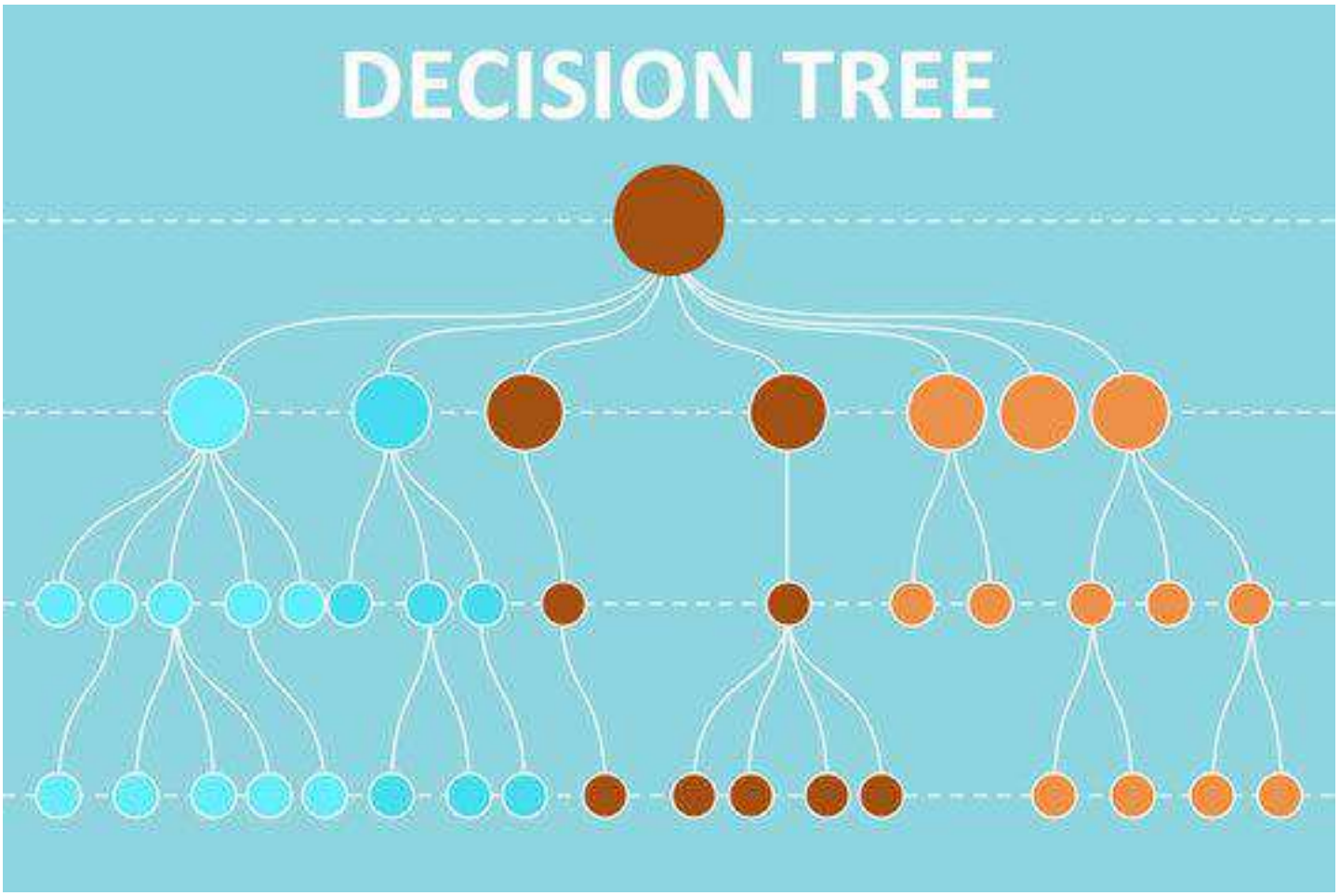


Table Of Contents

scaling is not needed for DT

(3) Decision Tree

for both Regression & classification (supervised learning)

→ As of now we have Regression models, which makes a best fit (line) or a best fit (polynomial)

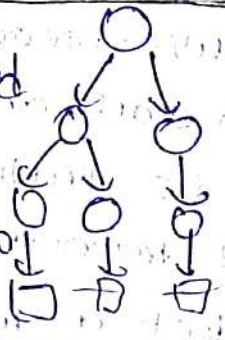
→ But in case of classification models. we have this logistic regression which can only classify by using lines.

* what if our data is classified in polynomial fig

} Logistic Regression can't be used here drawback

So, this is where "decision tree" came to picture

→ As the name suggests, this algorithm works by dividing the whole dataset into a "tree-like structure" based on some rules & conditions & then gives predictions based on those conditions. Let's see how it works.



1 → First it select a 'root node' based on given cond.

2 → then the root node will split into child nodes based on cond.

3 → If any node doesn't fulfill its cond, then it again split into new cond.

4 → continues till at the cond. are met or if you have pre-defined the depth of your tree

But, how do we select a root node?

~ this is where mathematical intuition comes in

→ usually Regression trees are used for quantitative data. Incase of qualitative/categorical data we use classification trees.

→ In Regression tree, we split the nodes based on RSS (Residual Sum of Squares) criteria.

In classification, it is done using 'classification error rate', 'Gini Impurity' & 'entropy'.

we need a pure element (root node) to build tree. so to find that we can use entropy.

(*) Entropy is the measure of randomness of data. It gives the impurity present in the dataset.

→ when we split our node into two regions and put different observations in both the regions, the main goal is to reduce the randomness in the region & divide our data cleanly than it was in the previous node. If splitting the node doesn't lead to entropy reduction, we try to split based on different cond, or we stop.

→ A region is clean (low entropy) when it contains data with the same labels & random if there is a mixture of labels present (high entropy).

we need to calculate entropy of each column.

$E = -P \cdot \log(P)$ P-Probability

$E = -P_1 \log P_1 - Q_1 \log_2 Q_1$

④ Information gain (Gn)

calculates the decrease in entropy after splitting a node. It is the difference b/w entropies before & after the split. The more the information gain, the more entropy is removed, & more clean the node.

$$G(A) = 1 - \sum \left(\frac{S_v}{S} \cdot E_i \right)$$

$$\text{Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

Ex) Let's take the data,

x	y	z	class
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

If we introduce new data,

x y z

1 0 1 , what is class?

Soln First we need individual column entropy.

(Ex, Ey, Ez)

In each column we have (1, 0) & total 2 class (I, II)

$$Gx(1) = \left(\frac{-2}{3} \cdot \log \frac{2}{3} \right) + \left(\frac{-1}{3} \cdot \log \frac{1}{3} \right) = -0.276$$

(I) (II)

↓

we have '3' 1's in x column, out of which '2' are I class

'1' 1 out of 3 is II class.

I

$$E_x(0) = \left(\frac{0}{-1}\cdot\log\left(\frac{0}{-1}\right)\right) + \left(\frac{-1}{1}\cdot\log 1/1\right) = 0$$

$$E_y(1) = \left(\frac{-2}{2}\cdot\log 2/2\right) + \left(\frac{0}{-2}\cdot\log\frac{0}{-2}\right) = 0$$

$$E_y(0) = \left(\frac{0}{-2}\cdot\log\frac{0}{-2}\right) + \left(\frac{-2}{2}\cdot\log 2/2\right) = 0$$

$$E_z(1) = \left(\frac{-1}{2}\cdot\log 1/2\right) + \left(-1/2\cdot\log 1/2\right) = 1$$

$$E_z(0) = \left(\frac{-1}{2}\cdot\log 1/2\right) + \left(-1/2\cdot\log 1/2\right) = 1.$$

Information Gain

$$G_x = 1 - \left[\frac{S(1)}{S}\cdot E(1) + \frac{S(0)}{S}\cdot E(0)\right]$$

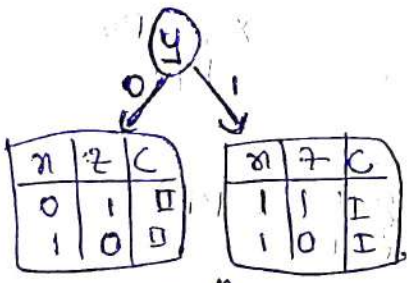
(S(1): no. of 1's; S(0): no. of 0's; S: total no. of values in x)

$$G_x = 1 - \left[\frac{3}{4}\times 0.276 + \frac{1}{4}\times 0\right] \cong 0.7$$

$$G_y = 1 - \left[\frac{2}{4}\times 0 + \frac{2}{4}\times 0\right] = 1$$

$$G_z = 1 - \left[\frac{2}{4}\times 1 + \frac{2}{4}\times 1\right] = 0$$

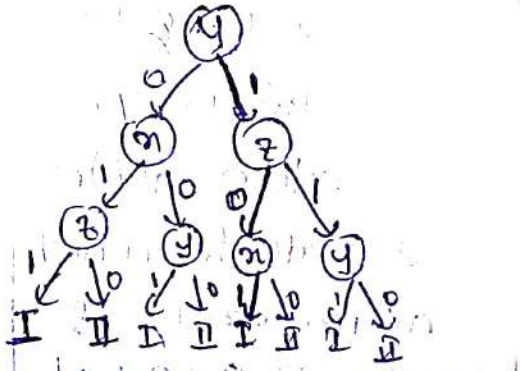
$G_y = 1$, is the highest info gain so local entropy. Hence it is 'Root Node: (y)'



Again we have to find E_x, E_z, G_x, G_z. And largest is parent node. Then leaf node gives a class.

Find E_x, E_z, G_x, G_z.

Finally it can be like this



So, 1-0-1 → I class

Gini Impurity

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset.

→ It is calculated by multiplying the probability that a given observation is classified into the correct class & sum of all the probabilities when that particular observ. is classified into the wrong class.

→ Let's suppose there are k no. of classes and an observation belongs to the class 'i', & it's probability is given by 'p_i'.

Then probability that observation belongs to any other class other than 'i' is,

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

then gini impurity $= \sum_{i=1}^{} p_i \times \sum_{k \neq i} p_k$

Gini impurity value will be b/w 0 & '1',
0 - no impurity
1 - Random distr.
The node for which Gini impurity is least is selected as Root node to split.

$$Gini = \sum_{i=1}^{J} p_i (1 - p_i)$$

$$Gini = \sum_{i=1}^{J} p_i - p_i^2$$

$$Gini = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2$$

$$Gini = 1 - \sum_{i=1}^{J} p_i^2$$

6) Given data,

class	gender	stay in hostel
9	M	Yes
10	F	No
8	F	Yes
8	F	No
9	M	Yes
10	M	No
11	F	Yes
11	M	Yes
8	F	Yes
9	M	No
11	M	No
11	M	Yes
10	F	No
10	M	Yes

Here understand how root node is selected by calc gini impurity.

We have 2 features which can use for hostel class & gender we will calculate gini for each of the features & then select that feature which has least gini impurity

Sol:

(1) class

class	stay	Total	P(Yes)	P(No)
8	Yes=2, No=1	3	2/3	1/3
9	Yes=2, No=1	3	2/3	1/3
10	Yes=1, No=3	4	1/4	3/4
11	Yes=3, No=1	4	3/4	1/4
		14		

Gini for class feature

$$G(class=8) = 1-(P(Yes))^2-(P(No))^2 = 1-(2/3)^2-(1/3)^2 = 4/9$$

$$G(class=9) = 1-(2/3)^2-(1/3)^2 = 4/9$$

$$G(class=10) = 1-(1/4)^2-(3/4)^2 = 6/16$$

$$G(class=11) = 1-(1/4)^2-(3/4)^2 = 6/16$$

Weighted sum of gini for class

$$G(class) = \frac{\text{no of class 8}}{\text{total}} * G(8) + \frac{\text{no. of 9}}{\text{total}} G(9) + \cdots$$

$$G(class) = \frac{3}{14} \times \frac{4}{9} + \frac{3}{14} \times \frac{4}{9} + \frac{4}{14} \times \frac{6}{16} + \frac{4}{14} \times \frac{6}{16}$$

$$G(class) = 0.404$$

(ii) Gini for gender

Gender	Stay	n	P(Y)	P(N)
M	Yes=5, No=3	8	5/8	3/8
F	Yes=3, No=3	6	1/2	1/2
		14		

$$G(M) = 1 - (5/8)^2 - (3/8)^2 = 0.469$$

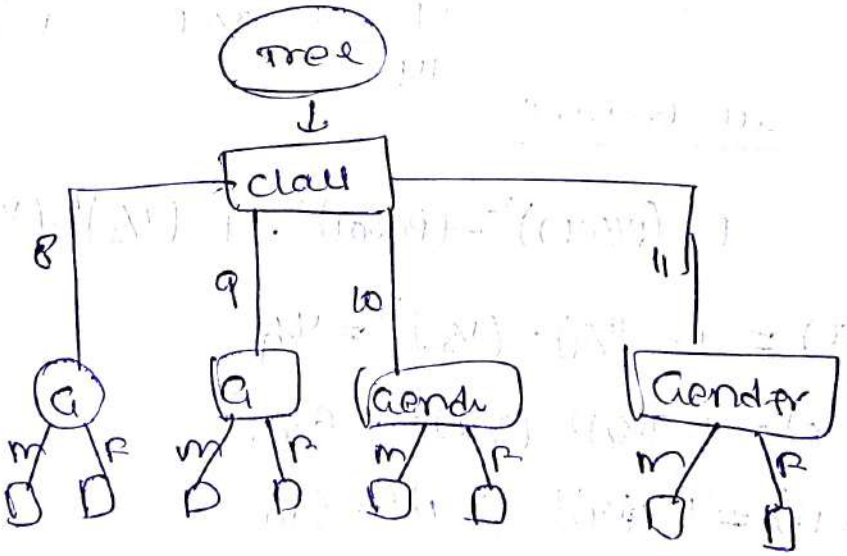
$$G(F) = 1 - (1/2)^2 - (1/2)^2 = 0.5$$

$$G(\text{Gender}) = \frac{8}{14} \times 0.469 + \frac{6}{14} \times 0.5$$

$$\boxed{G(\text{Gender}) = 0.4822}$$

we can see,

$$\boxed{G(\text{class}) < G(\text{Gender})}$$ thus, root node is class.



this is how decision tree node is selected by calculating gini impurity for each node individually. If two or more features, then we need to repeat same steps after selecting each nodes.

Note:-

① we should control growth of tree by 'min_sample_split' ~~so which way we will not~~, because a infinitely grown tree with depth (>100) also. there's a high chance at time of Prediction. it may go "overfitting"

Decision Tree for Regression:-

→ when performing regression with a decision tree, we try to divide the given values of x into distinct & non-overlapping regions.

Eg:- For a set of possible values $x_1, x_2, \ldots x_p$ we will try to divide them into 'J' distinct & non-overlapping regions $R_1, R_2, \ldots R_J$. For a given observation falling into the region "R_j", the Prediction is equal to the mean of the response value for each training observation in the Region.

(*) Regions are selected in a way to reduce residual sum.

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

$\hat{y}_{R_j}$ - mean of all the response vars. in the region 'j'.

Disadv of DT:-

① Small change in data can cause instability in the model because of greedy approach.	② Proba of overfitting is very high for DT	③ takes more time to train a DT model.

Recursive Binary splitting (Greedy Approach) :-

→ As mentioned, we try to divide the x values into J regions, but is very expensive in terms of computational time to try to fit every set of x values into J regions.

→ Thus decision tree opts for a greedy approach in which nodes are divided into regions based on the given condition is called **greedy**, because it does the best split at a given step at that point of time **rather than** looking for splitting a step for a better tree in upcoming steps.

→ It decided threshold to divide others into different regions.

$$\sum_{x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

this process have a high chance of overfitting the training data as it is very complex

Tree Pruning :-

→ It is the method of trimming down a full tree to reduce the complexity & variance in the data. Just as we regularized linear regression, we can also regularize the decision tree model by adding a new term.

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{Rm})^2 + \alpha |T|$$

where,

'T' is the subtree which is a subset of full tree T_0.

'α' is non-negative tuning parameter, which penalizes the MSE with an increase in tree len

Post-Pruning :- (Backward pruning).

→ It is the process when DT is generated first & then the non-significant branches are removed. cross-validation set of data is used to check the effect pruning & tells whether expanding a node will make an improvement or not. If node is having improvement we continue else convert to leaf node.

Pre-Pruning :- (Forward Pruning)

→ It stops the non-significant branches from generating. It uses a condition to decide when should it terminate splitting of some of the branches prematurely as tree is generated,

* different algorithms for Decision tree

① ID3 (Iterative Dichotomiser) :-

It is one of the algorithms used to construct decision tree for classification. It uses information gain as the criteria for finding the root node and splitting them.

It only accepts categorical attributes

② C4.5 :-

It is an extension of ID3 algorithm, & better than ID3 as it deals with both

continuous & discrete values.

Also used for classification purposes.

③ classification & Regression Algo (CART) :- (Baucalli, Decision ma

It is the most popular Algo used for constructing decision trees. It uses 'gini' impurity as the default calculation for selecting root node, however one can use "entropy" for criteria as well. Both for regression & classification problem.

Gini is easier to calculate, Entropy complicated.

which is why CART uses 'Gini'

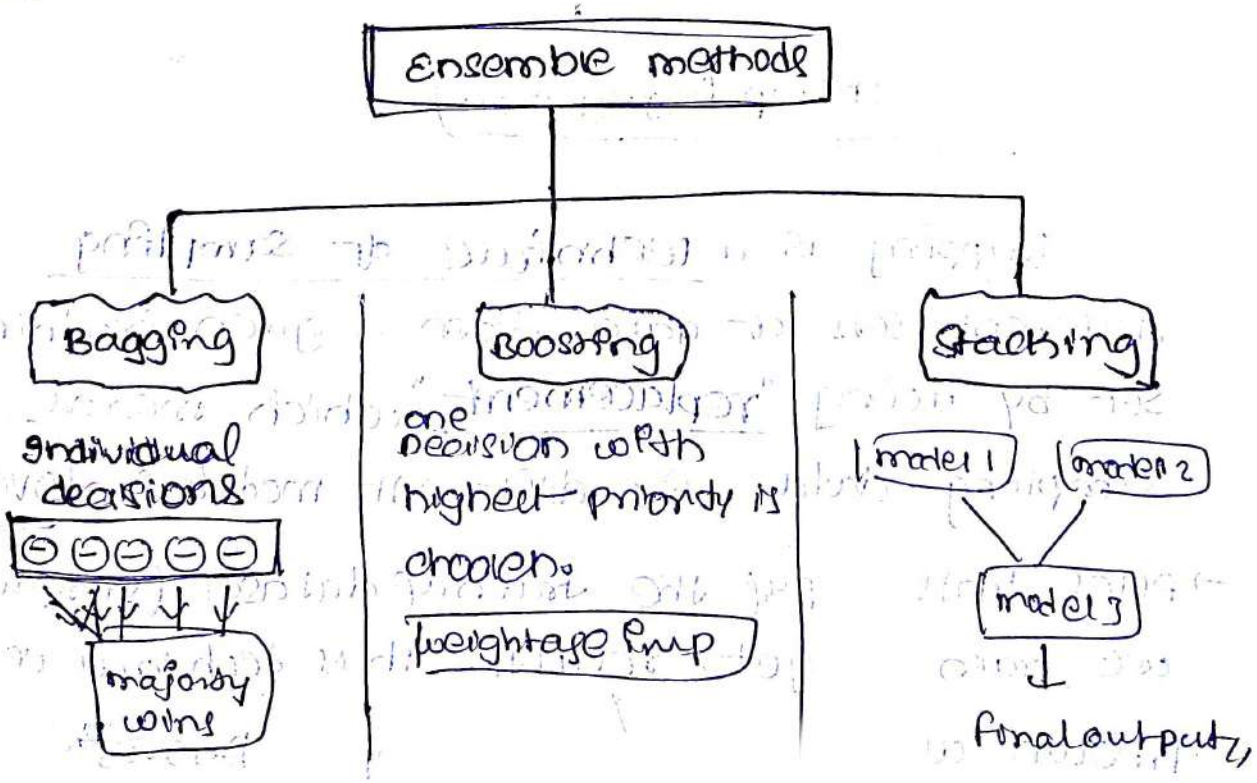
multiple decision makers

(*) Ensemble Techniques & Random Forest

→ until we are predicting using one model, but what if we can take models and predict the output, won't it be better than one. Of course it will be. this is the idea behind Ensemble

→ we regularly came across the option of "audience poll" and constant, mostly with when goes for option which had highest vote from audience. So, we can say, taking opinions from a majority of people is much more preferred than option of single person

→ Ensemble techniques has a similar underlying idea where we aggregate group of predictions from a group of Predictors. such algorithms are called "Ensemble methods" and such predictors are called "Ensembles".



(*) Let's suppose we have 'n' predictors (models).

→ $z_1, z_2, \ldots z_n$ with a standard deviation of σ

$$var(z) = \sigma^2$$ — var of single predictor.

$$Avg(u) = \frac{(z_1 + z_2 + \ldots z_n)}{n}$$ — Average of Predictors (Expected val)

If we use 'n' as predictor then expected val still remains the same, but variance is reduced so much.

$$var(u) = \frac{\sigma^2}{n}$$

This is why taking mean is preffered over single Predictor

(*) So Ensemble methods take multiple small models and combine their predictions to obtain a more powerful predictive power.

Bagging (Bootstrap Aggregation)

→ Bootstrapping is a technique of sampling different sets of data from a given training set by using "replacement", which means sampling data for different models can overlap

→ After bootstrapping the training dataset (sampling) we train & get results. This technique is known as Bagging / Bootstrapping Aggregation

→ Bagging is a type of ensemble technique in which a single training algorithm is used on a different subset of training data where sampling is done with replacement (bootstrap).

→ In case of Regression, Bagging prediction is simply the mean of all the predictions & in case of classifier, Bagging prediction is the most frequent prediction (majority vote).

④ Bagging is also known as "parallel mode" since we run all models parallely and combine results at the end.

Advantages

① Bagging significantly decreases variance without a bias.

② It works so well bcoz of diversity in training data.

③ works well with smaller dataset.

④ If training set is very huge, it can save computational time by training model on relatively smaller dataset & still can increase the accuracy of model.

Disadvantage

→ It improves the accuracy of model on the expense of "interpretability".

we can't interpret as there will be so many models.

Pasting - similar to bagging but no replacement in sampling this called less diversity in the sampled datasets and data ends up being correlated. thats why Bagging is more preferable than pasting.

Boosting

→ Boosting is an ensemble technique (involves several trees) that starts from a weaker decision & keeps on building the model such that the final prediction is the weighted sum of all the weaker decision makers.

> the weights are assigned based on performance of individual tree.

→ on boosting, ensemble parameters are calculated in "stagewise way" which means that while calculating the subsequent weight, the learning from previous tree is considered as well.

> i.e) the learning of previous tree boosts the learning of next tree and goes on — so called Boosting

→ we mostly used decision tree as weak classifier. Any other algorithm can be used as a base, but reasons for choosing trees are :-

Pros	cons
1) Robust to outliers	1) Inability to extract a linear comb of features
2) Feature Scaling not required	2) High variance leading to small computational power.
3) Handles missing values	
4) can deal with irrelevant input	
5) computational scalability	

→ Boosting minimizes the variance by taking into consideration the results from various trees.

General understanding eg of boosting

① you wanna travel to different place.

→ And you know a friend who is a traveller, so you ask him about the place (given your info) he will tell what he know & he will ask his friends.

→ He asks his friend who visited there to give some phone no or any agency.

→ And that friend asks another friend who lives there.

First, we have to boost other person with info we have, and he will boost next person with info he have & finally get result

② Dad wants to buy car.

→ He asks you & without any knowledge you say 'yes'. (less weightage)

→ Grandpa thinks and says what model should be bought considering grandchildren who wants car. (good weightage)

→ considers son & grandpa, mom says what to buy, new or second hand, based on financial status (Better weightage)

Stacking (Stacked Generalization)

→ Stacking is a type of ensemble technique which combines the predictions of 2 or more model, also called base models and use the combination as the input for a new model (meta model)
ie,

> meta model - A new model is trained on the predictions of base models

→ Suppose you have a classification problem & you can use several models like LogisticReg, SVM, KNN, Random forest etc.
the idea is to use few models like KNN, SVM as the base model & make predictions using that

→ Now predictions made by these models are used as an input feature for Random forest to train on & give prediction.

→ Stacking can be multi level;
using base model as level 1 the many predictions into another sub-base models at level 2 & so on

> then at last any meta-model which take predictions of last sub-base models as input & deal predictions

working

① Split the dataset into a training set & a holdout set. Generally we do a 50-50 split of training & holdout. training set = x_1, y_1 ; Holdout set = x_2, y_2 (validation set)

② Split the training set again into training & test sets. like, $x_1\_train, y_1\_train, x_1\_test, y_1\_test$

③ train all the base models on training set x_1_train y_1_train

④ After training is done, get the predictions of all the base models on the "validation set x_2"

⑤ Stack all the predictions by diff models together as it will be used as input feature for the meta model.

⑥ Again, get the predictions for all the base models on the test set x_1_test, and stack all their predictions together as it will be used as the

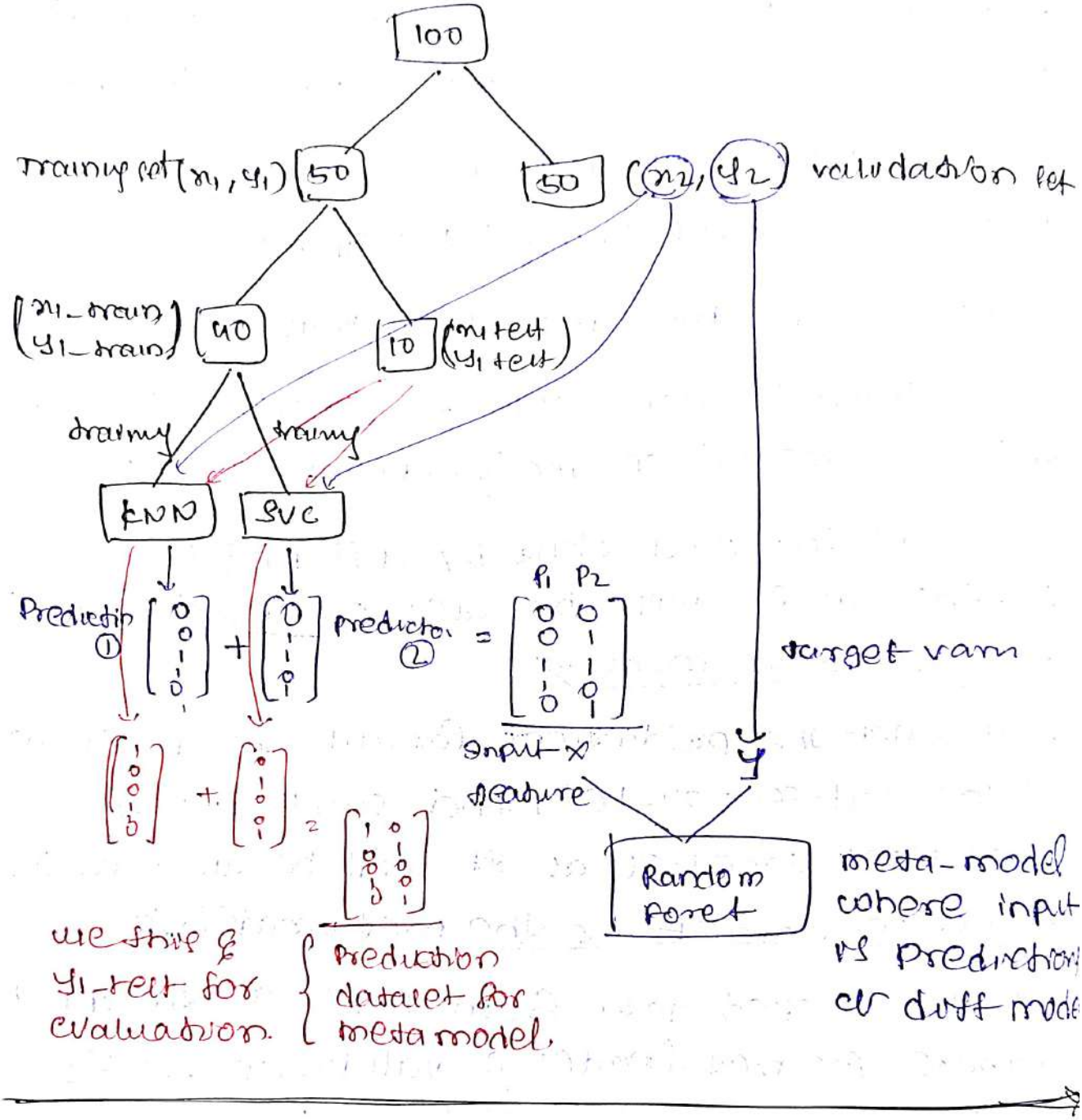
⑦ Prediction dataset for the metamodel.

⑧ use the stacked data from step 5 as the input feature for metamodel & validation set y_2 as the target variable & train the model on these data.

⑨ once the training is done check the accuracy of metamodel by using data from step 7 for prediction & y_1_test for evaluation.

⑩ End.

visual interpretability



Out-of Bag evaluation:

→ In Bagging, there might be some data which are never sampled at all. The remaining data which are not sampled are called out-of bag instances.

→ Since model never trains over these datas, they can be used for evaluating the accuracy of model by using these data for prediction.

---

(4) Random Forest:

→ Decision trees have low bias but high variance. And we know bagging technique is a very good solution for decreasing the variance for a decision tree.

→ Instead of using a bagging model with underlying model as a decision tree, we can also use Random Forest which is more convenient & well optimized for decision trees.

→ The main issue of bagging is that there is not much independence among sampled datasets (there is correlation).

→ The adv. of RF over bagging is that RF makes a tweak in working algorithm of bagging model to decrease the correlation in trees. The idea is to introduce more randomness while creating trees which will help in reducing correlation.

working of RF model

① different samples are selected from training dataset using bootstrapping.

② on each sample we train our tree model & we allow the trees to grow depth.

③ once the trees all formed, prediction is made by the random forest by aggregating the predictions of all the model. for regression model, the mean of all predictions is the final & for classification mode, the mode of all the predictions is called the final.

adv & disadv of Random Forest

① it can be used for both regrel & classfr problems.

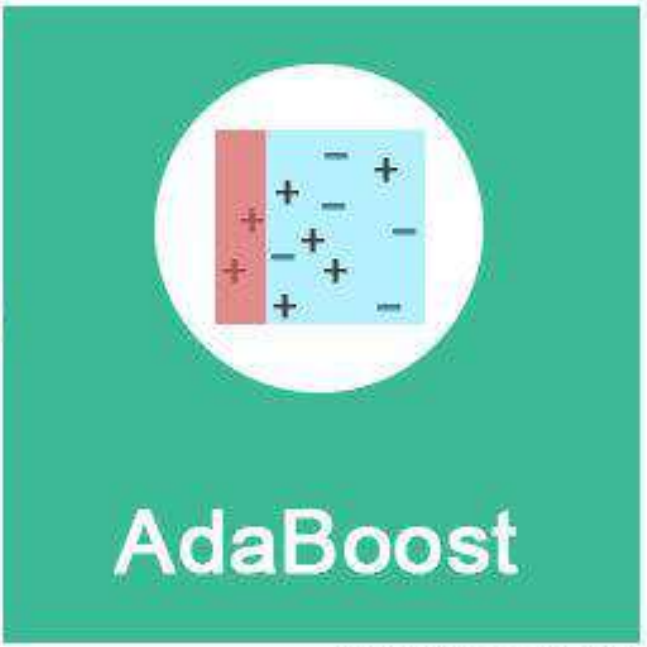
② since base model is a tree, handling of missing values is easy.

③ it gives very accurate result with very low variance.

④ results of a random forest are hard to interpret in comparison with DT.

⑤ high computational time than other respective models.

⑥ should be used when accuracy is optimal priority,



www.educba.com

Table of Contents

Boosting

→ Boosting is an ensemble technique (involves several trees) that starts from a weaker decision & keeps on building the model such that the final prediction is the weighted sum of all the weaker decision makers.

> the weights are assigned based on performance of individual tree.

→ on boosting, ensemble parameters are calculated in "stagewise way" which means that while calculating the subsequent weight, the learning from previous tree is considered as well.

> i.e) the learning of previous tree boosts the learning of next tree and goes on.. — so called Boosting

→ we mostly used decision tree as weak classifier. Any other algorithm can be used as a base, but reason for choosing tree are :-

Pros	cons
1) Robust to outliers	1) Inability to extract a linear comb of features
2) Feature scaling not required	2) High variance leading to small computational power.
3) Handles missing values	
4) can deal with irrelevant input	
5) computational scalability	

→ Boosting minimizes the variance by taking into consideration the results from various trees.

General understanding eg of boosting

① you wanna travel to different place.

→ And you know a friend who is a traveller, so you ask him about the place (Gives your info etc) he will tell what he know & he will ask his friends.

→ He asks his friend who visited them to give some phone no or any agency.

→ And that friend asks another friend who lives there.

> First, we have to boost other person with info we have, and he will boost next person with info he have & finally get result

② Dad wants to buy car.

→ He asks you & without any knowledge you say 'Yes'. (less weightage)

→ Grandpa thinks and says what model should be bought consider grandchildren who wants car. (good weightage)

→ consider son & grandpa, mom says what to buy, new or second hand, based on financial status (better weightage)

(6) Ada Boost (Adaptive Boosting)

→ Assume that the no. of training samples are "N", and the no. of iterations (trees created) is "M". Note that possible class outputs are $Y = \{-1, 1\}$.

① Always, total weightage should be equal "1".
So Initially consider observation weights as $w_i = \frac{1}{N}$

② For $m = 1$ to M,

→ Fit a classifier $G_m(x)$ to the training data using w_i

→ compute $err_m = \sum_{i=1}^{N} w_i \cdot I(y_i \neq G_m(x_i))$ ← Prediction

→ compute $\alpha_m = \frac{1}{2} \log \frac{(1-err_m)}{err_m}$ — this is the contribution of that tree to Final result

→ calculate new weights using the formula.

$$w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$$

③ Normalize the new sample weights, so that their sum is "1".

④ construct the new tree using the new weights.

i.e; consider weights obtained in 1st tree as the weights for the second tree and, weights of 2nd tree to the 3rd and so on.

⑤ At end, compare summation of results from all trees and Final result is either the one with highest sum (for Regr) or it is the class with most weightage (for classification).

Eg:-

① understandy Gradient Adaboost by

Simple dataset for heart patient prediction

Sol

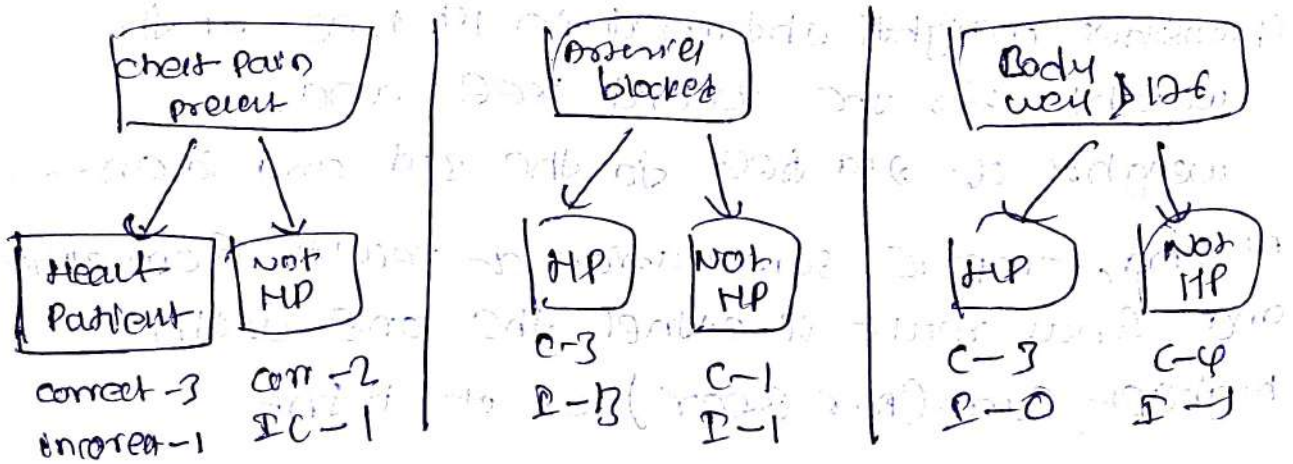
	is chest pain	Arteries blocked	weight of person	is Heart Patient	new weight
0	yes	yes	205	yes	0.05
1	no	yes	180	yes	0.05
2	yes	no	210	yes	0.05
3	yes	yes	167	yes	0.33
4	no	yes	156	no	0.05
5	no	yes	125	no	0.05
6	yes	no	168	no	0.05
7	yes	yes	172	no	0.05

Step① - Initialize weights

→ there are 8 rows in our dataset.

∴ $w = 1/N = 1/8$ (samples are equally important)

Step② - consider column to create a weak decision maker and then try to figure out what are correct & incorrect predictions based on that column



→ we will calculat "Gini index" of the column.

And we select tree with the lowest GI.
This will be the first decision maker for our model.

now, we

Step ③ → Now, we claclate the contribution of tree (columns) to our final decision

→ we got weight of Ant decision maker & we have one incorrectly predicted by it

error = 1/8 contribut = $\frac{1}{2}\log\left(\frac{1-err}{err}\right) = 0.97$.

→ Step ④ → modify the weights

→ Increase the sample weights for incorrectly predicted, new weight = old wei · $e^{\wedge} o_m$

$w_{i,0} = 1/8 \times e^{\wedge} 0.97 = 0.33$

→ Decrease the sample weight for correctly predicted, newweight = old weix $e^{\wedge}(-o_m)$

$w_i = \frac{1}{8} \times e^{\wedge}(-0.97) = 0.05$

And add weights & divide by summation to normalize it.

there, new normalized weights will act as sample weights for the next iteration.

Step ⑤

→ And we create new tree which consider dataset prepared with new sample weights.

→ Suppose, m tree classify a person as HP & n tree classify a person as Not HP.

then contribution of all m tree & all n-tree are added.

Highest is taken

⑦

* Gradient Boosting tree

→ Gradient Boosted tree use decision tree as estimator. It can work with different loss function, evaluate its gradient & approximate with simple tree

Adaboost is a special case of Gradient boosting where it use exponential loss function.

Algorithm

① calculate the avg. of the label column as initially this avg shall minimize the loss/error

② calculate pseudo residuals.

pseudo residual = actual label − predicted result (avg)

mathematically,

(*) derivative or pseudo residual = $\frac{\partial L(y_i, F(x_i))}{\partial (F(x_i))}$

→ Here, gradient or error term is getting calculated as the goal is to minimize the error. Hence, name is gradient boosted tree.

(3) create a tree to predict the pseudo residuals instead of a tree to predict for actual col. value.

new result = (previous result) + (learning rate × residual)

$F_1(x) = F_0(x) + \nu \cdot \gamma$

ν - learning rate
γ - residual.

(4) Repeat these steps until residual stops decreasing.

Eg: to understand GBM.

Q) we need to develop a model to predict an apartment rent based on sq. footage data.

Sq feet	Rent
750	1160
800	1200
850	1280
900	1450
950	2000

Sol: Step① - set original model

Lets set avg rent prices as our model : $F_0(x) = 1414$

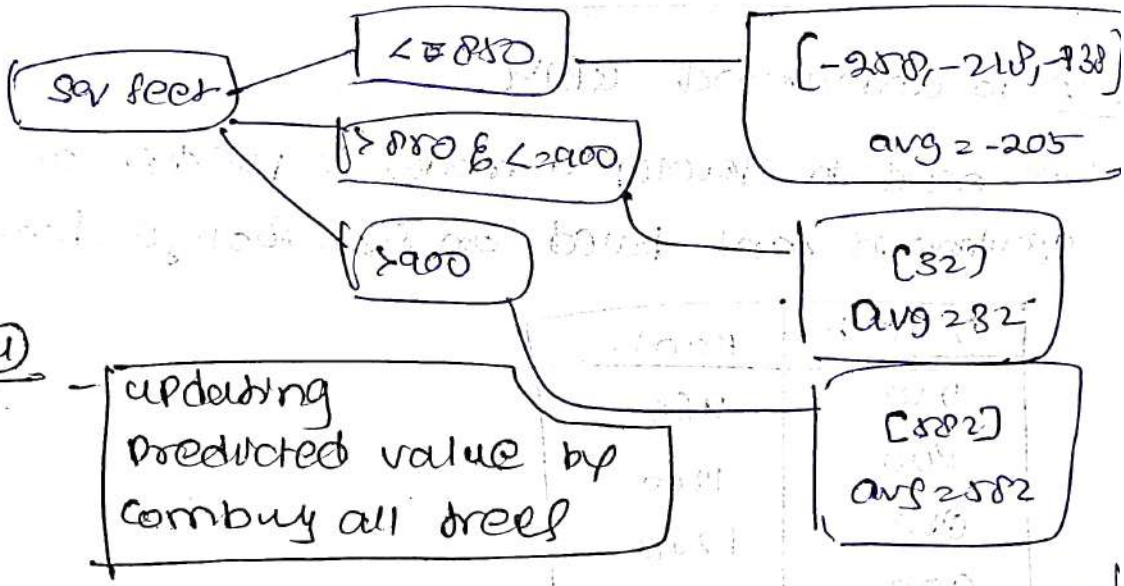
Step ① — calculate the residuals

→ for every instance, we compute the residual as diff b/w actual & predicted val (avg)

Sq feet	Actual rent	Predicted rent (avg)	Residual
250	1160	1418	-258
800	1200	1418	-218
850	1280	1418	-138
900	1450	1418	32
950	2000	1418	582

Step ③ — Build a model on residuals, instead on actual col. values

we build a decision tree to predict residual



Step ④ — updating predicted value by combing all trees

F(m) updated Predicted val = Fo(x) + (learning rate × Residual predicted)

Fo(x)	Residual Predicted	Learning rate	F(m)
1418	-205	1	1213
1418	-205	1	1213
1418	-205	1	1213
1418	32	1	1450
1418	582	1	2000

step 5 — Repeat step 2-4, for the predefined no. of iterations.

step 6 — Finally, the composite model sums together all of the weak models into one strong predictor.

GBM vs AdaBoost

→ Essentially both Adaboost & GBM have similar intuition, i.e. to convert a set of weak learner into a single strong learner.

(*) However, the only point of difference is, "How they create a weak learner during process?"

→ Adaboost, changes the weights assigned to each of the instances, such that every subsequent learner focuses more on the difficult ones and therefore, contributes to creating a strong learner.

→ Gradient boosting, trains the weak learner on errors. Each iteration computes the residuals & fits the next learner. Finally, using an optimization process, the algorithm computes the contribution of each weak learner which minimizes the overall error of the model.

(8) XG Boost – (eXtreme Gradient Boosting)

→ xa boost regularized data better than normal gradient boosted treel.

→ XG boosty objective function is the sum of loss function evaluated over all the predictions & a regularization func for all predictors (3 trees).

$$obj(\theta) = \sum_{i}^{n} l(y_i - \hat{y}_i) + \sum_{j=1}^{j} \Omega(f_j)$$

→ Loss function depends on the task being performed and a regularization term is described,

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

→ watched over scores.

→ for controlling overall no of created leaves.

→ unlike other tree-building algorithms, XG Boost doesn't use entropy or Gini index. Instead, it utilizes gradient (error term) and a "hessian" for creating the trees.

→ Hessian for a Regr. problem is no. of residuals and for classification problem, Hessian is a second order derivative of the loss at the current estimate.

$$h_m(x) = \frac{\partial^2 L(y, f(x))}{\partial f(x)^2}_{f(x) = f^{(m-1)}(x)}$$

Tree Building in XG Boost

① Initialize the tree with one leaf.

② compute the "Similarity" using,

$$\text{Similarity} = \frac{\text{Gradient}^2}{\text{Hessian} + \lambda}$$

where, λ - Regularization term.

③ Now, for splitting data into a treeform, calculate,

$$\text{Gain} = (\text{left Similarity}) + (\text{Right Similarity}) - (\text{Similarity for Root})$$

④ For tree pruning, the parameter γ is used. the algorithm starts from the lowest level of the tree & then starts pruning based on the value of γ.

If $(\text{Gain} - \gamma) < 0$, then remove that branch

else, keep the branch.

⑤ Learning is done using new eq.

$$\text{new value} = \text{old value} + \eta \times \text{Prediction}$$

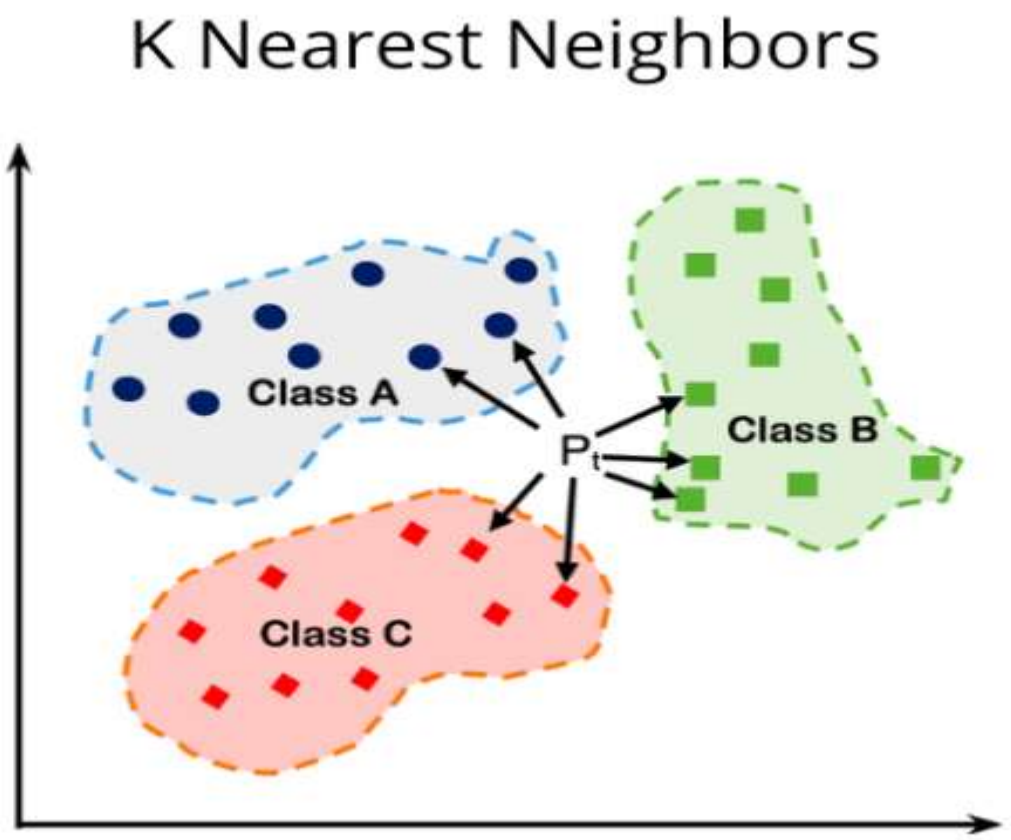


Table Of Contents

(9) K-Nearest neighbour (KNN)

→ KNN is a type of supervised learning algorithm which is used for both Regression & classification, mostly for classification.

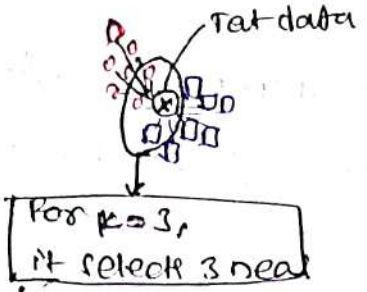
→ Given a dataset with different classes, KNN tries to predict the correct class for test data by calculating the distance between the test data and ALL THE TRAINING POINTS!! (ALL)
Each element in test is measured with all the training points.

→ And then select the k points which are closest to the test data.
Hence, 'K' in K-NN is,
"How many closest point are you selecting to predict data?"

→ once the points are selected, the algorithm calculates the probability (for classification) of the test point belonging to the class of the k-training points and the class with the highest probability is selected.

→ In case of Regression Problem,
the predicted value is the mean of the k-selected training points.

Algorithm / visual illustration



KNN algorithm calculates distance b/w the test data and the given training data. And then it will select k points which are nearest.

↳ And it's 2 red, 1 blue. Hence, it is Red'C

↳ If it is Regr probl, the predicted value is mean of those three.

How distance calculated?

(*) Euclidean distance:-

→ It is most commonly used method to calculate the distance b/w two points.

→ The euclidean distance b/w two points, $P(p_1, p_2)$ & $Q(q_1, q_2)$ is calculated as:-

$$d(P,Q) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2}$$

(*) Hamming distance:-

→ Hamming distance is the distance metric that measures the no. of mismatches b/w two vectors. It is most widely used on cate or categorical data.

→ Generally, if we have features of categorical data then we consider the difference to be 0 if both values are same, difference is 1 if both are different.

Eg: $(A, B, C, D) - (A, D, C, B) = (0, 1, 0, 1) = 2$

(*) manhattan distance

→ Also known as L1 norm, Taxicab norm, Rectilinear dist, city block distance.

→ this distance represents the sum of absolute differences b/w the opposite values of vector.

$$md(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

→ manhattan distance is less influenced by outliers than the Euclidean distance, worth very high dimensional data is more preferred.

(*) Lazy learners

→ KNN algorithm is often termed as lazy learner.

→ And most of the algorithms like Bayesian classific, logistic Regression & SVM etc. are called Eager learners these algorithms generalize over the training set before receiving the test data.

i.e, they create the model based on the training data before receiving the test data, and then do the prediction of the data.

→ But this is not the case with KNN algorithm, it doesn't create a generalized model (equation) for the training set but waits for the test data. once the test data is provided then only it starts generalizing the training data to classify the test data. Here, entire training data is the model, wait for test set.

→ So, a lazy learner just stores training data &

(*) weighted nearest neighbour

→ As name suggests, we assign weight to the k-nearest neighbour. The weights are typically assigned on the basis of distance.
Sometimes test or data are assigned '0' also.

→ the main intuition is that the points in neighbor should have more weights than farther points.

(*) choosing the value of K

→ The value of K- affects the KNN classifier drastically

→ the flexibility of model decreases with the increase of K.

→ If 'K' value is lower, then model has high variance & low bias.
As 'K' value is increased, then variance decreases & bias increases.

→ with very low value of K, there is a chance of algorithm overfitting the data, where as
with very high value of K, there is a chance of 'underfitting'

⊛ Different ways to Perform K-NN

→ the way K-NN classifies the data by calculating the distance of test data from each of the observations and selecting K-values, this approach is known as "Brute Force KNN". (which is computationally very expensive (time taking))

→ So, the idea behind using other algorithms for KNN classifier is to reduce the time during test-period by preprocessing the training data in such a way that the test data can be easily classified in the "appropriate cluster"

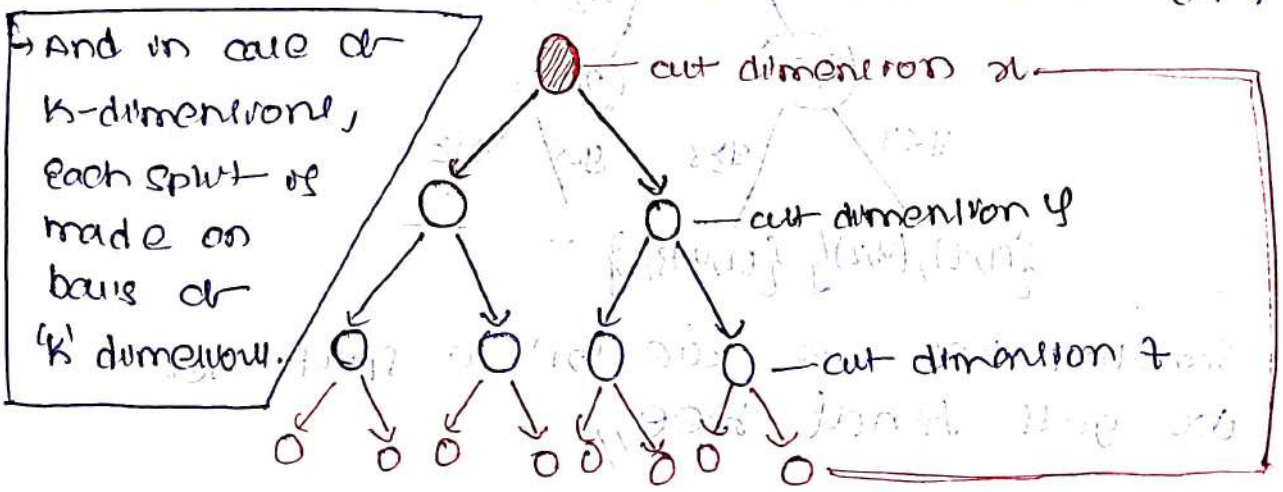
⊛ K-dimensional Tree (Kd Tree)

→ K-d Tree is a hierarchical binary tree.

→ It rearranges the whole dataset in a binary tree structure, so that when test data is provided, it could give out the result by traversing through the tree (every split eliminates half data), which takes less time than brute search.

Eg let's say we have 3 dimensional data (x, y, z).

↳ And in case of K-dimensions, each split is made on basis of 'K' dimensions.



Eg)

Q) training data = { (1,2), (2,3), (2,4), (3,6), (4,2), (5,7), (6,8), (7,5), (8,5), (9,1), (9,3) }

A) Here, K=2

Let's build our kd-tree

(i) Sort data & choose median as split point.

$x \in \{1, 2, 2, 3, 4, (5), 6, 7, 8, 9, 9\}$

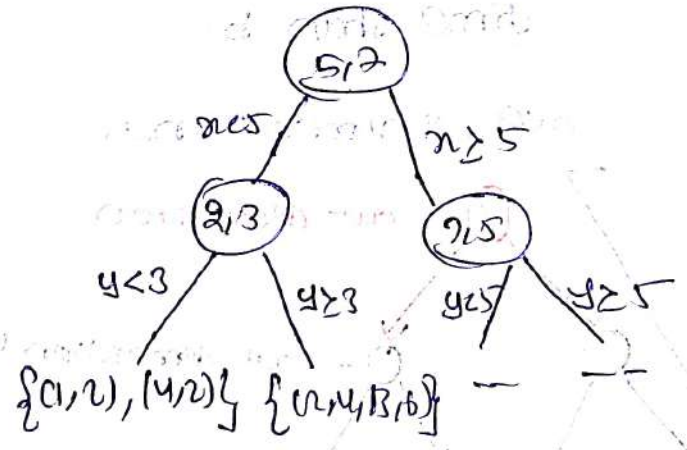
↳ median.

∴ our first node will be (5,7); $x \geq 5$ (split cond.)

$S_1 \in \{(1,2), (2,3), (2,4), (3,6), (4,2)\}$ $S_2 \in \{(6,8), (7,5), (8,5), (9,1), (9,3)\}$

(ii) Split S_1, S_2 on condition of y.

$y_{S_1} \in \{2, 2, (3), 4, 6\}$ $y_{S_2} \in \{1, 3, (5), 5, 8\}$



Similar, here we see x' to split and we get final tree.

note (kd-tree) :-

→ once the tree is formed, it is easy for algorithm to search for the probable nearest neighbor just by traveling tree.

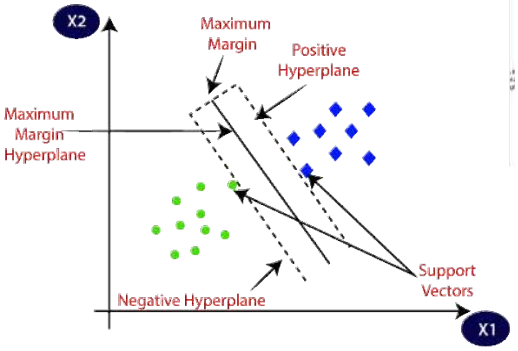
→ the main problem of kd tree is that it gives probable nearest neighbors but can miss out actual nearest neighbors.

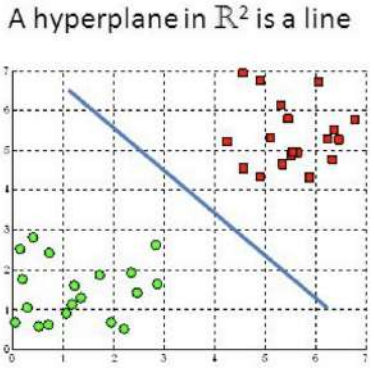
ii) Ball-Tree

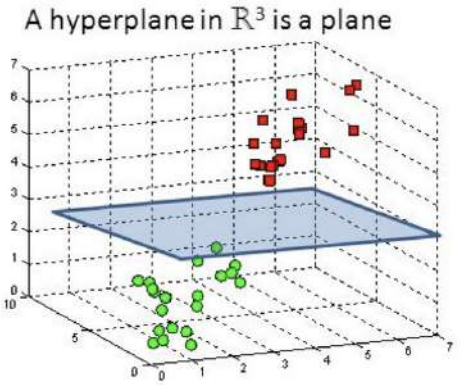
→ similar to kd trees, Ball trees are also hierarchical data structures. these are very efficient specially in case of higher dimensions.

Formed by following steps

→ two clusters are created initially.
→ All data points must belong to atleast one of the cluster.
→ one point cannot be in both clusters.
→ distance of point is calculated from centeroid of each cluster. the point closer to centeroid goes into that particular cluster.
→ each cluster is then divided into subclusters again, and then the points are classified into each cluster on the basis of distance from centroid.
→ this is how clusters are kept divided till a certain depth.







Understanding Support Vector Machines

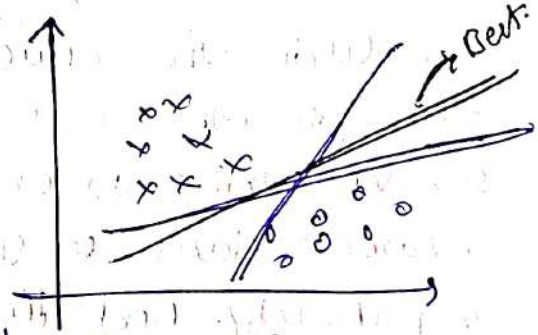
Table Of Contents

(*) Support Vectors Machine (SVM)

→ Support-vector machine is a supervised machine learning algorithm, which can be a classifier as well as Regressor. This is a linear model, and if consider linear classification models we have
→ Logistic Regression and support vector classifier.
→ In case of logistic Regression, we ultimately build a line / plane / hyperplane, and classify the points to the left or the line, and given a new point based on the probability, we can say which part or plane it belongs. Now, what about SVM.

⊙ Concept of SVM

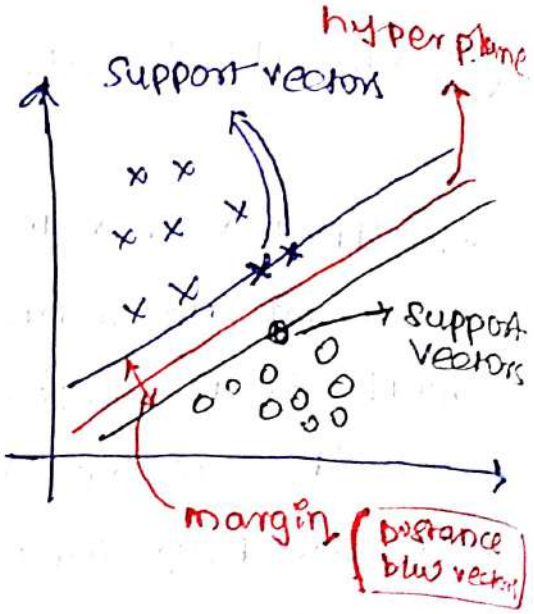
→ similar to logistic Regression, we get a plane / hyper plane in SVM, but how do we decide on the best hyperplane. There can be several planes passing through the datapoints.



→ So, the key principle behind SVM is to find the hyperplane / decision boundary, that maximally seperates data points belonging to distinct classes, that is, maximising the margin between datapoints.

① What is a support vector

→ Let's suppose we have a decision boundary and to both sides of line we take the closest points and build a parallel plane to our decision boundary.

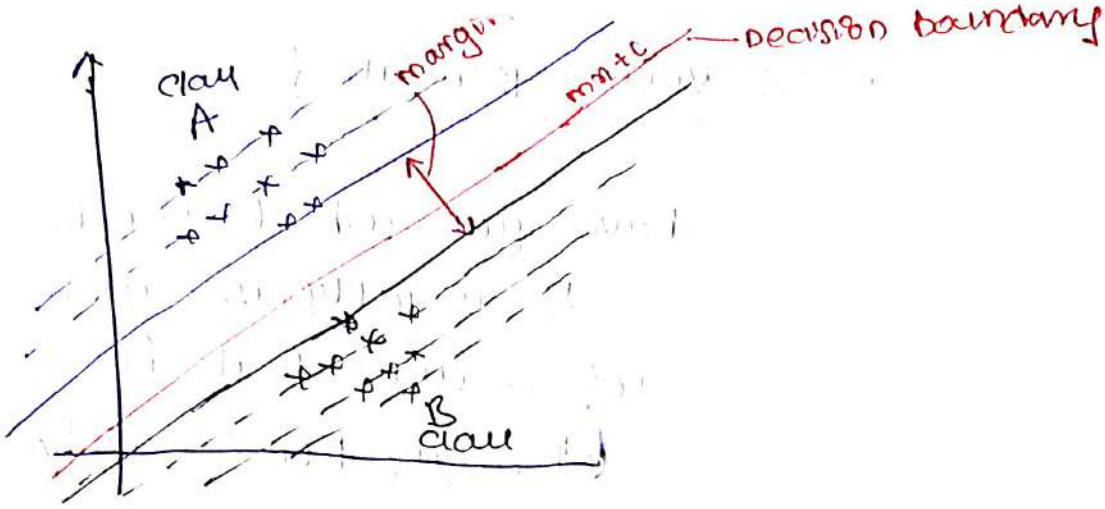


→ the points that are close to decision boundary are called "support vectors". And these support vectors are crucial for defining hyperplane so these support vectors help us find optimal plane

② what is margin

→ If we consider logistic Regression, based on the Probability we will decide the class for a new point, but there are chances for things to go wrong here, say probability is 0.49 & so we conclude the class, but it can be wrong!

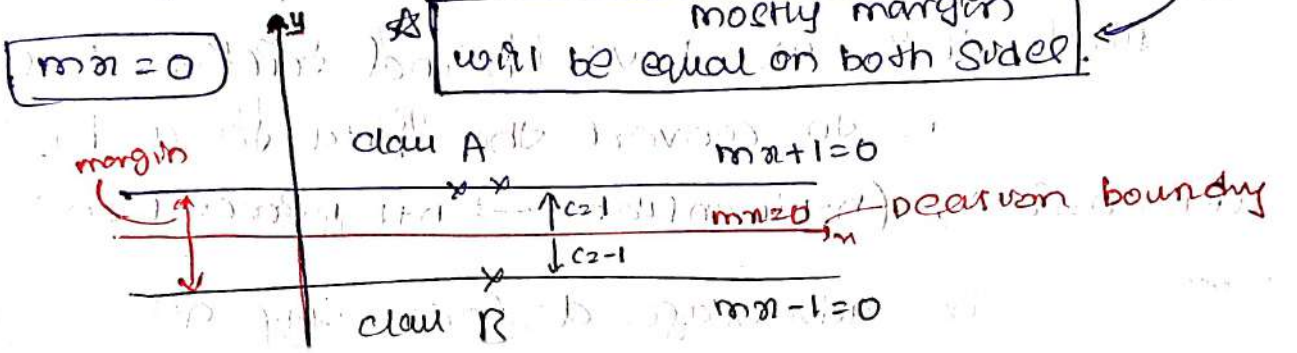
So, SVM tries to create maximum seperation between data to get rid of the confusion of Probability. And the maximum seperation can be achieved through the distance between two closest vector lines from opp. sides.



→ decision boundary can be represented as, $y = mx + c$,

→ consider decision boundary is at origin, we get, and assume support vectors have $c = 1$.

mostly margin will be equal on both sides.



→ Now, if we want to predict a new datapoint, then we just check the y value or mx if it falls in (0 to 1), it comes under class A.

or

if it falls in (0 to -1), it comes under class B and we don't need probability, just by the range of y, we are predicting the new datapoints.

⊙ Soft margin and Hard Margin

→ the concept of SM & HM, refer to different approaches in handling the margin violations & the presence of noise or outliers in the data.

Hard Margin

→ HM aims to find the optimal hyperplane that completely separates the classes without any margin violations. It

→ It works well when the data is linearly separable & free from noise.

$$y_i(w^T x_i + b) \geq 1$$

↳ signifies all data points are correctly classified.

Soft Margin

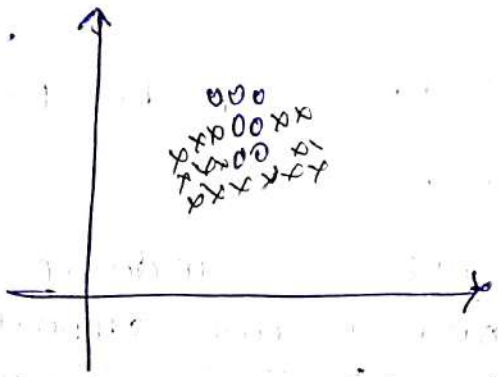
→ SM allows for some margin violations & misclassifications, thus providing flexibility when dealing with noisy / overlapping data. It introduces a slack variable (ξ_i) to handle margin violations, allowing some points to fall within the margin / on wrong side of boundary.

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

↳ where $\xi_i \geq 0$ for all points

⊙ Kernelized support vector machines

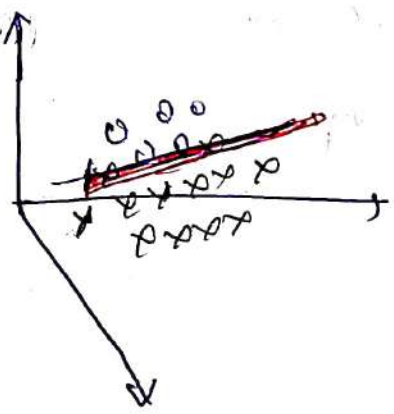
+ It's not always we have get which can be easily seperable by line!! usually, we will be having data in such a way that it cannot be seperable through lines easily! Such as below.



→ To deal this we have kernel tricks. Basically, the idea is to convert the data to a higher dimension, (n-dimension → n+1 dimension).

→ What is the advantage of increasing dimension? If we take above example, there is no way to apply SVM, we can't get a decision boundary line as the points are clustered together like as a 2D data.

Now, imagine it is converted to 3D dimension, then for sure we can get a layer or seperation between the data points.



⊙ How do we increase the dimension?

→ Say we have a 2D dataset as (x, y). Now if we want to add another dimension, we can simply use some mathematical function to find z.

Eg $z = x + y$, $z = x - y$, $z = x^2 + y^2$, $z = x^2 - y^2$ etc.

→ So we can project the points in a 3D space and then derive a plane which divides the data into two parts. In theory, that's what a Kernel function does without computing additional co-ordinates for the higher dimension.

⊙ Types of Kernels in SVM

→ There are many kernels, and here are a few popular ones.

1) Linear kernel: computes dot product between two feature vectors.

$$K(x, y) = x^T y$$

2) Polynomial kernel: $K(x, y) = (x^T y + c)^d$

3) Radial Basis Function (RBF) kernel:

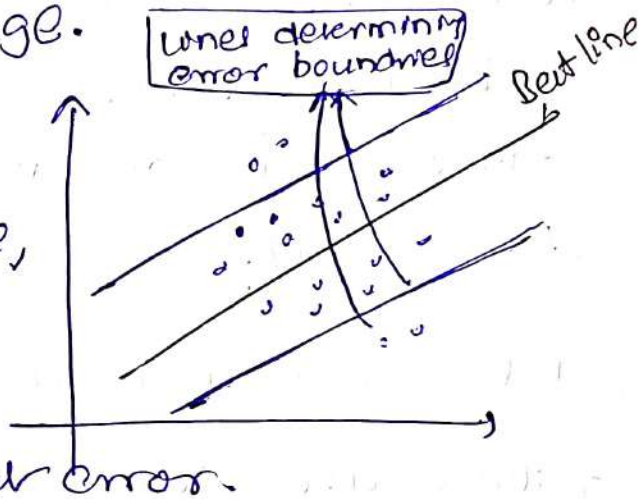
RBF kernel measures the similarity between two samples using a Gaussian radial function.

$$K(x, y) = \exp(-\gamma \|x - y\|^2)$$

⊙ Support vector Regressor

→ we know how Linear Regression works, where we determine the best fit line. And in LR, the idea is to create a line which minimizes the total Residual error.

→ In SVR, the approach is a bit different. Here, instead of trying to minimize the error, SVR focuses on keeping the error in a fixed range.



→ Here, the middle line is the best fit regressor line, and the other two lines are the bordering ones which denote the range of error.

→ The Best-fit line / hyperplane, will be the one which goes through the maximum number of data points and the error boundaries are chosen to ensure maximum inclusion.

→ So, we simply bring majority of data points in between the margin lines.

Naive Bayes Classifier

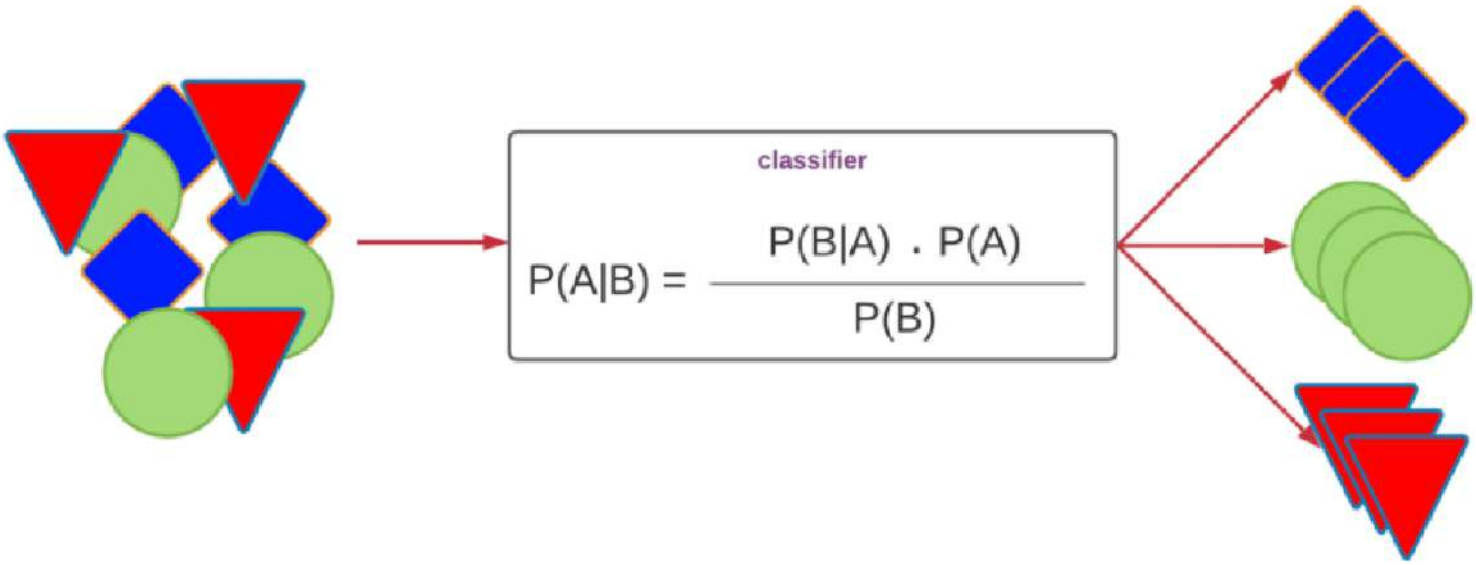


Table Of Contents

⊙ Naive Bayes classifier

→ If we have a dataset, where relationship with one single feature may effect the output/label, the we cannot generalize about all the features at a time!!

Eg:-

1) The food I ordered is good.
2) The food I ordered is bad.

Here only the last word signifies about the sentiment or statement.

⊗ This is where Naive Bayes helps in?

→ Naive Bayes will not try to generalise complete features at a time and instead tries to build a independent relationship each & every input feature with output label, so that we get a better idea.

⊙ How it works?

→ Naive Bayes is not a single algorithm, but a family of algorithms where all of them share a common principle, i.e; they are based on "Bayes"

→ Bayes' Theorem finds the probability of an event occuring given the probability of another event that has already occured.

mathematically:

$$P(A/B) = \frac{P(B/A) \cdot P(A)}{P(B)}$$

where A & B are events & $P(B) \neq 0$

→ Basically we are trying to find the Probability of A, given the event B is true. Event B is also termed as evidence.

→ P(A) is the Priori of A, that is the probability of event before evidence is seen (Prior Probability).

→ Here, event B occurance

→ P(A/B) is the posteriori probability, that is the probability of event after evidence is seen.

Eg: Let's take a dataset of Playing golf based on climate.

S.No	Outlook	Temperature	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

→ Here we can say that if it's sunny, mostly likely operion wouldn't go for game. Similar case if it's too windy / too rainy.

→ So we can clearly see that a single feature will affect the label directly. Hence, we will build model using independet relationship of each feature with output label by levraging Naive Bayes.

→ But before we apply Naive Bayes, we should make sure it follows the assumptions.

☆ ① the fundamentals Naive Bayes assumption is that each feature makes an independent, and equal contribution to the outcome.

→ And we can clearly say no pair of features are dependent, Eg- the temperature being hot has nothing to do with humidity / being rainy. Hence, features are independent.

→ knowing only temperature & humidity alone can't predict the outcome, hence assumed all features contribute equally to outcome.

we have, Bayes' theorem as

$$P(y/x) = \frac{P(x/y) \cdot P(y)}{P(x)}$$

y - dall variable

x - dependent feature

→ And by taking naive assumption to bayes theorem, which is independence among the features, we get. $P(A,B) = P(A) \cdot P(B)$

$$P(y \mid (x_1, x_2, \dots x_n)) = \frac{P((x_1, x_2, \dots x_n)/y) \cdot P(y)}{P(x_1, x_2 \dots x_n)}$$

Based on Naive assumption of independence

$$= \frac{[P(x_1|y) \cdot P(x_2|y) \dots P(x_n|y)] \cdot P(y)}{P(x_1) \cdot P(x_2) \dots P(x_n)}$$

$$= \frac{P(y) \cdot \prod_{i=1}^{n} P(x_i|y)}{\prod_{i=1}^{n} P(x_i)}$$

→ constant for any class variable in a class.

→ Now, to make a classifier model, we find the probability of a given set of inputs for all possible values of the class variable y and pick up the output with maximum probability.

$$\therefore y = \text{argmax}_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

→ So, to solve this problem we need to calculate $P(y)$ & $P(x_i|y)$ → class probability, Conditional Probability.

→ So now we calculate $P(\text{Sunny}|\text{yes})$, $P(\text{overcast}|\text{yes})$, $P(\text{Rainy}|\text{yes})$, $P(\text{Sunny}|\text{No})$, $P(\text{overcast}|\text{No})$, $P(\text{Rainy}|\text{No})$, similarly for each feature. $P(x_i|y_j)$

we get,

Outlook feature

	Y	N	P(Y)	P(N)
Sunny	3	2	3/9	2/5
Overcast	4	0	4/9	0/5
Rainy	2	3	2/9	3/5
Total	9	5	100%	100%

Temperature Feature

	Y	N	P(Y)	P(N)
Hot	2	2	2/9	2/5
mild	4	2	4/9	2/5
cold	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity Feature

	Y	N	P(Y)	P(N)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind Feature

	Y	N	P(Y)	P(W)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(yes) / P(no)
yes	9	9/14
No	5	5/14
Total	14	100%

→ Here we calculated $P(x_i^o | y_j^o)$ for each x_i^o & y_i^o, and also $P(y_j^o)$ for each class variable y_j

Prediction

→ Now, if we have a new datapoint as

(Sunny, Hot, Normal, False) = today climate.

$$P(y \mid \text{today climate}) = \frac{P(\text{Sunny}|\text{yes}) \cdot P(\text{Hot}|\text{yes}) \cdot P(\text{Normal}|\text{yes}) \cdot P(\text{False}|\text{yes}) \cdot P(\text{yes})}{P(\text{today})}$$

$$P(\text{yes}|\text{today}) \propto \frac{3}{9} \cdot \frac{2}{9} \cdot \frac{6}{9} \cdot \frac{6}{9} \cdot \frac{9}{14} \approx 0.0211$$

$P(N|today) \propto \frac{2}{5} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{2}{5} \cdot \frac{5}{14} \approx 0.1142$

However, we know,

$P(y|today) + P(N|today) = 1$

Apply reciprocal,

$$1 = \frac{1}{P(y|today) + P(N|today)} \quad \therefore \text{multiply } P(y|today)$$

$$P(y|today) = \frac{P(y|today)}{P(y|today) + P(N|today)}$$

$$P(y|today) = \frac{0.0141}{0.0211 + 0.1142} = \frac{0.0141}{0.1353} = 0.155$$

$$P(N|today) = \frac{0.1142}{0.0211 + 0.1142} = \frac{0.1142}{0.1353} = 0.845.$$

So, clearly $\boxed{P(N|today) > P(y|today)}$

Hence, Play golf → No for today's climate.

→ However, this above method is for the categorical data.

→ And in case of continuous data we need to make assumptions regarding the distribution of values of each feature.

→ ☆ The different Naive Bayes classifier differ mainly by the assumptions we make regarding the distribution of $P(x_i|y)$

⊙ Gaussian Naive Bayes classifier —

→ If the feature variables are a continuous variables, then we cannot use the Naive Bayes, and instead we should be using Gaussian Naive Bayes classifier.

→ In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution, also called "Normal" distribution

→ The likelihood of the features is assumed to be Gaussian, hence conditional probability is,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \cdot \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

Other popular Naive Bayes classifier are —

⊙ multinomial Naive Bayes: Feature vector represent the frequencies with which certain events have been generated by multinomial distribution. This is typically used for document classification.

⊙ Bernoulli Naive Bayes: Assumes that the features are binary or categorical. It is particularly useful for document classification like MNB, and where binary term occurrence features are used rather than term frequencies.

Types of Clustering Algorithms

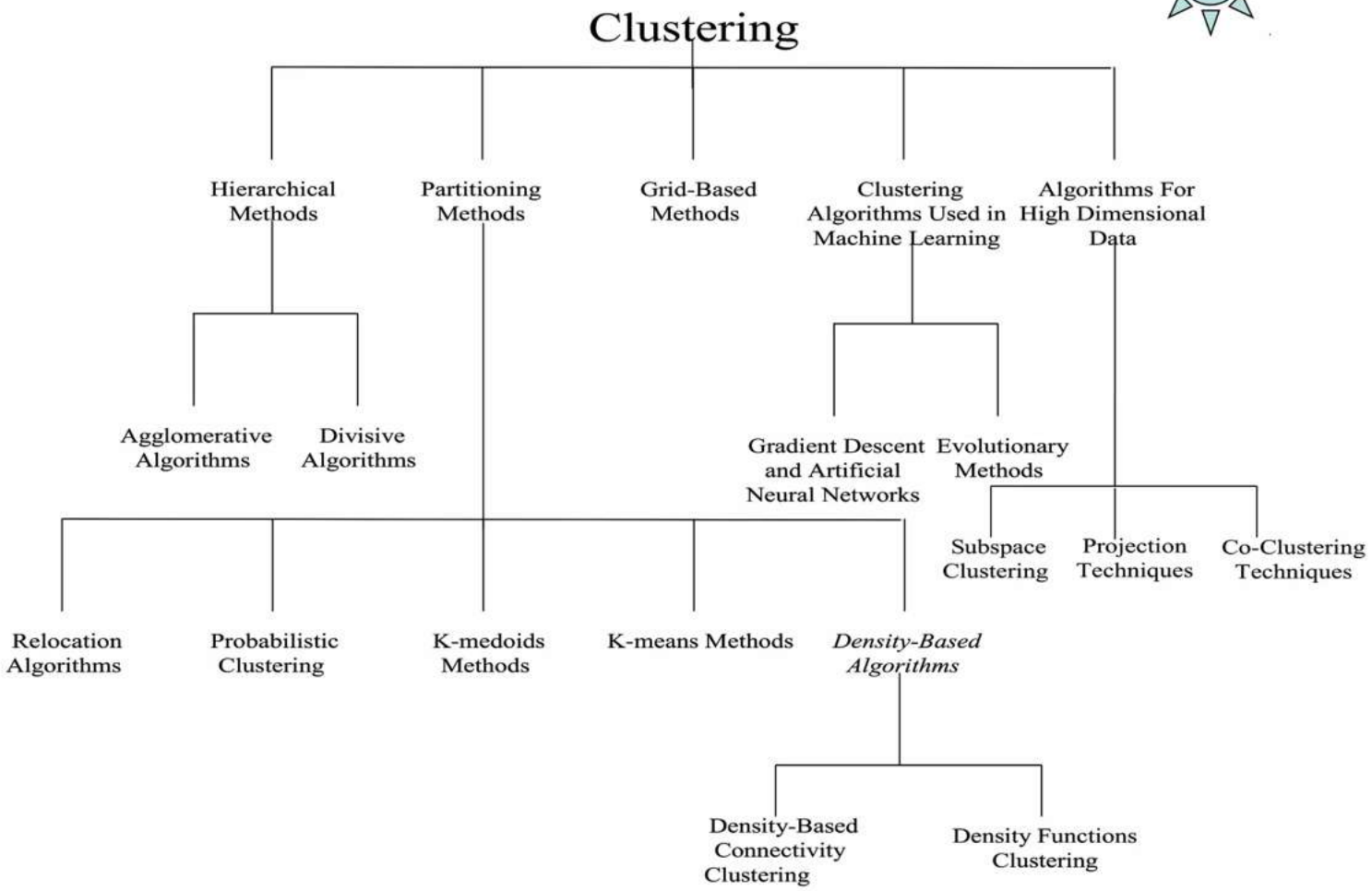


Table of Contents

Unsupervised learning Algorithms. (Pg no of 5 References)

i) K-Means
ii) Hierarchical clustering
iii) DBSCAN
} clustering - Evaluation metrics for clustering are important...

iv) Performance measurement
v) Principal component Analysis
vi) Dimensionality Reduction.

(*) clustering :-

→ First we need to understand clustering is different from classification.

Parameter	classification	clustering
1) Type	used for supervised learning.	used for unsupervised learning.
2) Basic	Process of classifying the input instances based on their corresponding "labels"	Grouping the instances based on their similarity without the help of class "labels"
3) complexity	more complex	less complex
4) Example Algo	Logistic Regression, Naive Bayes, SVM etc— KNN	K-means clustering, Hierarchical clustering, DBScan, BIRCH Algorithm etc

Applications of clustering

→ For customer segmentation:

You can cluster your customers based on their purchases, their activity on website and so And this can be used in Recommender systems to suggest content that other users in same cluster enjoyed (bingo).

Eg: marketing, Insurance, Libraries etc.

→ Earth-quake studies:

By learning old data it can make clusters and determine dangerous zones.

→ For search engines:

As you search, similar images would end up on same cluster.

→ To segment an Image:

By clustering pixels according to their color, then replacing each pixel's color with the mean color of its cluster, it is possible to reduce no. of different colors in image.

Eg: used in object detection & tracking systems.

→ For Anamoly detection (outliers)

→ As a dimensionality Reduction Technique.

→ For Data Analysis

clustering methods

① Density-Based Methods

These methods consider the clusters as the dense region having some similarity and different from the lower dense region or spa

these methods have good accuracy & ability to merge two cluster.

Eg:-

i) DBSCAN [Density Based Spatial Clustering of Applications with Noise

ii) OPTICS [Ordering Points to Identify Clustering Structure.

② Hierarchical-Based methods

The cluster formed in this method forms a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one.
It has two categories

i) Agglomerative [Bottom up approach - forms a single cluster & apply

ii) Divisive [top down approach - form a big cluster & divide

Eg:-

iii) CURE [Clustering Using Representatives]

iv) BIRCH [Balanced Iterative Reducing Clustering and using Hierarchies]

③ Partitioning methods:-

- these methods partition the objects into k-clusters and each partition forms one cluster. this method is used to optimize an objective criterion similarity function such as when the distance is a major parameter

Eg:
i) K-means
ii) CLARANS [Clustering Large Applications based upon Randomized Search]

④ Grid-Based methods:-

In this method the data space is formulated into a finite no. of cells that form a grid like structure. All the clustering operation done on these grids are fast and independent of the no. of data objects.

Eg:
i) STING [statistical Information Grid]
ii) CLIQUE [Clustering In QUEst]
iii) Wave cluster, etc.

Main Requirements for clustering Algorithms

(1) It should be scalable.

(2) It should be able to deal with attributes of different types.

(3) It should be able to discover arbitrary shape clusters.

(4) It should have an inbuilt ability to deal with Noise & outliers.

(5) The cluster should not vary with the order of input records.

(6) It should be able to handle data of high dimensions.

(7) It should be easy to interpret & use.