

+ overfitting relates to having high variance $\xrightarrow{\text{fight overfit}}$ { increase regularization; increase data size; decrease # features; use a smaller model.

+ underfitting relates to having high bias $\xrightarrow{\text{fit underfit}}$ { decrease regularization; use more features; use a larger model.

+ training error can be treated as the amount of bias in model/estimator. [high bias → underfitting]

+ gap between cross-validation error and training error can be treated as Variance.

We should always analyse the model performance by looking at training error and cross-validation error simultaneously. (then take steps to address either Bias or Variance purposefully)

[Note:] steps taken to fight overfitting (i.e. high Variance) generally do not help fight underfitting, it is futile, for example, to obtain more data (fight high Variance) when the training error is high (symptom of high Bias), vice versa.

Decision Trees

Generate approximate solution of region via greedy, top-down, recursive partitioning.
(select leaf node, feature, threshold *ONE* at a time.)

- take the loss function as splitting heuristic
- start at original input region X and split into child regions by *THRESHOLDING on SINGLE feature*.

given a parent region R_p, feature index j, and threshold $t \in \mathbb{R}$, we obtain child regions R_1, R_2 as:

$$R_1 = \{X \mid X_j < t, X \in R_p\}, \quad R_2 = \{X \mid X_j \geq t, X \in R_p\}$$

Within greedy partitioning framework, we want to select (leaf, feature, threshold) that maximize the decrease in loss: $L(R_p) - (|R_1|L(R_1) + |R_2|L(R_2))/(|R_1|+|R_2|)$ [child region decrease the loss of R_p]

specifically for classification problem, misclassification loss on R: $L_{mis}(R) = 1 - \max_c(\hat{p}_c)$ (1)

where $\hat{p}_c$ denotes the (proportion of) samples $\in R$ that are of class c. [the proportion of samples that would be missed if predicting the majority class for region R]

We notice that misclassification loss (eq.11) is not very sensitive to changes in class probabilities, propose a more sensitive loss (cross-entropy loss): $L_{cross}(R) = -\sum_c \hat{p}_c \log_2 \hat{p}_c$ [information gained through splitting]

[number of bits needed to specify the outcome (class) given the distribution is known]

since the cross-entropy loss is strictly concave, as long as $\hat{p}_1 \neq \hat{p}_2$ (in a binary classification problem), $L(R_p) > (|R_1|L(R_1) + |R_2|L(R_2))/(|R_1|+|R_2|)$, which make cross-entropy loss more sensitive/informative.

For regression settings, we can directly use squared loss to select splits:

$$L_{loss} = L_{squared}(R) = \sum_{i \in R} (y_i - \hat{y})^2 / |R|$$

final prediction for region R now becomes: $\hat{y} = \sum_{i \in R} y_i / |R|$ (plug in)

• Regularization

Decision tree is high variance and low biased, thus we turn to techniques of regularization:

* Minimum Leaf Size. Stop splitting R if cardinality falls below a fixed threshold.
* Maximum Depth. Stop splitting R if more than a fixed threshold of splits were taken to reach R.
* Maximum Number of Nodes. Stop if tree has more than a fixed threshold of leaf nodes.
* Pruning. *FULLY GROWN OUT* the tree and then pruning away nodes that minimally decrease loss.

Notice it's not a good idea to prune along the growing process since single-feature selection would miss higher order interactions if constrained on loss decrease.

• Ensemble Learning

Some of the primary benefits of decision trees are: categorical variable support, fast, interpretable.
While disadvantages include: * High variance *(imagine a separate region for every data point)
* Poor Additive Modeling *. (compare with simple linear model.)
leading to low predictive accuracy of individual decision tree.

→ * Bagging (Bootstrap Aggregation) ← [decrease Variance]

Main idea: assume training set is the true population (p), let card(S) = n, then sample n (S) bootstrap samples from S, and do it with replacement (always holds $|s| = n$)

Bootstrap samples $z_1, z_2 \cdots z_m$, train model G_i on z_i then $G(x) = \frac{\sum_{i=1}^{m} G_i(x)}{m}$. second term goes down as m↑.

Bootstrap drives down correlation between X_is', since $Var(\bar{x}) = \rho\sigma^2 + \frac{1-\rho}{m}\sigma^2$ ↗ sample more models until the second term stops changing.

Note: The tradeoff for reducing variance is that models get less complicated so bias goes up.

ρ decrease as #samples increase but has a lower bound since samples drawn can't be completely uncorrelated.

→ * Random Forest ←

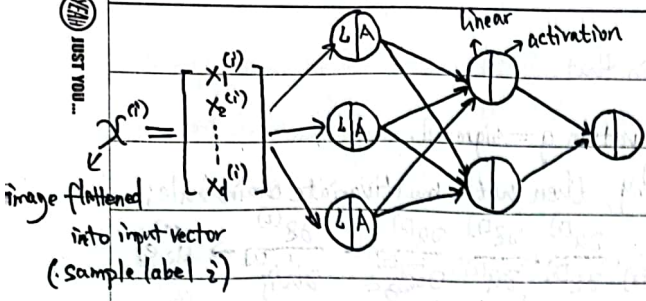
1. Introduce more randomization into each decision tree to further decorrelate the samples.
2. At each split, consider only a fraction of total features. (typically $\sqrt{p}$ features per split)

↗ randomly select features ↖ sampling with replacement

Neural Network

Hidden Layer: do not have the ground truth/training value for hidden units, the network itself is responsible for figuring out intermediate structures. Let $a_j^{[i]}$ denote the output from the j^{th} hidden unit in the i^{th} layer. ($i \geq 1$, $i = 0$ denoting input layer).

Architecture



$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$ ← image flattened into input vector (i: sample label i)

Parameters

With perspective of Forward Propagation:

$z^{[1]} = W^{[1]} x^{(i)} + b^{[1]}$ [linear part]

$a^{[1]} = g(z^{[1]})$ [activation]

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$, $a^{[2]} = g(z^{[2]})$

$z^{[3]} = W^{[3]} a^{[1]} + b^{[3]}$, $\hat{y}^{(i)} = g(z^{[3]}) = a^{[3]}$

In neural networks, with purpose of optimizing training speed, *vectorization* is performed in substitution of for loops, here we denote

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{3\times 1}} = \underbrace{\begin{bmatrix} -W_1^{[1]T}- \\ -W_2^{[1]T}- \\ \vdots \\ -W_3^{[1]T}- \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{3\times d}} \underbrace{\begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}}_{x^{(i)} \in \mathbb{R}^{d\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{3\times 1}}$$

, with a total of 3d+4 parameters

[With row being set of parameter for neuron]

- Initializing parameters

First, note that we can't initialize all parameters to zero, this would cause output of first layer to be zero regardless of input. Moreover, the output of network would always be 0.5 regardless of input. One solution is to randomly initialize parameters to small values (e.g. $\mathcal{N}(0, 0.1)$).

In practice, there is something better than random initialization, namely Xavier/He initialization which initializes weights: $W^{[l]} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}}\right)$, where $n^{[l]}$ is the number of neurons in layer l. [Intuition: consider variance(input to layer l) to be $\sigma^{(in)}$ and variance(output of layer l) to be $\sigma^{(out)}$, Xavier/He initialization encourages $\sigma^{(in)}$ being SIMILAR to $\sigma^{(out)}$]. Also, note that we can't initialize all parameters to be the same non-zero value, which would cause each element of $a^{[l]}$ to be the same, as a result, when computing gradient, all neurons in a layer will be equally responsible for anything contributed to final loss, then each neuron will be updated the same way and learn the exact same thing. [symmetry]

- Optimization

→ logistic regression for binary classification

Using the logistic loss $\mathcal{L}(\hat{y}, y) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$, we can perform SGD to optimize parameters in each layer l:

$$W^{[l]} := W^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial W^{[l]}} \quad (1), \qquad b^{[l]} := b^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial b^{[l]}} \quad (2)$$

the derivations we need to compute are $\left(\frac{\partial \mathcal{L}}{\partial W^{[1]}}, \frac{\partial \mathcal{L}}{\partial W^{[2]}}, \frac{\partial \mathcal{L}}{\partial W^{[3]}}\right)\left(\frac{\partial \mathcal{L}}{\partial b^{[1]}}, \frac{\partial \mathcal{L}}{\partial b^{[2]}}, \frac{\partial \mathcal{L}}{\partial b^{[3]}}\right)$.

We will be calculating $\frac{\partial \mathcal{L}}{\partial W^{[1]}}$ as an example.

[Note:] Loss function selection: ① regression $\mathcal{L}(\hat{y}, y) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2$.

② softmax regression – cross entropy loss $\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{k} 1\{y = i\}\log\hat{y}_i$

observe that, $\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}^{[2]}} & \frac{\partial \mathcal{L}}{\partial w_{12}^{[2]}} & \frac{\partial \mathcal{L}}{\partial w_{13}^{[2]}} \\ \frac{\partial \mathcal{L}}{\partial w_{21}^{[2]}} & \frac{\partial \mathcal{L}}{\partial w_{22}^{[2]}} & \frac{\partial \mathcal{L}}{\partial w_{23}^{[2]}} \end{bmatrix}$, and obtain that

$\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \frac{\partial}{\partial z^{[2]}}\left[-y\log\hat{y} - (1-y)\log(1-\hat{y})\right] = g(z^{[2]}) - y = a^{[2]} - y$, with activation g = sigmoid.

then with multivariate chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_{ij}^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w_{ij}^{[2]}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}}}_{(a^{[2]}-y)} \cdot \underbrace{\frac{\partial z^{[2]}}{\partial a^{[1]}}}_{(W^{[2]})} \cdot \underbrace{\frac{\partial a^{[1]}}{\partial z^{[1]}}}_{g(z^{[1]})(1-g(z^{[1]}))} \cdot \underbrace{\frac{\partial z^{[1]}}{\partial w_{ij}^{[1]}}}_{\to a_j^{[1]} e_i}$$

$(a^{[2]}-y) \in \mathbb{R}^{1\times 1}$, $(W^{[2]}) \in \mathbb{R}^{1\times 2}$, $g(z^{[1]})(1-g(z^{[1]})) \in \mathbb{R}^{2\times 2}$, $a_j^{[1]} e_i \in \mathbb{R}^{2\times 1}$

$$= \underbrace{(a^{[2]}-y)W^{[2]}}_{1\times 2} \circ g'(z^{[1]}) \underbrace{a_j^{[1]} e_i}_{2\times 1}$$

$$= \left[(a^{[2]}-y)W^{[2]} \circ g'(z^{[1]})\right]_i a_j^{[1]} \Rightarrow \frac{\partial \mathcal{L}}{\partial W^{[1]}} = \left[(a^{[2]}-y)W^{[2]} \circ g'(z^{[1]})\right] a^{[0]T}.$$

Notice when calculating $\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial W^{[2]}} = \underbrace{(a^{[2]}-y)}_{\in \mathbb{R}^{1\times 2}} a^{[1]T}$, the first term has been calculated in the derivation of $\frac{\partial \mathcal{L}}{\partial W^{[2]}}$, and $\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \underbrace{\frac{\partial \mathcal{L}}{\partial z^{[2]}}}_{\text{calculated in } \frac{\partial \mathcal{L}}{\partial W^{[2]}}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}}$.

The idea is, in order for errors to back propagate, we need to go through variables that are connected to each other and apply the multivariate chain rule of calculus.

Caching: Notice when expressing $\frac{\partial \mathcal{L}}{\partial W^{[1]}}$ we end up with $\frac{\partial \mathcal{L}}{\partial W^{[1]}} = W^{[2]T} \circ g^{[1]}(1-a^{[1]})(a^{[2]}-y)a^{[0]T}$, where $a^{[0]}, a^{[1]}$ are values there were computed during forward propagation, it's important to store all values that are computed during forward propagation for efficient backward update.

When optimizing variables, we usually won't choose stochastic gradient descent since loss for a single example might be noisy, while $J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}^{(i)}$ gives more accurate gradients but is computationally expensive. In practice, prioritize using mini-batch gradient descent, with $J_{mb} = \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}^{(i)}$, where B is the number of examples in the mini-batch.

Momentum: (for training phase optimization) Consider mini-batch stochastic gradient, for any layer l, the update rule becomes:

$$\begin{cases} V_{dW^{[l]}} = \beta V_{dW^{[l]}} + (1-\beta)\frac{\partial J}{\partial W^{[l]}} \\ W^{[l]} = W^{[l]} - \alpha V_{dW^{[l]}} \end{cases}$$

the weight update now depends on two terms: the cost J at this update step and velocity $V_{dW^{[l]}}$, with relative importance controlled by β. [velocity $V_{dW^{[l]}}$ will keep track of gradient over time].

Choice of Activation Functions:

One way to improve the performance of neural network is to use different activation functions.

Usually choose to use same activation function within the same layer, with some common choices being:

Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ ReLU (Rectified Linear Unit) $r(x) = \max(x, 0)$ $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$\sigma'(x) = \sigma(x)(1-\sigma(x))$. $r'(x) = \mathbb{1}\{x > 0\}$. $\tanh'(x) = 1 - \tanh(x)^2$.

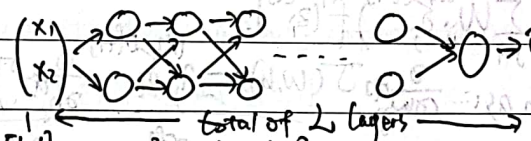
Reasons to choose these activations come along with different tasks, when doing classification work with labeling being $\{0,1\}$, then sigmoid is a common choice; In reinforcement learning when there is necessity for negative reward, tanh would be a reasonable choice; When performing regression tasks, ReLU might appear to be an ideal choice. The problem with sigmoid is that gradients are small when x becomes big/small, namely gradient vanishes at two sides, making it hard to train network at the early stage since gradient drops too slowly. In this case, having a ReLU layer as the first hidden layer might be a good adjustion. (avoid this using initialization technique.)

[Note:] one initialization technique to avoid input being too big/small is perform $\tilde{x} = x - \mu / \sigma$. What this causes is the gradient of Loss function (contour) always points to the center. (μ, σ from train set)

* $\tilde{x}_{test} = \frac{x_{test} - \mu_{train}}{\sigma_{train}}$ *.

- Vanishing/Exploding Gradients.

Derive a simple example of forward propagation:



with activation $g: z \mapsto z$.
bias $b := 0$.

total of L layers

then $\hat{y} = w^{[L]} a^{[L-1]} = w^{[L]} \cdot w^{[L-1]} a^{[L-2]} = \cdots = w^{[L]} \cdot w^{[L-1]} \cdots w^{[1]} \cdot x$.

if we assume $w^{[L]} = \mathrm{diag}(1+\varepsilon, 1+\varepsilon)$, with ① $\varepsilon > \frac{1}{2} \Rightarrow \hat{y} = \mathrm{diag}((1+\varepsilon)^L, (1+\varepsilon)^L) x$, if L is large (enough), then $\hat{y}$ would explode, similarly ② $\varepsilon < -\frac{1}{2}$, $\hat{y}$ will be extremely small if layer numbers are large.

Tackling: we would prefer weights to be as close to 1 as possible, so a reasonable (not perfect) way is to initialize input and weights properly. With sigmoid activation on layer l, set $w^{[l]} = \underset{\text{randomly set}}{\text{shape}^{[l]}} \times \sqrt{\frac{1}{n^{[l-1]}}}$.

With ReLU as activation, $w^{[l]} := \mathrm{randn}(\mathrm{shape}^{[l]}) \times \sqrt{\frac{2}{n^{[l-1]}}}$.

With tanh as activation, Xavier Initialization proposes $w^{[l]} \sim \mathcal{N}(0, \sqrt{\frac{1}{n^{[l-1]}}})$, if we also considers backward gradient, then He initialization with $w^{[l]} \sim \mathcal{N}(0, \sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}})$ would be a good choice.

Regularization and Parameter Sharing

Pile all parameters in layer l to form a single matrix W. Add L_2-norm regularization to the cost function:

$$J_{L_2} = J + \frac{\lambda}{2}\|W\|_2^2 = J + \frac{\lambda}{2}\sum_{ij}|W_{ij}|^2 = J + \frac{\lambda}{2}W^TW.$$

where λ is an arbitrary value with a larger value indicating more regularization, then the update rule becomes: $W = W - \alpha\frac{\partial J}{\partial W} - \alpha\frac{\lambda}{2}\frac{\partial W^TW}{\partial W} = (1-\alpha\lambda)W - \alpha\frac{\partial J}{\partial W}$. This indicates that with L_2 regularization, every update will include penalization $(\alpha\lambda W)$ depending on W.

Recall in logistic regression, the linear part $z = \theta^Tx$ demands θ_1 always look at the top left pixel(x_1), if all training sample appear at the right bottom, then θ_1 will never get trained. In order to let θ_i gain more insights about image, we turn to convolutional neural networks, and let θ be a matrix (kernel), and activation becomes $a = \theta * X$. [intuition: let θ "seen" all pixels of image].

Backpropagation

For each node i in layer l, let $\delta_i^{(l)}$ denote how much the node is responsible for errors in output.

Then backpropagation algorithm can be described as:

1. Perform a feedforward pass, compute activations $a^{[l]}$ for all layers $(l = 1, 2, \cdots n_l)$.
2. Let $\delta_i^{[n_l]} = \frac{\partial}{\partial z_i^{[n_l]}}\frac{1}{2}\|y - h_{W,b}(x)\|^2 = (a_i^{[n_l]} - y_i)\cdot f'(z_i^{[n_l]})$.
3. For $l = n_l-1, n_l-2, \cdots 1$, set $\delta_i^{[l]} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{[l]}\delta_j^{[l+1]}\right) f'(z_i^{[l]})$.
4. Compute the partial derivatives given as: $\frac{\partial}{\partial W_{ij}^{[l]}}J(W,b) = a_j^{[l]}\delta_i^{[l+1]}$; $\frac{\partial}{\partial b_i^{[l]}}J(W,b) = \delta_i^{[l+1]}$.

Using vectorization and "$\circ$" denote Hadamard product, we define $\delta^{[l]} = \nabla_{z^{[l]}}\mathcal{L}(\hat{y}, y)$.

For the output layer N we have $\delta^{[N]} = \nabla_{z^{[N]}}\mathcal{L}(\hat{y}, y) = (a^{[N]} - y)\circ g'(z^{[N]})$.

For $l = N-1, N-2, \cdots 1$, we have $\delta^{[l]} = (W^{[l+1]T}\delta^{[l+1]})\circ g'(z^{[l]})$, and the gradients become:

$\nabla_{W^{[l]}}J(W,b) = \delta^{[l]}a^{[l+1]T}$, $\nabla_{b^{[l]}}J(W,b) = \delta^{[l]}$.

For output layer N, $\delta^{[N]} = \nabla_{z^{[N]}}\mathcal{L}(\hat{y}, y) = \nabla_{\hat{y}}\mathcal{L}(\hat{y}, y)\circ\underbrace{g^{[N]\prime}(z^{[N]})}_{\text{elementwise derivative w.r.t. } z^{[N]}}$.

For $l = N-1, N-2, \cdots 1$, $\delta^{[l]} := (W^{[l+1]T}\delta^{[l+1]})\circ g'(z^{[l]})$, $\nabla_{W^{[l]}}J(W,b) = \delta^{[l]}a^{[l+1]T}$, $\nabla_{b^{[l]}}J(W,b) = \delta^{[l]}$.

Learning Theory

To begin foray into learning theory, let's restrict attention to binary-classification in which labels are $y \in \{0,1\}$. Assume training set being $S = \{(x^{(i)}, y^{(i)}), i = 1,2,\cdots n\}$, where training samples $(x^{(i)}, y^{(i)})$ drawn independently identically distributed (i.i.d) from probability distribution D.

Define training error (empirical risk) to be $\hat{\varepsilon}(h) = \frac{1}{n}\sum_{i=1}^{n} 1\{h(x^{(i)}) \neq y^{(i)}\}$. for a hypothesis h.

Define generalization error $\varepsilon(h) = P_{(x,y)\sim D}(h(x) \neq y)$. [generate test example from D, probability of missing]

Note: Assumption of test example drawn from same distribution is one of the PAC assumptions, among which ① training and testing on same distribution ② independently drawn training examples were the most important. [probably approximately correct].

Consider setting of linear regression, let $h_\theta(x) = 1\{\theta^T x \geq 0\}$, with *empirical risk minimization* we select $\hat{\theta} = \arg\min_\theta \hat{\varepsilon}(h_\theta)$.

- Define hypothesis class $\mathcal{H}$ to be set of all classifiers considered by the learning algorithm. Then ERM can be considered to happen within $\mathcal{H}$, thus $\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\varepsilon}(h)$.

- Consider the case of $|\mathcal{H}|$ being finite, say $\mathcal{H}$ is set of k functions mapping X to $\{0,1\}$.

Take any fixed $h_i \in \mathcal{H}$, let random variable $z = 1\{h_i(x) \neq y\}$ where $(x,y) \sim D$. then z is an indicator of test sample misclassification, similarly define training sample misclassification $z_j = 1\{h_i(x^{(j)}) \neq y^{(j)}\}$.

Thus $\hat{\varepsilon}(h_i) = \frac{1}{n}\sum_{j=1}^{n} z_j$, $\varepsilon(h_i) = \mathbb{E}[z(z_j)]$.

- Lemma. (Union Bound) $P(A_1 \cup A_2 \cup A_3 \cdots \cup A_k) \leq P(A_1) + P(A_2) + \cdots + P(A_k)$.

- Lemma. (Hoeffding Inequality) Let $z_1, z_2, \cdots, z_n$ be n iid random variables drawn from a Bernoulli(ϕ) distribution, let $\hat{\phi} = \frac{1}{n}\sum_{i=1}^{n} z_i$, and let $\gamma > 0$ be fixed, then

$$P(|\phi - \hat{\phi}| > \gamma) \leq 2\exp(-2\gamma^2 n).$$

With Chernoff bound (Hoeffding), we obtain $P(|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) \leq 2\exp(-2\gamma^2 n)$ (1) [for one h_i].

Our goal now is to extrapolate from one particular h_i to all h_i's $\in \mathcal{H}$. To do so, let's denote A_i as the event $|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma$, then with Eq(1):

$$P(\exists h_i \in \mathcal{H}. |\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) = P(A_1 \cup A_2 \cup A_3 \cdots \cup A_k) \leq 2k\exp(-2\gamma^2 n)$$

$$\Rightarrow \quad P(\forall h_i \in \mathcal{H}. |\varepsilon(h_i) - \hat{\varepsilon}(h_i)| \leq \gamma) \geq 1 - 2k\exp(-2\gamma^2 n) \quad \text{[uniform convergence result]}.$$

let $\delta = 2k\exp(-2\gamma^2 n)$. then with any two within $[n, \gamma, \delta]$ fixed, we can solve the bound

for the other. E.g. Given γ and $\delta>0$, how large must n be before we can guarantee that with probability at least $(1-\delta)$, training error with be within γ for generalization error?

$\Rightarrow \delta \geq 2k\exp(-2\gamma^2 n)$ for fixed δ,γ, then $n \geq \frac{1}{2\gamma^2}\log\frac{2k}{\delta}$ [sample complexity].

Similarly, let n and δ be fixed, then with probability $1-\delta$, we have that for all $h\in\mathcal{H}$,

$$|\hat{\varepsilon}(h)-\varepsilon(h)| < \sqrt{\frac{1}{2n}\log\frac{2k}{\delta}}.$$

Recall $\hat{h}=\arg\min_{h\in\mathcal{H}}\hat{\varepsilon}(h)$ is the output hypothesis that minimizes empirical error, define $h^*=\arg\min_{h\in\mathcal{H}}\varepsilon(h)$ the hypothesis that performs best on test sample, we observe a bound for $\varepsilon(\hat{h})$ using $\varepsilon(h^*)$:

$$\varepsilon(\hat{h}) \leq \hat{\varepsilon}(\hat{h})+\gamma \leq \hat{\varepsilon}(h^*)+\gamma \leq \varepsilon(h^*)+2\gamma.$$

- Theorem. Let $|\mathcal{H}|=k$ and let any n,δ be fixed, with probability at least $1-\delta$, the inequality holds:

$$\varepsilon(\hat{h}) \leq \min_{h\in\mathcal{H}}\varepsilon(h)+2\sqrt{\frac{1}{2n}\log\frac{2k}{\delta}} \quad (2).$$

Using Ineq(2), we can have a interpretation of bias-variance tradeoff. Say we switched to a larger hypothesis set $\mathcal{H}'\supseteq\mathcal{H}$. Then first term in (2) will not increase, hence, by learning using a larger hypothesis set, "bias" will only decrease (not strictly). However, as $|\mathcal{H}|$ increases, second term in (2) also increase, this increase corresponds to "variance" increasing when using a larger hypothesis class.

- Corollary (sample complexity bound). Let $|\mathcal{H}|=k$, and let any δ,γ be fixed, then for $\varepsilon(\hat{h}) \leq \min_{h\in\mathcal{H}}\varepsilon(h)+2\gamma$ to hold with probability at least $1-\delta$, it suffices that

$$n \geq \frac{1}{2\gamma^2}\log\frac{2k}{\delta} = O\left(\frac{1}{\gamma^2}\log\frac{k}{\delta}\right) = O_{\gamma,\delta}(k).$$

• The Case of infinite $\mathcal{H}$.

- Define. Vapnik-Chervonenkis dimension of hypothesis class $\mathcal{H}$, written $VC(\mathcal{H})$ to be size of largest set that is shattered by $\mathcal{H}$. Given a set $S=\{x^{(1)},\cdots,x^{(D)}\}$ of points $x^{(i)}\in\mathcal{X}$ (input space), we say that $\mathcal{H}$ shatters S if for any set of labels $\{y^{(1)},y^{(2)},\cdots y^{(D)}\}$ there exists some $h\in\mathcal{H}$ s.t. $h(x^{(i)})=y^{(i)}$ for $\forall i=1,2,\cdots D$. [Notice the set S is given in advance, not arbitrary]

[Note:] In order to prove that $VC(\mathcal{H})$ is at least D, we need to show only there's *at least one* set of cardinality D that $\mathcal{H}$ can shatter.

- Theorem. Let $\mathcal{H}$ be given and $VC(\mathcal{H})=D$, then with probability at least $1-\delta$, we have that for all $h\in\mathcal{H}$, $|\varepsilon(h)-\hat{\varepsilon}(h)| \leq O\left(\sqrt{\frac{D}{n}\log\frac{n}{D}+\frac{1}{n}\log\frac{1}{\delta}}\right) \Rightarrow \varepsilon(\hat{h}) \leq \varepsilon(h^*)+O\left(\sqrt{\frac{D}{n}\log\frac{n}{D}+\frac{1}{n}\log\frac{1}{\delta}}\right)$.

In other words, if a hypothesis class has finite VC dimension, then uniform convergence occurs as n becomes larger.

- Corollary. For $|\varepsilon(h)-\hat{\varepsilon}(h)|\le\gamma$ to hold for $\forall h\in\mathcal{H}$. (hence $\varepsilon(\hat{h})\le\varepsilon(h^*)+2\gamma$ with probability at least $1-\delta$) it suffices that $n=O_{\gamma,\delta}(D)$.

To conclude, for a given hypothesis class $\mathcal{H}$ and for algorithm that tries to minimizes training error, # training samples needed to achieve generalization error close to that of optimal classifier is usually roughly linear in the number of parameters (VC($\mathcal{H}$)) of $\mathcal{H}$.

K-means Clustering

We now begin our foray into unsupervised learning problem. In clustering problem, we are given a training set $\{x^{(1)}, x^{(2)}, \cdots x^{(m)}\}$ without labeling, and wish to group the data into cohesive "clusters". Before writing out the algorithm for k-means clustering, we should decide the value of K. There are Information-Theoretic criteria like Akaike information criterion (AIC) to determine optimal value of K, but in practice, K can be decided using prior knowledge of properties of data set and implementation needs.

The k-means clustering algorithm:

1. Initialize cluster centroids $\mu_1, \mu_2, \cdots \mu_K \in \mathbb{R}^n$. (Metric: randomly select k points from data set, say $\mathcal{X}_{init}=\{x^{(i)}, x^{(i+1)}, \cdots x^{(i+k)}\}$ and set $\mu_t := x^{(i+t)}$.)

2. Repeat until convergence: {

For every $i\in\{1,2,\cdots m\}$, set $c^{(i)} := \arg\min_j \|x^{(i)}-\mu_j\|_2^2$ [color the point]

For each $j\in\{1,2,\cdots K\}$, set $\mu_j := \sum_{i=1}^{m} 1\{c^{(i)}=j\}x^{(i)} \Big/ \sum_{i=1}^{m} 1\{c^{(i)}=j\}$.

[update μ_j to the center of points that share the j^{th} color]. }

In order to show the k-means algorithm converges, we will define the distortion function to be:

$$J(c,\mu)=\sum_{i=1}^{m}\|x^{(i)}-\mu_{c^{(i)}}\|_2^2$$

Thus, J measures the sum of squared distances between each training example $x^{(i)}$ and the cluster centroid $\mu_{c^{(i)}}$ to which it has been assigned. Let's take a digression to talk about the coordinate ascent algorithm.

- Coordinate Ascent

Consider the unconstrained optimization problem $\max_{\alpha} W(\alpha_1, \alpha_2, \cdots \alpha_n)$. Apart from gradient ascent and Newton's method, we introduce the coordinate ascent algorithm:

Loop until convergence: {

For $i=1,2,\cdots,n$, $\{\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1,\cdots\alpha_{i-1},\hat{\alpha}_i,\alpha_{i+1},\cdots\alpha_n)\}$ }.

Inside the loop, we hold all variables except for some α_i fixed, and reoptimize W w.r.t. just the parameter α_i, the ordering can be decided using heuristics to see which variable expect to allow us make the largest increase in $W(\alpha)$.

Note: When W happens to be of such a form that "argmax" can be performed efficiently, coordinate ascent can be a fairly efficient algorithm.

It can be shown that K-means algorithm is coordinate ascent performed on $J(c,\mu)$:

$\mu_j := \arg\min_{\hat{\mu}_j} J(c,\mu) = \arg\min_{\hat{\mu}_j} \sum_{i=1}^{m} \|x^{(i)}-\mu_{c^{(i)}}\|_2^2$ [since J is quadratic w.r.t. μ_j, it make sense to employ coordinate descent].

Thus, $\nabla_{\mu_j} J(\mu) = 0 \Rightarrow \nabla_{\mu_j} \sum_{i=1}^{m} 1\{c^{(i)}=j\}(x^{(i)}-\mu_j)^T(x^{(i)}-\mu_j)$

$= 2\sum_{i=1}^{m} 1\{c^{(i)}=j\}(x^{(i)}-\mu_j) = 0 \Rightarrow \mu_j := \sum_{i=1}^{m} 1\{c^{(i)}=j\}x^{(i)} \Big/ \sum_{i=1}^{m} 1\{c^{(i)}=j\}$.

[the update formula for $c^{(i)}$ can be obtained similarly when holding μ fixed and minimizing J w.r.t. $c^{(i)}$]

Thus, J must monotonically decrease. But notice J is non-convex, and so coordinate descent is not guaranteed to converge to global minimum. One common thing to do is run K-means many times, using different random initial value for cluster centroids, then pick the clustering that gives the lowest distortion $J(c,\mu)$.

Expectation-Maximization Algorithm

• Mixture of Gaussians.

Suppose we are given training set $\{x^{(1)},\cdots,x^{(m)}\}$ and wish to fit Gaussians to "clusters". Formally, we should specify a joint distribution $p(x^{(i)},z^{(i)}) = p(x^{(i)}|z^{(i)})\,p(z^{(i)})$, with $z^{(i)}$ being latent (unobserved) variables. Here, $z^{(i)} \sim \text{Multinomial}(\phi)$ (where $\phi_j \geq 0$, $\sum_{j=1}^{k}\phi_j = 1$ and parameter ϕ_j gives $p(z^{(i)}=j)$), and $x^{(i)}|z^{(i)}=j \sim \mathcal{N}(\mu_j,\Sigma_j)$. We let k denote the number of values that $z^{(i)}$ can take on (e.g. if $z^{(i)} \sim \text{Bernoulli}(\phi)$ then $k=2$ which is the case of GDA). Thus, our model posits that each $x^{(i)}$ was generated by ① randomly choosing $z^{(i)}$ from $\{1,2,\cdots k\}$ and ② $x^{(i)}$ was drawn from one of k Gaussians depending on $z^{(i)}$.

Notice if $z^{(i)}$'s are unknown, then the maximization of joint likelihood:

$\ell(\phi,\mu,\Sigma) = \sum_{i=1}^{m} \log p(x^{(i)};\phi,\mu,\Sigma) = \sum_{i=1}^{m} \log \sum_{z^{(i)}=1}^{k} p(x^{(i)}|z^{(i)})\,p(z^{(i)};\phi)$. [summing over k^n possibilities]

is relatively hard, but if like GDA we knew what the $z^{(i)}$'s were, then likelihood can be simplified as:

$$\ell(\phi,\mu,\Sigma) = \sum_{i=1}^{m}\sum_{j=1}^{k} 1\{z^{(i)}=j\}\left[\log p(x^{(i)}|z^{(i)}=j;\mu_j,\Sigma_j) + \log p(z^{(i)}=j)\right]$$

Maximizing this w.r.t. ϕ,μ and Σ gives the parameters:

$$\cdot\ \phi_j = \frac{1}{m}\sum_{i=1}^{m} 1\{z^{(i)}=j\},\quad \cdot\ \mu_j = \sum_{i=1}^{m} 1\{z^{(i)}=j\}x^{(i)} \Big/ \sum_{i=1}^{m} 1\{z^{(i)}=j\},\quad \cdot\ \Sigma_j = \frac{\sum_{i=1}^{m} 1\{z^{(i)}=j\}(x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T}{\sum_{i=1}^{m} 1\{z^{(i)}=j\}}.$$

EM algorithm comes in as we try to determine the value of $z^{(i)}$, the E-step in EM tries to "guess" the values of $z^{(i)}$ and pass it to the M-step. Iterative EM algorithm:

Repeat until convergence: {

(E-step) For each i,j, set $w_j^{(i)} := p(z^{(i)}=j|x^{(i)};\phi,\mu,\Sigma) = \frac{p(x^{(i)}|z^{(i)}=j)p(z^{(i)}=j)}{\sum_{j=1}^{k} p(x^{(i)}|z^{(i)}=j)p(z^{(i)}=j)}$.

(M-step) Update the parameters:

$$\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)},\quad \mu_j := \sum_{i=1}^{m} w_j^{(i)}x^{(i)} \Big/ \sum_{i=1}^{m} w_j^{(i)},\quad \Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)}(x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}.$$

}

Basically, we are substituting $1\{z^{(i)}=j\}$ with $\mathbb{E}[1\{z^{(i)}=j\}] = w_j^{(i)}$ in the M-step. The values $w_j^{(i)}$ calculated in E-step represent "soft" guesses for values of $z^{(i)}$. [probability instead of direct assignment as we did in k-means]

Similar to k-means algorithm, EM is also susceptible to local optima, so it will be necessary to reinitialize at several different initial parameters and choose ϕ_j, μ_j, Σ_j that achieves $\max \ell(\phi,\mu,\Sigma)$.

• EM Convergence

Say we wish to fit a model $p(x,z)$ to the data $\{x^{(1)},\dots x^{(m)}\}$, where the likelihood is given by $\ell(\theta) = \sum_{i=1}^{m}\log p(x^{(i)};\theta) = \sum_{i=1}^{m}\log\sum_{z^{(i)}} p(x^{(i)},z^{(i)};\theta)$, as we introduce latent variables, taking the joint distribution and marginalizing out these latent variables. [if $z^{(i)}$ is continuous, then substitute with integral].

Explicitly maximizing the estimate of θ may be hard given $z^{(i)}$ are unobserved. In this setting, EM algorithm proposes to repeatedly construct a lower-bound on $\ell(\theta)$ (E-step) and then optimizing this constructed lower bound. (M-step).

For each i, let Q_i be some distribution (left to be chosen) over the z's, s.t. $\sum_z Q_i(z)=1$, $Q_i(z)\geq 0$, using Jensen's inequality we obtain:

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

$$\geq \sum_i\sum_{z^{(i)}} Q_i(z^{(i)})\log p(x^{(i)},z^{(i)};\theta)/Q_i(z^{(i)}) \quad (1).$$

1. log being concave
2. Jensen: $f(\mathbb{E}X)\geq \mathbb{E}f(X)$ for concave f.

, worth taking a digression and talk about the conditions required for Jensen inequality to hold.
, case of f taking vector-valued inputs, f is convex $\Rightarrow$ (if f has a Hessian) H is positive semi-definite.
, over, if f is strictly convex ($H>0$), then $E[f(X)]=f(EX)$ holds true $\Leftrightarrow$ X is constant.

We now have Eq(1) giving a lower bound on $\ell(\theta)$, to make the bound tight for a particular value θ, we demand the inequality becomes equality, thus Jensen inequality becomes equality as we satisfy the condition $P(X=E(X))=1$, namely, $\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}=\text{Const}$. Since $\sum_z Q_i(z^{(i)})=1$,

$$Q_i(z^{(i)})=P(x^{(i)},z^{(i)};\theta)\Big/\sum_z p(x^{(i)},z;\theta)=P(x^{(i)},z^{(i)};\theta)\Big/p(x^{(i)};\theta)=p(z^{(i)}|x^{(i)};\theta)$$

Thus, simply set Q_i's to be the posterior distribution of $z_i^{(i)}$'s. With this tight lower-bound, we then maximize Eq(1) RHS w.r.t the parameters, the EM algorithm can then be written as:

Repeat until convergence {

(E-step) For each i, set $Q_i(z^{(i)}):=p(z^{(i)}|x^{(i)};\theta)$.

(M-step). Set $\theta:=\arg\max_\theta \sum_i \sum_{z^{(i)}} Q_i(z^{(i)})\log\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}$ }

With the above described algorithm, we obtain $\ell(\theta^{(t)})=\sum_i\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)})\log\frac{P(x^{(i)},z^{(i)};\theta^{(t)})}{Q_i^{(t)}(z^{(i)})}$ in the t^{th} iteration, parameters $\theta^{(t+1)}$ are then obtained by explicitly maximize RHS, thus:

$$\ell(\theta^{(t+1)})\underset{\text{Jensen}}{\geq}\sum_i\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)})\log\frac{P(x^{(i)},z^{(i)};\theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})}\underset{\max_\theta \text{RHS}}{\geq}\sum_i\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)})\log\frac{P(x^{(i)},z^{(i)};\theta^{(t)})}{Q_i^{(t)}(z^{(i)})}=\ell(\theta^{(t)})$$

[end of t^{th} iteration]

Hence, we showed that EM causes the likelihood to converge monotonically. One reasonable convergence test would be to check if the increase in $\ell(\theta)$ between successive iterations is smaller than some preset tolerance parameter. If we define $J(Q,\theta)=\sum_i\sum_{z^{(i)}} Q_i(z^{(i)})\log\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}$, then EM can also be viewed as coordinate ascent on J, in which E-step maximizes J w.r.t Q and M-step maximizes J w.r.t θ.

- Derivation of EM applying to Gaussians

Let's go back and revisit our example of fitting a mixture of Gaussians. We now see it as an application to demonstrate the derivation of M-step. The E-step is easy by following the established framework and calculate $w_j^{(i)}=Q_i(z^{(i)}=j)=P(z^{(i)}=j|x^{(i)};\phi,\mu,\Sigma)$. Here, $Q_i(z^{(i)}=j)$ denotes the probability of $z^{(i)}$ taking value of j under distribution Q, and now we view $\{w_j^{(i)}\}_{i=1,j=1}^{m;k}$ as constants.

Next, we seek to maximize the quantity $\sum_{i=1}^{m}\sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma)}{Q_i(z^{(i)})}$ w.r.t ϕ, μ, Σ.

In our particular case setting, the quantity becomes $\sum_{i=1}^{m}\sum_{j=1}^{k} Q_i(z^{(i)}=j) \log \frac{p(x^{(i)}|z^{(i)}=j)\, p(z^{(i)}=j)}{Q_i(z^{(i)}=j)}$

$= \sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} \log \frac{(2\pi)^{-n/2}|\Sigma_j|^{-1/2} \exp\left(-\frac{1}{2}(x^{(i)}-\mu_j)^T \Sigma_j^{-1}(x^{(i)}-\mu_j)\right)\phi_j}{w_j^{(i)}}$. If we take derivative w.r.t. μ_l, we find

$\nabla_{\mu_l} J(\phi, \mu, \Sigma) = -\frac{1}{2}\nabla_{\mu_l} \sum_{i=1}^{m} w_l^{(i)} (x^{(i)}-\mu_l)^T \Sigma_l^{-1}(x^{(i)}-\mu_l) = \frac{1}{2}\sum_{i=1}^{m} w_l^{(i)} \left(\Sigma_l^{-1}x^{(i)} - \Sigma_l^{-1}\mu_l\right) := 0$

Solving this for μ_l therefore yeilds the update rule: $\mu_l := \sum_{i=1}^{m} w_l^{(i)} x^{(i)} \Big/ \sum_{i=1}^{m} w_l^{(i)}$

If we take derivate w.r.t. ϕ_j, to deal with constraint that $\sum_{j=1}^{k}\phi_j = 1$, we construct the Lagrangian:

$$\mathcal{L}(\phi) = \sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} \log \phi_j + \beta\left(\sum_{j=1}^{k}\phi_j - 1\right), \text{ where } \beta \text{ is the Lagrange multiplier.}$$

Notice we didn't explicitly constrain $\phi_j \geq 0$, cause this constraint is automatically satisfied by the solution we are about to find. Moving on, $\frac{\partial}{\partial \phi_j}\mathcal{L}(\phi) = \sum_{i=1}^{m} \frac{w_j^{(i)}}{\phi_j} + \beta := 0$, I.e., $\phi_j \propto \sum_{i=1}^{m} w_j^{(i)}$.

Plugging in the constraint, we obtain $\sum_{j=1}^{k}\sum_{i=1}^{m} \frac{w_j^{(i)}}{-\beta} = 1 \Rightarrow \beta = -\sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} = -m$, thus the M-step update for parameters ϕ_j are $\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)}$.

The derivation for update rule of Σ_j is also straight-forward, if we take the quantity and take derivative w.r.t. Σ_j, we obtain:

$$\nabla_{\Sigma_j}\sum_{i=1}^{m} w_j^{(i)} \log |\Sigma_j|^{-1/2} \exp\left(-\frac{1}{2}(x^{(i)}-\mu_j)^T\Sigma_j^{-1}(x^{(i)}-\mu_j)\right) = \sum_{i=1}^{m} w_j^{(i)}\left[-\frac{1}{2}\Sigma_j^{-1} - \frac{1}{2}\nabla_{\Sigma_j}(x^{(i)}-\mu_j)^T\Sigma_j^{-1}(x^{(i)}-\mu_j)\right]$$

$$= -\frac{1}{2}\sum_{i=1}^{m} w_j^{(i)}\left[\Sigma_j^{-1} - (x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T\Sigma_j^{-1}\cdot\Sigma_j^{-1}\right].$$

By setting the derivative to zero and solving for Σ_j we yield the update rule $\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)}(x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$

Factor Analysis

In our previous mixture of Gaussians setting, we imagine problems where we have sufficient data to be able to discern the multiple-Gaussian structure in the data. For instance this would be the case when we have #samples $m \gg$ dimension of features d.

Now, consider the setting in which $d \gg m$, how the m points span only a low-dimensional subspace of $\mathbb{R}^d$, if we try to model the data with a single Gaussian, then with MLE we would find $\Sigma_{MLE} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{MLE})(x^{(i)}-\mu_{MLE})^T$, which poses the fact that $\text{rank}(\Sigma) \leq \sum_{i=1}^{m} \text{rank}\left[(x^{(i)}-\mu)(x^{(i)}-\mu)^T\right]$ $\leq m \ll d$, meaning that Σ_{MLE} is singular, which poses numerical problems when using multivariate Gaussian distribution, as a singular covariance matrix suggests that Gaussian places all of its probability in the affine space spaned by data.

To deal with the problem mentioned above, we posit a joint distribution on (x,z) as follows:

$z \sim \mathcal{N}(0, I_{k\times k})$, $x|z \sim \mathcal{N}(\mu+\Lambda z, \Psi)$. Parameters: $\mu\in\mathbb{R}^d$, $\Lambda\in\mathbb{R}^{d\times k}$, diagonal matrix $\Psi\in\mathbb{R}^{d\times d}$. with $z\in\mathbb{R}^k$ being latent variables. We can think (and thus choose) k to be the #"driving forces", namely, choose $k(<d)$ to be the number of things that might affect feature x.

The factor analysis model can be interpreted as: each datapoint $x^{(i)}$ is generated by sampling a k-dimension multivarate Gaussian $z^{(i)}$. Then it is* mapped to a d-dimensional affine space of $\mathbb{R}^d$* by performing linear operations $\mu+\Lambda z^{(i)}$ on $z^{(i)}$. Lastly, we* add gaussian noise* $\varepsilon\sim\mathcal{N}(0,\Psi)$ to it to obtain $x^{(i)}=\mu+\Lambda z^{(i)}+\varepsilon$.

It is vital to recall that the basic thought for our model is $p(x,z)=p(x|z)p(z)$ with z being latent variables, so our random variables x,z have a joint Gaussian distribution $\begin{bmatrix} z \\ x \end{bmatrix}\sim\mathcal{N}(\mu_{zx},\Sigma)$.

We observe that $\mathbb{E}[x]=\mathbb{E}[\mu+\Lambda z+\varepsilon]=\mu$, in order to find Σ, we should first note that

$$\mathrm{Cov}\left(\begin{bmatrix} z \\ x \end{bmatrix}\right)=\Sigma\triangleq\begin{bmatrix}\Sigma_{zz} & \Sigma_{zx}\\ \Sigma_{xz} & \Sigma_{xx}\end{bmatrix}=\mathbb{E}\left[\begin{pmatrix} z-\mu_z \\ x-\mu_x\end{pmatrix}\begin{pmatrix} z-\mu_z \\ x-\mu_x\end{pmatrix}^T\right]=\mathbb{E}\begin{bmatrix}(z-\mu_z)(z-\mu_z)^T & (z-\mu_z)(x-\mu_x)^T\\ (x-\mu_x)(z-\mu_z)^T & (x-\mu_x)(x-\mu_x)^T\end{bmatrix}$$

and our model gives $\Sigma_{zz}=\mathbb{E}\left[(z-\mu_z)(z-\mu_z)^T\right]=\mathrm{Cov}(z)=I_{k\times k}$,

(z,ε is independent)

$$\Sigma_{zx}=\mathbb{E}\left[(z-\mu_z)(x-\mu_x)^T\right]=\mathbb{E}\left[z(x^T-\mu^T)\right]=\mathbb{E}[zz^T]\Lambda^T+\mathbb{E}[z\varepsilon^T]=\Lambda^T$$

$$\Sigma_{xx}=\mathbb{E}\left[(x-\mu_x)(x-\mu_x)^T\right]=\mathbb{E}[\Lambda zz^T\Lambda^T+\underbrace{\Lambda z\varepsilon^T+\varepsilon z^T\Lambda^T}_{\text{expectation zero}}+\varepsilon\varepsilon^T]=\Lambda\Lambda^T+\Psi.$$

We now see that the marginal distribution of x is given by $x\sim\mathcal{N}(\mu,\Lambda\Lambda^T+\Psi)$. Thus, given a training set $\{x^{(i)}\}_{i=1}^m$, the log likelihood of parameters is $\ell(\mu,\Lambda,\Psi)=\sum_{i=1}^m\log\left((2\pi)^{-d/2}|\Lambda\Lambda^T+\Psi|^{-1/2}\exp\left(-\frac{1}{2}(x^{(i)}-\mu)^T(\Lambda\Lambda^T+\Psi)^{-1}(x^{(i)}-\mu)\right)\right)$.

Maximizing this quantity w.r.t. parameters explicitly is hard, and we turn to EM algorithm for help.

• EM for Factor Analysis.

First, let's remind ourselves that the conditional of a joint Gaussian is Gaussian, suppose that $\begin{matrix}n\updownarrow\\ n\updownarrow\end{matrix}\begin{bmatrix} x_A \\ x_B\end{bmatrix}\sim\mathcal{N}\left(\begin{bmatrix}\mu_A\\ \mu_B\end{bmatrix},\begin{bmatrix}\Sigma_{AA} & \Sigma_{AB}\\ \Sigma_{BA} & \Sigma_{BB}\end{bmatrix}\right)$, then the conditional densities are also Gaussian:

$x_A|x_B\sim\mathcal{N}(\mu_A+\Sigma_{AB}\Sigma_{BB}^{-1}(x_B-\mu_B),\ \Sigma_{AA}-\Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA})$, $x_B|x_A\sim\mathcal{N}(\mu_B+\Sigma_{BA}\Sigma_{AA}^{-1}(x_A-\mu_A),\ \Sigma_{BB}-\Sigma_{BA}\Sigma_{AA}^{-1}\Sigma_{AB})$.

Using this property we can conclude $z^{(i)}|x^{(i)};\mu,\Lambda,\Psi\sim\mathcal{N}(\mu_{z^{(i)}|x^{(i)}},\Sigma_{z^{(i)}|x^{(i)}})$, with

$$\mu_{z^{(i)}|x^{(i)}}=\Lambda^T(\Lambda\Lambda^T+\Psi)^{-1}(x^{(i)}-\mu),\quad \Sigma_{z^{(i)}|x^{(i)}}=I-\Lambda^T(\Lambda\Lambda^T+\Psi)^{-1}\Lambda.$$

Plug these into $Q_i(z^{(i)})$ for EM we obtain (E-step)

$$Q_i(z^{(i)})=(2\pi)^{-k/2}\left|\Sigma_{z^{(i)}|x^{(i)}}\right|^{-1/2}\exp\left(-\frac{1}{2}\left(z^{(i)}-\mu_{z^{(i)}|x^{(i)}}\right)^T\Sigma_{z^{(i)}|x^{(i)}}^{-1}\left(z^{(i)}-\mu_{z^{(i)}|x^{(i)}}\right)\right).$$

In M-step we need to maximize $\sum_{i=1}^{m}\int_{z^{(i)}} Q_i(z^{(i)})\log\frac{p(x^{(i)},z^{(i)};\mu,\Lambda,\Psi)}{Q_i(z^{(i)})}dz^{(i)} = \sum_{i=1}^{m}\mathbb{E}_{z^{(i)}\sim Q_i}[\log p(x^{(i)}|z^{(i)}) - \log Q_i(z^{(i)}) + \log p(z^{(i)})]$.

Dropping out $Q_i(z^{(i)})$ which is determined in the E-step and $p(z^{(i)})$ which contains no parameter, we find that we need to maximize $\sum_{i=1}^{m}\mathbb{E}[\log p(x^{(i)}|z^{(i)};\mu,\Lambda,\Psi)]$

$$= \sum_{i=1}^{m}\mathbb{E}\left[-\frac{d}{2}\log 2\pi - \frac{1}{2}\log|\Psi| - \frac{1}{2}(x^{(i)}-\mu-\Lambda z^{(i)})^T\Psi^{-1}(x^{(i)}-\mu-\Lambda z^{(i)})\right] \quad (1)$$

Maximize Eq(1) w.r.t. Λ, we obtain:

$$\nabla_\Lambda \sum_{i=1}^{m} -\frac{1}{2}\mathbb{E}\left[(x^{(i)}-\mu-\Lambda z^{(i)})^T\Psi^{-1}(x^{(i)}-\mu-\Lambda z^{(i)})\right]$$

$$= \nabla_\Lambda -\frac{1}{2}\sum_{i=1}^{m}\mathbb{E}\left[z^{(i)T}\Lambda^T\Psi^{-1}\Lambda z^{(i)} + (\mu-x^{(i)})^T\Psi^{-1}\Lambda z^{(i)} + (\Lambda z^{(i)})^T\Psi^{-1}(\mu-x^{(i)})\right]$$

$$= \nabla_\Lambda \sum_{i=1}^{m}\mathbb{E}\left[-\frac{1}{2}\mathrm{tr}(\Lambda^T\Psi^{-1}\Lambda z^{(i)}z^{(i)T}) + \mathrm{tr}(\Lambda^T\Psi^{-1}(\mu-x^{(i)})z^{(i)T})\right]$$

$$= \sum_{i=1}^{m}\mathbb{E}\left[-\Psi^{-1}\Lambda z^{(i)}z^{(i)T} + \Psi^{-1}(x^{(i)}-\mu)z^{(i)T}\right] \overset{\text{set}}{=\!=} 0 \Rightarrow \Lambda = \left(\sum_{i=1}^{m}(x^{(i)}-\mu)\mathbb{E}_{z^{(i)}\sim Q_i}[z^{(i)T}]\right)\left(\sum_{i=1}^{m}\mathbb{E}_{z^{(i)}\sim Q_i}[z^{(i)}z^{(i)T}]\right)^{-1}$$

And with our definition Q_i being Gaussian with mean $\mu_{z^{(i)}|x^{(i)}}$ and covariance $\Sigma_{z^{(i)}|x^{(i)}}$, we find that

$$\mathbb{E}_{z^{(i)}\sim Q_i}[z^{(i)T}] = \mu_{z^{(i)}|x^{(i)}}^T, \quad \mathbb{E}_{z^{(i)}\sim Q_i}[z^{(i)}z^{(i)T}] = \mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}$$

Similarly by taking derivative w.r.t. Ψ and μ we obtain:

$$\mu = \frac{1}{m}\sum_{i=1}^{m}x^{(i)}, \quad \Phi = \frac{1}{m}\sum_{i=1}^{m}x^{(i)}x^{(i)T} - x^{(i)}\mu_{z^{(i)}|x^{(i)}}^T\Lambda^T - \Lambda\mu_{z^{(i)}|x^{(i)}}x^{(i)T} + \Lambda(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}})\Lambda^T$$

Notice μ doesn't change as the parameters are varied, this can be calculated just once, and Ψ is a diagonal matrix as we demand, so Ψ can be found by setting $\Psi_{ii} = \Phi_{ii}$ $(i=1,2,\dots,d)$ (i.e. Ψ constrains only the diagonal entries of Φ. (Q: would this selection affect the optimal solution? or can proof Φ is already diagonal if ignoring precision issues?)

Principal Components Analysis

Like Factor Analysis, PCA view high-dimensional data as lying on low dimension subspace with some noise. Prior to running PCA per se, typically we first preprocess the data $x_j^{(i)} \leftarrow \frac{x_j^{(i)}-\mu_j}{\sigma_j}$ where $\mu_j = \frac{1}{n}\sum_{i=1}^{n}x_j^{(i)}$ and $\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)}-\mu_j)^2$. This normalization rescales the different attributes to make them more comparable and may be omitted if we had a priori knowledge that different attributes are all on the same scale. (e.g. grayscale image).

We propose two ways of deriving the idea of PCA. First, notice our basic view of a low dimension subspace, we want to choose a subspace that is "closest" to all the data points; Another perspective is to choose a unit vector u, and when we project the data onto the direction corresponding to u,

as much as possible of the variance in the data is retained. These two interpretations can be related as projection length and projection height is related by $\|x^{(i)}\|_2^2$, and lead to the optimization problem (here we use the second intuition and maximize the length of projection, which, given unit vector u and point x, is given by $\frac{\langle x,u\rangle}{\|u\|} = x^T u$): $\max_u \frac{1}{n}\sum_{i=1}^{n}((x^{(i)})^T u)^2$ s.t. $\|u\|_2 = 1$.

To tackle with the optimization problem we construct the Lagrangian $\mathcal{L}(u) = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)T}u)^2 + \beta(\|u\|_2 - 1)$
$= \frac{1}{n}\sum_{i=1}^{n} u^T x^{(i)} x^{(i)T} u + \beta(u^T u - 1) = u^T\left(\frac{1}{n}\sum_{i=1}^{n} x^{(i)}x^{(i)T}\right)u + \beta(u^T u - 1) = u^T \Sigma u + \beta(u^T u - 1)$, with Σ being the empirical covariance matrix of the data. (Recall that normalization step ensures zero mean). It is easy to find $\nabla_u \mathcal{L}(u,\beta) = 2(\Sigma u + \beta u)$ and by setting this to zero we see that u should be chosen as eigenvectors of Σ. If we wish to project our data into a k-dimensional subspace ($k<d$), we should choose $u_1, u_2, \cdots u_k$ to be the top k eigenvectors of Σ (↳ comparison of corresponding eigenvalue), and since Σ is symmetric, $u_1, u_2, \cdots u_k$ can be transformed to be orthogonal basis of k-dimension subspace in the original d-dimensional sample space. The original data can now be modelled as $\{x^{(i)}\}_{i=1}^{n} \mapsto (u_1^T x^{(i)}, u_2^T x^{(i)}, \cdots, u_k^T x^{(i)}) \triangleq y^{(i)} \in \mathbb{R}^k$. [dimensionality reduction from $\mathbb{R}^d \to \mathbb{R}^k$.]

If we wish to "uncompress" the data and model $x^{(i)}$ from $y^{(i)}$, we should be aware of the fact that $x^{(i)} \approx y_1^{(i)}u_1 + y_2^{(i)}u_2 + \cdots + y_k^{(i)}u_k$, with approximation coming from the omission of noises.

· Application Analysis

- Scenarios considered to use PCA:

1. Visualization. By reducing data to 2/3 dimension, we can plot the $y^{(i)}$'s to visualize the data, during which process we might be able to observe interesting features such as clusters (indicating types of the features) or trajectory of how features evolve.

2. Compression for Efficiency. It turns out most of high-dimensional data lies on low-dimensional subspace, which is the assumption that led us derive Factor Analysis & PCA (and also a fact of life). Apart from improving computational efficiency, reducing data's dimension can also reduce the *complexity of hypothesis class* considered (Lower dimensional input probably have lower VC dimension).

- Questionable scenarios and ideas:

1. Reduce Overfitting. This idea comes from lower feature numbers leads to lower variance, but this technique works as often as it fails, consider using regularization instead.

2. Outlier Detection. In early stages of face recognition, reducing original image data $x^{(i)} \in \mathbb{R}^{100\times100}$ to lower dimensions (say 2) and plotting the processed data results in eigenfaces, and measuring $\|y^{(i)} - y^{(j)}\|_2^2$ results in a surprisingly good face-matching algorithm. In eigenfaces, it seems the principal components can be interpreted (e.g. glasses, mustache) and PCA helps reduce noise on feature vectors, but individual eigenvector is quite noisy so avoid examining single eigenvector, in which case interpretation lacks theoratical proof and is a product of reaserchers' imagination.

↳ [positions (numerical) of individual eigenvector is unstable to perturbations in data]

- General Rule of Thumb: For implementations using PCA, consider how the algorithm does on original data to avoid adding PCA unnecessarily.

[But subspace spanned by these eigenvectors is quite stable]

A brief conclusion of our tour in unsupervised learning algorithms:

	Model $p(x)$ (e.g. For anomally detection)	Non-probalistic (e.g. compression, visualization)
Subspaces	Factor Analysis	PCA
Clusters	Mixture of Gaussians (not necessarily Clusters)	k-means

Apply EM since EM models $p(x)$ given $p(x, z)$ (latent variables)

Independent Components Analysis

The motivation for ICA is to extract independent sounds from their combinations, say $s^{(i)} \in \mathbb{R}^d$ with $s_j^{(i)}$ being j^{th} speaker's voice at time i, what we observe is $x^{(i)} = As^{(i)}$ with $x_j^{(i)}$ being observation at j^{th} microphone, (for now we assume # speakers = # microphone) and A an unknown matrix $\in \mathbb{R}^{d\times d}$. Repeated observations gives a dataset $\{x^{(i)}\}_{i=1}^n$ and our goal is to recover the sources $s^{(i)}$ that had generated our data $x^{(i)}$. We will denote $W = \begin{bmatrix} -w_1^T- \\ \vdots \\ -w_d^T- \end{bmatrix} = A^{-1}$, then j^{th} source can be recovered as $s_j^{(i)} = w_j^T x^{(i)}$.

- ICA Ambiguities

Under what circumstances can $W = A^{-1}$ be recovered? There are some inherent ambiguities in A that are impossible to recover, given only the $x^{(i)}$'s.

First, it's trivial to see that there is no way to recover the correct scaling of the c_i's. If a

single column of A were scaled by α, and the corresponding source was scaled by $1/\alpha$, then there is no way to determine that this has happened given only the $x^{(i)}$'s. Notice that the scaling factor controls the volume of that speaker's voice, this ambiguity would not matter since we are only trying to identify the independent speakers. (As is the case for most ICA applications).

Also notice given only the $x^{(i)}$'s, it is impossible to distinguish between W and PW, with P being any permutation matrix $\in \mathbb{R}^{d\times d}$. Intuitively, there are no "absolute" labeling for the speakers.

It is vital to point out the ambiguity with sources being generated from Gaussian. For illustration, let's say $s\sim\mathcal{N}(0, I_{d\times d})$, so the contours are circles ($d=2$ case) centered on the origin. Now, suppose we observe some $x=As\sim\mathcal{N}(0, AA^T)$, and let R be an arbitrary orthogonal matrix, and let $A'=AR$. The distribution of $x'=A's$ is still $\mathcal{N}(0, AA^T)$. Thus, there is no way to tell if sources were mixed using A or A', this is because the density of Gaussian $(\mathcal{N}(0,I))$ is rotationally symmetric. However, for natural voices, the data turns out not being Gaussian, with high/low pitches frequently observed but would be treated as outliers in Gaussian.

· Algorithm Derivation

We suppose that the distribution of each source s_j is given by density p_s, now given s_j are iid we obtain $p(s)=\prod_{j=1}^{d} p_s(s_j) \Rightarrow \prod_{j=1}^{d} p_s(w_j^T x)\,|A^{-1}| = p(x)$. [recall that $x=As=W^{-1}s$ gives $p(x)=p_s(Wx)|W|$].

A "default" choice of density p is $p_s(s)=\sigma'(s)$ with σ being sigmoid function, which, when chosen as CDF, gives a PDF that has "fatter" tails than the Gaussian.

[Note:] Default choice implies that we had no prior knowledge of the sources' densities, if indeed we would have, then that density would be a good substitute of default. The Laplace distribution $p(x)=\frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right)$ is also a reasonable choice.

The representation here assumes that either the data $x^{(i)}$ has been preprocessed to have zero mean, or that it can naturally be expected to have zero mean (such as acoustic signals), cause under the condition $\sigma'(s)$ is symmetric, $\mathbb{E}[s]=0 \Rightarrow \mathbb{E}(x)=\mathbb{E}[As]=0$

Moving on to MLE, the log likelihood is given by

$$\ell(W)=\sum_{i=1}^{n}\left[\sum_{j=1}^{d}\log p_s(w_j^T x^{(i)})+\log|W|\right]=\sum_{i=1}^{n}\left[\sum_{j=1}^{d}\log\sigma'(w_j^T x^{(i)})+\log|W|\right]$$

Taking derivative of W gives us the stochastic gradient ascent update:

$$W := W + \alpha \nabla_W \ell(W) = W + \alpha \left(\begin{bmatrix} 1 - 2\sigma(w_1^T x^{(i)}) \\ \vdots \\ 1 - 2\sigma(w_n^T x^{(i)}) \end{bmatrix} x^{(i)T} + (W^{-1})^T \right).$$

where α is the learning rate. After the algorithm converges, we then compute $s^{(i)} = Wx^{(i)}$ to recover sources. When deriving likelihood of data, we implicitly assumed that $x^{(i)}$'s were independent of each other, which is clearly incorrect for speech and other time series data. But it can be shown that having correlated training examples will not hurt performance of algorithm if we have sufficient data. However, for problems where successive training examples are correlated, it sometimes helps SGD to converge if we run stochastic gradient descent on a randomly shuffled copy of training set.

A Brief mentioning of #speakers ≠ # microphones scenarios, first note if #speakers < #microphones, then we can assume there are silent speakers, but #speakers > #microphones is still a cutting edge research area, with some findings suggest it's possible to distinguish speakers if pitches of voice differ to some extent.

Boosting

Roughly, the idea of boosting is to take a weak learning algorithm (one that is slightly better than random guessing) and transform it into a strong classifier. Boosting procedures proceed by taking a collection of weak classifiers, and then reweighting their contributions to form a classifier with much better accuracy than any individual classifier.

Let's formulate the problem. Assume we have raw inputs $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$, $x^{(i)} \in \mathbb{R}^n$, $y^{(i)} = \{-1, +1\}$. as usual in a binary classification setting. We also assume we have an infinite collection of feature functions $\phi_j : \mathbb{R}^n \to \{+1, -1\}$ and an infinite weight vector $\theta = (\theta_1, \theta_2, \cdots)^T$ with finite non-zero entries. Our hypothesis $h_\theta(x) = \text{sign}\left(\sum_{j=1}^{\infty} \theta_j \phi_j(x)\right) \triangleq \text{sign}(\theta^T \phi(x))$. During process of boosting, we start by assigning equal weight to each training example and continously perform reweighting where examples that it misclassifies receive higher weight and examples it correctly classified receive lower weight. We now formalize the concept of "weight" to be a distribution on the examples where $\sum_{i=1}^m p^{(i)} = 1$ and $\forall i, p^{(i)} \geq 0$. We say that there is a weak learner with margin $\gamma > 0$ if for any distribution $p = (p^{(1)}, p^{(2)}, \cdots p^{(m)})$ on the m training examples there exists some weak hypothesis ϕ_j s.t.

$$\sum_{i=1}^m p^{(i)} \mathbb{1}\{y^{(i)} \neq \phi_j(x^{(i)})\} \leq \frac{1}{2} - \gamma. \quad (1)$$

That is, we assume there is some classifier that does slightly better than random guessing.

Now let's focus on deriving a loss function, notice:

$$1\{\text{sign}(\theta^T\phi(x)) \neq y\} = 1\{y\theta^T\phi(x) \leq 0\} \leq \exp(-y\theta^T\phi(x))$$

Thus our misclassification error $\frac{1}{m}\sum_{i=1}^{m} 1\{\text{sign}(\theta^T\phi(x^{(i)})) \neq y^{(i)}\} \leq \frac{1}{m}\sum_{i=1}^{m}\exp(-y^{(i)}\theta^T\phi(x^{(i)}))$, which leads to assigning the loss function as $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\exp(-y^{(i)}\theta^T\phi(x^{(i)}))$. Notice the convenience of "argmin" operation on exponential loss, we perform coordinate descent update for risk $J(\theta)$. Suppose we wish to update coordinate k, define $w^{(i)} = \exp(-y^{(i)}\sum_{j\neq k}\theta_j\phi_j(x^{(i)}))$, and $W^+ := \sum_{i:\, y^{(i)}\phi_k(x^{(i)})=1} w^{(i)}$, $W^- := \sum_{i:\, y^{(i)}\phi_k(x^{(i)})=-1} w^{(i)}$. Thus:

$$\arg\min_{\theta_k} J(\theta) = \arg\min_{\theta_k}\{W^+e^{+\alpha} + W^-e^{+\alpha}\} = \frac{1}{2}\log W^+/W^-.$$

- Algorithm Derivation

We assume that we have access to a weak-learning algorithm which at iteration t takes as an input a distribution p on the training set and returns a hypothesis ϕ_t satisfying margin condition (1), then the set of weak hypotheses returned by the weak learning algorithm can be denoted as $\{\phi_1, \phi_2, \cdots \phi_t\}$, and the full boosting algorithm can be described as:

For each iteration $t = 1, 2, \cdots$: {

(1) Define weights $w^{(i)} := \exp(-y^{(i)}\sum_{\tau=1}^{t-1}\theta_\tau\phi_\tau(x^{(i)}))$. // current weak hypothesis: $\{\phi_1, \phi_2, \cdots \phi_{t-1}\}$.

and distribution $p^{(i)} := w^{(i)}/\sum_{j=1}^{m} w^{(j)}$. // idea of assigning bigger weights to misclassified examples

(2) Construct a weak hypothesis $\phi_t : \mathbb{R}^n \to \{-1, +1\}$ from the distribution $p = (p^{(1)}, p^{(2)}, \cdots p^{(m)})$ on the training set [output from (1)]

(3) Compute $W_t^+ := \sum_{i:\, y^{(i)}\phi_t(x^{(i)})=1} w^{(i)}$ and $W_t^- := \sum_{i:\, y^{(i)}\phi_t(x^{(i)})=-1} w^{(i)}$

and set $\theta_t := \frac{1}{2}\log W_t^+/W_t^-$. }.

Recall we construct our classifier as $h_\theta(x) = \text{sign}(\sum_{j=1}^{\infty}\theta_j\phi_j(x))$ and said θ only have finite non-zero entries. We now argue that the boosting procedure * achieves 0 training error * and provide a * rate of convergence to zero *.

- Convergence of Boosting

Lemma. Let

$$J(\theta^{(t)}) = \frac{1}{m}\sum_{i=1}^{m}\exp\left(-y^{(i)}\sum_{\tau=1}^{t}\theta_\tau\phi_\tau(x^{(i)})\right).$$

Then $J(\theta^{(t)}) \le \sqrt{1-4\gamma^2}\,J(\theta^{(t-1)})$, with γ being margin specified by condition (i).

Proof. We inspect that $J(\theta^{(t)}) = \min_\alpha\{W_t^+e^{-\alpha} + W_t^-e^{+\alpha}\} = 2\sqrt{W_t^+W_t^-}$.

While $J(\theta^{(t-1)}) = \frac{1}{m}\sum_{i=1}^{m}\exp\left(-y^{(i)}\sum_{\tau=1}^{t-1}\theta_\tau\phi_\tau(x^{(i)})\right) = W_t^+ + W_t^-$.

We know from condition (i) that $\sum_{i=1}^{m}p^{(i)}1\{y^{(i)}\neq\phi_t(x^{(i)})\} \le \frac{1}{2}-\gamma$ $\xrightarrow[\text{of } p^{(i)}]{\text{algorithm specification}}$ $\frac{W_t^-}{W_t^+ + W_t^-} \le \frac{1}{2}-\gamma$.

Rewriting this expression we get $W_t^- \le (\frac{1}{2}-\gamma)(W_t^+ + W_t^-)$ or $W_t^+ \ge \frac{1+2\gamma}{1-2\gamma}W_t^-$.

which indicates α (in the minimum defining $J(\theta^{(t)})$) $= \frac{1}{2}\log W^+/W^- \ge \frac{1}{2}\log\frac{1+2\gamma}{1-2\gamma}$.

Leading to $J(\theta^{(t)}) \le W_t^-\sqrt{\frac{1+2\gamma}{1-2\gamma}} + W_t^+\sqrt{\frac{1-2\gamma}{1+2\gamma}}$

$$= W_t^+\sqrt{\frac{1-2\gamma}{1+2\gamma}} + (1-2\gamma)W_t^-\sqrt{\frac{1+2\gamma}{1-2\gamma}} + 2\gamma\sqrt{\frac{1+2\gamma}{1-2\gamma}}W_t^-$$

$$\le W_t^+\left(\sqrt{\frac{1-2\gamma}{1+2\gamma}} + 2\gamma\sqrt{\frac{1-2\gamma}{1+2\gamma}}\right) + (1-2\gamma)\sqrt{\frac{1+2\gamma}{1-2\gamma}}W_t^-.$$

$$= (1-4\gamma^2)(W_t^+ + W_t^-) = (1-4\gamma^2)J(\theta^{(t-1)}).$$ □

We initialize the training procedure at $\theta^{(0)} = \vec{0}$ making the initial empirical risk 1. Notice

$$\frac{1}{m}\sum_{i=1}^{m}1\{\text{sign}(\theta^T\phi(x^{(i)})) \neq y^{(i)}\} \le J(\theta)$$

Thus if $^*J(\theta) < \frac{1}{m}$ then θ makes no mistakes on the training data*. After t iterations of boosting the empirical risk satisfies $J(\theta^{(t)}) \le (1-4\gamma^2)^{t/2}J(\theta^{(0)}) = (1-4\gamma^2)^{t/2}$. We demand $J(\theta^{(t)}) < \frac{1}{m}$ to find the number of iteration needed: $t > \frac{2\log m}{-\log(1-4\gamma^2)}$. Notice RHS $\le \frac{\log m}{2\gamma^2}$, thus with more than $\frac{\log m}{2\gamma^2}$ rounds of boosting we can come up with a classifier of zero training loss. Although it is also important for the model to have low generalization error, we will ignore test part and focus on training error in this chapter.

• Implementing Weak-learners

There are a number of strategies for weak learners and in this section we focus on one, known as decision stumps. For concreteness of discussion, let's suppose input variables $x \in \mathbb{R}^n$ are real-valued. A decision stump is parameterized by a threshold s and index j, namely select j^{th} entry of x and compare x_j with s, and returns $\phi_{j,s}(x) = \text{sign}(x_j - s) = \begin{cases} +1, & \text{if } x_j \ge s \\ -1, & \text{otherwise.} \end{cases}$

Recall our goal here is, given a distribution $(p^{(1)}, p^{(2)}, \cdots p^{(m)})$ on the training set, we wish to choose a decision stump of the above form to minimize error on training set, that is, find a threshold $s \in \mathbb{R}$ and feature index j s.t.

$\widehat{Err}(\phi_{j,s}, P) = \sum_{i=1}^{m} p^{(i)} 1\{\phi_{j,s}(x^{(i)}) \neq y^{(i)}\} = \sum_{i=1}^{m} p^{(i)} 1\{y^{(i)} sign(x_j^{(i)} - s) = -1\} = \sum_{i=1}^{m} p^{(i)} 1\{y^{(i)}(x_j^{(i)} - s) \leq 0\}$.

is minimized. Notice the* only values s for which training error can change are the values $x_j^{(i)}$*. if for each feature $j = 1, 2, \cdots n$, we sort the raw input features s.t. $x_j^{(i_1)} \geq x_j^{(i_2)} \geq \cdots \geq x_j^{(i_m)}$, then calculation of the above training error can be rewritten as $\sum_{k=1}^{m} p^{(i_k)} 1\{y^{(i_k)}(x_j^{(i_k)} - s) \leq 0\}$. Choose for s, iterating over # training samples and # features let us choose a index j and threshold s that gives the best decision stump. One important thing to note is that $Err(\phi_{j,s}, P) = 1 - Err(-\phi_{j,s}, P)$, thus it is important to also* track the smallest value of $1 - \widehat{Err}(\phi_{j,s}, P)$ over all thresholds*, because it may be smaller than $\widehat{Err}(\phi_{j,s}, P)$ which gives a better weak learner.

Now we discuss about variations on the basic boosted decision stumps. First, we do not need input features x_j be real-valued, let's say some x_j might be categorical with $x_j \in \{1, 2, \cdots K\}$, in which case decision stumps can take the form $\phi_j(x) = \begin{cases} 1, & \text{if } x_j = 1 \\ -1, & \text{otherwise} \end{cases}$ as well as variants setting $\phi_j(x) = 1$ if $x_j \in C$ for some set $C \subset \{1, 2, \cdots K\}$. Another natural variation is the boosted decision tree, in which instead of a single level decision-making, we consider conjuctions of trees of decisions

Reinforcement Learning

Note for many sequential decision making and control problems, there is an absence of (or difficult to come up with) explicit supervision (eg. labeling). In the reinforcement learning framework, we will instead provide our algorithm only a reward function, which would be an idicator for whether the algorithm is doing well, and it will be the algorithm's job to choose actions over time so as to obtain large rewards.

• Markov Decision Process

MDP formalizes our reinforcement learning setting by proposing a tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where:

* S: set of states (eg. in a chess game setting, S will be all possible chess positions)
* A: set of actions (e.g. the set of all possible moves we can adapt)
* P_{sa}: state transition probabilities. For each state $s \in S$ and $a \in A$, P_{sa} is a distribution over the state space that we can transit to, for all states s' we can arrive at; $\sum_{s'} P_{sa}(s') = 1$.
* γ: discount factor in $[0, 1)$.

* R: reward function mapping $S \times A$ to $\mathbb{R}$, if written as a function of state S only, $R: S \mapsto \mathbb{R}$.

The dynamics of an MDP proceeds as follows: Start in some state s_0 and get to choose a move $a_0 \in A$, as a result of our choice, the state of MDP randomly transitions to some successor state s_1, drawn according to $P_{s_0 a_0}$. pictorially, $s_0 \xrightarrow{a_0} s_1 \sim P_{s_0 a_0} \xrightarrow{a_1} s_2 \sim P_{s_1 a_1} \xrightarrow{a_2} s_3 \sim P_{s_2 a_2} \to \cdots$

Upon visiting the sequence of states $s_0, s_1, \cdots$ with actions $a_0, a_1, \cdots$, our total payoff is given by

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$$

Let's interpret the existence of discount factor, intuitively, with γ^t monotonically decreasing, we are encouraging the algorithm to get the reward (positive) as quickly as possible, pragmatically, since we are defining payoff in terms of series (which is the case when there is no absorbing state), by adding γ^t, with Dirichlet test of convergence, our series will surely converge. (and postpone negative reward as long as possible)

[One finicial interpretation is seeing γ as the value of money, which depreciate over time] (interest rate)

Our goal in reinforcement learning is to choose actions over time so as to maximize the expected value of the total payoff $\mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$.

We now introduce policy (controller) as a function $\pi: S \to A$, executing policy π indicates whenever we are in state s, we take action $a = \pi(s)$. Define the value function for a certain policy π:

$$V^\pi(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid s_0 = s, \pi]$$

(though not a random variable, we will notate as condition on π.)

which is simply the expected payoff upon starting in state s and act according to π.

Note that $V^\pi(s) = \mathbb{E}[R(s_0) + \gamma V^\pi(s_1)]$ and $s_1 \sim P_{s_0 \pi(s_0)}$ which gives rise to Bellman equation:

$$V^\pi(s) = \underbrace{R(s_0)}_{\text{immediate reward}} + \gamma \underbrace{\sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s')}_{\text{expected sum of discounted rewards after first step}} \quad (1)$$

With Bellman equation, given a specific policy π in a finite-state MDP ($|S| < \infty$), we can write down one such equation for each $s \in S$. This gives us a set of $|S|$ linear equations in $|S|$ variables, which can be efficiently solved.

We naturally ask what is the best possible value of expected sum for any policy? Leading to defining

$$V^*(s) = \max_\pi V^\pi(s).$$

and a natural variation of Bellman's equations

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V^*(s') \quad (2)$$

(V^*: optimal value function in the following process; a: optimal action current step)

We also define a policy $\pi^*: S \to A$ as follows:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \quad (3)$$

which is the action that attains the maximum in the "max" in Eq(2), with extrapolation to S, π^* is the optimal policy. Note that π^* has the property of being optimal policy for all states $s \in S$, this means we can use the same policy π^* regardless of intial state of our MDP.

• Solving finite-state MDPs

With the framework proposed in the previous section, our roadmap is now (1) find algorithm to compute $V^*(s) = V^{\pi^*}(s)$, (2) use Eq(3) to find the optimal policy. For now, we will consider only MDPs with $|S| < \infty$ and $|A| < \infty$. In this section we will also assume that the state transition probabilities $\{P_{sa}\}$ and reward function R is known.

* Algorithm: Value Iteration

1. For each state s, initialize $V(s) := 0$ // then $[V(s_1), V(s_2), \cdots V(s_n)]^T \in \mathbb{R}^{|S|}$ with all entries being 0
2. Repeat until convergence. {

   For every state s, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$ }.

There are two possible ways of performing updates in the inner loop,

i) Synchronous update. Here RHS is always going to be old estimation value, and we are computing RHS for $|S|$ times and then synchronously update all entries of $[V(s_1), V(s_2), \cdots V(s_n)]^T$.

We can proof, value iteration causes $[V(s_1), V(s_2), \cdots V(s_n)]^T$ to converge to V^* in exponential speed.

ii) Asynchronous update. Here we would loop over the vector in some order and update the value one at a time, so later computed $V(s_k)$ will be using / depending on refreshed value of $V(s_{k-1})$ (if s_k is the successive state of $s_{(k-1)}$).

* Algorithm: Policy Iteration

1. Initialize π randomly. // for each of the states, pick a random action
2. Repeat until convergence. { // actually policy iteration gets to exactly V^* (like normal equations in LR)

   a) Solve the value function V^π using the set of Bellman's equations, and set $V := V^\pi$.

   b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s') V(s')$ }. // pretend $V_{current}$ is optimal $\Rightarrow$ greedy algorithm.

Notice step 2.(a) in policy iteration is done at the expense of inverting a matrix, so for small problems (i.e. problems with small $|S|$)

policy iteration is a favorable choice, when in large problems, prioritize using value iteration.

After solving for V^*, if using policy iteration, then optimal policy is attained when algorithm converges, but if using value iteration, there is one more step to use Eq(3) and come up with π^*.

Practically, when we have absorbing states (e.g. winning/lossing, target) in our problem setting, we can model these states by assigning their transition probabilities to zero, therefore when using value/policy iteration, $V(\text{absorbing state}) = R(\text{absorbing state})$ which is the initial immediate reward, and won't alter as algorithm runs. For states that are not absorbing state, we would often assign a small negative penalty to each of these states (e.g. -0.02) to encourage algorithm to get the final reward as quickly as possible.

· Learning a Model for MDP

So far we have discussed MDPs assuming that both state transition probability and reward function are known, but in realistic problems, these might not be prior knowledge and have to be estimated from empirical data. [S, A, γ are usually known, and reward can be assigned]

In order to do so, let's say our MDP consists of a number of trials, we can then perform MLE to get state transition probabilities:

$$P_{sa}(s') = \frac{\#\text{ times we took action } a \text{ in state } s \text{ and got to } s'}{\#\text{ times we took action } a \text{ in state } s.}$$

(denominator)

Or, if numerator is 0 — corresponding to the case of never taking action a in state s, then we might simply estimate $P_{sa}(s')$ to be $1/|S|$. It turns out we can apply Laplace smoothing here, like we did in naive Bayes, but we don't have to, since these solvers for MDP is not sensitive to zero values, unlike naive Bayes where having zero probability will be really problematic.

Using a similar procedure, if R is unknown, we can pick our estimate of the expected immediate reward $R(s)$ to be the average reward observed in state s.

Putting it together, we propose one possible algorithm for learning in an MDP with unknown state transition probabilities.

* Algorithm: 1. Initialize π randomly. // generate a random policy

2. Take actions w.r.t. π for some number of trials to get experience in MDP.

Repeat { (a) Execute π for certain time.

(b) Using accumulated experience, update estimates for P_{sa}

[recall P_{sa} is estimated to be fraction of counts]

(c) Apply value/policy iteration with estimated P_{sa} to get a new estimated value function V.

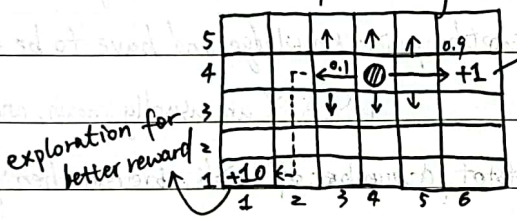
(d). Update π to be the greedy policy w.r.t. V, namely, $\pi(s) := \arg\max_a \sum_{s'} P_{sa}(s')V(s')$. }

One simple optimization is where we apply value iteration, instead of initializing with $V_1 = \vec{0}$, we can initialize it with the solution found during the previous iteration of our algorithm. This simple technique will help value iteration converge more quickly by providing a much better initial starting point.

In some problem settings, the reward might be a random function of the state (e.g. stock prize, self driving), and as mentioned before, this process can also helps us estimate the immediate reward $R(s)$.

• Exploration-Exploitation Tradeoff

The exploration vs. exploitation problem poses a question of how greedy should we be at just taking actions to maximize rewards, one simple setting for us to observe the problem would be:



[+10 reward wasn't there for advance, need to figure out].

Say the initial position of our robot is (4,4) and the initial random policy leads our robot to find reward +1 at (6,4), and this would lead our algorithm to always go to (6,4), which, formally put, is locally greedy choices lead to local optima instead of global optima. Also, when heading to reward at (6,4), our robot may wander off a bit and gets more information about right plane, and leave the left plane mostly unexplored.

Let's revise our algorithm, what we suggested is taking policies that is expected to maximize the total payoff, which is indeed locally greedy. What our algorithm did not do is exploration, which is, taking actions that seem less optimal currently (e.g. in our example, it would be going left multiple times just to see what happens) to gain more knowledge about the task.

It turns out the exploration vs. exploitation problem is not only academic, but occurs in practical settings. A typical example would be online ads company's strategy to pose ads. By posing ads the user is most likely to click on would drive short time interest, but might provide less knowledge about the user, If, say we are showing ads that, by our current knowledge of user interest, is not the ads that the user appears to be most interested in, but ads from other categories. By doing so, we might gain more

information about user interest and thus increase the effectiveness of these companies finding relevant ads.

One strategy to tackle this issue is a exploration strategy called ε-greedy, which instead of taking actions w.r.t. π, we now hold $(1-\varepsilon)$ chance of taking actions (greedy action) w.r.t. π and ε chance of taking actions (random action) randomly. With this strategy, every now and then our robot might get to (1,1) and end up exploring the state space more thoroughly. It turns out, with ε-greedy exploration and sufficient amount of time, our algorithm will converge to the optimal policy (global optimum) for any discrete state MDP. A common heuristic for choosing ϵ is, start with a relatively large ϵ and gradually decrease it.

Another common exploration strategy is Boltzmann exploration, where we would choose probabilistically according to expected rewards. for example we would choose state s with probability $\exp(-P(s)/T)$ where $P(\cdot)$ is the payoff probability, and would decrease T (temperature) over time, with high T alluding exploration and low T leading to increasing bias toward best action.

· Continuous State MDPs

For MDPs that have an infinite number of states, for example, when modeling a helicopter flying in 3d space, the states take the form of $(\underbrace{x, y, z}_{\text{3d coordinates}}, \underbrace{\phi, \theta, \psi}_{\text{3d orientation (roll, pitch, yaw)}}, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$, Another example would be the inverted pendulum, where states take the form of $(\underbrace{x, \theta}_{\text{position}}, \underbrace{\dot{x}, \dot{\theta}}_{\text{angle of pole}})$, generally, in problems where $S=\mathbb{R}^d$, some of our former algorithm don't work well, or won't work at all. In this section we will focus on solving continuous state MDPs.

Strategy 1. Discretization

This is perhaps the simplest thought, and usually works well for 1d and 2d problems (with the advantage of being simple and quick to implement) even without cleverly choosing the way to discretize. For example in 2d problem, we can use a grid to discretize the state space, and approximate the continuous-state MDP via a discrete-state one $(\bar{S}, A, \{P_{sa}\}, \gamma, R)$.

There are two main downsides of this strategy. First, it naively assumes the value function takes a constant value over each discretization intervals. To better understand the limitation, consider fitting a piecewise constant function to a linear model, this "stair" function just isn't a good representation for many smooth functions. we might need really small intervals to get a good representation.

The second downside is called the curse of dimensionality. Suppose $S=\mathbb{R}^d$ and each dimension is

discretized into k values, then we have a total of k^d discrete states, and grows exponentially as problem dimension increases.

As a rule of thumb, discretization works well for 1d and 2d problems. With problems of 3d and 4d, we may have to decide which dimension affects the outcome the most, and discretize this influential state to more discrete states, and discretization might work in this scenario. But discretization very rarely works for problems with $\geq 6d$.

Strategy 2. Value Function Approximation

An alternative method would be to approximate V^* directly. Like what we did in (linear) regression models, but here we will add some modifications and model $V^*(s) \approx \theta^T \phi(s)$, where $\phi(s)$ is the features of s.

* Using a model or simulator

Informally, a simulator takes state s_t (continuous-valued) and action a_t as input, and outputs next-state s_{t+1} sampled according to $P_{s_t a_t}$. (assume to be discretized) One way to derive such a model is using a physics simulation. For example the inverted pendulum problem setting obeys the general laws of physics, thus by using physics knowledge one can derive the position and orientation the cart/pole will be in at time $t+1$ given state and action at time t.

An alternative approach would be to derive model from data. Suppose we execute n trials with each trial consisting T timesteps, we would then observe n state sequences:

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \rightarrow \cdots \xrightarrow{a_{T-1}^{(1)}} s_T^{(1)}$$

$$\vdots$$

$$s_0^{(n)} \xrightarrow{a_0^{(n)}} s_1^{(n)} \rightarrow \cdots \rightarrow s_{T-1}^{(n)} \xrightarrow{a_{T-1}^{(n)}} s_T^{(n)}$$

We can then apply supervised learning algorithm to predict s_{t+1} as a function of s_t and a_t. For example in our helicopter problem setting, when helicopter is generally flying low-speed and experience no sharp turns, a linear model would work well, say we choose to fit

$$s_{t+1} = As_t + Ba_t, \text{ where } A \in \mathbb{R}^{d \times d}, B \in \mathbb{R}^{d \times d_{action}}$$

and farther performing maximum likelihood estimate leads us to pick A, B by

$$\arg\min_{A,B} \sum_{t=0}^{T-1} \sum_{i=1}^{n} \left\| s_{t+1}^{(i)} - (As_t^{(i)} + Ba_t^{(i)}) \right\|_2^2$$

Depending on our choice of features, modeling $s_{t+1} = A\phi(s_t) + B\phi'(a_t)$ can work better than our original linear model.

Having learned A, B, one option is to build a deterministic model, in which given input s_t and a_t, output s_{t+1} is exactly determined. Deterministic model works well for simulators, but once taken to the real-world, would usually work poorly on physical (robots, helicopters, etc). Alternatively we may also build a stochastic model, in which $s_{t+1} = As_t + Ba_t + \epsilon_t$, and noise term ϵ_t is usually modeled as $\epsilon_t \sim \mathcal{N}(0, \Sigma)$, where covariance matrix Σ can be estimated from data. The exact choice of noise distribution matters less than the fact that we remember to add some noise.

* Fitted Value Iteration

Recall our goal here is to approximate $V(s)$ as a linear or non-linear function of s, here we will use a supervised learning algorithm (linear regression in this case) and fit $V(s) = \theta^T \phi(s)$. Since value function on LHS suggest how well our model would be doing if start at state s (expected total payoff), it would be reasonable to choose features ($\phi(s)$) that help convey/evaluate the performance of model. For example in inverted pendulum problem, if the pole is leaning towards right, then we would want the car below to have larger velocity towards right, which suggest our feature function to contain $\theta\dot{x}$ as a entry, thus a reasonable feature function (mapping) in this problem setting would be $\phi(s) = \begin{bmatrix} x^2 \\ \dot{x} \\ x \cdot \dot{x} \\ \theta \cdot \dot{x} \end{bmatrix}$

One potentially nice thing about model based RL is we can generate tons of data by using our model, thus we can afford to choose a lot of features (even inelevant ones) without worrying about overfitting.

We will assume the problem has a continuous state space $s \in \mathbb{R}^d$, but that the action space is small and discrete, which is the case for most MDPs. Recall that in value iteration:

$$V(s) := R(s) + \gamma \max_a \int_{s'} P_{sa}(s') V(s') ds' = \max_a \mathbb{E}_{s' \sim P_{sa}} [R(s) + \gamma V(s')]$$

* Algorithm: Fitted Value Iteration

1. Sample $\{s^{(1)}, s^{(2)}, \cdots s^{(n)}\} \subseteq S$ randomly
2. Initialize $\theta := 0$. // Hence by $V(s) = \theta^T \phi(s)$, V would start with all entries being 0.
3. Repeat {
   - start for i: For $i = 1, 2, \cdots n$ { // having generated training sample $s^{(i)}$ (x-axis), what to estimate $V(s^{(i)})$ (y-axis)
     - start for action: For each action $a \in A$ {
       - Sample $s'_1, s'_2, \cdots s'_K \sim P_{s^{(i)}a}$ (using model of MDP) — this corresponds to using a stochastic model, if deterministic, set $K := 1$.
       - Set $q(a) := \mathbb{E}_{s' \sim P_{s^{(i)}a}} [R(s^{(i)}) + \gamma V(s')] = \frac{1}{K} \sum_{j=1}^{K} [R(s^{(i)}) + \gamma V(s'_j)]$ $\quad j = 1, 2, \cdots K$
     - end for action }.

Set $y^{(i)} = \max_a q(a)$. // Hence $y^{(i)}$ is an estimate of $R(s^{(i)}) + \gamma \max_a \mathbb{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.

endfor i } // Now $y^{(i)}$ is an estimate of $V(s^{(i)})$ which we want to model as $\theta^T \phi(s^{(i)})$

Set $\theta := \arg\min_\theta \frac{1}{2} \sum_{i=1}^{n} (\theta^T \phi(s^{(i)}) - y^{(i)})^2$. // minimizing loss function of linear regression

(end)

↑ can plug in other supervised learning alg as well. (or even neural network)

Unlike value iteration over discrete states, fitted value iteration cannot be proved to always converge. However in practice it often does converge (or approximately converge).

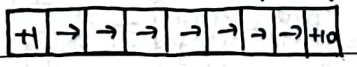
Finally, fitted value iteration outputs V which is an approximation of V^*, this implicitly defines our policy. When in state s, we would like to act according to $\pi^*(s) = \arg\max_a \mathbb{E}_{s' \sim P_{sa}}[V^*(s')]$, which in our approximation is $\arg\max_a \mathbb{E}_{s' \sim P_{sa}}[V(s')]$. Recall this is the inner loop of fitted value iteration. but for systems that need frequent update, the reiterate sampling process can be computationally expensive. Say our model is $S_{t+1} = AS_t + Ba_t + \epsilon_t$, during training time, adding noise is vital for robustness of policy, but when deploying in a physical simulator, one reasonable way is to set $\epsilon_t i = 0$ and correspondingly $K1 = 1$, which lead to optimal policy being $\arg\max_a V(f(s,a))$ $[S_{t+1} = f(s_t, a_t) + \epsilon_t]$

Conclusively, use a stochastic model during training phase and deterministic model during physical deployment.

- Finite Horizon MDP

We previously discussed the case where $\pi^*(s)$ is a stationary policy, that is, no matter the time stamp, our policy remains the same. In this section we put forward the case where $\pi^*(s)$ is a non-stationary policy, which, denotes as $\pi_t^*(s)$, varies over time.

We now assign a time duration for our MDP, and MDP becomes $(S, A, \{P_{sa}\}, T, R)$ with T being horizon time. The discount factor is not a must in this setting since our expected total payoff will always be a finite sum: $\mathbb{E}[R(s_0,a_0) + R(s_1,a_1) + \cdots + R(s_T, a_T)]$. We now explicitly write out the action term in our reward function, this state-action reward can be used when we want to assign different reward to different actions. (eg. For robot in a maze, staying still is preferred over wandering around, so we can charge this action; for helicopter setting, moving control sticks too aggressively is an action we want to penalize). One interesting property of finite horizon MDP is optimal policy depend on the time left, one dynamic illustration would be the policy derived for the following examples:



$t_{curr} \ll T$

$t_{curr} \to T$

Let's focus on the finite horizon property and forget about state transition probability varing over time, and our inductive equation becomes: $V_t^*(s) = \max_a [R(s,a) + \sum_{s'} P_{sa}(s') V_{t+1}^*(s)]$ with $V_t^*(s)$ being $V_t^*(s) = \mathbb{E}[R(s_t,a_t) + \cdots + R(s_T,a_T)]$ and correspondingly $\pi_t^*(s) = \arg\max_a [R(s,a) + \sum_{s'} P_{sa}(s') V_{t+1}^*(s)]$. One important thing is define $V_T^*(s) = \max_a R(s,a)$, which serves as base case for the following dynamic programming algorithm:

1. Calculate $V_T^*(s) = \max_a R(s,a)$. for $\forall s \in S$.
2. Use inductive step $V_t^*(s) = \max_a [R(s,a) + \sum_{s'} P_{sa}(s') V_{t+1}^*(s)]$ to sequentially compute $V_{T-1}^*, V_{T-2}^* \cdots$.

This algorithm is value iteration in finite horizon setting.

• Linear Quadratic Regulation (with relatively small size)

In small set of problems with finite horizon property, linear quadratic Regulation is an efficient algorithm that is capable for the derivation of great control policies. To specify problem setting, say our MDP consists of five tuples $(S, A, \{P_{sa}\}, T, R)$, where $S = \mathbb{R}^n$, $A = \mathbb{R}^d$, and states evolving according to $s_{t+1} = As_t + Ba_t + w_t$, with $w_t \sim \mathcal{N}(0, \Sigma_w)$. One final assumption for applying LQR is that our reward function takes the form $R(s,a) = -(s^T U s + a^T V a)$ where $U \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{d \times d}$ and U, V are both PSD. One example for this is suppose that we what our helicopter to stay in state $s \approx 0$, then we can choose $U = I$ and $V = I$ to obtain $R(s,a) = -(\|s\|^2 + \|a\|^2)$ which intuitively, penalizes drastic changes in state and action.

Now let's focus on calculating A, B. Previously in our discussion of model-based RL, we flew our helicopter m times to obtain m trajectories each consisting T time points, then we performed linear regression leading to minimize $\sum_{m=1}^{M} \sum_{t=0}^{T} \|s_{t+1} - (As_t + Ba_t)\|^2$. LQR derives from the idea that if $s_{t+1} = f(s_t, a_t)$ and f is determined (e.g. in inverted pendulum f can be derived from laws of physics), can we linearize f around certain point so that A, B can be derived from the linear representation directly? This idea leads us to $s_{t+1} \approx f(\bar{s}_t, \bar{a}_t) + \nabla_s f(\bar{s}_t, \bar{a}_t)^T (s_t - \bar{s}_t) + \nabla_a f(\bar{s}_t, \bar{a}_t)^T (a_t - \bar{a}_t)$. where $\bar{s}_t, \bar{a}_t$ are "typical points". We choose $\bar{s}_t$ and $\bar{a}_t$ to be the state/action where we want our model to spend most of its time on. Since in a small vincinity of $\bar{s}_t$ and $\bar{a}_t$, the linearized function is a fair estimate of the true curve. Based on the linearization, we estimate $A = \nabla_{s_t} f(\bar{s}_t, \bar{a}_t)^T$, $B = \nabla_{a_t} f(\bar{s}_t, \bar{a}_t)^T$, notice there are extra terms in the linear part, thus we adjust A to be $\hat{A} = \left[f(\bar{s}_t, \bar{a}_t) - \nabla_s f - \nabla_a f \mid A \right]$

and corresponding add an intercept term to s_t to obtain $s_t = \begin{bmatrix} 1 \\ s_t \end{bmatrix}$.

With linear quadratic regulation, hopefully we can model a MDP as a linear quadratic system, either S_{t+1} is a linear function of s_t, a_t, or $s_{t+1} = f(s_t, a_t)$ with f being non-linear mapping, but can be modelled linearly around the point of $(\bar{s_t}, \bar{a_t})$. It turns out that with s_{t+1} being linear function of s_t, a_t and cost function being quadratic, we can compute V^*, which would be a quadratic function, exactly.

We will be developing a dynamic programming algorithm to demonstrate this point. Our base case would be $V_T^*(s_T) = \max_{a_T} R(s_T, a_T) = \max_{a_T} -(s_T^T U s_T + a_T^T V a_T)$, since $U, V \geq 0$, the best we can do to maximize $V_T^*(s_T)$ is to choose a_T s.t. $a_T^T V a_T = 0$, which leads to $V_T^*(s_T) = -s_T^T U s_T$, and $\pi_T^*(s_T) = \vec{0}$. We will assume that $V_{t+1}^*(s_{t+1}) = s_{t+1}^T \Phi_{t+1} s_{t+1} + \psi_{t+1}$ where $\Phi_{t+1} \in \mathbb{R}^{n \times n}, \psi_{t+1} \in \mathbb{R}$ is a quadratic function, we will then show after one iteration of dynamic programming, $V_t^*(s_t)$ will also be a quadratic function, which inductively proves our hypothesis (claim).

Recall that $V_{a_t}^*(s_t) = \max_{a_t} (R(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim p(s_t, a_t)}[V_{t+1}^*(s_{t+1})])$

$= \max_{a_t} \left(-(s_t^T U s_t + a_t^T V a_t) + \mathbb{E}_{s_{t+1} \sim \mathcal{N}(As_t + Ba_t, \Sigma_w)} [s_{t+1}^T \Phi_{t+1} s_{t+1} + \psi_{t+1}] \right)$ [reminder: inside expectation is a scalar].

$= \max_{a_t} [-(s_t^T U s_t + a_t^T V a_t) + tr(\Phi_{t+1} \Sigma_w) + \mu_t^T \Phi_{t+1} \mu_t + \psi_{t+1}]$, where $\mu_t = As_t + Ba_t$.

$= \max_{a_t} [(-(s_t^T U s_t) + s_t^T A^T \Phi_{t+1} A s_t + \psi_{t+1}) + (-a_t^T V a_t + s_t^T A^T \Phi_{t+1} B a_t + a_t^T B^T \Phi_{t+1} A s_t + a_t^T B^T \Phi_{t+1} B a_t)]$

↓ constant w.r.t a_t ; $+ tr(\Phi_{t+1} \Sigma_w)$; ↓ big quadratic function of a_t.

If we take derivative of the above function and set to zero to solve a_t, we would obtain:

$$a_t = -(B^T \Phi_{t+1} B - V)^{-1} B^T \Phi_{t+1} A s_t$$

we will denote the linear part $-(B^T \Phi_{t+1} B - V)^{-1} B^T \Phi_{t+1} A$ as L_t, thus $\pi_t^*(s_t) = L_t s_t$. To summarize, in linear dynamical system with quadratic cost function, optimal action is a linear function of the state s_t. By plugging a_t back in we are able to obtain $V_t^*(s_t) = s_t^T [A^T (\Phi_{t+1} + \Phi_{t+1} B (V - B^T \Phi_{t+1} B)^{-1} B^T \Phi_{t+1}) A - U] s_t + tr(\Phi_{t+1} \Sigma_w) + \psi_{t+1}$, which directly shows that $V_t^*(s_t)$ is also a quadratic function of s_t which supports our claim.

One thing to notice is that in $a_t = L_t s_t$, L_t is only represented through Φ_{t+1} but not ψ_{t+1}, which indicates, when doing dynamic programming and iteratively update Φ_t, ψ_t using Φ_{t+1} and ψ_{t+1}, we can get rid of the calculation of ψ_t cause it's unrelated to a_t. Moreover, since Σ_w only appears in the calculation of ψ_t, then the specific choice of Σ_w won't affect our calculation.

of the optimal policy. To summarize, when modeling MDP as a linear dynamical system, the specific choice for the covariance of noise does not matter since it doesn't affect the optimal policy. Remember this is a property that is very specific to LQR, so do not overgeneralize it to other reinforcement learning algorithms.