ML

- Supervised learning (both input and output data given and finding relationship in between them) → regression / classification
- Unsupervised learning (creating groups just from input data) → clustering
- Semi-supervised ML → Reg + classification + clustering { Combo of supervised and unsupervised }

Linear Regression ($y = mx + c$)

- When to apply linear regression
  - y vs x → linear tendency
  - mean of the residual should be zero.
  - error terms are not supposed to be correlated (exogenity)
  - x and residuals must be uncorrelated
  - error terms are supposed to be normal dist
- Error term must showcase constant variance (homoscedasticity)
- No multicollinearity

{ $\hat{y}$: predicted ; y : observed }

- residual = $r = y - \hat{y} = y - (mx + c)$
  - $r^2 = (y - \hat{y})^2 \longrightarrow \sum_{i=1}^{M} r^2 = \sum_{i=1}^{M} (y - \hat{y})^2$ → must be made minimum for best fit

$$\frac{d\left(\sum_{i=1}^{M} r^2\right)}{dm} = 0 = \frac{d\left(\sum_{i=1}^{M} r^2\right)}{dc}$$

$$m_{new}^{slope} = m_{old}^{slope} - \eta \frac{\partial}{\partial m}\left(\sum_{i=1}^{m} (y_i - \hat{y}_i)^2\right) = m_{old}^{slope} - \eta \nabla E_m$$

$$c_{new} = c_{old} - \eta \frac{\partial}{\partial m}\left[\sum_{i=1}^{m} (y_i - \hat{y}_i)\right] = m_{old}^{slop} - \eta \nabla E_c$$

Gradient descent method

[η → controls the rate of convergence → learning rate]

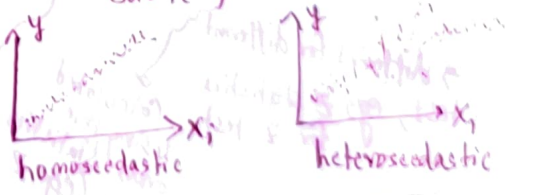
{ exogenity :- x and residuals must be uncorrelated }

{ Multicollinearity :- no collinearity expected from two or more independent variables (x_1 and x_2 not correlated) }

- because it will otherwise start learning relations among feature-feature and will affect feature-label relations

homoscedasticity

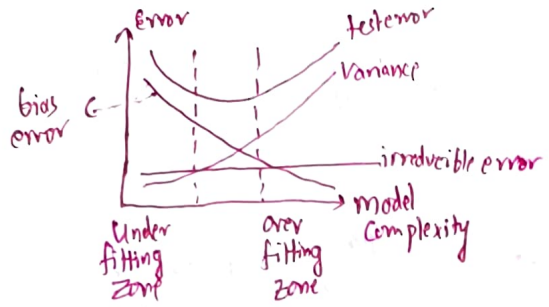
- all random variables have the same finite variance



- Accuracy of the model → $R^2 = 1 - \frac{RSS}{TSS}$
  - RSS → residual summation of squares $\sum_{i=1}^{n} (y - \hat{y})^2$
  - TSS → total summation of squares $\sum_{i=1}^{n} (y - y_m)^2$

→ Bias - Variance trade-off

↳ test error (error in test data) dependent on → bias error (error from wrong model assumptions)
→ Variance (error from sensitivity to small fluctuations in train data)
↳ irreducible error (inherent noise in problem itself)



→ Comes from oversimplification of the model (taking less parameters)
→ Comes from over complexity of the model (taking more parameters)

↳ **Adjusted R-square** → Corrected gof (model accuracy) measure for linear models.
→ % of variance in the data (dependent variable) explained by independent variable(s) in regression models
→ adjusted for the no. of attributes taken (minimizes the apparent effect of overfitting)

$$R^2_{adj} = 1 - \frac{(1-R^2)(n-1)}{n-p-1}$$

$n \rightarrow$ no. of samples
$p \rightarrow$ no. of features (attributes)

↳ **Selecting features for doing ML (linear regression)** → model (Multilinear Regression) $\{y = m_1x_1 + m_2x_2 + \dots + c\}$

↳ Check for multicollinearity: if two features multicollinear ⇒ take only one of them (Check correlation values) (Otherwise overfitting)

↳ if to take a particular feature for regression

↳ check p-values :: H_0 :: feature x_i doesnot affect y ⇒ $y = mx$ with $m = 0$

→ if pvalue ($P(TS \le ts|H_0)$ ↓ lower tail test, $P(TS \ge ts)$ ↓ upper tail test, $P(TS \ge |ts|)$ ↓ 2 tail test) $< \alpha$ ↓ Significance value

TS: test statistics
ts: observed test statistics from the data

→ different for different test, eg: z-statistics for z-test

Calculated using the various statistics tables (eg: z-table)

→ by dependent on x_i

$cdf(ts)$ | $1 - cdf(ts)$ | $2(1 - cdf(|ts|))$

* Note → Student's t-distribution (t-test) used in statsmodels formula api (Python)

↳ **Variance Inflation Factor (VIF)** to check multicollinearity → $VIF = \frac{1}{1-R_i^2}$

Obtained by regressing one feature (x_i) with another (x_j)

VIF = 1 (not correlated)
= 1-5 (moderately correlated)
= >5 (high correlation)

↳ Dummy variables trap
- Dummy variables → Creating additional attributes with values (0 or 1) for each class of a categorical value → one-hot encoding
- Dummy variable trap → high Correlation among variables
  - ↳ One dummy variable can be predicted from other dummy variables (due to one-hot encoding) and if all the dummy variables included ⇒ dummy variable trap
    - ↳ drop one of the dummy variable
    - ↳ So, need to take only independent dummy variables!!

↳ Regularization → technique to discourage learning a more complex or flexible model, ↳ reducing the risk of overfitting → trying to incorporate noise hence less predictability for test dataset.

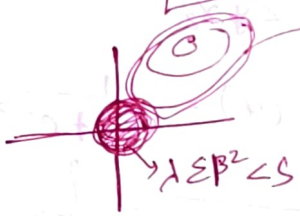
↳ Loss function for model → $Y = \beta_0 + \beta_1 X_1 + \cdots \beta_p X_p$

↳ $RSS = \sum_{i=1}^{n} \left[y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right]^2$

↳ Ridge regression (L_2 norm) → $\text{Loss function} = RSS + \lambda \sum_{j=1}^{p} \beta_j^2$ → minimize this func

↳ λ: tuning parameter to decide how much to penalize the abrupt change in β_j

↳ Lasso regression (L_1 norm) → $\text{Loss function} = RSS + \lambda \sum |\beta_j|$



→ Ellipses are the equivalued RSS values

↳ But in L_1 RSS and $\Sigma \beta^2 \leq s$ can touch at axes ⇒ dimensionality ~~de~~reduction probable

↳ in L_2 RSS and $\Sigma \beta^2 \leq s$ will never ~~touch~~ touch at axes ⇒ no variable can ever be ~~used~~ reduced to zero ⇒ no chance of dimensionality reducti

↳ Elastic net regression : → linear Combination of both L_1 and L_2

↳ Standardization (using sklearn.preprocessing.StandardScaler)
- ↳ to bring all the atributes to the same scale
- ↳ This helps the model to have more robust relations.
- ↳ $\left(z = \frac{x - \mu}{\sigma}\right)$

↳ **Cross-Validation**

(a model validation technique using resampling)

Used as a method to find the best possible tuning parameter in regularization (or elsewhere)

Used for tuning the hyper parameters everywhere (like λ)

CV = 5 ⇒ divides the data-set into 5 chunks and performs iteratively testing and training by taking different portions of the data ↘ one-round of CV

↳ multiple rounds of ~~itera~~ CV done taking into account the number of iterations given.

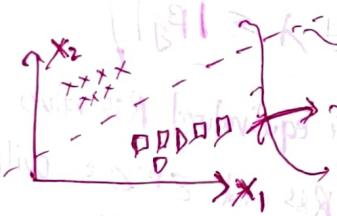
→ Logistic Regression

↳ is often used for classification and predictive analysis.

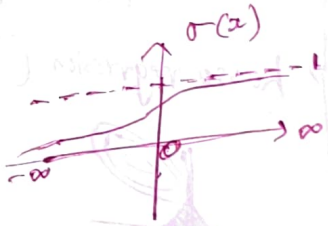
↳ estimates the probability of an event occuring, such as voted / didn't vote

↳ logistic regression not a classification algorithm on it own. Only classification in combination with a decision rule. It is regression because it estimates the probability of class membership

↳ drawing a line to divide the classes. $(\beta_1 x_1 + \beta_0)$



↳ Probability model :→ Sigmoid function $\sigma(x) = \left(\frac{1}{1+e^{-x}}\right)$



Binary Classification

$$P(x) \text{ or } = \frac{1}{1 + \exp[-(m_1 x_1 + m_2 x_2 \ldots + c)]}$$ → category-1

Can be rewritten as $P(x) = \frac{1}{1+\exp[-(\beta_1 x + \beta_0)]}$ → normalized $P(x) = \frac{e^{\beta_1 x + \beta_0}}{1 + e^{\beta_1 x + \beta_0}}$ —①—

$$P(\bar{x}) = 1 - P(x) = \frac{1}{1+\exp[\beta_1 x + \beta_0]}$$ —①—

→ here learnable parameters are β_1, β_0

↳ need to find β_1 β_0 to get $P(x)$

logit function (logistic function)

$$\log\left(\frac{P(x)}{P(\bar{x})}\right) = \beta_1 x + \beta_0$$

used to get in which region the parameter is present.

based on $P(x)$ we get y

$$y = \begin{cases} 0, & P(x) \leq 0.5 \\ 1, & P(x) > 0.5 \end{cases}$$

→ To get β_1, β_0 we use MLE

↳ Cost function for logistic regression (Binary)

$$\text{Cost}(P(x), y) = \begin{cases} -\log(P(x)) ; & y = 1 \\ -\log(1 - P(x)) ; & y = 0 \end{cases}$$

generalisation

$$\text{Cost} = \frac{-1}{m} \sum_{i=1}^{m} \left[y^i \log(P(x^i)) + (1 - y^i) \log(1 - P(x^i)) \right]$$

m → no. of data points

↳ log-loss equation.

Nt:→ the eqⁿ:→ $(m_1 x_1 + \dots m_n x_n + c = 0)$ defines the dividing line in the parameter space where at either side the two classes exist. at that line $P(x) = 1/2$

↳ Minimize Cost function → $C = \frac{-1}{m} \left[y \log(\sigma(\theta)) + (1 - y) \log(1 - \sigma(\theta)) \right]$

→ Gradient descent →

here $\theta = m_1 x_1 + m_2 x_2 + \dots c$

$$m_j^{new} = m_j^{old} - \eta \nabla E_m$$

$$\nabla E_m = \sum_{i=1}^{m} \frac{1}{m} [\sigma(\theta) - y] x_j = \frac{\partial C}{\partial x_j}$$

→ Evaluation metrics for logistic regression

↳ Confusion matrices

Predicted value \ Actual values	(1)	(0)
(1)	True Positive (TP)	False positive (FP)
(0)	False Negative (FN)	True negative (TN)

Accuracy $= \left(\frac{TP + TN}{TP + TN + FP + FN} \right)$

↳ not always reliable

$\begin{pmatrix} 10 & 40 \\ 10 & 70 \end{pmatrix}$ → Accuracy $= \frac{70}{130}$

but $TP/FP = \frac{10}{50}$

↳ Recall (Sensitivity) $= \left(\frac{TP}{TP + FN} \right)$ → $P(\text{True} \mid \text{it is True})$

↳ how much probable you are to detect a positive case

↳ Precision $= \left(\frac{TP}{TP + FP} \right)$ → $P(\text{it is True} \mid \text{predict true})$

↳ given the tve predictions how many are actually positive

↳ F1 score $= \left(\frac{2(PR)}{P + R} \right)$ → takes both precision and recall into account

↳ ROC (Receiver Operating Characteristic) → graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

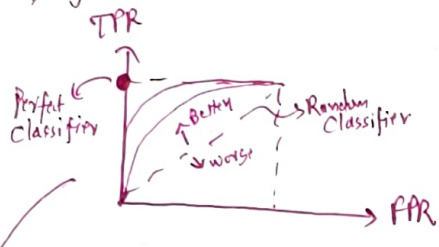
↑ True positive rate

→ False positive rate

↳ TPR (true positive rate) → sensitivity (recall)

↳ FPR (false positive rate) → 1 − (specificity) → $\frac{FP}{TN+FP}$

↳ for different thresholds draw ROC curve → take the point (threshold) where TPR high and FPR low

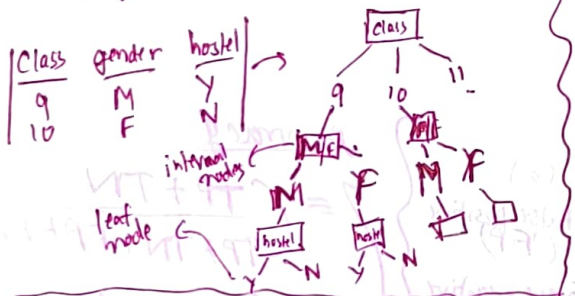


↳ AOC (Area under the Curve) → for different models (different (m_i, c) values) we have different ROC curves.

↳ higher AOC ⇒ better model

Decision Tree → helps in uncovering non-linear and non-clustered relationships among the attributes.

↳ flowchart-like tree structure with
- each internal node: a test on an attribute
- each branch: an outcome of the test (or record)
- each leaf node (terminal node): class label → class of the predicted



↳ How to build a Decision tree ? → using Gini / using entropy

↳ to increase the certainity in finding the final leaf node we must always try to zero down on it by starting from least cluttered decisions. This will help to get high purity nodes.

↳ root nodes should be of attributes with least disorder.

Gini Impurity Index

$$G_I(t) = 1 - \sum_{i=1}^{j} [P(i|t)]^2$$

j :- no. of classes of the label (prediction)

t :- one category of the given attribute (whose impurity needs to be calculated)

↳ ranges from 0 (pure) → 0.5 (most impure)

↳ Gini for an attribute :-

$$G_I(\text{attribute}) = \sum_t \frac{n_t}{n} G_I(t)$$

t: runs over all the categories of the attribute

n_t :- no. of instances having that attribute

$\sum_t n_t = n$ = total no. of instances

↳ Root node :- attribute with least G_I(attribute)

Entropy and Information Gain

$$E(t) = \sum_{i=1}^{j} P(i|t) \log_2 [P(i|t)]$$

↳ higher entropy ⇒ higher impurity.

↳ pure node → $E(t) = 0$

↳ Entropy for an attribute :-

$$E(\text{attribute}) = \sum_t \frac{n_t}{n} E(t)$$

↳ Information gain :- $IG = E_{initial} - E_{split}$

↳ $E_{initial}$ = entropy without the split (ie $\sum P(i) \log_2(P(i))$)

↳ E_{split} = entropy after the split using the attribute

↳ Root node :- attribute with highest information gain should be preferred as root node.

↳ CART algorithm (Classification and regression tree)

for attributes having categorical values, the gini index is used to ~~minimize~~ build the decision tree (seen earlier)

↳ for continuous attributes: → subnodes are generated by considering the best attribute and threshold value

↳ which threshold value and attribute is the best?

↳ minimize, RSS (residual sum of squares) → cost function

{Greedy Algorithm}

$$= \sum_i [y_i - \hat{y}_i]^2$$

↳ y_i: label from the training set

$\hat{y}_i$: predicted value by taking avg (y_i) in the subnode.

Nt: for a single-node, there can be multiple threshold values. If only one ⇒ binary classification

Stopping-Criterion

↳ minimum amount of training data allocated to each leaf node

↳ tree length has some maximum.

↳ etc.

Pruning → to reduce the likelihood of overfitting of decision tree in CART

↳ Post pruning (or just Pruning)

↳ after building tree

↳ by minimum error

↳ pruned back to a point where cross-validation error is minimum.

↳ by smallest tree

↳ tree pruned back slightly ~~further~~ further than the minimum error.

↳ 1 SD of minimum error

↳ more intelligible smaller tree at the expense of small increase of error.

Early stoping (Pre-pruning)

↳ at each stage of tree-splitting, cross-validation error checked

↳ if error doesn't decrease significantly enough ⇒ we stop

↳ it may underfit as it may not allow to have subsequent splits where the error ~~d~~reduce significantly

↳ Ensemble Learning → meta approach to ML which seeks better predictive performance by combining the predictions from multiple models.

↳ Bagging (Similar models)

↳ Stacking (different models)

↳ Boosting (similar models)

Bagging (Bootstrap AGGregatING) → many samples generated from the given data by bootstrapping (drawing samples with replacement)

↳ Simple voting or averaging of the outcomes

(in case of decision trees)

↳ Smaller bootstrap samples helps in case of large datasets.

↳ difficult to visualize individual trees.

↳ The decision trees are allowed to grow deep and are unpruned

⇒ low bias & high variance

↳ treated by taking multiple decision trees.

Random Forest

↳ one kind of bagging of decision trees (DT) in which the correlation between the individual DTs is reduced by restricting the no. of attributes taken into consideration for node splitting

↳ if m total attributes present

↳ $p = \sqrt{m}$ (classification) } default no. of random attributes that can be taken for splitting

↳ $p = m/3$ (regression) }

↳ leads to lower correlations ⇓ improvement in accuracy

↳ DT ⇒ handling of missing values is easy

↳ high comp. time.

Boosting

- A method in which the training dataset for each subsiquent model is increasingly focussed on instances misclassified by previous model.
- Use of very simple DTs (called weak learners) in case of DTs as models
- Combine prediction using a weighted average of models

(1) Ada Boost
(2) Gradient Boosted Trees
(3) XGBoost

Ada Boost (Adaptive boost)

- initial weights → $w_i = \frac{1}{N}$ to all samples $\{\bar{x}_i, y_i\}$
- for m = 1-M
  - fit a model $G_m(x)$ to the training data
  - $err_m = \frac{\sum_i^N w_i \, I(y_i \neq G_m(\bar{x}_i))}{\sum_i^N w_i}$ → gives weights of the wrong predictions
    - $I(\text{True}) = 1$
    - $I(\text{False}) = 0$
  - $\alpha_m = \frac{1}{2} \log\left[\frac{(1 - err_m)}{err_m}\right]$
  - new weights → $w_i = w_i \exp[\pm \alpha]$
    - + → incorrect classified
    - − → correct classified
  - normalize weights
  - Draw new sample by weighted random draw and ascribe equal initial weights
- Combine the predictions using a weighted average of models.

Gradient boosting

- updating the decision trees based on minimization of the loss function $\mathcal{L}(y_i, F(x_i))$
- Steepest descent approach → updated model → $F_{m+1}(x_i) = F_m(x_i) - \gamma \left(\frac{\partial \mathcal{L}}{\partial F_m(x_i)}\right)$
  $= F_m(x_i) + \gamma h_m(x_i)$
- different loss functions for different models
  - least square regressor → $h_m(x_i) \propto \frac{\partial \mathcal{L}}{\partial F(x_i)} \left[\frac{1}{n} \sum (y_i - F(x_i))^2\right] \propto (y_i - F_m(x_i))$
    - residual → $\boxed{h_m(x_i) = y_i - F_m(x_i)}$
- Algorithm →
  (1) get the base model, $F_0(x_i) = avg(y_i)$
  (2) fit decision tree to the residuals (ie find a model for residuals)
  $h_0(x_i) = y_i - avg(y_i)$ → DT-1
  (γ → learning rate)
  (3) get the new values → $F_1(x_i) = F_0(x_i) + \gamma h_0(x_i)$
  (4) get the next updated model by finding a new model for new residuals
  $h_1(x_i) = y_i - F_1(x_i)$ → DT-2
  $F_2(x_i) = F_0(x_i) + \gamma [h_0(x_i) + h_1(x_i)]$

Generalization

(till M-step) → $\boxed{h_{m+1}(x_i) = y_i - F_m(x_i)}$ $\quad \boxed{F_M(x_i) = F_0(x_i) + \gamma \sum_{m=1}^{M-1} h_m(x_i)}$

↳ XGBoosting (Xtreme Gradient Boosting)

- very powerful model
  - → high speed and high performance
- developed by Tianqi Chen

↳ Steepest descent approach to minimize the objective function :→

$$Obj_t(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} w(f_i)$$ → at t^{th} iteration

- $l(y_i, \hat{y}_i^{(t)})$: loss function (eg- RMSE, log logistic error)
- $w(f_i)$: regularization term

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i)$$

$$w(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

- γT : Control no. of leaves
- $\frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$: Controls the score

here, T → no. of leaves

↳ The $f_{k=(t+1)}$ tree is made such that it minimizes the $Obj_t(\theta)$

↳ building the $f_{k=(t+1)}$ tree :→

↳ not built based on traditional methods like gini index or entropy

↳ taylor expansion of loss function (upto 2nd order) :→

$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right]$$

gradient ← $g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$

hessian ← $h_i = \frac{\partial^2}{\partial \hat{y}_i^{(t-1)2}} l(y_i, \hat{y}_i^{(t-1)})$

def :→ divide the θ function leaf wise

So, for j^{th} leaf :→

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

I_j = indices of the data points in the leaf j

↳ Similarity at one leaf :→ $\frac{G_j^2}{H_j + \lambda}$

↳ Algo :→

(i) Initialize the tree with one leaf

(ii) leaf split based on gain (maximize the gain)

$$\text{gain} = \frac{1}{2}\left[\underbrace{\frac{G_L^2}{H_L + \lambda}}_{\text{score of new left leaf}} + \underbrace{\frac{G_R^2}{H_R + \lambda}}_{\text{score of new right leaf}} - \underbrace{\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}}_{\text{score of original leaf}} \right] - \underbrace{\gamma}_{\text{regularization term}}$$

↳ Pruning the tree :→ if gain $< \gamma$ ⇒ do not split the leaf

(iii) Stoping criterion

K-NN (k- Nearest Neighbours) → distance based lazy learner

- doesnot learn from data rather each test set is compared with the whole training data to prediction
- the model size increases with training set size

algorithm:-
- select no. of neighbors (k) to be considered → usually odd to resolve conflicts
  - is a hyper-parameter
- Calculate the "distance" of the test point with all the train points
- based on the y_i of the k-NN assign the y_i of the test point (voting or avg)

Distance measures
- L2 norm → Euclidean distance :- $D = \text{sqrt}\left[\sum_i (x_i^{(1)} - x_i^{(2)})^2\right]$
- L1 norm → Manhattan distance :- $D = \text{Sum}\left[\sum_i |x_i^{(1)} - x_i^{(2)}|\right]$ → for 2D $D = |x^{(1)} - x^{(2)}| + |y^{(1)} - y^{(2)}|$
- L_p norm → Minkowski distance :- $D = \left[\sum_i |x_i^{(1)} - x_i^{(2)}|^p\right]^{1/p}$
- Hamming distance :- (strings of same lengths) $\Rightarrow D = \sum_{i=1}^{L} I(x_i^{(1)} \neq x_i^{(2)})$
  - L = length of the strings
  - I(True) = 1
  - I(False) = 0

Optimization of K-NN → partitioning of the feature space so that less number of comparisions needs to be done at testing

Box-shaped partitioning (K-Dimensional trees)
- binary tree structure
  - split data recursively in half on exactly one feature
  - rotate through features (pick features with max variance)

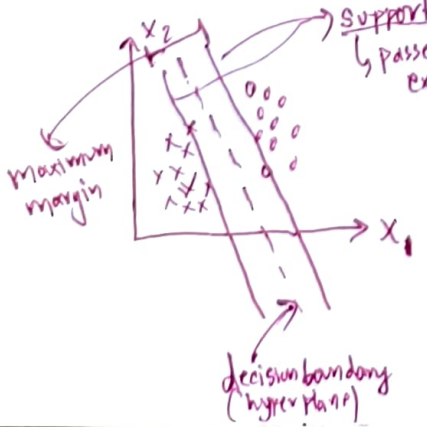
Hyper-sphere partitioning (ball-tree)
- if distance to a ball > currently closest neighbour safely ignore the ball and all points within
- allows to partition the data along an underlying manifold that points are on, rather than repeated dissection of features

Pros :- Exact, easy to build

Cons :- Curse of dimensionality
- ball tree shower the K-D in low dimensions but a lot faster in higher dimensions.

Support Vector Machine (SVM)

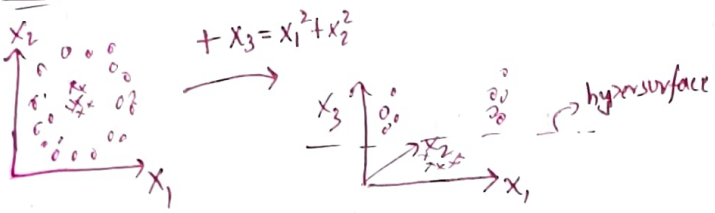
- to create decision boundary (hyperplanes) separating classes.
- Can also be used for regression.



support vectors
- passes through the extreme points of either class

- SVM algorithm tries to find the hyperplane with highest margin
- The class of the test data is determined by it's proximity to either of the support vectors.

↳ SVM Kernel trick (non-linear SVM) → when the data points are separated by non-linear hyper-surface.



↳ Add a new feature x_{n+1} with the equation of the non-linear hypersurface
↳ Can then separate the data using hyperplane and support vectors.

↳ One vs One (OvO) and One vs Rest (OvR) for multiclass classification

↳ One vs One (OvO) → each class vs every other class
eg.→ red vs blue,
red vs green,
blue vs green

↳ One vs Rest (OvR) → each class vs rest of classes
eg: red vs [blue, green]
blue vs [red, green]
green vs [red, blue]

↳ binary classification algorithms can be used for multiclass classification.
↳ Multiclass problem - splited into many binary class problem.

↳ Semi-supervised Learning → for supervised learning: dataset needs to be hand labelled
→ for unsupervised learning: application spectrum is limited

↳ Algorithm is trained upon a combination of labelled and unlabelled data
↳ First use clustering to clusted similar data and then use the existing labelled data to label the rest of the unlabelled data.
↳ Assumptions: → Continuity: closer points ⇒ same output label
Cluster assumption: → data can be divided into discrete clusters having same labels.
manifold assumption: → data lie approx on a manifold of much lower dimension than input space so that distances and densities can be defined.

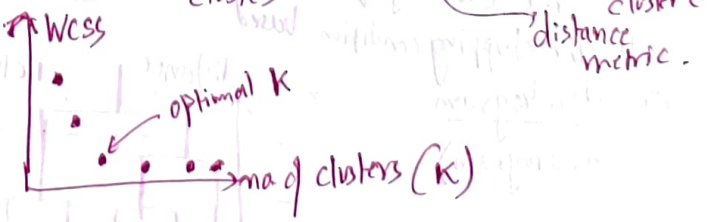
→ K-means Clustering → K determines the number of clusters to be made
→ centroid based algorithm where each cluster is associated with a centroid → (average of all the points)

Algorithm
↳ Select K random points in the input space as centroids
↳ assign each data point to their nearest K-point to form K-clusters
↳ Recalculate the position of K-points as the centroid of the points in clusters.
↳ End the loop when no more reassignment of points occur.

↳ Choosing the right K → Elbow method
↳ for each value of K calculate the WCSS (within Cluster sum of Squares)

$$WCSS = \sum_{\text{Clusters}} \sum_{\text{in cluster}} \| P_i, C_{\text{cluster}} \|^2$$

C_{cluster} → centroid of cluster c
$\| \cdot \|$ → distance metric.



→ K means ++ algorithm → Smart initialization algorithm
- random first centroid from data point
- for each data point compute its distance from the nearest chosen centroids (ip till now)
- assign the next centroid based on a distance based probability from the nearest previously chosen centroid ($P(\text{next centroid}) \propto ||\text{last centroid} - \text{new centroid}||$)
- Stop :- when K-centroids are initialized.

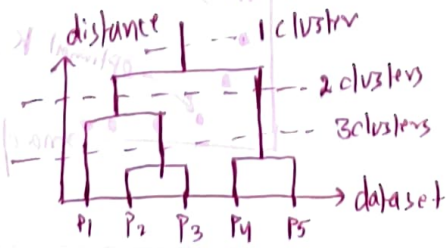
Drawback of k-means (Susceptible to initialization of centroids)
- if far-off point initialized ⇒ might end-up with no points in cluster and more than one cluster linked to same centroid
- if centroids too close ⇒ same cluster having >1 centroids

→ Mini-batch k-means clustering → high speed performance, good for large volume of data
- to use small random samples of fixed in size data,
- each time a new random sample is taken to update the cluster until the convergence
  - clusters are updated based on a learning rate which decreases as the no. of iterations increases.

→ Agglomerative vs Divisive clustering (top-down)
- (bottom-up)
- each observation starts in it's own cluster and pairs of clusters are merged
  - eg:- Hierarchical clustering
- → all observation start in one cluster, and splits are performed recursively
  - eg:- k-means

⇒ Hierarchical clustering analysis (HCA) → don't need pre-determined number of clusters.

Algorithm
- take every point as a cluster
- club the two nearest clusters together to reduce the no. of clusters by one
- repeat the clubbing of clusters until the stopping condition based on dendogram (dendrogram)



Linkage methods to measure the distance b/w clusters
- Single linkage :- distance b/w closest points considered
- Complete linkage :- distance b/w farthest points considered
- Average linkage :- distance b/w each pair of datasets added up and then divided by total number of data sets. one of most popular
- Centroid linkage :- distance b/w centroids considered.

→ DBSCAN (density based spatial clustering of applications with noise)

↳ drawbacks of K-means and heirarchial clustering :- (i) only good for convex clusters
(ii) can't work with noise as they are highly susceptible to them

↳ DBSCAN can resque

↳ Two parameters :-

↳ epsilon (ϵ or eps) :- defines the neighborhood around a data point, two points considered neighbors if their distance $< \epsilon$
↳ if ϵ too small ⇒ large no. of outliers // ϵ too large ⇒ clusters will merge

↳ Minimum points (MinPts) :- minimum no. of data points within eps radius.
↳ larger dataset ⇒ larger MinPts ; general rule MinPts $\geq D+1$ (D = dimension of dataset)

↳ Types of data points

↳ Core-point
if it has $\geq$ MinPts within its ep radius

↳ border point
if it has $<$ MinPts within its ep radius but is in a neighborhood of Core-point

↳ Noise/outlier
$<$ MinPts in its ep radius and not in any neighborhood of core points.

↳ Algorithm

↳ get all the neighbor points for each point and identify the core-points
↳ if a core-point not assigned to any cluster, assign it to a new one
↳ find all its densely connected points and assign them to the same cluster.
↳ two points a and b are densely connected if $\exists$ a point c which is a core point and is in the neighborhood of both a and b

Stop :- when visited all the points. Those points that don't belong to any cluster are noise.

★ Check Scikit learn's page on clustering for comparision on different clustering algorithms ⇒ DBSCAN performs best both in terms of making clusters and time complexity.

↳ Performance Metrics for clustering

↳ Silhouette Score $= \frac{(n-i)}{\max(i,n)}$ // n :- avg (dist b/w each sample and its nearest cluster Cinter-cluster distance)
i :- avg (intra-cluster distance)

only applicable to convex clusters

↳ near +1 ⇒ cluster's samples further away from nearby clusters (good clustering)
↳ near 0 ⇒ sample very close to decision boundary (ambiguous clustering)
↳ <0 ⇒ samples assigned to wrong cluster.

↳ When ground-truth clusters labels are present (gold-standards)

P → partition created by our clustering algorithm // P' → gold-standard partition of clusters

(true positive) SS: → no. of pairs which are in same clusters in both P and P'

(false positive) SD: → no. of pairs which are in same cluster in P and in different clusters in P'

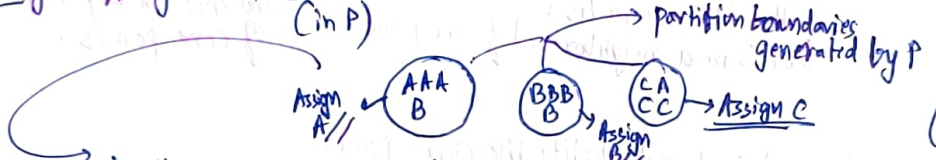
(False negative) DS: → no. of pairs which are in different clusters in P and in same cluster in P

(True negative) DD: → no of pairs which are in different clusters in both P and P'

↳ Rand index :→ $R = \frac{SS + DD}{SS + DD + DS + SD}$

↳ Jaccard index : → $J = \frac{SS}{SS + DS + SD}$

↳ Purity :→ Assign each cluster with the class which comes the max time when compared with P'



purity = $\frac{\#\text{ of correct class labels}}{\text{total points}} = \frac{3+4+3}{4+4+4} = \frac{10}{12}$

↳ Principal Component Analysis → Converts observations of correlated features into a set of linearly uncorrelated features (Principal Components)

↳ used for dimensionality reduction

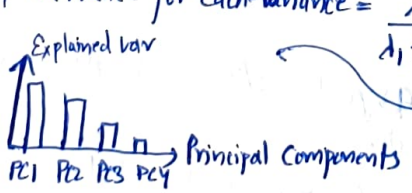
Step-1 :→ standardization :→ $\tilde{x}_i = \frac{\text{Value} - \text{mean}}{\text{Stdev}}$ { to compare the variances of all attributes }

Step-2 :→ Covariance matrix computation :→ $\begin{bmatrix} Cov(X_1,X_1) & Cov(X_1,X_2) & \cdots \\ Cov(X_2,X_1) & Cov(X_2,X_2) & \end{bmatrix}$

Step-3 :→ Get eigenvalues and eigenvectors of the covariance matrix

↳ explained variance for each-variance = $\frac{\lambda_i}{d_1 + d_2 \cdots d_n}$

(Scree plot)



→ the greater the explained variance, the more information the PC contents.

Step-4 :→ get the new features

↳ multiply P^* matrix ($[PC_1, PC_2, \cdots]$) to Z (original dataset) to get Z^*

↳ each column of Z^* is independent of one another

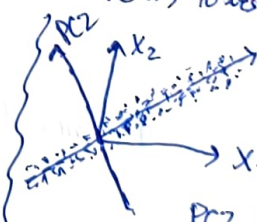
Step-5 → remove the less important vectors

must be linear combination of original features

must be orthogonal

PC1 → highest importance

↓ PCn → lowest importance.



PC2 can be dropped here

PC1 → with highest information ∈ with highest variance