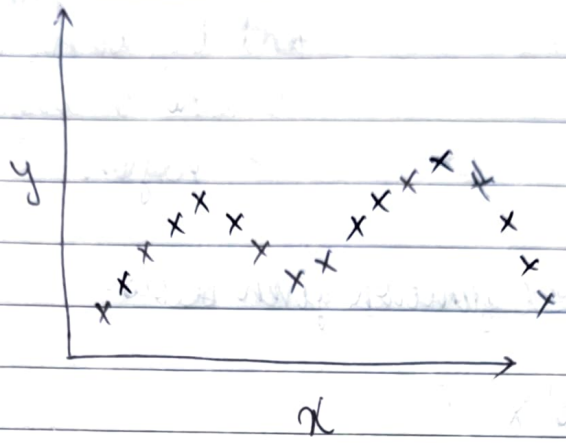
Locally weighted linear regression: →

suppose we have a dataset

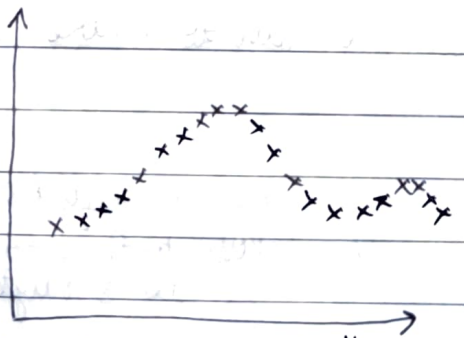


we want to find a curve that fits this shape. We will do this using locally weighted L.R.

"parametric" learning algorithm → we fit set of parameters (θ_i) to data
↳ example linear regression

"Non parametric" learning algorithm → Locally weighted L.R is an example of this

↳ amount of data/parameters you need to keep grows (linearly) with size of data.



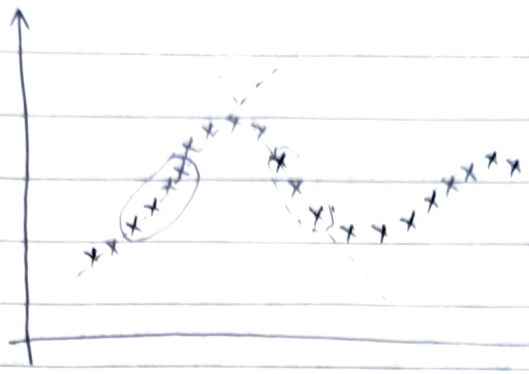
"FOR Linear regression"
to evaluate h at certain x
Fit θ to minimize.

$$\frac{1}{2}\sum_i \left(y^{(i)} - \theta^T x^{(i)}\right)^2 .$$

return $\theta^T x$ → gives best st. line.

"FOR locally weighted L.R"

For locally weighted regression



in locally weighted region regression we select a subregion and try to fit a st. line through that region

Fit θ to minimise cost function given below.

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

where $w^{(i)}$ is a "weighting function".

$$w^{(i)} = \exp\left(- \frac{(x^{(i)} - x)^2}{2\tau^2} \right)$$

for each point (x^i, y^i).

this is a value between 0 and 1.

↳ points have more weight if $|x^{(i)} - x|$ is small

" " less " " " is large

x is the point at which we have to make a prediction.

affects overfitting and underfitting

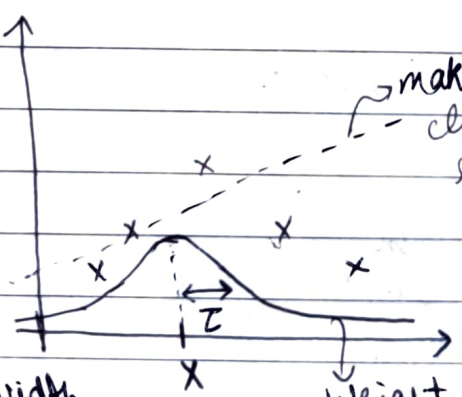
↑

lesser width ⇒ more local.

↑

to determine the width which determines the weight dist.

← τ is the bandwidth parameter.



use locally weighted regression → no. of features is low (n=2 or 3)
↳ huge amt. of data.

Probabilistic interpretation:→

Q Why do we use least square in error?

→ Assume, $y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$

↳ error term to capture unmodelled effects.

$\varepsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$.

"twiddle" → distributed.

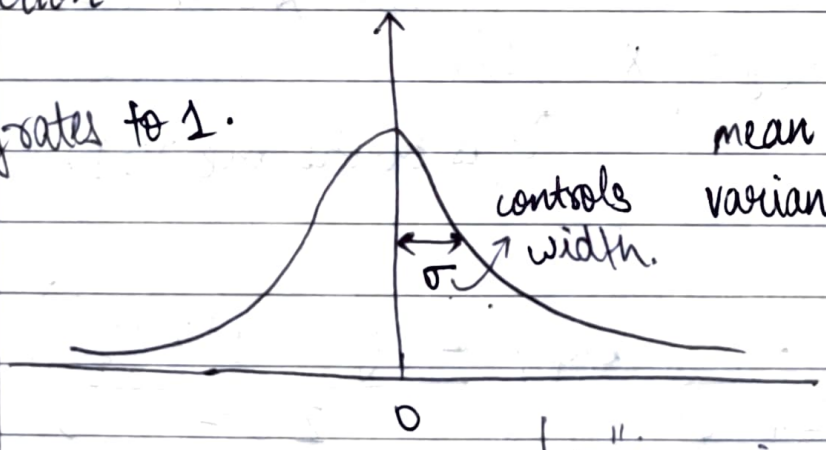
↳ mean 0 → variance σ^2.

↳ "normal distribution" / "gaussian distribution".

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

↓ prob. density function.

↓ integrates to 1.



↳ "Gaussian function"

$\varepsilon^{(i)}$ are IID (Independently and Identically Distributed)

↳ Error term for one input is independent from error ~~input~~ term of another input.

↳ not necessarily true.

This implies :→

$$P(y^{(i)} \mid x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{\left[y^{(i)} - \theta^T (x^{(i)})\right]^2}{2\sigma^2}\right)$$

↑ "parameterised as"

i-e,

$$y^{(i)} \mid x^{(i)}; \theta \sim N(\theta^T x^{(i)}, \sigma^2)$$

↳ mean ↳ variance.

↳ given x and θ what is the probability of a particular house's price

$\mathcal{L}(\theta)$ ↓ "likelihood of θ" $= P(\vec{y} \mid X; \theta)$ → probability of ys given all xs and θs

$$= \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)}; \theta).$$

↳ as errors are IID.

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

"log likelihood"

$$\ell(\theta) = \log\left[\mathcal{L}(\theta)\right].$$

$$l(\theta) = \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp(\dots).$$

$$l(\theta) = \sum_{i=1}^{m} \left[\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \log \exp(\dots) \right]$$

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

MLE → maxm. likelihood estimation.

↳ choose θ to maximise the likelihood (choose θ to maxm. $L(\theta)$)

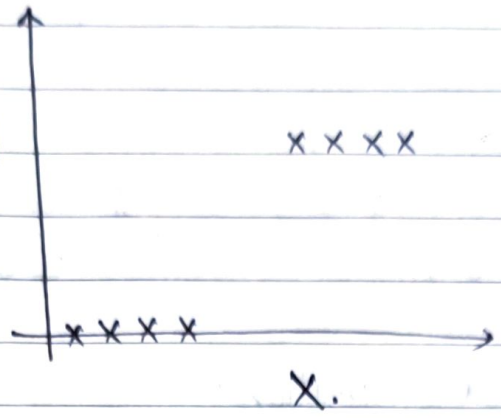
i.e choose θ that minimises $\frac{1}{2} \sum_{i=1}^{m} \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}$.

$$= J(\theta).$$

This proof shows finding θ such that $J(\theta)$ is minimum (least square errors) is finding the maximum likelihood estimate for the parameters θ under this set of assumptions that error terms are gaussian and IID.

classification :

$y \in \{0, 1\}$ [binary classification]



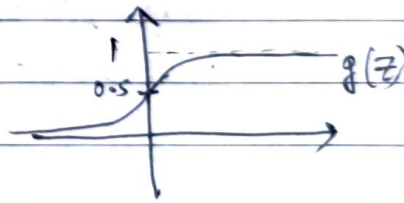
Logistic regression : →

want $h_\theta(x) \in [0,1]$

$$h_\theta(X) = g(\theta^T X) = \frac{1}{1+e^{-\theta^T X}} \qquad g(z) = \frac{1}{1+e^{-z}}$$

↓

"sigmoid" or "logistic function"



$p(y=1 \mid x;\theta) = h_\theta(X)$.

$p(y=0 \mid x;\theta) = 1 - h_\theta(X)$.

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

→ $p(y \mid x;\theta) = h(x)^y \, (1-h(x))^{1-y}$.

if $y=1 \rightarrow h(x)$

$y=0 \rightarrow 1-h(x)$.

$$L(\theta) = p(\vec{y} \mid X; \theta)$$

$$= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} \cdot \left(1 - h_\theta(x^{(i)})\right)^{1 - y^{(i)}}$$

we need to find θ that maxm. likelihood of parameters.

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

find θ to maxm. $\ell(\theta)$.

↓

we will use gradient ascent

Batch gradient ascent:

~~$\theta_j := \theta_j + \alpha \frac{\partial}{\partial \theta_j} \ell(\theta)$~~

$$\theta_j := \theta_j + \alpha \frac{\partial}{\partial \theta_j} \ell(\theta).$$

$$\boxed{\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) \cdot x_j^{(i)}}$$ [proof in lecture notes].

↓

we find parameters such that $\ell(\theta)$ is maxm.

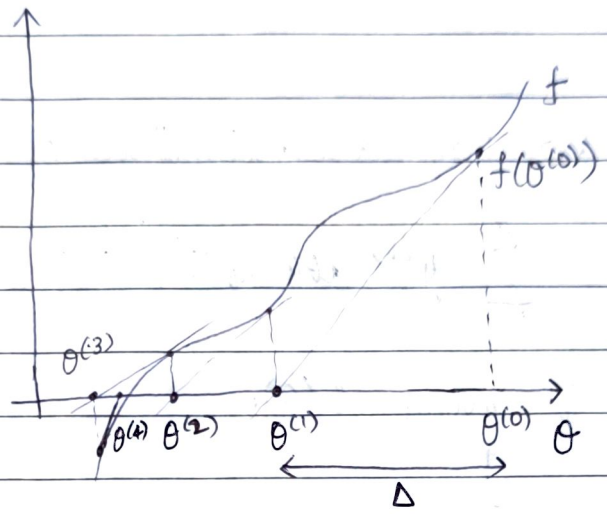
↳ same eqn. as linear regression but different $h_\theta(x^{(i)})$

Newtons method

- → faster algorithm
- → might take too long depending on no. of parameters

given a func. $f(\theta)$ find θ s.t. $f(\theta) = 0$

[for our prob. we want to maxm. $\ell(\theta) \Rightarrow \ell'(\theta) = 0$]



at each point create the slope line and use it's x-intercept as the new point.

Can overshoot too but returns back.

$$f'(\theta^{(0)}) = \frac{f(\theta^{(0)})}{\Delta}$$

$$\Delta = \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$$

$$\theta^{(1)} = \theta^{0} - \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$$

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

let $f(\theta) = \ell'(\theta)$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\ell'(\theta^{(t)})}{\ell''(\theta^{(t)})}$$

Newtons method has "quadratic convergence" ie. i.e. if after one iteration error is 0.01 then error becomes 0.0001 then it becomes 0.00000001 and so on. Hence it takes lesser number of iterations. → converges rapidly

↳ when θ is a vector : $\theta \in \mathbb{R}^{n+1}$

$$\theta^{(t+1)} := \theta^{(t)} - H^{-1} \nabla_\theta \ell.$$

$H^{-1} \in \mathbb{R}^{(n+1)\times(n+1)}$

$\nabla_\theta \ell$ → vector $\mathbb{R}^{n+1}$

H is a hessian matrix.

$$H_{ij} = \frac{\partial^2 \ell}{\partial\theta_i \partial\theta_j}$$

↓

takes large no. of computations to calculate for larger data sets.

↳ no. of parameters = 10,000 → 10^{10} computations at each step for hessian

↳ Hence, gradient descent is better for cases with large number of parameters.

"IGNORE"

+ sign as gradient "ASCENT" for logistic regression otherwise for grad "descent"

"−" sign

Exponential family:

PDF (prob. density function)

$$P(y, \eta) = b(y) \exp\left[\eta^T T(y) - a(\eta)\right]$$

$y \rightarrow$ data

$\eta \rightarrow$ natural parameter

$T(y) \rightarrow$ "sufficient statistic" [$= y$ for this lecture]

$b(y) \rightarrow$ base measure — only funct. of y

$a(\eta) \rightarrow$ log-partition — only func. of η ↳ indicates normalising function of PDF.

$$P(y, \eta) = \frac{b(y)\ \exp(\eta^T T(y))}{e^{+a(\eta)}}$$

bernoulli (binary PDF)

ϕ = prob. of an event.

$$P(y; \phi) = \phi^y (1-\phi)^{(1-y)}$$ we want to make this into above form

$$= \exp\left(\log\left(\phi^y (1-\phi)^{1-y}\right)\right)$$

$$= \exp\left[\log\left(\frac{\phi}{1-\phi}\right) y + \log(1-\phi)\right]$$

$b(y) = 1$.

$\eta = \log\left(\frac{\phi}{1-\phi}\right) \Rightarrow \phi = \frac{1}{1+e^{-\eta}}$

$T(y) = y$

$a(\eta) = -\log(1-\phi) = -\left(1 - \frac{1}{1+e^{-\eta}}\right) = \log(1+e^{\eta})$

Gaussian (with fixed variance)

Assume $\sigma^2 = 1$

$$P(y;\mu) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y-\mu)^2}{2}\right)$$

(μ: canonical parameters)

$$= \underbrace{\frac{1}{\sqrt{2\pi}} e^{-y^2/2}}_{b(y)} \exp\left(\underbrace{\mu}_{\eta} \underbrace{y}_{T(y)} - \underbrace{\frac{1}{2}\mu^2}_{a(\eta)}\right)$$

$$b(y) = \frac{1}{\sqrt{2\pi}} e^{-y^2/2}.$$

$$T(y) = y$$

$$\eta = \mu$$

$$a(\eta) = \frac{\mu^2}{2} = \frac{\eta^2}{2}$$

Properties of exponential families

(a) MLE wrt η → concave

(maxim. likelihood of exponential family)

NLL (negative log likelihood) is convex

(b) $E[y;\eta] = \frac{\partial}{\partial \eta} a(\eta)$.

(c) $\text{Var}[y;\eta] = \frac{\partial^2}{\partial \eta^2} a(\eta)$

} easy as no integrals

GLM (generalised linear models)

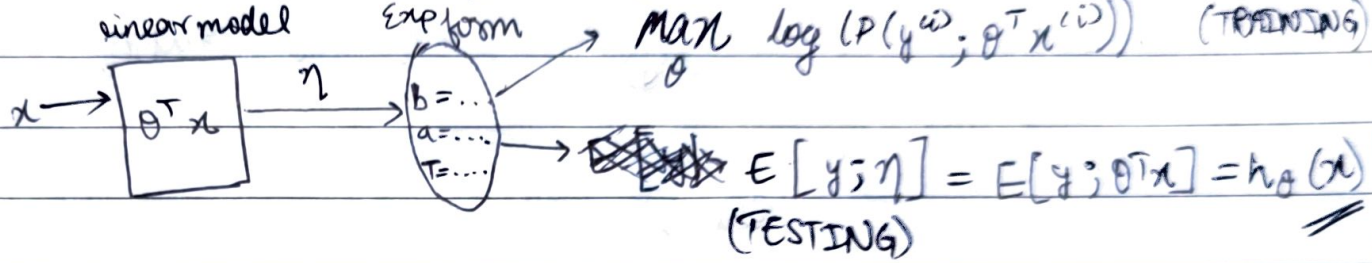
assumptions / design choices for GLM

(i) $y|x;\theta \sim$ Exponential family (member of)

(ii) $\eta = \theta^T x$

(iii) Test time : output $E[y|x:\theta]$

$\Rightarrow h_\theta(x) = E[y|x:\theta]$ → gradient ascent on log likelihood



θ to train the exponential family model whose mean predicts the ~~value of~~ hypothesis.

GLM Training

for each GLM learning update rule is the same;

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

Terminology $\eta \rightarrow$ natural parameter

μ $E[y;\eta] = g(\eta)$ → connonical response function

$\eta = g^{-1}(\mu)$ → " link function

3 parameterisations for GLM

mode param θ — normal parath η — cannonical param ϕ

ϕ → Bernoulli

$\mu\ \sigma^2$ → Gaussian

λ → poisson

$\theta^T x \to \eta$ design choice link b/w these

g and g^{-1} link b/w these.

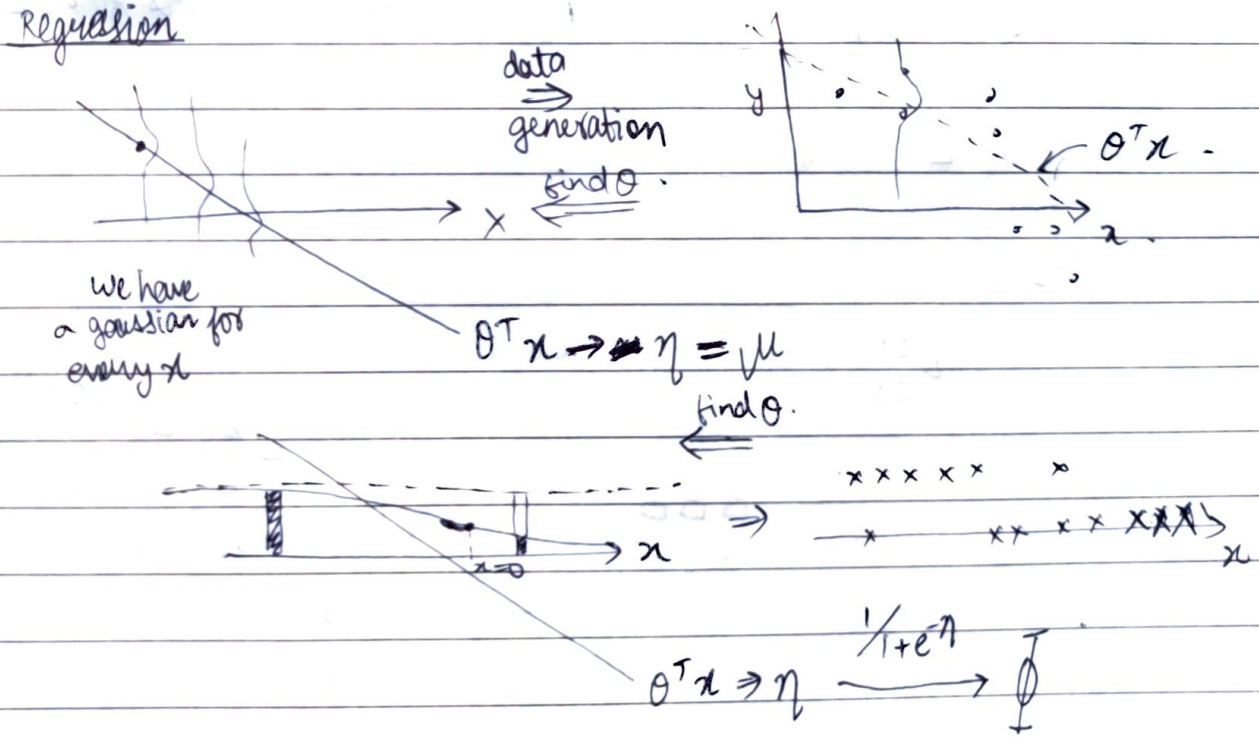
Logistic regression → bernoulli

$$h_\theta(x) = E[y|x;\theta] = \phi = \frac{1}{1+e^{-\eta}} = \frac{1}{1+e^{\theta^T x}}$$

We pick GLM distribution depending on the task at hand.

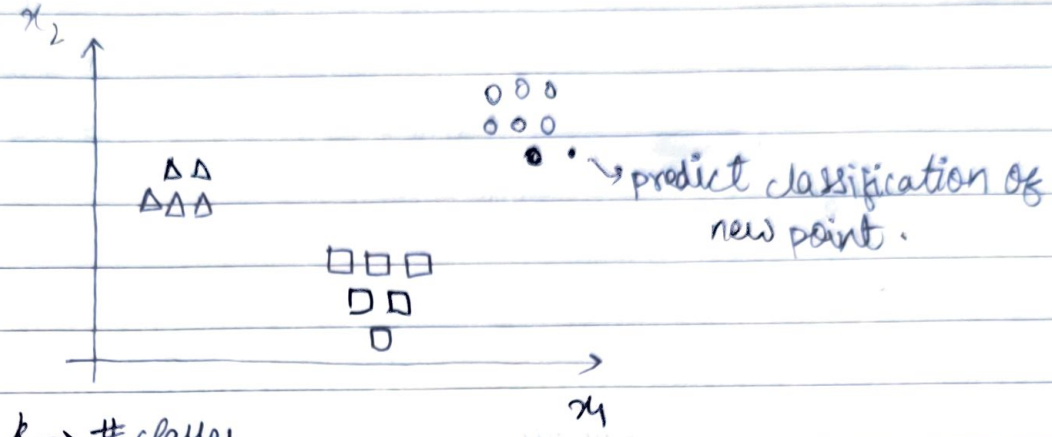
Assumptions

Regression



Softmax regression

is non GLM but we get similar eqns.
now softmax does cross entropy minm.



$k \rightarrow$ # classes

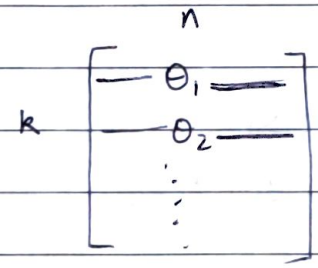
$x^{(i)} \in \mathbb{R}^n$

label $y = [\{0,1\}^k]$ eg $[0,0,1,0]$

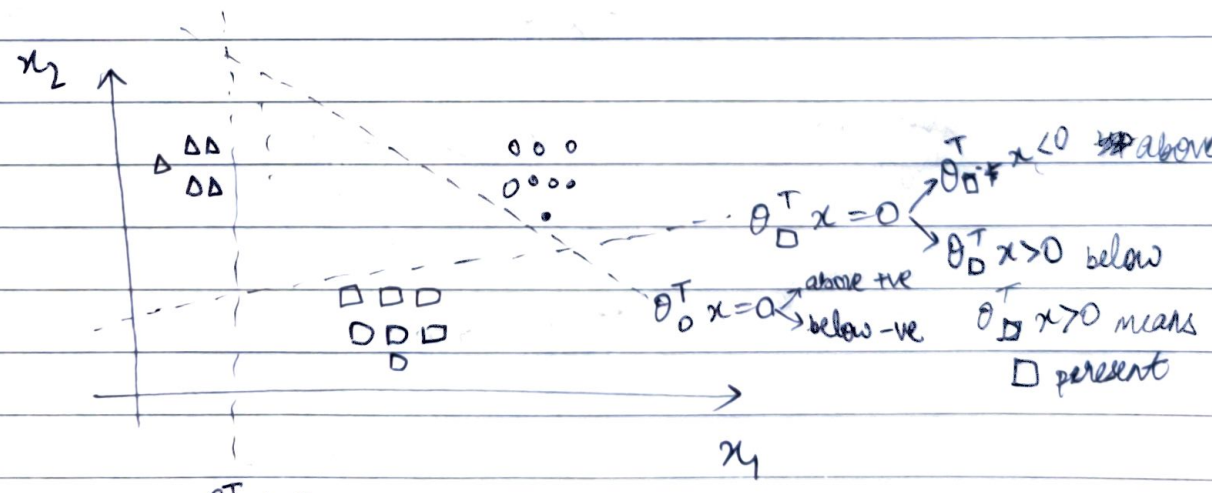
↳ for $k=4$

↳ tells state for each classification

$\theta_{class} \in \mathbb{R}^n$ k such class $\in \{\Delta, O, \square \dots\}$



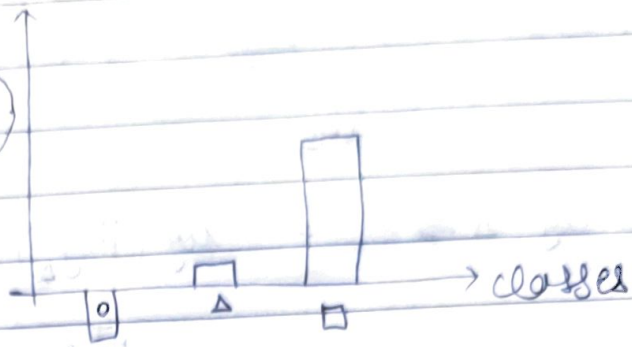
θ_1 is all class parameters for 1



given x

$\theta_{class}^T x$

logit space

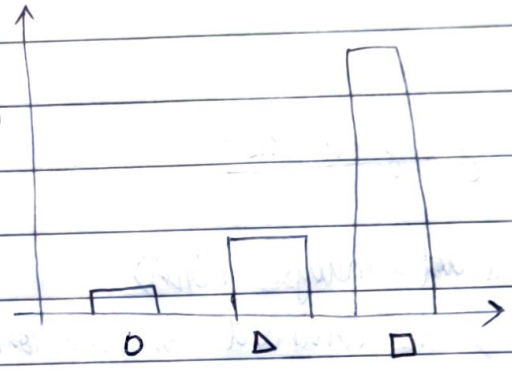


let's say class here is □

↓

we need to find a distribution s.t. for ○ & △ the value is less than □

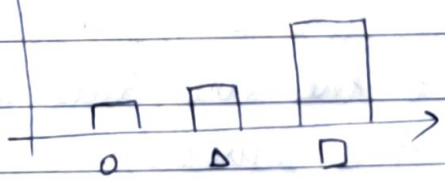
$\downarrow \exp(\theta_{class}^T x)$ first exponentiate our logit func.



$\Downarrow$ normalise this ⇒ divide all by their sum

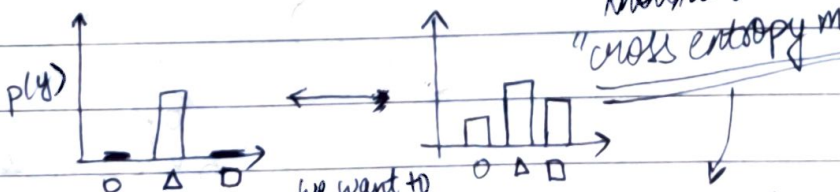
$$\frac{e^{\theta_{class}^T x}}{\sum_{i \in \{\triangle, \square, \bigcirc\}} e^{\theta_i^T x}}$$

$\hat{p}(y)$ → is function that gives us prob. dist. over classes



sum of heights = 1

true y will look like $p(y)$ for △ →



$$\text{cross Ent}(p, \hat{p}) = \sum_{y=\{\circ, \triangle, \square\}} p(y) \log \hat{p}(y)$$

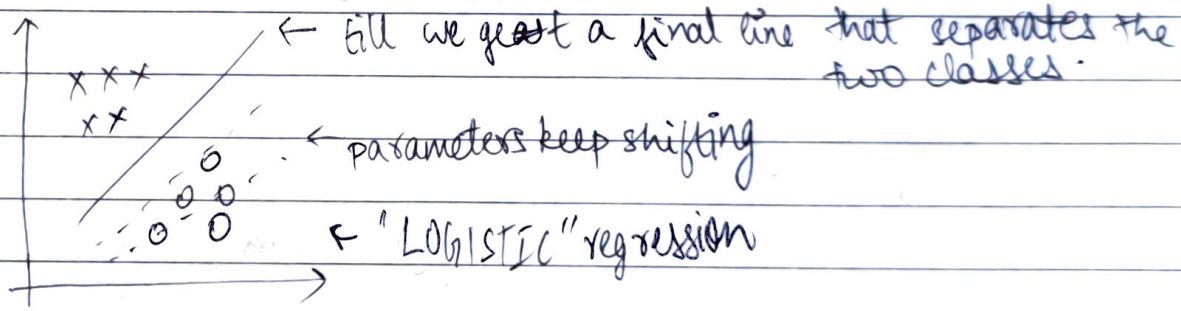
for let's say $\triangle$ $= -\log(\hat{p}(y_\triangle))$

$$\text{CrossEnt}(p, \hat{p}) = -\log\left(\frac{e^{\theta_\triangle^T x}}{\sum_{c \in \{\circ, \triangle, \square\}} e^{\theta_c^T x}}\right)$$

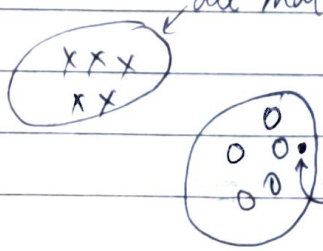
↙ do gradient descent wrt. parameters.

Generative learning algorithms.

- Gausian discriminantinanalysis (GDA)
- Generative & discriminant comparision
- Naive Bayes.



Generative learning algorithm → rather than trying to get the line, this algorithm looks at all the classes separately



Discriminative:

learn $p(y|x)$
(or learn $h_\theta(x) = \{0, 1$ directly

Generative LA:

learns $p(x|y)$ → given a class ⇒ what would its features look like
↑ features ↖ class

$p(y)$ → ~~pbb~~ prob. of a class.
↳ a patient walks in → what is prob. that he has cancer malignant

acc. to bayes rule.

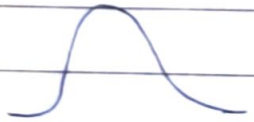
$$p(y=1|x) = \frac{p(x|y=1)\,p(y=1)}{p(x)}.$$

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

Gaussian discriminitive analysis. (GDA)

$x \in \mathbb{R}^n$ (~~droop~~ drop $x_0 = 1$ convention)

assume $p(x|y)$ is Gaussian — multivariable.



single var. gaussian.

$z \sim \mathcal{N}(\vec{\mu}, \Sigma)$ — expected value → mean, → covariance

$z \in \mathbb{R}^n$, $\vec{\mu} \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$.

$E[z] = \mu$

covariance $\text{cov}(z) = E[(z-\mu)(z-\mu)^T]$

$\text{cov}(z) = E[zz^T] - (Ez)(Ez)^T$

→ multivariable gaussian distribution.

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

(Refer lecture notes.)

GDA model

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

same Σ

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

parameters for both eqns. are $\mu_0, \mu_1, \Sigma, \phi$

$\mu_0, \mu_1 \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$, $\phi \in \mathbb{R}$

$p(y) = \phi^y (1-\phi)^{1-y}$ ← random bernoulli variable.

suppose we have a training set: $\{ (x^{(i)}, y^{(i)}) \}_{i=1}^m$

Joint likelihood:

$$\mathcal{L}(\phi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^m p(x^{(i)}, y^{(i)}, \phi, \mu_0, \mu_1, \Sigma)$$

$$= \prod_{i=1}^m p(x^{(i)} | y^{(i)}) \, p(y^{(i)})$$

we maximize the above function.

for generative algo we do not find the parameters.
↳ we try to know $p(x|y)$.

for discriminative algos

conditional likelihood

$$\mathcal{L}(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

we try to find θ s.t. cond. likelihood is maximized.

Maximum Likelihood estimation

$$\max_{\phi, \mu_0, \mu_1, \Sigma} \ell(\phi, \mu_0, \mu_1, \Sigma) = \log \mathcal{L}(\ldots).$$

take derivative ; equate it to 0 ; solve for parameters

final answer $\Rightarrow$ $\phi = \dfrac{\sum_{i=1}^{m} y^{(i)}}{m} = \dfrac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}{m}$ ("indicator")

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

$1(\text{true}) = 1$

$1(\text{false}) = 0$.

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

← sum of feature vectors for all ex. with $y = 1$

→ no. of examples with $y = 1$.

→ mean of parameters of $y = 1$ values

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

↳ trying to fit ellipses around the mean for a class.

Prediction: $\min_z (z-5)^2 = 0$. $\min_z \rightarrow$ min of over z

$\arg\min_z (z-5)^2 = 5$ //

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

$\min_z f(z)$ → returns smallest attainable value

$\arg\min_z f(z)$ → returns the argument for the smallest value.

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

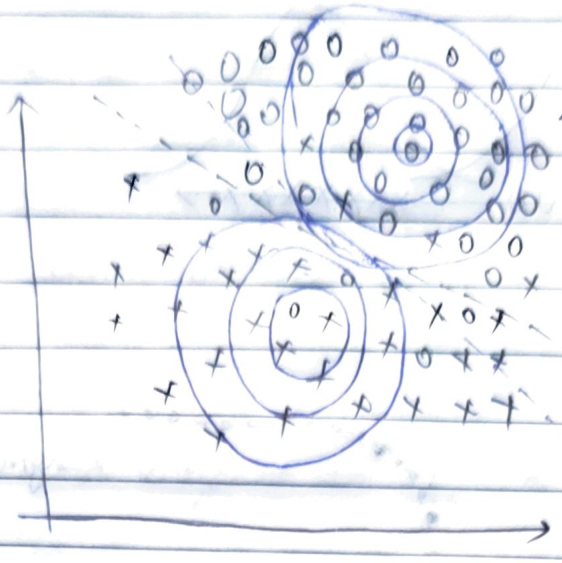
//

$$\arg\max_y p(x|y)\, p(y)$$

↳ constant as independent of y

as we only want the argument.

Discriminative vs generative learning algo :→



↝ for GDA we look at two classes separately

↝ line for logistic Linear regression.

GDA line different from logistic linear regression

Compare GDA to logistic regression:→

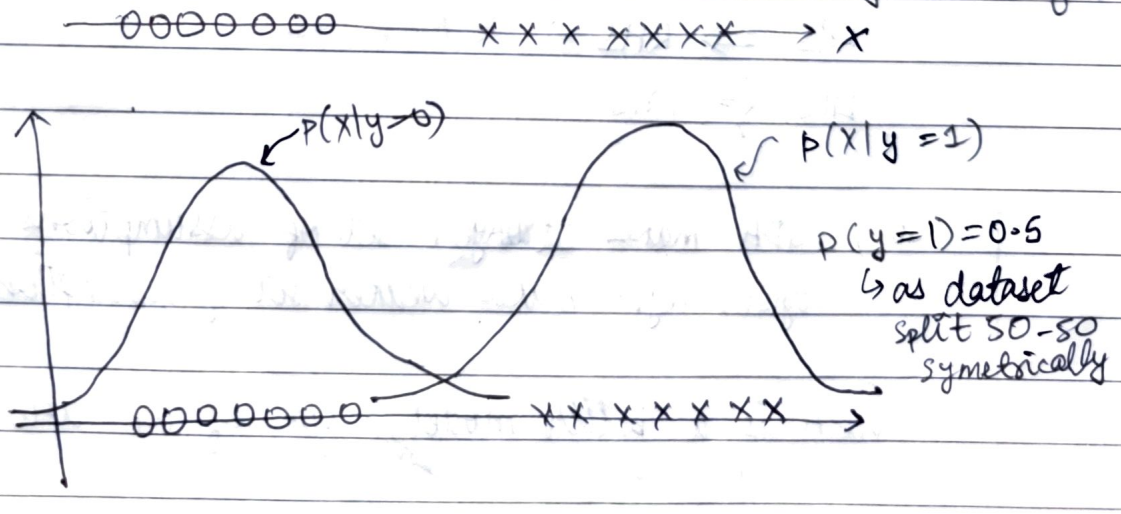
for fixed $\phi, \mu_0, \mu_1, \Sigma$, lets

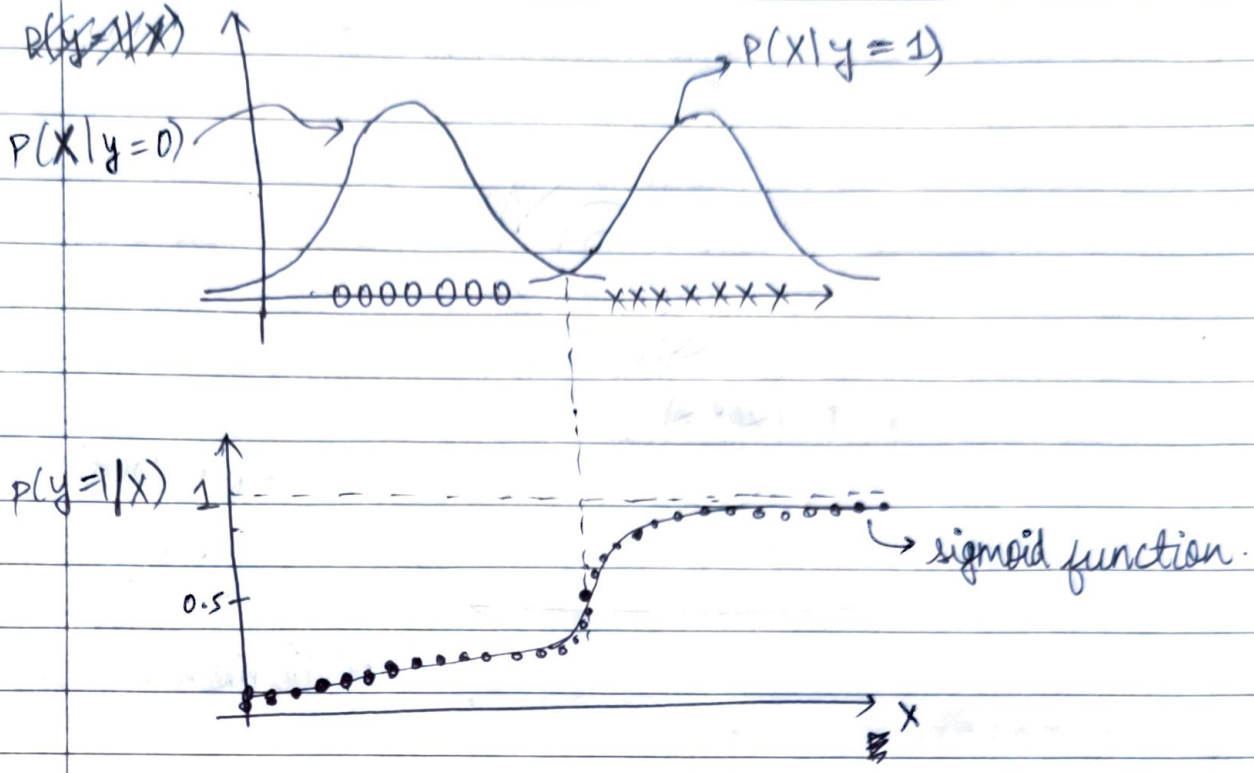
plot $p(y=1|x; \phi, \mu_0, \mu_1, \Sigma)$ as a function of x.

↝ Bernoulli prop.

$$= \frac{p(x|y=1;\mu_1) \; p(y=1;\phi)}{p(x;\phi,\mu_0,\mu_1,\Sigma)}$$

↝ case where we only have 1 feature





when is GDA superior or inferior to Logistic regression:

Logistic regression

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

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

$$y \sim \text{Ber}(\phi)$$

LHS

implies.

$$p(y=1|x) = \frac{1}{1+e^{-\theta^T x}}$$

("$x_0 = 1$")

Logistic RHS

LHS $\Rightarrow$ RHS.

LHS $\not\Leftarrow$ RHS

So, GDA makes stronger set of assumptions

Logistic regr. makes weaker set of assumptions

GDA is a better model if we are given LHS condition.

GDA is also computationally efficient.

let's say;

$$\left.\begin{array}{l} x|y=1 \sim \text{poisson}(\lambda_1) \\ x|y=0 \sim \text{poisson}(\lambda_0) \\ y \sim \text{bernoulli}(\phi) \end{array}\right\} \Rightarrow p(y=1|x) \text{ is logistic.}$$

LHS RHS

LHS ⇒ RHS //

Logistic regression is a rough fit if we don't know if our data is poisson or gaussian.

Naive Bayes Learning algorithm :-

This is our first intro to NLP.
we have been given email text and we want to classify it as spam or not spam.

what will be the feature vector?

$$X = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

a
aardvard
⋮
buy
⋮
cs 229
⋮
zymurgy

→ pick the 10,000 most occuring words from your email.

↳ binary feature vector $\quad X \in \{0,1\}^n$

$X_i = 1$ { word i appears in email }

in naive bayes we finally want;

$p(y|X)$ so we'll need to model $p(X|y)$ and $p(y)$

$2^{10,000}$ possible values of X → not possible ✗

Assume X_i's are conditionally independent given y

$$p(X_1, X_2 \dots, X_{10000}|y) = P(X_1|y) \cdot P(X_2|X_1, y) \cdot P(X_3|X_1, X_2, y) \dots P(X_{10000}| \dots)$$

nothing assumed yet
this is chain rule of prob.

we here assume that

LHS $= p(X_1|y) \cdot p(X_2|y) \cdot p(X_3|y) \dots p(X_{10000}|y)$.

as we assume that they are independent of each other

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

parameters: if a spam email → what is chance of j appearing

$$\phi_{j|y=1} = p(x_j = 1 | y = 1)$$

↝ if not a spam → what is chance of j appearing.

$$\phi_{j|y=0} = p(x_j = 1 | y = 0)$$

$$\phi_y = p(y = 1)$$ ↝ prob. that next email you recieve is a spam email

Joint likelihood:

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

maxm. likelihood estimates (MLE):

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

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

↝ find all spam containing jth word

↝ no. of spam

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

proof in lecture notes 2

at prediction time

$$p(y=1|x) = \frac{p(x|y=1)\,p(y=1)}{p(x|y=0)p(y=0) + p(x|y=1)\cdot p(y=1)}$$

This algorithm almost works but raises a very important problem.

↳ what happens when we get an email containing a new word or a less occured word.

↳ What do we take the prob. as

↳ eg:→ x_{6017} = "NIPS" which we don't get in mails that often

$\phi_{6017|y=1}$ $\Rightarrow p(x_{6017}=1|y=1) = \frac{0}{\#\{y=1\}}$

$\phi_{6017|y=0}$ $p(x_{6017}=1|y=0) = \frac{0}{\#\{y=0\}}$

bad idea to assume it is zero if we haven't seen it

so $p(x|y=1) = \prod_{i=1}^{m} p(x_i|y)$ ↝ becomes zero so gives an error.

So, what do we take our new probability to be?

we use **Laplace's smoothing**

Laplace's smoothing

Let's say a football team plays 5 games against A, B, C, D, E

	win / lost (x)	0 → lost, 1 → won
A	0	
B	0	
C	0	
D	0	
E	2 / 5	what is the prob of them winning E.

$$P(x=1) = \frac{\#\text{"1"s}}{\#\text{"1"s} + \#\text{"0"s}}$$

$$= \frac{0}{4} = 0 \quad \text{✗ not a good estimate.}$$

what Laplace smoothing does?

↳ imagine #"1"s was #"1"s +1
and #"0"s was #"0"s +1

$$P(x=1) = \frac{(\#\text{"1"s} + 1)}{(\#\text{"1"s} + 1) + (\#\text{"0"s} + 1)} = \frac{1}{6} \quad \text{new}$$

optimal estimate

more generally

$x \in \{1, \ldots k\}$

estimate

$$P(x=j) = \frac{\sum_{i=1}^{M} 1\{x^{(i)} = j\} + 1}{M + k}$$

we applied laplace smoothing

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

Laplace smoothing

j can be 0 or 1.

New representation of Naive Bayes :→

Multivariate Bernoulli event model

Old: index

0	a ← 1
0	aardvark ← 2
⋮	⋮
1	buy ← 8000
⋮	
1	drugs ← 1600
⋮	
1	now ← 6200

Drugs: buy drugs now

above is the conventional representation

we lose the # time a word occurs

drugs is stored as 1 only

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

New: Multinomial event model representation

$$x = \begin{bmatrix} 1600 \\ 800 \\ 1600 \\ 6200 \end{bmatrix} \in \mathbb{R}^{n_i}$$

$n \rightarrow$ no. of words in our email

$x_j \in \{1, 2 \ldots 10000\}$

n_i is length of our email

$$P(x, y) = p(x|y) \cdot p(y)$$

$$= \prod_{j=1}^{n} p(x_j|y) \cdot p(y)$$

$$\phi_y = P(y = 1)$$

$$\phi_{k|y=0} = P(x_j = k | y = 0)$$

chance of word jth being k if $y = 0$

$$\phi_{k|y=1} = p(x_j = k \mid y = 1)$$

MLE:

$$\phi_{k|y=0} = \frac{\sum_{i=0}^{m} 1\{y^{(i)}=0\} \sum_{j=1}^{n_i} 1\{x_j^{(i)}=k\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)}=0\} \cdot n_i + 10000}$$

↑ for laplace smoothing

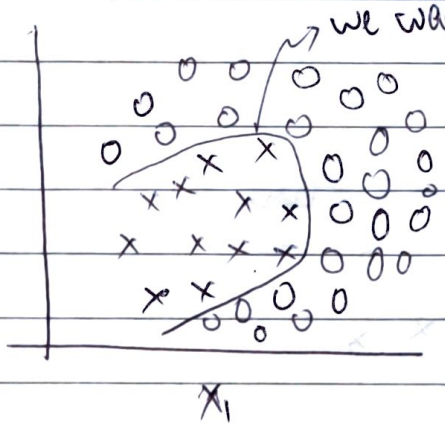
→ no. of outcomes possible

look at all words in all non spam emails

↳ what fraction of them have x_j ~~so~~ occurring in them.

↳ also takes repetition into account in a mail

Support vector machine



we want to get a non linear decision boundary

↓

can't use ~~no~~ basic logistic regression

feature vector given ←

x_1
x_2

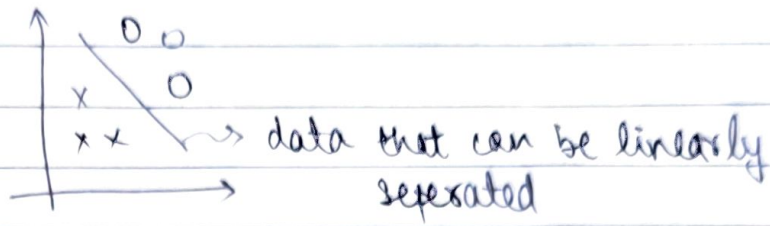
$$\phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$$

→ we map our original feature vector to ~~a~~ a higher dimensional feature vector

↓

then we apply logistic regression for this new vector to get non linear decision boundary.

Optimal margin classifier (seperable case)

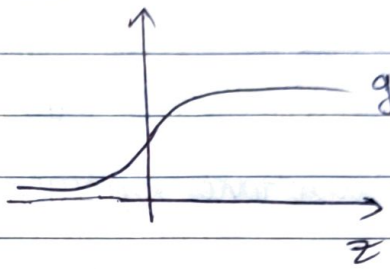


functional margin of a classifier → how well/confidently we classify an example

$h_\theta(x) = g(\theta^T x)$

predict "1" $\theta^T x > 0$

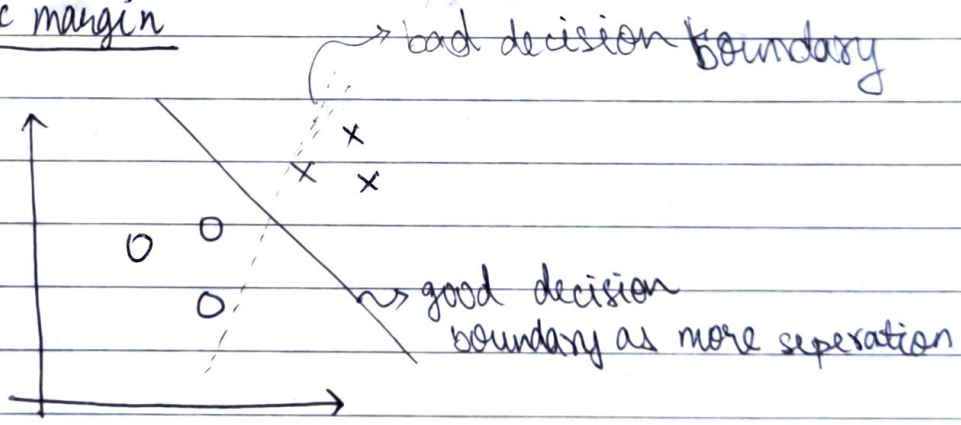
"0" otherwise



If ~~y=1~~ $y^{(i)} = 1$ ~~h~~ we want $\theta^T x \gg 0$

If $y^{(i)} = 0$ we want $\theta^T x \ll 0$

Geometric margin



Notations for SVM

labels $y \in \{-1, +1\}$

h outputs $\{-1, +1\}$.

$g(z) = 1$ if $z \geq 0$

-1 otherwise

previously for LR

$$h_\theta(x) = g(\theta^T x)$$

$\hookrightarrow \mathbb{R}^{n+1}$, $x_0 = 1$.

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

$\hookrightarrow \mathbb{R}^n$ $\hookrightarrow \mathbb{R}$ drop $x_0 = 1$ convention.

Function margin of (w,b) wrt $(x^{(i)}, y^{(i)})$

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$ ↳ for one training sample

read as "gamma head" ←

if $y^{(i)} = 1$ we want $w^T x^{(i)} + b >> 0$

if $y^{(i)} = -1$ we want $w^T x^{(i)} + b << 0$

∴ we always want $\hat{\gamma}^{(i)} >> 0$.

if $\hat{\gamma}^{(i)} > 0$, means $h(x^{(i)}) = y^{(i)}$

functional margin wrt training set.

$$\hat{\gamma} = \min \hat{\gamma}^{(i)} \quad i = 1, 2, \ldots m.$$

$\hat{\gamma}$ ↳ for training set.

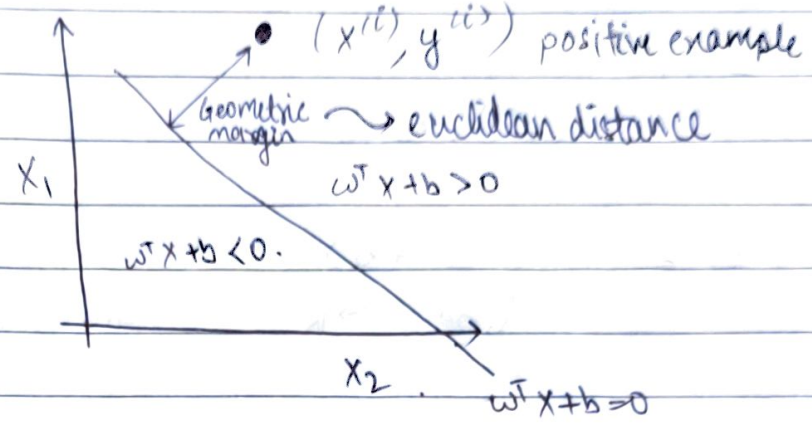
$\min \hat{\gamma}^{(i)}$ ↳ pick minimum from all training cases.

we want $\|w\| = 1$ ⇝ to normalise the ~~length of~~ functional margin

$(w,b) \rightarrow \left(\frac{w}{\|w\|}, \frac{b}{\|w\|}\right)$ ↓ because suppose if we multiply all param by 2.

functional margin doubled tho ← decision boundary doesn't change

Geometric margin :



Geometric margin of hyperplane (w, b) wrt $(x^{(i)}, y^{(i)})$

no cap

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

$$\gamma^{(i)} = \frac{\hat{\gamma}^{(i)}}{\|w\|}$$

$\hat{\gamma}$ → functional
γ → geometric

Geometric margin wrt training set.

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

Optimal margin classifier: → choose w, b to maximise γ

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

$$\text{s.t. } \frac{y^{(i)}(w^T x^{(i)} + b)}{\|w\|} \geq \gamma \quad i = 1, 2, \ldots m$$

optimisation prob ↑

LHS can be rewritten as ↓

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

$$\text{s.t } y^{(i)}(w^T x^{(i)} + b) \geq 1$$

↓ "convex" optimisation problem
↳ solvable easily.

derivation:→

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

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

so, we want to find a value γ such that it is always less than $\gamma^{(i)}$ and we want to maximise this value

$$\max_{\gamma,w,b} \gamma \quad \text{s.t.} \quad \frac{y^{(i)}(w^T x^{(i)} + b)}{\|w\|} \geq \gamma \quad \text{for all } i$$

example:→ geometric margins are → 17
→ 2
→ 5

γ will be 2 here
↓
max value for γ such that $\gamma \leq \gamma^{(i)}$

We can scale w & b by any factor and decision boundary remains the same

let $\|w\| = \frac{1}{\gamma}$

we want

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

$$y^{(i)}(w^T x^{(i)} + b)\,\gamma \geq \gamma \quad \text{for all } i$$

$$\max_{\gamma,w,b} \gamma \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 \Rightarrow \min_{\gamma,w,b} \|w\| \quad y^{(i)}(w^T x^{(i)} + b) \geq 1$$

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1$$

- Optimal margin classifier only works when $x^{(i)} \in \mathbb{R}^{100}$
  ↳ finite

what if our dataset has 100 trillion or infinite ~~not~~ features
↳ use SVM

assume that,

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

→ w can be represented as a linear combination of the features.

↓

this assumption turns out to be the optimum value. → proof in lecture notes

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

$$\min_{w,b} \frac{1}{2}\|w\|^2$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad i = 1, \dots m$$

$$\min \frac{1}{2}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}\right)^T \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}\right)$$

$$= \min \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y^{(i)} y^{(j)} \underbrace{x^{(i)T} \cdot x^{(j)}}_{\langle x^{(i)}, x^{(j)} \rangle}$$

↳ "inner product of vectors $x^{(i)}, x^{(j)}$"

s.t

$$y^{(i)}\left(\sum_j \alpha_j y^{(j)} \langle x^{(i)}, x^{(i)} \rangle + b\right) \geq 1$$ → putting val. of w

can be written as

$$\max \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

↳ "dual optimisation prob."

$$\text{s.t.} \quad \alpha_i \geq 0$$
$$\sum_i y^{(i)} \alpha_i = 0$$

(i) solve for α_i and b
(ii) making prediction from this

$$h_{w,b}(x) = g(w^T x + b) = g\left(\left(\sum_i \alpha_i y^{(i)} \langle x^{(i)}, x \rangle\right) + b\right)$$

we basically need the ability to compute these inner products quickly → kernel helps with this.

Kernel trick:

$\langle x, z \rangle$

1) write algo in terms of $\langle x^{(i)}, x^{(j)} \rangle$
2) Let there be mapping from $x \rightarrow \phi(x)$.

↳ ex. let $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

3) find way to compute

$x \rightarrow \phi(x) \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_1 x_2 \\ x_3 x_4 \\ x_1 x_2 x_3 \end{bmatrix}$

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

4) Replace $\langle x, z \rangle$ in algorithm with $K(x, z)$

Examples of above trick: →

let $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $\quad x \rightarrow \phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$ $\quad z \rightarrow \phi(z) = \begin{bmatrix} z_1 z_1 \\ \vdots \\ z_2 z_1 \\ \vdots \\ z_3 z_1 \\ \vdots \\ z_3 z_3 \end{bmatrix}$

$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$

$x \in \mathbb{R}^n$
$z \in \mathbb{R}^n$

$\phi(x), \phi(z) \in \mathbb{R}^{n^2}$

we need n^2 time to compute $\phi(x)$ or $\phi(x)^T \phi(z)$

$$K(x,z) = \phi(x)^T \phi(z) = (x^T z)^2$$

$x^T z$: $O(n)$; squaring: $O(n)$; overall $O(n)$.

proof :→

$$\left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right)$$

$$= \sum_i \sum_j x_i z_i x_j z_j = \sum_{i=1}^{n}\sum_{j=1}^{n} (x_i x_j)(z_i z_j) = \phi(x)^T \phi(z)$$

suppose our data was

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_3 x_3 \\ \sqrt{2c}\, x_1 \\ \sqrt{2c}\, x_2 \\ \sqrt{2c}\, x_3 \\ c \end{bmatrix} \qquad \phi(z) = \begin{bmatrix} z_1 z_1 \\ \vdots \\ z_3 z_3 \\ \sqrt{2c}\, z_1 \\ \sqrt{2c}\, z_2 \\ \sqrt{2c}\, z_3 \\ c \end{bmatrix}$$

$$K(x,z) = \phi(x)^T \phi(z) = (x^T z + c)^2 .$$

$c \in R$

if $K(x,z) = (x^T z + c)^d$

$\phi(x)$ has all features of ~~degree~~ order $<= d$.

$d = 5 \rightarrow x_1 x_2 x_5 x_6 x_7$ ✓ order $<= 5$

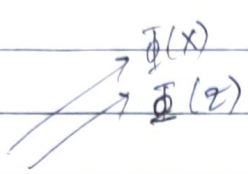
$x_1 x_3 x_2$ ✓ order $<= 5$.

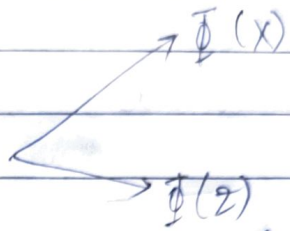
DATE

Support vector machine ⇒ optimal margin classifier + kernel trick

how to make kernels?

If x, z are "similar" $K(x,z) = \phi^T(x)\phi(z)$ is "large"
" " " "dissimilar" $K(x,z)$ is "small"





$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$ is a kernel func. that satisfies above both criterieas

gaussian kernel

we can use the above kernel function only when there exists ϕ s.t.

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

suppose,

$\{x^{(1)}, \ldots, x^{(d)}\}$ be d inputs.

$K \in \mathbb{R}^{d \times d}$ → "kernel matrix"

$K_{ij} = K(x^{(i)}, x^{(j)})$ → kernel function applied to this

K should be "positive semi definite"

↳ necessary and sufficient condition.

given any vector z,

$$z^T K z = \sum_i \sum_j z_i K_{ij} z_j = \sum_i \sum_j z_i \underline{\Phi}(x^{(i)})^T \underline{\Phi}(x^{(j)}) z_j$$

$$= \sum_i \sum_j \sum_k z_i \left(\underline{\Phi}(x^{(i)})\right)_k \left(\underline{\Phi}(x^{(j)})\right)_k z_j$$

$$= \sum_k \sum_i \sum_j z_i \underline{\Phi}(x^{(i)})_k \, z_j \left(\underline{\Phi}(x^{(j)})\right)_k$$

$$= \sum_k \left(\sum_i z_i \underline{\Phi}(x^{(i)})_k\right)^2 \geq 0 .$$

$\therefore \quad K \geq 0 \quad \rightarrow$ positive semi definite.

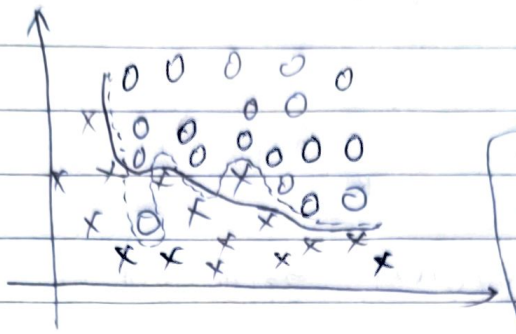
Mercer's Theorem :→ K is valid kernel func. i.e. $\exists \underline{\Phi}$
s.t $K(x, z) = \underline{\Phi}(x)^T \underline{\Phi}(z)$
and for any d inputs $\{x^{(1)}, \ldots x^{(d)}\}$
if and only if $K \geq 0$

Linear kernel → $K(x, z) = x^T z$.
$\underline{\Phi}(x) = x$.

gaussian kernel → $\underline{\Phi}(x) \in \mathbb{R}^\infty$ → meaning can be of ∞ order

—X—

Sometimes we have noisy data that we dont have to try too hard to seperate



— → our boundary/decision because data is noisy

---- → what $\min_{\frac{1}{2}} \|w\|^2$ gives us

↳ optimal margin classifier

We use "L_1 ~~soft~~ norm soft margin sum" for this

instead of $\min \frac{1}{2}\|w\|^2$

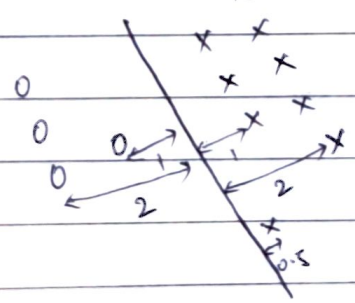
$$\text{s.t. } \underbrace{y^{(i)}(w^T x^{(i)} + b)}_{\text{functional margin}} \geq 1$$

for L_1 norm soft margin sum we take

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \quad \text{"psi"}$$

$$\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \quad i = 1, \ldots, m$$

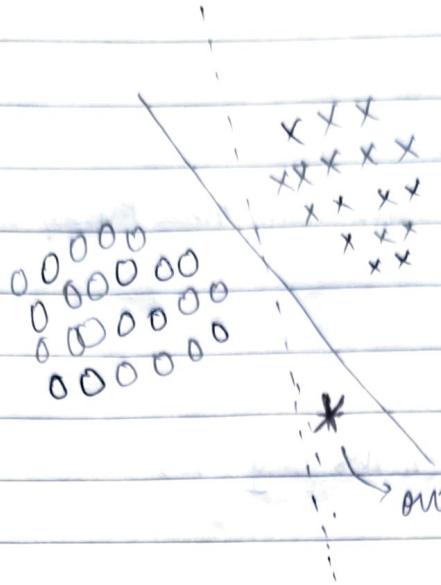
$$\xi_i \geq 0$$



~~according to~~ L_1 norm

↓

lets us get away with functional margins less than 1.



outlier caused a huge change in decision boundary

↓

L_1 norm helps with this and helps keep decision boundary close to — instead of ---

Our new eqn. for L_1 norm becomes

$$\max \sum_{i=1}^{m} \alpha_i - \sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t.} \quad \sum_{i=1}^{m} y^{(i)} \alpha_i = 0$$

$$\text{s.t.} \quad \boxed{0 \le \alpha_i \le C \quad i = 1, \ldots, m} \rightarrow \text{only change to prev eqn.}$$

same eqn.
change in condn.

protien sequence classifier · using SVM

A, B, C · · · , Z → amino acids

BA I SIAIBAJTAV · · · ·

$\mathbb{I}(x) = ?$ $\quad X \mapsto Y$

we make all sequences using 4 amino acids

feature is how many times this seq occurs

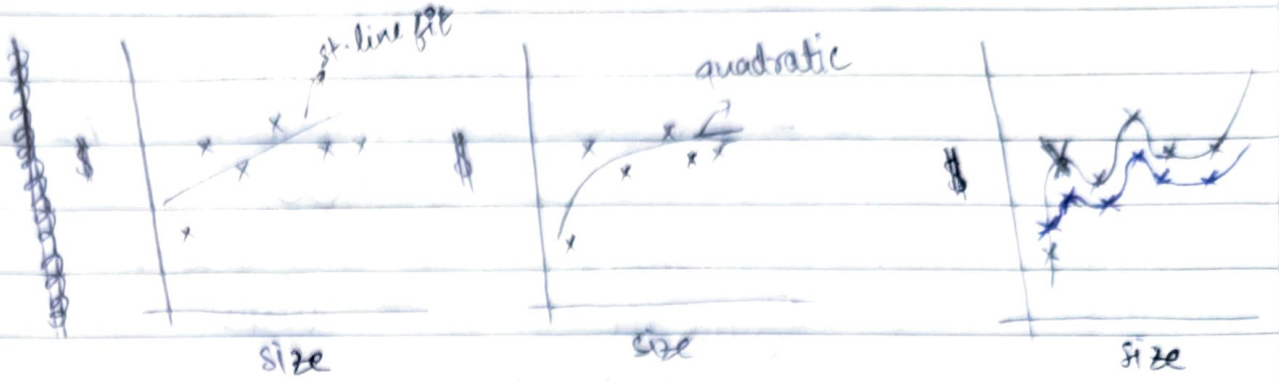
AAAA	0
AAAB	0
⋮	⋮
AAAAZ	0
AABA	0
AABB	0
⋮	⋮
AABZ	0
⋮	⋮
BAJT	2
⋮	⋮
TSIA	1
⋮	⋮
ZZZZ	0

26^4 entries $= \Phi(X)$

BAJTSIAIBAJTAV ...

$\Phi(X)^T \Phi(Z) = K(X,Z)$ → using a dynamic programming algorithm

bias / variance in ML models



$\theta_0 + \theta_1 x$ $\theta_0 + \theta_1 x + \theta_2 x^2$ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \ldots + \theta_5 x^5$

"just right" for the example

"Underfits" the data
"High-bias"

doesn't capture the trend

This learning algo had a strong preconception that data could be fit using linear algorithm which isn't true

"Overfits" the data
"High variance"

we get accurate values for the 6 values

↓

Suppose data was slightly different in blue

↓

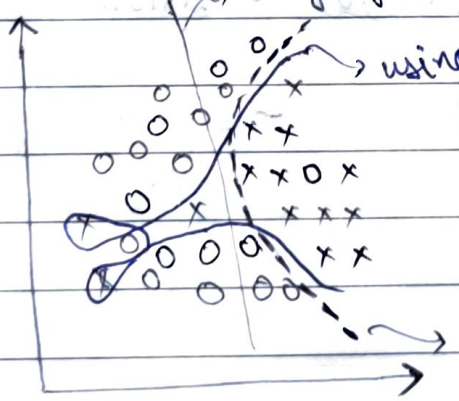
The model fits a new variant function

↓

predictions will have high variance

high bias
↑
underfitting
↑
using logistic regression

using SVM
↳ overfitting
↓
high variance



somewhere in between

↓

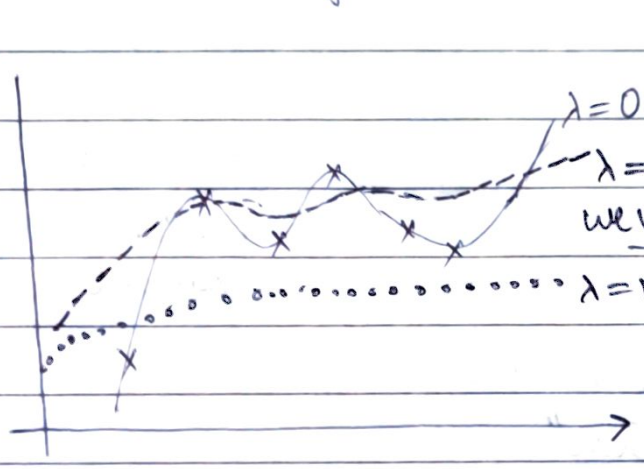
"Just right"

Regularisation

$$\min_{\theta} \left(\left(\frac{1}{2} \sum_{i=1}^{m} \| y^{(i)} - \theta^T x^{(i)} \|^2 \right) + \left(\frac{\lambda}{2} \|\theta\|^2 \right) \right)$$

"/2 sometimes taken to help with derivation"

- $\left(\frac{1}{2} \sum_{i=1}^{m} \| y^{(i)} - \theta^T x^{(i)} \|^2 \right)$ → optimisation objective for linear regression
- $\left(\frac{\lambda}{2} \|\theta\|^2 \right)$ → regularisation term



in overfitting ← ||θ|| is high

for logistic regression :→

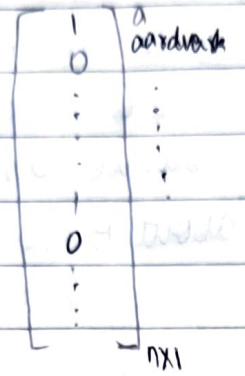
$$\arg\max_{\theta} \sum_{i=1}^{n} \log p(y^{(i)} | x^{(i)}; \theta) - \lambda \|\theta\|^2$$

'−' sign as $\max_{\theta}$ instead of min

example :→

Text classification

$m = 100$
$n = 10{,}000$
} we generally use logistic regression when n is a similar order to n

$X =$ 

we can use logistic regression for this classification problem
↳ but it overfits the data //
↓
However logistic regression with regularisation gives a good fit for our data

another way to think of regularisation

$S = \{x^{(i)}, y^{(i)}\}_{i=1}^{m}$ $S \rightarrow$ ~~same~~ training set

we need to find θ given training set

$$p(\theta|S) = \frac{p(S|\theta) \cdot p(\theta)}{p(S)}$$

$$\underset{\theta}{\text{argmax}}\ p(\theta|S) = \underset{\theta}{\text{argmax}}\ p(S|\theta) \cdot p(\theta).$$

we get the regularisation technique we discussed

solving for this using complex methods

$$= \underset{\theta}{\text{argmax}} \left(\prod_{i=1}^{m} p(y^{(i)} | x^{(i)}, \theta) \right) p(\theta).$$

↳ GLM model

$$p(\theta) = \frac{1}{\sqrt{2\pi}^{n} |\tau^2 I|^{1/2}} \exp\left(-\frac{1}{2}\theta^T (\tau^2 I)\theta\right)$$

$p(\theta)_{\frac{1}{2}}$: $\theta \sim \mathcal{N}(0, \tau^2 I)$ → "assume" $p(\theta)$ is gaussian.

schools of statistics.

Frequentist

we determine θ that makes our data as likely as possible

i.e. maximise $p(s|\theta)$.

"Maximum likelihood Estimate"

Bayesian

θ is unknown

but we have a preconception about θ and how it is generated in the real world

This preconception is captured by a probability distribution

$\leftarrow p(\theta) \rightarrow$ prior distribution

ex:→

$p(\theta)$

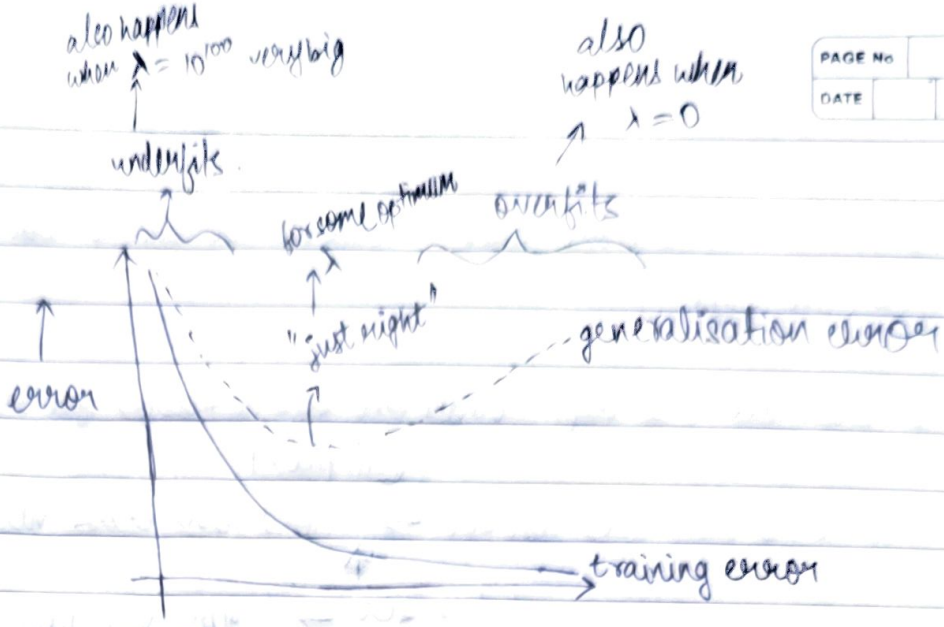
s.t. $\theta \sim N(0, \tau^2 I)$

is a reasonable practice mostly.

goal is to find theta that is most likely upon seeing the data.

find θ s.t. $\max_{\theta} p(\theta|s)$

known as maximum a posteriori [MAP] estimation



model complexity →
(eg:→ deg. of polynomial)

higher degree fits training set better

how to determine "just right" λ?

→ take your dataset and split it into → train, dev, test } 3 sets

eg :→ 10,000 examples.
and we are trying to do model selection
i.e.

- determine optimum polynomial degree
- choose λ for regularisation
- choose τ for locally weighted regression
- choose C for SVM → $\min \frac{1}{2}\|w\| - C \sum_i \xi_i$

so we split S as

S → dataset.

S → S_{train}, S_{dev}, S_{test}

Train each model on S_{train} i (options on ~~what we want to optimise~~ degree of ~~polynomial~~)
↳ get some hypothesis h_i → hypothesis for ith model

→ measure error on S_{dev}. Pick model with lowest error on S_{dev}

→ Evaluate the algorithm on set (S_{test}) for report.

what we did with dev/test set ⇒ [(simple)] "Hold-out cross validation"

dev set → "cross validation set"

how to split data?

Traditional methods → good for small datasets (~500 to thousands)

S → 70% train, 30% test. } good when we have picked model and want to revise the model

S → 60% train, 20% dev, 20% test } model selection

If S is very large ~ 10 million or so

↓

% of dev and test gets very less

↓

choose big enough dev and test s.t. you can make out meaningful differences between accuracy of algorithms.

eg:

90% } → can be distinguished by say 1000 examples in dev and test.
95%

90.1% } → If this is meaningful to our project we need to increase % allocation to dev and test
90%

↓

cannot be distinguished by 1000 examples in dev & test

Suppose we have a small dataset of 100 inputs → 70 S_{train}, 30 S_{dev}

not a good way of splitting as we lose critical data by splitting like this

So we use the **k-fold cross validation**:→

S

$x^{(1)}, y^{(1)}$

⋮

$x^{(100)}, y^{(100)}$

we split the dataset into $k=5$ in this example but $k=10$ is the norm

for each group we train on $k-1$ inputs and test on the kth input then take the average the error estimates.

For $i = 1, \ldots k$

{

Train (fit param) on $k-1$ pieces

test on remaining 1 piece }

Average

if $m = 20$ → very small dataset

↓

"Leave one out cross validation"

↓

we do our training on $m-1$ inputs and test on one input and we do this m times leaving each input exactly once and averaging over m.

"Feature selection" → find a small subset of features that are most useful → "takes judgement"

feature selection → start with $\mathcal{F} = \emptyset$

repeat :

1) Try adding each feature i to $\mathcal{F}$ and see which single feature addition ~~makes~~ improves dev set performance

2) Add that feature to $\mathcal{F}$

this procedure is known as "forward search"

↓

we start empty and keep adding features.

Learning theory

Assumptions :→

1) $\exists$ data distribution D

$(x, y) \sim D$

i.e. all examples / inputs we get in our training set is from a data distribution.

2) Independent samples

$\hat{} \rightarrow$ estimate

$* \rightarrow$ true

in this model $\exists \theta^*$ or h^* which is "True param"

$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
$(x^{(3)}, y^{(3)})$
$\vdots$
$(x^{(m)}, y^{(m)})$

$\sim D \Rightarrow$ [learning algo] $\rightarrow \hat{h}$ or $\hat{\theta}$ → "sampling distribution"

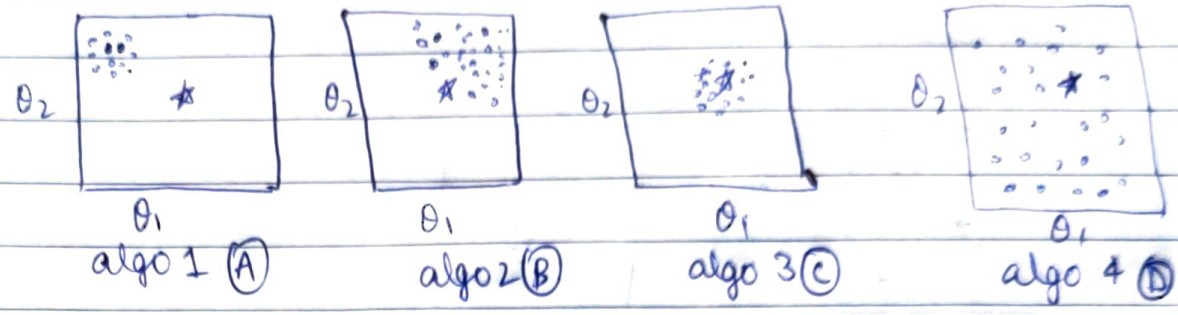
learning algo → deterministic function

$\hat{h}$ or $\hat{\theta}$ → Random variable

$S \rightarrow$ Random variable

Let's say we have 4 different learning algos. with θ_1 θ_2 as features

we make the following parameter view



★ → true θ

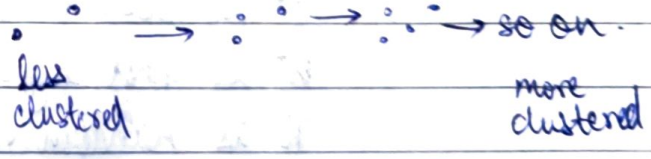
we run above algos for multiple samples to obtain different $\hat{\theta}$

(C) and (D) → low bias → "bias" means if it is centred around true parameter

(A) and (C) → low variance → "variance" measures how dispersed the parameters are

★ as $m \to \infty \Rightarrow \text{var}[\hat{\theta}] \to 0$ as for sufficiently large no. of inputs data will be more clustered

rate at which $\text{var}[\hat{\theta}]$ decreases is called "Statistical efficiency"



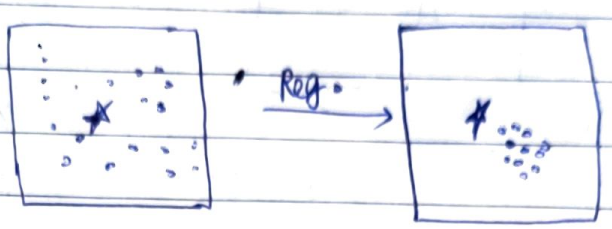
★ if $\hat{\theta} \to \theta^*$ as $m \to \infty$ then algo is Consistent algorithm

★ if $E[\hat{\theta}] = \theta^*$ regardless of m → the algorithm is called unbiased algo.

reducing variance

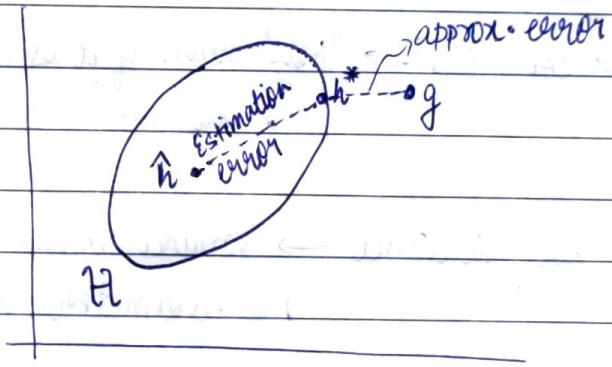
(i) $n \to \infty$ (i.e get more data)

(ii) regularisation



low bias
high variance

small bias → slightly increased
low variance



space of hypothesis

$\mathcal{H}$ is the set of hypothesis
g^* is best possible hypothesis
h^* is best hypo in class $\mathcal{H}$
$\hat{h}$ is random variable learnt from finite data

$\mathcal{E}(h)$: Risk / Generalisation error $= E_{(x,y)\sim D}\left[1\{h(x) \neq y\}\right]$

↓ expected number of "non-matches" from data → infinite steps

data dist. used to create samples ← distribution

$\hat{\varepsilon}_S(h)$: Empirical risk

$$= \frac{1}{m} \sum_{i=1}^{m} 1\{h(x^{(i)}) \neq y^{(i)}\}$$

$\varepsilon(g)$: Bayes Error / Irreducible error

$\varepsilon(h^*) - \varepsilon(g)$ = Approximation Error → i.e. price to pay for limiting to a class $\mathcal{H}$

$\varepsilon(\hat{h}) - \varepsilon(h^*)$ = Estimation Error

$\varepsilon(\hat{h})$ = Estimation error + approx. error + Irreducible error.

- Estimation error: because we are working on limited data
- approx. error: depends on chosen class
- Irreducible error: cannot reduce no matter what

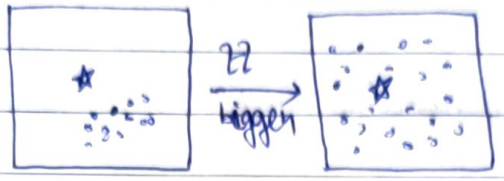
Estimation Error

Estimation Var + Estimation bias + Approx. Error + Irreducible

$\varepsilon(\hat{h})$ = Var + bias + Irreducible

Reducing high bias

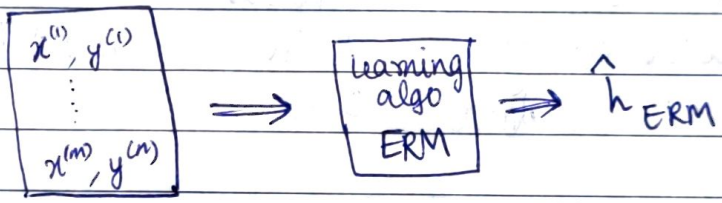
① make $\mathcal{H}$ bigger



high bias
some variance

reduces bias
increases variance

Empirical risk minimisation (ERM) → is a learning algo



S
dataset

$$\hat{h}_{ERM} = \arg\min_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} 1\{h(x^{(i)}) \neq y^{(i)}\}$$

Uniform convergence

① consider a hypothesis with traing error $\hat{\varepsilon}(h)$ then what can we say about it's generalisation error $\varepsilon(h)$? $\hat{\varepsilon}(h)$ vs $\varepsilon(h)$

(random variable; empirical error)

② $\varepsilon(\hat{h})$ vs $\varepsilon(h^*)$ → how does generalisation error of our "learnt hypothesis" compare to best possible generalisation error of our class.

Tools to solve these questions :→

1) Union bound (we have k events)
$A_1, A_2, \ldots A_k$ (need not be independent)

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

2) Hoeffding's Inequality

Let $z_1, z_2 \ldots z_m \sim Bern(\phi)$ we have parameters from bernoulli dist.

$$\hat{\phi} = \frac{1}{m} \sum_{i=1}^{m} z_i$$

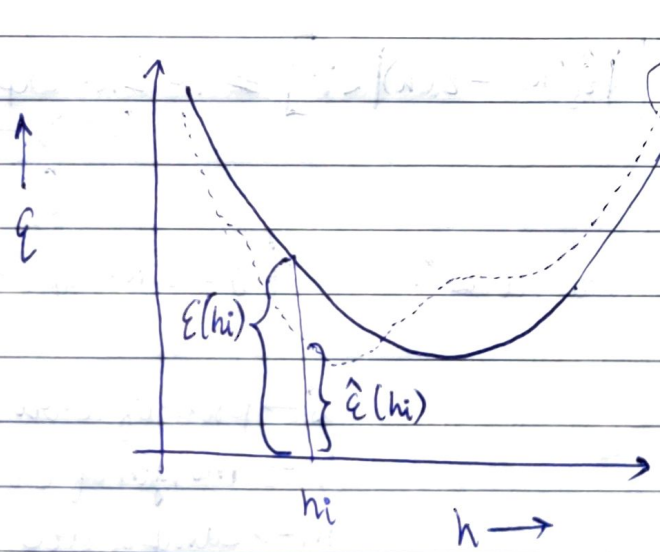
$\hat{\phi}$ is average of these.
↳ "learnt hypothesis"

let $\gamma > 0$ [margin]

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

proof in lecture notes

($\hat{\phi} - \phi$: Error; γ: margin)
↳ prob. that $|\hat{\phi} - \phi| > \gamma$ is RHS



curves don't match as for $\hat{\varepsilon}_S(h)$ we take a sample from our data distribution on the other hand $\varepsilon(h)$ is an infinity step.

$$E\left[\hat{\varepsilon}(h_i)\right] = \varepsilon(h_i)$$

↓

as $\hat{\varepsilon}$ is over some m random inputs and when we keep doing this throughout our dataset, expected value is $\varepsilon(h_i)$

on applying Hoeffding's Ineq.

$$\text{pr}\left[\left|\hat{\varepsilon}(h_i) - \varepsilon(h_i)\right| > \gamma\right] \leq 2\exp(-2\gamma^2 m)$$

* as we increase m, the dotted curve ($\hat{\varepsilon}(h_i)$) gets closer to solid curve ($\varepsilon(h_i)$).

we got above bound for a fixed $\mathcal{H}$
we want bound for all $\mathcal{H}$

Finite hypothesis class $\mathcal{H}$:->

$|\mathcal{H}| = k$ → cardinality

~~$\text{pr}\left[\exists h \in \mathcal{H} \left|\hat{\varepsilon}_s(h) - \varepsilon(h)\right| > \gamma\right] \leq k \cdot 2\exp(-2\gamma^2 m)$~~

prob that atleast one hypothesis from class $\mathcal{H}$ ←

$$\text{pr}\left[\exists h \in \mathcal{H} \left|\left|\hat{\varepsilon}_s(h) - \varepsilon(h)\right| > \gamma\right.\right] \leq k \cdot 2\exp(-2\gamma^2 m)$$

$$\Rightarrow \text{pr}\left[\forall h \in \mathcal{H} \left|\left|\hat{\varepsilon}_s(h) - \varepsilon(h)\right| < \gamma\right.\right] \geq 1 - \underbrace{2k \cdot \exp(-2\gamma^2 m)}_{\delta}$$

"proof in lecture notes".

$$\boxed{\delta = 2k \cdot \exp(-2\gamma^2 m)}$$

δ → prob. of error
γ → margin of error
m → sample size

as we incr. m emp. gets closer to generalisation-

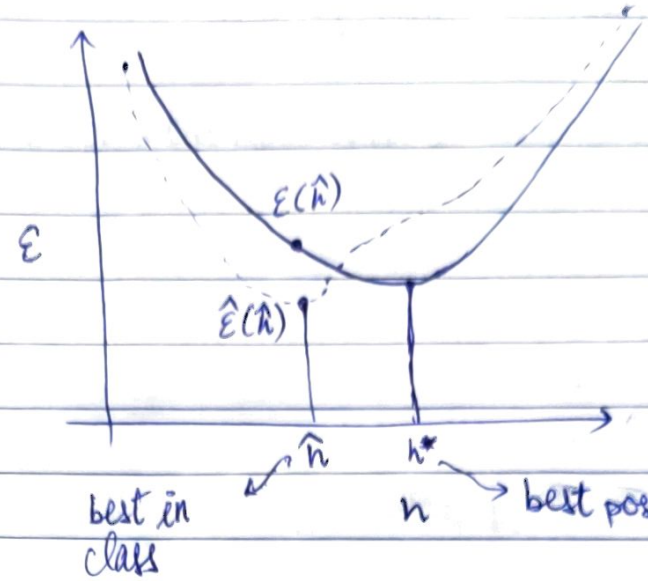
↑

"sample complexity result" ←

fix γ; $\delta > 0$

$$\boxed{m \geq \frac{1}{2\gamma^2}\log\left(\frac{2k}{\delta}\right)}$$

margin of error between empirical and generalisation error will be less than γ if $m \geq$ RHS



relation b/w ĥ and h^*

$$\varepsilon(\hat{h}) \leq \hat{\varepsilon}(\hat{h}) + \gamma$$

$$\varepsilon(\hat{h}) \leq \hat{\varepsilon}(h^*) + \gamma \quad \text{as } \hat{\varepsilon}(\hat{h}) \text{ is minima of } \hat{\varepsilon} \text{ func.}$$

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma \quad \text{as } |\hat{\varepsilon}(h) - \varepsilon(h)| \leq \gamma \text{ is true throughout } h$$

with probability $1-\delta$ and training size m

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\sqrt{\frac{1}{2m} + \log\frac{2k}{\delta}}$$

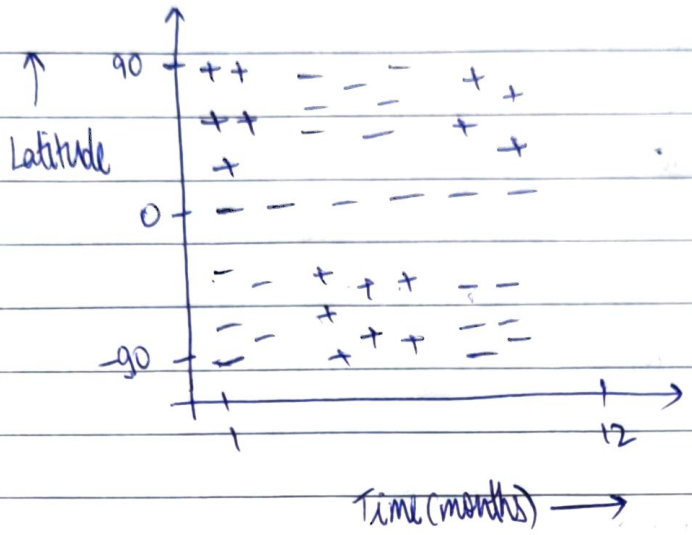
VC dimension → tries to assign size to ∞ size class
VC(H) → can be taken as size of ∞ size class

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

↳ for ∞ size class H.

Decision trees → non linear algo

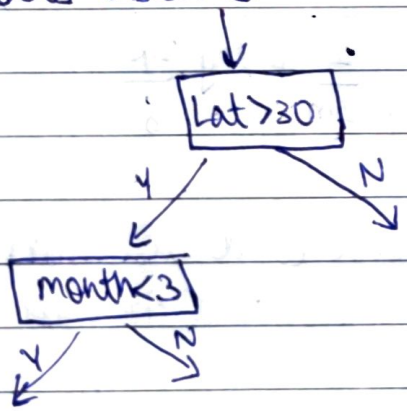
suppose we like skiing, and want to know when and where is it possible



we might be able to use SVM to split this data but decision trees give us a more natural way of handling this problem.

Decision trees → Greedy, top down, recursive partitioning

Basically we ask certain questions to seperate out the data into sets. Then we repeat the steps for the sets obtained.



more formally
given a region R_p
we are tryng to find a
split function S_p s.t.

$$S_p(j, t) = (\{X \mid X_j < t, X \in R_p\}, \{X \mid X_j \geq t, x \in R_p\})$$

j → jth feature of X; R_1, R_2

j is feature number; t threshold

How to choose these splits?

→ Define loss on a region :→ $L(R)$

→ obv example → miss classification loss
~~Assume~~ given C classes, define $\hat{p}_c$ to be proportion of examples in R that are of class c

$L_{misclass} = L(R) = 1 - \max_c \hat{p}_c$ → reasoning :→
we have C classes in our region R
we find the class which has the maximum proportion and we say the rest are misclassified.

we want to pick a split that minimises the loss as much.

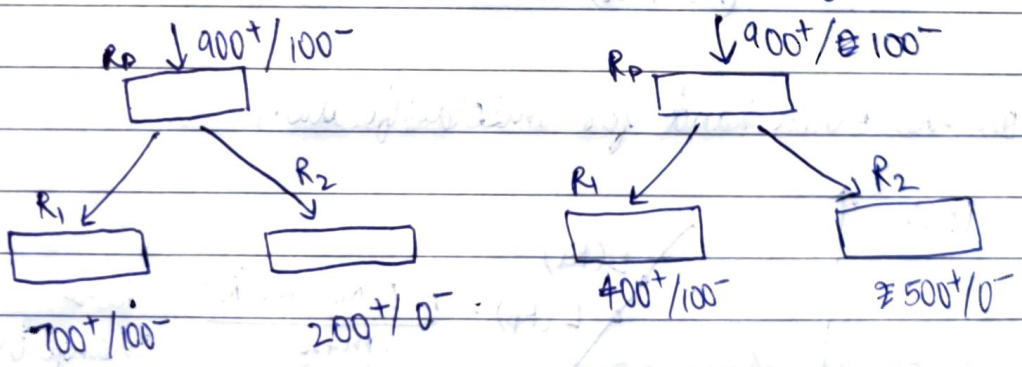
refer notes as to why?

$\leftarrow \max_{j,t} \; L(R_p) - (L(R_1) + L(R_2))$

$L(R_p)$: parent loss
$L(R_1) + L(R_2)$: children loss

we pick the parameters j and t to min loss

misclassification loss has issues (here we are talking about absolute numbers rather than fraction)



$L(R_1) + L(R_2) = 100 + 0 = 100$ $\quad$ $L(R_1) + L(R_2) = 100 + 0 = 100$

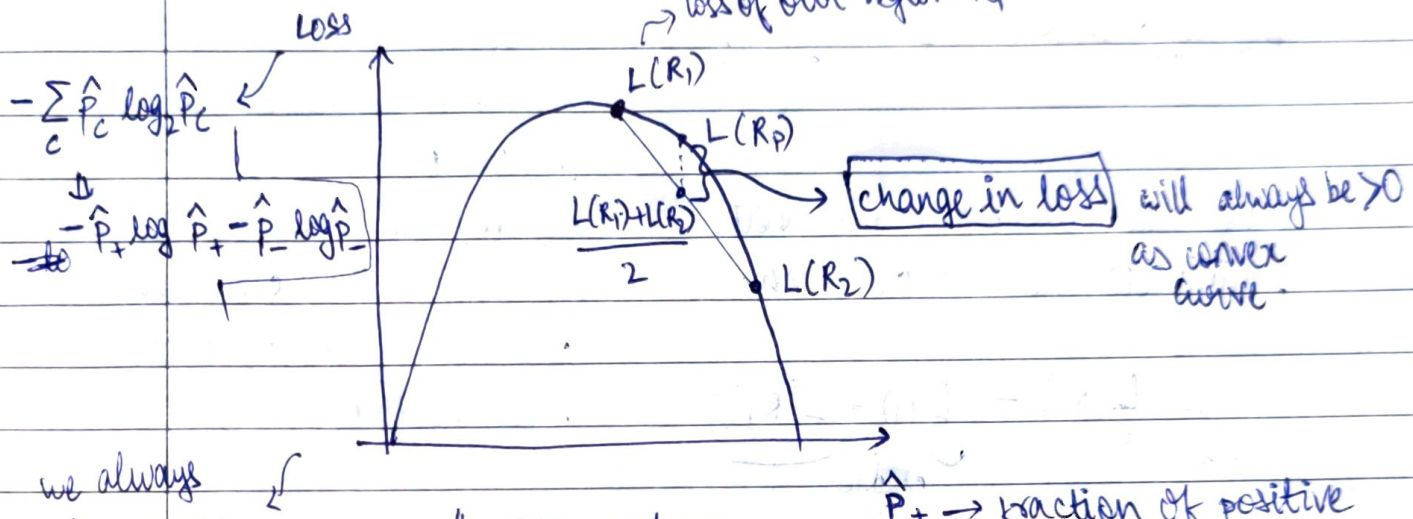
$L(R_p) = 100$

both have same loss but rightone is better → misclass. loss not sensitive enough //

we define a cross entropy loss

$$L_{cross} = -\sum_c \hat{P}_c \log_2 \hat{P}_c$$

suppose we split a region into two classes of equal size

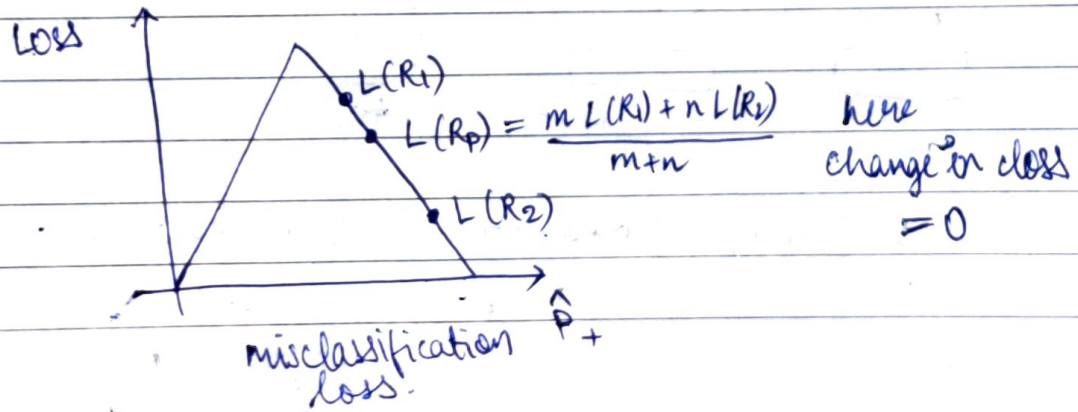


suppose the classes weren't of same size → ~~thrown't spt~~

suppose R_1 has size m and R_2 has size n

$\frac{m\,L(R_1) + n\,L(R_2)}{(m+n)}$ would be our new point

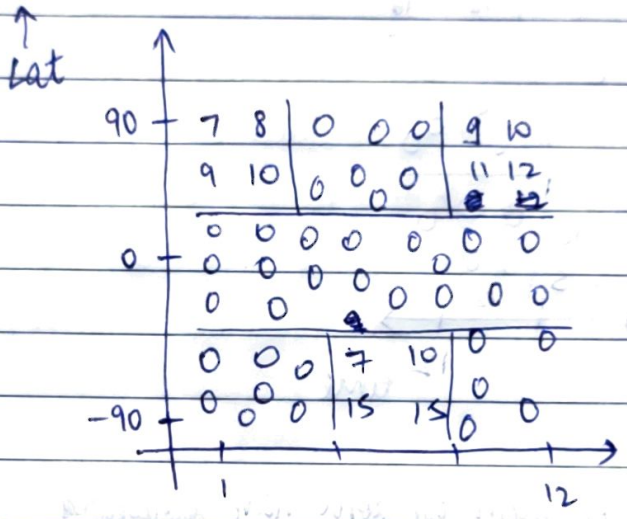
on the other hand for misclassification



Gini-loss = $-\sum_{c} \hat{p}_c (1 - \hat{p}_c)$ → also has a curve similar to cross-entropy loss.

Regression trees

Instead of classification we try regression
eg :→ we predict amount of snowfall rather than whether we can ski or not.



R_M

predict $\hat{y}_M = \frac{\sum_{i \in R_M} (y_i - \hat{y}_M)^2}{}$

we have a region R_M for which we want to predict $\hat{y}_M$

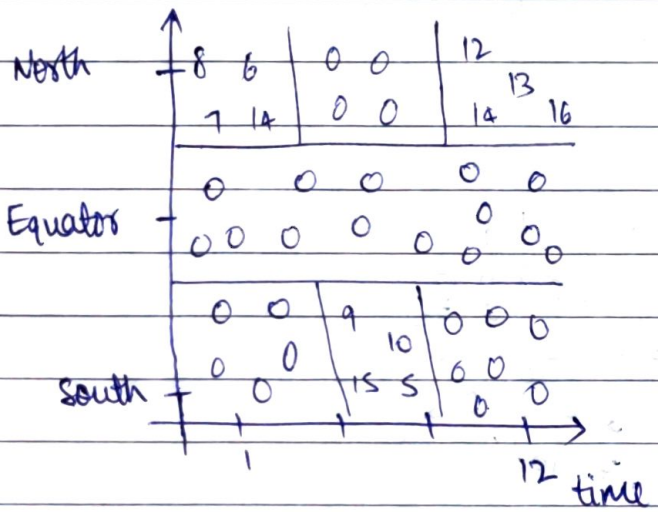
given R_M,

$$\hat{y}_M = \frac{\sum_{i \in R_M} y_i}{|R_M|}$$ → summing values in region and take the average

The loss we take in this case is $L_{squared} = \frac{\sum_{i \in R_M} (y_i - \hat{y}_M)^2}{|R_M|}$

Categorical variables

if we have prev example as



for these kinds of problem we don't have threshold

↓

lat $\in \{N\}$ we ask questions is latitude is northern hemisphere

if we have q categories we deal with 2^q possible split.

If we keep running a decision tree → we end up with each point having its own class → overfitting

so, we need to reduce the variance for a decision by regularization.

Regularization of decision Tree

heuristic meaning → proceeding to solution by trial and error.

we regularize decision trees by using a number of heuristics

(i) min leaf size
(ii) max depth
(iii) max number of nodes
(iv) minimum decrease in loss → generally not a good idea

before split → $L(R_p)$
after split → $L(R_1) + L(R_2)$
we do not pick a split if decrease is ~~big~~

* but sometimes we need to ask multiple questions to get to a good split

(v) grow full tree then ~~prune~~ prune out nodes

↳ pruning done using validation set, evaluate ~~error~~ misclassification error on validation set for each example/leaf we remove.

Runtime

n examples
f features
d depth

- Test time : $O(d)$ typically $d < \log_2 n$

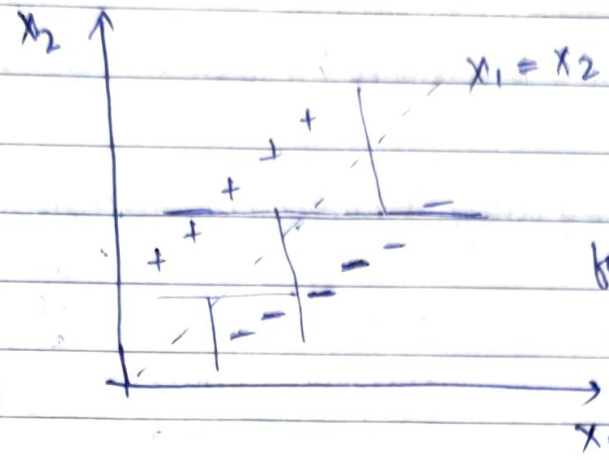
- Traintime : Each point is a part of $O(d)$ nodes. (basically has $O(d)$ parents)
  A ~~Each~~ point has a cost $O(f)$ at each node.
  ↳ consider f splits

after a split a point belongs to the left or the right of it

we have n points

Total cost = $O(nfd)$ //

No additive structure



easy for logistic regression

for decision trees we ~~are~~ need a lot of splits to approximate the $\neq$ split.

Recap

Advantage

- Easy to explain
- Interpretable (can see exactly what is happening)
- Categorical vars (can deal with cat. vars)
- Fast

disadvantage

- High variance
- Bad at additive
- low predictive accuracy.

Ensembling

Take X_i's which are random variables (RV) that are independently identically distributed (iid) (as they are independent.)

$$\text{Var}(X_i) = \sigma^2 \qquad \text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{n}\sum_i X_i\right) = \frac{\sigma^2}{n}$$

$= \left(\frac{1}{n}\right)^2 \text{Var}(\Sigma X_i) = \left(\frac{1}{n}\right)^2 (n\sigma^2)$

↓

variance of one of variables is σ^2 $\Rightarrow$ var. of mean of many variables $= \frac{\sigma^2}{n}$

↳ variance of all variables will be σ^2 and they will have same mean as they are identically distributed.

Drop the independence assumption.

now X_i's are identically distributed

X_i's correlated by ρ

$$\text{var}(\bar{x}) = \rho\sigma^2 + \frac{(1-\rho)}{n}\sigma^2$$

Ways to ensemble

1) use different algorithms

2) different training sets

3) Bagging (Random forests) } commonly used

4) Boosting (Adaboost, XG boost) }

Bagging → Bootstrap aggregation.

Have a true population P
Training set $S \sim P$ (S is sampled from P)

for bootstraping does: Assume P = S (population is training set)

Bootstrap samples $Z \sim S$

we can train our separate models on several bootstrap samples and then average them.

bootstrap samples $Z_1, \ldots, Z_M$

Train model G_m on Z_m

$$G(m) = \frac{\sum_{m=1}^{M} G_m(X)}{M}$$

bias variance analysis on above result

$$\text{Var}(\bar{X}) = \rho\sigma^2 + \frac{(1-\rho)}{M}\sigma^2$$

bootstrapping is reducing down ρ

more M is → less variance as it reduces ρ

the variance is reduced but the bias is increased.

↳ as bootstrap samples have lesser data and the models are thereby less complex. (Random subsampling)

Decision trees & bagging :→

D.T are high variance and low bias

↳ Ideal fit for bagging.

Random forests

At each split → consider only a fraction of your total features.

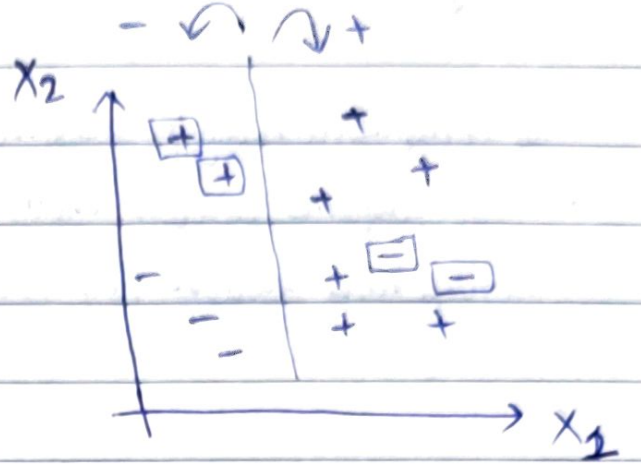
this decreases ρ.

helps decorrelate models. → suppose we have a very good predictor which gives very good performance on its own.

will cause all models to be correlated at first split ← regardless of bootstrap sample we use model will use that predictor as first split

Boosting

Decrease bias
more Additive



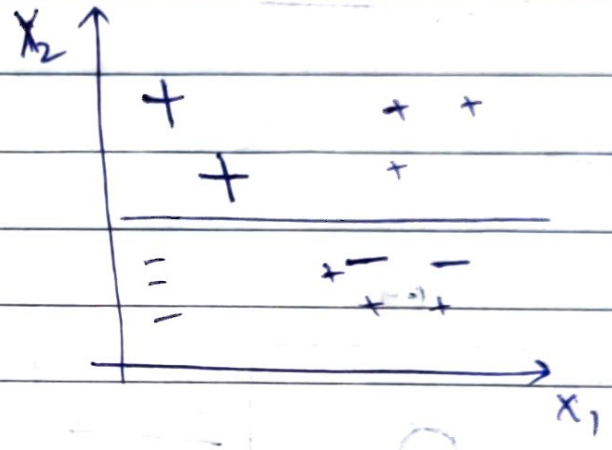
First we ask a question and come up with a decision boundary

↓

we highlight the mistakes and increase their weight.

↓

next model trained on the dataset with previous mistakes having high weight.



Determine for classifier G_m a weight α_m

for adaboost $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$

$$G(X) = \sum_m \alpha_m G_m$$

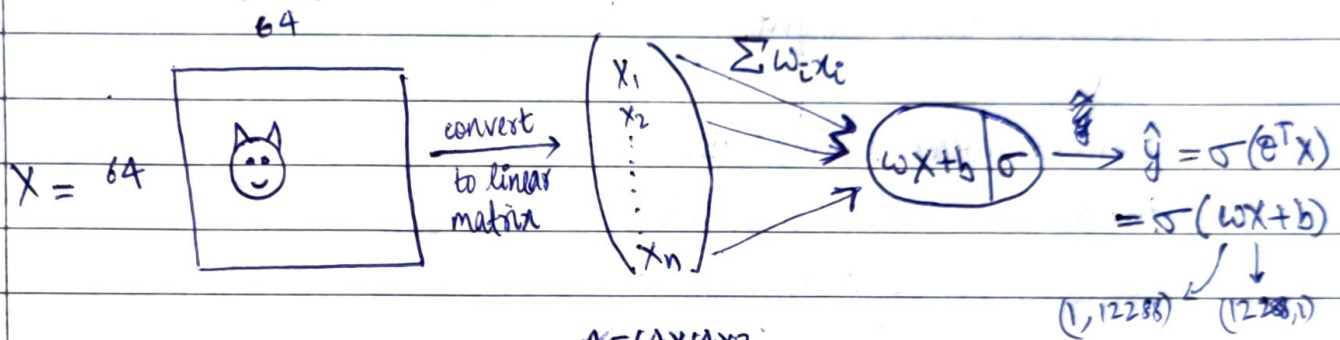
Each G_m trained on reweighted training set.

Deep learning :→

- is computationally expensive
- algo performs better with more data
- algorithms which help reduce the computational power needed

Logistic regression

goal → Find cats in images { 1 → presence, 0 → absence



$n = 64 \times 64 \times 3$

RGB value of each pixel

weights, bias

i) Initialise w, b

ii) Find optimal w, b → minimise the loss function

$$\mathcal{L} = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$$

$$w = w - \alpha \frac{d\mathcal{L}}{dw}$$

$$b = b - \alpha \frac{d\mathcal{L}}{db}$$

iii) use $\hat{y} = \sigma(w+b)$ to predict

neuron = linear + activation

$\sum w_i x_i$; in this case sigmoid function

model = architecture + parameters

the neurons / network

w, b here

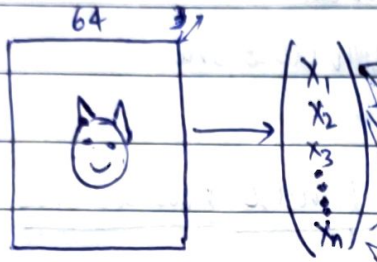
goal 2 → Find cat / lion / iguana in images → we use 3 neurons here

represent layers → where neurons dont communicate with eachother

will the loss function given earlier work for this goal?

↓ NO

as prev 2 has only two outcomes and cant match the labeling



$a_1^{[1]} \rightarrow \hat{y}_1 = a_1^{[1]} = \sigma(w_1^{[1]} X + b_1^{[1]}) = z_1^{[1]}$

$a_2^{[1]} \rightarrow \hat{y}_2 = a_2^{[1]} = \sigma(w_2^{[1]} X + b_2^{[1]}) = z_2^{[1]}$

$a_3^{[1]} \rightarrow \hat{y}_3 = a_3^{[1]} = \sigma(w_3^{[1]} X + b_3^{[1]}) = z_3^{[1]}$

dataset will look like

$\Rightarrow \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ cat, lion, iguana

we train the network as earlier and get all our weights and biases

here output is a 3X1 vector.

→ gives exact same derivative

$$\mathcal{L}_{3N} = -\sum_{k=1}^{3} \left[y_k \log \hat{y}_k + (1 - y_k) \log (1 - \hat{y}_k) \right]$$

suppose we had an input like

$\rightarrow \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$

yes, as the neurons dont communicate amongst themselves.

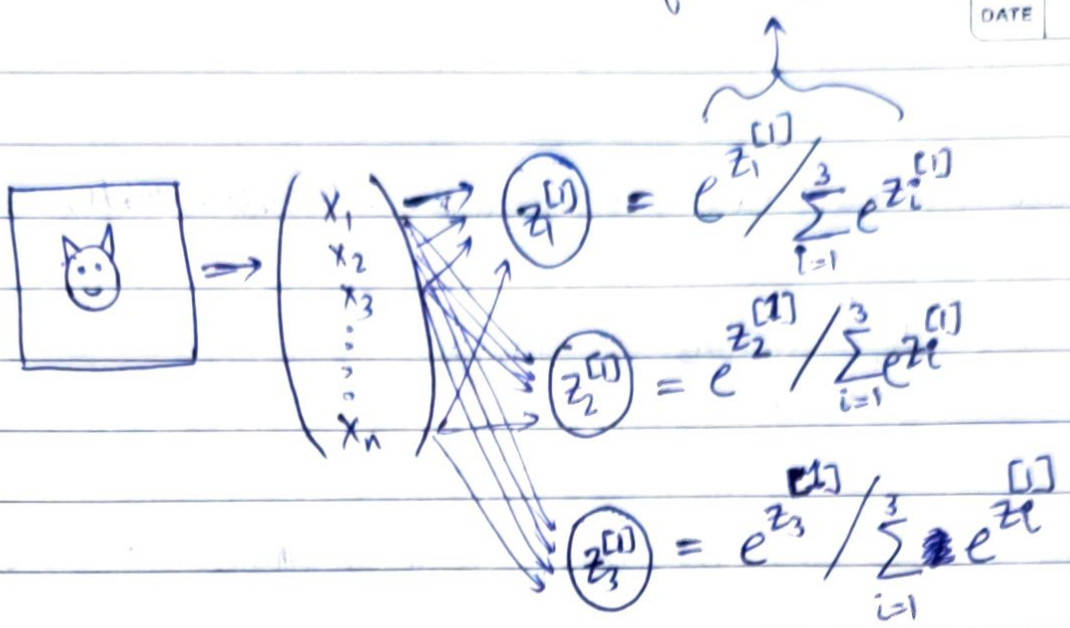
will our neural network be robust for inputs as above?

goal 3 → + constraint

unique animal on an image

(atmost one animal)

softmax formula



now three prob. are dependent on each other as they sum upto one.

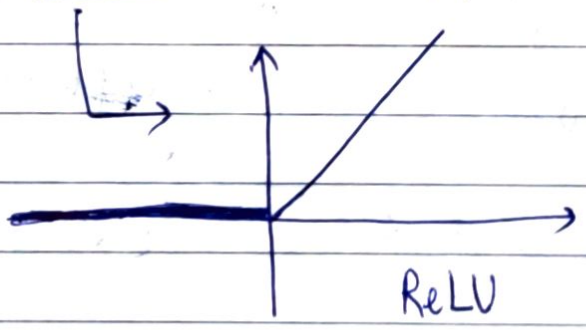
This model is called a <u>softmax multiclass network</u>.

Loss function here is ~~or~~ cross entropy loss

$$\mathcal{L}_{CE} = -\sum_{k=1}^{3} y_k \log \hat{y}_k$$

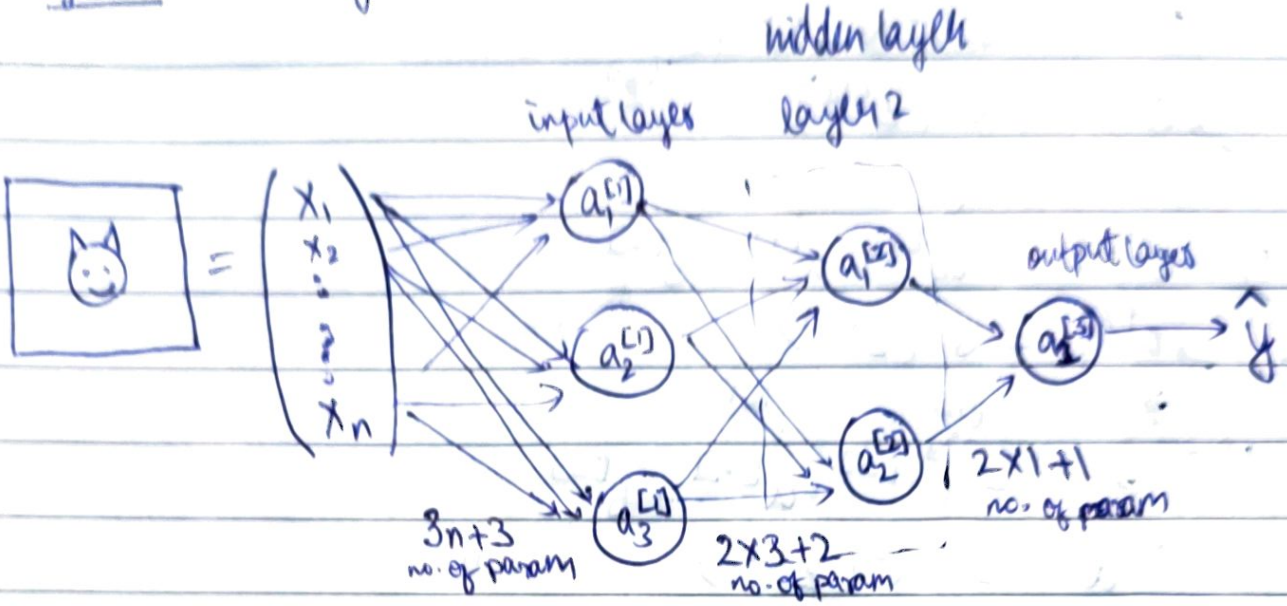
ReLU

~~Relu~~ ~~ReLU~~ <u>function</u> → Rectified Linear Unit



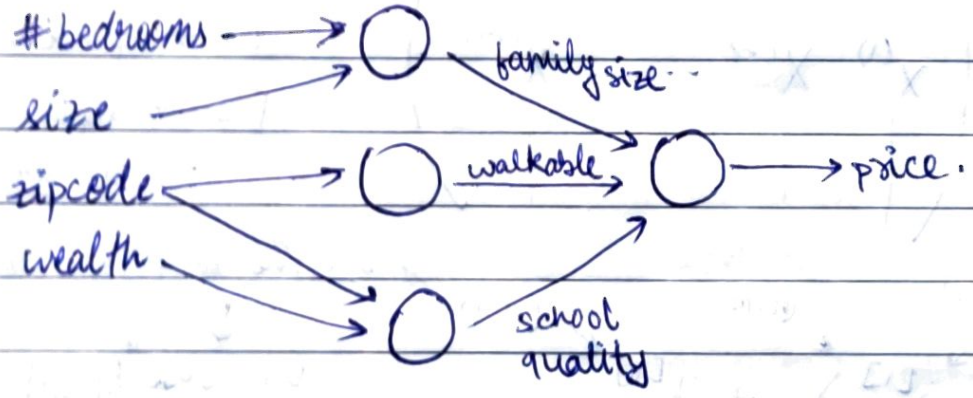
Neural Networks → also called end to end learning
→ also known as blackbox models

goal → image has cat or no cat.



last layer should have as many neurons as the no. of classifications

house price prediction



[now humans would construct a neural network]

↓

In practice we fully connect two layers

← "we let the next layer figure out which neuron is an important feature"

propogation equation

$$z^{[1]} = w^{[1]} x + b^{[1]} \quad (3,n)\cdot(n,1)+(3,1)$$

$$a^{[1]} = \sigma(z^{[1]}) \quad (3,1)$$

$$z^{[2]} = w^{[2]} \cdot a^{[1]} + b^{[2]} \quad (2,3)\cdot(3,1)+(2,1)$$

$$a^{[2]} = \sigma(z^{[2]}) \quad (2,1)$$

$$z^{[3]} = w^{[3]} \cdot a^{[2]} + b^{[3]} \quad (1,2)\cdot(2,1)+(1,1)$$

$$a^{[3]} = \sigma(z^{[3]}) \quad (1,1)$$

What happens for an input batch of m examples

$$X = \begin{pmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{pmatrix}$$

m → inputs

() → represent id of input

[] → represents layer number

$$z^{[1]} = w^{[1]} X + b^{[1]} \quad (3,n)\cdot(n,m)+(3,1)$$

but the dimensions dont add up

↓

we use "BROADCASTING"

$$z^{[1]}_{(3,m)} = \begin{bmatrix} | & & | \\ z^{[1](1)} & \cdots & z^{[1](m)} \\ | & & | \end{bmatrix}$$

$$\tilde{b}^{[1]} = \begin{pmatrix} | & | & & | \\ b^{[1]} & b^{[1]} & \cdots & b^{[1]} \\ | & | & & | \end{pmatrix} \quad \leftrightarrow \text{m times}$$

↓

(3, m)

Optimising $w^{[1]}, w^{[2]}, w^{[3]}, b^{[1]}, b^{[2]}, b^{[3]}$

Define loss / cost function

↙ 1 example ↳ multiple examples in the batch.

$$J(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}^{(i)} \qquad \hat{y} = a^{[3]}$$

with $\mathcal{L}^{(i)} = -\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})\right]$

Backward propogation

$$\forall l = 1, \dots 3 \begin{cases} w^{[l]} = w^{[l]} - \alpha \frac{\partial J}{\partial w^{[l]}} \\ b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}} \end{cases}$$

we start this process from $w^{[3]}$

↳ as it has a direct impact on the output

↓

easier to see how $w^{[3]}$ changes impact output than how $w^{[1]}$ changes affect output

$$\frac{\partial J}{\partial w^{[3]}} = \frac{\partial J}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial w^{[3]}}$$

$$\frac{\partial J}{\partial w^{[2]}} = \left(\frac{\partial J}{\partial z^{[3]}}\right) \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[1]}}$$

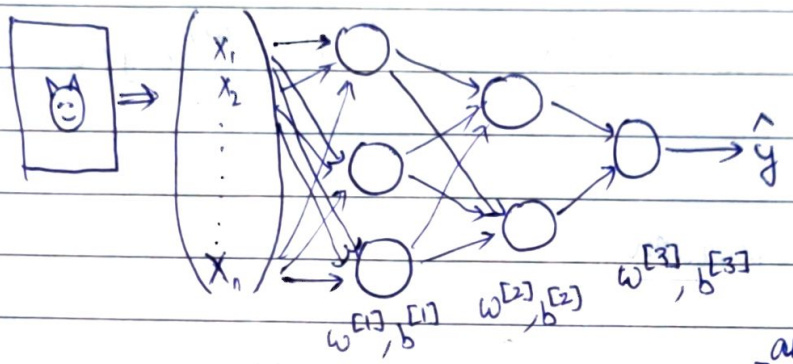
$$\frac{\partial J}{\partial w^{[1]}} = \left(\frac{\partial J}{\partial z^{[2]}}\right) \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial w^{[1]}}$$

cost function : $J(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}^{(i)}(\hat{y}, y)$

for ith data

with $\mathcal{L}^{(i)} = -\left[y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \right]$

assume our neural network



we need to optimise these ws and bs and update each as $w^{[l]}$ as $w^{[l]} := w^{[l]} - \alpha \frac{\partial J}{\partial w^{[l]}}$

we start by optimising outermost param first.

↳ as outermost have direct effect on cost.

we need $\frac{\partial J}{\partial w^{[3]}}$

we find $\frac{\partial \mathcal{L}}{\partial w^{[3]}}$ and we can add this for m examples

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = -\left[y^{(i)} \frac{\partial}{\partial w^{[3]}} (\log \sigma(w^{[3]} a^{[2]} + b^{[3]})) + (1-y^{(i)}) \frac{\partial}{\partial w^{[3]}} (\log(1 - \sigma(w^{[3]} a^{[2]} + b^{[3]}))) \right]$$

where $a^{[3]} = (w^{[3]} \cdot a^{[2]} + b^{[3]})$

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = -\left[y^{(i)} \frac{1}{a^{[3]}} \cdot a^{[3]} \cdot (1-a^{[3]}) \cdot (a^{[2]})^T + (1-y^{(i)}) \frac{1}{(1-a^{[3]})} \cdot a^{[3]} \cdot (1-a^{[3]}) \cdot a^{[2]T} \right]$$

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = -\left[y^{(i)} \cdot (1-a^{[3]})\, a^{[2]^T} + (1-y^{(i)})\, a^{[3]} \cdot a^{[2]^T} \right]$$

$$= -\left[y^{(i)} a^{[2]^T} - a^{[3]} a^{[2]^T} \right] = \boxed{-(y^{[i]} - a^{[3]})\, a^{[2]^T}}$$

$$\boxed{\frac{\partial J}{\partial w^{[3]}} = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - a^{[3]})\, a^{[2]^T}} \rightarrow \text{we update } w^{[3]} \text{ like this}$$

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}}$$

we did this as we want the error to backpropogate

$(a^{[3]} - y)$ $\quad w^{[3]^T}$ $\quad a^{[2]}(1-a^{[2]})$ $\quad a^{[1]^T}$

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \underbrace{(a^{[3]} - y)}_{(1,1)} \cdot \underbrace{w^{[3]^T}}_{(2,1)} * \underbrace{a^{[2]} \cdot (1-a^{[2]})}_{(2,1)} \cdot \underbrace{a^{[1]^T}}_{(1,3)}$$

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \underbrace{w^{[3]^T}}_{(2,1)} \circledast \underbrace{a^{[2]}(1-a^{[2]})}_{(2,1)} \cdot \underbrace{(a^{[3]} - y)}_{(1,1)} \underbrace{a^{[1]^T}}_{(1,3)}$$

$(2,1)$.

Element wise multiplication. $\quad (2,3)$

"REFER NOTES FOR RIGOROUS PROOF."

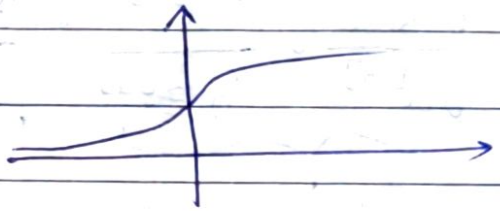
$$\frac{\partial J}{\partial w^{[2]}} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L}{\partial w^{[2]}}$$

Improving your NN (Neural network)

(1) Experiment with activation functions.

sigmoid ($\sigma(z)$)

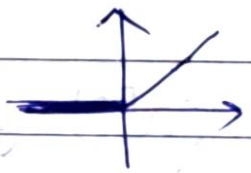
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$$\sigma'(z) = \sigma(z)(1-\sigma(z)).$$

ReLU (z)

$$\text{ReLU}(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases}$$

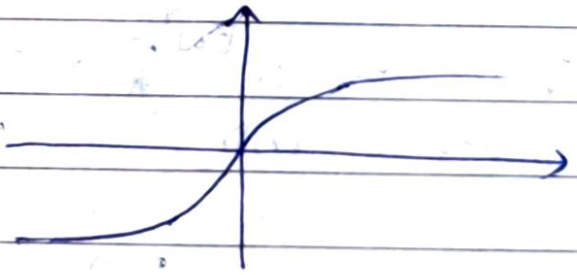


$$\text{ReLU}'(z) = \mathbb{1}\{z > 0\}$$

↳ Indicator function

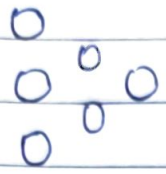
tanh (z)

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



$$\tanh'(z) = 1 - (\tanh(z))^2$$

Why do we need activation functions?



let activation = Id func. $z \mapsto z$.

$$\hat{y} = a^{[3]} = z^{[3]} = w^{[3]} a^{[2]} + b^{[3]}$$
$$\hat{y} = w^{[3]}\left(w^{[2]} a^{[1]} + b^{[2]}\right) + b^{[3]}$$
$$\hat{y} = w^{[3]} w^{[2]} \left(w^{[1]} x + b^{[1]}\right) + w^{[3]} b^{[2]} + b^{[3]}$$
$$\hat{y} = wx + B.$$

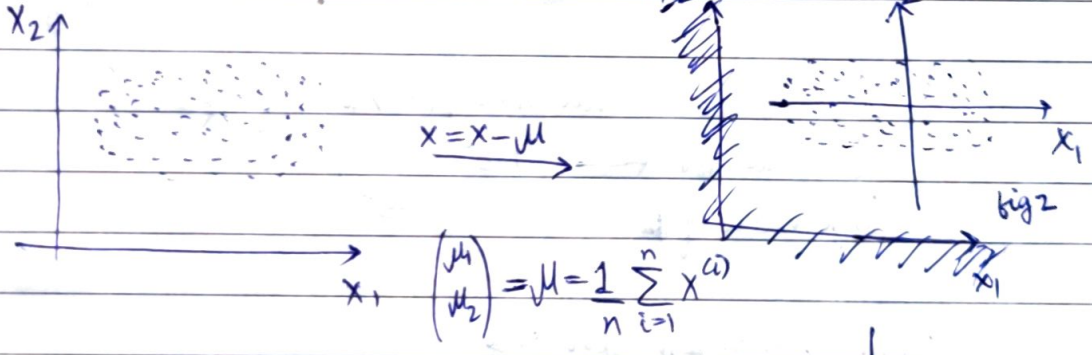
happens if we have no activation function

regardless of depth NN becomes linear.

(II) Use initialisation or normalisation

Normalizing your input

assume data $\rightarrow X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$



for fig 2 loss function

for grad. descent the slope bounces around

$$x = \frac{x}{\sigma}$$

$$\sigma = \frac{1}{n}\sum_{i=1}^{n} (x^{(i)})^2$$

for fig 3 loss function.

hence we converted the data

↳ points consistently inwards for grad. descent

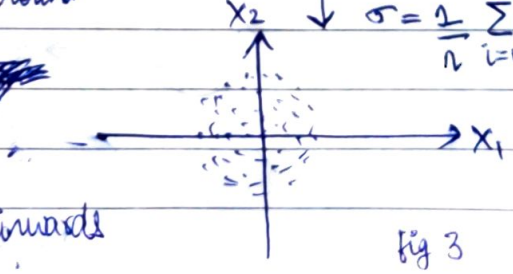
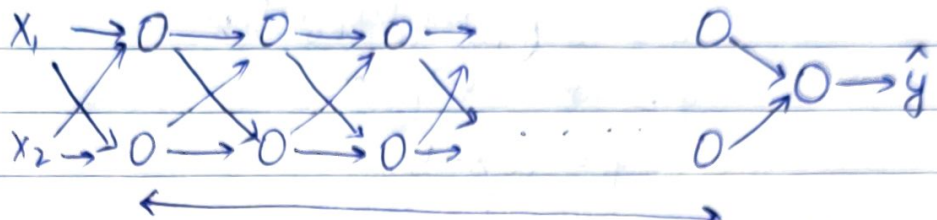


fig 3

Vanishing / Exploding gradients.

assume a NN



10 layers

assume $g: z \mapsto z$

$b = 0$

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

$$\hat{y} = w^{[L]} \cdot w^{[L-1]} \cdot \dots \cdot w^{[1]} \cdot X$$

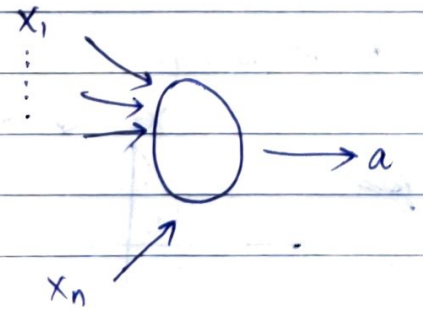
(2,2) matrices

assume that $w^{[L]}$ is slightly larger than Identity matrix $\rightarrow \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}$

all mult become $\begin{pmatrix} 1.5^L & 0 \\ 0 & 1.5^L \end{pmatrix}$

"explodes" for large values of L

Example with 1 neuron :→



$a = \sigma(z)$

$z = w_1 x_1 + \dots + w_n x_n$

large n → small w_i as we want around 1 because of exploding gradients

initialise $w_i \sim \frac{1}{n}$

some initialisation schemes are :→ (practical obs)

$w_i = np.random.randn(shape) * np.sqrt\left(\frac{1}{n^{[L-1]}}\right)$

for sigmoid ↑

L → layer we are at
n → no. of neurons in a layer.

for ReLU → same above eqn. with $\frac{2}{n^{[L-1]}}$ instead of $\frac{1}{n^{[L-1]}}$

Xavier initialisation

$$w^{[L]} \sim \sqrt{\frac{1}{n^{[L-1]}}} \quad \text{for tan h}$$

He Initialisation :

$$w^{[L]} \sim \sqrt{\frac{2}{n^{[L]} + n^{[L-1]}}}$$

less computationally expensive.

computing J for each mini batch only takes 1000 steps

$$J = \frac{1}{1000} \sum_{i=1}^{1000} L^{(i)}$$

(III) Optimisation (Refer notes for more clarity).

$X = (x^{(1)}, \ldots x^{(m)})$

$Y = (y^{(1)} \ldots y^{(m)})$

mini-batch

$X = (X^{\{1\}}, \ldots X^{\{T\}})$

each mini batch is of 1000 examples ← $(x^{(1)} \ldots x^{(1000)})$

$Y = (Y^{\{1\}} \ldots Y^{\{T\}})$

algo

for iteration t = 1 .....

select batch $(X^{\{t\}}, Y^{\{t\}})$

update $w^{[L]}, b^{[L]}$ by backprop and such

Gradient descent + Momentum Algorithm

→ spread out

assume we have an extended loss func.

"INTUITION"

— → normal G.D

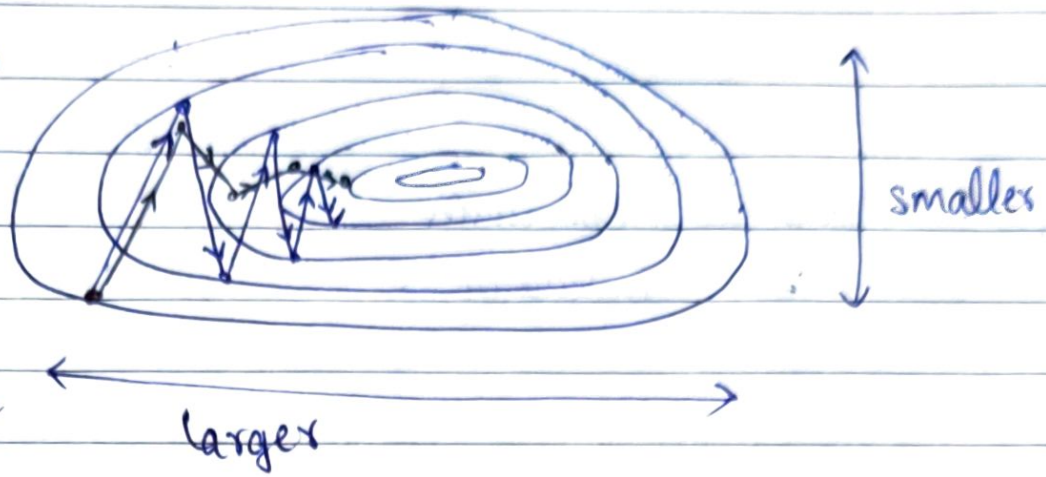
— → using momentum

↓

looks at how we moved in the past by taking average movement

↓

↑↓ average movement is little along y axis and more along x-axis.



this G.D bounces all over the place

↓

we want more movement in the horizontal dirn. than in vertical dirn. as that'd allow us to get optimum value earlier

$$w = w - \alpha \frac{\partial J}{\partial w} \rightarrow \text{normal gradient descent.}$$

velocity

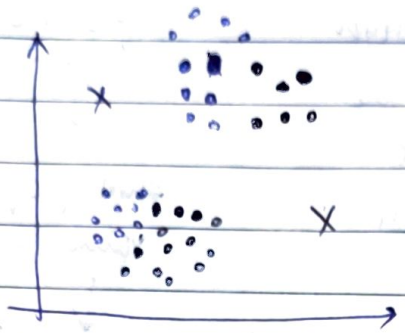
$$v = \beta v + (1-\beta)\frac{\partial J}{\partial w}$$

$$w = w - \alpha v$$

} gradient descent with momentum

Unsupervised Learning

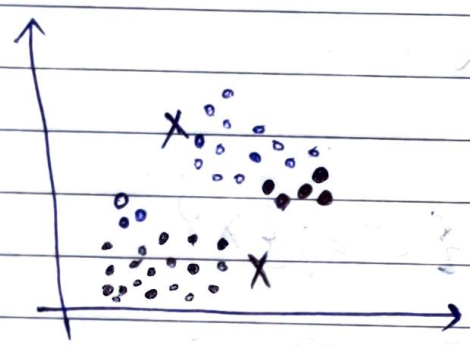
K-means-clustering



● → data point

X → centroid

We pick 2 centroids randomly here and we colour all data points depending on which centroid it is closer to.



centroid is updates as new centroid of coloured values.

new black centroid is centre of all black points.

and we keep on doing this process.

Mathematical notation for K-means clustering

Data : $\{x^{(1)}, \ldots x^{(m)}\}$.

1) Initialize cluster centroid,

$\mu_1, \mu_2, \ldots \mu_K \in \mathbb{R}^n$ (for low dimensions)

how to initialise → randomly ~~pick~~ assign them value.

OR

→ pick k examples from dataset and make them as centroids. (High dimensions ke liye)

2) Repeat until convergence

a) Set $c^{(i)} := \underset{j}{\operatorname{argmin}} \|x^{(i)} - \mu_j\|_2$

$\|\ \|_2$ subscript denotes L_2 norm

Sum of squares of values of the vector

$$\sum_{k=1}^{n} \left(x_k^{(i)} - \mu_k^{j}\right)^2$$

for each $x^{(i)}$ in our dataset we assign it a colour $c^{(i)}$ depending on the index ~~number~~ of centroid closest to it.
like μ_1 is closer or μ_2 is closer.

number here

b) For $j = 1, \ldots, k$

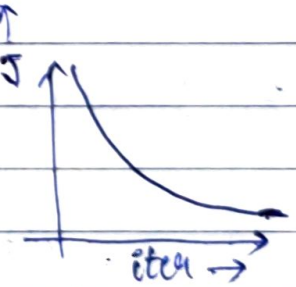
$$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\}\, x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}$$

"updates the centroid" by ~~fi~~ relocating it as the centre of all points coloured that colour.

✱ cost function :→

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

"assigned colour" "centroid"

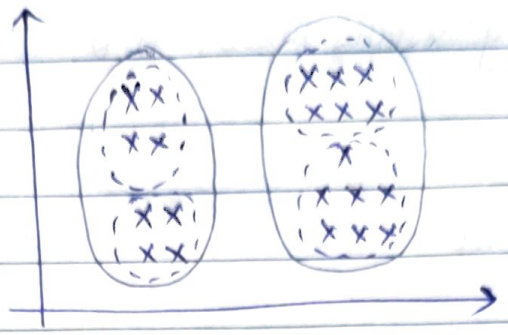


→ proof in lecture notes.

→ J converges as it goes down every iteration

how to decide how many clusters to pick?

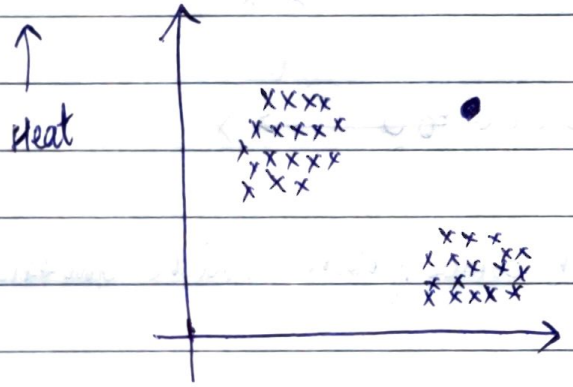
↳ you can pick it by hand.



someone might see 2 ~~segments~~ clusters while someone may see 4.

Density estimation

suppose we have an aircraft engine manufacturer.



x → good engines

• → we want to test this engine against the ones we know are good

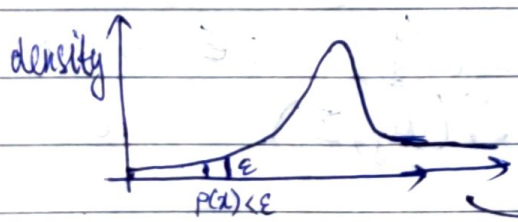
we make a ~~probability~~ density distribution curve

example :→

just consider vibration

we find $p(x)$ according to that model.

if $p(x) < \epsilon$ → anomaly



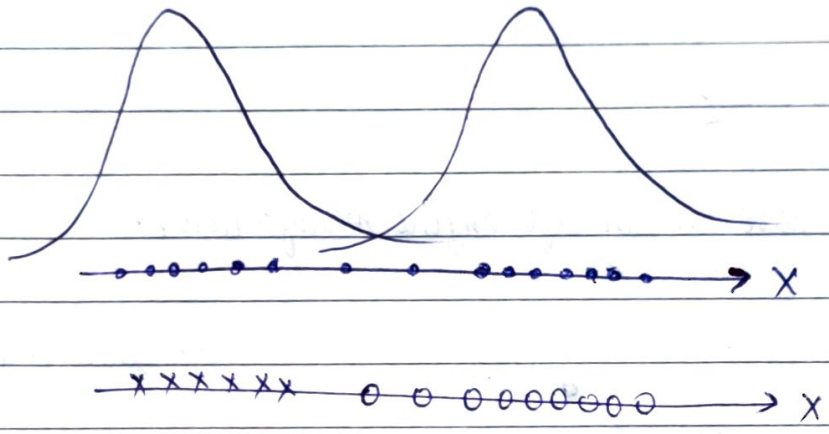
↳ now try to visualise this in 3D (we will get hills)

There is no standard ~~algorithm~~ model distribution to accomplish all these density distributions.

So, we use mixture of gaussians model to make these distributions.

Mixture of Gaussians model

1-D example



How to know which dataset comes from which Gaussian??

Mixture of Gaussians model :→

suppose there is a latent (hidden/unobserved) random variable z, and $x^{(i)}$, $z^{(i)}$ are distributed

- $p(x^{(i)}, z^{(i)}) = P(x^{(i)} | z^{(i)}) P(z^{(i)})$

- where $z^{(i)} \sim \text{Multinomial}(\phi)$ $\quad z \in \{1, 2, \cdots k\}$

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

↓ if we are representing as a mixture of k Gaussians.

$z^{(i)}$ → denotes which gaussian does a point belong to.

how is mixture of gaussian model different from Gaussian discriminant Analysis ?

→ In GDA $(x^{(i)}, z^{(i)})$ we knew $z^{(i)}$; i.e. to which Gaussian/ class a data item belongs. On the other hand $z^{(i)}$ is unknown is mixture of Gaussian model.

If we knw values of $z^{(i)}$ we could use Maximum Likelihood Estimate (MLE) as below.

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

upon differentiating to ~~minimi~~ maximise we get $\left(\frac{dl}{d\phi} = 0 \text{ and so on}\right)$

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

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

$$\Sigma_j = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{z^{(i)}})(x^{(i)} - \mu_{z^{(i)}})^T$$

↳ verify from lecture notes

EM (expectation maximisation)

has 2 steps :→

1) E-step (Guess value of $z^{(i)}$'s)

set $w_j^{(i)} = p(z^{(i)} = j \mid x^{(i)}; \phi, \mu, \Sigma)$ → by bayes rule

$$w_j^{(i)} = \frac{p(x^{(i)} \mid z^{(i)} = j) \cdot p(z^{(i)} = j)}{\sum_{l=1}^{k} p(x^{(i)} \mid z^{(i)} = l) \cdot p(z^{(i)} = l)}$$

prob. that $z^{(i)} = j$ given $x^{(i)}$

$$\mathcal{N}(\mu_j, \Sigma_j) = \frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)\right)$$

$z \sim$ multinomial so $p(z^{(i)} = j) = \phi_j$

2) M-step

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

(when we knew $z^{(i)}$) earlier formula had $1\{z^{(i)} = j\}$.

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

now we have $E[1\{z^{(i)} = j\}]$

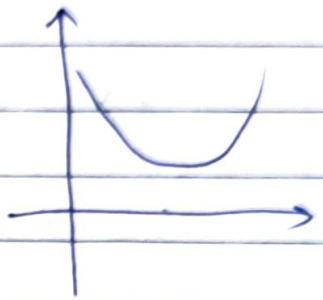
$\Sigma_j :=$ .... (refer notes)

expected value of $1\{z^{(i)} = j\}$ which is the prob. of $z^{(i)} = j$ which is $w_j^{(i)}$ in place of $1\{z^{(i)} = j\}$

~~E-M step is another~~ EM is a pretty decent estimate of deciding which cluster a point belongs to. Is an alternative for K-means clustering.

Jensen's inequality

Let f be a convex func. $\Rightarrow (f''(x) > 0)$



Let X be a random variable

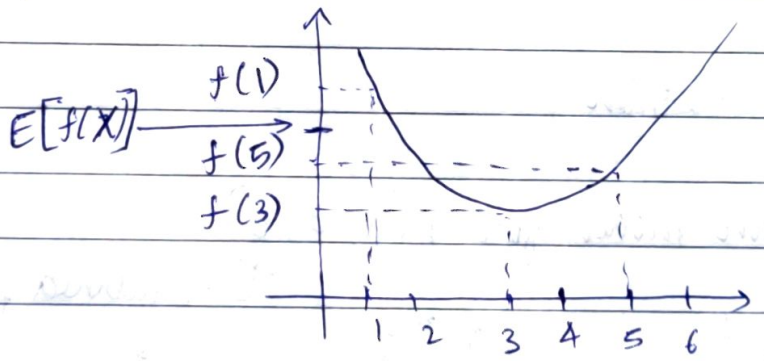
then

$$f(E[X]) \leq E[f(X)]$$

$E[X]$ → expected value of X

$E[f(X)]$ → expected value of $f(X)$

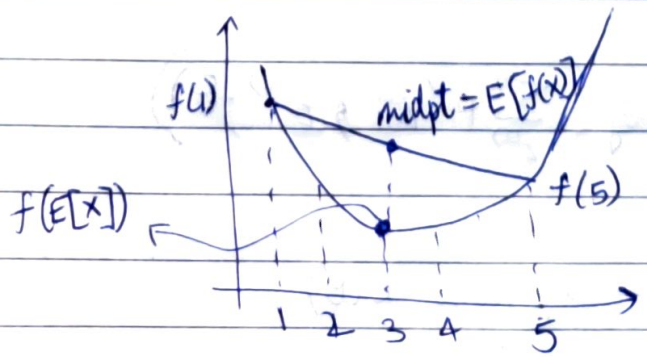
Example :→



$X = \begin{cases} 1 & \text{with prob} = \frac{1}{2} \\ 5 & \text{with prob} = \frac{1}{2} \end{cases}$

$f(3) = f(E[X])$

$E[f(X)] = \frac{1}{2} f(1) + \frac{1}{2} f(5)$



geographical non-rigourous proof

Further if $f''(x) > 0$ (f is strictly convex)

then if

$$E[f(x)] = f(E[x]) \iff x \text{ is a constant}$$

(i.e. $x = E[x]$ with prob. 1)

for concave function

Let f be a concave funct. $\Rightarrow f'' < 0$

Let X be a random variable

then $f(E[X]) \geq E[f(X)]$

Further if $f'' < 0$ then if $f(E[X]) = E[f(X)] \iff X$ is constant

EM model visualisation

suppose we have model for $p(x, z; \theta)$

↳ combined parameter

↓

has all values

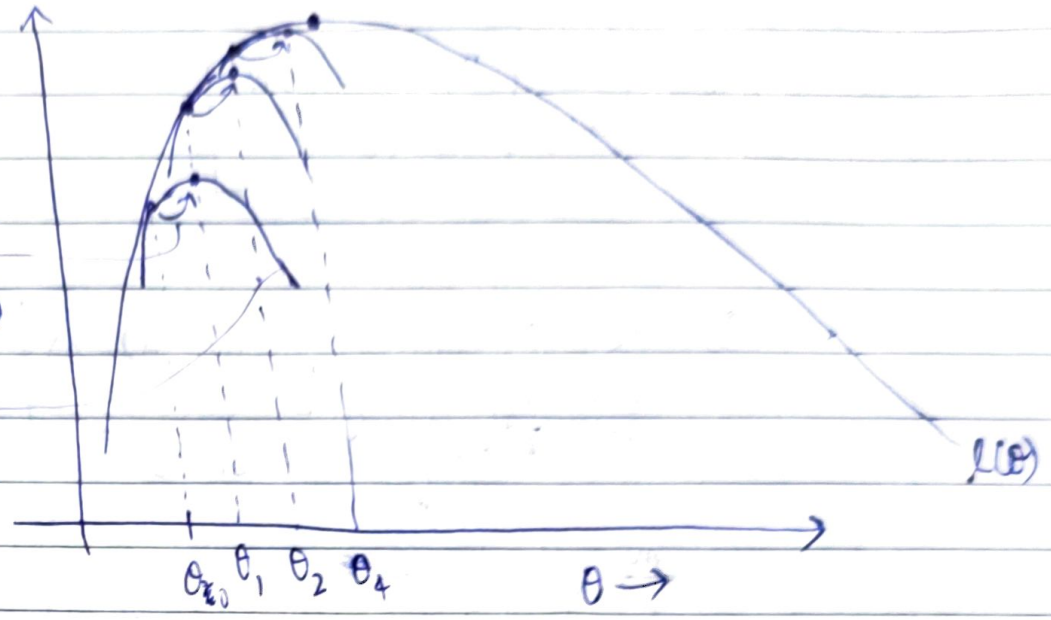
μ, Σ, ϕ contained

on $X \in \{x^{(1)}, x^{(2)} \ldots x^{(m)}\}$

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

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

what we want : $\arg\max_{\theta} \ell(\theta)$



M-step maxm. the lower curve to give a new θ

this is a lower curve; only having the same value at θ_0.

doing this construction and maximisation we slowly converge to the local maxima.

Our goal is to $\max_\theta l(\theta)$

$$= \max_\theta \sum_{i=1}^{m} \log\left(P(x^{(i)};\theta)\right)$$

$$= \sum_{i=1}^{m} \log\left(\sum_{z^{(i)}} P(x^{(i)}, z^{(i)};\theta)\right)$$

$$= \sum_{i=1}^{m} \log\left(\sum_{z^{(i)}} Q_i(z^{(i)}) \left[\frac{P(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}\right]\right)$$

where $Q_i(z^{(i)})$ is a probability distribution.

$$\left(\text{i.e. } \sum_{z^{(i)}} Q_i(z^{(i)}) = 1\right)$$

(expected value)

$$= \sum_{i=1}^{m} \log \underset{z^{(i)} \sim Q_i}{\mathbb{E}} \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

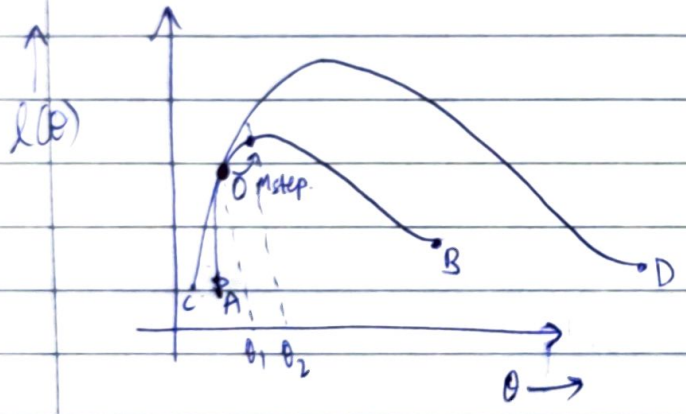
$$\geq \sum_{i=1}^{m} \underset{z^{(i)} \sim Q}{\mathbb{E}} \left[\log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

using Jensen's inequality

$$= \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \left(\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right)$$

$f(Ex) \geq E[f(x)]$

$f(x) = \log x$



On a given iteration of EM with current param. θ we want :→

(i) → $\log \underset{z^{(i)} \sim Q_i}{\mathbb{E}} \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] = \underset{z^{(i)} \sim Q_i}{\mathbb{E}} \left[\log \left[\frac{P(x^{(i)}, z^{(i)}, \theta)}{Q_i(z^{(i)})} \right] \right]$

for lower curve to touch upper curve at θ.

(i) is true when random variable is constant. (By Jensen's inequality)

i.e. $$\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = \text{constant}$$

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

but since $$\sum_{z^{(i)}} Q_i(z^{(i)}) = 1$$

hence;

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

$$Q_i(z^{(i)}) = p(z^{(i)} | x^{(i)}; \theta)$$ [Refer "Lecture notes"]

E-step therefore involve equality at θ (point in Graph)

Set, (these were taken as $w_j^{(i)}$ earlier on)

$$Q_i(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta)$$

M-step involves finding new θ where $f(\theta)$ is maxm. ($f(\theta)$ is the lower curve here)

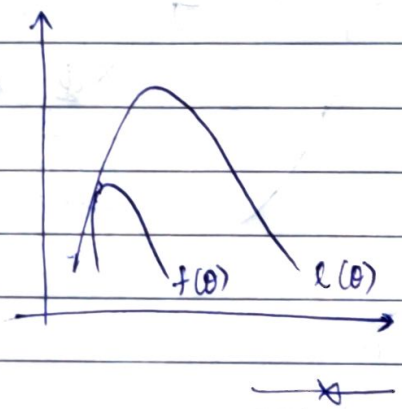
$$\theta := \arg\max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}\right)$$

E-M convergence

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

M-step: $\theta := \underset{\theta}{\text{argmax}} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \left(\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right)$

$f(\theta)$ → lower bound of log likelihood



For mixture of Gaussians model we had :→

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

$z^{(i)} \sim \text{Multimial}(\phi)$

i.e. $\left[p(z^{(i)} = j) = \phi_j \right]$

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

E-step for mixture of gaussian

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

"$p(z^{(i)} = j)$"

$w_j^{(i)}$ determines what fraction of each gaussian combine to give $x^{(i)}$

M-step

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

collectively were taken as θ

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

($p(x^{(i)} | z^{(i)}; \dots)$ ← $p(z^{(i)})$)

we want to maxm. above expn.

so, we take derivative wrt. params.

$$\nabla_{\mu_j} (\dots) \overset{\text{set}}{=} 0 \Rightarrow \mu_j = \frac{\sum_i w_j^{(i)} x^{(i)}}{\sum_i w_j^{(i)}}$$

$w_j^{(i)}$: strength with which $x^{(i)}$ is assigned jth gaussian; $w_j^{(i)} = p(z^{(i)} = j | x^{(i)}; \dots)$

$$\nabla_{\phi} (\dots) \overset{\text{set}}{=} 0 \Rightarrow \phi_j = \frac{\sum_i w_j^{(i)}}{\sum_i \sum_j w_j^{(i)}}$$

$$\nabla_{\Sigma_j} (\dots) \overset{\text{set}}{=} 0 \Rightarrow$$

$\Sigma_j = \dots$ (refer notes)

Factor analysis model

We will be using mixture of Gaussian model for this but, instead → instead of $z^{(i)}$ being discrete it is continuous

$$z^{(i)} \sim \mathcal{N}$$

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

we proved via Jensen's,

$$\ell(\theta) \geq J(\theta, Q)$$

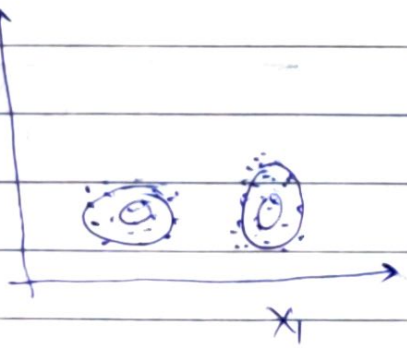
for any θ, Q

E-step: maximise J wrt. Q → set it such that equality occurs at a initial point

M-step: maxm. J wrt. θ

coordinate ascent → we have multiple features we want to maxm. func. on and we go back an forth b/w them to maxm. our func.

Mixture of gaussians example → say $n = 2$ and $m = 100$



$m \gg n$ here

we will not use mixture of gaussians when $m \not\gg n$ or $m << n$

say $m = 30$, $n = 100$

Suppose we (try to) model above case as a single gaussian

$$x \sim \mathcal{N}(\mu, \Sigma)$$

MLE: $\mu = \frac{1}{m} \sum_{i} x^{(i)}$

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

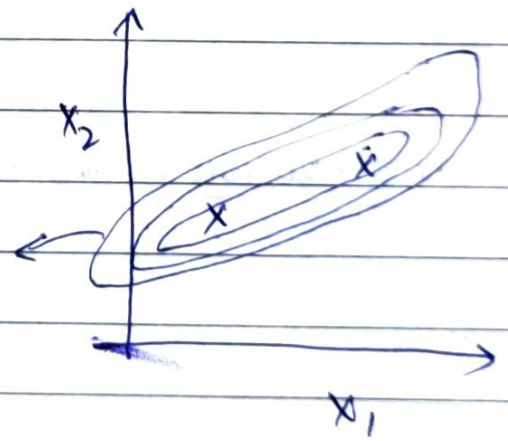
If $m \leq n$, then Σ is "singular / non-invertible"

$$\frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-(\dots)\Sigma^{-1}(\dots)\right)$$

problem as Σ is singular. → hence we can't model as a single Gaussian

illustration of above example.

$n = 2$ $m = 2$



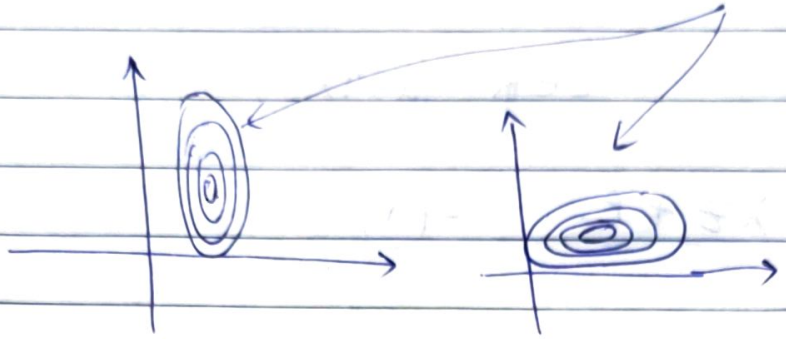
contours as essentially straight lines with zero width and infinite length

Hence we need a new model

Option 1: constraint Σ to be diagonal

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 & 0 \\ & \sigma_2^2 & & 0 \\ 0 & & \ddots & \\ 0 & 0 & & \sigma_n^2 \end{bmatrix} \rightarrow \sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

→ this constrains our gaussian to be along axis



MLE: is same ; Above model is limiting as it assumes any two features are completely "uncorrelated".

↓

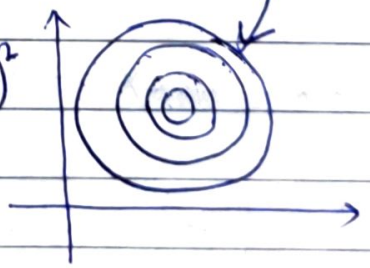
we won't use this model

Option 2:

Constrain Σ to be $\Sigma = \sigma^2 I$

i.e. $\Sigma = \begin{bmatrix} \sigma^2 & & 0 & 0 \\ & \sigma^2 & & 0 \\ 0 & & \ddots & \\ 0 & 0 & & \sigma^2 \end{bmatrix}$ → gives circular gaussians.

$$\sigma^2 = \frac{1}{m} \frac{1}{N} \sum_i \sum_j (x_j^{(i)} - \mu_j)^2$$



This is even more limiting than option 1 and so isn't used

Consider a model where;

$p(x, z) = p(x|z) \cdot p(z)$

z is hidden

(example
$d = 3$
$n = 100$
$m = 30$)

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

$x = \mu + \lambda z + \varepsilon$. — (i)

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

parameters :→ $\mu \in \mathbb{R}^n$; $\lambda \in \mathbb{R}^{n \times d}$; $\Psi \in \mathbb{R}^{n \times n}$ and Ψ_{ii} is diagonal

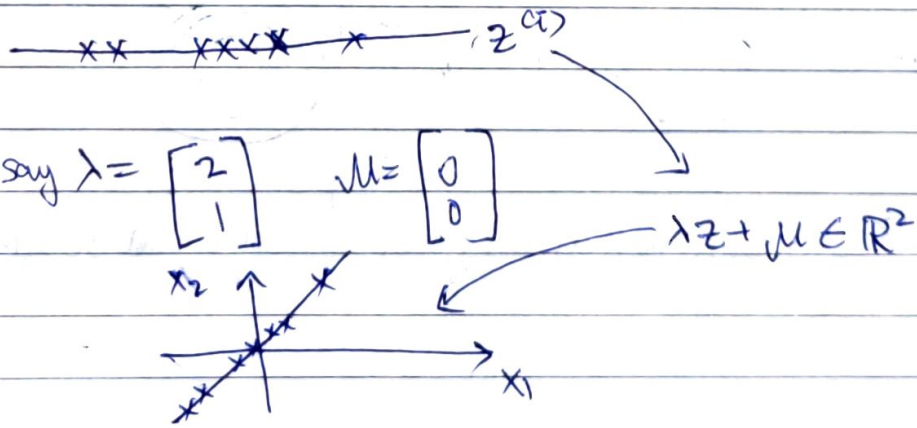
→ $x|z \sim \mathcal{N}(\mu + \lambda z, \Psi)$ is equivalent to (i)

Examples :→

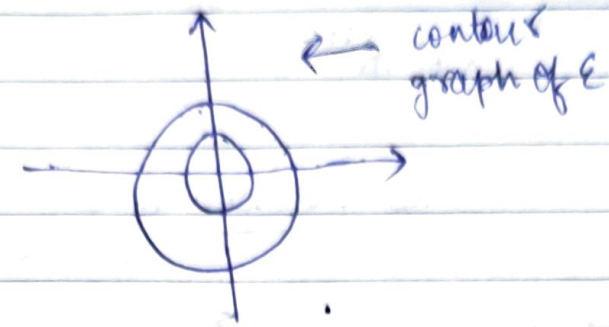
$z \in \mathbb{R}^1$ $\quad x \in \mathbb{R}^2$ $\quad d = 1$

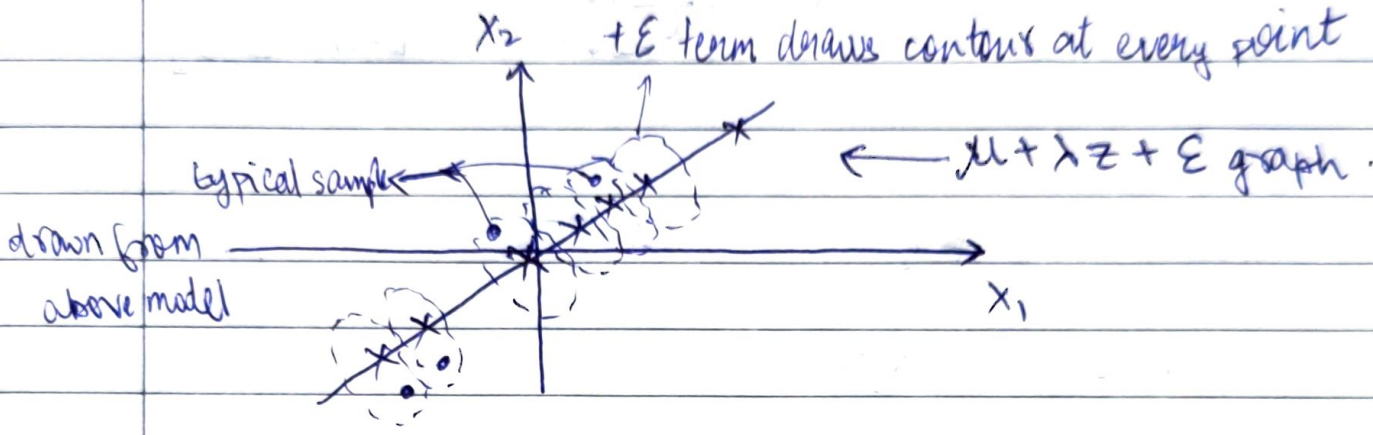
$n = 2$

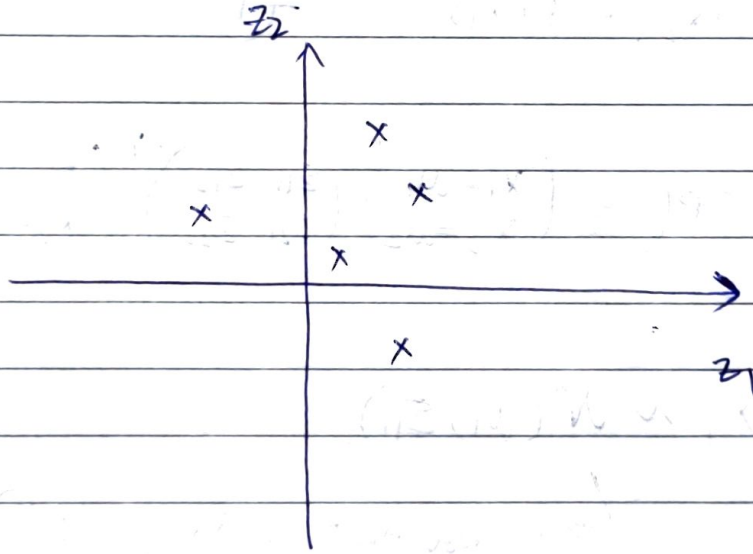
$m = 7$



say Ψ = $\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$







$z \in \mathbb{R}^2$ $d=2$

$x \in \mathbb{R}^3$ $n=3$

$m = 5$.

$\lambda z + \mu$

3×2. 3×1

this maps our points onto a 3-D plane (still all planar points).

$\Psi \in \mathbb{R}^{3\times3}$ ←

so we can imagine like spherical contours.

Multivariate Gaussian (Refer notes)

$$x = \begin{bmatrix} x_1 \\ \hline x_2 \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow s \end{matrix}$$ r components $x_1 \in \mathbb{R}^r$, $x_2 \in \mathbb{R}^s$ ... $x \in \mathbb{R}^{r+s}$

$x \rightarrow$ is "partition" vector.

$$x \sim \mathcal{N}(\mu, \Sigma) \Rightarrow \mu = \begin{bmatrix} \mu_1 \\ \hline \mu_2 \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow s \end{matrix} \quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$ (r | s)

Marginal : $P(x_1) = ?$

$$P(x) = P(x_1, x_2)$$

$$\int_{x_2} P(x_1, x_2)\, dx_2 = P(x_1) \quad \text{--- (1)}$$

$$P(x_1, x_2) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}^T \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}^{-1} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \right)$$

$$P(x_1) \Rightarrow x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$$

↳ to show this we integrate (1)

Conditional : $P(x_1 | x_2) = ?$

$$\frac{P(x_1, x_2)}{P(x_2)}$$

$$x_1 | x_2 \sim \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$$

$$\mu_{1|2} = \mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (x_2 - \mu_2)$$

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

deriving E-M algo. using multivariate gaussians

1. Derive $p(x,z)$. [Refer notes]

$$\begin{pmatrix} z \\ x \end{pmatrix} \sim \mathcal{N}\left(\mu_{x,z}, \Sigma\right)$$

$z \sim \mathcal{N}(0, I)$ $\quad E_z = 0 \longrightarrow$ as mean is 0 for z.

$x = \mu + \lambda z + \varepsilon$ $\quad E_x = E\left[\mu + \lambda z + \varepsilon\right] = E[\mu] = \mu$.

($\lambda z \to 0$, $\varepsilon \to 0$)

$$\mu_{x,z} = \begin{bmatrix} 0 \\ \hline \mu \end{bmatrix} \begin{matrix} \updownarrow d \\ \updownarrow n \end{matrix}$$

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} = \begin{bmatrix} E\left[(z - E_z)(z - E_z)^T\right] & E\left[(z - E_z)(x - E_x)^T\right] \\ E\left[(x - E_x)(z - E_z)^T\right] & E\left[(x - E_x)(x - E_x)^T\right] \end{bmatrix}$$

(Σ_{11}: $d \times d$, Σ_{12}: $d \times n$, Σ_{21}: $n \times d$, Σ_{22}: $n \times n$)

$$\Sigma_{22} = E\left[(x - E_x)(x - E_x)^T\right] = E\left[(\lambda z + \mu + \varepsilon - \mu)(\lambda z + \mu + \varepsilon - \mu)^T\right]$$

$$= E\left[(\lambda z + \varepsilon)(\lambda z + \varepsilon)^T\right]$$

$E[\varepsilon\varepsilon^T] = \Psi \to$ defn.

$$\Sigma_{22} = E\left[\lambda z z^T \lambda^T + \lambda z \varepsilon^T + \varepsilon z^T \lambda^T + \varepsilon\varepsilon^T\right]$$

$E[zz^T] = I$ as variance is I.

$\lambda z \varepsilon^T$: zero expected value as z and ε^T have 0 expected value and are not correlated. $\varepsilon z^T \lambda^T$: likewise.

$$\Sigma_{22} = E\left[\lambda z z^T \lambda^T\right] + E\left[\varepsilon\varepsilon^T\right]$$

$$\Sigma_{22} = \lambda E\left[z z^T\right] \lambda^T + \Psi$$

$$\Sigma_{22} = \lambda I \lambda^T + \Psi = \lambda\lambda^T + \Psi.$$

deriving similarly for all we get

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} = \begin{bmatrix} I & \lambda^T \\ \lambda & \lambda\lambda^T + \Psi \end{bmatrix}$$

so we derived,

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ \mu \end{bmatrix}, \begin{bmatrix} I & \lambda^T \\ \lambda & \lambda\lambda^T + \Psi \end{bmatrix} \right)$$

~~we can~~ $p(x^{(i)})$ will be drawn from above Gaussian density

we can find the log likelihood for $p(x^{(i)})$ and differentiate it wrt parameters but we find out there is no closed form solution.

So, we use ~~this~~ E-M algorithm.

E-step:

$Q_i(z^{(i)}) = p(z^{(i)} | x^{(i)}; \theta)$. $z^{(i)}$ → continuous here for factor analysis

we can't use $w_j^{(i)}$ here as $z^{(i)}$ is continuous.
↳ we will need infinitely many $w_j^{(i)}$'s to store all

tracks value for each state of $z^{(i)}$

$$z^{(i)} | x^{(i)} \sim \mathcal{N}\left(\mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}} \right)$$

where, $\mu_{z^{(i)}|x^{(i)}} = \vec{0} + \lambda^T (\lambda\lambda^T + \Psi)^{-1} (x^{(i)} - \mu)$

$$\Sigma_{z^{(i)}|x^{(i)}} = I - \lambda^T (\lambda\lambda^T + \Psi)^{-1} \lambda .$$

we got above equations using our formula for multivariate gaussians.

M-step : (refer lecture notes)

mathematical trick for derivation :→

$$Q_i(z^{(i)}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\ldots|}\left(\exp\left(-\frac{1}{2}(\ldots)\right)\right)$$

if we get something expr. as below then,

$$\int_{z^{(i)}} Q_i(z^{(i)})\, z^{(i)}\, dz^{(i)} \longrightarrow \text{don't subm. } Q_i(z^{(i)})$$

same as →

$$E_{z^{(i)}\sim Q_i}[z^{(i)}] = \mu_{z^{(i)}|x^{(i)}}$$

as Q_i is gaussian so mean is $\mu_{z^{(i)}|x^{(i)}}$

$$\theta := \arg\max_{\theta} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)})}{Q_i(z^{(i)})}\, dz^{(i)}$$

$$= \sum_i E_{z^{(i)}\sim Q_i}\left[\log \frac{P(x^{(i)}, z^{(i)})}{Q_i(z^{(i)})}\right]$$

plug in gaussian density.

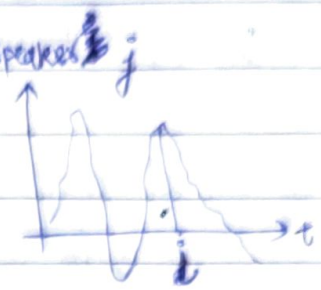
$$= \sum_i E_{z^{(i)}\sim Q_i}\left[\text{quad. func.}\right]$$

$\nabla_\mu(\ldots) = 0$. solve for μ.

please refer lecture notes.

Independent Componet analysis (ICA) part-2

suppose we have n speakers, $s \in \mathbb{R}^n$.



$s_j^{(i)}$ → speaker j at time i

$x^{(i)} = A s^{(i)}$ → $(n,1)$, $x \in \mathbb{R}^n$

$(n,1)$ ← here no. of microphones = no. of speakers

(n,n)

what microphones record.

Goal: Find $w = A^{-1}$ so,

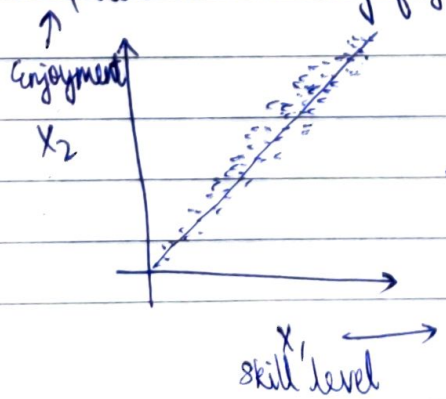
$s^{(i)} = w x^{(i)}$ → so we can tell about each speaker individually

ICA continued later

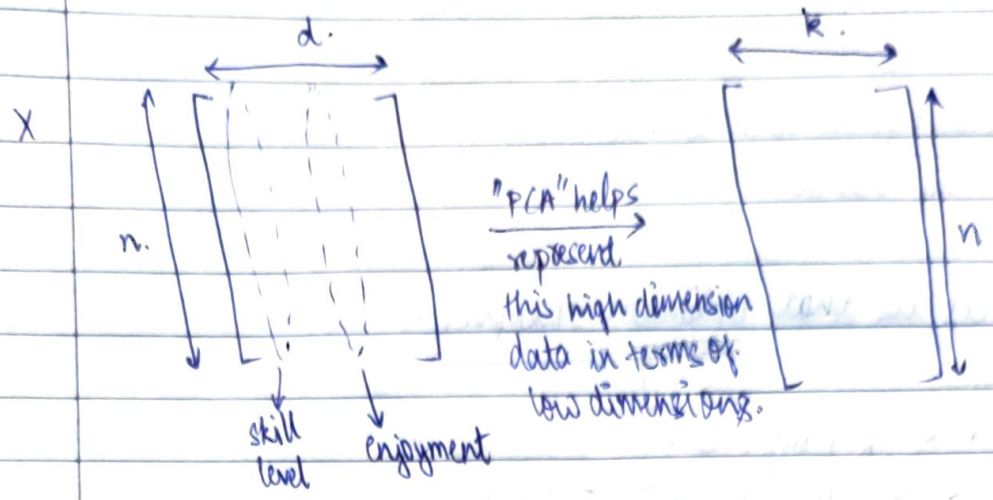
Principal Component Analysis

$\{x^{(i)}\}_{i=1}^{n}$ $x^{(i)} \in \mathbb{R}^d$

consider two features x_1 and x_2 which indicate skill level and enjoyment a pilot has during flight. There are other features too



∴ x_1 and x_2 are correlated.



Steps in PCA.

(i) Standardize your dataset.

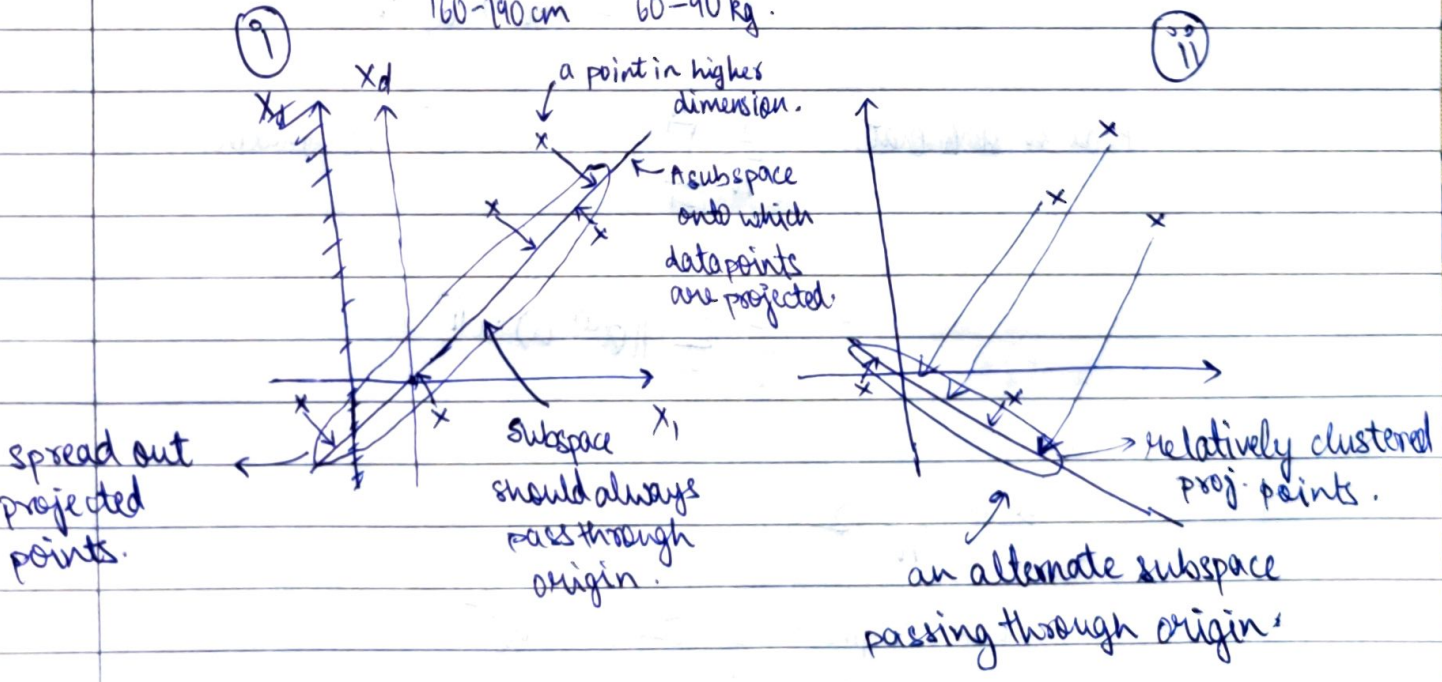
$$x_j^{(i)} := \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

basically each column is updated in such a manner that it is centered around its mean $\Rightarrow$ new mean = 0
and is scaled such that $\sigma_{new} = 1$

we do so because units may be in different units.

↳ example → height and weight

↓ 160–190 cm ↓ 60–90 kg.



in (i) projected points are sparse
in (ii) they are dense.

PCA tells us that optimum subspace.

variance of projected points is maximised → "sparsest"

idea is that we want to lose only the dimensions while retaining the variance.

$u \in R^d$ unit length of subspace (along the subspace) which we want to project the data.

So, projection of X on u will be $= \text{proj}(u) \cdot \vec{x}$

$$= \frac{uu^T}{u^Tu} \vec{x} \qquad u^Tu = 1$$

$$= (x^{(i)T} u)\, \vec{u}.$$

Find u such that $\frac{1}{n} \sum_{i=1}^{n} \| \text{proj}(u) \cdot x \|^2$ is maxm.

$$= \frac{1}{n} \sum_{i=1}^{n} \| (x^{(i)T} u)\, u \|_2^2$$

$$u = \underset{u}{\text{argmax}} \; \frac{1}{n} \sum_{i=1}^{n} (x^{(i)T} u)^2$$

$$u = \underset{u}{\text{argmax}} \; u^T \left(\frac{1}{n} \sum_{i=1}^{n} x^{(i)} (x^{(i)})^T \right) u$$

sample covariance matrix as mean is zero

solution to an eqn. of form, $u = \underset{u}{\text{argmax}} \; u^T A u$

is u = eigenvector of largest eigenvalue of A

proof :→ $u^T A u$ has to be maxm. ~~cond~~ s.t. $u^T u = 1$

let $g(u, \lambda) = u^T A u - \lambda(u^T u - 1)$ → using Lagrange's multiplier technique

critical points :→ $\frac{\partial g}{\partial u} = 2Au - 2\lambda u = 0$

$$Au = \lambda u$$

we see u is an eigenvector corresponding to an eigen value λ.

$$f(x) = u^T A u = u^T \lambda u = \lambda u^T u = \lambda$$

so λ will be the largest eigen ~~vecto~~ value of A and u will be λ's corresponding eigen vector.

so, in PCA if we first have data as X so first we calculate

$X^T X$ → (λ_1, u_1), (λ_2, u_2), (λ_3, u_3), ⋮, (λ_d, u_d)

find k s.t $\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} = 95\%$ or some percentage.

↑ k largest eigen values.

λ → eigen value of $X^T X$

u → corresponding eigenvector

we retained 95% of ~~data~~ variance in our data even though we went from d-dimensions to k dimensions.

Suppose we have a matrix M

if M is square & symmetric ⇒ M has orthogonal eigen vectors & real eigen values.

if M is PSD ⇒ M is square and symmetric ⇒ " "

in our case, we know X^TX is PSD

Eigen decomposition on X^TX $\cong$ singular value decomposition on X

↓ is equivalent to

~~X~~

ICA part-1

Suppose we have a cocktail party with
- d speakers
- d microphones

what speakers are saying at an instance ← $s \in \mathbb{R}^d$

$x \in \mathbb{R}^d$ ← what mics are picking up at an instance

$s_j^{(i)}$ → jth speaker say at ith time

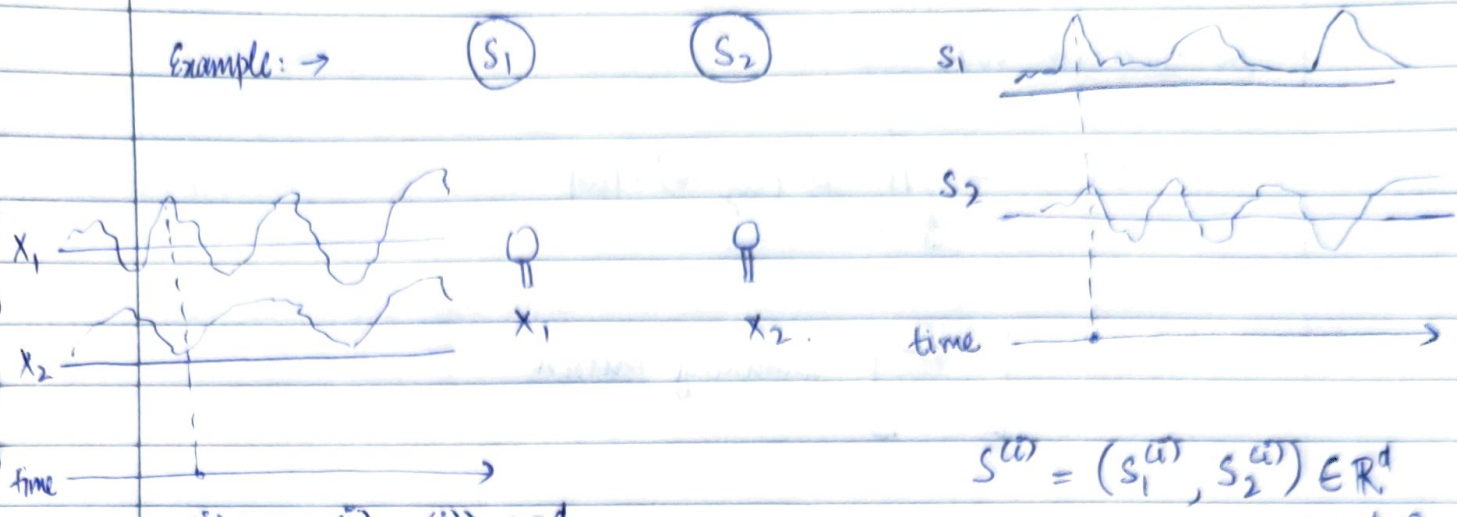
$x_j^{(i)}$ → recording in jth mic at ith time.

$X = A \cdot S$

↓

Mixing matrix

Example: → S_1 S_2



$x^{(i)} = (x_1^{(i)}, x_2^{(i)}) \in \mathbb{R}^d ; d=2$

$s^{(i)} = (s_1^{(i)}, s_2^{(i)}) \in \mathbb{R}^d$ where $d=2$

$x^{(i)} = A\, s^{(i)}$

(d,d)

$W = A^{-1}$ (unmixing matrix) $[s^{(i)} = W x^{(i)}]$

Assumptions : → (i) $\#s = \#x$

(ii) $s = Wx$ → (linearly associated)

(iii) $s_j \perp s_k \quad j \neq k$

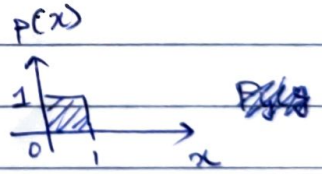
↳ R.V are independent

↓

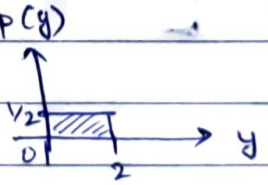
not necessarily true in a convo

(iv) s_j is not Gaussian

suppose we have $x \sim \text{Unif}[0,1]$



$y = 2x$.



$P_y(y) = P_x(y/2) \cdot \frac{1}{2}$

$y = Wx$. Hey $p_y(y) = p_x(W^{-1}y) \cdot \frac{1}{|W|}$

$y \in \mathbb{R}^d$, $W \in \mathbb{R}^{d \times d}$, $x \in \mathbb{R}^d$

↳ known as Jacobian

in ICA we have

$$p(x) = \prod_{j=1}^{d} p_s(w_j^T x) \cdot |W|$$

$W = \begin{bmatrix} -w_j- \end{bmatrix}$ unmixing matrix.

$$\begin{bmatrix} -w_j- \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = s_j$$

$p_s \sim$ logistic distribution $\longrightarrow$ pdf of logistic func $\rightarrow \frac{1}{1+e^{-x}}$

$$\ell(W) = \sum_{i=1}^{n} \left[\left(\sum_{j=1}^{d} \log \left[\sigma(x^{(i)}) \cdot (1 - \sigma(x^{(i)})) \right] \right) + \log |W| \right]$$

do gradient ascent till convergence

$$W := W + \alpha \left[\begin{bmatrix} 1 - 2\sigma(w_1^T x^{(i)}) \\ 1 - 2\sigma(w_2^T x^{(i)}) \\ 1 - 2\sigma(w_3^T x^{(i)}) \\ \vdots \end{bmatrix} x^{(i)T} + (W^T)^{-1} \right]$$

Reinforcement learning

In games like chess or go, common reward function R is

$R(s)$ = +1 for win
reward func. ↙ ↓ state
−1 for lose
0 for draw

Credit assignment problem :→

(i) Consider a chess game which was lost at move 50. Suppose, the blunder step was at move 20. How does the computer know this?

(ii) Consider a car crash. Before crashing (−ve outcome) most often brakes are applied. But blunder/fault occurs earlier on.

helps figure out what to do more of in +ve example and what exactly to do less of in −ve example.

Markov Decision Process (MDP)

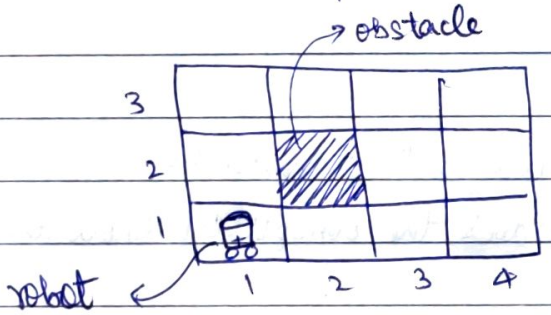
- helps model RL problems
- notation for modelling how world works

MDP involves 5-tuples →

$$(S, A, \{P_{sa}\}, \gamma, R)$$

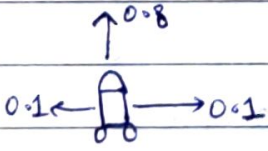
$S \rightarrow$ set of states
$A \rightarrow$ set of actions
$P_{sa} \rightarrow$ state transition probabilities $\left(\sum_{s'} P_{sa}(s') = 1\right)$
$\gamma \rightarrow$ discount factor, $\gamma \in [0, 1)$
$R \rightarrow$ reward function

Consider an example to show MDP



① Here the robot can take 11 states

② Actions robot takes are $\{N, S, E, W\}$

③ 

suppose we press N command for our robot
suppose it going N has 0.8 prob., 0.1 for east and west

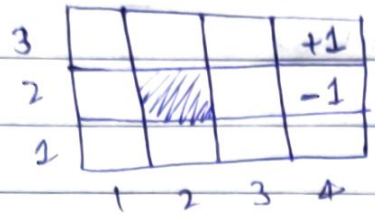
So,

$P_{(3,1)N}((3,2)) = 0.8$; $P_{(3,1)N}((4,1)) = 0.1$, $P_{(3,1)N}((2,1)) = 0.1$

state s ↓ action a ↓ state s'

$P_{(3,1)N}(\text{anything else}) = 0$ → so we have P_{sa} for all states.

(5) Reward for this example :→



$R((4,3)) = +1$ → incentivise going there

$R((4,2)) = -1$ → punish for going there.

if we want robot to reach $R((4,3))$ ASAP → we put a small negative reward for all other states (ex → −0.01) $R(s) = -0.01$

punishes robot for taking long.

our robot start at s_0 and picks an action a_0.

s_0
choose action a_0
Get to $s_1 \sim P_{s_0 a_0}$
choose action a_1
Get to $s_2 \sim P_{s_2 a_2}$
choose action a_2
⋮
and so on

Total payoff $= R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 (R(s_3)) \ldots\ldots$

γ here is <u>discount factor</u>

- helps ~~get~~ take positive actions sooner
- and delays negative rewards to later
- generally chosen to be like 0.99 and such.

in finance terms
helps relate to time value of money.
- get profit/loan sooner (inflation/interest)
- ~~get~~ get debt later (same amount.)

γ is useful as it helps total payoff converge.

Goal: choose actions over time to maxm. EXPECTED total payoff

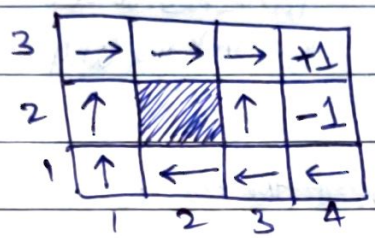
$$E\left[R(S_0) + \gamma R(S_1) + \gamma^2 R(S_2) + \gamma^3 R(S_3) \ldots\right]$$

policy (controller) $\pi : S \mapsto A$ (π maps states to actions)

↓

when in state s take action $\pi(s)$

Optimal policy for MDP in example (solved on PC by prof not by hand)



$\pi((3,1)) = \cup$

How to compute optimal policy?

Define: V^π, V^*, & π^*

for a policy π, $V^\pi : S \mapsto \mathbb{R}$ is s.t. $V^\pi(s)$ is the expected total payoff for starting in state s and executing π.

$$V^\pi(s) = E\left[R(S_0) + \gamma R(S_1) + \ldots\ldots \;\middle|\; \pi, S_0 = s\right]$$

↓

"value func. for policy π"

Bellman's equations

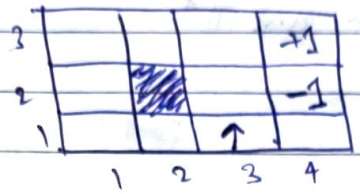
$$V^\pi(s) = R(S_0) + \gamma \sum_{s'} P_{s,\pi(s)}(s')\, V^\pi(s')$$

" immediate reward"

$$V^{\pi}(s) = E\left[R(s_0) + \gamma\left(R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) \cdots\right) \mid \pi, s_0 = s\right]$$

$$V^{\pi}(s) = R(s_0) + \gamma \sum_{s'} P_{s,\pi(s)}(s') \cdot V^{\pi}(s')$$

Given π, get a linear system of eqns. in terms of $V^{\pi}(s)$



$\pi(3,1) = \text{North}$

$$V^{\pi}(3,1) = R((3,1)) + \gamma\left(0.8 * V^{\pi}((3,2)) + 0.1 * V^{\pi}((2,1)) + 0.1 * V^{\pi}((4,1))\right)$$

□ → unknowns.

we will get **11** such linear eqns. here which we have to solve

V^* defn :→ V^* is the optimal value function

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Bellman's eqn. for V^* :

$$V^*(s) = R(s) + \max_{\pi(s)} \gamma \cdot E\left[V^*(s')\right]$$

$$\Rightarrow V^*(s) = R(s) + \max_{a} \gamma \sum_{s'} P_{s,a}(s') \cdot V^*(s')$$

↑ if we take action a.

$$\Rightarrow \pi^*(s) = \underset{a}{\text{argmax}} \gamma \sum_{s'} P_{s,a}(s') \cdot V^*(s')$$

π^* → optimal policy

property :→ $V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$

Strategy for finding π^* (optimal policy) →

1) find V^*
2) use argmax eqn. to find π^*.

Finding V^* :→

Value iteration :→ algo to find V^*

↳ Initialise $V(s) := 0$ for every s.

for every s, update :

$$V(s) := R(s) + \max_a \gamma \sum_{s'} P_{s,a}(s') \cdot V(s')$$

$$\begin{bmatrix} V(1,1) \\ V(1,2) \\ \vdots \\ V((4,3)) \end{bmatrix} := 0_{(11,1)} \rightarrow \text{0 vector}$$

$\uparrow$
$\mathbb{R}^{11}$

then we update this vector

"syncronous update" → all are updated simultaneously → used as earlier to code

"asyncronous update" → updated step by step.

Bellman backup operator.

$V := B(V)$ as each new vector updated from old one.

V converges to V^* ←

how to get π^* from V^*?

consider the solution from example

V^*

3	0.86	0.90	0.95	+1
2	0.81		0.69	−1
1	0.78	0.75	0.71	0.xx
	1	2	3	4

π^*

3	→	→	→	+1
2	↑		↑	−1
1	↑	←	←	←
	1	2	3	4

use eqn. $\pi^*(s) = \underset{a}{\text{argmax}} \sum_{s'} P_{sa}(s') \cdot V^*(s')$

Suppose we want to find $\pi^*((3,1))$

suppose we picked the action W here

$$W \Rightarrow \sum_{s'} P_{s,a}(s')\, V^*(s') = 0.8 \times 0.75 + 0.1 \times 0.69 + 0.1 \times 0.7$$

0.75 — west; 0.69 — north; 0.7 — go south and bounce off wall

$$= 0.74$$

for North it comes $N \Rightarrow 0.676$

so, going west is a better course of action at $(3,1)$.

Policy iteration :→

→ Initialise π randomly

Repeat :→ till convergence {

Set $V := V^{\pi}$ (solve bellman's eqns to get V^{π}) → using linear eqn. solving as we saw before. — computationally expensive if large set of states.

set $\pi(s) := \underset{a}{\text{argmax}} \sum_{s'} P_{sa}(s')\, V(s')$ } → we update $\pi(s)$ for each state using the values we obtained.

Continuous state MDPs

Consider the following examples,

suppose we have a car,

6 dimens.

states → $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$

suppose we have an helicopter,

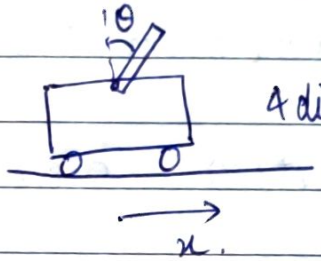
12 dimensional

states → $(x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$

roll, pitch, yaw

consider case of an inverted pendulum, we have a car with a rod with free hinge.

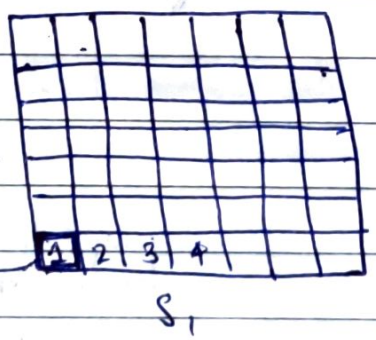
→ we want to move car in such a way that rod ~~doesn't~~ is balanced.



4 dimens. states → $(x, \theta, \dot{x}, \dot{\theta})$

suppose our problem has state space as n dimensional → $s \in \mathbb{R}^n$

Discretisation: →



1 → all values in square 1 are taken as one state.

suppose we have a two dimensional state space S_1 and S_2

↓

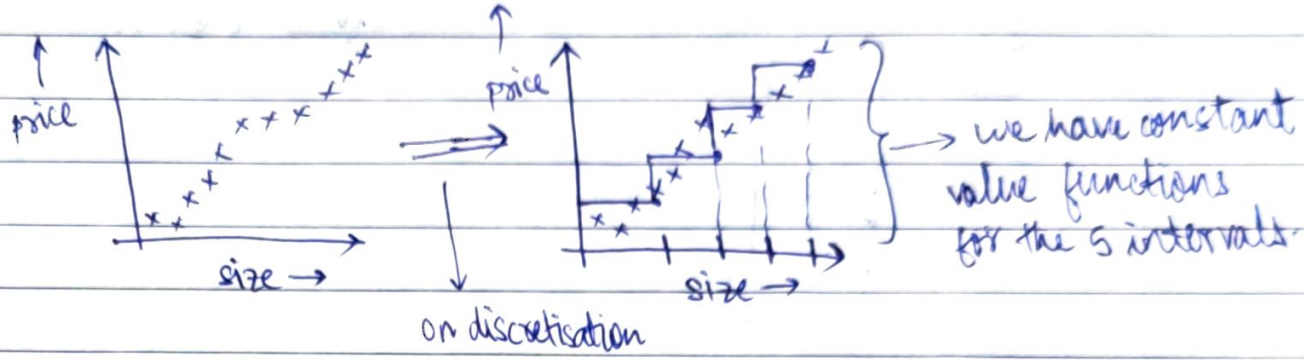
we discretise each state and convert it from continuous to discrete MDP problem.

This is a good method for low dimensional state space.

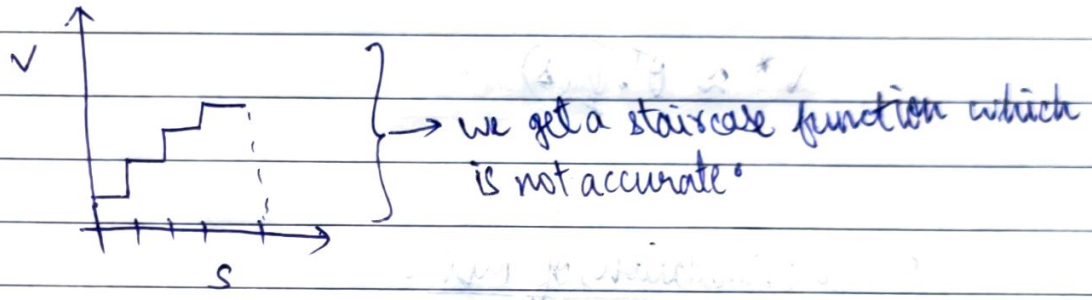
Disadvantages of discretisation :→

→ Naive representation for V^* and π^*.

consider an analogy to predicting housing price based on size.



for RL :→



→ Curse of dimensionality

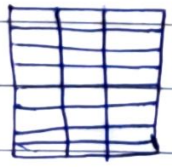
$S \in \mathbb{R}^n$, and we discretize each dimension into k values ; then we get k^n discrete states.

↳ problematic for high dimensional state space.

2 to 3 dimensions → discretisation okay

4 to 6 dimensions → discretise carefully.

7 to 8 dimensions → discretisation won't work



→ suppose S_2 has more influence on output so it is discretised finely, but S_1 has larges intervals as we want to minimise # of states.

Approximate V^* directly without discretisation

consider another analogy to linear regression;

$$y \approx \theta^T \cdot \phi(x)$$

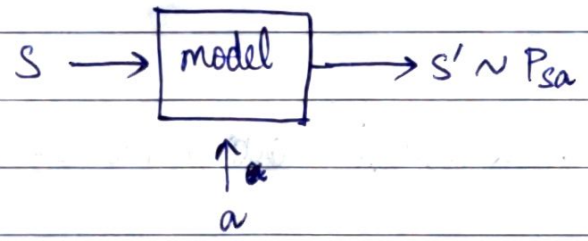
y can be approximated as linear func. of $\phi(x)$.

$$\phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \\ \vdots \\ \vdots \end{bmatrix}$$

so, here for RL we try to approx V^* as linear func. of $\phi(s)$

$$V^* \approx \theta^T \cdot \phi(s)$$

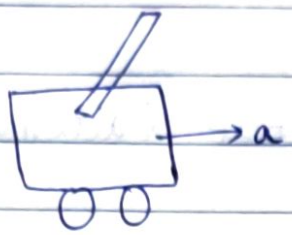
Model (simulation) of MDP:



here we are assuming, that we can discretise action space.
In most cases action space has very low dimensions as compared to state space.
Hence, here action space is discretised.

How to get a model?

→ **Physics simulator** :→ consider the inverted pendulum problem.



$$s = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix} \qquad \dot{s} = \begin{pmatrix} \dot{x} \\ \alpha - L\beta\cos(\theta)/M \\ \dot{\theta} \\ \beta \end{pmatrix}$$

simulator said so.

$$s' = s + (\Delta t)\,\dot{s}$$

$\Delta t = 0.1$ seconds or so on

→ **Learn model from data** → consider a helicopter flying problem.

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \longrightarrow \cdots\cdots \longrightarrow s_T^{(1)}.$$

basically above example → we took our helicopter out to the field and recorded the states every 10 seconds or so and the action taken to get to that state.

we do above step M times.

$$s_0^{(2)} \xrightarrow{a_0^{(2)}} \cdots\cdots$$

$$\vdots$$

$$s_0^{(M)} \xrightarrow{a_0^{(M)}} \cdots\cdots$$

} basically generating data.

Apply supervised learning to estimate s_{t+1} (s') as function of s_t, a_t (s, a)

Eg :→ Linear regression version :

$$s_{t+1} = A \cdot s_t + B \cdot a_t$$

$A \in \mathbb{R}^{n \times n}$

} okay model for slow speed helicopter

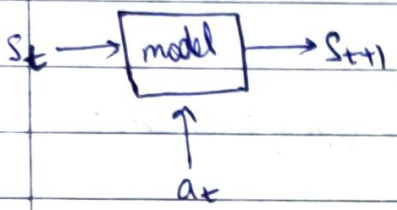
for above lin. reg model,

$$\min_{A,B} \sum_{i=1}^{M} \sum_{t=0}^{T} \left\| s_{t+1}^{(i)} - \left(A s_t^{(i)} + B a_t^{(i)} \right) \right\|^2$$

suppose if our model is $s_{t+1} = A s_t + B a_t$ (deterministic)

if it was $s_{t+1} = A s_t + B a_t + \varepsilon_t$ (stochastic)

$\varepsilon_t \sim \mathcal{N}(0, \sigma^2 I)$ helps simulates random noise like wind and such.



All Above was Model based Reinforcement Learning.

Model free RL → we let the robot free and let it crash and such.

→ good for videogame RL but not for real life RL as we don't want crashes.

→

Fitted value iteration:

choose feature $\underline{\phi}(s)$ of state s

$V(s) = \theta^T \underline{\phi}(s)$ $\quad\quad$ $\underline{\phi}(s) = \begin{bmatrix} x \\ \dot{x} \\ x^2 \\ \dot{x}\dot{x} \\ x\theta \\ \vdots \end{bmatrix}$ } example for inverted pendulum.

we can generate lots of $\underline{\phi}(s)$ from our model we obtained earlier.

Value iteration for discrete states :→

$$V(s) := R(s) + \gamma \max_a \sum_{s'} P_{sa}(s')\, V(s')$$

$$V(s) := R(s) + \gamma \max_a E_{s' \sim P_{sa}}[V(s')]$$

Fitted value iteration :→

- in lin. reg we find $x \to y$.
- here we find $s \to V(s)$
- we take some s and try to find $V(s)$ lin. relation

step 1) Sample $\{s^{(1)}, s^{(2)}, \ldots s^{(m)}\} \subseteq S$ randomly.
↳ not same m as modelling one.

step 2) initialise $\theta := 0$

step 3) Repeat {

For $i = 1, 2, \ldots m$ {

For each action $a \in A$ { → run over all actions at that state

sample $s_1', s_2', \ldots s_k' \sim P_{s^{(i)}a}$ → we take k. s' because of stochastic model

set $q(a) = \frac{1}{k} \sum_{j=1}^{k} [R(s^{(i)}) + \gamma V(s_j')]$

}

set $y^{(i)} = \max_a q(a)$

}

$\theta := \arg\min_\theta \frac{1}{2} \sum_{i=1}^{m} \| \theta^T \phi(s^{(i)}) - y^{(i)} \|^2$

}

we obtain this using $\theta^T \phi(s_j')$ → θ^T is θ from prev. iteration

for deterministic we won't do this as s_{t+1} will take a fixed value. $k = 1$ for deterministic

$s_{t+1} = A s_t + B a_t + \varepsilon_t$

because of this we take k samples as each s' is different because of Gaussian term.
↓
then we average them.

$q(a) = E[R(s) + \gamma V(s')]$

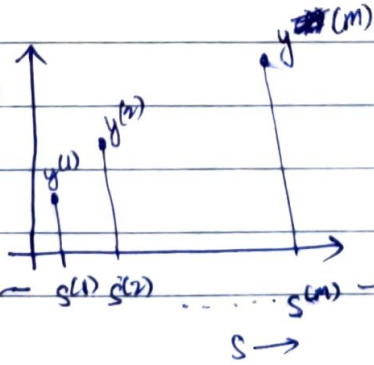
$y^{(i)} = \max_a E[R(s) + \gamma V(s')]$

fit lin. regression

$\theta^T \phi(s^{(i)}) \approx y^{(i)}$

we get θ^T from here and we can get $V(s)$ for any s using $V(s) = \theta^T \phi(s)$

basically →



generated from model

→ we took m samples here

(value iteration)

Fitted VI give approximation to V^*

we can get π^* from fitted VI

as,

$$\pi^*(s) = \arg\max_a E_{s' \sim P_{sa}} \left[V^*(s') \right]$$

we can draw k states for this to approx next state

$s_1', s_2', \ldots s_k' \sim P_{sa}$ to approximate expectation

but this is like a random numbers generator.

Say our model is of the form

$$s_{t+1} = f(s_t, a_t) + \varepsilon_t. \qquad f(s_t, a_t) = As_t + Ba_t.$$

so, during runtime we set,

$\varepsilon_t = 0$ and $k = 1$ } → we make it deterministic

so, when in state s, pick a s.t.

$$\pi^*(s) = \arg\max_a \left[V(f(s,a)) \right]$$

↑ simulation without noise

State action rewards :→

$R : S \times A \mapsto \mathbb{R}$ → reward is a function of S and A (S: states, A: action)

Total payoff $= R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) \ldots$

Bellman's eqns :→

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right)$$

$R(s,a)$: immediate reward; $\gamma \sum_{s'} P_{sa}(s') V^*(s')$: future reward

we can do value iteration as we did prev. :→ $V(s) = RHS$ (RHS from above eqn)

$$\pi^*(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right]$$

Finite Horizon MDP :→

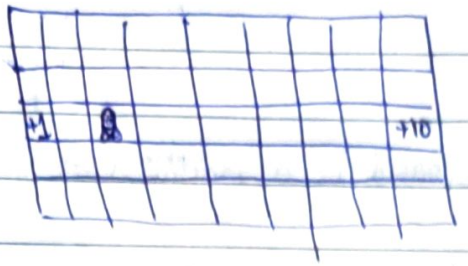
$(S, A, \{P_{sa}\}, T, R)$ → (Horizon time T) (no γ here)

example can be helicopter with limited fuel

$s_0, a_0 \rightarrow s_1, a_1 \rightarrow \ldots \rightarrow s_T, a_T$. ↑ only T transitions happen

Total payoff $= R(s_0, a_0) + R(s_1, a_1) + \cdots R(s_T, a_T)$

goal is to max. $E[R(s_0, a_0) + R(s_1, a_1) + \ldots + R(s_T, a_T)]$



action of robot here depends on horizon time.

If there are 10-12 steps it'll go to +10 ← if there are 2-3 steps left it'll go to +1.

↓

So, the policy here depends on horizon time T

$\pi^*(s)$ → is a function of t here

↓

$\pi^*_t(s)$ → non-stationary policy → as it changes over time

Non stationary state transitions:

$S_{t+1} \sim P^{(t)}_{s_t a_t}$ the state transitions here are a function of time here

example :→ driving car from A to B → depends on time i.e. rush hour and so on.

Non-stationary rewards:

$R^{(t)}_{s,a}$ → reward is a function of time

when to use non-stationary ~~models~~ MDPs?

→ changing dynamic → ~~it~~ in airplanes, 1/3rd weight is fuel, so it is lighter when landing compared to takeoff

→ weather forecast → depending on whether it'll rain and ~~so~~ all.

→ Industrial automation → It is easier to get labour at 3pm than 3am → so this too is non stationary.

$$V_t^*(s) = E\left[R(s_t, a_t) + R(s_{t+1}, a_{t+1}) + \cdots + R(s_T, a_T) \,\middle|\, \pi^*, s_0 = s \right]$$

Expected total payoff starting in state s_t at time t executing π^*

Value iteration: (Dynamic programming)

$$V_t^*(s) = \max_a \left(R(s,a) + \sum_{s'} P_{sa}(s')\, V_{t+1}^*(s') \right)$$

$$\pi^*(s) = \operatorname{argmax}_a \left(\quad // \quad \right)$$

defines V_t as function of V_{t+1}

at time T final V^* is V_T^*

$$\boxed{V_T^*(s) = \max_a (R(s,a))}$$ → as we can't take further steps after time T

so, we calculate V_T^* first then we find

$V_{T-1}^*, V_{T-2}^* \cdots V_0^*$

then we find π_t^* using V_t^* here $0 \le t \le T$

Linear Quadratic Regulation (LQR):

condn. for LQR: → finite horizon

$(S, A, \{P_{sa}\}, T, R)$

$\omega_t \sim N(0, \Sigma_\omega)$

$S = \mathbb{R}^n \quad A = \mathbb{R}^d \quad P_{sa}: S_{t+1} = \underset{\mathbb{R}^{n\times n}}{A} S_t + \underset{\mathbb{R}^{n\times d}}{B} a_t + \omega_t$

$R(s,a) = -(s^T U s + a^T V a)$ where $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{d\times d}$

$U, V \ge 0$ (positive semi definite) ← implies

$s^T U s \ge 0, \; a^T V a \ge 0$

If we want $s \approx 0$ → i.e. want our helicopter to hover

and if we choose $U = I$, $V = I$.

$R(s,a) = -(\|s\|^2 + \|a\|^2)$ → basically penalise state s and action a from deviating from zero.

penalises large jerky motions

if we want non-stationary transitions then;

$A, B \xrightarrow[\text{with}]{\text{replace}} A^{(t)}, B^{(t)}$

if we want non stationary Rewards;

$U, V \xrightarrow[\text{with}]{\text{replace}} U^{(t)}, V^{(t)}$

where to get A, B?

run our helicopter enough m trajectories of time T.

$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} \cdots \longrightarrow s_T^{(1)}$

$\vdots$

$s_0^{(m)} \xrightarrow{a_0^{(m)}} \cdots \cdots \cdots \cdots s_T^{(m)}$

"Learning from data here"

$$s_{t+1} \approx A s_t + B a_t$$

$$\min_{A,B} \sum_{i=1}^{m} \sum_{t=0}^{T-1} \left\| s_{t+1} - (A s_t + B a_t) \right\|^2$$

↓

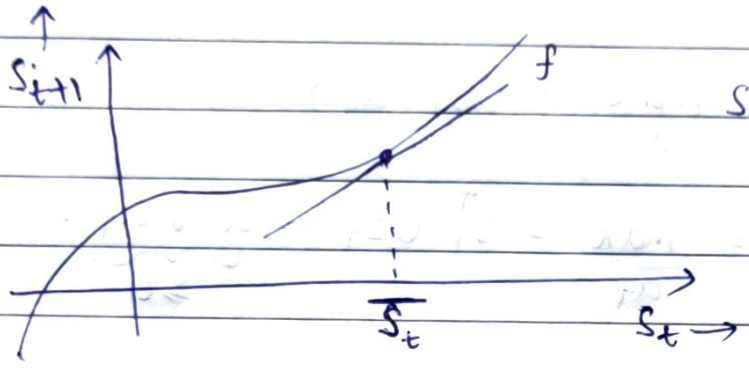
like linear regression

Second method to get A, B → Linearize a non linear model :→

consider our case of inverted pendulum :→

$S_{t+1} = f(S_t, a_t)$ → suppose our physics friend helped with this

$$\begin{pmatrix} x_{t+1} \\ \dot{x}_{t+1} \\ \theta_{t+1} \\ \dot{\theta}_{t+1} \end{pmatrix} = f\left(\begin{pmatrix} x_t \\ \dot{x}_t \\ \theta_t \\ \dot{\theta}_t \end{pmatrix}, a_t \right)$$



$$S_{t+1} \approx f'(\bar{S}_t) \cdot (S_t - \bar{S}_t) + f(\bar{S}_t)$$

$\bar{S}_t$ here is constant.

"typical value"

suppose we want our helicopter to hover, so $\bar{S}_t$ will be 0 here as S_{t+1} will be close to $\bar{S}_t$

Suppose for our inv. pendulum, we assume bar almost upright and low speeds, $\bar{S}_t$ will be zero

in general, $S_{t+1} \approx f(\bar{S}_t, \bar{a}_t) + \left(\nabla_s f(\bar{S}_t, \bar{a}_t)\right)^T (S_t - \bar{S}_t) + \left(\nabla_a f(\bar{S}_t, \bar{a}_t)\right)^T (a_t - \bar{a}_t)$

$S_{t+1} = A S_t + B a_t$

we may need to add an intercept term to $S_t \rightarrow \begin{pmatrix} 1 \\ x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix}$ because of constants.

problem we want to solve:

$\omega \sim \mathcal{N}(0, \Sigma_\omega)$ ← noise

$$S_{t+1} = AS_t + Ba_t + \omega_t.$$

$$R(s,a) = -(s^T U s + a^T V a)$$

→ if these condns are met we can compute exact V^*

total payoff $= R(s_0, a_0) + R(s_1, a_1) + \cdots R(s_T, a_T)$

Dynamic programming solution to find V^*

$$V_T^*(S_T) = \max_{a_T} R(S_T, a_T)$$

$$= \max_{a_T} -S_T^T U S_T - \underbrace{a^T V a_T}_{\to 0}.$$

so max. → we choose $a_T = 0$

$$= -S_T^T U S_T$$

$$\pi_T^*(S_T) = \vec{0}$$ as

Inductive step:→

suppose, $V_{t+1}^*(S_{t+1}) = S_{t+1}^T \Phi_{t+1} S_{t+1} + \Psi_{t+1}$

where, $\Phi_{t+1} \in \mathbb{R}^{n \times n}$, $\Psi_{t+1} \in \mathbb{R}$

$$V_t^*(S_t) = \max_{a_t} \left(R(S_t, a_t) + E_{S_{t+1} \sim P_{S_t, a_t}} \left[V_{t+1}^*(S_{t+1}) \right] \right)$$

this format as state transition is continuous.

$$V_t^*(s_t) = \max_{a_t} \left(-s_t^T U s_t - a_t^T V a_t + \underbrace{\left[\underbrace{\left[s_{t+1}^T \Phi_{t+1} s_{t+1} + \psi_{t+1} \right]}_{V_{t+1}^*(s_{t+1})} \right]}_{s_{t+1} \sim \mathcal{N}(As_t + Ba_t, \Sigma_w)} \right)$$

$\downarrow$ comes out to be a big quadratic func. of a_t

take derivatives $\swarrow$ and solve for a_t

$$a_t = \underbrace{\left(B^T \Phi_{t+1} B - V \right)^{-1} B^T \Phi_{t+1} A}_{L_t} \cdot s_t$$

$$\pi_t^*(s_t) = L_t \cdot s_t$$

$\therefore$ optimal action is a linear function of s_t and

$$V_t^*(s_t) = s_t^T \Phi_t s_t + \psi_t$$

where,

$$\Phi_t = A^T \left(\Phi_{t+1} - \Phi_t B \left(B^T \Phi_{t+1} B - V \right)^{-1} B \Phi_{t+1} \right) A_t - U$$

$$\psi_t = -tr(\Sigma_w \Phi_{t+1}) + \psi_{t+1}$$

To summarize,

initialize $\Phi_T = -U$ and $\psi_T = 0$

recursively calculate, Φ_t, ψ_t using Φ_{t+1}, ψ_{t+1} (for $t = T-1, T-2 \cdots 0$)

calculate L_t

$$\pi^*(s_t) = L_t s_t$$

Policy search & POMDPs

("direct policy search") → try to find good π directly.

normally in RL we first find V^* then we find π^*

consider the case of inverted pendulum → with two actions
↓
left & right

$$s = \begin{pmatrix} 1 \\ x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{pmatrix}$$

set of
we want to come up with functions to approximate the policy.

$$\pi_\theta(s) = \frac{1}{1+e^{-\theta^T s}}$$

→ say / assume
→ like how we did for logistic func.

Stochastic policy → function $\pi: S \times A \mapsto \mathbb{R}$ where $\pi(s,a)$ is the prob. of taking action a in state s.

$$\left(\sum_a \pi(s,a) = 1\right)$$

→ basically on each timestamp give probs. that you want to take action a.

↓ for this example consider something like

$$\pi_\theta(s, \text{"R"}) = \frac{1}{1+e^{-\theta^T s}}$$

$$\pi_\theta(s, \text{"L"}) = 1 - \frac{1}{1+e^{-\theta^T s}}$$

this is a reasonable policy as consider $\theta = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, so $\theta^T s = \phi$

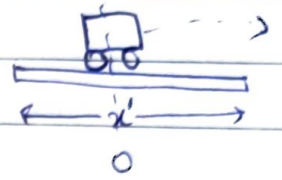
if $\phi \gg 0$ → it moves more aggressively to right
if $\phi \ll 0$ → move " " " left.

however, this isn't a good policy as it ignores features other than $\dot{\phi}$,

if, $\theta = \begin{bmatrix} 0 \\ -0.5 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ $\quad \theta^T s = -0.5x + \dot{\phi}$

→ helps stay cart in an interval

if position more to right than more prob. of going left



goal :→ Find θ so that we execute $\pi_\theta(s,a)$, we maxm.

$$\max_\theta E\left[R(s_0,a_0) + \cdots + R(s_T,a_T) \middle| \pi_\theta\right]$$

or we assume initial state has a fixed distribution

here s_0 is fixed initial state. → assumption for direct policy search

↳ in general RL ~~we~~ s_0 is not a fixed state → we see for all states

$$\max_\theta E\left[R(s_0,a_0) + R(s_1,a_1) + \cdots + R(s_T,a_T) \middle| \pi_\theta\right]$$

↳ using $T=1$ to simplify the math and not write such a long summation

↳ all derived results work for longer summation too

$$= \sum_{s_0 a_0 s_1 a_1} P(s_0,a_0,s_1,a_1)\left[R(s_0,a_0) + R(s_1,a_1)\right]$$

all possible state and action sequence.

payoff

initial state dist. · state transition prob to s_1

$$= \sum_{s_0 a_0 s_1 a_1} P(s_0)\,\pi(s_0,a_0)\,P(s_1)\,\pi(s_1,a_1)\cdot[\text{payoff}]$$

$s_0 a_0$

prob. of choosing action a_0

prob. of picking action a_1

we will use gradient ascent as func. of θ to maxm. above funct.

Reinforce algorithm to maxm. total payoff :→

Loop : {

Sample : $s_0, a_0, s_1, a_1, \ldots\ldots s_T, a_T$ → random state seq. generated → run it for T timestamps

compute payoff $= R(s_0) + R(s_1) + \ldots\ldots + R(s_T)$

Update : $\theta := \theta + \alpha \left[\frac{\nabla_\theta \pi_\theta(s_0, a_0)}{\pi_\theta(s_0, a_0)} + \frac{\nabla_\theta \pi_\theta(s_1, a_1)}{\pi_\theta(s_1, a_1)} + \ldots\ldots \right] \times$ payoff

payoff of the random seq. used ← multiplied by payoff

}

this is stochastic gradient ascent

i.e. bounces all over the place while on an avg. converging ← because of the random sequence ~~used~~ used.

on an avg. in same dirn. as actual gradient.

$$\nabla_\theta \sum_{s_0 a_0 s_1 a_1} p(s_0)\, \pi_\theta(s_0, a_0)\, P_{s_0 a_0}(s_1)\, \pi_\theta(s_1, a_1) \cdot [\text{payoff}]$$

$$= \sum_{s_0 a_0 s_1 a_1} \underbrace{p(s_0) \cdot \pi_\theta(s_0, a_0)\, P_{s_0 a_0}(s_1)\, \pi(s_1, a_1)}_{p(s_0 a_0 s_1 a_1)} \left[\frac{\nabla_\theta(\pi_\theta(s_0, a_0))}{\pi_\theta(s_0, a_0)} + \frac{\nabla_\theta(\pi_\theta(s_1, a_1))}{\pi_\theta(s_1, a_1)} \right] \times [\text{payoff}]$$

$$= \sum_{s_0 a_0 s_1 a_1} p(s_0 a_0 s_1 a_1) \left[\frac{\nabla_\theta(\pi_\theta(s_0, a_0))}{\pi_\theta(s_0, a_0)} + \frac{\nabla_\theta \pi_\theta(s_1, a_1)}{\pi_\theta(s_1, a_1)} \right] \times [\text{payoff}]$$

$$= \sum_{s_0 a_0 s_1 a_1} p(s_0 a_0 s_1 a_1) [\text{gradient update}].$$

direct policy search is better than finding V^* then π^* in 2 settings :→

(i) POMDP

↑ partially observable MDP

↓

at each step, we get a partial (and potentially noisy) measurements of the state. Have to choose action according to these incomplete measurements.

ex :→

$\rightarrow x$

$s = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix}$

↓

but suppose we can only measure x, θ

← so it is POMDP

(ii) is π^* simpler or is V^* simpler.

low level control task → can a skilled human do the task.

↳ flying a helicopter

↳ balancing inverted pendulum.

playing chess → not low level → have simpler π^* so better to use this.