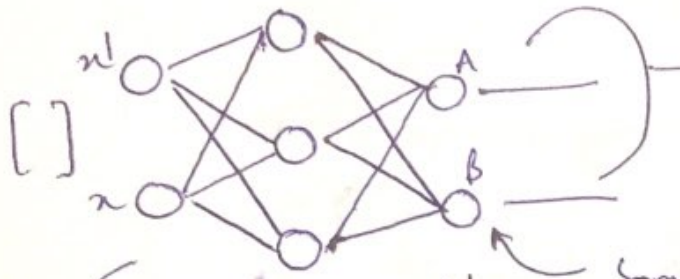
partial derivatives of output with respect to parameters (input) and should be computable.

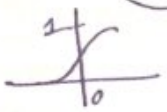
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial x_3} \cdot \frac{\partial x_3}{\partial x_2} \cdot \frac{\partial x_2}{\partial x_1}$$

"Chain rule"

then we can apply chain rule to compute E wrt any of the parameters.



→ independent predictions
— "given a sample, you are not forced to categorize as sample A or B"



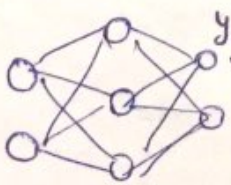
Sometimes we dont use a nonlearity in the output layer
↳ If we dont want to scale down to [0 1]

given samples, (email)
↓
classification rule ≡ for A → $\begin{cases} 1, & \phi(\sum w_i x_i) > 0 \\ 0, & \text{else} \end{cases}$

[Suppose, the email gets classified as ~~personal~~ spam by both
↳ use "softmax" across neurons

Ex: [A — 0.49 (value)
B — 0.001]

* Learn w?



[2×3] [3×2]

≡ Error = $[\hat{y} - y^i]$
↑
minimise the error
— adjust the weights

[(x_i, y_i), $i = 1, N$
↓
Example set

→ Find optimal w!

Classic gradient descent

$$w^{k+1} \leftarrow w^k - \eta \frac{\partial L}{\partial w}$$

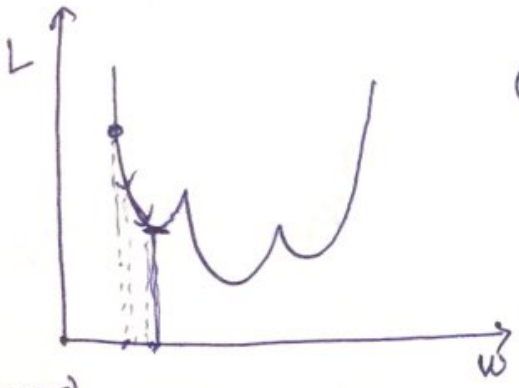
gets modified to ↓

One class problem

Classifier?

has a structure

negatives — have no structure ~ reject

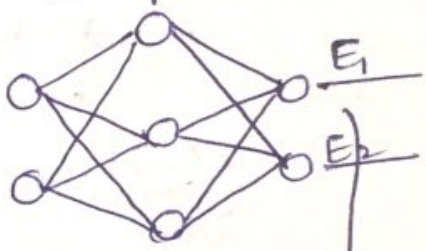


(Error)

← Not a convex optimisation problem

3×2

$Y = W'X \Rightarrow Y = \phi(Y)$



⇒ E_1 in the output layer indirectly depends on the first layer weights

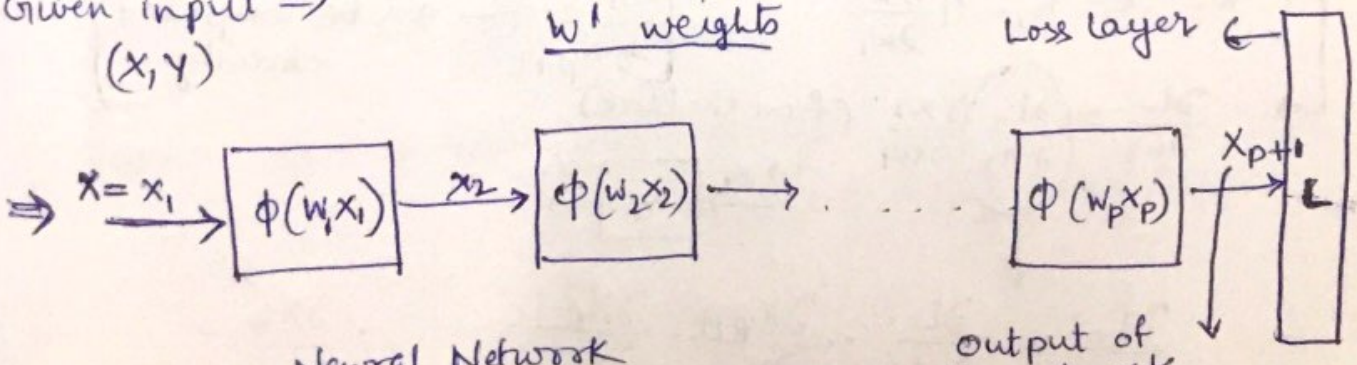
→ adjust the weights and go to better solution in steps (less error)

Error here doesn't solely depend on w^2 weights but also on the outputs of layer 1

outputs of this layer depends on w^1 weights ←

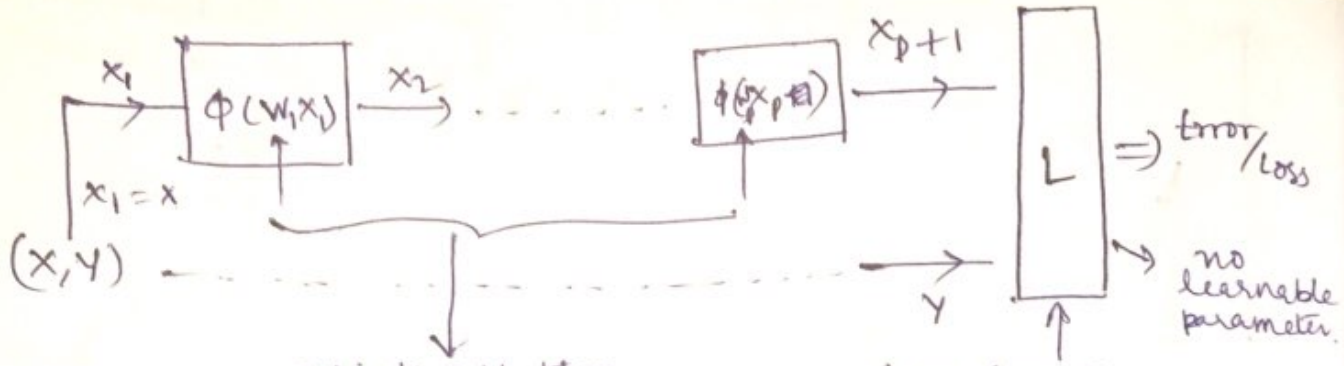
Given input ⇒ (X, Y)

Loss layer ←



Neural Network

output of the network.



adjust all these weights

Loss layer

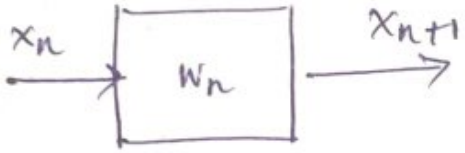
loss function can be anything.

$\|X_{p+1} - Y\|$

$$* \quad W^{K+1} \leftarrow W^K - \eta \left(\frac{\partial L}{\partial w}\right)$$

→ perform the update for all the w's in the network.

one layer:-



$$x_{n+1} = \phi(w_n x_n)$$

we need –

1.) $\frac{\partial x_{n+1}}{\partial x_n}$

2.) $\frac{\partial x_{n+1}}{\partial w_n}$.

* if we want to update w_1, we will need $\left[\frac{\partial L}{\partial w_1}\right]$

Computing loss ≡

$$L = \sum_{i=1}^{N} \left[x^i_{p+1} - y_i\right]^T \left[x^i_{p+1} - y_i\right]$$

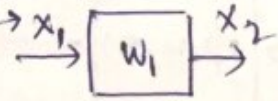
i^{th} sample.

sum of squared differences of the predicted and actual output.

$$w_1^{k+1} \leftarrow w_1^k - \eta \frac{\partial L}{\partial w_1}$$

$\left[\frac{\partial L}{\partial x_{p+1}}\right]$ ← can be computed directly

$$\frac{\partial L}{\partial w_1} = \left(\frac{\partial L}{\partial x_2}\right) \frac{\partial x_2}{\partial w_1}$$ (from the block)



$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial x_{p+1}} \cdot \underbrace{\frac{\partial x_{p+1}}{\partial x_p} \cdot \frac{\partial x_p}{\partial x_{p-1}} \cdots \frac{\partial x_3}{\partial x_2}}_{\text{diff output wrt input} \rightarrow \text{from the block.}}$$

To update a weight, we need to know the derivatives of the blocks before.

Ex:

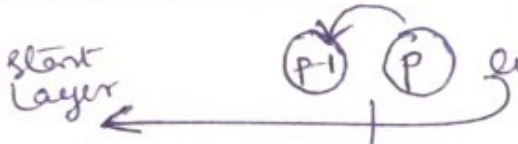
$$w_p^{k+1} \leftarrow w_p^{k} - \eta \frac{\partial L}{\partial w_p}$$

last block

$$\frac{\partial L}{\partial w_p} = \frac{\partial L}{\partial x_{p+1}} \cdot \frac{\partial x_{p+1}}{\partial w_p}$$

↑ can be computed directly.

Back propagation — order ≡ { back to front }



≡ to make easy computations
↓
removing redundant computations.

computation of p^{th} block is used in $(p-1)^{th}$ block.

↓

In "chain rule"

Pseudo code ≡

>1. Initialise $w_1 .. w_p$ randomly

>2. Compute loss for all N samples. (forward propagation)

>3. Update w_i;

$$w_i^{k+1} \leftarrow w_i^{k} - \eta \left(\frac{\partial L}{\partial w_i}\right), \quad i = p, \ldots 1$$ (backward pass).

(or) $i = 1, \ldots p$ [wastage of computation] ×

product of partial derivatives.

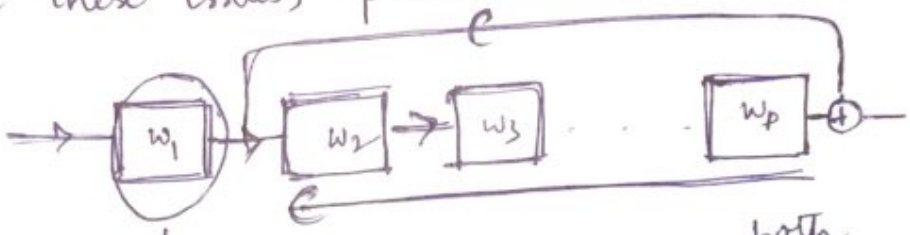
>4. Repeat step 2-4 until convergence

(No significant change in loss/weight

* Vanishing gradient problem } → as you propagate, the gradient might diminish
* Exploding gradient problem

for these issues; provide skip connections

skip connections
⇓
Not dynamic architecture



changing this w_1 → depends on both the channels

[If one of the gradients in chain rule becomes zero
↓
very dangerous ⇒ because output doesn't depend on the input and also the previous blocks
might be due to convergence.]

Back Propagation Tricks:

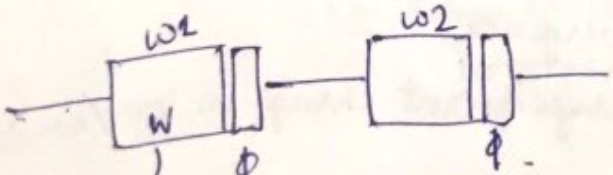
$\frac{1}{1+e^{-ax}} \equiv$  curve changes wrt a.

we can have —
learnable parameters in the non-linearity 

J/L → loss function.

if $\frac{\partial L}{\partial o/p}$ is available → then, $\frac{\partial L}{\partial i/p} = \frac{\partial L}{\partial o/p} \boxed{\frac{\partial o/p}{\partial i/p}}$

↓
should be known.



$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial o/p} \boxed{\frac{\partial o/p}{\partial w}}$ → after computing this,

update rule ≡

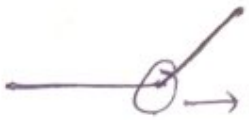
$w^{k+1} \leftarrow w^k - \eta \frac{\partial L}{\partial w}$.

sigmoid and tanh activation functions

↓

the derivatives of such functions can be expressed in terms of the original function.

↓

New activation function ≡ ReLU ⇒  → continuity?

↓

derivative is easy to compute.

SVM Loss function → [enforcing margin] → hinge loss

using "subgradients"

↓

derivated evaluated at x

$\phi'(x)$

max (0, 1 − t.o)

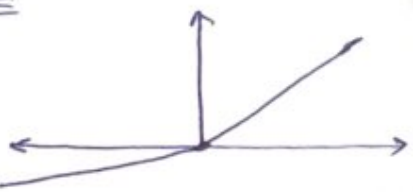
↑ +1 ↑ $w^T x$

~ similar to ReLU

↑

loss can be computed at any point (derivative)

Leaky ReLU ≡



Stochastic

$\frac{\partial L}{\partial w}$ → loss on the samples that we have access to.

↓

so the estimate can be calculated over a subset

↓

Assume: out of billion samples → take random 100 → if the loss is same for any random 100

← then there is no point in considering loss over all billion samples.

← so we look at a subset

$\frac{\partial L}{\partial w}$ ← "over a subset"

← "batch" – extremely reliable but slow

← "single sample" – not reliable

"mini batch"

$\frac{\partial L}{\partial w}$ → computing derivative for every sample and adding

↳ $\frac{\partial}{\partial w}\sum_i ( )$ → $\sum_i \frac{\partial ( )}{\partial w}$ ∵ summation of each derivative.

that's why we look at mini batch, 'batch' ↑ computationally not attractive.

Algorithm ≡

1. Initialise
2. Forward pass and compute "Loss"
3. Back propagate and update "w"
4. Repeat 2-4 until convergence.

MSE, $L \geq 0$

$L^0, L^1, L^2, L^3 \ldots L^i, L^{i+1}$

moving across negative gradient

$L^i > L^{i+1}$

Problem = Non convex in nature ‖ Loss function

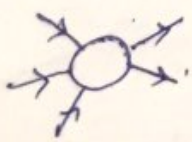
* 1.

Initialisation – 'dont know where to start?'

↳ pretraining – similar but different problem ↓ can give a good hint of the starting point.

↳ picking "w" of appropriate sizes.

↳ Random Initialisation –



— network can have many inputs/outputs "depends on number of inputs/outputs" ⇓

↳ Xavier's Initialisation
↳ He's Initialisation. (heuristics)

Update Rule :- $\theta^{k+1} \leftarrow \theta^k - \eta \frac{\partial L}{\partial \theta}$ (gradient based update)

η (eta) can be controlled.

Momentum] previous change is added to the update rule.

idea: ball on the surface, moving in the same direction ≡ pickup velocity. momentum ↑

$\theta^{k+1} \leftarrow \theta^k - \eta \frac{\partial L}{\partial \theta} + \gamma (\text{Prevdirecn})$.

→ if previous direction of motion is same, we add them else, they cancel each other and become slow

⇒ Reinforces the continuity

Nesterov Momentum — look ahead prediction of gradient.

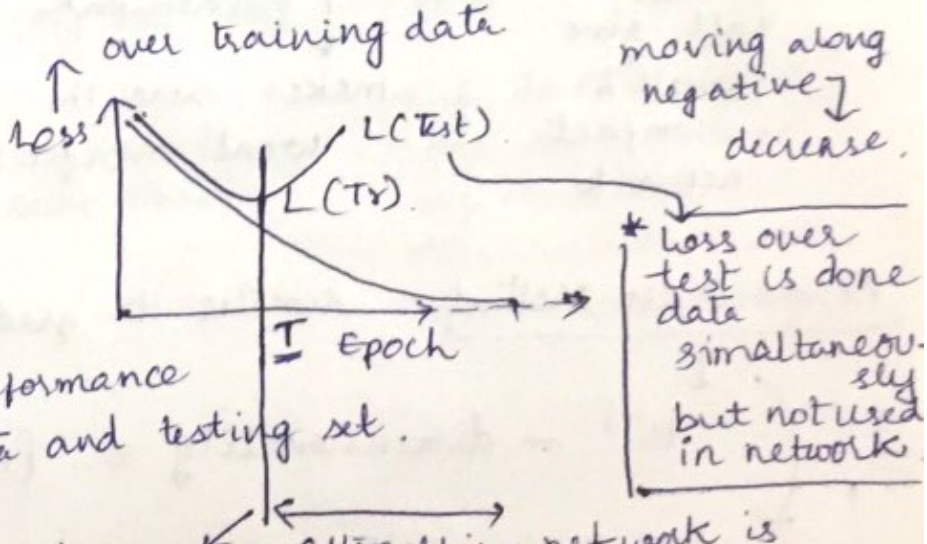
issue: skips the minima and climbs on

Adaptive Gradient Change

- Ada delta
- Ada grad
- RMS ~~delta~~
- Adam

Termination :-

- overfitting



* should give same performance on both training data and testing set.

L(Test) need not be smooth.

Overfitting → neural network – is "over parametrised".

L(Test) → not a minimum, because it is not a smooth curve.
↓
has no distinct property (might have zig-zags).

has extra weights.

to counter "over parametrised" ⇒
add a norm of weight $J' = J + \|\theta\|$ ← learnable parameters.

∴ Small network – cannot byheart, so we reduce the parameters

$$J' = J + \|\theta\|$$

$$\frac{\partial J'}{\partial \theta_1} = \frac{\partial J}{\partial \theta_1} + \frac{\partial \|\theta\|}{\partial \theta_1}$$

L0 (nonzero) L1 L2 ⇒ norms.
L2 ✓
L0 → not easily differentiable.

→ weight decay – or Regularisation → will give small & compact network.

Remove small θs in the network → pruning the network → makes sure the network wont overfit.

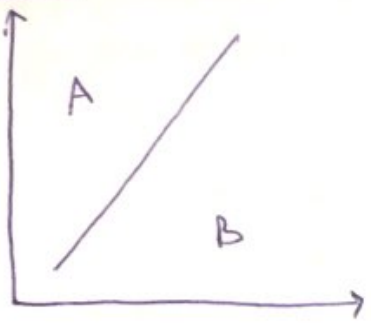
* If w is very small, it can be removed from the network. → Not much change.

Second Order Method – similar to gradient descent.

H^{-1} – dimensionality ≡ [no. of parameters]^2

parallelisability vs memory (very high)
↓
tradeoff

Anoop :)



→ how A-like

$d_A(x)$
$d_B(x)$
$d_C(x)$
$d_D(x)$

} linear classifiers → multi-class classification.

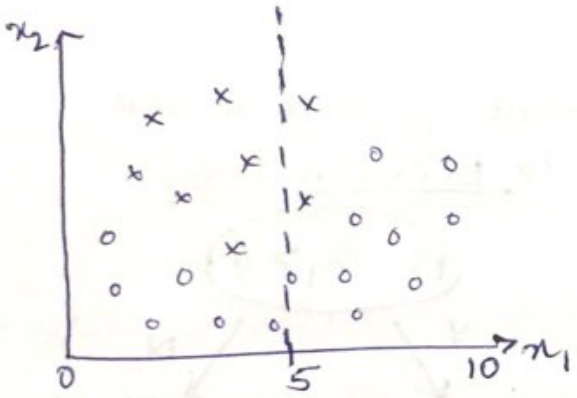
SVMs → discriminant classifier. "learns a line".

multi layer perception – generative model.

decision trees (~ 20 questions game).

↳ for cat/non-cat classification [length of the tail < 5cm?]

↓ answers of these questions – to decide the class.



→ 2 feature (2D feature space).

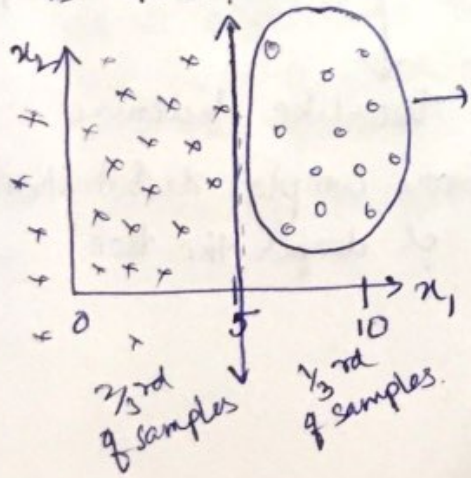
Is $x_1 < 5$? → Not a good question

↓ won't be able to classify x, o.

[Question :- answer] → reduction in feature space of set of samples

(best question)

Is $x_1 > 5$? → not exactly reducing into half "optimal"



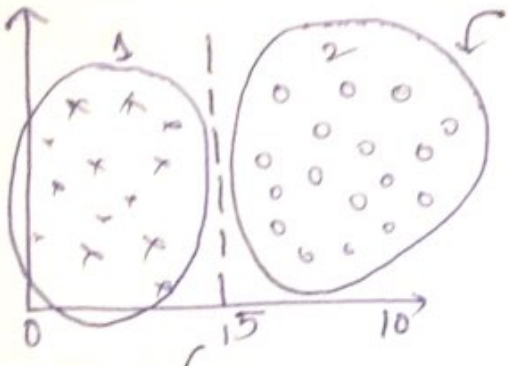
→ all belong to the same class

→ if due to a split, one side contains one class and the other contains the other class – then done!

Entropy – randomness / uncertainity.

$$H(x) = -\sum_i P_i \log_2 P_i \quad) i \rightarrow \text{class}.$$

high entropy - high uncertainity — when a sample is picked from the whole set of samples.



when a sample is taken from a side, entropy is zero, as all the samples on one side belong to the same class

on side-2 =

$H = -P_1 \log_2 P_1 + -P_2 \log P_2 = 0$ (Entropy)

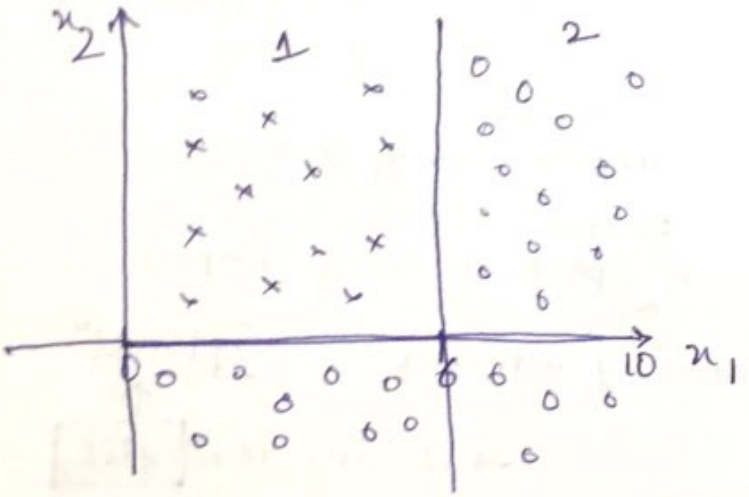
$0 \times \infty$ $\quad 1 \times 0$

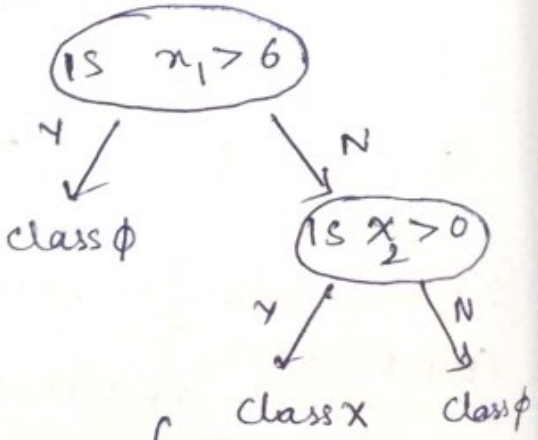
these type of classifiers are called "decision trees".

* If the data is mixed up,

→ with one question → entropy doesnt go down to zero.

⇒ Decision Tree.





↓ series of questions ∾ classifier

↓ tree-like structure

[more complex distribution ∝ deeper the tree]

* what questions to ask?
* when to stop?

↓

Compute total entropy

∾ chose questions such that it reduces the entropy

Mathematically, entropy cannot go higher.

half-half samples (x, o) ⟹

→ Entropy, $H = -[0.5 \times -1 + 0.5 \times -1]$ (Total Entropy)

$= 1$ (before splitting).

→ after splitting, ($2/3^{rd}$ x, $1/3^{rd}$ o)

$H_o = -1 \log 1 + 0 \log 0$

$= 0.$

on one side

$H_x = -2/3 \log 2/3 - 1/3 \log 1/3$

$= 0.72$

on the other side

H (total) $= 0.72$

Entropy reduced from $1 \rightarrow 0.72$. (with just one question).

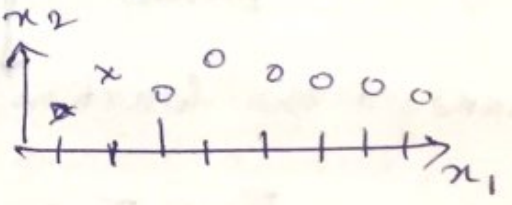
⟹ How to decide which question to ask?

* How many questions? (infinite). → but answers might not be infinite. ~~same~~.

⟹ splits possible? (Not infinite)

— all possible values of x_1

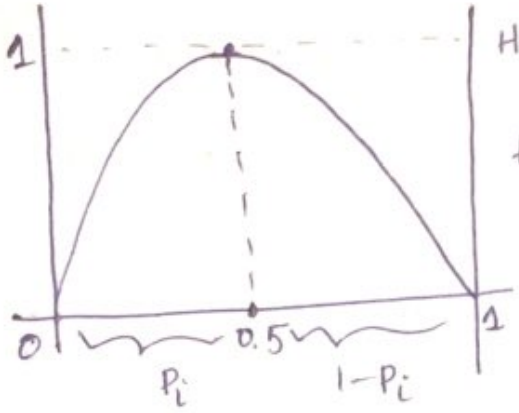
if n samples in the database, n−1 splits



* decision trees are the worst algorithms to train but once trained → they are the fastest to test

* [extremely inefficient to train, extremely efficient to test]

H vs P_i (change in entropy function wrt P_i).



Entropy $= -\Sigma P \log P$

(valid for 2 classes)

Impurity ≡ Measure [If pure, low value; if highly mixed up; high value]

Entropy function $= -\Sigma P \log P$ ← Calculating entropy for every question — became difficult.

- Entropy function → computationally expensive
- $\log P$ → hard to compute

$P \log P$ — ?

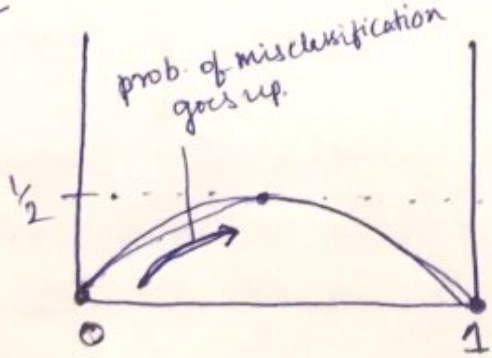
- P → probability of sample being picked

changed the function ⇒ GINI IMPURITY.

(not ~~Entropy~~).

$= \sum_i P_i \sum_{K \neq i} P_K$

- P_K → prob. of misclassifying ≡ Prob doesn't belong to i $= 1 - P_i$
- highly pure, 0; mixed up, ½



(valid for 2 classes)

$$\sum_i p_i \sum_{k \neq i} p_k = \sum_i p_i (1 - p_i)$$

$$\text{gini impurity} = \sum_i p_i - \sum_i p_i^2 = 1 - \sum_i p_i^2$$

($\sum_i p_i = 1$)

↓ computed millions of times (for each question)

↓ We can learn the set of questions to be asked.

~ much better than the computation of $\log p_i$

~ other classifiers assumed euclidean space, decision trees don't care [categorical data].

↓ impurity can be computed.

generally, categorical data → converted to → Numerical data ~ most of the classifiers.

* Advantages :-

→ Efficient in practice

→ Handles categorical data very well.

→ Tells you the reasons (~answers), about the classification. → human interpretable (decisions - answers).

[this is the problem of deep neural network.]

Eg: length of the tail, whiskers or not?

→ Tells you what features to look for.

[feedback about what features to collect]

↳ Importance of features

* Limitation :-

→ Training time (expensive to train)

→ Not good with "Numerical" data

(split cannot give a line → ↙↘ only axis parallel.)

→ Extreme case of overfitting → decision trees can fit any complex distribution.
can happen

⇒ Stop overfitting? — stop the growth of the tree.

→ Early stopping (threshold on the entropy).

→ Pruning (chop of the bottom till you reach simplicity)

how? training - validation

semitesting [testing data while training]

when the accuracy of validation starts to drop [due to overfitting of training data] = you stop!

* Missing data can create issues in decision trees (you cannot skip questions) as you wont be able to go ahead.

Missing data ≡ average of features (or) value of previous data.

* Random Forest Classifiers → built on decision trees.

Multiple classifiers (neural network, SVM, decision tree, bayesian ...)

↳ $h_1(x)$, $h_2(x)$, $h_3(x)$ } 3 different classifiers for an input, X

↳ how to learn from all of the classifiers (even though few classifier's performances are bad).

→ let suppose the accuracies are given;

- $h_{NN}(x) = 92\%$
- $h_{bayes}(x) = 90\%$
- $h_{SVM}(x) = 88\%$

} — one choice is to take the best out of all the classifiers. (high accuracy) ≡ Majority Vote.

→ Majority Vote – due to its simplicity, it works with any classifier.

→ Weighted Voting – mixing the accuracy with the results.

Ex:

NN → 0 → 0

Bayes → 0 → 1 → 1

SVM → 0 → 1 → 1

$$y_K = f(y_1, y_2, y_3)$$

$$= \left\{ \frac{92}{270} \times 0 + \frac{90}{270} \times 1 + \frac{88 \times 1}{270} \right\}$$

$$= \left\{ \frac{178}{270} > 0.5 \right\} \equiv 1.$$

In general; $y' = f(y_1, y_2 \dots y_n, \varepsilon_1, \dots \varepsilon_n)$

where $y_1, y_2 \dots y_n$ – outputs of n classifiers

$\varepsilon_1, \varepsilon_2 \dots \varepsilon_n$ – errors of classifiers.

$$\equiv \frac{\sum_K y_K (1 - \varepsilon_K)}{\sum_K 1 - \varepsilon_K} < 0.5$$ ↗ if ↘ else. for classification.

→ Classifier not only gives the output, but also the a (a single number) confidence measure ≡ (if the classifier can output it).

$y \Rightarrow f(y_k, c_k) \equiv \Rightarrow \frac{\sum_k y_k c_k}{\sum_k c_k}$ ~ determines the result.

↑ confidence measure

→ Feature level Fusion = (information fusion for classification)

each classification is trained on a different representation of features.

can learn the function.

C

* $y = f(y_k, c_k)$

↓

train a second level classifier

Stack all the outputs and train a classifier based on it.

$[c_1, c_2 \cdots c_k]$ into a feature vector.

* Will combining many classifiers improve the result?

↳ Accuracy doesn't go much beyond the best classifier accuracy. (best of the experts).

it works good with a group of layman.

→ Strong classifier

→ Weak classifier.

* Combining a large no. of weak classifiers can give a better result than a group of strong classifiers (ML justification).

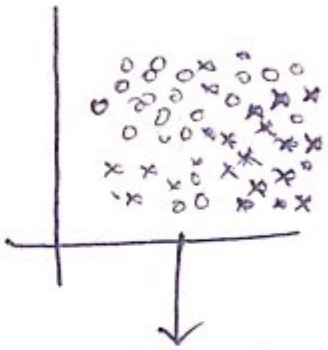
- Methods to combine a group of classifiers to give a single classifier ≡ Ensemble methods.

→ Bagging : Bootstrap Aggregation.

↓

Sampling on the training data

⇒ Since we cant train all the classifiers on the same data [give similar results]



N sampling

→ created a subset

[called as bootstrapping]

fair (random sampling) → sample N times, with replacement

[to make sure two data sets are diff]

[selections – independent]

N samplings

D → make bootstraps

D_1 — h_1 on 1st iter

D_2 — h_2 on 2nd iter [if you sample N times again]

⋮

D_{100} — h_{100} → Train 100 classifiers on these 100 datasets and average the results.

⇒ if we sample less than N times, variability will be high.

⇒ if we sample 100 times, the expected no. of times each sample will be selected is 100.

↓

N done to create variations.

→ Boosting [works better in diversity] (not uniform)

↳ weak classifiers but with complementary strengths are combined.

⇒ After training h_1 on D_1, test it on the training data D

→ Now, testing on D might give classification errors
— modify the sampling for D_2 ✓

↳ we assign some weights to the samples

$$\delta_1(x) = 1/N \quad \forall i$$

$$\delta_2(x) = \begin{cases} e^{-\alpha_t} \\ e^{\alpha_t} \end{cases}$$, if misclassified, weight goes up

if classified correctly, weight goes down

should be Normalised

→ Now for the next step, sample the data such that it contains misclassified samples [for the next classifier to test on] (weighted) decides

This is a sequential process [not a parallel one]

* So, h_2 is a complement of h_1

h_{100} is a complement of $h_1, h_2, \ldots h_{99}$.

⇒ D_{100} has only those samples on which the other classifiers have failed on. [h_{100} doesn't work that well on the whole dataset as it has the complex samples]

[an expert dataset].

⟹ So instead of directly averaging, they have to be weighted.

- Combine the classifiers based on weights ⇒

$$\alpha_t = \frac{1}{2}\log\left[\frac{1-\varepsilon_t}{\varepsilon_t}\right], \quad \varepsilon_t = \text{error on the whole set given by the classifier.}$$

(weight)

Output ⇒ $H = \sum \alpha_t h_t(x)$

called as ADA Boost ≡ Adaptive boosting.

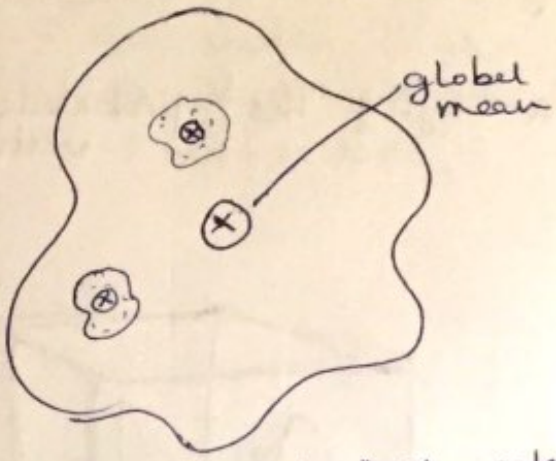
* Variance of estimate –

⇒ as we take small subsets, mean tends to jump around.

variance of the subset estimate → variance of the whole set

↓

underestimate.



⇒ [Ensemble methods work well even in the case of ~~bad~~ week classifiers.

↑

decision trees are not good for ensemble methods.

→ weaken the decision tree [as decision tree is a strong classifier]

↓ to a week classifier.

⇒ ignoring few features [access restricted] → * select a subset of features.

[For Ex [pick up 1000 out of 5000]

↳ causes significant reduction of feature space

⇒ Stop the tree growth → stumps

– short decision trees.

"Limiting the height".

⇒ Bagging (selecting a subset of samples → build a decision tree on it).

↓

doing such simplifications – underfit

→ generate such weak decision trees = $T_1, T_2 \cdots T_{200}$.

↓

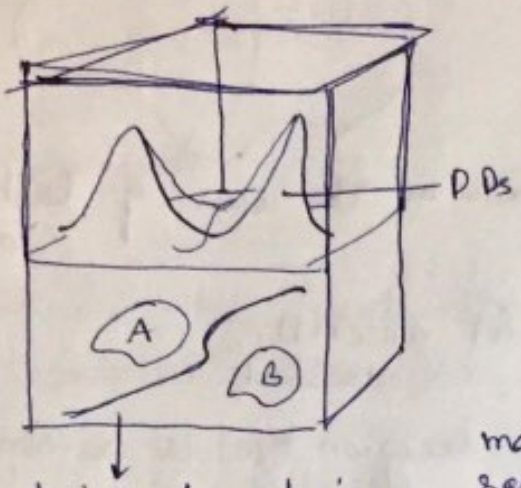
and average the output [mixture of classes]

↓

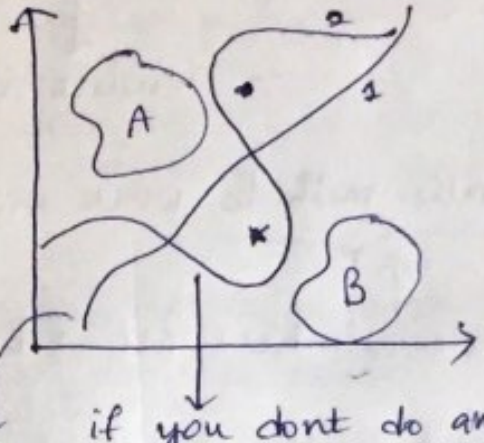
⇒ the output generated by the trees is not a perfect class [as we cut it off before way], well have a PD of classes.

Height is limited → so ensemble of trees is efficient.

⇒ averaging the probabilities for all the trees [ensemble]
vector



decision boundaries are smooth with more # samples



many samplesets miss out those outliers

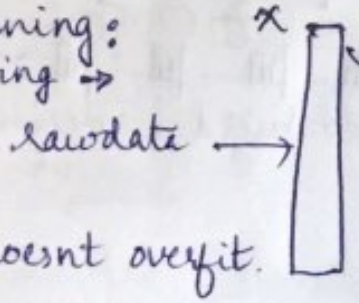
if you dont do any subset sampling, and because decision trees are extremely strong, your decision boundary tourns out this way.

Random features are selected along with random sampling

↓

Random Trees ⇒ Random Forests

Deep learning: is growing →



rawdata →

instead of handcoded features like [random trees, SVMs]

doesnt overfit.

→ Random Forests → easy to train as each tree is very small

↓

might be faster than decision trees.

- extremely parallelisable
- efficient to train/test
- doesnt overfit
- can model any complex distribution

Kinect → runs a random forest classifier. 'Microsoft

[using very simple features — like whether it is a shoulder or not? (based on depth)]

computes depth & rgb } ≡ rgbd

classification = extremely fast done on a weak processor.

- Random forests → can be used for regression also.

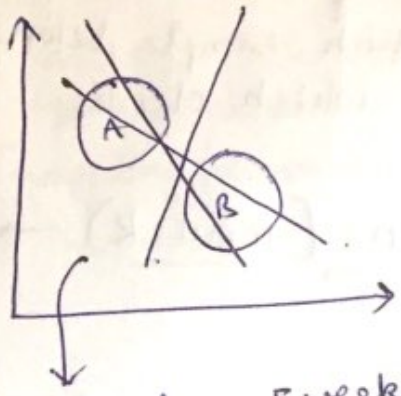
↓

most important features "feature selection".

* gene data + cancer patient data

⇓ Random forest

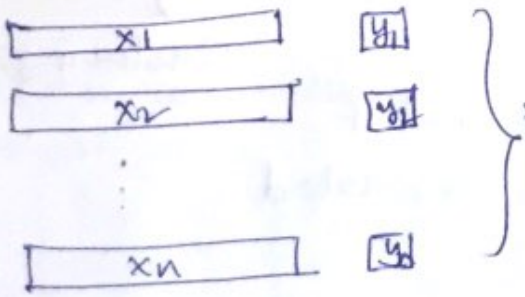
give you the genes that are important in classification ≡ { can also give non-linear boundaries ← random lines [weak classifiers] ↓ decision trees.



Unsupervised Learning ⇒

* supervised learning ≡ $y \Leftarrow f(x)$ ↓ mapping is learnt

{x, y} ↑ pairs of data is given to you.



⇒ f is modelled

Unsupervised learning

data is given → dont know what to learn?

find patterns in the data

* clustering - group the data with similar patterns into multiple clusters.
  - "ill-defined problem" ≡ Find clusters in the data

* multiple unknowns -
  - no. of clusters
  - which sample belongs to which cluster.

Data given

grouping into no. of groups ≡ unknown

can be 2 or 3

K-Means (input: K) →

if K=2, clusters = 2

on defining K=2,

cluster 1 cluster 2

find clusters in the data

converted to

There are 2 clusters, find out those

* which sample belong to which cluster?

each sample could be put into cluster-1 or cluster-2 = No. of ways to split into 2 sets

$= 2^{n-1}$ clusterings

* Naive approach ≡>
  - Compare any two clusterings, but if n becomes large, it becomes complicated.

* Assume, you know the means of the clusters
(close samples belong to the cluster) -A
to the mean-A

these are hard problems. ← Mean ⇄ clusters — just 2 comparisons for every sample [one with mean-A and other with mean-B]

given clusters, means can be computed
a given means, clusters can be computed.

↓ with both as unknowns (chicken and egg problem).

* Trick: Randomly pick means. and assign each sample to the cluster (computing distance)

- once you have clusters, compute the means of the clusters.

↓

Recompute the clustering

↓ ↑

Recompute the means.

→ Termination:- No change in assignment, stop.

↓

Algorithm: KMeans (very powerful).

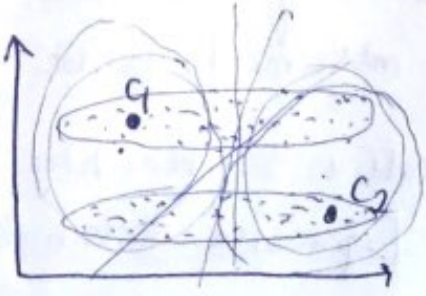
deep neural learning ⇝ improved unsupervised learning (unlabelled data).

* Data clustering by Jain & Dubes (classic algorithms)

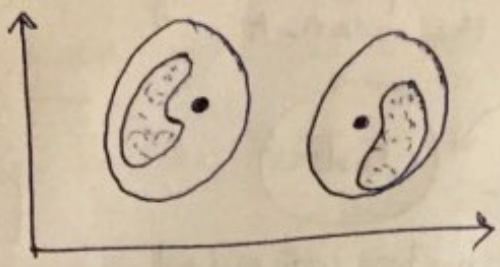
KMeans - computes the distance to the centers (euclidean).

↓

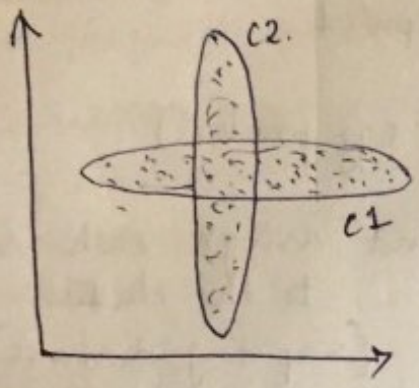
shapes are circular, spherical in shape clusters.



boundary between the clusters → far apart



(a)



↳ To solve –

→ group them into more clusters and can later combine the clusters

- Mahalanobis distances [→ distances in different directions are different] instead of Euclidean distance.

- Euclidean distance –

$$\sqrt{\sum_i (x_i - y_i)^2}$$

- Mahalanobis distance – depends on the distribution. [covariance matrix]

↓

distances are "weighted". based on the distribution of points.

$$D = (x-\mu)^T \Sigma^{-1} (x-\mu)$$

↑ covariance matrix [of the cluster]

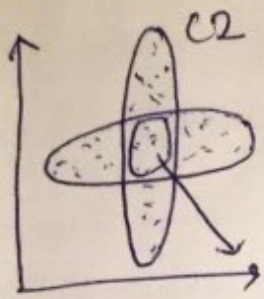
P.C. Mahalanobis – Indian Mathematician

⇓ Mahalanobis dist. instead of Euclidean dist.

[• clusters are no more assumed to be spherical]

if circular – covariance matrix ≡ identity matrix. ($\Sigma = I$)

• clusters → Elliptical [gaussian distribution]



what about these points? where do they belong.?

(1) → cluster shapes are made not spherical [elliptical]

↓ modelled as gaussians.

$C [\mu_c, \Sigma_c]$

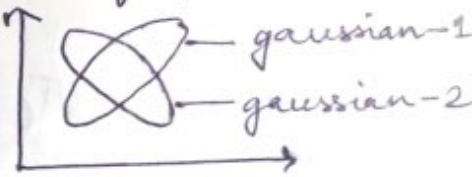
(2)

→ Soft Assignments = [every point is assigned to each cluster with some probability (different likelihoods)]

cluster = $N(x; \mu_c, \Sigma_c)$

Combination of

→ Multiple gaussians model the distribution of points → becomes ≡ density estimate.



→ size of gaussians ≡ π_c

↓

* samples assigned to cluster C

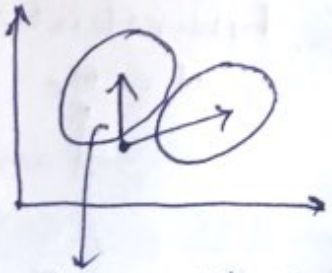
model ≡ sum of weighted gaussians.

[based on size]

Given a point x_i, cluster assignment ⇒

E-step ≡ Expectation (of assignments)

Compute the likelihood of point x_i being generated by the gaussian ≡ $\pi_c \cdot N(x_i, \mu_c, \Sigma_c)$



Not euclidean distance, instead likelihood estimate

TO sum it up to one ⇒ sum it over all the clusters

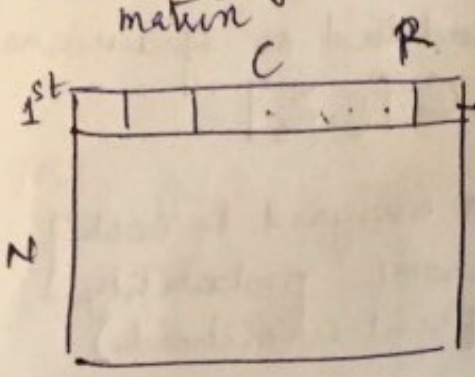
$$\gamma_{i,c} = \frac{\pi_c \cdot N(x_i, \mu_c, \Sigma_c)}{\sum_{c_j} \pi_{c_j} \cdot N(x_i, \mu_{c_j}, \Sigma_{c_j})}$$

* responsibility of i^{th} sample for cluster c

r_{ic} → computed for N × C

N → no. of samples; C → no. of clusters.

↓ represented by a matrix



rows sum up to one

row contains for each sample-i, soft assignment to all the clusters.

For c^{th} cluster, all the responsibilities together ⇒

$$m_c = \sum_i r_{ic}$$

$$\pi_c = \frac{m_c}{m} \quad \text{(updating } \pi_c\text{)}$$

you are the cluster, m_c = all the responsibilies of data points towards you by sum of responsibilities to all the clusters.

updating, μ_c, Σ_c ⇒ **M-step**

$$\mu_c = \frac{\sum_i r_{ic} \cdot x^i}{m_c}$$

Maximum likelihood estimate of μ

[∵ Equivalent of $N = m_c$ ↑ soft assignment]

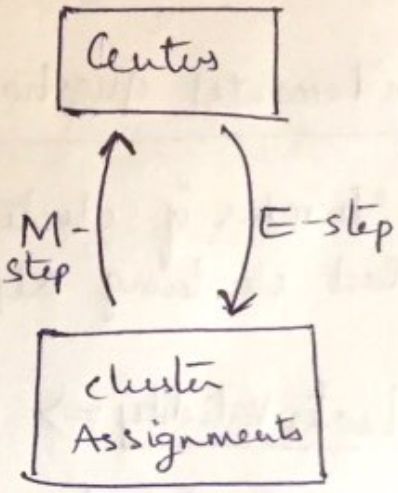
$$\Sigma_c = \frac{1}{m_c} \sum_i (x_i - \hat{\mu}_i)^T (x_i - \hat{\mu}) r_{ic} \qquad \frac{\sum x_i}{N} \equiv \frac{\sum r_{ic} x_i}{m_c}$$

MLE of Σ

K-means algorithms

↳ 2 relerations ≡
- non-spherical clusters
- soft assignments (instead of hard assignments)

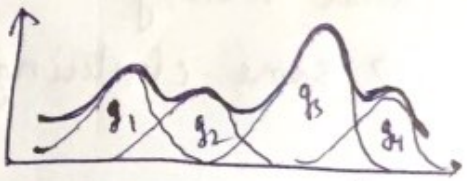
→ Computing r_{ic} [cluster assignments]
→ m_c
→ updating π_c
→ Updating μ_c, Σ_c [cluster centers]



Complex distributions ≡ can be effectively modelled by a combination of multiple gaussian distributions.

↓

Gaussian Mixture Model (GMM)



E-M algorithm: (Z) (unobserved variables).

↳ they are called "Latent Variables"

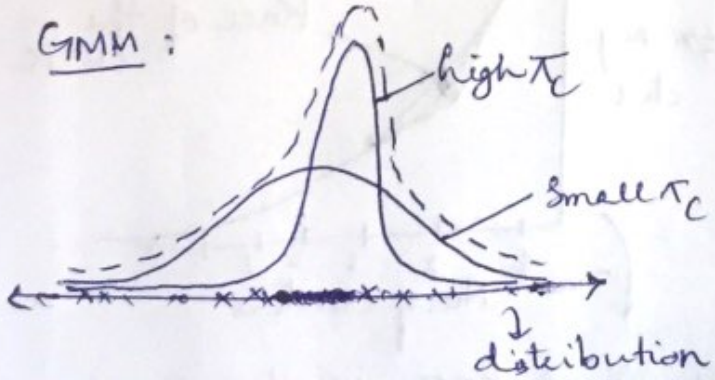
↓ for generic chicken & egg problems

↳ guaranteed to converge.

- gaussians can grow larger/smaller by the varying π_c

↓ not the spread of points but the no. of points.

GMM:



Gaussian Mixture Model ≡ Density Estimate [Generative Model]

↳ can also generate samples to decide which cluster it belongs to.

- K-means – doesn't get stuck in local minima, but that's not the case with GMM Estimation

GMM Estimation ≡ Latent Variable in here is "r_{ic}"

K-Means ≡ Latent Variables ≡ centers, assignments.

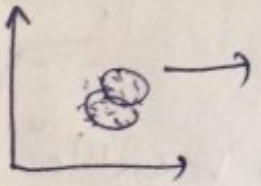
Fundamental Questions in Clustering :-

* Number of clusters ?
* Best clustering algorithm.

Cluster Validity ⇒ is the clustering valid or not ?

(1.) Cluster Stability.

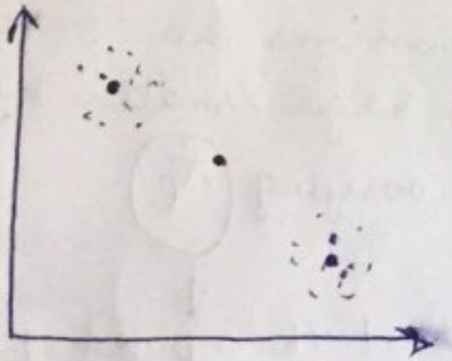
→ same clustering result for multiple initialisations



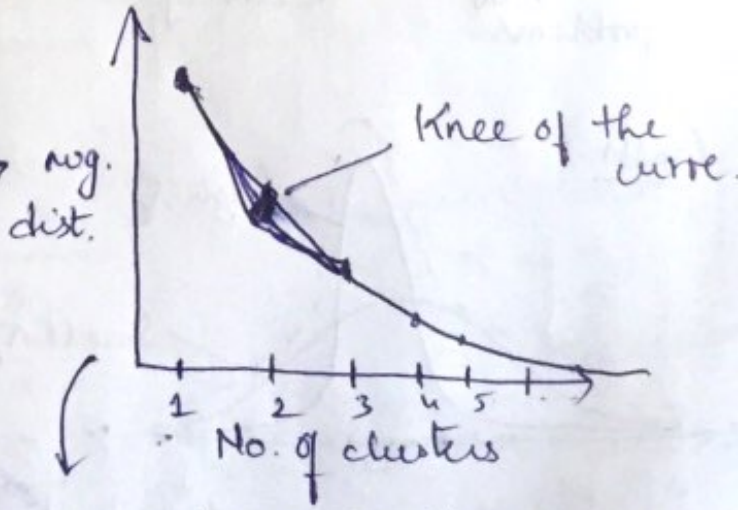
→ wont be stable (Not real clusters)

* clusters centers will be moving for every iteration.

(2.) Average distance (MSD) to Cluster Center.



⟹



You see a knee in the curve if the data is well seperated.

SMAI – Continued ⇒

Hierarchical Clustering [Slides]

- K-Means ≡ simple clustering algorithm.
  - → Randomly select k samples as cluster centers, C_k
  - → Repeat until convergence ≡
    - → Assign each sample to nearest cluster center (C_k)

$$k \text{ of } x_i^k = \text{argmin} \|x_i - C_k\|$$

    - → Update cluster center as mean of samples (x_i^k) assigned to that cluster.

$$C_k = \frac{1}{n_k} \sum_i x_i^k$$

Convergence?

→ Distance of sample to the cluster, over all the samples

$$G_k = \sum_i (x_i - C_k)^2$$

Goodness, $G = \sum_k G_k$ (overall clusters)

if goodness ↓, it is more compact.

> Reassignment → G decreases monotonically. }

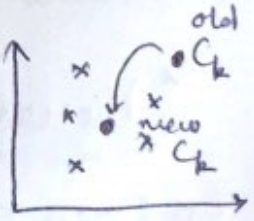
x_i, if it has to change clusters, then distance from x_i to C_p has to be more than the new reassigned cluster distance, x_i to C_k distance

⇒ Hence, it is decreasing.

> Updating the cluster centers –

↓

on computing average, distances go down from that point.



- At every step of K-Means, G goes down, but it has to either have,
  - Bound on the step size.
  - No. of G values is finite [permutations = all possible grouping subsets]

Possible G values ← $\{2^K\}$

Data $\xrightarrow{PCA}$ Transformed Data $\xrightarrow{KMeans}$?

↓

⇒ you got proper clustering

↙

if there is no loss

[sufficient principal components]

KMeans ↓

highly sensitive to outliers because of squared distance.

↓

define a distance threshold.

Value of K (No. of clusters) ?

→ generally prefer lower K

→ balance between, K vs G_K

⇒ G_K goes down, when no. of clusters is increased.

* Instead of square distances in G_K, we can use some distances [easier to compute].

> In GMM, you end up with elliptical clusters

K-Means " " spherical clusters.

↓

comes to a hard stop, after some time [convergence]

* clusters → sub-clusters.

↓

so, clustering algorithm ⇒ output a hierarchy of entities

⇓

* Hierarchical Clustering.

① → Top Down (Divisive) Clustering = Apply K-means repeatedly on each cluster.

② Bottom Up (Agglomerative) clustering. [mae efficient]

bottom up grouping ⇒ start with each sample in a diff cluster ——— and then join the clusters till you get a single cluster

* Dendrogram — Heirarchy of clusters represented as a tree.

↳ as we increase the distance threshold, we keep grouping into clusters. [inter-cluster distance]

↳ Similar to Kruskal's for minimum spanning tree

~ Time Complexity ≡ $O[n^2 \log n] \equiv O[E \log E]$ E-edges

$O[m^2 \log m]$ m-points.

$E = [m^2]$

Inter-cluster distance ≡ min [distance between the sample points in the clusters]

Hierarchical Algorithms → don't know where to cut? for appropriate clusters

{just like the value of K in K-Means}

lengths of lines in the tree ⇒ represents "lifeline" of cluster [birth to death]

* Valid cluster — long lifeline.

⇒ Inter-Cluster distance ≡

* Single link clustering algorithm / MST clustering ≡

$$D(C_p, C_q) = \min_{x_i \in C_p, x_j \in C_q} (d(x_i, x_j))$$

C_p, C_q → are the clusters.

↳ as you cross the smallest distance, a link is created.

- Complete Link — max of distances
- Average Link — average of distances.
- Centroid based.

» Single Link Clustering Algorithm:

→ prefers long chain clusters.

» Complete Link ≡ $O[N^3]$ ~ Time complexity.

→ spherical clusters.

* Centroid based — compute the centroid of cluster

n_p n_q

C_p C_q } ≡ centroid ≡ $\frac{n_p C_p + n_q C_q}{n_p + n_q}$

↓

implemented easily and efficiently.

⟹ How good is a clustering algorithm?

Comparing clustering algorithms ≡

↳ feed "standard data / labelled data / ground truth".

Let,

- Standard data ≡ C gold standard classes
- Clustering algorithms ≡ K clusters, $\{w_1, w_2, \ldots w_K\}$ with n_i members

→ measure the Accuracy?

⇒ Purity metric [cluster-level] → "Biased"

↓

$$\text{Purity}[w_i] = \frac{\max_j (n_{ij})}{n_i} \quad j \in C$$

if majority in the cluster are apples, the how many impurity ← oranges are in it?

→ Rand Index [overall] — (2×2 matrix)

→ If 2 samples are together in ground truth, — are they together in the clusters or not?

similarly, verify for the inverse also. on clustering.

~~depends~~ defined on pairs, not single data points

ground truth \ clustering	Same cluster	Different cluster
Same Class [ground truth]	(A)	B
Different Class [ground truth]	C	(D)

Rand Index $\equiv \frac{A+D}{A+B+C+D}$ ~ related to precision metric of retrieval problems

• Semi Supervised Learning — partially labelled data.

↳ partial labels. (not for every pair).

(such hard constraints — are usually softened).

[* adding a penalty term to the optimisation problem]

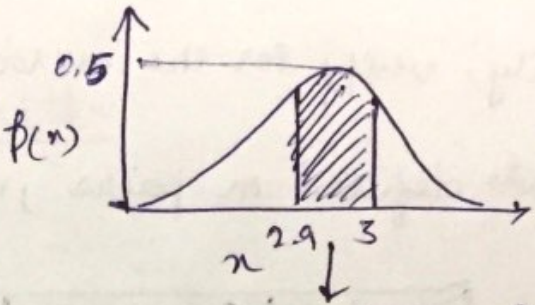
→ Many variations of partial labelling can be present.

Probabilistic Graphical Models — Daphne Koller. [Stanford]
(PGM).

Probability density function

↳ area is the probability, not the height

Bayes Rule



$$\Rightarrow P(A/\bar{x}) = \frac{P(\bar{x}/A) \cdot P(A)}{P(\bar{x})}$$

$P(\bar{x})$ → can be ignored

→ area here ≡ Probability that x lies between 2.9 & 3.

- if this is known, (density is unknown)
  - given a class, likelihood of $\bar{x}$ vector..
  - then bayes becomes accurate (the best).

d – dimensional vector ⟶ $O[Kd^2]$ complexity

- d → $d\mu$'s, $d^2\Sigma$ parameters
  - for K

GMM ≡ model. → parameters have to be estimated.

Can be solved using —

Naive Bayes [where independent features]

~ correlation = zero

covariance matrix ≡ diagonal matrix

⇒ Parameters change to ≡ $O[Kd]$ from $O[Kd^2]$

$$P(\bar{x}/A) = P(x_1/A) \times P(x_2/A) \ldots P(x_d/A)$$

multiple features ~ need not be same distribution [not all gaussian]

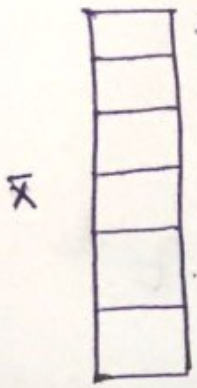
one can be normal density, other can be exponential density.

Probabilistic Graphical Models ≡

* completely independent features — not always true.

Model what is dependent and what is not dependent..

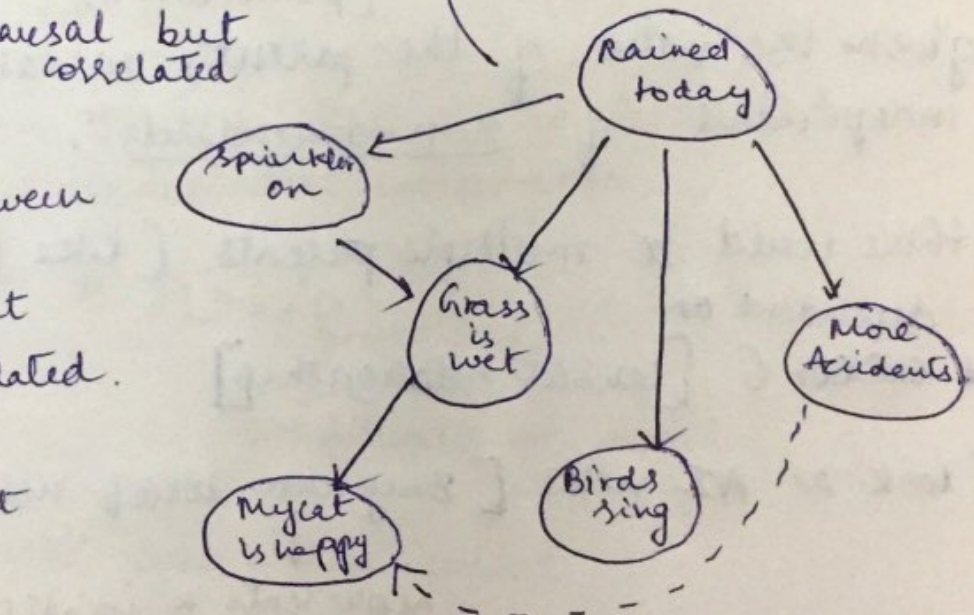
→ Let, 6 vectors/features — then we have a 6-dimensional feature vector



→ 0, 1, No. of possible values ≡ 2^6 [without any probabilistic relationship information]

$\vec{x}$ → Compute $p(x)$, density function

Lines indicate causal relationships.



not causal but correlated

→ There's no direct relation / line between sprinkler on and more accidents, but they might be correlated.

causality is different from correlation.

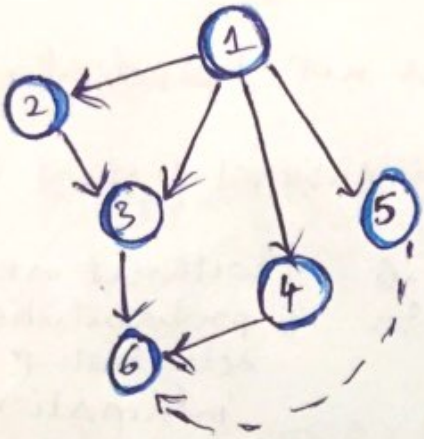
defined over huge no. of observations [statistically]

Causality vs Correlation.

∴ only correlated, not causal

When independent, can use naive bayes but
When dependent, use bayesian belief networks.

Conditional Independence ≡



- 6, 5 – not completely independent
- 6,5 – donot have a causal relationship.
  - → knowing the value of 6 can affect the value of 5.

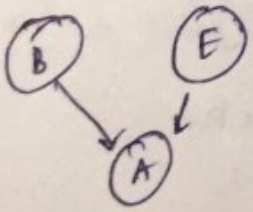
⇒ 2,5 – conditionally independent, given the value of its parent (1)

(predecessors).

- given the value of the parent, variable is conditionally independent of "non-descendants".
  - there could be multiple parents (like (1,2) for 3).
- 4 dependent on 6 [causal relationship]

* look at AI notes [Bayesian belief networks]
  - represents a joint probability distribution of variables.

$$P(x_1, x_2 \ldots x_n) = \prod_i P(x_i / \text{parent}(u_i))$$

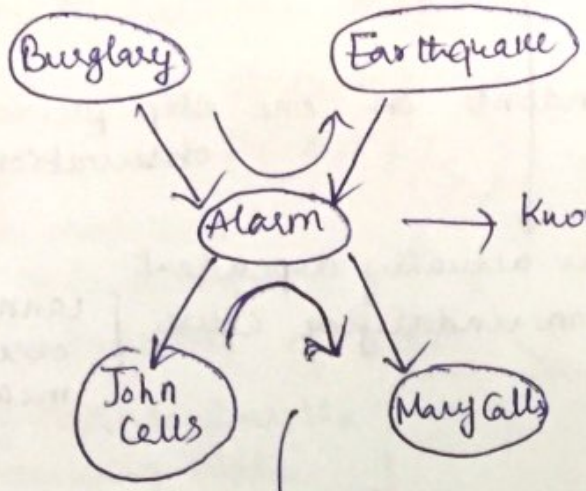


$$P(A/B) = P(A/B,E)P(E) + P(A/B,\neg E)P(\neg E)$$

$P(B/A) \Rightarrow$ use bayes rule

$$\frac{P(A/B)\,P(B)}{P(A/B)\,P(B) + P(A/\neg B)\,P(\neg B)}$$

Inter-Causal ≡



→ Knowing the values of common descendents, there exists an inter-causal inference.

When alarm goes off, (not knowing this = makes them dependent)
John calling, → probability of Mary calling ↑

Hidden Markov Model →

$P(x_1, x_2, \dots x_d)$ – Joint density can be factorised into independent components.

↓
multiple features observed at time t $= \pi P(x_i / P_{parent}(x_i))$
↓
conditioned on parent.

x_t, $t = 0, 1, 2, \dots T$
↓
A single entity measured over a time period.

→ $P(x_0, x_1, \dots x_T)$
↳ there exists some dependency

↓
correlation between what happens today / and tomorrow

we make a markovian assumption.

→ dependency is only 1 step [simplest] ↷ on the previous one, not on anything in the feature.

$P(x_i)$ → depends on $P(x_{i-1})$ only.

$$P(x_0, x_1, \dots x_T) = P(x_0) \cdot P(x_1/x_0) \cdot P(x_2/x_1) \dots P(x_T/x_{T-1})$$

$$= P(x_0) \cdot \prod_{t=1}^{T} P(x_t / x_{t-1})$$

⇒ every observation is dependent on one step previous observation.

[first-order markovian process]

it is actually dependent on an underlying cause. [cannot be observed/measured]

Underlying Cause

Underlying state (Hidden): $s_0 \rightarrow s_1$ ... s_n

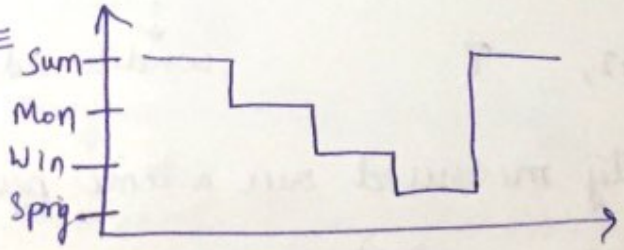
observations: o_0 o_1 ... o_t

[dependency is understood ~ should be learned]

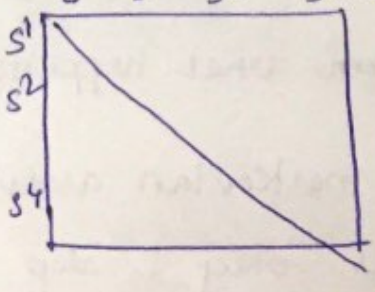
is only dependent on underlying cause → modelled by markov chains.

For Ex: for predicting the weather,

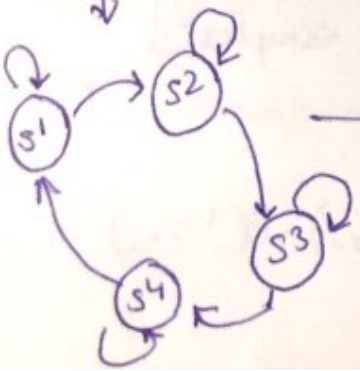
underlying states ≡ $S = (Sum, Mon, Win, Spring)$



transition matrix ≡ (probabilities)



↑ possible values of state (S) {Sum, Mon, Win, Spring}



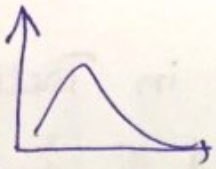
→ with probabilities

→ we should know —

* state transitions — states are infered through observations. $S_0 \to S_1 \cdots S_T$ [unobserved]
* likelihood of observing an observation given the underlying state.

$P(O = y / S)$ = observation probability.

can be modelled as a distribution function



[purely dependent on the underlying state]

$$P(O_0, O_1 \cdots O_T, S_0 S_1 \cdots S_T) = P(S_0) \cdot P(O_0/S_0) \cdots \prod_{t=1}^{T} P(S_t/S_{t-1}) P(O_t/S_t)$$

$O_0, O_1 \cdots O_T$ → available

sequences possible $\equiv 4^n$ [no. of states = 4]

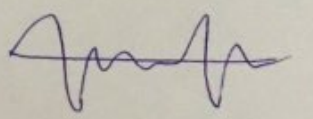
① given O, find most likely sequence of S_i.

given O, find the likelihood.

$$P(O) = \sum_{S} P(O_1, O_2 \cdots O_T, S_0 S_1 \cdots S_T)$$

summing over all possible sequences.

for ex: observation ≡ data

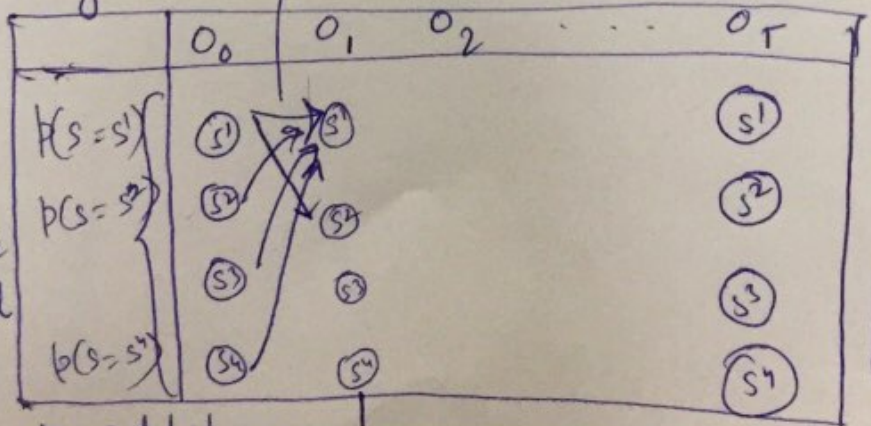


it can be → hidden or → kitten

out of all sequences, find the best.

can be solved using dynamic programming

$P(S_1/S)$ transition probability (have to be known) multiply and add it.



gives the likelihood

Trellis diagram/ decoding [which sequence do we pass thru?]

→ add it up $P(S)\,P(O/S)$ [n terms] → d^2 complexity.

Computations to be done $= T.d^2$

$d^n \rightarrow T.d^2$ [Dynamic Programming]

(Reduction in complexity)

Most likely path sequence in Trellis diagram

→ use "argmax" instead of Σ

↓ Viterbi decoding / algorithm

→ Learning problem - given O, learn transition probs.

* left to right - forward computation / algorithm

right to left - backward computation / algorithm

↓ Forward-backward algorithm