Machine learning :-

DSC7456

UnSupervised Learning

- Association Rules
- Clustering (Finding similar groups among data points)
  - Row similarity
  - ↳ Hierarchial Clustering
  - ↳ DBscan
  - ↳ K-means
- Principal Component Analysis (PCA)

Supervised Learning

Regression

- Linear Regress

Classification

- Logistic Reg
- Naive Bayes
- Decision Trees
- K-NN
- SVM
- Random Forest
- Time Series

- Gradient Decent Method → Optimization Algorithm
- Singular value Decomposition (Collaborative filtering) → Matrix Factorization method

Association Rules :-

→ Affinity Analysis :-

* How likely the person would buy 'y' object if he buy's 'x' object :

Note :- (1) Main thing is identifing the important association rules
(2) Find the goodness of the A. rules based on bussiness value.

Basic Definitions :-

* Support {🍎} = 4/8 = no of times bought / Total. Trans

* Confidence {🍎 → 📦} = Support {🍎, 📦} / Support {🍎}

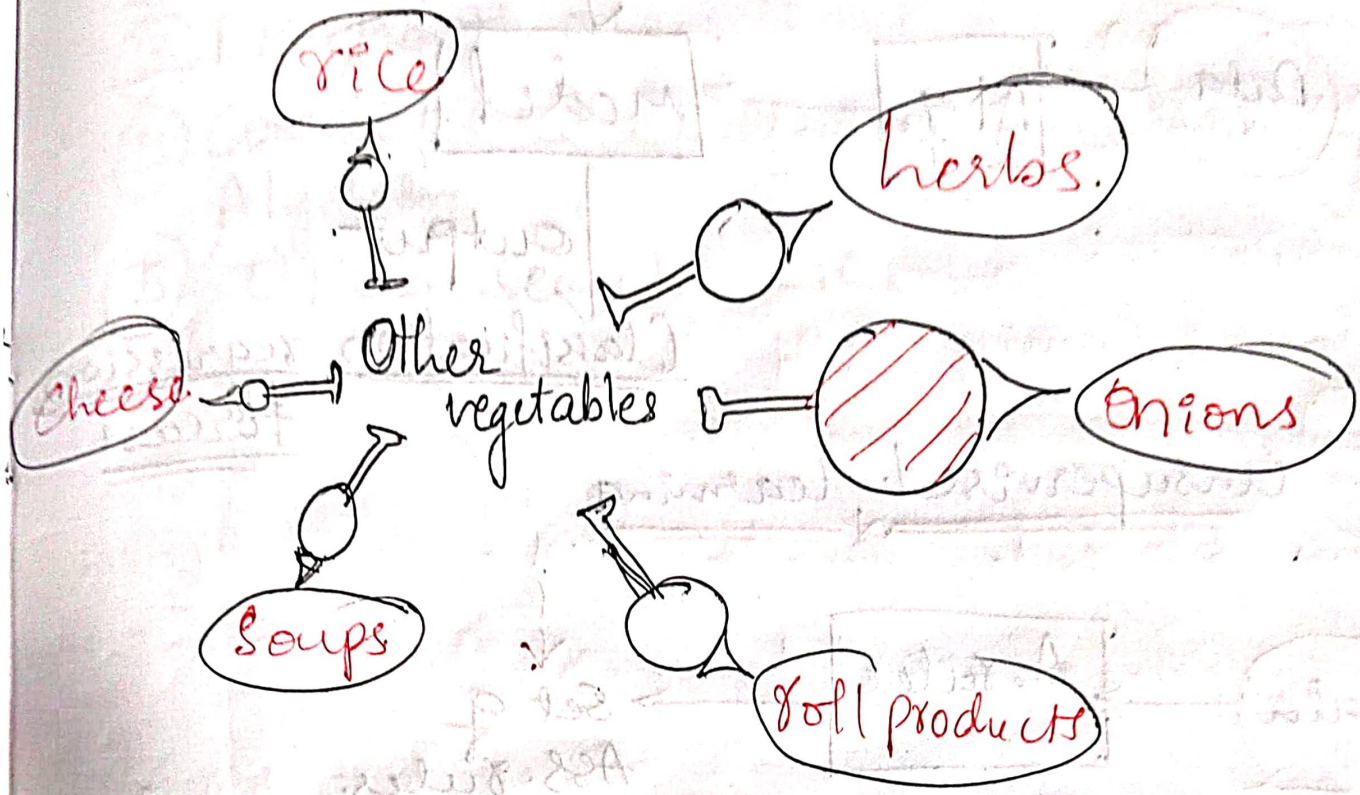
* Recall, Lift {🍎 → 📦} = Support {🍎, 📦} / Support {🍎} x Support {📦}

A measure of importance of the rule

* The above metrics are used to find the goodness of the A. Rule

Transaction 1	🍎 📦 🥫 🍐
T2	🍎 📦 🥫
T3	🍎 📦
T4	🍎 🍌
T5	🍾 📦 🥫
T6	🍾 📦 🥫
T7	🍾 📦
T8	🍾 🍌

Simple representation of all the 3 metrics of association rules.



(*) larger the circle → higher Support

(*) Red circles → higher lift

Point of Sale Data.

- (POS data is the data of customers along with prod they bought in any marts, shops, etc.)

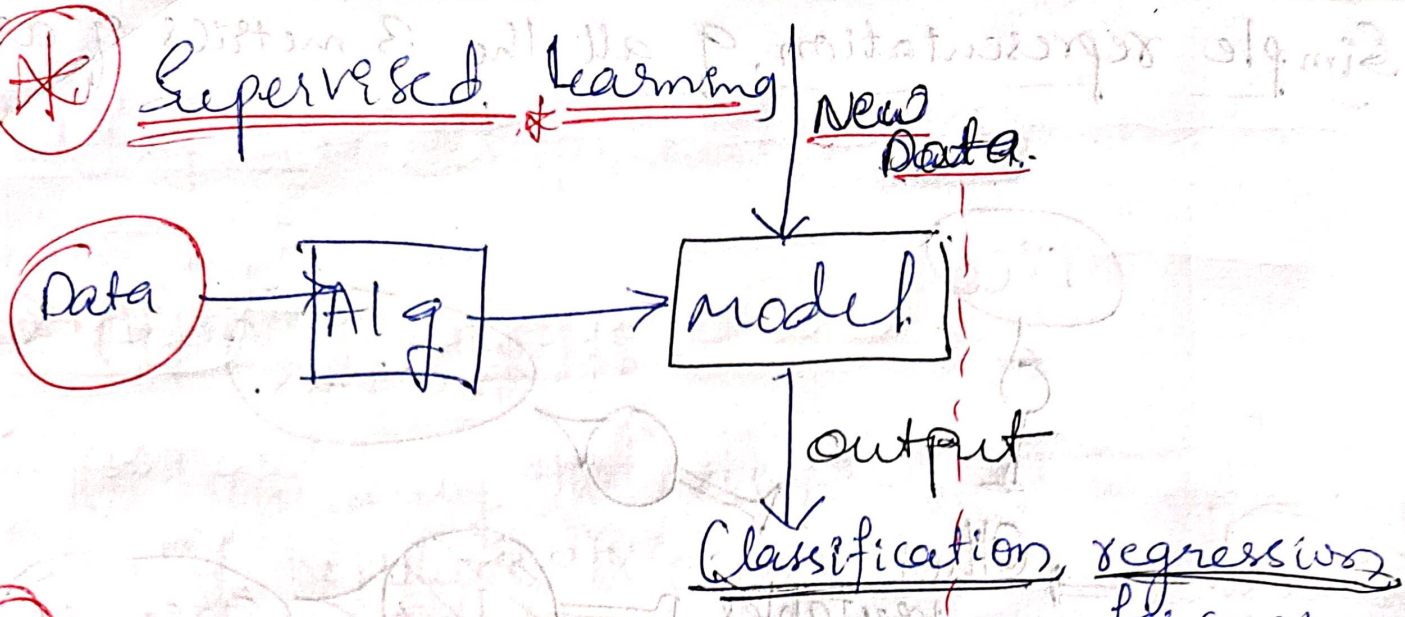
Steps :-

Hardest Part

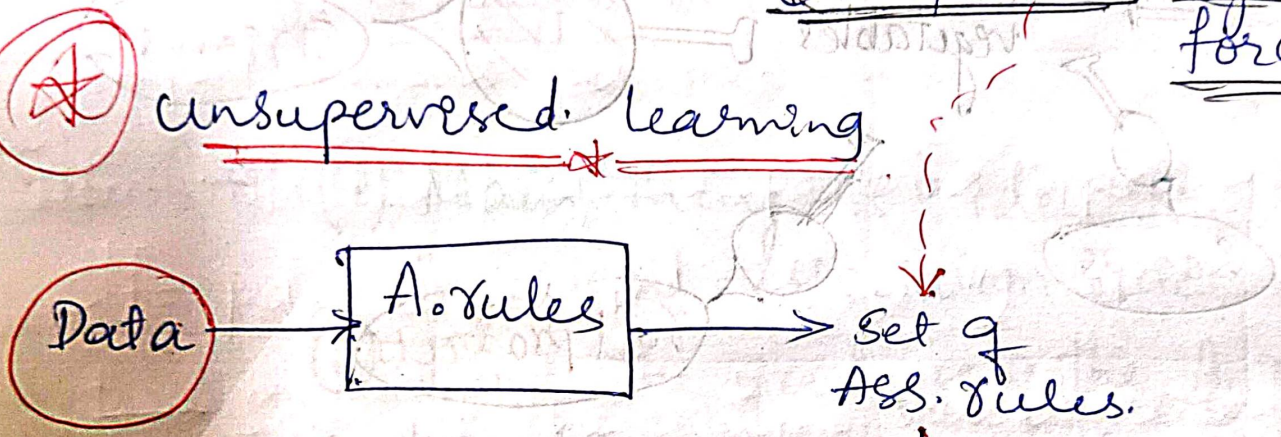
1. Item sets (sets of 2, 3, etc which are relevant)
2. Ass. rules for relevent Item set
3. Find Support, lift, confidence of all the relevant item set
4. Keep the best one's (with good Supp, lift, Conf)

(*) From these Ass. rules we cannot do classification but we can merchantise; bundling, pricing, merchantize insights, etc

* Supervised learning



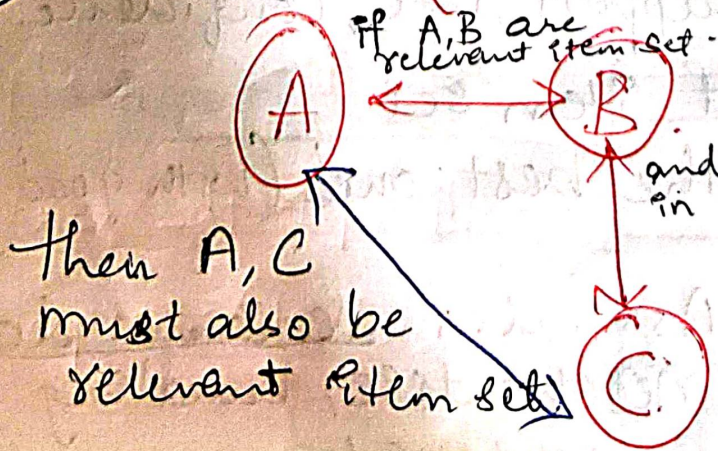
* unsupervised learning



---

* Since hardest part is finding the relevant item set :. there are some algorithms to get them.

(1) APRIORI. (Downward Closure)



Distances — Association Rules in Python.

mlxtend → Package is used.

```
import os
import numpy as np
import pandas as pd.
from mlxtend.frequent_patterns import apriori, association_rul
df = pd.read_excel("xxxx.xls")
```

↳ See all the unique values using .unique in all columns

↳ Choose in which level (ex → sub cat / category) the ass.rul must be built

```
xxxxx_df = pd.crosstab(df['orderID'], df['sub-cat'])
```

↳ Build crosstab(matrix) of orderID and all the attributes in chosen level.

↳ Replace > 1 terms := 1.

```
xxxxx_df[xxxxx_df > 1] = 1.
itemsets = apriori(transaction_df, min_support=0.01, use_colnames=True)
```

↳ use apriori to make relevant itemsets

```
rules = association_rules(itemsets, metric = 'lift',
                          min_threshold = 1.1)

sorted_rules = rules.sort_values(['support', 'confidence'
                                  ,'lift'],
                                 ascending = [True, False, False])
```

↳ Check for the highest confidence value itemsets and make decision from those rules.

```
sorted_rules.to_csv('rules_extracted.csv')
```

Distance :- (Euclidian, Manhattan Distance, Cosine similarity), Pearson Distance

(*) The distance b/w two data attributes is used to find the similarty among the data.

↳ distance is very helpful in clustering the datasets

↳ The distance b/w the data attributes is measured and weighted summation of all attributes is taken to know the overall Distance (Similarity) among the datasets.

(*) If d_1 is near to d_2, and d_2 near to d_3, then d_1 is not far from d_3.

(*) Practical data has different types of data attributes

1. Numeric → Euclidian → Assumption is attributes are ~~1~~
2. Interval-scaled → Manhattan
3. Symmetric binary
4. asymmetric binary
5. ratio-scaled.
6. ordinal and nominal.
7. Binary → asymmetric, symmetric

Euclidian

$$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

Manhattan.

$$|x_2-x_1|+|y_2-y_1|$$

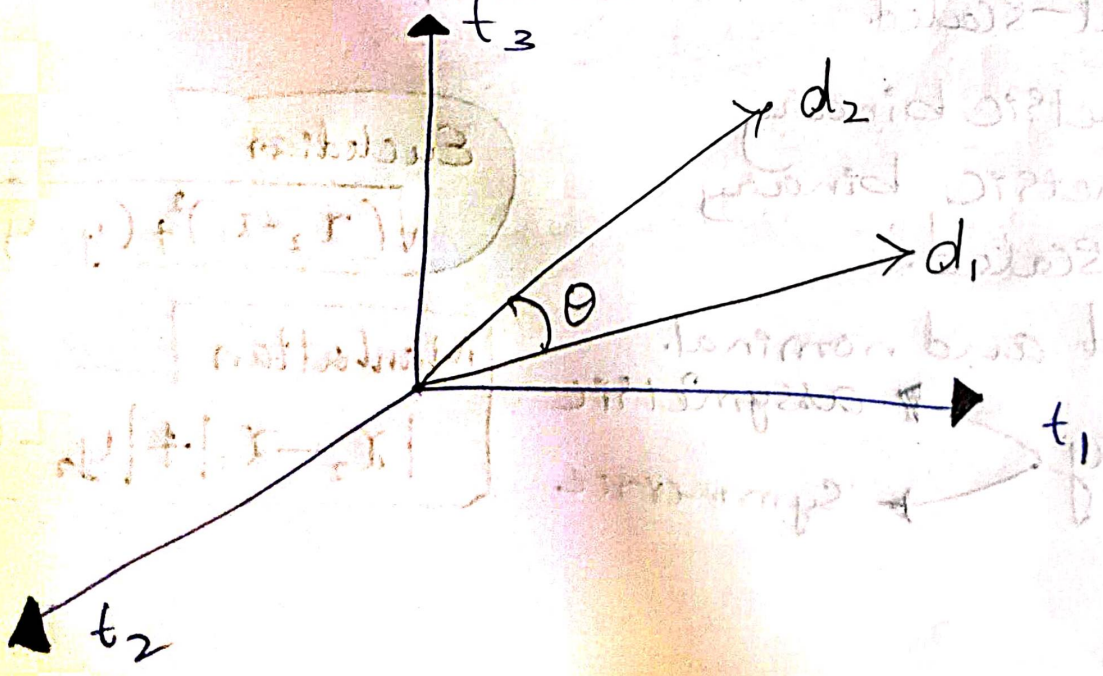
Some other common metrics:-

1. weighted distances
2. Minkowski distance.
   → $\sqrt[n]{(x_2-x_1)^n+(y_2-y_1)^n}$
3. The maximum distance amongst all attributes.
4. Co-relation b/w rows.

* Distance in higher dimensions become meaningless.

→ Cosine similarity

* Distance b/w vectors d_1 & d_2 captured by the cosine of the angle θ b/w them



$$sim(d_j, d_k) = \frac{\vec{d_j} \cdot \vec{d_k}}{|\vec{d_j}||\vec{d_k}|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \cdot \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

(*) Cosine of angle b/w two vectors

(*) The denominator involves the length of the vectors

(*) The cosine measures is also known as the normalised inner product.

(*) For Categorical Attributes in unsupervised settings: ||

Data i \ Data j	1	0
1	a	b
0	c	d

$$\text{Hamming Distance} = \frac{\text{no of dissimilar attributes}}{\text{no of dissimilars} + \text{no of similars}}$$

$$\text{Hamming Distance} = \frac{b+c}{b+c+a+d}$$

Binary → asymmetric $= \frac{b+c}{a+b+c}$

Binary → Symmetric $= \frac{b+c}{b+c+a+d}$

Pearson Linear Correlation :-

$$P(x,y) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

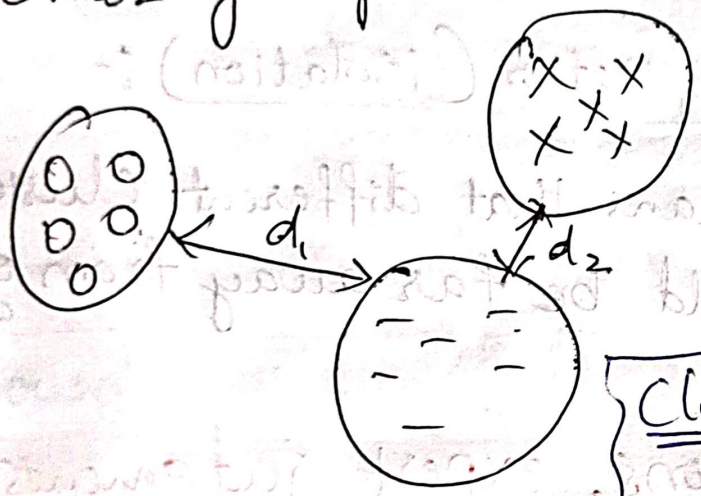
* Pearson linear co-relation (PLC) is a measure that is invariant to scaling and shifting of attribute vectors.
* The range is b/w -1 & $+1$
* The pearson.L.Corr is a similarity measure we can make it as disimilarity measure as

$$dp = \frac{1 - p(x,y)}{2}$$

* If we measure distance by any method, finally we must combine all distance to get a finalized distance by weighted Summ

Clustering

(*) In general a grouping of objects such that the objects in a group (cluster) are similar and different from (or unrelated to) the object in other group.



(*) This is the task (i.e Clustring) that must be done in some space

Cluster → Similar locally, dissimilar globally

Data input for clustering can be in two forms.

(1) Data matrix ⟶ $\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & & \cdots & & \cdots \\ x_{i1} & & x_{if} & & x_{ip} \\ \cdots & & \cdots & & \cdots \\ x_{n1} & & x_{nf} & & x_{np} \end{bmatrix}$

It consists of the values of the all unknown attributes (Columns)

(2) Dissimilarity (Distance matrix) matrix. ⟶ $\begin{bmatrix} 0 & & \\ d(2,1) & 0 & \\ d(3,1) & d(3,2) & 0 \\ \vdots & \vdots & \cdots\ 0 \end{bmatrix}$

$\frac{n(n-1)}{2}$ → input

Quality of Clusters :- (To find how good is the Clusters)

(1) Intra-cluster Cohesion (compactness) :-

(*) It measures how near the data points in a clusters are to the cluster centroid.

(2) Inter-cluster seperation (isolation) :-

(*) Separation means that different cluster centroids should be far away from one another.

(*) In most applications, expert judgments are still the key. (expert judgments involves the the type of the cluster, and its compact

1 Hierarchical Clustering :-

→ There are two main types of hierarchical clust

1 Agglomerative Clustering :-

Bottom-up → most Prefferd

↳ Start with the points as individual clusters

↳ At each step, merge the closest pair of clusters until only one cluster is left.

2 Divisive :-

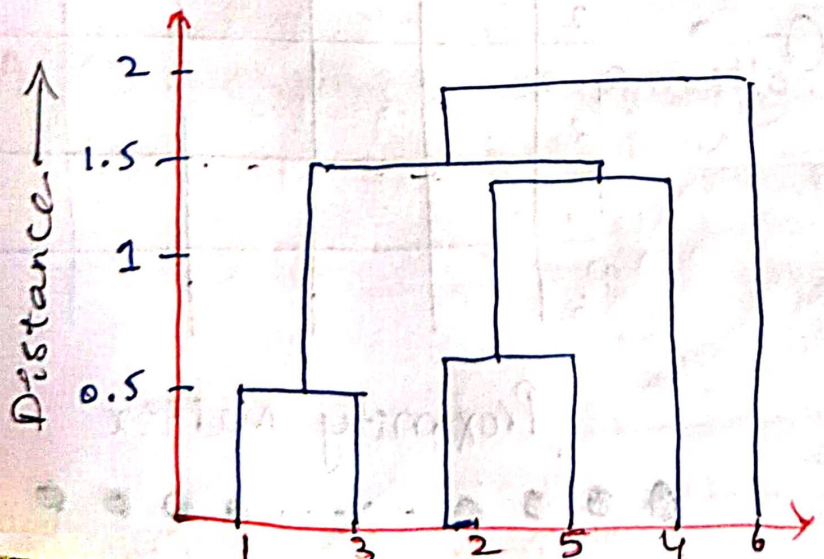
Top down →

↳ Start with one, all-inclusive cluster

↳ At each step, split a cluster until each cluster contains a point.

→ Produces a set of nested clusters organised as a hierarchical tree

→ Can be visualized as a dendrogram.



* The critical distance for Clustering of the data Points into a clusters must be decided according to the bussiness problem

Agglomerative clustering Algorithm:

1. Compute the proximity matrix (distance matrix)
2. Let each data point be a cluster
3. Repeat
   - Merge the two closest cluster
   - Update the proximity matrix.

   (Hardest step)
4. Until only a single cluster remains.

↳ Key operation is the computation of the proximity / distance of two clusters

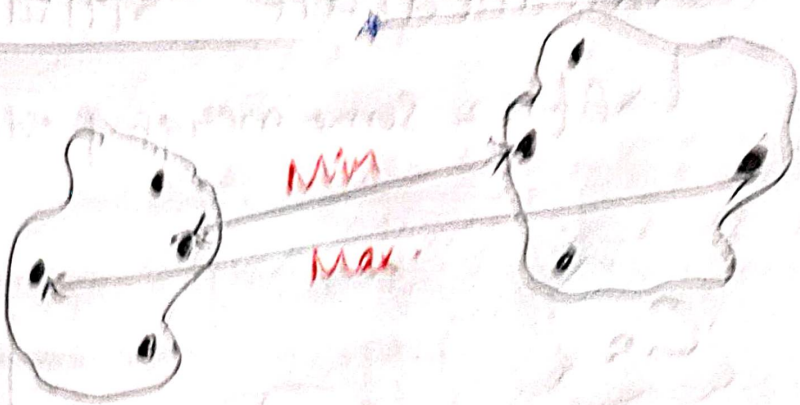
---

Example:-

→ Starting Situation

↳ Start with clusters of individual points and a

How to Define Inter-cluster Similarity



- → MIN
- → MAX
- → Group Average
- → Distance Between Centroids.
- → Other methods driven by an Objective fun
  - ↳ Ward's Method uses squared error

Min → Single Link, will merge two clusters when a single pair of elements is linked.

- ⊛ min works very well on non-elliptical shapes
- ⊛ min is sensitive to noise and outlayers.

Max → Complete linkage will merge two clusters when all pairs of elements have been linked.

- ⊛ Less susceptible to noise and outliers
- ⊛ Tends to break large clusters.
- ⊛ Baised towards globular clusters.

Group Average

- ⊛ Compramise b/w single and complete link.
- ⊛ Less susceptible to noise and outliers
- ⊛ Baised towards globular clusters

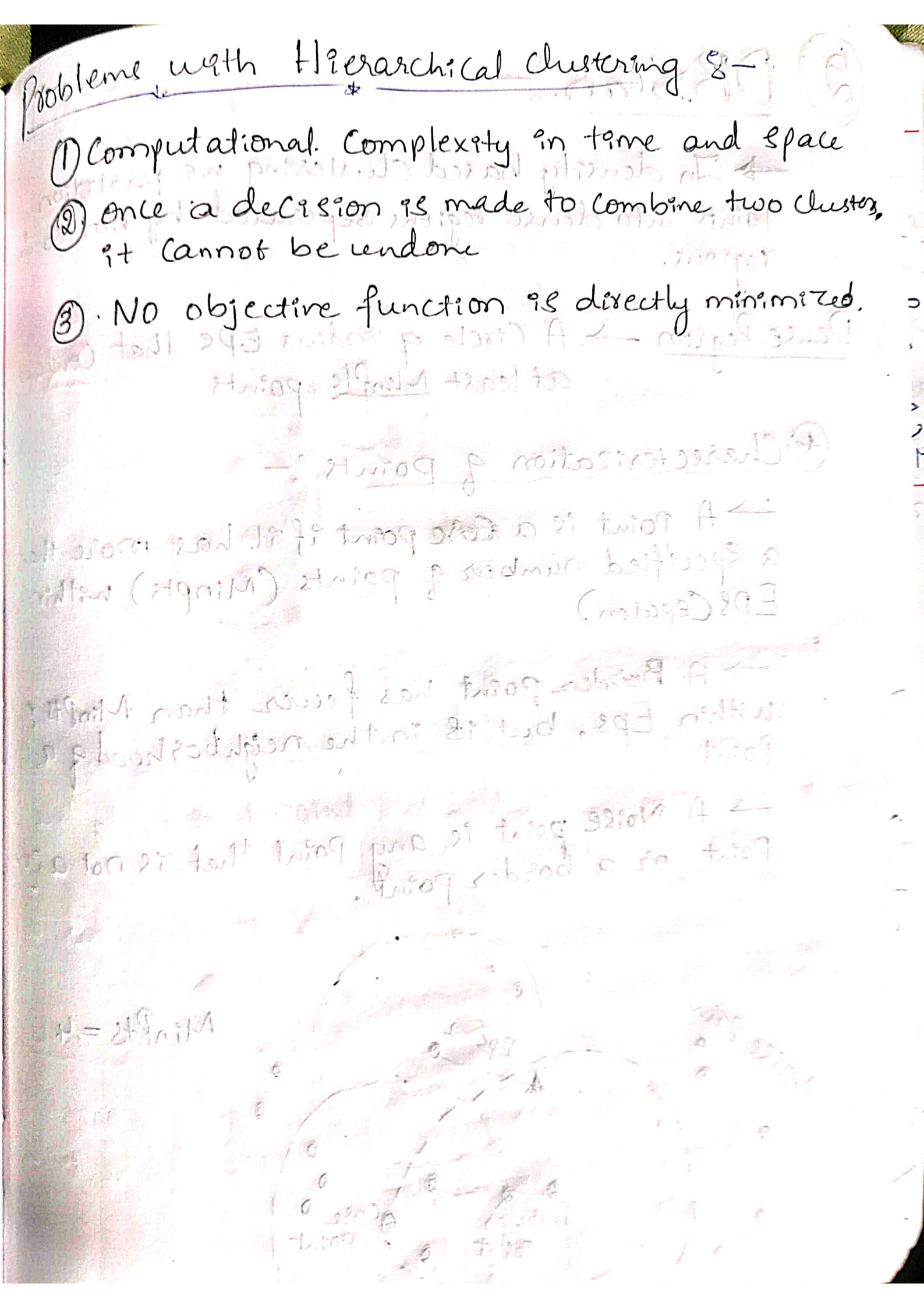
Wards Method → Similarity of two clusters is based on the increase in SSE, when two cluster are merged.

- ⊛ Less susceptible to noise and outliers
- ⊛ Biased towards globular clusters

K-means to initialize

Problems with Hierarchical clustering :-

1. Computational Complexity in time and space
2. once a decision is made to combine two clusters, it cannot be undone
3. No objective function is directly minimized.



② DBSCan :-

→ In density based clustering we partition points into dense regions seperated by not-so-dense regions.

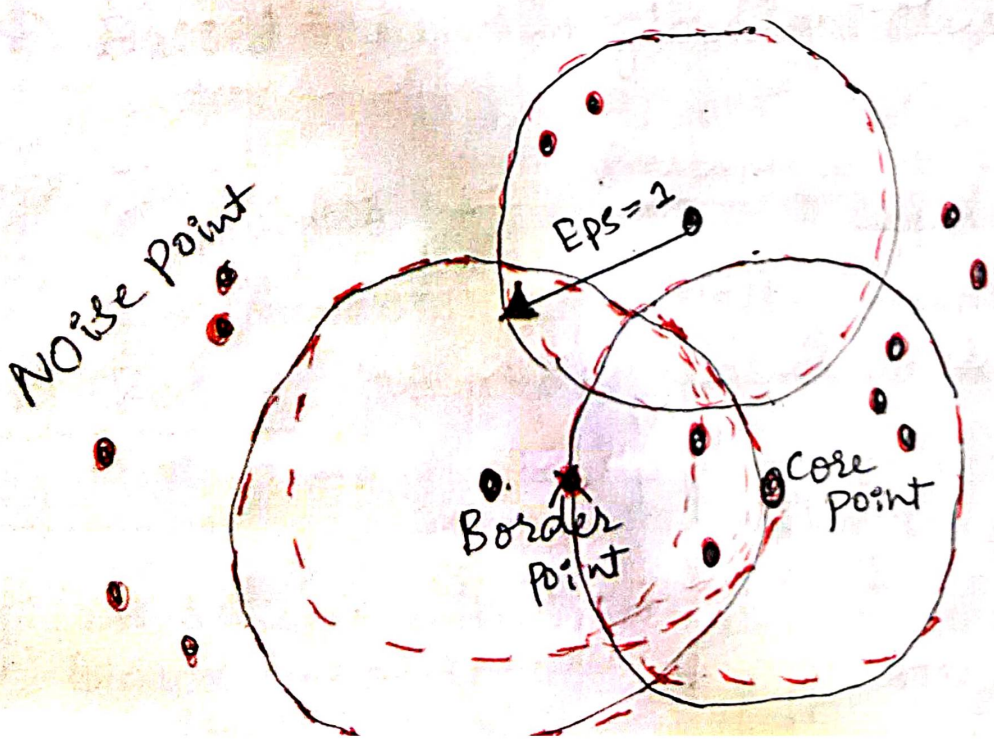
Dense Region → A Circle of radius Eps that Contains at least MinPts points

✱ Charecterization of points :-

→ A point is a **Core point** if it has more than a specified number of points (Minpts) within Eps (Epsilon)

→ A **Border point** has fewer than MinPts points within Eps, but is in the neighborhood of a Core point

→ A **Noise point** is any point that is not a Core Point or a border point.

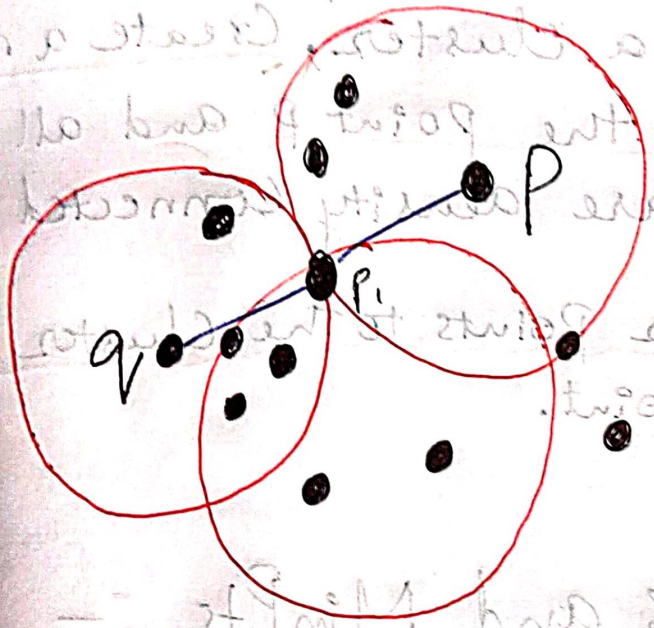


MinPts = 4.

Density-Connected Points :-

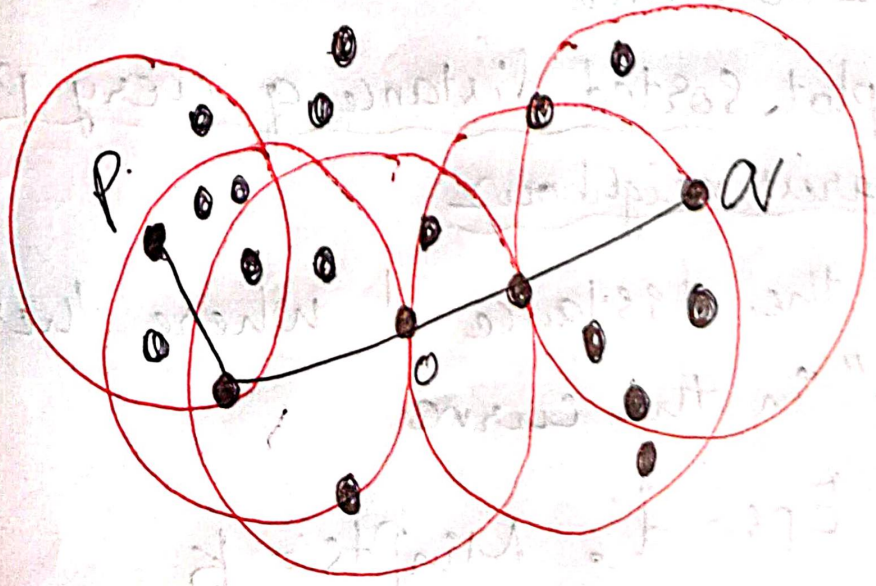
- **Density edge :-**

  ↳ we place an edge b/w two core points q & P if they are within distance Eps.



- **Density-Connected :-**

  ↳ A point p is density-connected to a Point q if there is a path of edges from P to q.

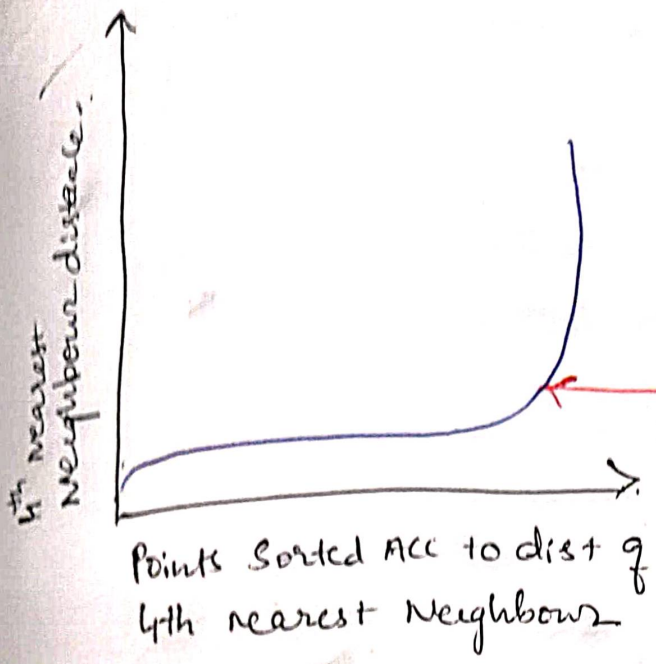


DBSCAN Algorithm :-

1. Label points as core, border and noise
2. Eliminate noise points
3. For every core point p that has not been assigned to a cluster, Create a new cluster with the point p and all the points that are density connected to p.
4. Assign border points to the cluster of the closest core point.

Determining the Eps and MinPts :-

- Idea is that for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance.
- Noise points have k^{th} nearest neighbour at farther distance
- So, plot sorted distance of every points to its k^{th} nearest neighbours
- Find the distance d where there is a "knee" in the curve
- Eps = d, MinPts = k.



Eps ~ 7-10
MinPts = 4.

Advantages :-

* Resistant to Noise
* Can handle clusters of diff shapes & sizes

Disadvantages :-

* doesnot work well for varying densities
* " " High-dimensional data

* Partitioning Algorithms → K-Means

↓

Takes the whole data & globally cuts into cluster

↳ K-means is a partitional clustering algorithm as it partitions the given data into K-cluster.

↳ Each cluster has a center, called centroid

↳ 'K' is specified by the user.

K-Means Algorithm :-

→ For a given (K), the K-means algo works as follows.

↓

Initial assuption using expert advise.

(1) Randomly choose K data points (seeds) to be the initial centroids, cluster centers.

(2) Assign each data point to the closest centroid.

(3) Re-compute the centroids using the current cluster membership

(4) If a convergence criterion is not met, or if some clusters dont get any points, go to (2).

Random chossing of Centroid and Calculate the nearest Points to each of them

Centroid shift

Compute the nearest points to the shifted Centroid

again Centroid Shift

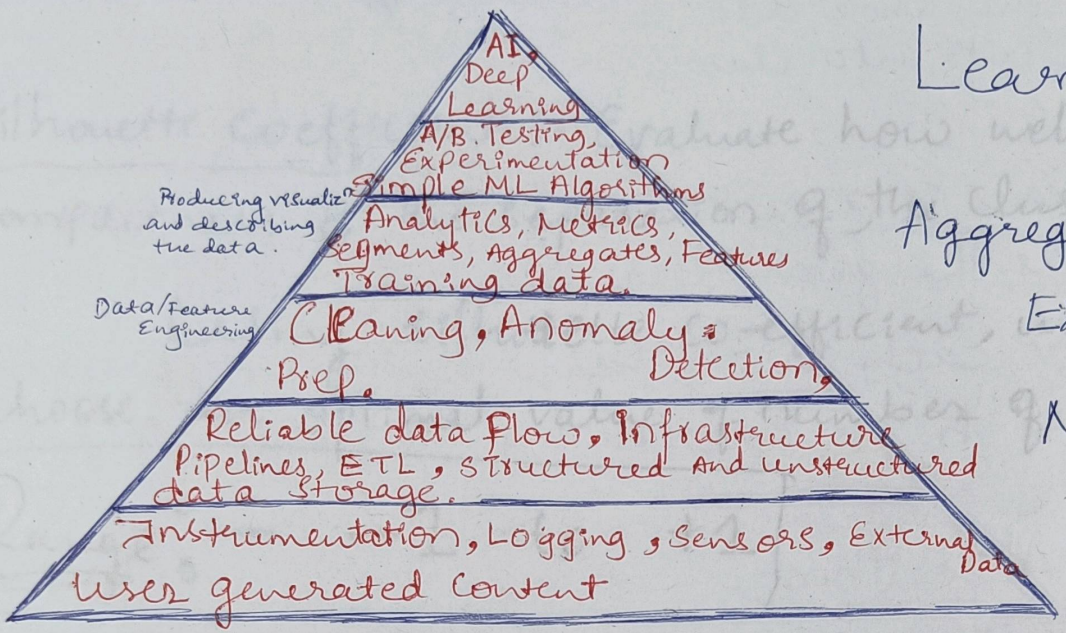
Silhoutte Score :-

Silhouette coefficient :- Evaluate how well the compactness & the separation of the clusters are.

Using silhouette co-efficient, we can Choose an optimal value of numbers of clusters.

Range :- -1 to $+1$

Supervised learning :-



Learn/optimise

Aggregate/Label

Explore/Transform

Move/Store

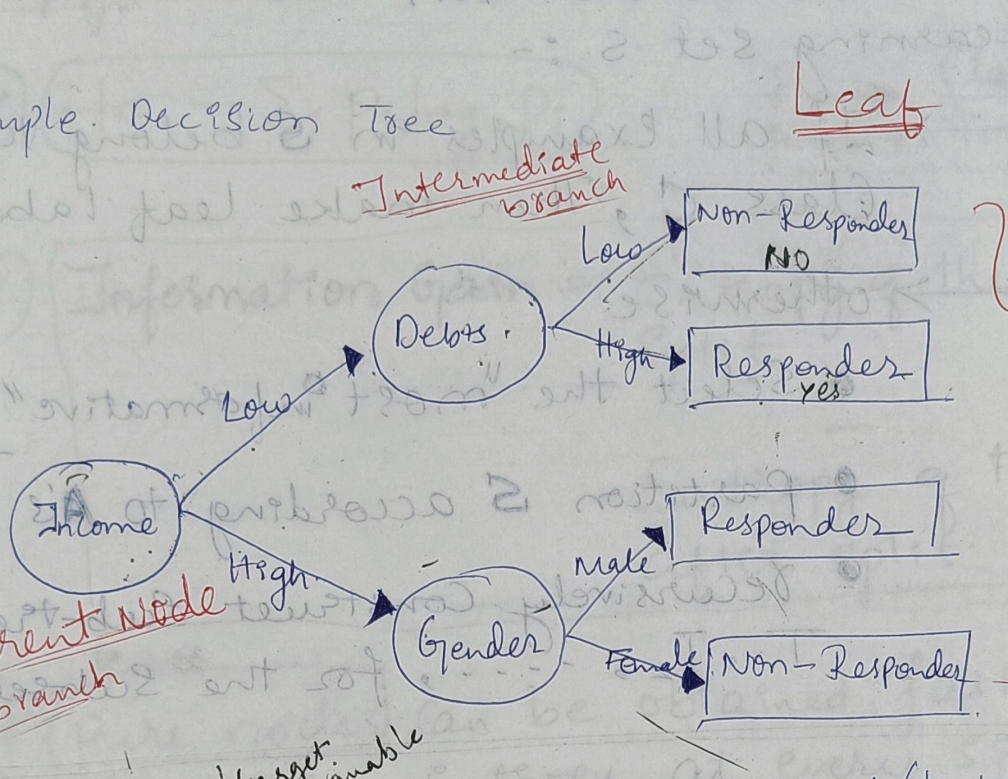
Collect

3 Decision Trees

Attribute/Feature → Independent Variable

lable → Dependent variable

Sample Decision Tree



(*) Lables | target variable are always at the leaf (leaf is homogenous subset)

(*) D-T can be used for classification (also multiclass) & Regression problems.

→ TDIDT → Top-down Induction of D-T.

Recursive partitioning.

→ ID3 (Quinlan) → Iterative Dichotomizer

→ C-ART → Classification And Regression Tree.

→ Assistant

→ C4.5 (Quinlan) ***

→ See5/C5.0 (Quinlan) ***

(*) Small D-Trees are always better. *****

TDIDT Algorithm :-

↳ To construct decision tree T from learning set S :-

→ If all examples in S belong to same Class C, Then make leaf labeled C

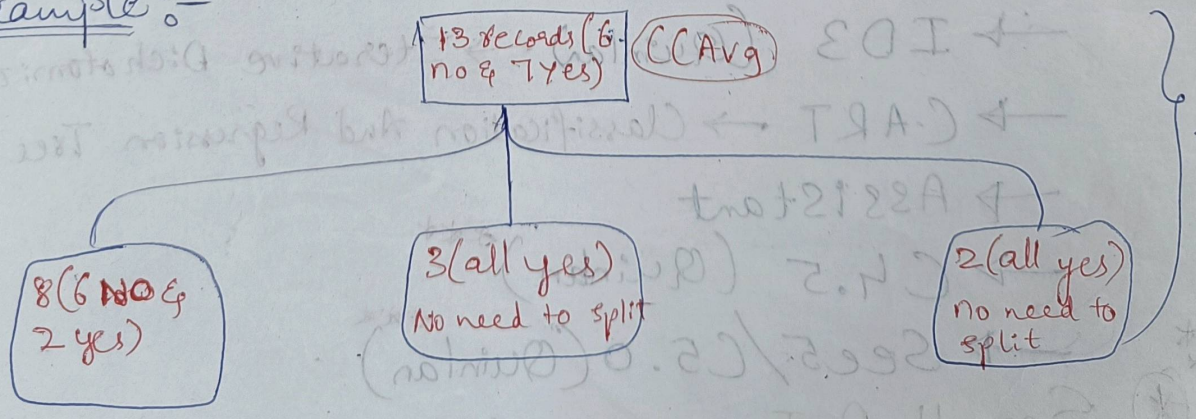
→ otherwise

- Select the "most informative" attribute (A)
- partition S according to A's value
- Recursively construct sub-trees $T_1, T_2, \ldots$, for the subsets of S

During Constructing a Decision Tree there will be following questions

- What types of decisions
  - → Which attribute to choose for split
  - → When to stop (to avoid overfitting)

Example :-



(i) Information Gain.

Entropy and Information Gain (ID3) :-

(*) $H = -\sum_i P_i \log_2 P_i$

P_i (Probability) → Class 1 / Total

(*) Information Gain = Entropy of the System before Split − Entropy of the System after split.

(*) Pure nodes (leaves) can be obtained faster by minimizing entropy at every split or maximize the Information gain at every split

(*) For an pure Class, entropy will be zero and entropy will be max(1) when both the Class in the lables are equal.

Entropy before split

$$H = -\frac{6}{13} \times \log_2 \frac{6}{13} - \frac{7}{13} \times \log_2 \frac{7}{13} = 0.9957$$

→ Entropy after split on CCAvg

$$H = \frac{8}{13}\left(\frac{-6}{8} \times \log_2 \frac{6}{8} - \frac{2}{8} \times \log_2 \frac{2}{8}\right) + \frac{3}{13}\left(\frac{-3}{3} \times \log_2 \frac{3}{3}\right) + \frac{2}{13}\left(\frac{-2}{2} \times \log \frac{2}{2}\right)$$

$H = 0.4992$

∴ Information gain = 0.9957 − 0.4992 = 0.4965

Advantages of Trees:-

- Fast
- Robust
- Explicable. (Explainable)
- Requires very little experimentation
- We can easily build some intuitions about the data.

Other measures of purity:-

→ Gini index

→ Information-theoretic

→ χ^2 (Chi-squared)

- Some time Information gain will fail if attribute choosen to split has all unique values and results in leaf which is equal to number of unique value. (like Id)
  - ↳ To prevent this problem we define a term called Information Content

(ii) Gain Ratio :-

Information Content :-

$$I.C = -\sum f_i \log f_i .$$

$f_i \rightarrow$ fraction of the members in a state divided by the total members

I.C for (I.D) → It has 13 states. So, the I.D

$$IC = \left(-\frac{1}{13} \times \log_2\left(\frac{1}{13}\right)\right) \times 13 = 3.7$$

Gain Ratio :- (used in C4.5)

(*) Since Information gain is baised towards attributes with many classes (levels)

.. Gain ratio normalizes information gain by dividing by the information content at the attribute.

$$\text{Gain Ratio}(A) = \frac{\text{Gain}(A)}{\text{Information Content}(A)}$$

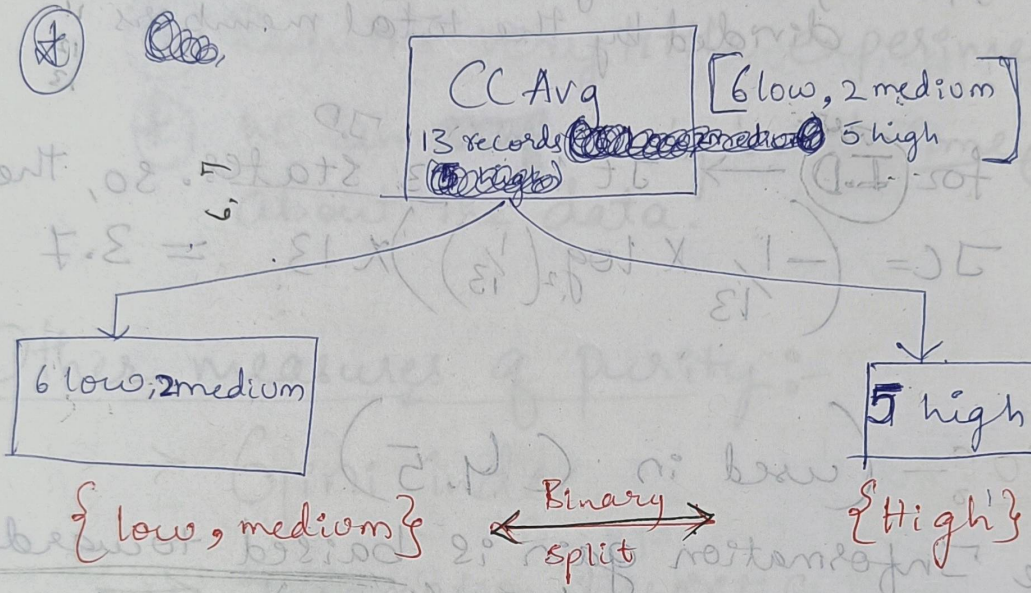
(*) Attributes with maximum gain ratio is selected as the splitting attribute.

(3) Gini Index :- (used in CART)

$$1 - \sum_{1}^{n} p_i^2$$

n → Classes
P → Probability

* It is computed on binary splits only.



* The one with least gini index is picked

Eg :- Gini index before split $= 1 - \left(\frac{6}{13}\right)^2 - \left(\frac{7}{13}\right)^2 = 0.497$

Gini index after split =

$$\frac{8}{13}\left(1 - \left(\frac{6}{8}\right)^2 - \left(\frac{2}{8}\right)^2\right) + \frac{5}{13}\left(1 - \left(\frac{5}{5}\right)^2\right) = 0.231$$

Regression Trees · uses (variance) · for splitting

* Minimizing the variance
  → variance before split
  → weighted average of variance after split
  → The attribute that reduces the variance most is choosen for the node.

↑ χ^2 ↓ Gini index

↑ Info gain ↓ variance.

↑ Gain Ratio

Bayesian optimal classifier → highest accuracy a decision tree can have but it is harder.

(*) D-T are unstable at leaf (Small change in leaf will change a lot) but stable at top

---

Pruning:- (*) Avoid overfitting

(*) The better method would be growing the entire tree and pruning the unwanted branches later.

(*) After choping, Check the errors on training & test data. to avoid overfitting. (Cost-Complexity Pruning)

(*) The previous method of growing a tree is called Reduced Error Pruning

CART	C 4.5
→ Binary split	→ multi split
→ [illegible]	→ Gain ratio
→ Cost complexity pruning	→ Pessimistic Pruning

Some Notes on Decision Tree Induction.

(*) For decision trees the decision boundry is rectangle.

Tessilation

K-Nearest Neighbour :-

* It is called lazy learner becoz KNN doesnot build any model, they just remember the training data and try to do the classification that we require.

* Lazy learner is large group of technique, KNN is one among them

* Can trivially adapt to evolving data, i.e since it will not built model, change in data can be handled and classify accordingly.

* Lot more computation must be done during Classifing than that of training.

⟹ Eg:- If a Customer came to mart and wants to buy something, we can compare with the other customers in the data like income, age at what time he came, etc.

By keeping track of these data and the buying pattern of those similar customer we can recomend the products they have bought earlier.

* Here Local learning is involved.

(*) In K-NN for every test sample we have to search the nearest neighbour.

→ Another example :-

If in case a new case is filed in the high court and we want to chose the bench members (judge) to take a decision, we can see the similar case in past year and the judges who given judgement and according those judges are preffered for bench members.

(*) Most Suitable for problems where one cannot assume that training and test samples are drawn from the same distribution.

(*) Selecting K is primary challenge.

Lazy learning (Fewer assumption about future is done compared to other Algorithms)

Similar Test Sample / Local data

Directly work with data (Instance Based learning / Case Based Reasoning)

Build a local model (Local reg / local sum etc.)

* Since we make fewer assumption, KNN has twice the Bayesian optimal error rate.

- KNN Data must not contain NOISE

KNN - Algorithm :-

* classification → Assign a class to a new data point based on its neighbours (mode)
* regression → Identify a numberic value of a new data point based on its neighbours (mean/median)

* weighted mean/mode of entire data.

* Store all input data in the training set
* For each instance in the test set
* Search for the k-nearest instances to the input instance using appropriate distance measure
* For classification, compute the confidence for each class as C_i/k.

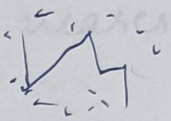
* The classification for the input instance is the class with the highest confedence.

(*) Curse of dimensionality → K-NN is fully distance based hence heavily impacted by C.o.f. dimens^

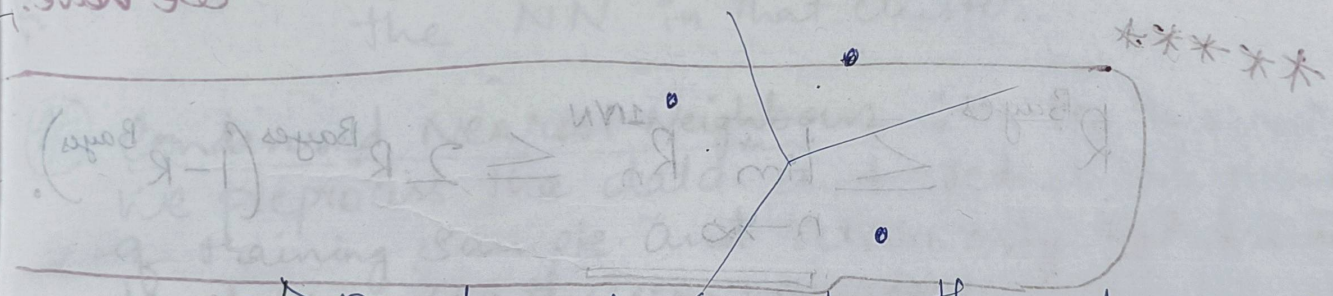
∴ In pre-processing, we must reduce the dimensionation using any feature Selection method.

(*) Decision Surfaces created by KNN :-

→ Voronai Diagram

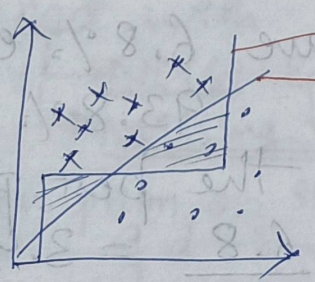


↳ Every point of different class is seperated by a perpendicular bisector b/w them



→ much more complex than decision trees

→ In voronai diagram there is no error/less error than that of axis parallel boundries in D-T



(*) 'K' is the number of the nearest neighbours, usually it will be odd number.

5

① Noisy training is an issue
② Can overfit
③ Missing values

→ Issues of KNN.

K should be less than the square root of the total number of training patterns

(*) KNN can predict labels of any type.

- Binary, multiclass, regression, multidimensional regression.
- Text mining (Part-of-speech sequence)

6

(*) Theorem 7:- In a two-class classification problem we have,

*****

$$R^{Bayes} \leq \lim_{n \to \infty} R_n^{1NN} \leq 2R^{Bayes}\left(1 - R^{Bayes}\right).$$

(*) → we can find how well can machine learning do on the particular data.

for eg:- K-NN gave 6.8% error i.e 93.8% Accuracy

∴ Acc to theorem the best possible error rate is $\frac{6.8}{2} = 3.4\%$ error rate

∴ we cannot get 96.6% Accuracy for that particular data.

(*) As K-value increases the decision curtail become linear and error rate slightly increases

(*) We can use K-means with KNN to speed up the process

↳ If in KNN, we have given a new test data then we will find the nearest Neighbours for that data

Speeding KNN using K-means

By using K-means, it will give the clusters that are present in data. Therefore the new data given can be first clustered and only find the NN in that clusters.

(*) Condensed Nearest Neighbour :- In this method we preprocess the data and reduce the number of training sample and retain only that are needed to define the decision boundry.

Decision Trees (Eager Learning)	K-Nearest Neighbours (Lazy learner)
(*) very good with → Noise & Missing value.	(*) Sensitive to Noise & Missing value
(*) Make assuption on shape of d-boundry	(*) Doesnot make any assumpn
(*) DT-Trees are not good when future looks diff from past	(*) Can work when future looks diff from past
(*) No Gaurantee of Accuracy	(*) Gaurantee of accuracy 2X R Bayes

SVD → Singular Value Decomposition

& Recommendation Systems :-

Molecules —to understand the molecule→ Composition.

Number —to understand a number→ Factorization

Matrix. —to understand a matrix.→ Decomposition

$$\begin{bmatrix} N_1 & N_2 & N_3 \\ P_1 & P_2 & P_3 \end{bmatrix}$$

In PCA → Identifing Principal Component

In SVD → Factorizing a Matrix.

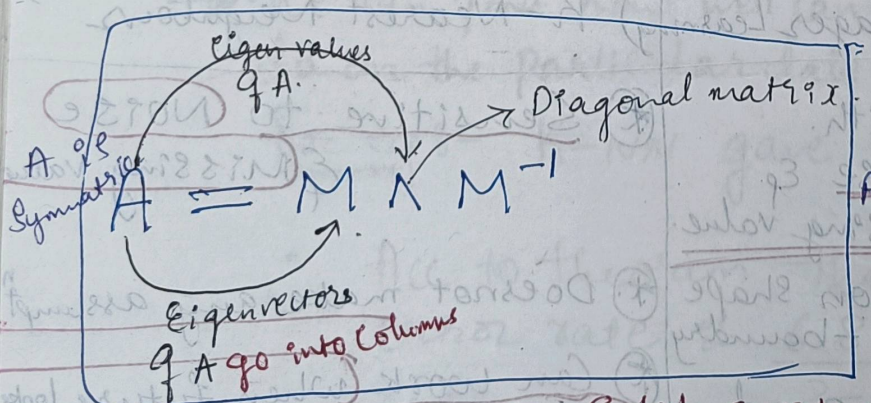
Factorization.

Ex :- 1728

$= 32 \times 54$

$= 64 \times 27$

$= 2^6 \times 3^3$



$A = AA^H \wedge A^H A$

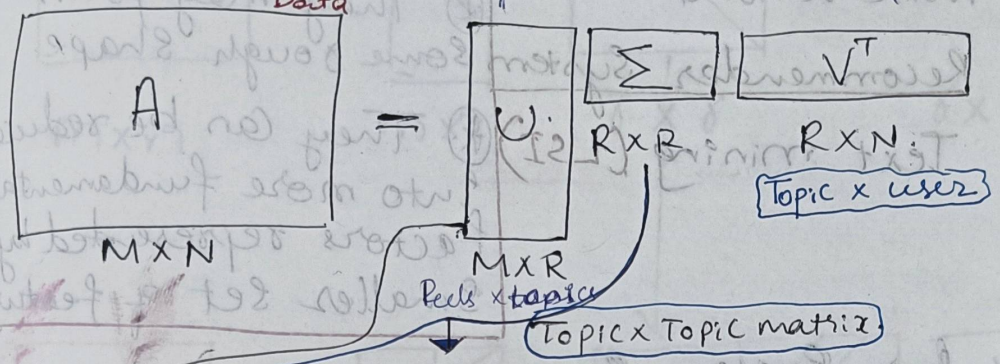
A^H = Har matrix.

SVD enables us to split overlapping influence data into their own space and remove noise

Rank (R) → must be decided by user

From these matrix we can find the similar users and similar reels or some relation using Cosine Similarity or Pearson Co-relation

Matrix of Data



- Diagonal matrix (Every entry other than diagonal are zero)
- Diagonal values = $\sqrt{\text{eigen values of } A \text{ \& } A^T}$
- Each row is orthonormal (independent to each other)

* Σ contains the eigen values in diagonal, as we go down the eigen values decrease i.e, the magnitude of stretch decreases ∴ we can set a threshold

* We use SVD to remove overlapping influences and also removing the noise (least impact on Eval) Since Attributes in rows and column have overlapping influences.

* These features are called latent dimensions, in raw data we have to find this.
  - This latent content is calculated by matrix Factorization (latent level)

Applications

- Data Compression
- Noise removal
- Recommender System
- Text mining (LSI)

SVD intuition of features

- The original space has a cloud (large) of points
- They may form some rough shape.
- They can be reduced into more fundamental factors represented by smaller set of features

Recommender System :-

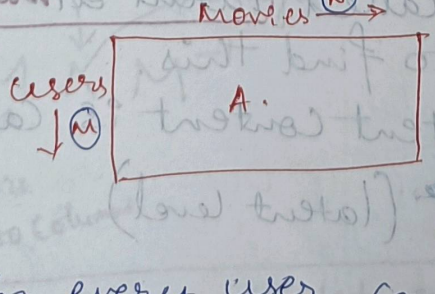
→ Here Recommender System is studied using an example of user ratings for Netflix movies.

Content Based → Netflix can hire expert to identify qualities of movie / users that tell how similar their tastes are.

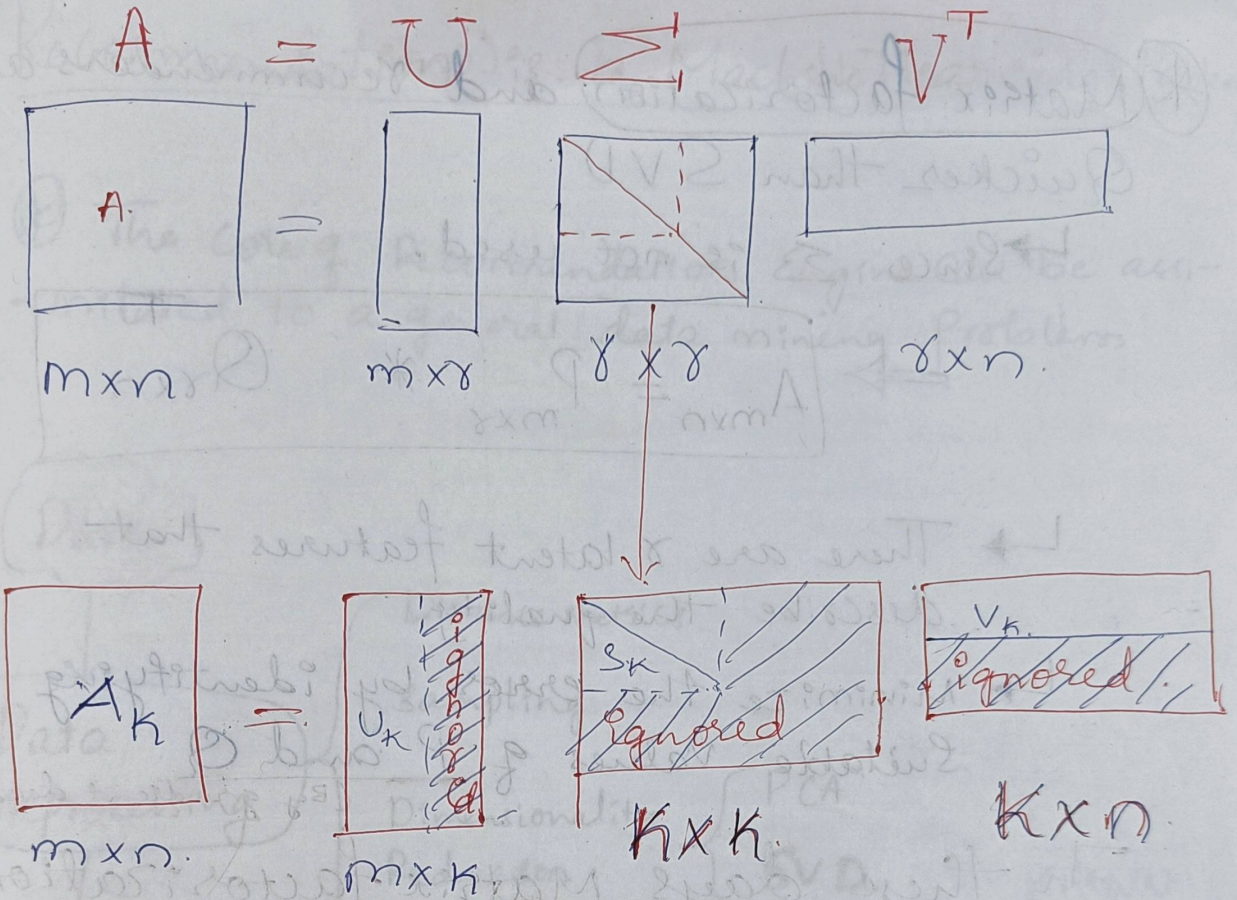
User-Behaviour Based → Identify similar movies (or users) using distance metric

Recommenders through SVD

- Create a user space through SVD
- Remove unwanted columns (Low eigen values)



- For every user compute the cosine similarity based distance and Identify closest users
- Look up the ratings they gave to the movie



(*) The reconstructed matrix $A_k = U_k S_k V_k$ is the closest rank-k matrix to the original matrix R.

(*) In rank matrix we will cut down to 'k' from 'r' i.e we remove the diminesion which have less eigen values

(*) This implies each user is in smaller dimensional space which helps to find user-similarity

(*) If we make U and V orthogonal and Σ diagonal, this can be the best factorization.

$$A = U \Sigma V^T$$

$$A^T A = V \Sigma^T U^T U \Sigma V^T = V \Sigma^2 V^T$$

V's → Eigen vector of $A^T A$

U's → Eigen vector of $A A^T$

(*) Matrix factorization and recommenders are Quicker than SVD

↳ Since Σ is not used.

$$\Rightarrow \boxed{A_{m \times n} = P_{m \times r} * Q^{T}_{r \times n}}$$

↳ There are r latent features that describe the quality

↳ Minimize the error by identifying suitable values of P and Q (By gradient decent)

In these days Matrix factorization is preffered more than SVD

Recommender Systems → Estimate a utility function that automatically predicts how a user will like an item. Based on:

- Past behaviours
- Relations to other users
- Item Similarity
- Context.

--- etc.

Recom

(*) Th

- mil

Data

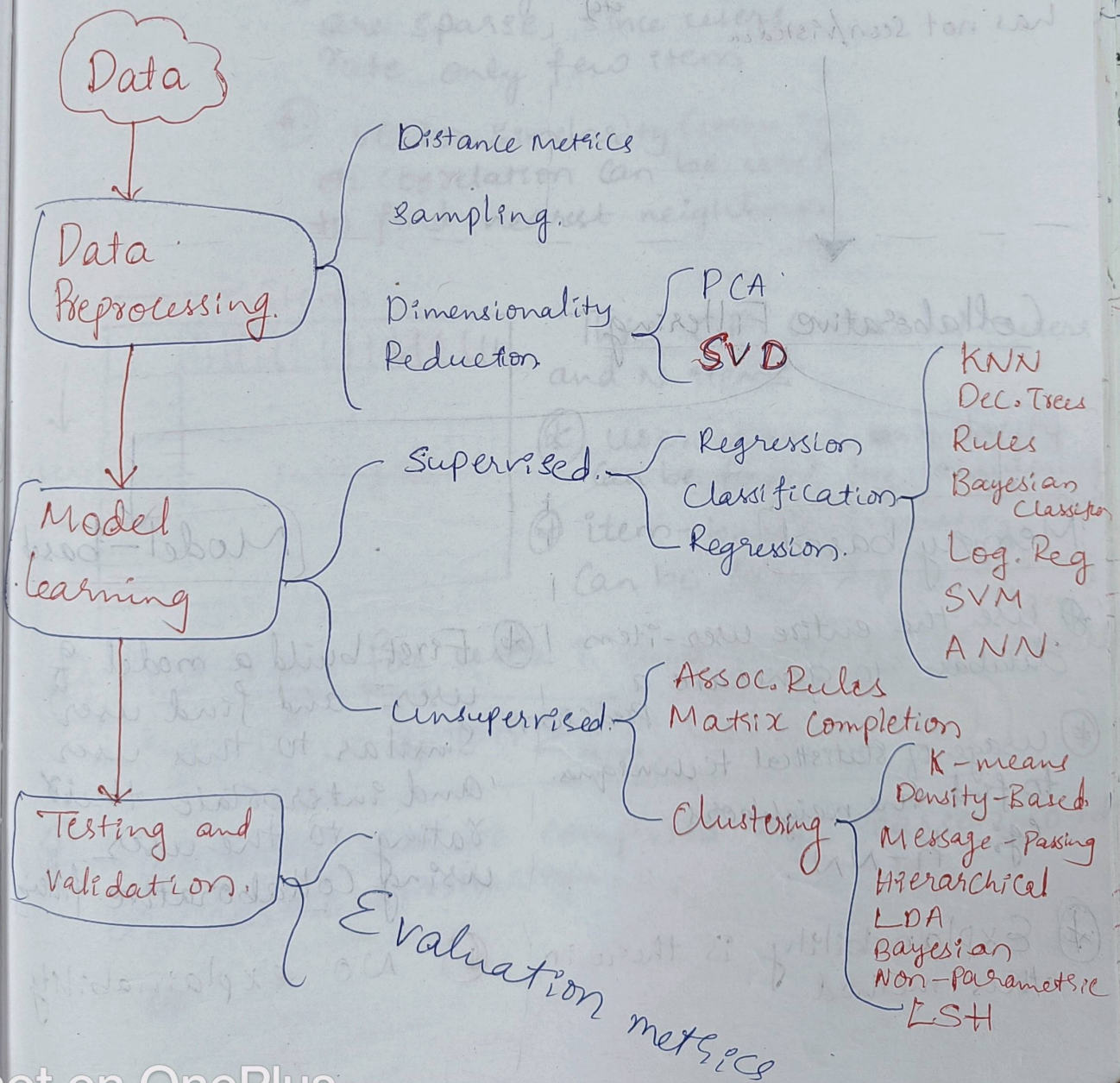
Data Preproc

Mode Learnin

Testin Valida

Recommendation is a Machine learning Problem

* The core of Recommendation Engine can be assimilated to a general data mining problem



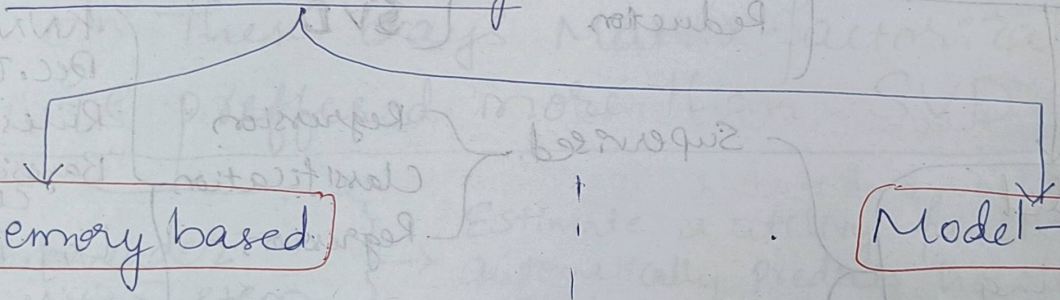
Collaborative Filtering

* Collaborative filtering find the similar rows (eg:- user) and recommends the column (eg:- movies, books etc) to other user who has not seen/visited etc

Content-Based Filtering

* Content-Based filtering recommends the column (eg:- movies, books, etc) based.

Collaborative Filtering



* Use the entire user-item database to generate a Prediction
* Usage of statistical techniques to find the neighbors eg:- K-NN.
* Explainibility is there in this method

* First build a model of user and find user similar to this user and interpolate their rating to the user using Collaborative filtering
* NO explainability

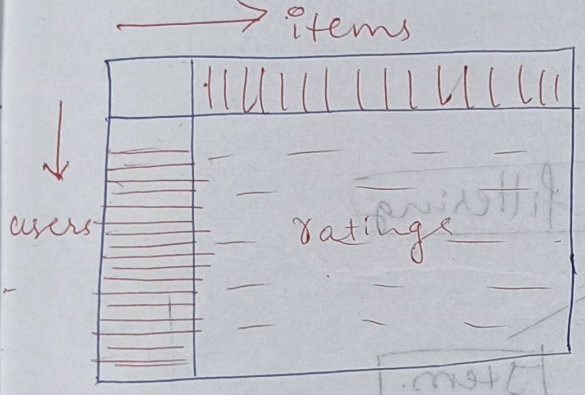
user-based

"users who liked this item also liked"

Item-based

"users who are similar to you also liked"

- Construct a vector for each user
- On average, user vector are sparse, since users rate only few item.
- vector similarity (cosine sim) or correlation can be used to find nearest neighbours



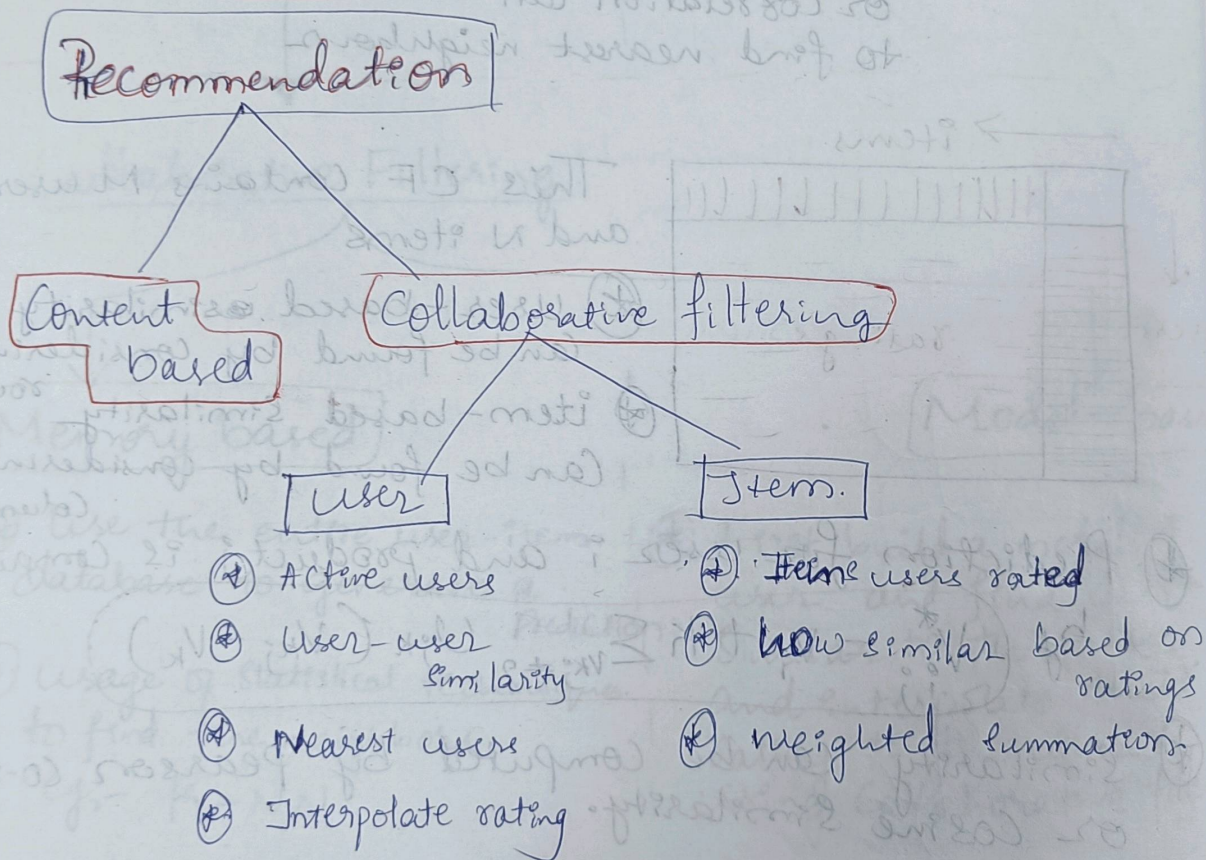
This CF contains M users and N items

- user-based similarity can be found by considering rows
- item-based similarity can be found by considering column

- Prediction for user i and product j is computed by $V^*_{ij} = V_i + k \sum_{vkj \neq ?} U_{jk} (V_{kj} - V_k)$
- Similarity can be computed by pearson co-rel or cosine similarity.

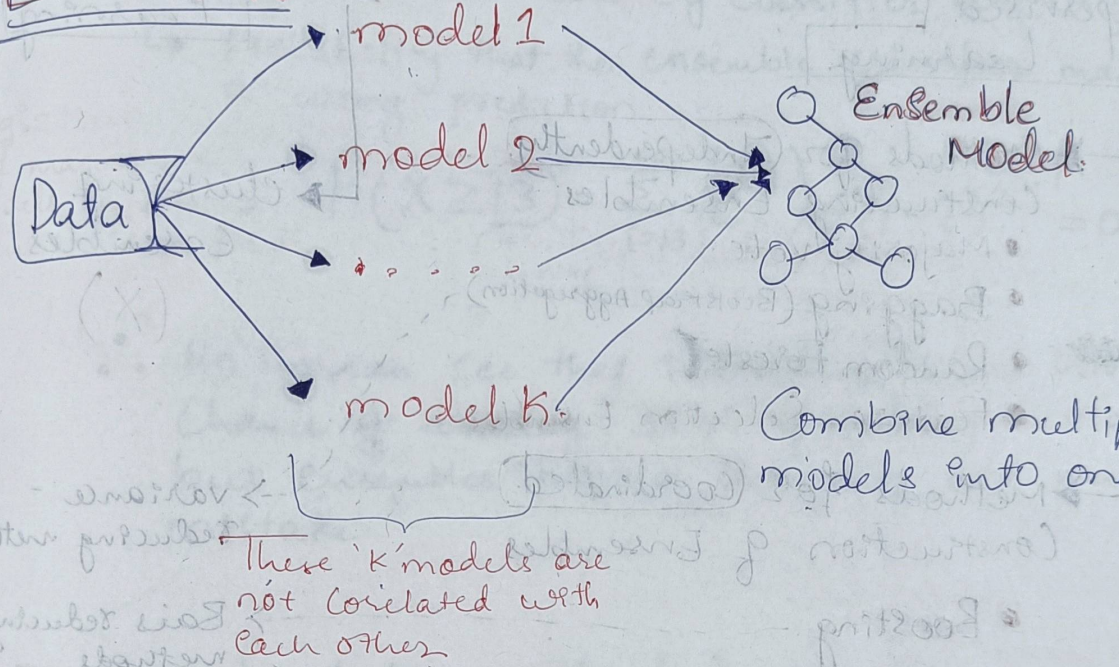
For item-based Col-filtering

1. Look into the items the target user has rated
2. Compute how similar they are to the target item
   - ↳ Similarity only using past ratings for other users.
3. Select k most similar items.
4. Compute Prediction by taking weighted average on the target users ratings on the most similar items.



Ensembles, RF, Stacking & SVM

Ensembles :-



Combine multiple models into one.

These 'K' models are not Corelated with each other.

Basic Idea → Build different "experts" and let them Vote.

Advantages :-

- Improve predictive performance.
- other types of Classifier can be directly included.
- Easy to implement.
- Not much parameter tuning.

Disadvantages :-

- The Combined model is not so transparent (black box)
- Not a compact representation.

Types of Ensembles.

Supervised Learning

- Methods for **Independently** Constructing Ensembles
  - Majority Vote.
  - Bagging (Bootstrap Aggregation) --> variance - reducing methods.
  - Random Forests
  - Feature-Selection Ensembles
- Methods for **Coordinated** Construction of Ensembles
  - Boosting ---------> Bais reducing methods.
  - Stacking
  - Mixture of Experts

Unsupervised Learning

- Clustering Ensembles (X)

Eg:-
If majo
models sa
then it

(2) Over

(3) Ens

Why do Ensembles work :- (Mathematically)

1. Suppose there are 25 base classifiers
   - Each classifier has error rate, $\varepsilon = 0.35$ (Accuracy = 0.65)
   - Assume errors made by classifier are uncorrelated
   - probability that the ensemble classifier makes a wrong prediction.

Eg:- If majority in 25 models says fraud, then it is a fraud;

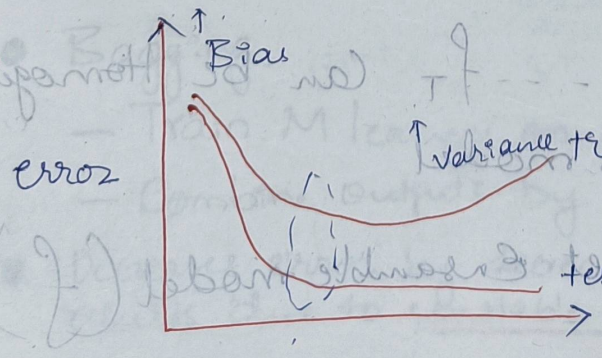
$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

(majority)

∴ Here we can see that individual model have Chance of 35% of making mistake but ensembles have a chance of 6% making mistake.

2. Overcome limitations of single hypothesis.
   - The target function may not be implementable with individual classifiers, but may be approximated by model averaging.

3. Ensemble methods combine learners to reduce bias & variance.

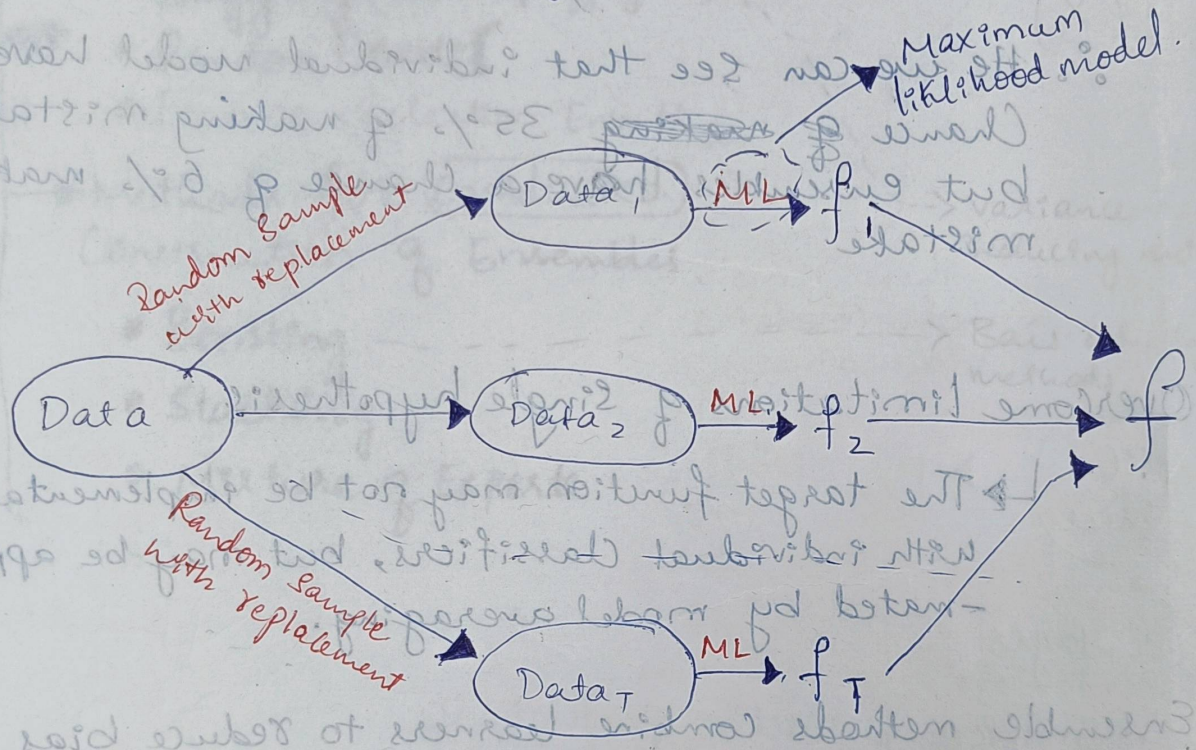


* Ensembles were most popular method before Neural Network came to existance.

* BAGGING

(Ensembles - Method 1)

Bootstrap Aggregation



* $f_1, f_2, f_3 \dots f_T$ can be Homogeneous or Heterogeneous model

* Average all to get Ensamble model (f)

Bootstrap aggregation :- Example. (Random Sampling with Replacement)

our initial dataset :- (1) (2) (3) (4) (5) (6)

First draw :	(1) (2) (2) (2) (3) (5)	Model 1
Second draw :	(1) (1) (3) (4) (4) (6)	Model 2
Third draw :	(2) (3) (3) (4) (5) (5)	Model 3
Forth draw :	(2) (4) (4) (5) (5) (6)	Model 4

Uncorrelated datasets :: uncorrelated model.

Average : Final model

Bagging :-

- Create ensembles by "bootstrap aggregation", i.e, repeatedly randomly resampling the training data.
  - Bootstrap :- draw N items from ~~X~~ with replacement
- Bagging
  - Train M learners on M bootstrap samples.
  - Combine outputs by voting (eg :- majority vote)
- Decreases error by decreasing the variance in the results due to unstable learners,

(*) Bagging is good for unstable learners as it reduces Variance and overfitting

→ To generate large number of unstable learners

(in Bagging) ↳ Choose records randomly

Random forest ↳ Choose attributes randomly.

(*) In Naive Bayes, Logistic regression, Linear regression

↳ Bootstrap of records fails (Because these are global classifiers Not large variance)

↳ but not bootstrap of features (works well)

↳ Decisiontrees will do wonders on both.

An example of Bagging

(*) 500-1000 trees Commonly used

(*) Random Forest :-

(1) Decision trees

↳ Non linear Classifiers

↳ Easy to use

↳ Easy to interpret

↳ Succeptible to overfitting

$m = \sqrt{p}$ → regression

$m = p/3$ → Classifi

10,000 + and 100

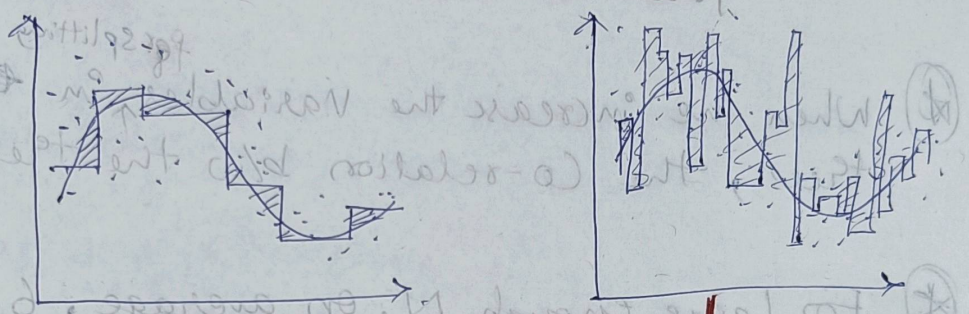
N = 10

M =

T = 5

* The problem with single decision tree which is NOT proned is poor predictor (noisy)
i.e it is overfit, it doesnot do well on test data. (on test data.)

* We use such unproned predictors (d-Trees) and average there output in boosting.



Bagging uses this high variance to our advantage!

10,000 training parameters and 100 dimentions

N = 10000
M = 100
T = 500

m = 10 (random subset)

m try

Random Sampling with Replacement

1000 sample

Choose 10 attributes out of 100 randomly

Best attr to split out of 10

K points

Choose 10 att out of 100

→ Best attr to split

1000 − K Points

Choose 10 attr out of 100

→ Best attr to split

Berkley & Stanford

* Bagging introduces randomness into the rows of the data

* Random forest introduces randomness into the rows and columns of the data

* Single decesion Tree will be used only for understand the rules in most cases after this radom forest is built to solve the prediction problem

* When we increase the variables for splitting in every tree, the Co-relation b/w the trees increases

* For large enough N, on average, 63.21% of the original records ends up in any bootstrap sample. Other 36.79% of the observations are not used in the construction of a particular tree.

∴ these unused 36.79% of attributes can be used as test sample for testing the model.

These observations are called or Considered Out of Bag (OOB)

* Random forest is extremly useful in feature selection. Since in every tree we can see which attributes are used more and select the important Variable.

Shot on OnePlus
By Krishna

Parameters for tuning in Random forest :-

1. Number of trees
2. mtry
3. Node size ---?
4. Sampling Scheme ---?
5. Split rule.

Advantages :-

- Competitive performance.
- Doesnot overfit
- Robust to outliers
- Handles missing data (imputation is not required)
- Provide automatic feature selection.
- Minimal preprocessing time.

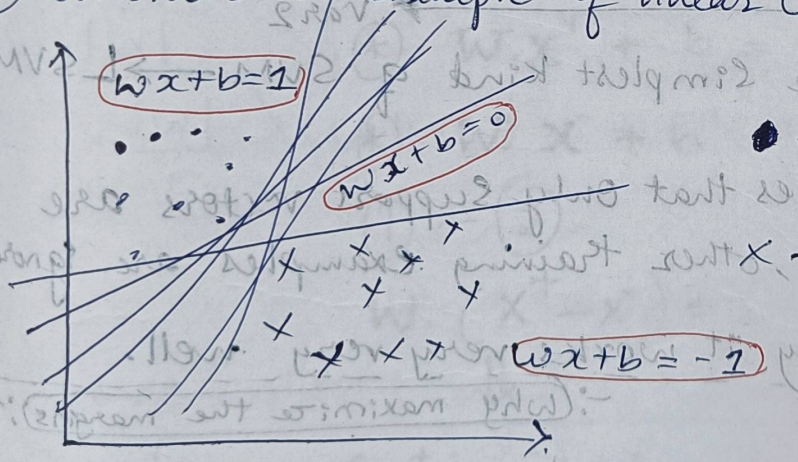
Disadvantages :-

- Cannot compete with advance boosting algorithms
- Can become slow on large data sets
- Less interpretable

Support Vector Machines :- (SVM)

* SVMs have clever way to prevent over-fitting.
  - SVM is arguably most imp & intresting recent discovery in ML by Vladmir Naumovich Vapnik
  - SVMs have a very clever way to use huge (number of features) without requiring nearly as much computation as seems to be necessary. (works well in higher dimensions.)

* Consider an example of linear classifier.



● → denotes +1

× → denotes −1

* Here which linear seperator to pick, so that it minimizises the error for training set and works well for test set.

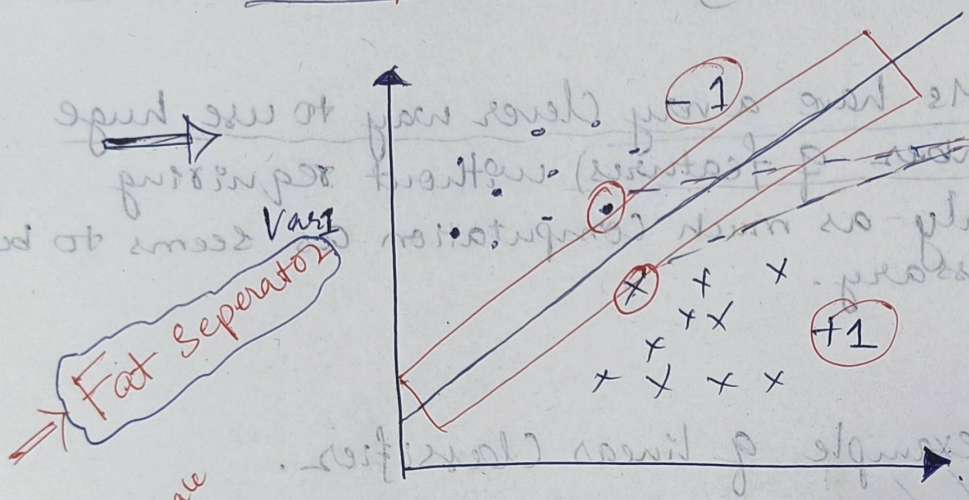
↓

Problem

(*) To work well on test set the classes must not be nearer to seperator (as equally far from both of the seperator)

i.e the seperator must maximize the margin at the boundry

∴ We need maximum margin linear classifier, which maximizes the margin



This is the simplest kind of SVM → LSVM

↳ This implies that only support vectors are important, other training examples are ignored

↳ Empirically it works very very well.

For this reason it reduces the search space and runs faster

∴ (Why maximize the margins) ∴

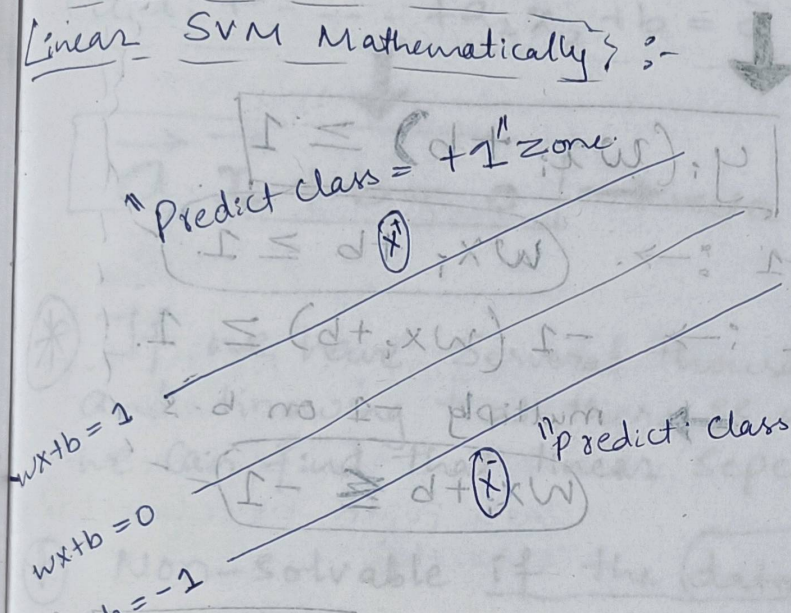
↳ A Classifier with a large margin makes no (low-certainty) classification decisions.

Since points near decision surface are uncertain classification decisions (50% either way)

↳ Fewer choices of where it can be put

(*) The Search problem becomes simpler in higher dimensions because in hig.dims the margin classifies inflate in all direction and finding the best fit becomes easy.

Linear SVM Mathematically :-



$\frac{2}{|w|} = M$

what we know :-

(*) $w.x^+ + b = +1$ —(1)

(*) $w.x^- + b = -1$ —(2)

(1) − (2)

$w.(x^+ - x^-) = 2$

$\Downarrow$

For any dimension $\Longrightarrow$ $\boxed{M = \frac{(x^+ - x^-).w}{|w|} \Rightarrow \frac{2}{|w|}}$ M → Margin width.

(*) we must Maximize $\boxed{M}$ margin width.

Goal : (1) Correctly classify all training data

$wx_i + b \geq 1$ if $y_i = +1$

$wx_i + b \leq -1$ if $y_i = -1$

↓

Simplifing those two eqn. → $\boxed{y_i(wx_i + b) \geq 1}$

if $y_i = +1$:→ $(wx_i + b \geq 1)$

if $y_i = -1$:→ $-1(wx_i + b) \geq 1$.

→ multiply -1 on b.s

$(wx_i + b \leq -1)$

(2) Maximize the Margin $\boxed{M = \frac{2}{|w|}}$

→ minimize $\frac{1}{2} w'w$ / $\frac{|w|}{2}$

$\therefore |w| \Rightarrow w'.w$

This is a quadratic optimization problem in linear constraints. Solve for w & b.

Squared quantity → quadratic

$\therefore$ Minimize $\phi(w) = \frac{1}{2} w'w$ (obj func^n)

Subject to $y_i(wx_i + b) \geq 1$ for all i $\forall_i$

(Constraints)

$ax + b = 0$ → 2-D

$a_1x_1 + a_2x_2 + b = 0$ → 3-D (plane)

$a_1x_1 + \text{----} + a_2x_2 + b = 0$ → Hyperplane

$$\vec{a} \cdot \vec{x} + b = 0$$ → vector representation of hyperplane

(*) If we have several thousand dimensional data points and knowing that there is a linear seperator, we can find that linear seperator guaranteed by LSVM.

(*) Non-solvable if the data is not linearly seperable

(*) Final Conclusion, For aurbitary no of dimensions for aurbitary no of datapoints, if there exists linear seperator, we can find the seperator by simple Quatratic programming.

(*) LSVM is 2-class classifier, for 3-class classifier we build 3 LSVM seperator and choose the one with highes confidence.

For (A)(B)(C)

1st LSVM → A, BC

2nd LSVM → B, CA

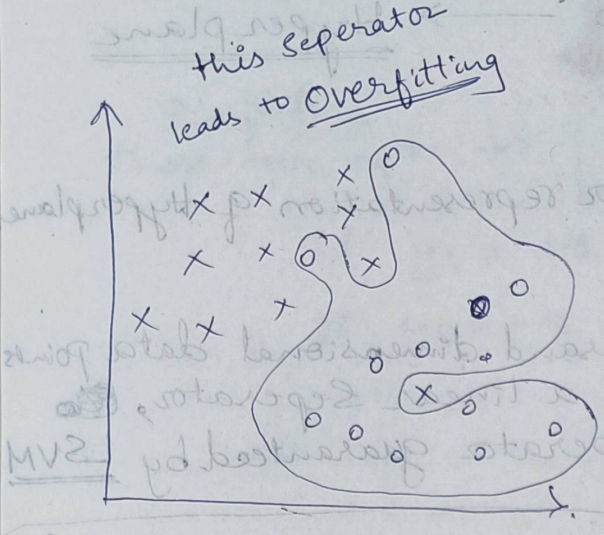
3rd LSVM → C, AB

} Choose one with highest confidence.

Dataset with noise

① Initially try with soft margin classifier with linear seperator

② If ① not work try non-linear seperator.



x : denotes → +1

o : denotes → −1

Hard margin :- NO training error.

↳ So far we require all data points be classified correctly

→ What if the training set is noisy.

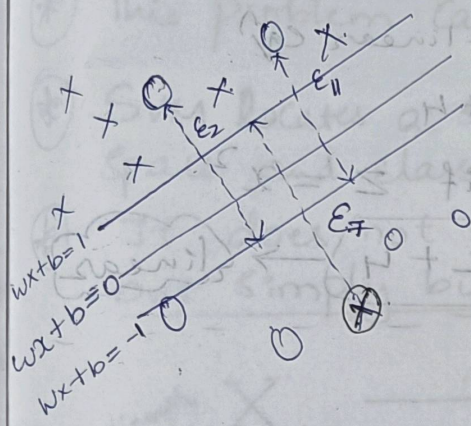
① Solution 1 :- use very powerful kernels

To avoid overfitting we introduce

↳ Soft Margin Classification :-

→ Slack variables can be added to allow misclassification of difficult or noisy examples.

(ε)



∴ we must minimize

$$\frac{1}{2} w \cdot w + C \sum_{k=1}^{R} \varepsilon_k$$

C → Cost Parameter (weightage of Error rate)

(*) The distance of noisy points from the seperate (ε) must be minimal ∴ we add the (ε_k) term of all noisy points to the objective function

---

Hard margin Vs Soft Margin

(*) Soft Margin also, always has a solution

(*) Soft-Margin is more robust to outliers

(*) Hard margin doesnot require to guess the Cost parameter (requires no parameter at all)

---

for soft margin, constraint is $y_i (\vec{w} \cdot \vec{x} + b) \geq 1 - \varepsilon_i$

Non-linear SVMs :-

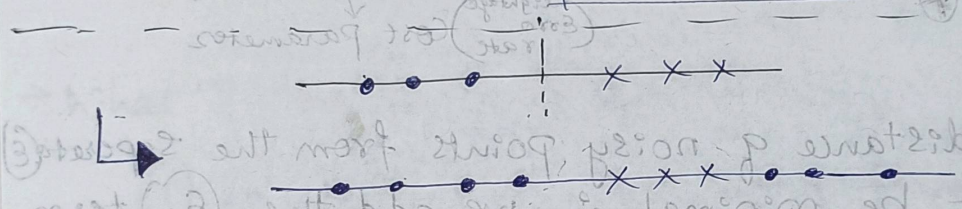
* The main theme of non-linear SVMs is, a non-linear equation in a particular dimension becomes linear in higher dimensions.

for eg: consider non-linear eqn

$$y = x^2 + 3x + 4$$

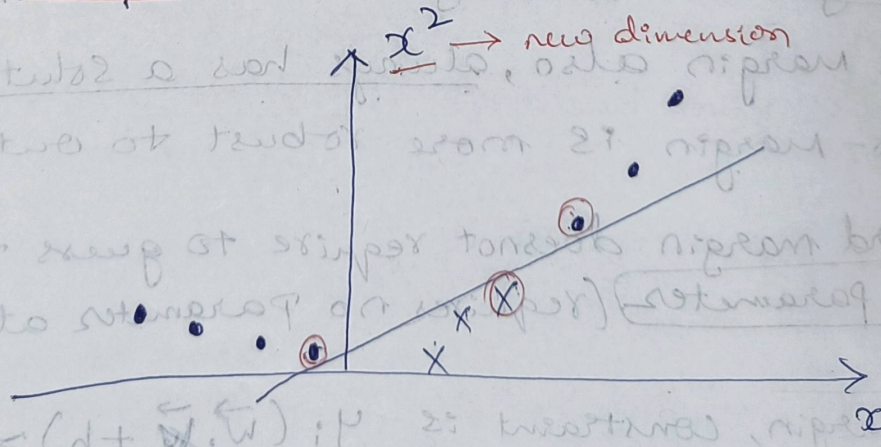
New dimens^n → if $z = x^2$

$$\Rightarrow \boxed{y = 3x + z + 4} \rightarrow \text{linear}$$



The above points cannot be linearly seperable.

* Map the points to higher dimensions



The original input space can always be mapped to some higher-dimensional feature space where the training set is seperable.

$$\phi : X \longrightarrow \psi(x)$$

* This problem can be solved using KERNEL TRICK
* SVM locates a separating hyperplane in the feature space and classify points in that space.
* It does not need to represent the space explicitly, but simply by defining a kernel function

non-linearly seperable $\begin{cases} X_1 \longrightarrow \phi(x_1) \\ X_2 \longrightarrow \phi(x_2) \end{cases}$ → In this space they are linearly seperable

Lower dim → higher dim.

↳ we find a linear seperator in higher dimension by doing the dot product

$$\Rightarrow \phi(x_1)^T \cdot \phi(x_2)$$

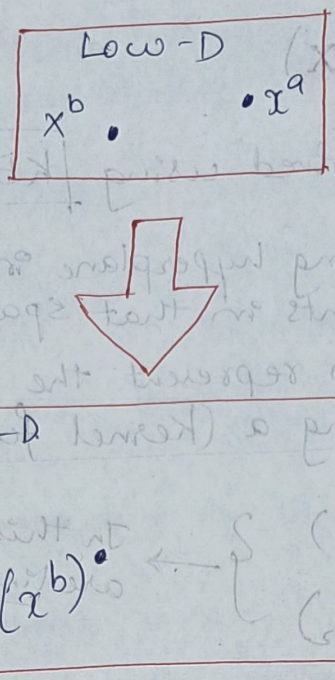
↳ Therefore kernel function is that function, if it is applied on X_1, X_2 ($f(X_1, X_2)$) such that the output is $\phi(x_1)^T \cdot \phi(x_2)$.

we define.

This trick helps to get these scalar product without moving to higher dimension i.e by staying in current dimensional space.

Kernel TRICK

(*) The Kernel function plays the role of the dot product in the higher-dimensional feature space.



$$K(x^a, x^b) = \phi(x^a) \cdot \phi(x^b)$$

↑ letting the kernel to do the work

↑ doing the scalar product in the obvious way.

Examples of kernel functions :- (Picking the K.function is our choice)

- Linear : $K(x_i, x_j) = x_i^T x_j$
- Polynomial of power p : $K(x_i, x_j) = (1 + x_i^T x_j)^p$
- Gaussian (radial-basis function network):

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

- Sigmoid : $K(x_i, x_j) = \tanh(\beta_0 x_i^T x_j + \beta_i)$

etc (Many kernels are there)

Properties of SVM :-

- Flexibility in choosing a similarity function - - ?
- Ability to handle large feature spaces (Higher diminsions)
- overfitting can be controlled by soft margin approach
- Nice Math property :- a simple convex optimization Problem which is guaranteed to converge to a single global solution.
- Feature selection - - ?

Examples of kernel functions :- (Picking the k.function is our choice)

- Linear : $K(x_i, x_j) = x_i^T x_j$
- Polynomial of power p : $K(x_i, x_j) = (1 + x_i^T x_j)^p$
- Gaussian (radial-basis function network):

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

- Sigmoid : $K(x_i, x_j) = \tanh(\beta_0 x_i^T x_j + \beta_1)$

etc (Many kernels are there)

Properties of SVM :-

- Flexibility in choosing a similarity function - - ?
- Ability to handle large feature spaces (Higher diminsions)
- overfitting can be controlled by soft margin approach
- Nice Math property :- a simple convex optimization problem which is guaranteed to converge to a single global solution.
- Feature selection - - ?

SVM Applications :-

↳ used in many real world problems

→ text & hypertext categorizations (Features: word count)

→ image classification

→ bioinformatics (Protein, Cancer classific)

→ hand-written character recognition

→ DNA array expression data analysis

Disadvantages :-

- Sensitive to more noise.
- It only considers two classes.
- Choice of kernel.
- Choice of kernel parameters.

The Classification rule :-

The final classification rule is quite simple.

$$\text{bias} + \sum_{s \in SV} w_s K(x^{test}, x^s) > 0 \quad ---?$$

- x^{test} → test vector
- x^s → support vector
- $\sum_{s \in SV}$ → The set of support vectors.

$> 0 \rightarrow C1$
$< 0 \rightarrow C2$

All the cleverness goes into selecting the support vectors that maximize the margin and computing the weight to use each support vector.

Advance ML models :-

- SVM.
- Deep learning
- Ensembles (RF, XGboost.)

→ 95% sofesticated ML Models