Machine Learning Algorithms

List of concepts

1 Dot product in Feature space
2 State space and Feature space
3 Correlation and Convolution
4 Markovian vs Martingale

List of learning algorithms

1.1	Bayes rule			
1.2	Naïve Bayes classifier			
1.3	Maximum likelihood		*X is data*	*$\mu\Sigma$ are parameters*
1.4	Maximum aposteria		*X is data*	*$\mu\Sigma$ are parameters*
1.5	Linear regression *(AX-B in ASM)*	*max likelihood*	*A,B are data*	*X is parameter*
2.1	Logistic regression	*max likelihood*	*X,Y are data*	*θ is parameter*
2.2	Expectation maximization	*max likelihood*	*X(Z) are data*	*θ is parameter*
2.3	Gaussian mixture model	*max likelihood*	*X(Z) are data*	*$\mu\Sigma$ are parameters*
2.4	K mean clustering	*min distance*	*X(Z) are data*	*μ is parameter*
2.5	K nearest neightbour	*min distance*		
2.6	KL divergence	*min divergence*	*X is data*	*θ is parameter*
2.7	Decision tree	*min entropy*	*X,Y are data*	
3.1	Kalman filter	*max likelihood*	*A,B are data*	*X is parameter*
3.2	Principal component analysis	*variance analysis*	*X is data*	
3.3	Linear discriminant analysis	*variance analysis*	*X,Y are data*	
3.4	Neural network (RNN, CNN)	*min empirical risk*	*X,Y are data*	*w is parameter*
3.5	Support vector machine	*min structural risk*	*X,Y are data*	*α is parameter*
3.6	Generative adversarial network	*game theory (Oligopoly)*		
4.1	Reinforcement learning	*Markov chain*		
4.2	Hidden Markov model	*Markov chain*		

Generative vs Discriminative classifers

Generative classifers models distribution of each class without considering interclass differences, and find the nearest in inspection. Discriminative classifers model the differences or boundary between classes without considering distribution of each class. There is no *decision boundary* between classes for generative classifiers, there is no *uniclass distribution model* for discriminative classifier.

Generative classifer models P(X,y)		*Discriminative classifier models P(X|y)*
class 1	*P(X, y=1)*	
class 2	*P(X, y=2)*	
…		
example : Naïve Bayes, GMM, GAN		*example : logestic regression, NN, SVM*

Summary 9*3,3*3,3*10,3*4 + 354113333

9*3

		objective	data	para	content
2.1	LR	*max likelihood*	X,Y	θ	3 = *define L(θ), max L(θ), linear case*
2.2	EM	*max likelihood*	$X(Z)$	θ	5 = *define convex, log, define L′(θ), max L′(θ), algo*
2.3	GMM	*max likelihood*	$X(Z)$	$\mu\Sigma$	4 = *define pdf, Estep, Mstep, weight matrix*
2.4	Kmean	*min distance*	$X(Z)$	μ	1
2.5	KNN	*min distance*			1
2.6	KLdiv	*min divergence*	X	θ	3 = *define entropy, define divergence, max likelihood*
2.7	Decision tree	*min entropy*	X,Y		3 = *inspection, construct tree, train tree*
3.4	NN	*min empirical risk*	X,Y	w	3 = *define topology, algorithm, proof*
3.5	SVM	*min structural risk*	X,Y	α	3 = 10,10,10

Comparison among Gauss Newton / logistic regression and neural network

3*3

	objective	matrix	algorithm
Gauss Newton	$\min_{\Delta X}(F(A\|X^{(t)})+J_F\Delta X-B)^T\Sigma^{-1}(F(A\|X^{(t)})+J_F\Delta X-B)$	$F(A\|X),J_F$	*Newton Raphson*
logistic regression	$\max_{\theta}\ln\prod_{n=1}^{N}[s(f(x_n\|\theta))]^{y_n}[1-s(f(x_n\|\theta))]^{1-y_n}$	$f(x_n\|\theta),J_F$	*grad ascent* $\Delta\theta=J_F^TE$
neural network	$\min_{W}(F(X\|W)-Y)^T(F(X\|W)-Y)$	$F(X\|w)$ *only*	*grad descent* $\Delta w=-s\partial_wE$

Definition of F and J in Gauss Newton, please modify accordingly for logistic regression and neural network :

$$F(A|X)=\begin{bmatrix} f(A_1|X)\\ f(A_2|X)\\ f(A_3|X)\\ \ldots\\ f(A_N|X)\end{bmatrix}\qquad J_F=\begin{bmatrix} \partial f(A_1|X)/\partial x_1 & \partial f(A_1|X)/\partial x_2 & \partial f(A_1|X)/\partial x_3 & \ldots & \partial\partial f(A_1|X)/\partial x_M\\ \partial f(A_2|X)/\partial x_1 & \partial f(A_2|X)/\partial x_2 & \partial f(A_2|X)/\partial x_3 & \ldots & \partial f(A_2|X)/\partial x_M\\ \partial f(A_3|X)/\partial x_1 & \partial f(A_3|X)/\partial x_2 & \partial f(A_3|X)/\partial x_3 & \ldots & \partial f(A_3|X)/\partial x_M\\ \ldots & \ldots & \ldots & \ldots & \ldots\\ \partial f(A_N|X)/\partial x_1 & \partial f(A_N|X)/\partial x_2 & \partial f(A_N|X)/\partial x_3 & \ldots & \partial f(A_N|X)/\partial x_M\end{bmatrix}$$

3*10

	SVC		SVC		SVR	
#arguments in step1 obj-fct	2	w,b	3	w,b,ξ	4	w,b,ξ^+,ξ^-
#constraints in step1	1	$y_nf(x_n)\ge 1$	2	$y_nf(x_n)\ge\ldots,\ \xi_n\ge 0$	4	$f(x_n)\pm\varepsilon\ldots\ge y_n\pm\xi_n^\pm,\ \xi_n^\pm\ge 0$
#arguments in step2 obj-fct	3=21	w,b,α	5=32	w,b,ξ,α,β	8=44	$w,b,\xi^\pm,\alpha^\pm,\beta^\pm$
#constraints in step2	1	$\alpha\ge 0$	2	$\alpha\ge 0,\ \beta\ge 0$	4	$\alpha^\pm\ge 0,\ \beta^\pm\ge 0$
#arguments in step3 obj-fct	1	α	1	α	1	A
#constraints in step3	2	$\alpha\ge 0,\ y^T\alpha=0$	2	$\alpha\in[0,C],\ y^T\alpha=0$	2	$A\in\pm C,\ l^TA=0$
y in Lag 2nd order term		*yes*		*yes*		-
y in Lag 1st order term?		-		-		*yes*
y in linear constraint?		*yes*		*yes*		-
y in w vector?		*yes*		*yes*		-

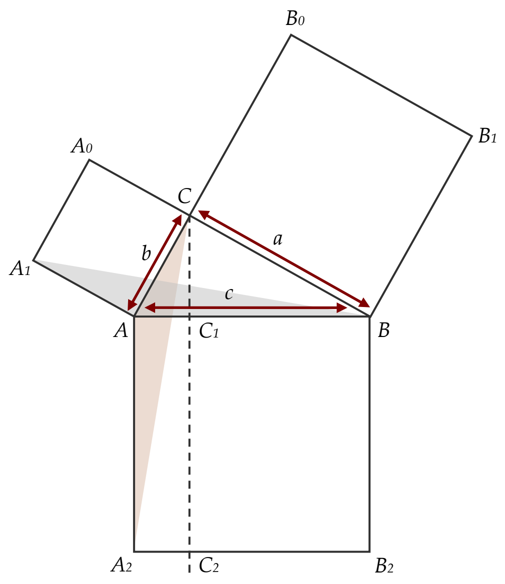
3*4 *SVC* $\max_{\alpha}-\frac{1}{2}\alpha^TH\alpha+l^T\alpha$ *st* $\alpha\ge 0$ *and* $y^T\alpha=0$ *where* $H_{nm}=y_ny_mx_nx_m^T$ *and* $\alpha=\begin{bmatrix}\alpha_1\\ \alpha_2\\ \alpha_3\\ \ldots\end{bmatrix}$

SVC $\max_{\alpha}-\frac{1}{2}\alpha^TH\alpha+l^T\alpha$ *st* $0\le\alpha\le C$ *and* $y^T\alpha=0$ *where* $H_{nm}=y_ny_mx_nx_m^T$ *and* $\alpha=\begin{bmatrix}\alpha_1\\ \alpha_2\\ \alpha_3\\ \ldots\end{bmatrix}$

SVR $\max_{\alpha^\pm}-\frac{1}{2}\alpha^TH\alpha+y^T\alpha-l^T\alpha*\varepsilon$ *st* $0\le\alpha\le C$ *and* $y^T\alpha=0$ *where* $H_{nm}=x_nx_m^T$ *and* $\alpha=\begin{bmatrix}\alpha_1^+-\alpha_1^-\\ \alpha_2^+-\alpha_2^-\\ \alpha_3^+-\alpha_3^-\\ \ldots\end{bmatrix}$ $\alpha*=\begin{bmatrix}\alpha_1^++\alpha_1^-\\ \alpha_2^++\alpha_2^-\\ \alpha_3^++\alpha_3^-\\ \ldots\end{bmatrix}$

1 Dot product in Feature space

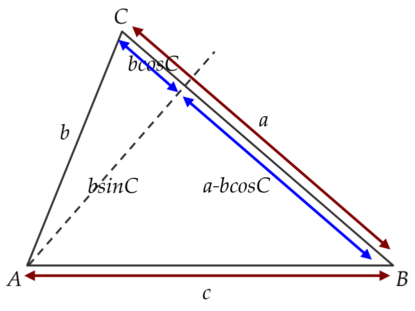
Dot product of 2 feature vectors measures their cosine similarity. Let's prove it starting from Pythagoras theorem and law of cosine.



- *points ACB_0 are collinear and*
- *points BCA_0 are collinear only if C is a right angle*
- *ABA_1 has same area as ACA_2 (same sides and included amgle)*
- *ACA_1 has same area as AC_1A_2 (shifting B to C, shifting C to C_1)*
- *ACA_0A_1 has same area as $AC_1C_2A_2$ (doubling triangles into square & rectangle)*
- *BCB_0B_1 has same area as $BC_1C_2B_2$ (repeating all above steps for RHS)*
- *$AB^2 = AC^2 + BC^2$ (sum of the two results above)*

or $c^2 = b^2 + a^2$

After that, we will generalise Pythagoras theorem to non-right angle triangle.



$$\begin{aligned} c^2 &= (b\sin C)^2 + (a - b\cos C)^2 \\ &= b^2\sin^2 C + a^2 - 2ab\cos C + b^2\cos^2 C \\ &= b^2 + a^2 \underbrace{- 2ab\cos C}_{\substack{\text{extra-team} \\ \text{comparing} \\ \text{Pythagoras}}} \end{aligned}$$

The extra term vanishes if C is right angle.

This is called the law of cosine. Now we can show why dot product is equivalent to projection, and thus related to cosine similarity. Given vector $A,B \in \Re^M$, and $C = A\text{-}B$, then we have :

$$\begin{aligned} c^2 &= \vec{C}\cdot\vec{C} \\ &= (\vec{A}-\vec{B})\cdot(\vec{A}-\vec{B}) \\ &= \vec{A}\cdot\vec{A} - 2\vec{A}\vec{B} + \vec{B}\cdot\vec{B} \\ &= b^2 + a^2 - 2\vec{A}\vec{B} \\ &= b^2 + a^2 - 2ab\cos C \end{aligned}$$

using law of cosine

Recall :		
$\vec{A}\cdot\vec{B}$	=	*dot product*
$\vec{A}\cdot\vec{B}/\lvert\vec{B}\rvert$	=	*projection of A on B*
$\vec{A}\cdot\vec{B}/(\lvert\vec{A}\rvert\lvert\vec{B}\rvert)$	=	*cosine similarity*

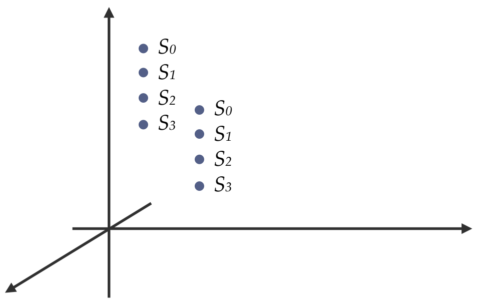
Therefore we have :

$$\vec{A}\vec{B} = ab\cos C = \lvert\vec{A}\rvert\lvert\vec{B}\rvert\cos(\angle AOB)$$

where O is the origin

2 State space and Feature space

Discrete state space is represented as a set, $S = \{S_0, S_1, \ldots, S_N\}$, while continuous state space is usually represented as multidimension feature space, $F \in \Re^M$ under some boundary constraints. Here is the correspondence :



3 Correlation and Convolution

Correlation is defined as inner product :

$$\begin{aligned} c(t) &= \int f(s)g(s+t)ds \\ c(0) &= \int f(s)g(s)ds && \textit{i.e. dot product of functions} \end{aligned}$$

Convolution is defined as :

$$c(t) = \int f(s)g(t-s)ds$$

4 Markovian vs Martingale

Markovian system is a memoryless system, probability of future states conditional on current information depends on current state only, while historical states are not involved, in other words :

$$\begin{aligned} & \Pr(X_{n+1} = x \mid X_n, ..., X_1, X_0) &= \Pr(X_{n+1} = x \mid X_n) && \textit{for discrete space} \\ \textit{or} \quad & \Pr(X_T \in A \mid I_t) &= \Pr(X_T \in A \mid X_t) && \textit{for continuous space} \end{aligned}$$

Examples of Markovian process include : Markov chain which is usually represented as a directed graph, chess and Black Scholes :

$$dX_t = \mu X_t dt + \sigma X_t dz_t \qquad \textit{depending on current } X_t \textit{ only}$$

Non Markovian process include autoregressive time series which can be generalized into *ARMA* and *GARCH*. *AR* model is :

$$X_{n+1} = \sum_{m=0}^{M} w_m X_{n-m} + \varepsilon_{n+1} \qquad \textit{depending on historical states, declare M extra variables for history}$$

Given *M+1* observation set, the parameter vector *w* can be solved by least square, resulting in the Yule Walker equation.

Martingale process is defined as a process, which has future expectation conditional on current information equals to current value.

$$\begin{aligned} & E(X_{n+1} \mid X_n, ..., X_1, X_0) &= X_n && \textit{for discrete space} \\ \textit{or} \quad & E(X_T \mid I_t) &= X_t && \textit{for continuous space} \end{aligned}$$

Example of martingale process include risk neutral Black Scholes with risk free discounting. Martingale concept is important in the fundalmental theorem of asset pricing, from which risk neutral pricing is derived. Continuous martingale is equivalent to :

$$\begin{aligned} & E(X_{t+dt} \mid I_t) &= X_t \\ \Rightarrow \quad & E(X_t + dX_t \mid I_t) &= X_t \\ \Rightarrow \quad & E(dX_t \mid I_t) &= X_t - E(X_t \mid I_t) \\ & &= 0 && \textit{hence it implies process with no drift} \end{aligned}$$

Comparison

Therefore Markovian is defined using conditional probability, whereas martingale is defined using conditional expectation, besides, the former has *RHS* depending on current state, while the latter has *RHS* equivalent to current state. Comparison in discrete space :

$$\begin{aligned} \Pr(X_{n+1} = x \mid X_n, ..., X_1, X_0) &= \Pr(X_{n+1} = x \mid X_n) && \textit{for Markovian} \\ E(X_{n+1} \mid X_n, ..., X_1, X_0) &= X_n && \textit{for Martingale} \end{aligned}$$

There are *4* combinations :

	Martingale	*non martingale*
Markovian	$dX_t = \sigma f(X_t)dz_t$	$dX_t = \mu dt + \sigma f(X_t)dz_t$
non Markovian	$dX_t = \sigma f(X_s)dz_t \quad \forall s < t$	$dX_t = \mu dt + \sigma f(X_s)dz_t \quad \forall s < t$

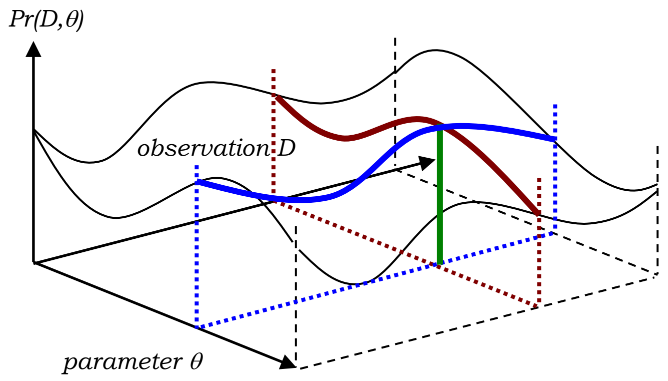
1.1 Bayes rule

Bayes rule relates posterior probability with prior probability.

$$\Pr(\theta \mid D) = \frac{\Pr(D \mid \theta)\Pr(\theta)}{\Pr(D)}$$

where	θ	=	hypothesis	(i.e. model parameters)
	D	=	evidence	(i.e. noisy observations)
	$\Pr(\theta)$	=	prior probability	(i.e. probability of hypothesis without observation)
	$\Pr(\theta \mid D)$	=	posterior probability	(i.e. probability of hypothesis given some observations)
	$\Pr(D \mid \theta)$	=	likelihood	(i.e. probability of having observations given the hypothesis)

In terms of joint distribution, we have :



$$\Pr(\theta \mid D) = \frac{\Pr(D,\theta)}{\Pr(D)}$$

$$\Pr(D \mid \theta) = \frac{\Pr(D,\theta)}{\Pr(\theta)}$$

where	$\Pr(\theta)$	=	volume under blue curve
	$\Pr(D)$	=	volume under red curve
	$\Pr(D,\theta)$	=	volume of green pillar

Two methods of model parameter estimation :

Maximum likelihood $\quad \arg\max_{\theta} \ln \Pr(D \mid \theta)$

Maximum aposterior

$$\arg\max_{\theta} \ln \Pr(\theta \mid D) = \arg\max_{\theta} \ln \frac{\Pr(D \mid \theta)\Pr(\theta)}{\Pr(D)}$$

$$= \arg\max_{\theta} \ln[\Pr(D \mid \theta)\Pr(\theta)] \qquad \textit{where Pr is pmf or pdf}$$

1.2 Naïve Bayes Classifier

Naïve Bayes is a maximum-joint-probability-classifier, which breaks down the joint-probability into a chain of single-dimensional conditional-probabilities. Given N training vectors having M attributes together with class label $x_n \in \Re^M$ $y_n \in [1,K]$ $\forall n \in [1,N]$, there are K possible classes, it classifies test vector $x \in \Re^M$ naively by considering conditional-probabilities in each dimension separately :

$$y = \arg\max_{k} \Pr(x, y=k)$$

$$= \arg\max_{k} \Pr(x_{[1]} \mid x_{[2:M]}, y=k) \times \Pr(x_{[2:M]}, y=k) \qquad \textit{index } [m] \textit{ denotes attribute } m$$

$$= \arg\max_{k} \Pr(x_{[1]} \mid x_{[2:M]}, y=k) \times \Pr(x_{[2]} \mid x_{[3:M]}, y=k) \times \Pr(x_{[3:M]}, y=k)$$

$$= \arg\max_{k} \Pr(x_{[1]} \mid x_{[2:M]}, y=k) \times \Pr(x_{[2]} \mid x_{[3:M]}, y=k) \times \ldots\times r(x_{[M-1:M]} \mid x_{[M]}, y=k) \times \Pr(x_{[M]} \mid y=k) \times \Pr(y=k)$$

$$= \arg\max_{k} \prod_{m=1}^{M} \underbrace{\Pr(x_{[m]} \mid y=k)}_{cond-prob-1D-attr} \Pr(y=k) \qquad \textit{if all M attributes are independent}$$

For continuous case

We have to make model-assumptions *(such as 1D Gaussian)* on conditional-probability for each *1D* attribute, we estimate the model parameters using training dataset (and some algorithms). Finally we can classify incoming test cases using the above formula.

For discrete case

If all the attributes are discrete values, then we can find the conditional probabilities from training set simply by counting.

$$\Pr(x_{[m]} \mid y=k) = \frac{\sum_{n=1}^{N} 1(x_{n[m]} = v, y_n = k)}{\sum_{n=1}^{N} 1(y_n = k)} \qquad \textit{for all } v \in [\textit{possible values of attribute } m]$$

for all $m \in [1,M]$ *and for all* $k \in [1,K]$

Thus the learnt record is a *3D* tensor.

1.3 Maximum Likelihood

Lets try maximum likelihood estimation with multivariate Gaussian. Given a set of *i.i.d.* (independent-identically distributed) data points $X=\{x_1, x_2, x_3, \ldots, x_N\}$ in $\Re^M$, drawn from $\aleph(x;\mu,\Sigma)$, suppose all data x_n and μ are row vectors, then log likelihood function is :

$$\begin{aligned}
\ln P(X\mid\mu,\Sigma) &= \ln\left[\prod_{n=1}^{N}\frac{1}{\sqrt{(2\pi)^{rank}\det(\Sigma)}}\exp\left(-\frac{(x_n-\mu)\Sigma^{-1}(x_n-\mu)^T}{2}\right)\right] && \textit{we then apply the "trace-trick"}\\
&= -\frac{N}{2}\ln\det(\Sigma)-\frac{1}{2}\textstyle\sum_{n=1}^{N}(x_n-\mu)\Sigma^{-1}(x_n-\mu)^T+const && (1)\\
&= -\frac{N}{2}\ln\det(\Sigma)-\frac{1}{2}\textstyle\sum_{n=1}^{N}tr((x_n-\mu)\Sigma^{-1}(x_n-\mu)^T)+const && \textit{since trace(scalar) = scalar}\\
&= -\frac{N}{2}\ln\det(\Sigma)-\frac{1}{2}\textstyle\sum_{n=1}^{N}tr(\Sigma^{-1}(x_n-\mu)^T(x_n-\mu))+const && \textit{since trace is invariant to cyclic permutation}\\
&= -\frac{N}{2}\ln\det(\Sigma)-\frac{1}{2}tr(\Sigma^{-1}\textstyle\sum_{n=1}^{N}(x_n-\mu)^T(x_n-\mu))+const && (2)
\end{aligned}$$

Differentiate log likelihood in *(1)* wrt μ and setting it to zero, we have :

$$\begin{aligned}
0 &= -\frac{1}{2}\textstyle\sum_{n=1}^{N}(-2)\Sigma^{-1}(x_n-\mu)^T && \textit{since } \partial_X(AX-B)\Sigma(AX-B)^T=2A\Sigma(AX-B)^T\\
\mu &= \frac{1}{N}\textstyle\sum_{n=1}^{N}x_n
\end{aligned}$$

Differentiate log likelihood in *(2)* wrt Σ^{-1} and setting it to zero, we have :

$$\begin{aligned}
\partial_{\Sigma^{-1}}\left(\frac{N}{2}\ln\det(\Sigma)\right) &= \partial_{\Sigma^{-1}}\left(-\frac{1}{2}tr(\Sigma^{-1}\textstyle\sum_{n=1}^{N}(x_n-\mu)^T(x_n-\mu))\right)\\
&= -\frac{1}{2}\textstyle\sum_{n=1}^{N}((x_n-\mu)^T(x_n-\mu))^T && \textit{since } \partial_A Tr(AB)=\partial_A Tr(BA)=B^T\\
&= -\frac{1}{2}\textstyle\sum_{n=1}^{N}(x_n-\mu)^T(x_n-\mu) && \textit{since } (AB)^T=B^TA^T\\
\partial_\Sigma(\ln\det(\Sigma))\partial_{\Sigma^{-1}}\Sigma &= -\frac{1}{N}\textstyle\sum_{n=1}^{N}(x_n-\mu)^T(x_n-\mu)\\
\Sigma^{-T}\partial_{\Sigma^{-1}}\Sigma &= -\frac{1}{N}\textstyle\sum_{n=1}^{N}(x_n-\mu)^T(x_n-\mu) && \textit{since } \partial_A(\ln\det(A))=(A^{-1})^T=A^{-T}\\
\Sigma^{-T}(-\Sigma\Sigma) &= -\frac{1}{N}\textstyle\sum_{n=1}^{N}(x_n-\mu)^T(x_n-\mu) && \textit{since } \partial_A A^{-1}=-A^{-1}A^{-1} \textit{ and } \partial_{A^{-1}}A=-AA\\
\Sigma &= \frac{1}{N}\textstyle\sum_{n=1}^{N}(x_n-\mu)^T(x_n-\mu) && \textit{result is intuitive}
\end{aligned}$$

Remark for matrix differentiation

Lets prove $\partial_A(\ln\det(A))=(A^{-1})^T=A^{-T}$ *. Consider derivative wrt element* a_{ij} *:*

$$\begin{aligned}
\partial_{a_{ij}}(\ln\det(A)) &= (\det(A))^{-1}\partial_{a_{ij}}\det(A)\\
&= (\det(A))^{-1}cofactor(A)_{ij}\\
&= (A^{-1})_{ji} \quad\Rightarrow\quad \partial_A(\ln\det(A))=(A^{-1})^T=A^{-T}
\end{aligned}$$

Recall for matrix X with row data
X^TX is covariance matrix
XX^T is dot-product matrix

Lets prove $\partial_A A^{-1}=-A^{-1}A^{-1}$.

$$\begin{aligned}
I &= AA^{-1}\\
\partial_A I &= A(\partial_A A^{-1})+(\partial_A A)A^{-1}\\
0 &= A(\partial_A A^{-1})+A^{-1} \quad\Rightarrow\quad \partial_A A^{-1}=-A^{-1}A^{-1}
\end{aligned}$$

1.4 Maximum Aposterior

If we are given prior knowledge about μ and Σ, we can incorporate them into maximum aposterior.

$$\ln[P(X\mid\mu,\Sigma)\Pr(\mu,\Sigma)] \quad = \quad \ln\left[\prod_{n=1}^{N}\frac{1}{\sqrt{(2\pi)^{rank}\det(\Sigma)}}\exp\left(-\frac{(x_n-\mu)\Sigma^{-1}(x_n-\mu)^T}{2}\right)\right]+\ln P_{prior}(\mu,\Sigma)$$

Suppose we know in prior that, parameter μ follows a M dimensional Gaussian centred at μ_0, having covariance matrix Σ_0, whereas nothing is known about Σ, then we have (recall that both μ and μ_0 are row vectors) :

$$\begin{aligned}\ln[P(X\mid\mu,\Sigma)\Pr(\mu,\Sigma)] &= \ln\left[\prod_{n=1}^{N}\frac{1}{\sqrt{(2\pi)^{rank}\det(\Sigma)}}\exp(-\frac{1}{2}(x_n-\mu)\Sigma^{-1}(x_n-\mu)^T)\right]+\ln\left[\frac{1}{\sqrt{(2\pi)^{M}\det(\Pi)}}\exp(-\frac{1}{2}(\mu-\mu_0)\Sigma_0^{-1}(\mu-\mu_0)^T)\right] \\ &= -\frac{N}{2}\ln\det(\Sigma)-\frac{1}{2}\sum_{n=1}^{N}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)-\frac{N}{2}\ln\det(\Pi)-\frac{1}{2}(\mu-\mu_0)^T\Sigma_0^{-1}(\mu-\mu_0)+const\end{aligned}$$

If Σ is given, the problem becomes maximum likelihood with regularization (analogous to linear regression with regularization) :

$$\ln[P(X\mid\mu,\Sigma)\Pr(\mu,\Sigma)] \quad = \quad -\frac{1}{2}\sum_{n=1}^{N}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)-\frac{1}{2}(\mu-\mu_0)^T\Sigma_0^{-1}(\mu-\mu_0)+const^* \qquad \textit{moving known terms into const}$$

Differentiate log likelihood above wrt μ and setting it to zero, we have :

$$\begin{aligned}0 &= -\frac{1}{2}\sum_{n=1}^{N}(-2)\Sigma^{-1}(x_n-\mu)^T-\frac{1}{2}(2)\Sigma_0^{-1}(\mu-\mu_0)^T \\ 0 &= \sum_{n=1}^{N}(x_n-\mu)\Sigma^{-T}-(\mu-\mu_0)\Sigma_0^{-T} && \textit{transpose on both sides} \\ \mu(N\Sigma^{-T}+\Sigma_0^{-T}) &= (\sum_{n=1}^{N}x_n)\Sigma^{-T}+\mu_0\Sigma_0^{-T} \\ \mu &= (N\Sigma^{-T}+\Sigma_0^{-T})^{-1}(\sum_{n=1}^{N}x_n)\Sigma^{-T}+\mu_0\Sigma_0^{-T} \\ &= \textit{weighted average between given } \mu_0 \textit{ and mean of data} \\ \textit{weight} &\propto \textit{inverse of covariance}\end{aligned}$$

1.5 Linear Regression

Please read ASM's AX=B documents for details.

Comparison between *ML* for mean estimation and *ML* for *AX=B* :

	Gaussian paramter estimation	Linear regression
A	*no*	*N×M*
B	*no*	*N×1*
X	*N×M*	*M×1*
variable	*X_n i.e. $\Re^M$ row vector*	*$(AX-B)_n$ i.e. $\Re^1$ scalar*
μ	*mean vector of X_n*	*mean vector of $(AX-B)_n$, i.e. all zero ideally*
Σ	*covariance between M axes*	*covariance between N data*

2.1 Logistic Regression

Logistic regression is a maximum likelihood classifier, it is an extension from regression to classification, via treating the regression function as a separating plane. Given N *i.i.d.* labelled data points $\{x_n \in \Re^M, y_n \in [0,1], n \in [1,N]\}$, the objective is to find separating plane $f(x;\theta) = 0$ so that $s(f(x;\theta)) \in \Re^1$ returns a scalar value above 0.5 for class 1 and a scalar value below 0.5 for class 0, where $s(z)$ is known as sigmoid function or logistic function, which tends to 1 as $z \to \infty$ and tends to 0 as $z \to -\infty$. Sigmoid function is selected because of its mathematical convenience in find first order and second order derivatives :

$$\begin{aligned}
s(z) &= (1+e^{-z})^{-1} \\
s'(z) &= (1+e^{-z})^{-2}e^{-z} \\
&= \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}} \\
&= s(z)(1-s(z)) \\
s''(z) &= \partial_z s(z)(1-s(z)) \\
&= s(z)(1-s(z))^2 - s^2(z)(1-s(z)) \\
&= s(z) - 2s^2(z) + s^3(z)
\end{aligned}$$

2.1a Define likelihood

Let's define likelihood for one data point :

$$\begin{aligned}
P(x = x_n, y = 1 \mid \theta) &= s(f(x_n \mid \theta)) \\
P(x = x_n, y = 0 \mid \theta) &= 1 - s(f(x_n \mid \theta)) \\
\text{or} \quad P(x, y \mid \theta) &= [s(f(x \mid \theta))]^y [1 - s(f(x \mid \theta))]^{1-y}
\end{aligned}$$

Let's define likelihood for one dataset :

$$\begin{aligned}
L(\theta) &= \prod_{n=1}^N P(x_n, y_n \mid \theta) \\
&= \prod_{n=1}^N [s(f(x_n \mid \theta))]^{y_n} [1 - s(f(x_n \mid \theta))]^{1-y_n} \\
\ln L(\theta) &= \sum_{n=1}^N y_n \ln[s(f(x_n \mid \theta))] + (1-y_n)\ln[1 - s(f(x_n \mid \theta))]
\end{aligned}$$

looks like Kelly criteria, which has E[], just a coincidence

2.1b Maximise likelihood

Logistic regression is maximizing likelihood by gradient ascent.

$$\begin{aligned}
\partial_\theta \ln L(\theta) &= \sum_{n=1}^N \frac{y_n}{s(f(x_n \mid \theta))}\partial_\theta s(f(x_n \mid \theta)) - \frac{1-y_n}{1-s(f(x_n \mid \theta))}\partial_\theta s(f(x_n \mid \theta)) \\
&= \sum_{n=1}^N \left[\frac{y_n}{s(f(x_n \mid \theta))} - \frac{1-y_n}{1-s(f(x_n \mid \theta))}\right]\partial_f s(f) \times \partial_\theta f(x_n \mid \theta) \\
&= \sum_{n=1}^N \left[\frac{y_n}{s(f(x_n \mid \theta))} - \frac{1-y_n}{1-s(f(x_n \mid \theta))}\right] s(f(x_n \mid \theta))(1-s(f(x_n \mid \theta))) \times \partial_\theta f(x_n \mid \theta) \\
&= \sum_{n=1}^N \left[y_n(1-s(f(x_n \mid \theta))) - (1-y_n)s(f(x_n \mid \theta))\right] \times \partial_\theta f(x_n \mid \theta) \\
&= \sum_{n=1}^N \left[y_n - s(f(x_n \mid \theta))\right] \times \partial_\theta f(x_n \mid \theta) \\
&= J_F^T E \\
\text{where} \quad J_F[n,m] &= \partial_{\theta_m} f(x_n \mid \theta) \\
E[n] &= y_n - s(f(x_n \mid \theta))
\end{aligned}$$

J_F is NxM Jacobian matrix

E is Nx1 error col matrix

2.1c Linear case

If we pick linear separating plane $f(x;\theta) = x\theta = 0$ (recall that we can always append each vector x_n by 1 and extend to $\Re^{M+1}$, assuming data vectors in X are row vectors, parameter vector θ is column vector, i.e. *ASM* notation : *AX+B*), we have the first implementation method, gradient ascent with size step α :

$$\begin{aligned}
\partial_\theta \ln L(\theta) &= \sum_{n=1}^N \left[y_n - s(f(x_n \mid \theta))\right] x_n^T \\
\Rightarrow \quad \theta_t &= \theta_{t-1} + \alpha J_F^T E \\
&= \theta_{t-1} + \alpha X^T E
\end{aligned}$$

2.2 Expectation Maximization

Suppose an observable random vector $X \in \Re^{M1}$ and a non-observable random vector $Z \in \Re^{M2}$ are drawn from parametric distribution $P(X,Z|\theta)$ with unknown parameter vector θ, expectation maximization is a 2-step iterative approach that estimates parameter from one given realization of X using maximum likelihood. Main difference between *ML* and *EM* is that the latter needs to handle latent space, which is the space spanned by hidden data. Among the M_1+M_2 dimensions of one realization, M_2 are latent variables. Latent variables cannot be directly measured, but they can be inferred from observable variables. For instance, in economics, quality of life and happiness are latent, they can be inferred from wealth, employment & consumption. EM algorithm can be summarised as :

$$
\begin{aligned}
\theta_{t+1} &= \arg\max_{\theta} E_{Z|X,\theta_t}[\ln P(X,Z|\theta)] && \textit{where X and Z are observable and non-observable part of one realization set} \\
&= \arg\max_{\theta} \textstyle\sum_Z [P(Z|X,\theta_t)\ln P(X,Z|\theta)] && \textit{where P is joint distribution of one realization set} \\
\theta_{opt} &= \lim_{t\to\infty} \theta_t
\end{aligned}
$$

The implementation can be broken down into E-step and M-step. E-step estimates latent variables, M-step estimates parameters.

E-step : *update expectation by finding $P(Z|X,\theta_t)$ given estimated parameters from previous iteration t*
M-step : *maximize expectation with respect to θ using various numerical methods (analytic solution for Gaussian mixture model)*

2.2a Define convexity - three approaches

Convexity definition 1

First of all, we need to define convex function and derive its properties. A function is said to be convex on interval *[L,U]* if :

$$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda) f(x_2) \qquad \forall x_1, x_2 \in [L,U], \lambda \in [0,1]$$

Convexity definition 2

If a function is twice differentiable on interval *[L,U]* and $f''(x) \ge 0$, $\forall x \in [L,U]$, then the function is convex on interval *[L,U]*, let's prove.

$$
\begin{aligned}
x &= \lambda x_1 + (1-\lambda)x_2 && \textit{hence we have } L \le x_1 \le x \le x_2 \le H \\
\lambda x + (1-\lambda)x &= \lambda x_1 + (1-\lambda)x_2 \\
\lambda(x - x_1) &= (1-\lambda)(x_2 - x) \\
\lambda f'(s_1)(x - x_1) &\le (1-\lambda) f'(s_2)(x_2 - x) && \exists s_1 \in [x_1, x] \textit{ and } \exists s_2 \in [x, x_2] \textit{, since } f'' \ge 0, f' \textit{ is increasing, so } f'(s_1) \le f'(s_2) \\
\lambda(f(x) - f(x_1)) &\le (1-\lambda)(f(x_2) - f(x)) && \textit{apply mean value thm to LHS : there exists tangent at } s_1 \textit{ parallel to secant } [x_1,x] \\
& && \textit{apply mean value thm to RHS : there exists tangent at } s_2 \textit{ parallel to secant } [x,x_2] \\
\lambda f(x) + (1-\lambda) f(x) &\le \lambda f(x_1) + (1-\lambda) f(x_2) && \textit{i.e. f is convex within [L,U]}
\end{aligned}
$$

Convexity definition 3

We now generalize convexity definition to Jensen inequality, given function *f* being convex on *[L,U]*, then :

$$f(\textstyle\sum_n \lambda_n x_n) \le \textstyle\sum_n \lambda_n f(x_n) \qquad \textit{where } x_n \in [L,U], \forall n \textit{ and } \lambda_n \ge 0, \forall n \textit{ such that } \textstyle\sum_n \lambda_n = 1$$

The proof is done by mathematical induction. The *N=1* case is trival. Here is case *N* :

$$
\begin{aligned}
f(\textstyle\sum_{n=1}^{N} \lambda_n x_n) &= f(\lambda_N x_N + \textstyle\sum_{n=1}^{N-1} \lambda_n x_n) \\
&= f(\lambda_N x_N + (1-\lambda_N)\textstyle\sum_{n=1}^{N-1}(1-\lambda_N)^{-1}\lambda_n x_n) \\
&\le \lambda_N f(x_N) + (1-\lambda_N) f(\textstyle\sum_{n=1}^{N-1}(1-\lambda_N)^{-1}\lambda_n x_n) && \textit{apply definition of convexity} \\
&\le \lambda_N f(x_N) + (1-\lambda_N)\textstyle\sum_{n=1}^{N-1}(1-\lambda_N)^{-1}\lambda_n f(x_n) && \textit{apply case N-1} \\
&= \lambda_N f(x_N) + \textstyle\sum_{n=1}^{N-1}\lambda_n f(x_n) \\
&= \textstyle\sum_{n=1}^{N}\lambda_n f(x_n)
\end{aligned}
$$

2.2b Convexity of log

Show that *ln(x)* is strictly concave on *(0,∞)* using 2nd order differentiation :

$$\begin{aligned} \partial_x(-\ln x) &= -1/x \\ \partial_{xx}(-\ln x) &= 1/x^2 > 0 \qquad \forall x \in (0,\infty) \end{aligned}$$

With Jensen inequality, we can derive "*log of sum is greater than sum of log*" which is useful for derivation of EM algorithm.

$$\ln\left(\sum_{n=1}^N \lambda_n x_n\right) \ge \sum_{n=1}^N \lambda_n \ln(x_n)$$

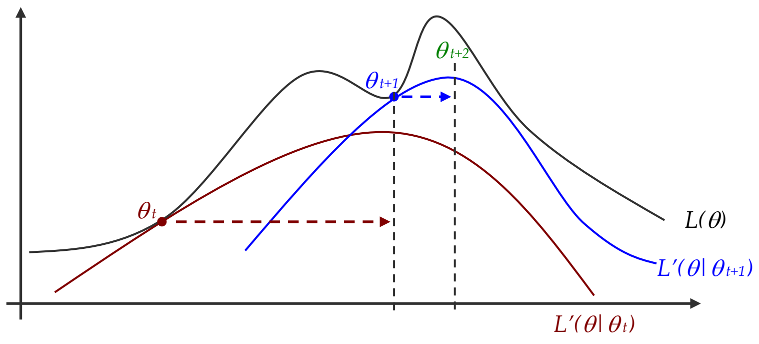
2.2c Define likelihood lower bound

Let's derive *EM* algorithm and show its convergence. In general for *EM*, we dont assume *X* to be a set of *N* independent data points. The idea is to derive the likelihood of parameter θ by incorporating latent space information. Instead of maximising this likelihood, we maximise the lower-bound of the likelihood, as the lower bound is expressed in terms of the likelihood of parameter θ_t found in iteration *t*, this results in an iterative algorithm that ensures a monotonic increase in likelihood per iteration.

$$\begin{aligned} L(\theta) &= \ln P(X\,|\,\theta) \\ &= \ln \sum_Z P(X,Z\,|\,\theta) && \textit{step 1 : construct M step probability} \\ &= \ln \sum_Z P(Z\,|\,X,\theta_t)\frac{P(X,Z\,|\,\theta)}{P(Z\,|\,X,\theta_t)} && \textit{step 2 : construct E step probability} \\ &= \ln\left[\sum_Z P(Z\,|\,X,\theta_t)\frac{P(X,Z\,|\,\theta)}{P(Z\,|\,X,\theta_t)P(X\,|\,\theta_t)}\right]P(X\,|\,\theta_t) && \textit{step 3 : construct likelihood of previous iteration } P(X|\theta_t) \\ &= \ln\left[\sum_Z P(Z\,|\,X,\theta_t)\frac{P(X,Z\,|\,\theta)}{P(Z\,|\,X,\theta_t)P(X\,|\,\theta_t)}\right] + \underbrace{\ln P(X\,|\,\theta_t)}_{L(\theta_t)} \\ &= \ln\left[\sum_Z P(Z\,|\,X,\theta_t)\frac{P(X,Z\,|\,\theta)}{P(X,Z\,|\,\theta_t)}\right] + L(\theta_t) \\ &\ge \sum_Z P(Z\,|\,X,\theta_t)\ln\frac{P(X,Z\,|\,\theta)}{P(X,Z\,|\,\theta_t)} + L(\theta_t) && \textit{log of sum greater than sum of log} \\ &\equiv L'(\theta\,|\,\theta_t) && \textit{define } L' \textit{ as lower bound of log likelihood} \end{aligned}$$

We define $L'(\theta|\theta_t)$ as the lower bound $L(\theta)$. We can show that $L(\theta)$ and $L'(\theta|\theta_t)$ meet at one point :

$$\begin{aligned} L'(\theta_t\,|\,\theta_t) &= \sum_Z P(Z\,|\,X,\theta_t)\ln\frac{P(X,Z\,|\,\theta_t)}{P(X,Z\,|\,\theta_t)} + L(\theta_t) \\ &= \sum_Z P(Z\,|\,X,\theta_t)\ln 1 + L(\theta_t) \\ &= L(\theta_t) \end{aligned}$$



2.2d Maximise likelihood lower bound

Hence by approximating $L(\theta)$ using its lower bound $L'(\theta|\theta_t)$, which touch at θ_t, we can guarantee an improvement in likelihood by maximizing the lower bound $L'(\theta|\theta_t)$, which leads to EM algorithm.

- $L(\theta) \geq L'(\theta|\theta_t)$
- $L(\theta_t) = L'(\theta_t|\theta_t)$
- $L(\theta_{t+1}) \geq \underbrace{L'(\theta_{t+1}|\theta_t) > L'(\theta_t|\theta_t)}_{} = L(\theta_t)$

update from $L(\theta_t)$ to $L(\theta_{t+1})$ can be done by max $L'(\theta|\theta_t)$

$$\begin{aligned}
\theta_{t+1} &= \arg\max_\theta L'(\theta|\theta_t) \\
&= \arg\max_\theta \sum_Z P(Z|X,\theta_t)\ln\frac{P(X,Z|\theta)}{P(X,Z|\theta_t)} + L(\theta_t) \\
&= \arg\max_\theta \sum_Z P(Z|X,\theta_t)\ln P(X,Z|\theta) && \textit{removing all non } \theta \textit{ related terms} \\
&= \arg\max_\theta E_{Z|X,\theta_t}[\ln P(X,Z|\theta)]
\end{aligned}$$

What is gained by trading the maximization of $L(\theta)$ for the maximization of $L'(\theta|\theta_t)$? The answer is that the latter approach offers a framework to take latent space Z into consideration through the two-step iterative algorithm. Prior knowledge about latent space is introduced via $P(Z|X,\theta_t)$.

2.2e EM Algorithm

EM algorithm is thus consisted of two steps :

- estimation of $P(Z|X,\theta_t)$ in ***expectation step*** which will be used as weight for next step
- estimation of $\theta_{t+1} = \arg\max_\theta E_{Z|X,\theta_t}[\ln P(X,Z|\theta)]$ in ***maximization step*** which maximises joint likelihood of X and Z

Please read "The Expectation Maximization Algorithm, a short tutorial" by Sean Borman.

2.3 Gaussian Mixture Model

Given $\aleph$ modal-GMM, generate a set of N data $[x_1, x_2, x_3, .., x_N]$ by repeatedly :

- picking one ***class label*** Z from $z \in [1, \aleph]$ with probability $[\pi_1, \pi_2, \pi_3, ..., \pi_\aleph]$
- randomizing ***data vector*** X from $x \in \Re^M$ with probability $G(x \mid \mu_z, \Sigma_z)$

Given a set of N data $[x_1, x_2, x_3, .., x_N]$, estimate $\aleph$ modal-GMM parameters by *EM* :

- in E step : update the conditional probability of data x_n belonging to a cluster z, for all N and all $\aleph$
- in M step : update the estimation of $[\pi_1, \mu_1, \Sigma_1, \pi_2, \mu_2, \Sigma_2, ..., \pi_\aleph, \mu_\aleph, \Sigma_\aleph]$ using conditional probability found in E step as weights

$\aleph$	=	*number of clusters*
N	=	*number of data vectors*
M	=	*dimension of data vectors*

	rand var set	*single rand var*	*realized value*	*possible value*
hidden latent space	Z	z_n	z	$z \in [1,2,3,...,\aleph]$
observable data space	X	x_n	x	$x \in \Re^M$
parameter vector	θ	$\begin{bmatrix} \pi_1, \pi_2, ..., \pi_\aleph \\ \mu_1, \mu_2, ..., \mu_\aleph \\ \Sigma_1, \Sigma_2, ..., \Sigma_\aleph \end{bmatrix}$		$\begin{bmatrix} \pi_1 + \pi_2 + ... + \pi_\aleph = 1 \\ \mu_1, \mu_2, ..., \mu_\aleph \in \Re^M \\ \Sigma_1, \Sigma_2, ..., \Sigma_\aleph \in \Re^{M^2} \end{bmatrix}$

2.3a Define pdf

The complete *pdf* (including latent space) of a sample X_n is :

$$\begin{aligned} P(X_n = x, Z_n = z \mid \theta) &= [\pi_1 G(x \mid \mu_1, \Sigma_1)]^{\delta(z=1)} \times [\pi_2 G(x \mid \mu_2, \Sigma_2)]^{\delta(z=2)} \times ... \times [\pi_\aleph G(x \mid \mu_\aleph, \Sigma_\aleph)]^{\delta(z=\aleph)} \\ &= \pi_z G(x \mid \mu_z, \Sigma_z) \qquad (1) \end{aligned}$$

The incomplete *pdf* (excluding latent space) of a sample X_n is :

$$\begin{aligned} P(X_n = x \mid \theta) &= \textstyle\sum_{z=1}^{\aleph} P(X_n = x, Z_n = z \mid \theta) \\ &= \textstyle\sum_{z=1}^{\aleph} \pi_z G(x \mid \mu_z, \Sigma_z) \qquad (2) \end{aligned}$$

The *pdf* in latent space is :

$$\begin{aligned} P(Z_n = z \mid \theta) &= \textstyle\int_{-\infty}^{+\infty} P(X_n = x, Z_n = z \mid \theta) dx \\ &= \textstyle\int_{-\infty}^{+\infty} \pi_z G(x \mid \mu_z, \Sigma_z) dx \\ &= \pi_z \textstyle\int_{-\infty}^{+\infty} G(x \mid \mu_z, \Sigma_z) dx \\ &= \pi_z \qquad (3) \end{aligned}$$

2.3b E-step

E-step is to calculate probability of class label z_n conditional on data vector x_n and current parameter θ_t. To facilitate the process :

We define

$$\begin{aligned} w_{nz} &= P(Z_n = z \mid X_n = x, \theta_t = \theta) && \textit{which is the prob inside } E_{Z|X,\theta_t}[\ln P(X, Z \mid \theta)] \\ &= \frac{P(X_n = x, Z_n = z \mid \theta_t = \theta)}{P(X_n = x \mid \theta_t = \theta)} \\ &= \frac{\pi_z G(x \mid \mu_z, \Sigma_z)}{\sum_{z'=1}^{\aleph} \pi_{z'} G(x \mid \mu_{z'}, \Sigma_{z'})} && \textit{substitute (1) and (2)} \end{aligned}$$

2.3c M-step

M-step is to update parameter estimation by maximizing expected log likelihood under the measure found in *E-step* :

$$
\begin{aligned}
\theta_{t+1} &= \arg\max_{\theta} E_{Z|X,\theta_t}[\ln P(X,Z\,|\,\theta)] \\
&= \arg\max_{\theta} E_{Z|X,\theta_t}[\ln \textstyle\prod_{n=1}^{N} \pi_{z_n} G(x_n\,|\,\mu_{z_n},\Sigma_{z_n})] && \textit{substitute (1)} \\
&= \arg\max_{\theta} E_{Z|X,\theta_t}[\textstyle\sum_{n=1}^{N} \ln(\pi_{z_n} G(x_n\,|\,\mu_{z_n},\Sigma_{z_n}))] && \textit{log of product} \rightarrow \textit{sum of log} \\
&= \arg\max_{\theta} \textstyle\sum_{n=1}^{N} E_{Z|X,\theta_t}[\ln(\pi_{z_n} G(x_n\,|\,\mu_{z_n},\Sigma_{z_n}))] && \textit{expectation of sum} \rightarrow \textit{sum of expectation} \\
&= \arg\max_{\theta} \textstyle\sum_{n=1}^{N}\sum_{z=1}^{\aleph} w_{nz} \ln(\pi_z G(x_n\,|\,\mu_z,\Sigma_z)) && \textit{expectation random variable } z_n \textit{ replaced by realized value } z \\
&= \arg\max_{\theta} \textstyle\sum_{n=1}^{N}\sum_{z=1}^{\aleph}\left[w_{nz}\left(\ln\pi_z - \frac{1}{2}\mathit{rank}(\Sigma_z)\ln(2\pi) - \frac{1}{2}\ln\det(\Sigma_z) - \frac{1}{2}(x_n-\mu_z)\Sigma_z^{-1}(x_n-\mu_z)^T\right)\right] && (4) \\
&= \textit{linear wrt each parameter in } \{\pi_z\} \\
&\ \textit{quad wrt each parameter in } \{\mu_z, \Sigma_z\}
\end{aligned}
$$

By taking derivative *wrt* one of the mean value μ_z and setting zero, we have :

$$
\begin{aligned}
0 &= \frac{\partial}{\partial \mu_z}\sum_{n=1}^{N}\sum_{z'=1}^{\aleph}\left[w_{nz'}\left(-\frac{1}{2}(x_n-\mu_{z'})\Sigma_{z'}^{-1}(x_n-\mu_{z'})^T\right)\right] \\
0 &= \sum_{n=1}^{N} w_{nz}(x_n-\mu_z) \\
\mu_{z,t+1} &= \frac{\sum_{n=1}^{N} w_{nz} x_n}{\sum_{n=1}^{N} w_{nz}}
\end{aligned}
$$

By taking derivative *wrt* one of the mean value Σ_z and setting zero, without proof we have :

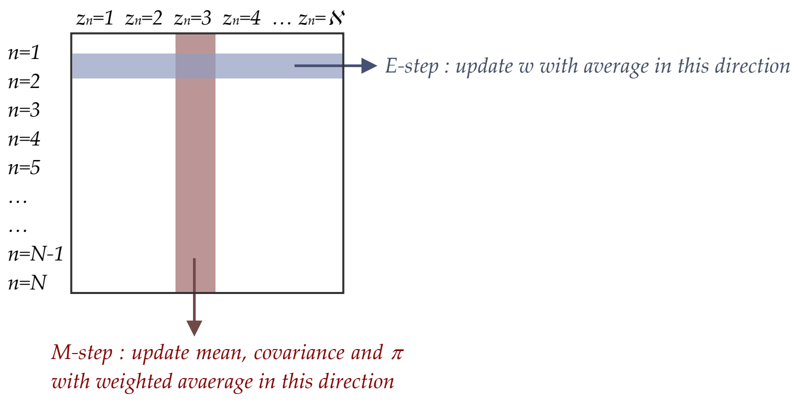
$$
\Sigma_{z,t+1} = \frac{\sum_{n=1}^{N} w_{nz}(x_n-\mu_{z,t})^T(x_n-\mu_{z,t})}{\sum_{n=1}^{N} w_{nz}}
$$

Equation *(4)* is linear *wrt* π_z under constraint $\Sigma\pi_z = 1$, it should be solved using Lagrangian. Without proof, we have :

$$
\pi_{z,t+1} = \frac{\sum_{n=1}^{N} w_{nz}}{\sum_{n=1}^{N}\sum_{z'=1}^{\aleph} w_{nz'}}
$$

2.3d Algorithm using weight matrix

It facilitates the algorithm by defining weight matrix : $w_{n,z} = P(Z_n = z\,|\,X_n = x, \theta_t = \theta)$.



z_n=1 z_n=2 z_n=3 z_n=4 … z_n=ℵ
n=1
n=2
n=3
n=4
n=5
…
…
n=N-1
n=N
E-step : update w with average in this direction
M-step : update mean, covariance and π
with weighted avaerage in this direction

2.4 K Mean Clustering

Given a set of N observations $\{x_n \in \Re^M, n \in [1,N]\}$ with labels $z_n \in [1, \aleph]$, the objective is to minimise error :

$$\min_{\mu}\left[\sum_n \min_z |x_n - \mu_z|^2\right]$$

The implementation can be broken down into assignment step and update step.

assign step : $z_n = \min_z |x_n - \mu_z|^2$ *E-step in EM : assign each data to closest cluster mean*

update step : $\mu_z = \dfrac{\sum_n \delta(z_n = z) x_n}{\sum_n \delta(z_n = z)}$ *M-step in EM : update cluster mean with assigned data*

K mean clustering can be considered as a special case of Gaussian mixture *EM* having :

- minimising Euclidean distance instead of maximising likelihood
- covariance being removed

2.5 K Nearest Neighbour

Given N training vectors having M attributes together with class labels $x_n \in \Re^M$ $y_n \in [1, \aleph]$ $\forall n \in [1,N]$, there are $\aleph$ possible classes. *KNN* classifies test vector $x \in \Re^M$ by considering the K nearest neighbours according to Euclidean distance :

$$\begin{aligned}
n_1 &= \arg\min_{n\in[1,N]} |x - x_n|^2 \\
n_2 &= \arg\min_{\substack{n\in[1,N] \\ n\neq n_1}} |x - x_n|^2 \\
n_3 &= \arg\min_{\substack{n\in[1,N] \\ n\neq n_1 n_2}} |x - x_n|^2 \\
n_4 &= \arg\min_{\substack{n\in[1,N] \\ n\neq n_1 n_2 n_3}} |x - x_n|^2 \\
&\dots \\
y &= \operatorname{mod}\{n_1, n_2, n_3, \dots, n_K\}
\end{aligned}$$

KNN is a non-parametric classifer without explicit training step.

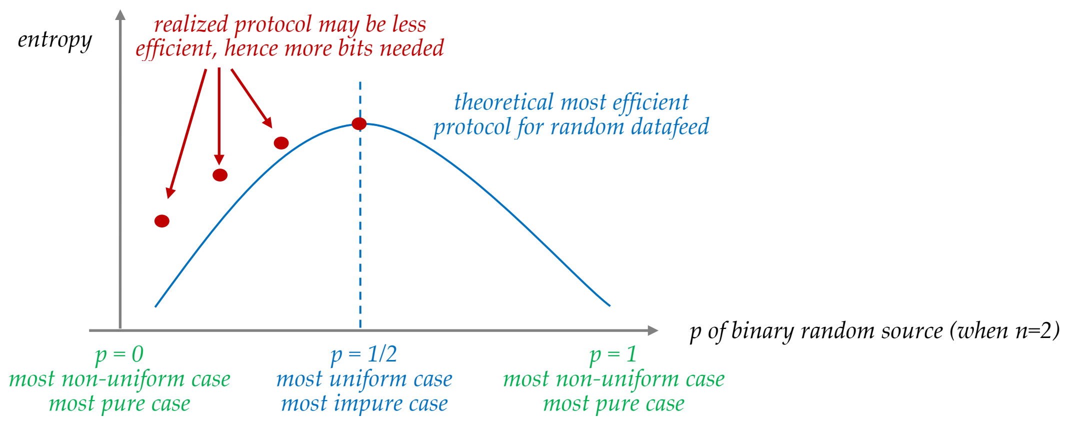
2.6 KL Divergence *(based on Entropy)*

2.6a Define entropy

Deterministic data source contributes no information (zero entropy), a fair coin flip generates exactly 1 bit information, while unfair coin flip generates less than 1 bit information. Entropy is defined in three ways :

- theoretical minimum number of bits used for serializing stream of datafeed
- measurement of information in number of bits produced by random data source *(uniform randomness, bigger entropy)*
- measurement of impurity *within **one** output class* produced by separating hyperplane *(impure classification, bigger entropy)*

$$\begin{aligned} entropy &= \textstyle\sum_n(-p_n\log_2 p_n) \\ &= E_P[-\log_2 p_n] \end{aligned}$$


entropy
realized protocol may be less efficient, hence more bits needed
theoretical most efficient protocol for random datafeed
p of binary random source (when n=2)
p = 0
most non-uniform case
most pure case
p = 1/2
most uniform case
most impure case
p = 1
most non-uniform case
most pure case

Examples

Given a datafeed as a sequence of labels randomly drawn from set *{A,B,C}* with probability *{1/2,1/4,1/4}*, then the optimum protocol should be *{A=0,B=10,C=11}*, thus a stream *{1001100100}* means *{BACAABA}*, entropy and expected length of protocol are :

$$\begin{aligned} entropy &= \frac{1}{2}\log_2 2^1+\frac{1}{4}\log_2 2^2+\frac{1}{4}\log_2 2^2 &&= 1.5\ bits \\ E[length] &= \frac{1}{2}length(A)+\frac{1}{4}length(B)+\frac{1}{4}length(C) &&= 1.5\ bits \end{aligned}$$

As the expected length of proposed protocol equals to the entropy implying that the protocol is optimum. Lets consider another set *{A,B,C,D}* with probability *{1/2,1/4,1/8,1/8}*, the optimum protocol should be {*A=0,B=10,C=110,D=111*}, let's check :

$$\begin{aligned} entropy &= \frac{1}{2}\log_2 2^1+\frac{1}{4}\log_2 2^2+\frac{1}{8}\log_2 2^3+\frac{1}{8}\log_2 2^3 &&= 1.75\ bits \\ E[length] &= \frac{1}{2}length(A)+\frac{1}{4}length(B)+\frac{1}{8}length(C)+\frac{1}{8}length(D) &&= 1.75\ bits \end{aligned}$$

Enthopy gives a theoretical minimum, however in practice, the optimum protocol may not be achievable. Consider the set *{A,B,C,D}* with probability *{5/8,1/8,1/8,1/8}*, the best achievable protocol *{A=0,B=10,C=110,D=111}* is no longer optimum :

$$\begin{aligned} entropy &= \frac{5}{8}\log_2(2^3/5)+\frac{1}{8}\log_2 2^3+\frac{1}{8}\log_2 2^3+\frac{1}{8}\log_2 2^3 &&\sim 1.549\ bits \\ E[length] &= \frac{5}{8}length(A)+\frac{1}{8}length(B)+\frac{1}{8}length(C)+\frac{1}{8}length(D) &&= 1.625\ bits \end{aligned}$$

2.6b Define divergence

KL divergence is defined as the number of extra bits required to serialize sample stream from distribution P using optimal protocol for distribution Q relative to using optimal protocol for distribution P itself. Mathematically, it can be denoted as :

depends on randomness of datafeed (pdf)

depends on protocol definition (#bits)

$$\begin{aligned} D_{KL}(P\,\|\,Q) &= \sum_n(-p_n\log_2 q_n)-\sum_n(-p_n\log_2 p_n) &&= E_P[-\log_2 q_n]-E_P[-\log_2 p_n] \\ &= \sum_n\left(-p_n\log_2\frac{q_n}{p_n}\right) &&= E_P\left[-\log_2\frac{q_n}{p_n}\right] \end{aligned}$$

KL divergence in continuous domain is :

$$D_{KL}(P\,\|\,Q) = -\int P(X)\ln\frac{Q(X)}{P(X)}dX = E_P\left[-\log_2\frac{Q(X)}{P(X)}\right]$$

KL divergence is ***NOT*** the difference in entropy between two distributions :

$$D_{KL}(P\,\|\,Q) \neq E_Q[-\log_2 q_n]-E_P[-\log_2 p_n] \qquad \textit{which can be positive or negative}$$

KL divergence is always non-negative, it is zero when P and Q are the same distribution. Here is the proof by Jensen inequality.

$$\begin{aligned} D_{KL}(P\,\|\,Q) &= \sum_n\left(-p_n\log_2\frac{q_n}{p_n}\right) \\ &\geq -\log_2\left(\sum_n\left(p_n\frac{q_n}{p_n}\right)\right) && \textit{since } -\ln(x) \textit{ is convex, hence } -\ln(\Sigma w_n x_n)\leq-\Sigma w_n\ln(x_n) \\ &= -\log_2\left(\sum_n q_n\right) \\ &= -\log_2 1 \\ &= 0 && \textit{hence when 2 distributions are equivalent, they have zero KL div} \end{aligned}$$

2.6c KL divergence minimization is equivalent to likelihood maximization

KL divergence offers another approach for model parameter estimation. Given a set of N data points $x_n\in\Re^m$ $\forall n\in[1,N]$, we contruct an empirical data distribution $P(X|\{x_n\})$, and define parametric distribution model $Q(X|\theta)$, then we can estimate parameter vector θ by minimizing KL divergence, as it is equivalent to maximization of likelihood $Q(\{x_n\}|\theta)$. Let's prove, the empircal data distribution is simply a summation of delta function locating on data points. In the following, we use continuous version KL divergence.

$$\begin{aligned} \arg\min_\theta D_{KL}(P(X|\{x_n\})\,\|\,Q(X|\theta)) &= \arg\min_\theta E_P\left[-\log_2\frac{Q(X)}{P(X)}\right] \\ &= \arg\max_\theta \int P(X|\{x_n\})\ln\frac{Q(X|\theta)}{P(X|\{x_n\})}dX \\ &= \arg\max_\theta \int P(X|\{x_n\})\ln Q(X|\theta)dX-\int P(X|\{x_n\})\ln P(X|\{x_n\})dX \\ &= \arg\max_\theta \int P(X|\{x_n\})\ln Q(X|\theta)dX \\ &= \arg\max_\theta \int\frac{1}{N}\sum_n\delta(x-x_n)\ln Q(X|\theta)dX && \textit{since } P(X|\{x_n\})=\frac{1}{N}\sum_n\delta(x-x_n) \\ &= \arg\max_\theta \frac{1}{N}\sum_n\ln Q(x_n|\theta) \\ &= \textit{maximum likelihood} \end{aligned}$$

KL divergence is closely related to various machine learning algorithms :

- maximum likelihood
- independent component analysis
- generative adversarial network

2.7 Decision Tree

2.7a Inspection

Decision tree is a tree (not necessarily binary tree) such that each node should either have no child or more than one children. Each node is a separating plane in ***ONE*** particular dimension of the feature space. On receiving an input feature vector, a node will then forward the vector to one of its child (subtree) according to the separating plane of the node (which is simply a thresholding of that particular dimension in the feature space). The process is recursive and terminated when the input vector reaches one of the leaves, each leaf should be labelled with a class, this is the output of the input feature vector. In short, inspection in decision tree is simply a tree traversal with repeated decision making until a terminal node is reached.

2.7b Tree construction

Tree is constructed with :

- each non-terminating node acting as a decision boundary (thresholding in one selected feature)
- each terminating node acting as a class label output
- the tree is parameterized by : hyperplanes at non-terminating nodes and labels at terminating nodes

2.7c Tree training

Training is to find all hyperplanes using a greedy algorithm. Starting from the root recursively search the optimal split for the node of interest. For a *K-class* decision tree, let's consider a particular node of interest :

- select one out of *M* features
- select number of children it has, suppose it is *L*, which can be different from the number of class *K*
- select non-overlapping feature range for each of the *L* children
- calculate impurity (entropy) of each child and
- calculate weighted average of entropy, or equivalently, impurity

```
// suppose K is customizable, while L is fixed at 3
double node::average_entropy(const std::vector<data>& training, const std::string& feature_id, double thres_L, double thres_H)
{
    std::vector<data> vecA; // training data fwd to childA
    std::vector<data> vecB; // training data fwd to childB
    std::vector<data> vecC; // training data fwd to childC
    for(const auto& x:training)
    {
        const auto& feature = x.get(feature_id);
        if      (feature < thres_L) vecA.push_back(x);
        else if (feature < thres_H) vecB.push_back(x);
        else                        vecC.push_back(x);
    }
    double HA = entropy(vecA); double wA = vecA.size()/training.size();
    double HB = entropy(vecB); double wB = vecB.size()/training.size();
    double HC = entropy(vecC); double wC = vecC.size()/training.size();
    return wA*HA + wB*HB + wC*HC;
}

double node::entropy(const std::vector<data>& vec)
{
    std::vector<int> counts(K,0); // vector of K zeros
    for(const auto& x:vec) ++counts[x.label];

    double output = 0;
    for(const auto& n:counts)
    {
        double p = n/vec.size();
        output += -p * log2(p);
    }
    return output;
}
```

We have to search `feature_id` together with appropriate `thresholds` to minimize `average_entropy`. Mathematically, we have :

$$\min_{\substack{m\in[1,M]\\ threshold}} \frac{N_A}{N} entropy_A(m, threshold) + \frac{N_B}{N} entropy_B(m, threshold) + \frac{N_C}{N} entropy_C(m, threshold) \qquad where\ \ N = N_A + N_B + N_C$$

$$entropy_A(m, threshold) = -\sum_k \frac{N_{Ak}}{N_A}\left(\log_2 \frac{N_{Ak}}{N_A}\right) \quad and\ \ N_{Ak} = num\ of\ data\ labelled\ k\ fwd\ to\ childA \qquad where\ \ N_A = \sum_k N_{Ak}$$

$$entropy_B(m, threshold) = -\sum_k \frac{N_{Bk}}{N_B}\left(\log_2 \frac{N_{Bk}}{N_B}\right) \quad and\ \ N_{Bk} = num\ of\ data\ labelled\ k\ fwd\ to\ childB \qquad where\ \ N_B = \sum_k N_{Bk}$$

$$entropy_C(m, threshold) = -\sum_k \frac{N_{Ck}}{N_C}\left(\log_2 \frac{N_{Ck}}{N_C}\right) \quad and\ \ N_{Ck} = num\ of\ data\ labelled\ k\ fwd\ to\ childC \qquad where\ \ N_C = \sum_k N_{Ck}$$

3.1 Kalman Filter

Reference "How a Kalman filter works, in pictures", a Bzarg blog.

Given Markovian random process with Gaussian noise (where n is time index, F_n and B_n are time-dependent known-dynamics) :

$$X_n = \underbrace{F_n}_{M\times M}\underbrace{X_{n-1}}_{M\times 1} + \underbrace{B_n}_{M\times 1} + \underbrace{\varepsilon(0,Q_n)}_{M\times 1} \qquad \textit{where} \quad \underbrace{Q_n}_{M\times M} = E(\varepsilon\varepsilon^T)$$

where X_{n-1} is random. Suppose we keep tracing X_n since beginning of time, we can perform prediction using maximum likelihood :

$$\begin{aligned}\hat{X}_n &= \arg\max_X p(X_n = X \mid X_{n-1} = \hat{X}_{n-1}) \\ &= \arg\max_X \frac{1}{\sqrt{(2\pi)^M \det(Q_n)}}\exp[-(X-(F_n\hat{X}_{n-1}+B_n))^T Q_n^{-1}(X-(F_n\hat{X}_{n-1}+B_n))/2] \\ &= F_n\hat{X}_{n-1}+B_n \qquad \textit{since ML estimation of Gaussian random variable is just its mean vector}\end{aligned}$$

This estimation is theoretically unbiased if the previous estimation is unbiased :

$$\begin{aligned}E[X_n-\hat{X}_n] &= E[X_n-(F_n\hat{X}_{n-1}+B_n)] \\ &= E[F_nX_{n-1}+B_n+\varepsilon(0,Q_n)-(F_n\hat{X}_{n-1}+B_n)] \\ &= E[F_n(X_{n-1}-\hat{X}_{n-1})]+E[\varepsilon(0,Q_n)] \\ &= F_nE[X_{n-1}-\hat{X}_{n-1}] \\ &= 0 \qquad \textit{since } E[X_{n-1}-\hat{X}_{n-1}]=0\end{aligned}$$

but the variance of estimation error keeps increasing, as variance is accumulated through time :

$$\begin{aligned}V[X_n-\hat{X}_n] &= V[X_n-(F_n\hat{X}_{n-1}+B_n)] \\ &= V[F_nX_{n-1}+B_n+\varepsilon(0,Q_n)-(F_n\hat{X}_{n-1}+B_n)] \\ &= V[F_n(X_{n-1}-\hat{X}_{n-1})]+V[\varepsilon(0,Q_n)] \\ &= V[F_n(X_{n-1}-\hat{X}_{n-1})]+Q_n \\ &= E[F_n(X_{n-1}-\hat{X}_{n-1})(F_n(X_{n-1}-\hat{X}_{n-1}))^T]+Q_n \\ &= F_nE[(X_{n-1}-\hat{X}_{n-1})(X_{n-1}-\hat{X}_{n-1})^T]F_n^T+Q_n\end{aligned}$$

For convenience, we declare P_n as the variance of error, hence we have :

$$P_n = F_nP_{n-1}F_n^T+Q_n \qquad \textit{accumulation of variance through time}$$

If we are given an observation in measurement domain right after each prediction step, we can then update our current estimation for the sake of the next prediction, again this is done using maximum likelihood, the conversion from state domain to measurement domain is done by a time-dependent transformation H_n, where Z_n is the measurement reading, again there is measurement noise :

$$Z_n = \underbrace{\varepsilon'(Z_n^{read},R_n)}_{M'\times 1} \qquad \textit{where} \quad \underbrace{R_n}_{M'\times M'} = E(\varepsilon'\varepsilon'^T)$$

Our prediction in measurement domain is H_nX_n, mean and variance error of our estimation in measurement space is :

$$\begin{aligned}E[H_n(X_n-\hat{X}_n)] &= H_nE[X_n-\hat{X}_n] \quad = \quad 0 \\ V[H_n(X_n-\hat{X}_n)] &= E[H_n(X_n-\hat{X}_n)(H_n(X_n-\hat{X}_n))^T]-E[H_n(X_n-\hat{X}_n)]E[H_n(X_n-\hat{X}_n)]^T \\ &= E[H_n(X_n-\hat{X}_n)(X_n-\hat{X}_n)^T H_n^T]-H_nE[X_n-\hat{X}_n](H_nE[X_n-\hat{X}_n])^T \\ &= H_nE[(X_n-\hat{X}_n)(X_n-\hat{X}_n)^T]H_n^T \\ &= H_nP_nH_n^T\end{aligned}$$

In this update step, we maximize the joint probability between :

(1) $p(Z_n = Z \mid X_{n-1}=\hat{X}_{n-1}) = \varepsilon(H_n\hat{X}_n, H_nP_nH_n^T)$ *distribution of prediction in measurement domain*

(2) $p(Z_n = Z) = \varepsilon(Z_n^{read},R_n)$ *distribution of observation in measurement domain*

The joint likelihood of prediction and observation in measurement domain is a product of two Gaussians, which is also a Gaussian.

$$\begin{aligned}
\hat{Z}_n &= \arg\max_Z \varepsilon_Z(H_n\hat{X}_n, H_nP_nH_n^T)\varepsilon_Z(Z_n^{read}, R_n) \\
&= \arg\max_Z \varepsilon_Z\begin{pmatrix} mean = W_AH_n\hat{X}_n + W_BZ_n^{read} \\ \text{var} = ((H_nP_nH_n^T)^{-1} + R_n^{-1})^{-1}\end{pmatrix}
\end{aligned}$$

where $\begin{bmatrix} W_A \\ W_B \end{bmatrix} = \begin{bmatrix} ((H_nP_nH_n^T)^{-1} + R_n^{-1})^{-1}(H_nP_nH_n^T)^{-1} \\ ((H_nP_nH_n^T)^{-1} + R_n^{-1})^{-1}R_n^{-1} \end{bmatrix}$ *These matrices look like those in AX=B doc.*

$= \begin{bmatrix} R_n(H_nP_nH_n^T + R_n)^{-1} \\ H_nP_nH_n^T(H_nP_nH_n^T + R_n)^{-1} \end{bmatrix}$ *refer to product of Gaussians*

$= \begin{bmatrix} I-K \\ K \end{bmatrix}$ *refer to product of Gaussians, define gain vector* $K = W_B$

Since

$$\begin{aligned}
K &= W_B \\
&= I - W_A \\
&= I - ((H_nP_nH_n^T)^{-1} + R_n^{-1})^{-1}(H_nP_nH_n^T)^{-1} \\
(I-K)(H_nP_nH_n^T) &= ((H_nP_nH_n^T)^{-1} + R_n^{-1})^{-1} \qquad \textit{equation \#}
\end{aligned}$$

For machine learning, we prefer to express as updating equation with Kalman factor as step size (instead of weighted average) :

$$\begin{aligned}
\hat{Z}_n &= W_AH_n\hat{X}_n + W_BZ_n^{read} && \textit{MLE as weighted sum} \\
&= (I-K)H_n\hat{X}_n + KZ_n^{read} \\
&= H_n\hat{X}_n + K(Z_n^{read} - H_n\hat{X}_n) && \textit{MLE as an updated equation} \\
\tilde{X}_n &= H_n^{-1}\hat{Z}_n \\
&= \hat{X}_n + K'(Z_n^{read} - H_n\hat{X}_n) && \textit{where } K' = P_nH_n^T(H_nP_nH_n^T + R_n)^{-1}
\end{aligned}$$

If we use updated estimate instead of the original estimate, accumulation of variance error in measurement domain becomes :

$$\begin{aligned}
H_n\tilde{P}_nH_n^T &= ((H_nP_nH_n^T)^{-1} + R_n^{-1})^{-1} \\
&= (I-K)(H_nP_nH_n^T) && \textit{using equation \#} \\
&= H_nP_nH_n^T - K(0 - H_nP_nH_n^T) \\
\tilde{P}_nH_n^T &= P_nH_n^T - K'(0 - H_nP_nH_n^T) \\
\tilde{P}_n &= P_n - K'(0 - H_nP_n) && \textit{variance estimation as an updated equation}
\end{aligned}$$

Please note that $\hat{X}_n \neq \tilde{X}_n$, though both are ML estimations, they are best estimates given different informations. With updating step, the prediction step is based on updated estimate and becomes :

$$\begin{aligned}
& \hat{X}_n = \arg\max_X p(X_n = X \mid X_{n-1} = \hat{X}_{n-1}) && \textit{prediction without updating} \\
\Rightarrow\; & \hat{X}_n = \arg\max_X p(X_n = X \mid X_{n-1} = \tilde{X}_{n-1}) && \textit{prediction with updating}
\end{aligned}$$

Difference in roles of *Q* in prediction and *R* in updating : in prediction, we add *X* with *ε*, while in updating, we join *HX* with *ε (Z,R).*

$$\begin{aligned}
X_n &= F_nX_{n-1} + B_n + \varepsilon(0, Q_n) && \textit{which results in sum of variances} \\
H_nX_{n-1} &\;\; vs \;\; \varepsilon(Z_n^{read}, R_n) && \textit{which results in inverse variance weighs}
\end{aligned}$$

Kalman filter appendix – comparison with Recursive Least Square (RLS)

We can simulate RLS with a Kalman filter by removing predict step of Kalman filter and setting the dimension of measurement space to *1*. Here is the correspondence between the variables in RLS and Kalman filter :

	Kalman filter	Recursive least square	
predict step :	$\hat{X}_n = F_n\tilde{X}_{n-1}+B_n$	*no*	
	$P_n = F_nP_{n-1}F_n^T+Q_n$	*no*	
update step :	H_n $M'\times M$	$A^{(n)}{}_n$ $1\times M$	*measurement trasnsformation*
	Z_n $M'\times 1$	$b^{(n)}{}_n$ 1×1	*measurement space (with M′=1)*
	R_n $M'\times M'$	$\sigma_n{}^2$ 1×1	*measurement error*
	$K' = P_nH_n^T(H_nP_nH_n^T+R_n)^{-1}$	$K' = \dfrac{P_{n-1}A_n^{(n)T}}{A_n^{(n)}P_{n-1}A_n^{(n)T}+\sigma_n^2}$	
state estimate	$\tilde{X}_n = \hat{X}_n+K'(Z_n^{read}-H_n\hat{X}_{n\vert})$	$\tilde{X}_n = X_{n-1}+K'(b_n^{(n)}-A_n^{(n)}X_{n-1})$	
state covariance	$\tilde{P}_n = P_n+K'(0-H_nP_n)$	$\tilde{P}_n = P_{n-1}+K'(0-A_n^{(n)}P_{n-1})$	

Kalman filter appendix – multiplication of multivariate Gaussians

When two Gaussians sum together, it becomes a Gaussian as convolution of Gaussians gives a Gaussian. When two Gaussians join together (i.e. joint probability in maximum likelihood), it also becomes a Gaussian as product of Gaussians gives a Gaussian. These two results are consistent as we know that Fourier transform of a Gaussian is a Gaussian, while convolution in time is equivalent to product in frequency domain and vice versa. Please make a summary here :

$$\begin{aligned}
p(X_1=x) &= \varepsilon(\mu_1,\sigma_1) \\
p(X_2=x) &= \varepsilon(\mu_2,\sigma_2) \\
p(X_1+X_2=x) &= \varepsilon(\mu_1,\sigma_1)\otimes\varepsilon(\mu_2,\sigma_2) && \textit{sum of Gaussians} \\
&= \varepsilon(\mu_1+\mu_2,\sqrt{\sigma_1{}^2+\sigma_2{}^2}) \\
p(X_1=x)p(X_2=x) &= \varepsilon(\mu_1,\sigma_1)\times\varepsilon(\mu_2,\sigma_2) && \textit{joint of Gaussians} \\
&= \varepsilon(w_1\mu_1+w_2\mu_2,(\sigma_1{}^{-2}+\sigma_2{}^{-2})^{-1}) \\
&= \varepsilon(w_1\mu_1+w_2\mu_2,w_1\sigma_1{}^2=w_2\sigma_2{}^2) \\
\textit{where}\quad w_1 &= \sigma_2{}^2/(\sigma_1{}^2+\sigma_2{}^2) \\
w_2 &= \sigma_1{}^2/(\sigma_1{}^2+\sigma_2{}^2)
\end{aligned}$$

<u>Proof of product of Gaussians</u>

$$\begin{aligned}
p_k(x) &= \frac{1}{\sqrt{(2\pi)^M\det(P_k)}}\exp(-\frac{1}{2}(X-E_k)^TP_k^{-1}(X-E_k)) && \textit{where } k=1,2 \\
q(x) &= p_1(x)p_2(x)/s_1 && \textit{where } s_1=\int_{-\infty}^{\infty}p_1(x)p_2(x)dx \ \textit{(by note 1)} \\
&= \exp(-\frac{1}{2}(X-E_1)^TP_1^{-1}(X-E_1)-\frac{1}{2}(X-E_2)^TP_2^{-1}(X-E_2))\Big/s_2 && \textit{where } s_2=\frac{s_1}{(2\pi)^M\sqrt{\det(P_1)\det(P_2)}} \\
&= \exp(-\frac{1}{2}(X^T(P_1^{-1}+P_2^{-1})X-X^T(P_1^{-1}E_1+P_2^{-1}E_2)-(E_1^TP_1^{-1}+E_2^TP_2^{-1})X)\Big/s_3 && \textit{where } s_3=\frac{s_2}{\exp(-\frac{1}{2}(E_1^TP_1^{-1}E_1+E_2^TP_2^{-1}E_2))} \\
&= \exp(-\frac{1}{2}(X^T\underbrace{(P_1^{-1}+P_2^{-1})}_{A}X-X^T\underbrace{(P_1^{-1}E_1+P_2^{-1}E_2)}_{B}-\underbrace{(P_1^{-1}E_1+P_2^{-1}E_2)}_{B}{}^TX)\Big/s_3 && P_1\textit{ and }P_2\textit{ are symmetric} \\
&= \exp(-\frac{1}{2}(X-A^{-1}B)^TA(X-A^{-1}B))\Big/s_4 && \textit{where } s_4=s_3\exp(-\frac{1}{2}(B^TA^{-1}B))\ \textit{(by note 2)} \\
&= \textit{Gaussian}(E=A^{-1}B,P=A^{-1})
\end{aligned}$$

The new mean vector and new covariance matrix are :

$$\begin{aligned}
E &= (P_1^{-1}+P_2^{-1})^{-1}(P_1^{-1}E_1+P_2^{-1}E_2) && \textit{define } K=(P_1^{-1}+P_2^{-1})^{-1}P_2^{-1} \\
&= (I-K)E_1+KE_2 && \textit{weighted average of two original means} \\
P &= (P_1^{-1}+P_2^{-1})^{-1} && \textit{which is like two resistors in parallel}
\end{aligned}$$

The new mean is weighted average of the two constituents' mean. The covariance matrix evolves like adding one new resistor P_2 in parallel with the original circuit P_1. Please note that $K = W_B$ can be represented in 4 different ways :

$$\begin{aligned}
K &= (P_1^{-1} + P_2^{-1})^{-1} P_2^{-1} && (1) \\
&= I - (P_1^{-1} + P_2^{-1})^{-1} P_1^{-1} && (2) \\
&= P_1 (P_1 + P_2)^{-1} && (3) \\
&= I - P_2 (P_1 + P_2)^{-1} && (4)
\end{aligned}$$

<u>*Note 1*</u>
We omit all the constant terms and combine them into s because :

given two pdf with unity integral	$\int_{-\infty}^{\infty} p_1(x)dx = 1$ *and* $\int_{-\infty}^{\infty} p_2(x)dx = 1$
integral of their product is non unity	$\int_{-\infty}^{\infty} p_1(x) p_2(x) dx \neq 1$
hence product of pdf should be defined as	$q(x) = \dfrac{p_1(x)p_2(x)}{\int_{-\infty}^{\infty} p_1(x)p_2(x)dx}$ *so that* $\int_{-\infty}^{\infty} q(x)dx = 1$

<u>*Note 2*</u>
Given symmetric square matrix A, completing square in matrix form is :

$$\begin{aligned}
& X^T AX - X^T B - B^T X \\
= \; & X^T AX - X^T B - B^T X + B^T A^{-1} B - B^T A^{-1} B \\
= \; & X^T \sqrt{A}\sqrt{A} X - X^T B - B^T X + B^T \sqrt{A}^{-1} \sqrt{A}^{-1} B - B^T A^{-1} B && \text{as we define } A = \sqrt{A}\sqrt{A}\text{, then } A^{-1} = (\sqrt{A}\sqrt{A})^{-1} = \sqrt{A}^{-1}\sqrt{A}^{-1} \\
= \; & X^T \sqrt{A}^T \sqrt{A} X - X^T B - B^T X + B^T \sqrt{A}^{-1T} \sqrt{A}^{-1} B - B^T A^{-1} B && \text{besides, we pick a symmetic } \sqrt{A}^T = \sqrt{A} \\
= \; & (\sqrt{A}X)^T \sqrt{A} X - X^T B - B^T X + (\sqrt{A}^{-1} B)^T \sqrt{A}^{-1} B - B^T A^{-1} B \\
= \; & (\sqrt{A}X - \sqrt{A}^{-1} B)^T (\sqrt{A}X - \sqrt{A}^{-1} B) - B^T A^{-1} B && \text{using property 2a and 2b below} \\
= \; & (\sqrt{A}(X - A^{-1}B))^T \sqrt{A}(X - A^{-1}B) - B^T A^{-1} B \\
= \; & (X - A^{-1}B)^T \sqrt{A}^T \sqrt{A}(X - A^{-1}B) - B^T A^{-1} B && \text{property 2a : } (\sqrt{A}X)^T \sqrt{A}^{-1} B = X^T \sqrt{A}^T \sqrt{A}^{-1} B = X^T B \\
= \; & (X - A^{-1}B)^T A (X - A^{-1}B) - B^T A^{-1} B && \text{property 2b : } (\sqrt{A}^{-1} B)^T \sqrt{A} X = B^T \sqrt{A}^{-1T} \sqrt{A} X = B^T X
\end{aligned}$$

recall : transpose of inverse equals to inverse of tranpose

<u>*Note 3*</u>
More about matrix inverse :

$$\begin{aligned}
(A^{-1} + B^{-1})^{-1} &= AA^{-1}(A^{-1} + B^{-1})^{-1} B^{-1} B \\
&= A(B(A^{-1} + B^{-1})A)^{-1} B \\
&= A(BA^{-1}A + BB^{-1}A)^{-1} B \\
&= A(A + B)^{-1} B \\
(A^{-1} + B^{-1})^{-1} &= B(A + B)^{-1} A && \text{follow similar logic}
\end{aligned}$$

Besides, weights by <u>inverse covariance</u> can be written as weights by <u>covariance</u> :

$$\begin{aligned}
W_A &= (\Sigma_A^{-1} + \Sigma_B^{-1})^{-1} \Sigma_A^{-1} \\
&= \Sigma_B (\Sigma_A + \Sigma_B)^{-1} \Sigma_A \Sigma_A^{-1} \\
&= \Sigma_B (\Sigma_A + \Sigma_B)^{-1} && \text{property 3d} \\
W_B &= (\Sigma_A^{-1} + \Sigma_B^{-1})^{-1} \Sigma_B^{-1} \\
&= \Sigma_A (\Sigma_A + \Sigma_B)^{-1} \Sigma_B \Sigma_B^{-1} \\
&= \Sigma_A (\Sigma_A + \Sigma_B)^{-1} && \text{property 3c} \\
W_A &= (\Sigma_A^{-1} + \Sigma_B^{-1})^{-1} \Sigma_A^{-1} \\
&= (\Sigma_A^{-1} + \Sigma_B^{-1})^{-1} (\Sigma_A^{-1} + \Sigma_B^{-1} - \Sigma_B^{-1}) \\
&= (\Sigma_A^{-1} + \Sigma_B^{-1})^{-1} (\Sigma_A^{-1} + \Sigma_B^{-1}) - (\Sigma_A^{-1} + \Sigma_B^{-1})^{-1} \Sigma_B^{-1} \\
&= I - (\Sigma_A^{-1} + \Sigma_B^{-1})^{-1} \Sigma_B^{-1} \\
&= I - W_B
\end{aligned}$$

3.2 Principal Component Analysis

Given a $N\times M$ row-wise data matrix X, its mean row-vector μ and covariance matix Σ respectively are :

$$\begin{aligned}
\mu &= L^T X / N && \textit{where L is a all-one column-vector} \\
\Sigma &= (X-U)^T (X-U)/N && \textit{where } U = [\mu; \mu; \mu; \ldots\mu] \textit{ is a stack of N } \mu \textit{ forming a } N\times M \textit{ matrix}
\end{aligned}$$

Principal component analysis finds the projection column-vector w so that variance of transformed data is maximized, transformed mean vector and covariance matrix (both are 1×1) can be written in terms of the original mean vector and covariance matrix :

$$\begin{aligned}
\mu' &= L^T (Xw)/N && \textit{we denote } \underline{\textit{data x as row-vector}} \textit{ and } \underline{\textit{parameter w as column-vector}} \\
&= \mu w \\
\Sigma' &= (Xw-U')^T (Xw-U')/N && \textit{where } U' = [\mu'; \mu'; \mu'; \ldots\mu'] \textit{ is a stack of N } \mu' \textit{ forming a } N\times M \textit{ matrix} \\
&= (Xw-Uw)^T (Xw-Uw)/N \\
&= w^T (X-U)^T (X-U) w/N \\
&= w^T \Sigma w && \textit{note that } \mu \textit{ is } 1\times M,\ \Sigma \textit{ is } M\times M, \textit{ while both } \mu' \textit{ and } \Sigma' \textit{ are } 1\times 1
\end{aligned}$$

In order to maximize Σ' respect to column-vector w under the constraint $w^Tw = 1$, we set up a Lagrangian, take the gradient and set it zero (gradient is also a column-vector like w), we have :

$$\begin{aligned}
L &= \Sigma' - \lambda(w^T w - 1) \\
&= w^T \Sigma w - \lambda(w^T w - 1) \\
\nabla_w L &= 2\Sigma w - 2\lambda w \quad = \quad 0 \\
\Sigma w &= \lambda w
\end{aligned}$$

With eigen decomposition, eigen vector gives the desired projection vector, while eigen value gives the Lagrange multiplier. Yet in practice, we wish to obtain all the principal components W, which has column-vectors representing orthogonal basis. As covariance Σ' is no longer a scalar, we need some way to define a scalar objective : the answer is trace. Trace of transformed covariance is sum of diagonal variances, while ignoring off-diagonal covariances :

$$\begin{aligned}
tr(\Sigma') &= \textstyle\sum_{n=1}^N (\Sigma')_{nn} \\
\textit{and}\quad \Sigma' &= W^T \Sigma W && \textit{where W is column-wise projection basis, both W and } \Sigma' \textit{ are } M\times M
\end{aligned}$$

We are going to prove two properties (1) eigen basis can diagonalize covariance matrix, i.e. off-diagonal covariances of transformed data are zeros, (2) trace of covariance $tr(\Sigma') = tr(W^T\Sigma W)$ is maximized when <u>columns of W</u> are set equal to eigen vectors of original covariance Σ. First of all, the proof for diagonalization is straight forward, suppose the eigen decomposition of Σ is :

$$\begin{aligned}
\Sigma &= Q^T \Lambda Q && \textit{where Q is row-wise eigen basis, hence } QQ^T = I \textit{ and } \Lambda_{nm} = 0\ \forall n=m \\
\Rightarrow \quad \Sigma' &= W^T \Sigma W \\
&= W^T Q^T \Lambda Q W \\
&= (QW)^T \Lambda (QW) \\
\Sigma'|_{W=Q^T} &= \Lambda && \textit{when columns of W are eigen vectors of } \Sigma, \textit{ i.e. } W = Q^T \textit{ and } QW = QQ^T = I
\end{aligned}$$

Secondly, we have to show that trace of $\Sigma'|_{W=Q^T}$ is greater than trace of Σ' for all other W. It involves the trace trick :

$$\begin{aligned}
tr(\Sigma') &= tr((QW)^T \Lambda (QW)) \\
&= tr(\Lambda (QW)(QW)^T) && \textit{trace is invariant to cyclic permutation} \\
&\le tr(\Lambda) tr((QW)(QW)^T) && \textit{prove it later using Cauchy Schwarz inequality (see remark after LDA)} \\
&= \textstyle\sum_{n=1}^N \Lambda_{nn} tr(QWW^TQ^T) \\
&= \textstyle\sum_{n=1}^N \Lambda_{nn} tr(WW^TQ^TQ) && \textit{trace is invariant to cyclic permutation} \\
&= \textstyle\sum_{n=1}^N \Lambda_{nn} tr(WW^T) && \textit{since } QQ^T = Q^TQ = I \textit{ as it is eigen basis} \\
&= \textstyle\sum_{n=1}^N \Lambda_{nn} tr(I) && \textit{since } WW^T = W^TW = I \textit{ as it is projection basis (constraint for optimization)} \\
&= \textstyle\sum_{n=1}^N \Lambda_{nn} \times rank
\end{aligned}$$

Therefore, our objective function is bounded above by the sum of eigen values scaled by rank of data, equality holds when $QW = I$, that is when $W = Q^T$ (columns of W equal to rows of Q). In short eigen basis is the projection basis that maximise covariance trace.

3.3 Linear Discriminant Analysis

If logistic regression is regarded as a supervised linear classifier, then linear discriminant analysis can be regarded as unsupervised linear classifier, the former is derived from maximum likelihood, while the latter is derived from between-class against within-class variance ratio, which can be considered as an extension from *PCA*. Lets consider two-classes problem, given a $N_1 \times M$ row-wise data matrix X_1 for class *1* and a $N_2 \times M$ row-wise data matrix X_2 for class *2*, the mean row-vectors $\mu_1\ \mu_2$ and covariance matrices $\Sigma_1\ \Sigma_2$ are :

$$\begin{aligned}
\mu_k &= L_k^T X_k / N_k && \textit{where } L_k \textit{ is a all-one column-vector} \\
\Sigma_k &= (X_k - U_k)^T (X_k - U_k) / N_k && \textit{where } U_k = [\mu_k; \mu_k; \mu_k; \ldots \mu_k] \textit{ is a stack of } N_k\ \mu_k \textit{ forming a } N_k \times M \textit{ matrix} \\
k &= 1{,}2
\end{aligned}$$

Between class and within class covariance matrix are defined as the following (both matrices are $M \times M$) :

$$\begin{aligned}
\Sigma_{between} &= \textstyle\sum_{k=1}^{2} (\mu_k - \mu)^T (\mu_k - \mu) && \textit{where } \mu = LX/N \textit{ and } X = [X_1; X_2],\ N = N_1 + N_2 \\
\Sigma_{within} &= \textstyle\sum_{k=1}^{2} (X_k - U_k)^T (X_k - U_k) / N_k \\
&= \textstyle\sum_{k=1}^{2} \Sigma_k
\end{aligned}$$

Linear discriminant analysis finds the projection column-vector *w* such that the between-class and the within-class variance ratio of transformed data is maximized, transformed between-class covariance and transformed within-class covariance are both scalar and they can be written in terms of the original between-class covariance and within-class covariance :

$$\begin{aligned}
\Sigma'_{between} &= \textstyle\sum_{k=1}^{2} (U_k w - Uw)^T (U_k w - Uw) \\
&= \textstyle\sum_{k=1}^{2} w^T (U_k - U)^T (U_k - U) w \\
&= w^T \Sigma_{between} w \\
\Sigma'_{within} &= \textstyle\sum_{k=1}^{2} (X_k w - U_k w)^T (X_k w - U_k w) / N_k \\
&= \textstyle\sum_{k=1}^{2} w^T (X_k - U_k)^T (X_k - U_k) w / N_k \\
&= w^T \Sigma_{within} w
\end{aligned}$$

Both covariance matrices above are scalar as *w* is column-vector. Objective of *LDA* is to maximize the following ratio with $w^T w = 1$:

$$\begin{aligned}
\max_w \frac{\Sigma'_{between}}{\Sigma'_{within}} &= \max_w \frac{w^T \Sigma_{between} w}{w^T \Sigma_{within} w} && st\ w^T w = 1 \\
&= \max_w \frac{(\alpha w)^T \Sigma_{between} (\alpha w)}{(\alpha w)^T \Sigma_{within} (\alpha w)} && st\ w^T w = 1 && \textit{(as objective is invariant to scaling in } w\textit{)} \\
&= \max_w \frac{(\alpha w)^T \Sigma_{between} (\alpha w)}{w^T (\alpha^T \Sigma_{within} \alpha) w} && st\ w^T w = 1 \\
&= \max_w (\alpha w)^T \Sigma_{between} \underbrace{(\alpha w)}_{w'} && st\ w^T w = 1 && \textit{(let's pick } \alpha \textit{ such that } \alpha^T \Sigma_{within} \alpha = I \textit{, then apply } w^T w = 1\textit{)} \\
&= \max_{w'} w'^T \Sigma_{between} w' && st\ (\alpha^{-1} w')^T (\alpha^{-1} w) = 1 && \textit{(put } w' = \alpha w \textit{, hence } w = \alpha^{-1} w'\textit{)} \\
&= \max_{w'} w'^T \Sigma_{between} w' && st\ w'^T \Sigma_{within} w = 1 && \textit{(since } (\alpha^{-1} w')^T (\alpha^{-1} w) = w'^T (\alpha^{-T} \alpha^{-1}) w = w'^T \Sigma_{within} w\textit{)}
\end{aligned}$$

Changing notation *w'* back to *w*, we then set up a Lagrangian, take the gradient and set it zero, we have :

$$\begin{aligned}
L &= w^T \Sigma_{between} w - \lambda (w^T \Sigma_{within} w - 1) \\
\nabla_w L &= 2\Sigma_{between} w - 2\lambda \Sigma_{within} w = 0 \\
\Sigma_{between} w &= \lambda \Sigma_{within} w \\
(\Sigma_{within})^{-1} \Sigma_{between} w &= \lambda w
\end{aligned}$$

As matrix $(\Sigma_{within})^{-1}\Sigma_{between}$ is not necessarily symmetric (please see remark for the reason), the above equation isn't an ordinary eigen value decomposition, instead it is known as a generalized eigen problem. In order to deal with ordinary eigen value decomposition, we need to make further simplication by taking square root of $\Sigma_{between}$, that is $\Sigma_{between} = (\Sigma_{between})^{1/2}(\Sigma_{between})^{1/2}$.

$$\begin{aligned}
(\Sigma_{within})^{-1}\Sigma_{between}^{1/2}\Sigma_{between}^{1/2}w &= \lambda w \\
\Sigma_{between}^{1/2}(\Sigma_{within})^{-1}\Sigma_{between}^{1/2}\Sigma_{between}^{1/2}w &= \lambda\Sigma_{between}^{1/2}w \\
\underbrace{\Sigma_{between}^{1/2}(\Sigma_{within})^{-1}\Sigma_{between}^{1/2}}_{A}w' &= \lambda w' \qquad \textit{by putting } w' = (\Sigma_{between})^{1/2}w
\end{aligned}$$

Therefore, *LDA* projection can be found by eigen decomposition of matrix A and apply inverse transform $w = (\Sigma_{between})^{-1/2}w'$ on eigen vector w', besides square root of matrix can be found by *SVD*, i.e. if $\Sigma_{between} = U\Lambda U^T$, then we have $(\Sigma_{between})^{1/2} = U\Lambda^{1/2}U^T$. Finally above *LDA* can easily be extended to multiclass by having $K > 2$. For case $K = 2$, most researchers prefer to define between-class covariance as the following :

$$\Sigma_{betweenII} = (\mu_1 - \mu_2)^T(\mu_1 - \mu_2)$$

which is equivalent to the original definition, lets prove :

$$\begin{aligned}
\Sigma_{between} &= \textstyle\sum_{k=1}^{2}(\mu_k - \mu)^T(\mu_k - \mu) \\
&= (\mu_1 - (r_1\mu_1 + r_2\mu_2))^T(\mu_1 - (r_1\mu_1 + r_2\mu_2)) + (\mu_2 - (r_1\mu_1 + r_2\mu_2))^T(\mu_2 - (r_1\mu_1 + r_2\mu_2)) \\
&= (r_2\mu_1 - r_2\mu_2)^T(r_2\mu_1 - r_2\mu_2) + (r_1\mu_2 - r_1\mu_1)^T(r_1\mu_2 - r_1\mu_1) \\
&= r_2^2(\mu_1 - \mu_2)^T(\mu_1 - \mu_2) + r_1^2(\mu_2 - \mu_1)^T(\mu_2 - \mu_1) \\
&= (r_1^2 + r_2^2)(\mu_1 - \mu_2)^T(\mu_1 - \mu_2) \\
&\propto \Sigma_{betweenII} \\
\textit{where} \quad r_{1,2} &= N_{1,2}/(N_1 + N_2)
\end{aligned}$$

Remark for PCA

Proof of $tr(DA) \le tr(D)tr(A)$ given D is diagonal.

$$\begin{aligned}
(DA)_{nm} &= (D_{nm}A_{nm}) && \textit{if D is doagonal} \\
tr(DA) &= \textstyle\sum_{n=1}^{N}(DA)_{nn} \\
(tr(DA))^2 &= (\textstyle\sum_{n=1}^{N}(DA)_{nn})^2 \\
&\le (\textstyle\sum_{n=1}^{N}D_{nn}^2)(\textstyle\sum_{n=1}^{N}A_{nn}^2) && \textit{Cauchy Schwarz states } (\vec{x}\vec{y})^2 \le |\vec{x}|^2|\vec{y}|^2 \textit{ or equivalently } (\textstyle\sum_n x_n y_n)^2 \le (\textstyle\sum_n x_n^2)(\textstyle\sum_n y_n^2) \\
&\le (\textstyle\sum_{n=1}^{N}D_{nn})^2(\textstyle\sum_{n=1}^{N}A_{nn})^2 && \textit{sum of square is smaller than square of sum (this is proved by induction on n)} \\
&= (tr(D))^2(tr(A))^2 \\
tr(DA) &\le tr(D)tr(A)
\end{aligned}$$

Remark for LDA

The inverse of a symmetric matrix is also symmetric, here is the proof :

$$\begin{aligned}
AA^{-1} &= I \\
A^TA^{-1} &= I \\
(A^TA^{-1})^T &= I \\
A^{-T}A &= I \\
A^{-T} = (A^{-1})^T &= A^{-1} && \textit{inverse of symmetric matrix is symmetric}
\end{aligned}$$

However the product of two symmetric matrices, including $(\Sigma_{within})^{-1}\Sigma_{between}$, is not necessarily symmetric (find counter example).

3.4 Neural Network

A partite in a graph is a set of vertices having no edges connecting any two vertices in the set. Graphs which can be partitioned into multiple partites are called multipartite-graph. Network is a specialized multipartite-graph in which each partite is allowed to have edges connecting to at most two other neighbouring partites.

3.4a Topology of neural network `5 = [L,M0,M1,K,N]`

- 1 *L+1* layers, indiced as $l \in [0,L]$
- layer *0* is input layer, on which, each node is simply a scalar value x
- layer *[1,L]* are hidden/output layer, on which, each node is a complicated `struct` $\{y, x, z, e, \delta\}$

about layer

- 23 there are M_0 input nodes, M_1 output nodes
- 4 there are K nodes in whole network, nodes are indiced as $k \in [1,K]$
- nodes are not labelled with layer index

about node

- there exist edges between layer l and layer $l+1$ only, $\forall l \in [0,L-1]$
- w_{ij} is weight of edge linking node i in layer l and node j in layer $l+1$, for $\forall l \in [0,L-1]$
- weights are tuned so as to minimise training error uising gradient descent
- key is to find gradient $\partial E/\partial w_k$ for all weights efficiently, without duplicated calculations

about edge

- network function is denoted as $B = f(A|w)$, where $A \in \Re^{M0}$ and $B \in \Re^{M1}$
- 5 network weights are trained with N data points $(A_n \in \Re^{M0}, B_n \in \Re^{M1})$ $\forall n \in [1,N]$, error $E \in \Re^1$ is defined as :

$$E = \frac{1}{2}\sum_{n=1}^{N}\|f(A_n|w) - B_n\|^2 \qquad \text{A and B are input/output of network fct}$$

$$= \frac{1}{2}(F(A|w)-B)^T(F(A|w)-B) \qquad \text{where } F(A|w) = \begin{bmatrix} f(A_1|w) \\ f(A_2|w) \\ \dots \end{bmatrix}$$

3.4b Topology of a node `5 = [y,x,z,e,δ] and 5 steps`

A node can be visualized with x as node value and w as edge value *(hide y, z, e and δ for clarity)*.

$$\begin{tikzcd}
y_k \arrow[rr, "x_k = s(y_k)"] & {} \arrow[d] & x_k \\
& z_k = s'(y_k) \arrow[d] & \\
\delta_k & {} & e_k \arrow[ll]
\end{tikzcd}$$

$\delta_k \equiv \partial E/\partial y_k = z_k$

For node k in layer l

1	*node value*		=	x_k
2	*node delta (we will define it later)*		=	δ_k
3	*combo of node values from layer l-1*		=	y_k
4	*combo of node deltas from layer l+1*		=	e_k
	s is activation fct, such that	$s(x)$	=	$1/(1+e^{-x})$
5	*activation slope*	z_k	=	$s'(y_k)$

`Step1` Initialize value in layer *0* :

$$\{x_k\} = A_n \qquad \text{where } \{x_k\} \text{ is all } M_0 \text{ node-values in layer 0}$$

`Step2` Forward propagation of input :

$$y_k = \sum\nolimits_{i \in layer[l-1]} w_{ik}x_i + \theta_k \qquad \text{by definition}$$
$$x_k = s(y_k) \qquad \text{by definition}$$
$$z_k = s'(y_k) \qquad \text{by definition}$$

`Step3` Calculate error in layer *L* :

$$\{\delta_k\} = f(A_n|w) - B_n \qquad \text{where } \{\delta_k\} \text{ is all } M_1 \text{ node-errors in layer L}$$

`Step4` Backward propagation of error by chain rule :

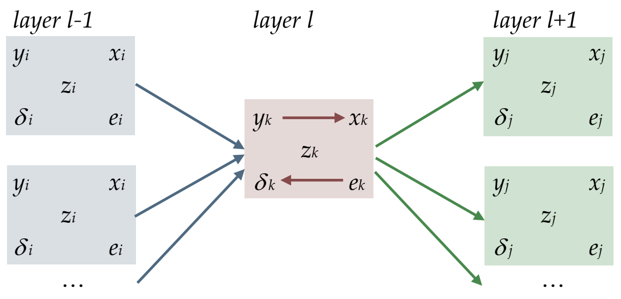
$$e_k = \sum\nolimits_{j \in layer[l+1]} w_{kj}\delta_j \qquad \text{by definition}$$
$$\delta_k = z_k e_k \qquad \text{(equation 1) to be proved in next section}$$
$$\frac{\partial E}{\partial w_{ik}} = x_i\delta_k \qquad \text{(equation 2) to be proved in next section}$$

Step5 We can do online training *(sample-by-sample weight update)* or offline training *(batch weight update)* :

$$
\begin{aligned}
w_{ik}' &= w_{ik} - c\frac{\partial E}{\partial w_{ik}} && \textit{for all node k in layer l and all node i in layer l-1, } \forall l \in [1,L] \\
&= w_{ik} - c x_i \delta_k \\
&= w_{ik} - c x_i z_k e_k
\end{aligned}
$$

3.4c Proof of equation 1-2

The idea of backpropagation is to breakdown the gradient by chain rule into a two-step recursive function. Let's consider node *k* in layer *l*, node *i* in layer *l-1* and node *j* in layer *l+1*.



layer l-1
layer l
layer l+1
y_i x_i z_i δ_i e_i
y_i x_i z_i δ_i e_i
…
y_k x_k z_k δ_k e_k
y_j x_j z_j δ_j e_j
y_j x_j z_j δ_j e_j
…

If we consider the network as a DAG, we can differentiate a variable v wrt another variable u, as long as there exists a path that starts from v and reaches u. Don't do it the opposite way.

Firstly, we define δ_k and apply chain rule to the subgraph started from node *k* in layer *l* :

$$
\begin{aligned}
\delta_k &\equiv \frac{\partial E}{\partial y_k} && \textit{objective is to expand } \frac{\partial output}{\partial input} \\
&= \frac{\partial E}{\partial x_k}\frac{\partial x_k}{\partial y_k} && \textit{since a shock in } y_k \textit{ trigger a shock only in } x_k \\
&= \left[\sum_{j \in layer[l+1]} \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_k}\right]\frac{\partial x_k}{\partial y_k} && \textit{since a shock in } x_k \textit{ trigger a shock only in all } y_j \textit{ in layer l+1} \\
&= \left[\sum_{j \in layer[l+1]} \frac{\partial E}{\partial y_j} w_{kj}\right] z_k && \textit{since } \frac{\partial y_j}{\partial x_k} = \frac{\partial \sum_{\eta \in layer[l]} w_{\eta j} x_\eta}{\partial x_k} = w_{kj} \textit{ and } \frac{\partial x_k}{\partial y_k} = z_k \\
&= [\sum_{j \in layer[l+1]} \delta_j w_{kj}] z_k && \textit{since } \frac{\partial E}{\partial y_j} = \delta_j \\
&= z_k e_k && \textit{(1) is proved}
\end{aligned}
$$

Secondly, we solve the derivative of error wrt each weight and wrt each bias term :

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ik}} &= \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial w_{ik}} \\
&= \delta_k \frac{\partial}{\partial w_{ik}}(\sum_{i \in layer[l-1]} w_{ik} x_i + \theta_k) && \textit{since } \frac{\partial E}{\partial y_k} = \delta_k \\
&= x_i \delta_k && \textit{(2) is proved for weight}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial \theta_k} &= \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial \theta_k} \\
&= \delta_k \frac{\partial}{\partial \theta_k}(\sum_{i \in layer[l-1]} w_{ik} x_i + \theta_k) \\
&= \delta_k && \textit{(2) is proved for bias}
\end{aligned}
$$

3.5 Support Vector Machine

Hyperplane in $\Re^M$ can be specified as $xw+b=0$, where w (column matrix) is the normal vector of hyperplane, and x (row matrix) is any point on the hyperplane. It is ***NOT*** necessary that $w^T w=1$, then we have :

$$\begin{aligned} xw &= -b \\ |x||w|\cos\theta &= -b \\ \underbrace{|x|\cos\theta}_{\text{projection of } x \text{ on } w} &= \underbrace{-b/|w|}_{\text{normal distance}} \end{aligned}$$

where θ is included angle between point x and normal vector w

10 remarks for three next sections ...

```
(1) Formulation              (4) SVM objective     (8) How to solve b?
(2) Introduce Lagrange       (5) SVM constraint    (9) How to solve y?
(3) Differentiate wrt w&b    (6) Definition of H   (0) Kernel trick
                             (7) Definition of α
```

3.5a Linear separable classification 10

Given N data points x_n $n\in[1,N]$ in $\Re^M$ and boolean labels $y_n\in[-1,+1]$, we find a double-hyperplane with maximum margin such that all data points are correctly classified. Double hyperplane in $\Re^M$ is defined by $M+1$ parameters, $w\in\Re^M$ and $b\in\Re^1$:

$$xw+b = \pm 1 \qquad \textit{we do } \textbf{NOT} \textit{ assume } w^T w=1$$

$$margin = \frac{-b+1}{|w|}-\frac{-b-1}{|w|} = \frac{2}{|w|}$$

$$\max_{w,b}\frac{2}{|w|} \qquad \textit{such that} \begin{bmatrix} x_n w+b\ge +1 & \textit{for} & y_n=+1 \\ x_n w+b\le -1 & \textit{for} & y_n=-1 \end{bmatrix}$$

$$\Rightarrow \min_{w,b}\frac{w^T w}{2} \qquad \textit{such that} \begin{bmatrix} y_n(x_n w+b)\ge 1 & \textit{for} & y_n=+1 \\ y_n(x_n w+b)\ge 1 & \textit{for} & y_n=-1 \end{bmatrix} \qquad (1)$$

$$\Rightarrow \min_{w,b}\max_{\alpha}\frac{w^T w}{2}-\sum_n\alpha_n[y_n(x_n w+b)-1] \qquad \textit{such that } \alpha_n\ge 0,\forall n\in[1,N] \qquad (2)$$

$$\Rightarrow \min_{w,b}\max_{\alpha}\underbrace{\frac{w^T w}{2}-(\sum_n\alpha_n y_n x_n)w-(\sum_n\alpha_n y_n)b+\sum_n\alpha_n}_{L} \qquad \textit{such that } \alpha_n\ge 0,\forall n\in[1,N]$$

We can differentiate above objective function with respect to w b and α, therefore setting up $M+1+N$ equations of $M+1+N$ unknowns. Yet in most *SVM* derivations, we differentiate with respect to w and b only, as we want to get an objective purely in α (no w nor b).

$$\frac{\partial L}{\partial w} = w^T-\sum_n\alpha_n y_n x_n = 0 \quad\Rightarrow\quad \sum_n\alpha_n y_n x_n = w^T$$

$$\frac{\partial L}{\partial b} = \sum_n\alpha_n y_n = 0 \quad\Rightarrow\quad \sum_n\alpha_n y_n = 0$$

Substituting into L, we have (term highlighted in blue is removed by $\sum_n\alpha_n y_n=0$) :

$$\max_{\alpha}\frac{(\sum_n\alpha_n y_n x_n)(\sum_n\alpha_n y_n x_n)^T}{2}-(\sum_n\alpha_n y_n x_n)(\sum_n\alpha_n y_n x_n)^T+\sum_n\alpha_n \qquad \textit{such that } \alpha_n\ge 0,\forall n\in[1,N] \textit{ and } \sum_n\alpha_n y_n=0 \qquad (3)$$

$$\Rightarrow \max_{\alpha}-\frac{1}{2}(\sum_n\alpha_n y_n x_n)(\sum_n\alpha_n y_n x_n)^T+\sum_n\alpha_n \qquad \textit{such that } \alpha_n\ge 0,\forall n\in[1,N] \textit{ and } \sum_n\alpha_n y_n=0$$

$$\Rightarrow \max_{\alpha}-\frac{1}{2}\sum_{n,m}\alpha_n\alpha_m(y_n x_n)(y_m x_m)^T+\sum_n\alpha_n \qquad \textit{such that } \alpha_n\ge 0,\forall n\in[1,N] \textit{ and } \sum_n\alpha_n y_n=0$$

$$\Rightarrow \max_{\alpha}-\frac{1}{2}\alpha^T H\alpha+l^T\alpha \qquad \textit{such that } \alpha_n\ge 0,\forall n\in[1,N] \textit{ and } y^T\alpha=0 \qquad (4)\&(5)$$

$\leftarrow$ *Negative quadratic term coeff, hence maximization (as opposed to minimization)*

$$H = [H_{nm}] \qquad \textit{H is } N\times N \textit{ matrix of dot products} \qquad (6)$$

$$H_{nm} = y_n y_m x_n x_m^T$$

$$\alpha = \begin{bmatrix}\alpha_1 \\ \alpha_2 \\ \dots\end{bmatrix} \qquad \alpha \textit{ is column matrix of Lagrange multiplers} \qquad (7)$$

y is column matrix of labels
l is column matrix of all one

The algorithm can be summarised as :

- solve α by quadratic programming (such as *sequential quadratic programming* or *sequential minimal optimization*)
- solve b by picking any support vector such that :

$$\begin{aligned} b &= y_{sv} - x_{sv}w \\ &= y_{sv} - x_{sv}(\textstyle\sum_n \alpha_n y_n x_n)^T \\ &= y_{sv} - \textstyle\sum_n \alpha_n y_n (x_{sv} x_n^T) \end{aligned} \tag{8}$$

- there is no need to solve for w, in testing we simply apply :

$$\begin{aligned} y &= xw + b \\ &= x(\textstyle\sum_n \alpha_n y_n x_n)^T + b \\ &= \textstyle\sum_n \alpha_n y_n (x x_n^T) + b \end{aligned} \tag{9}$$

- it can be extended to nonlinear hyperplane using the kernel trick, i.e. replacing all dot product $x_n x_m^T$ by kernel :

$$\begin{aligned} H_{nm} &= y_n y_m K(x_n, x_m) \qquad\qquad \textit{where } K(x_n, x_m) : (\Re^M, \Re^M) \to \Re^1 \\ b &= y_{sv} - \textstyle\sum_n \alpha_n y_n K(x_n, x_{sv}) \\ y &= \textstyle\sum_n \alpha_n y_n K(x_n, x) + b \\ &= \textstyle\sum_n \alpha_n y_n K(x_n, x) - \sum_n \alpha_n y_n K(x_n, x_{sv}) + y_{sv} \\ &= \textstyle\sum_n \alpha_n y_n (K(x_n, x) - K(x_n, x_{sv})) + y_{sv} \end{aligned} \tag{10}$$

3.5b Linear non-separable classification

By introducing an error term ξn for each data point, and an error weight C, we have new objective *(new items are in red)* :

$$\min_{w,b,\xi} \frac{w^T w}{2} + C\textstyle\sum_n \xi_n \qquad \textit{such that } \begin{bmatrix} y_n(x_n w + b) \ge 1 - \xi_n & \textit{for} & y_n = +1 \\ y_n(x_n w + b) \ge 1 - \xi_n & \textit{for} & y_n = -1 \end{bmatrix} \tag{1}$$

$$\textit{and } \ \xi_n \ge 0, \forall n \in [1, N]$$

$$\Rightarrow \min_{w,b,\xi} \max_{\alpha,\beta} \frac{w^T w}{2} + C\textstyle\sum_n \xi_n - \sum_n \alpha_n [y_n(x_n w + b) - 1 + \xi_n] - \sum_n \beta_n \xi_n \qquad \textit{such that } \ \alpha_n, \beta_n \ge 0, \forall n \in [1, N] \tag{2}$$

$$\Rightarrow \min_{w,b} \max_{\alpha} \frac{w^T w}{2} + C\textstyle\sum_n \xi_n + \begin{bmatrix} -(\sum_n \alpha_n y_n x_n) w - (\sum_n \alpha_n y_n) b \\ + \sum_n \alpha_n - \sum_n \alpha_n \xi_n - \sum_n \beta_n \xi_n \end{bmatrix} \qquad \textit{such that } \ \alpha_n, \beta_n \ge 0, \forall n \in [1, N]$$

We can differentiate above objective function with respect to w, b and ξ.

$$\begin{aligned} \frac{\partial L}{\partial w} &= w^T - \textstyle\sum_n \alpha_n y_n x_n &= 0 \quad &\Rightarrow \quad \textstyle\sum_n \alpha_n y_n x_n &= w^T \\ \frac{\partial L}{\partial b} &= \textstyle\sum_n \alpha_n y_n &= 0 \quad &\Rightarrow \quad \textstyle\sum_n \alpha_n y_n &= 0 \\ \frac{\partial L}{\partial \xi_n} &= C - \alpha_n - \beta_n &= 0 \quad &\Rightarrow \quad \alpha_n + \beta_n &= C \end{aligned}$$

Substituting into L, we have the **same objective** as separable case, there are differences in constraints only. The terms highlighted in red are all cancelled by $\alpha_n + \beta_n = C$.

$$\max_{\alpha} \frac{(\sum_n \alpha_n y_n x_n)(\sum_n \alpha_n y_n x_n)^T}{2} - (\textstyle\sum_n \alpha_n y_n x_n)(\sum_n \alpha_n y_n x_n)^T + \sum_n \alpha_n \qquad \textit{such that } \ 0 \le \alpha_n \le C, \forall n \in [1, N] \ \textit{and } \ \textstyle\sum_n \alpha_n y_n = 0 \tag{3}$$

$$\Rightarrow \ \ldots$$

$$\Rightarrow \max_{\alpha} -\frac{1}{2} \alpha^T H \alpha + I^T \alpha \qquad \textit{such that } \ 0 \le \alpha_n \le C, \forall n \in [1, N] \ \textit{and } \ y^T \alpha = 0 \tag{4)\&(5}$$

where H, α, b and y are defined and calculated in the same way as linear separable case. Kernel applies too. (6)-(10)

3.5c Linear regression

By introducing an insenstive tube of $\pm\varepsilon$, error term ξ_n is considered as regression error outside the tube.

$$\min_{w,b,\xi} \frac{w^T w}{2} + C\sum\nolimits_n(\xi_n^+ + \xi_n^-) \qquad where \begin{bmatrix} x_n w + b + \varepsilon \ge y_n - \xi_n^+ \\ x_n w + b - \varepsilon \le y_n + \xi_n^- \end{bmatrix} \qquad such\ that\ \ \xi_n^\pm \ge 0, \forall n \in [1,N] \qquad (1)$$

$$\Rightarrow \min_{w,b,\xi^\pm} \max_{\alpha^\pm,\beta^\pm} \frac{w^T w}{2} + C\sum\nolimits_n(\xi_n^+ + \xi_n^-) - \begin{bmatrix} \sum_n \alpha_n^+[(x_n w + b) - y_n + \varepsilon + \xi_n^+] + \sum_n \beta_n^+ \xi_n^+ \\ \sum_n \alpha_n^-[y_n - (x_n w + b) + \varepsilon + \xi_n^-] + \sum_n \beta_n^- \xi_n^- \end{bmatrix} \qquad such\ that\ \ \alpha_n^\pm, \beta_n^\pm \ge 0, \forall n \in [1,N] \qquad (2)$$

$$\Rightarrow \min_{w,b,\xi^\pm} \max_{\alpha^\pm,\beta^\pm} \frac{w^T w}{2} + C\sum\nolimits_n(\xi_n^+ + \xi_n^-) - \begin{bmatrix} +(\sum_n \alpha_n^+ x_n)w + (\sum_n \alpha_n^+)b + \sum_n \alpha_n^+(-y_n + \varepsilon) + \sum_n(\alpha_n^+ + \beta_n^+)\xi_n^+ \\ -(\sum_n \alpha_n^- x_n)w - (\sum_n \alpha_n^-)b + \sum_n \alpha_n^-(+y_n + \varepsilon) + \sum_n(\alpha_n^- + \beta_n^-)\xi_n^- \end{bmatrix} \qquad such\ that\ \ \alpha_n^\pm, \beta_n^\pm \ge 0, \forall n \in [1,N]$$

We can differentiate above objective function with respect to *w*, *b* and $\xi^\pm$.

$$\begin{array}{lclclcrcl}
\dfrac{\partial L}{\partial w} & = & w^T - \sum_n(\alpha_n^+ - \alpha_n^-)x_n & = & 0 & \Rightarrow & \sum_n(\alpha_n^+ - \alpha_n^-)x_n & = & w^T \\
\dfrac{\partial L}{\partial b} & = & \sum_n(\alpha_n^+ - \alpha_n^-) & = & 0 & \Rightarrow & \sum_n(\alpha_n^+ - \alpha_n^-) & = & 0 \\
\dfrac{\partial L}{\partial \xi_n^+} & = & C - (\alpha_n^+ + \beta_n^+) & = & 0 & \Rightarrow & \alpha_n^+ + \beta_n^+ & = & C \\
\dfrac{\partial L}{\partial \xi_n^-} & = & C - (\alpha_n^- + \beta_n^-) & = & 0 & \Rightarrow & \alpha_n^- + \beta_n^- & = & C
\end{array}$$

Substituting into *L*, we have term in blue cancelled by $\sum_n(\alpha_n^+ - \alpha_n^-) = 0$ and terms in red cancelled by $\alpha_n^\pm + \beta_n^\pm = C$.

$$\min_{w} \max_{\alpha^\pm} \frac{w^T w}{2} - \begin{bmatrix} +(\sum_n \alpha_n^+ x_n)w + \sum_n \alpha_n^+(-y_n + \varepsilon) \\ -(\sum_n \alpha_n^- x_n)w + \sum_n \alpha_n^-(+y_n + \varepsilon) \end{bmatrix} \qquad such\ that\ \ 0 \le \alpha_n^\pm \le C, \forall n \in [1,N]\ \ and\ \ l^T\alpha = 0 \qquad (3)$$

$$\Rightarrow \min_{w} \max_{\alpha^\pm} \frac{w^T w}{2} - \sum\nolimits_n(\alpha_n^+ - \alpha_n^-)x_n w + \sum\nolimits_n(\alpha_n^+ - \alpha_n^-)y_n - \sum\nolimits_n(\alpha_n^+ + \alpha_n^-)\varepsilon \qquad such\ that\ \ 0 \le \alpha_n^\pm \le C, \forall n \in [1,N]\ \ and\ \ l^T\alpha = 0$$

$$\Rightarrow \max_{\alpha^\pm} - \frac{(\sum_n(\alpha_n^+ - \alpha_n^-)x_n)(\sum_n(\alpha_n^+ - \alpha_n^-)x_n)^T}{2} + \sum\nolimits_n(\alpha_n^+ - \alpha_n^-)y_n - \sum\nolimits_n(\alpha_n^+ + \alpha_n^-)\varepsilon \qquad such\ that\ \ 0 \le \alpha_n^\pm \le C, \forall n \in [1,N]\ \ and\ \ l^T\alpha = 0$$

$$\Rightarrow \max_{\alpha^\pm} - \frac{1}{2}\alpha^T H \alpha + y^T\alpha - l^T\alpha^* \cdot \varepsilon \qquad such\ that\ \ 0 \le \alpha_n^\pm \le C, \forall n \in [1,N]\ \ and\ \ l^T\alpha = 0 \qquad (4)\&(5)$$

$$\begin{array}{llll}
where & H & = & [H_{nm}] \\
 & H_{nm} & = & x_n x_m^T \qquad (6) \\
 & \alpha & = & \begin{bmatrix} \alpha_1^+ - \alpha_1^- \\ \alpha_2^+ - \alpha_2^- \\ \alpha_3^+ - \alpha_3^- \\ \dots \\ \alpha_N^+ - \alpha_N^- \end{bmatrix} \quad and \quad \alpha^* = \begin{bmatrix} \alpha_1^+ + \alpha_1^- \\ \alpha_2^+ + \alpha_2^- \\ \alpha_3^+ + \alpha_3^- \\ \dots \\ \alpha_N^+ + \alpha_N^- \end{bmatrix} \qquad (7)
\end{array}$$

Finally solve *b* by picking any support vector such that :

$$\begin{array}{lcl}
b & = & \begin{bmatrix} y_{sv} - x_{sv}w - \varepsilon - \xi_{sv}^+ \\ y_{sv} - x_{sv}w + \varepsilon + \xi_{sv}^- \end{bmatrix} \\
 & = & \begin{bmatrix} y_{sv} - x_{sv}(\sum_n(\alpha_n^+ - \alpha_n^-)x_n)^T - \varepsilon - \xi_{sv}^+ \\ y_{sv} - x_{sv}(\sum_n(\alpha_n^+ - \alpha_n^-)x_n)^T + \varepsilon + \xi_{sv}^- \end{bmatrix} \\
 & = & \begin{bmatrix} y_{sv} - \sum_n(\alpha_n^+ - \alpha_n^-)(x_{sv}x_n^T) - \varepsilon - \xi_{sv}^+ \\ y_{sv} - \sum_n(\alpha_n^+ - \alpha_n^-)(x_{sv}x_n^T) + \varepsilon + \xi_{sv}^- \end{bmatrix} \qquad (8) \\
y & = & xw + b \\
 & = & x(\sum_n(\alpha_n^+ - \alpha_n^-)x_n)^T + b \\
 & = & \sum_n(\alpha_n^+ - \alpha_n^-)(xx_n^T) + b \qquad (9)
\end{array}$$

Please read "Support Vector Machines Explained" by Tristan Fletcher

3.6 Generative Adversarial Net

Given a set of training data $x_n \in \Re^M \ \forall n \in [1,N]$ drawn from distribution $p_{data}(x)$, *GAN* is a learning algorithm, which approximates the distribution $p_{data}(x)$ with a pair of neutral networks (G and D) combating each other in a multistage game. G is a ***generative network*** which generates output $x = G(z|w_G)$ with a random number generator z as input. D is a ***discriminative network*** which estimates the probability $y = D(x|w_D) \in [0,1]$ that an input x is a sample drawn from $p_{data}(x)$ rather than generated by $G(z|w_G)$. In short :

- G is a counterfeit generator, which performs exact imitation
- D is a counterfeit detector, which differentiates training data *(hopefully $y \sim 1$)* from counterfeit data *(hopefully $y \sim 0$)*
- G has L input nodes, for $z \in \Re^L$ is a L dimensional random number generator
- G has M output nodes, for $x \in \Re^M$ is a M dimensional simulated data
- D has M input nodes, for $x \in \Re^M$ is a M dimensional input data
- D has 1 output node, for $y \in [0,1]$ is the probability that input data is drawn from $p_{data}(x)$
- weight sets for the two networks are w_G and w_D respectively
- distribution of random number generator z is $p_{rng}(z)$
- distribution of counterfeit generator $G(z|w_G)$ is $p_G(x)$

3 different pdfs

The 3 distributions are related by :

$$\begin{aligned} p_{rng}(z)dz &= p_G(x)dx && \text{(1) suppose both } L,M=1 \\ &= p_G(x)\partial_z G(z|w_G)dz \end{aligned}$$

$$\boxed{\begin{aligned} p_G(x) &= \frac{p_{rng}(z)}{\partial_z G(z|w_G)} \\ p_G(x) &\sim p_{data}(x) \end{aligned}}$$

and we hope

adjust generator's weight and RNG z so as to…

achieve exact imitation of training data

Furthermore from *(1)*, we have :

$$\begin{aligned} f(x)p_G(x)dx &= f(x)p_{rng}(z)dz && \text{for any function } f(x) \\ &= f(G(z|w_G))p_{rng}(z)dz && \text{since } x = G(z|w_G) \\ \int f(x)p_G(x)dx &= \int f(G(z|w_G))p_{rng}(z)dz \\ E_{x\sim p_G}[f(x)] &= E_{x\sim p_{rng}}[f(G(z|w_G))] && (2) \end{aligned}$$

GAN concept

Log likelihood of correct discrimination is :

$$\begin{aligned} &\ln(D(x|w_D)) && \text{for all } x \text{ drawn from } p_{data}(x) \\ &\ln(1-D(x|w_D)) && \text{for all } x \text{ drawn from } p_G(x) \end{aligned}$$

Discriminator maximizes the likelihood *wrt* w_D, while generator minimizes ***discriminator's optimal*** *wrt* w_G.

$$\begin{aligned} & \min_{w_G}\max_{w_D} E_{x\sim p_{data}}[\ln(D(x|w_D))] + E_{x\sim p_G}[\ln(1-D(x|w_D))] \\ = & \min_{w_G}\max_{w_D} E_{x\sim p_{data}}[\ln(D(x|w_D))] + E_{z\sim p_{rng}}[\ln(1-D(G(z|w_G)|w_D))] && \text{applying (2)} \\ = & \min_{w_G}\max_{w_D}\left(\int_\infty p_{data}(x)\ln(D(x|w_D))dx + \int_\infty p_{rng}(z)\ln(1-D(G(z|w_G)|w_D))dz\right) && \text{minmax } f(x,y) \neq \text{maxmin } f(x,y) \text{ consider matrix example} \end{aligned}$$

GAN implementation

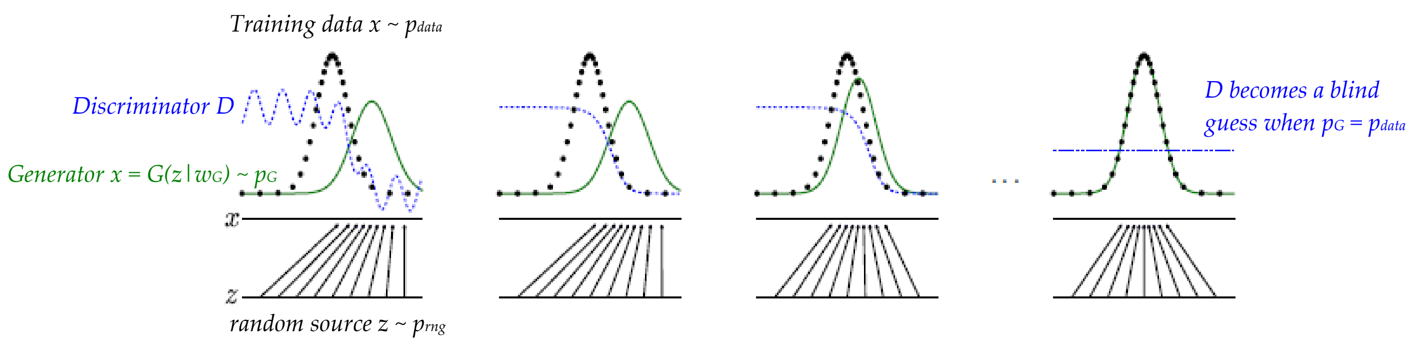
GAN is implemented as *K discriminator-iterations* plus *1 generator-iteration*.

different gradients

```
while(!converge)
{   for(int k=0; k!=K ++k)
    {
        draw N samples from Pdata(x)
        draw N samples from Prng(z) and plug in G(z|wG)
        perform (stochastic) gradient ascent of $\nabla_{w_D}\sum\ln(D(x|w_D)) + \nabla_{w_D}\sum\ln(1-D(G(z|w_G)|w_D))$   [D-step]
    }
    draw N samples from Prng(z) and plug in G(z|wG)
    perform (stochastic) gradient descent of $\nabla_{w_G}\sum\ln(1-D(G(z|w_G)|w_D))$   [G-step]
}
```

GAN convergence

The algorithm converges as illustrated, the approximation by *G* gets improved, discrimination by *D* becomes a blind guess.



Training data x ~ pdata
Discriminator D
Generator x = G(z | wG) ~ pG
x
z
random source z ~ prng
D becomes a blind guess when pG = pdata

[D-step] Given *G(z | wG)* in previous iteration, optimal of *D* in this iteration is given by :

$$\arg\max_{w_D} \int_\infty [p_{data}(x)\ln(D(x\,|\,w_D)) + p_G(x)\ln(1-D(x\,|\,w_D))]dx$$

$$\rightarrow \int_\infty [p_{data}(x)\nabla_{w_D}\ln(D(x\,|\,w_D)) + p_G(x)\nabla_{w_D}\ln(1-D(x\,|\,w_D))]dx = 0 \qquad \textit{since } p_{data}(x) \textit{ and } p_G(x) \textit{ are independent on } w_D$$

$$\rightarrow p_{data}(x)\nabla_{w_D}\ln(D(x\,|\,w_D)) + p_G(x)\nabla_{w_D}\ln(1-D(x\,|\,w_D)) = 0 \qquad \textit{for all } x\textit{, given enough degree of freedom in } w_D$$

$$\rightarrow D(x\,|\,w_D) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)} \qquad \textit{see remark}$$

Remark : find $\max_D[a\ln D + b\ln(1-D)]$

$$\begin{aligned} 0 &= \partial_D(a\ln D + b\ln(1-D)) \\ 0 &= a/D + b/(1-D) \\ bD &= a(1-D) \\ D &= a/(a+b) \end{aligned}$$

[G-step] Given optimum *D* in previous iteration, optimal of *G* in this iteration is given by :

$$\min_{w_G}\left(\max_{w_D} E_{x\sim p_{data}}[\ln(D(x\,|\,w_D))] + E_{x\sim p_G}[\ln(1-D(x\,|\,w_D))]\right)$$

$$= \min_{w_G}\left(\max_{w_D} E_{x\sim p_{data}}\left[\ln\frac{p_{data}(x)}{p_{data}(x)+p_G(x)}\right] + E_{x\sim p_G}\left[\ln\frac{p_G(x)}{p_{data}(x)+p_G(x)}\right]\right) \qquad \textit{plug in result from D-step}$$

$$= \min_{w_G}\left(\max_{w_D} E_{x\sim p_{data}}\left[\ln\frac{p_{data}(x)}{p_{data}(x)+p_G(x)}+\ln 2\right] + E_{x\sim p_G}\left[\ln\frac{p_G(x)}{p_{data}(x)+p_G(x)}+\ln 2\right]\right) - \ln 4$$

$$= \min_{w_G}\left(\max_{w_D} E_{x\sim p_{data}}\left[\ln\frac{p_{data}(x)}{(p_{data}(x)+p_G(x))/2}\right] + E_{x\sim p_G}\left[\ln\frac{p_G(x)}{(p_{data}(x)+p_G(x))/2}\right]\right) - \ln 4 \qquad \textit{read KL div doc}$$

$$= \min_{w_G}\left(D_{KL}(p_{data}(x)\,\|\,\frac{p_{data}(x)+p_G(x)}{2}) + D_{KL}(p_G(x)\,\|\,\frac{p_{data}(x)+p_G(x)}{2})\right) - \ln 4 \qquad \textit{since } D_{KL}(P\,\|\,Q) = E_P\left[\log_2\frac{P}{Q}\right]$$

$$= -\ln 4 \qquad \textit{since KL divergence must be non-negative}$$

Divergence is non-negative, thus minimum of above objective happens when :

$$p_{data}(x) = \frac{p_{data}(x)+p_G(x)}{2}$$

i.e. $p_{data}(x) = p_G(x)$

In short, when *GAN* is done, generator distribution converges to training data distribution. Can we introduce other magic numbers such as 3&6 instead of 2&4? Of course not, we need to construct another measure to generate the KL divergence.

Comparison to other algorithms

- *GAN* is liked *EM* as a bistep recursion :

- *E-step* is analogous to *D-step*
- *M-step* is analogous to *G-step*

- *GAN* is liked KL divergence :

- maximization of expected log likelihood

- *GAN* is liked reversed Monte Carlo simulation :

- *MC* finds function expectation of $F(z)$ from samples of random variable z, given $p_Z(z)$
- *GAN* finds parameters of $x = F(z|w)$ from samples of random variable x_{data}, given $p_Z(z)$