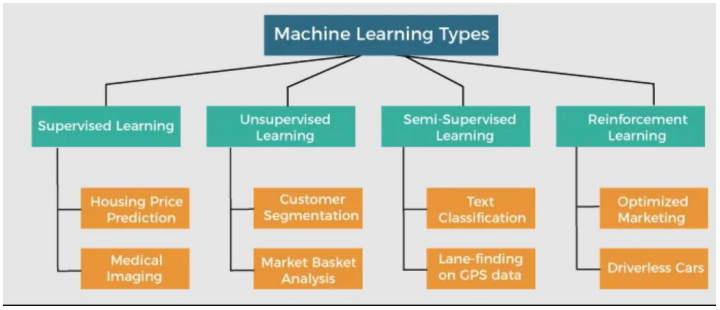
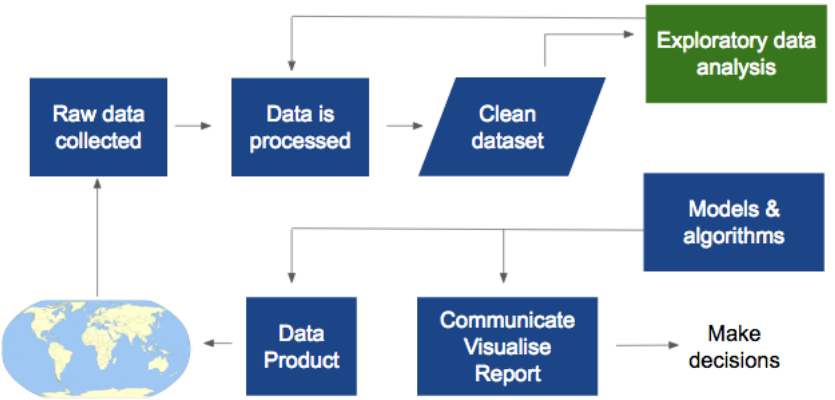
- Understanding Machine Learning Algorithms: Handwritten Notes -

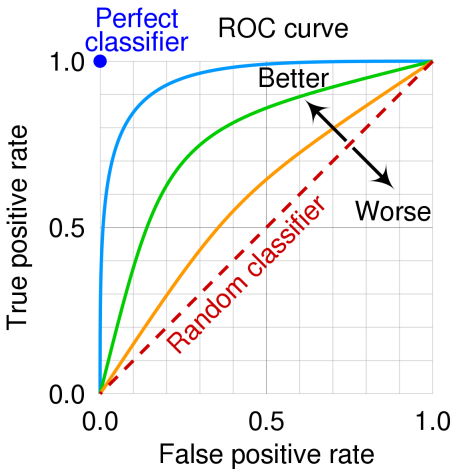
Table of Contents

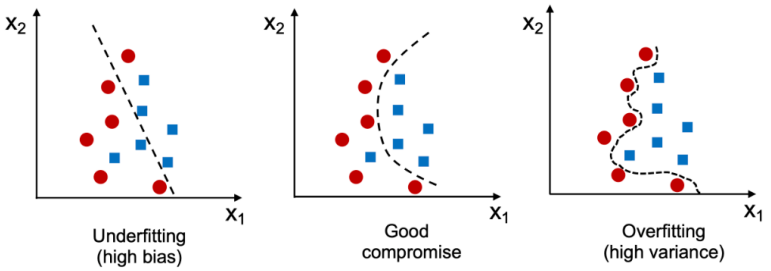


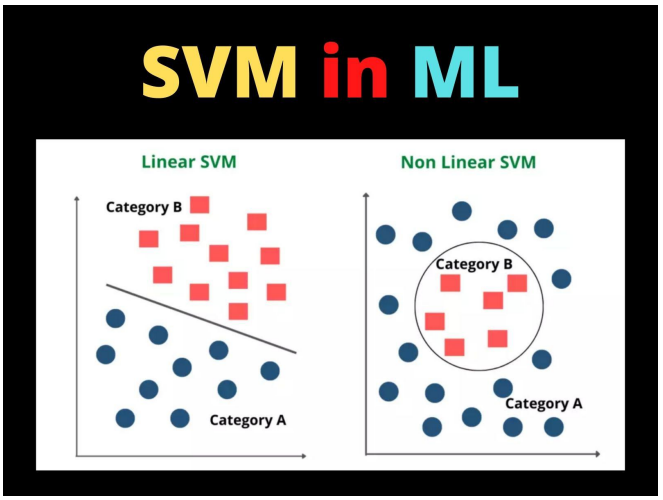


Notes by RaviTeja G

10. Exploring Step 3 - Train Model on Dataset
    - Types of Learning
    - Under Fitting and OverFitting
    - Regularization techniques
    - Hyperparameter Tuning
11. Exploring Step 4 - Evaluation of a Model
    - Evaluation Metrics
    - Confusion Matrix
    - Recall/Sensitivity
    - Precision
    - Specificity
    - F1 Score
    - AUC and ROC Curve
    - Analysis of a Model
12. Supervised Learning
    - Linear Regression
    - Regularization Techniques
    - Logistic Regression
    - Decision Trees
    - Ensemble Techniques
    - Random Forests
    - AdaBoost
    - Gradient Boost
    - XG Boost
    - K-Nearest Neighbours
    - Support Vector Machines
    - Naive Bayes Classifiers
13. Unsupervised Learning
    - Clustering Techniques
    - K-Means Clustering
    - Hierarchical Clustering
    - DB Scan Clustering
    - Evaluation of Clustering Models
    - Curse of Dimensionality
    - Principal Component Analysis
14. Cheat Sheet of Supervised and Unsupervised Algorithms







Download Deep Learning Notes

https://t.me/AIMLDeepThaught/663

Machine Learning

Introduction:-

→ At a high-level, machine learning is simply the study of teaching a computer program/ algorithm how to progressively improve upon a set task that it is given.

So, more practically it is the study of how to build application that exhibit this, iterative improvement.

there are many ways to implement it and largely there are 3 major categories.

1) Supervised Learning.

2) unsupervised Learning.

3) Reinforcement Learning.

And also there is semi-supervised learning.

machine learning algorithms are the 'engine of machine learning', meaning it is the algorithms that turn a dataset into a model.

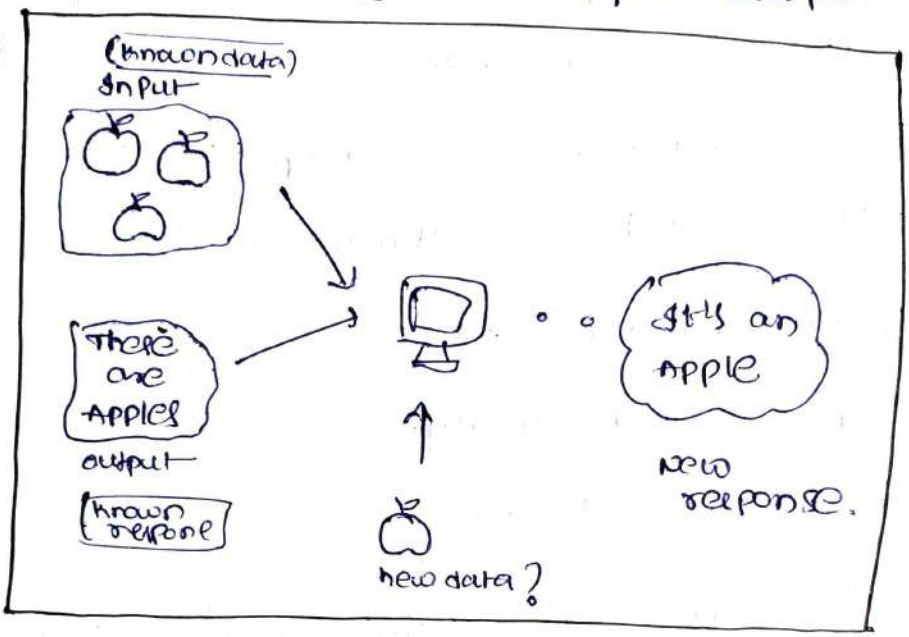
→ which kind of algorithm works best and what to choose (supervised, unsupervised, classification, regression, etc..) depends on the kind of problem you're solving, the computing resources available & the nature of the data

Briefing the types:-

(1) Supervised learning

Supervised learning is the most popular paradigm for performing machine learning operations. It is widely used for data where there is a precise mapping b/w input-output data.

i.e;



→ Given data in the form of examples with labels, we can feed a learning algorithm these 'example-label' pairs one by one. overtime, the algorithm will learn to approximate the exact nature of the relationship b/w examples & their labels. (It's called training the data)

→ when fully trained, the supervised learning algorithm will be able to observe a new, never-before seen example & predict a good label for it.

→ And so supervised algorithms are called 'task-oriented'. As we provide it with more & more examples, it is able to learn more & more properly so that it can undertake the task & yield us the output more accurately.

It is exhibited in many of following applicatns

Face Recognition:-

Face lock app in our phone has been using a supervised learning algorithm that it is trained to recognize your face. Having a system that takes a photo, finds faces & guesses who that is in the photo (suggesting a tag) is a supervised process.

Spam classifications:-

Spam filter is a supervised learning system. Fed email examples & labels (spam/not-spam) these systems learn how to filter out malicious emails so that the user is not harassed.

Advertisement popularity:-

Selecting advertisements that will perform well is often a supervised learning task. Many of ads you see as you browse the are placed there because a learning algorithm said that they would be reasonable popularity (more clicked).

and many examples... (3)

Types of Supervised Learning

1) Regression
2) Classification
3) Neural networks
4) Boosting techniques.

→

Regression

→ Regression algorithms predicts a '**continuous value**' based on the input variables.

Regression algorithms:-

1) Linear Regression
2) Polynomial Regression
3) Exponential Regression

) Logarithmic Regression

Applications:-

① Risk Assessment
② Score prediction etc.

④ ——— ouc ——— Temp predict

Classification

→ we use classification algorithms for predict set of items class or category.

Classification algorithms:-

Logistic Regression
1) K-nearest neighbours
2) Decision trees
3) Random forest
4) Support vector machine (SVM)
5) Naive Bayes
6) Ada boost
7) XG boost

Applications:-

① Fraud detection
② Email spam detection
③ Image classification
④ Diagnostics etc...

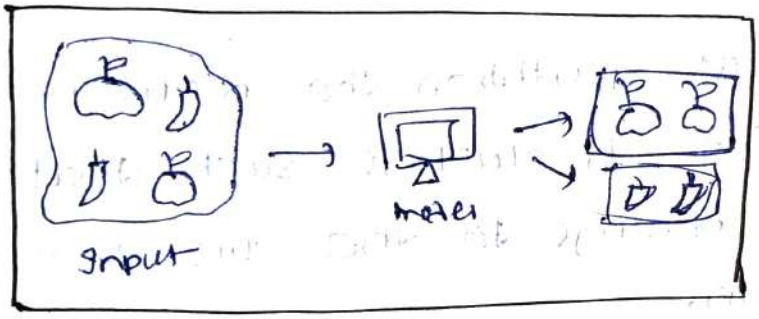
Hot	cold

② unsupervised learning

[Based on Hebbian learning]

→ To easy understand, in case of unsupervised learning algorithm, the data is not explicitly labeled into diff. datas, that is no labels. The model is able to learn from the data by finding implicit patterns.

→ unsupervised learning algorithms identify the data based on their density / structures, similar segments & other similar features.



Types of unsupervised learning :-

1) clustering
2) Dimensionality reduction & visualization
3) Anamomaly detection
4) Association
5) Auto encoders

1) clustering :-

Clustering, also known as cluster analysis, is a technique of grouping similar sets of objects in the same group that is diff from the objects in other group.

→ Some of the essential clustering techniques are as follows:-

a) K-means

In this we partition the n observations in the data into K cluster such that each observation belongs to the cluster with nearest mean.

b) DBSCAN

It groups the data based on density. It groups together the points that are given in space & mark the outliers in low density regions.

c) Hierarchical clustering

A hierarchy of cluster is built.

d) Mean Shift

2) Dimensionality Reduction & visualization:

a) Principal Component Analysis (PCA):

Reduce data from more dimensions to lower dimensions while attempting to preserve the variance.

→ we Reduce the size of data to extract useful information.

b) t-Distributed Stochastic Neighbor Embedding (t-SNE):

It is used to visualize high-dimensionality data in a 2D / 3D space.

3) Anomaly detection:

Anomaly detection techniques detect outliers in the unlabeled data under an assumption that most of the data examples are normal. by observing the instances that fit the remainder dataset.

one-class classification:

Train a model on only one-class, if anything lays outside of this class, it may be an anomaly.

Algorithms for doing so include..

(i) one-class k-means
(ii) one-class SVM
(iii) Isolation forest

4) Association

We use association algorithm for associating co-occurring items / events.

Association algorithms

- Apriori

5) Autoencoders :- (Dimensionality Reduction)

Autoencoders take input data, compress data into a code, then tried to recreate the input data from summarized code.

→ Autoencoders can remove noise from visual data like images, video / medical scans to improve quality.

Note :-

→ Autoencoders can also be used to find outliers (Anomaly).

③ Reinforcement Learning

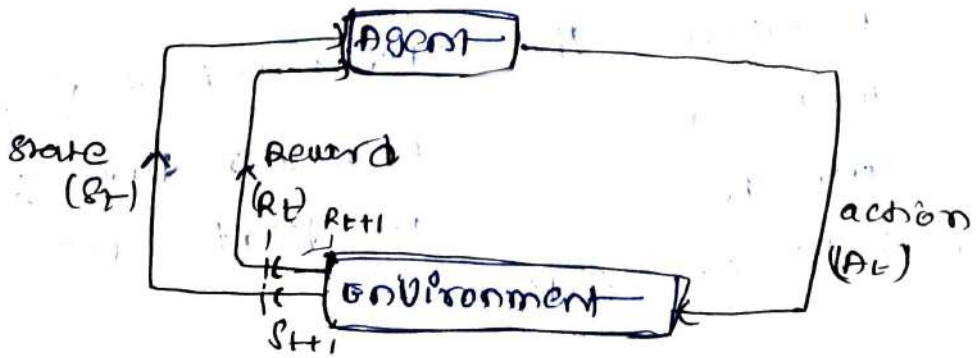
It doesn't require any data as labeled input or labeled output. All it has is the main data.

→ Like, trial & error method. If it fails in a trial then it learns & tries & it learns etc.

→ Games are a real example of RL.
Take a chess game model. All it has is the environment. First it takes a step & sees the outcomes & learns from it and so on.

→ In this kind of RL, AI agents are attempting to find the optimal way to accomplish a particular goal, or improve performance on specific task. As the agent takes action that goes toward the goal, it receives a reward.

overall aim: Predict the best next step to earn the best reward.



when there is no train dataset, it learns from experience.

④ Semi-supervised data

→ Semi-supervised learning is, for the most part, just what it sounds like: a training dataset with both labeled & unlabeled data.

→ this method is particularly useful when extracting relevant features from the data is difficult & labelling examples is a time-intensive task for experts.

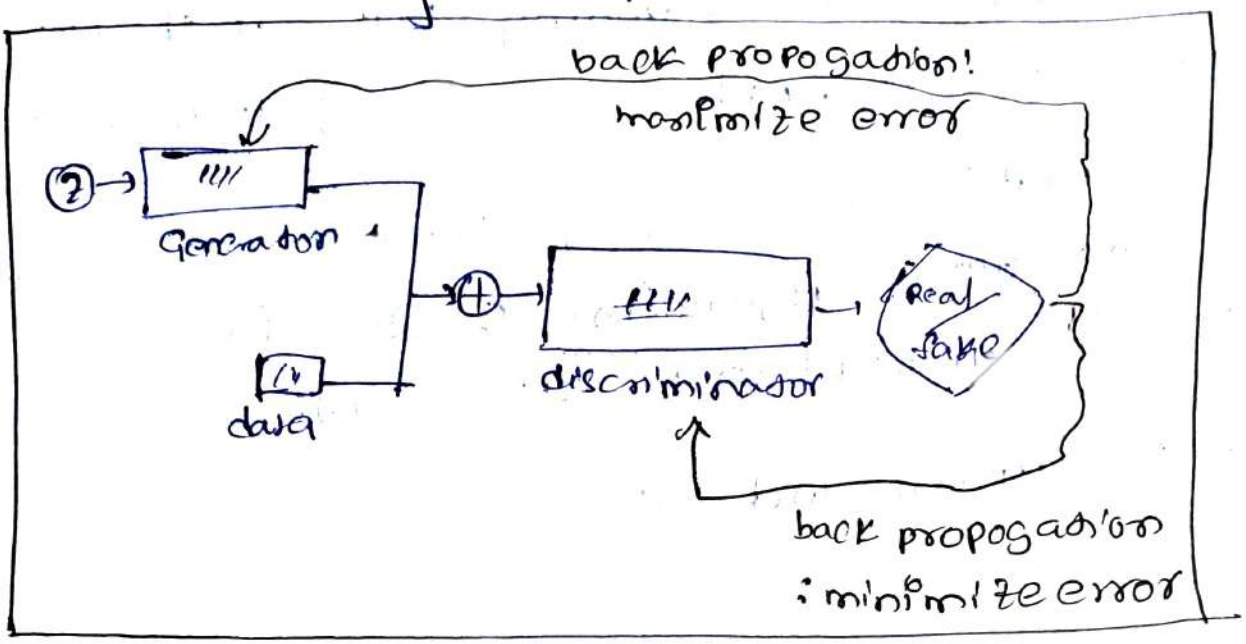
→ A popular training method that starts with a very small set of labelled data is using 'general adversarial networks' / GAN's

like two deep learning networks in competition, each trying to outsmart other. that's a GAN.

→ one of the networks called generator, tries to create new data points that mimic the training data.

→ the other network the discriminator, pulls in these newly generated data & evaluates whether they are part of training data or fake.

→ the networks improve in a positive feedback loop - as the discriminator gets better at separating the fakes from originals, the generator improves its ability to create convincing fakes.



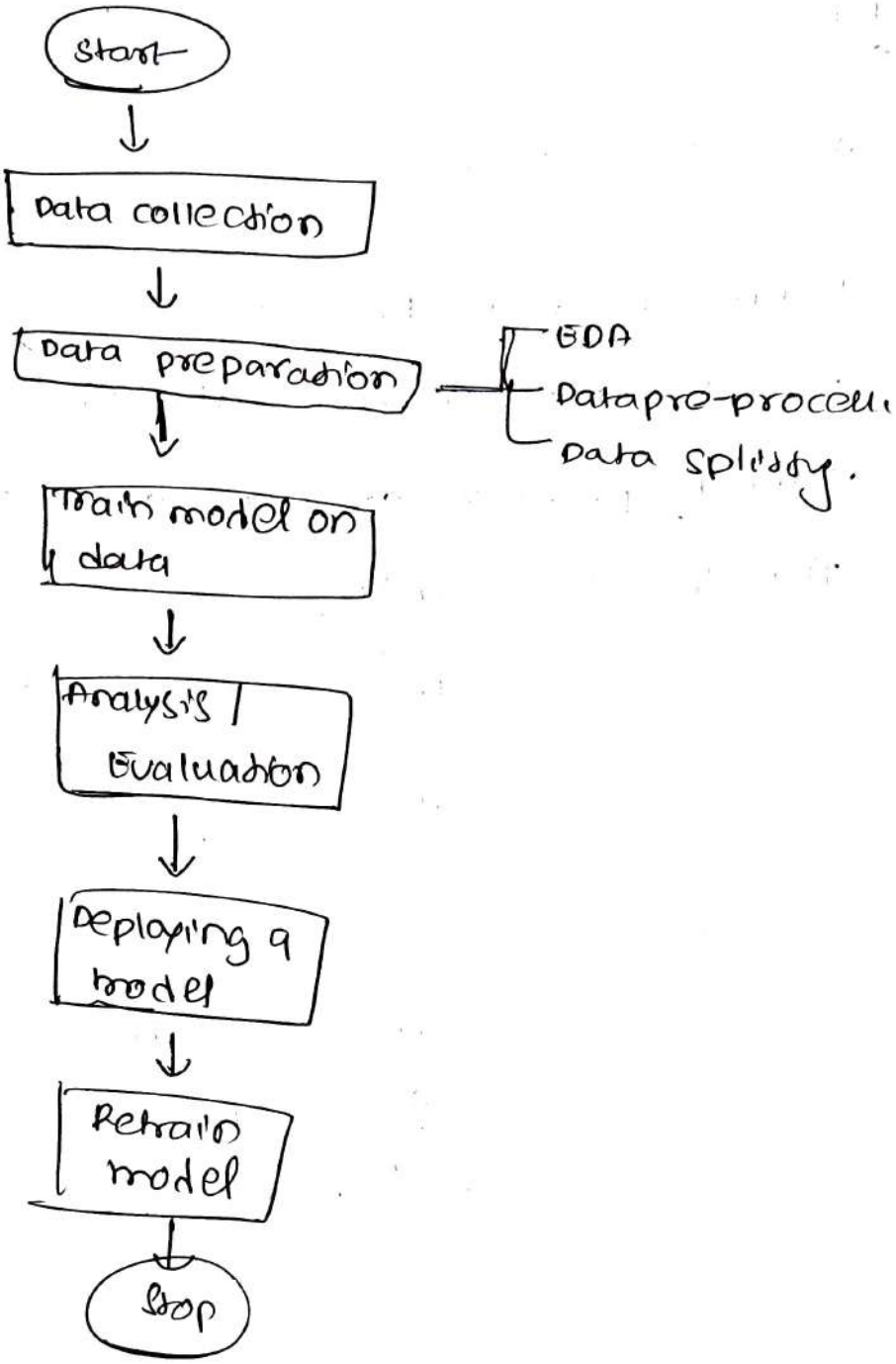
eg:-

→ common situations for this kind of learning are medical images like CT Scan (or) MRI's. A trained radiologist can go through & label a small subset of scans for tumors or diseases.

→ It would be too time-intensive & costly to manually label all the scans — but the deep learning network can still benefit from the small proportion of labeled data & improve its accuracy compared to fully unsupervised models.

→ It's the basic knowledge about algorithms.
→ Before going deep into algorithms.
better to know the flow chart of
machine learning. I feel ML is like a
christopher Nolan movie, first time you
don't understand anything but u should
keep watching, At last everything makes sense
Same works with ML projects, take a small
proj on a algorithm & see it.
you will understand the path.
So, let's understand the path.

ML Road map (Steps in machine learning project)



(*) understanding the process :-

Step (1)

Data collection

a) Questions to ask ?

1) what kind of problem are we trying to solve?
2) what data sources already exist?
3) what privacy concerns are there?
4) is the data public?
5) where should we store the data?

b) understanding the types of data

1) Structured data ?

Data which appear to be tabulated format can contain diff types of data.

like
i) nominal / categorical.
ii) numerical
iii) ordinal
iv) time series

2) unstructured data?

Data with no rigid structure.

(Images, Video, natural language text, speech)

Step (2)

Data preparation

It has 3 main process.

a) Exploratory data analysis (EDA), understanding data.

b) Data preprocessing, preparing your data to be modelling

c) Data splitting.

---

a) EDA, understanding the data

1) what are feature variables (input) and the target variables (output)?

Eg: For predicting heart diseases, feature variable may be a person's age, weight, heart rate etc. And target variable is whether or not they have heart disease

2) what kind of data do you have?
Structured, unstructured, categorical / numerical.

3) Are there missing values? should you remove them / fill them with feature imputation?

4) what are outliers?
How many of them are there?
Are they out by much (3+SD)?
Why are they there?

5) Are there questions you could ask a domain expert about the data?

b) data preprocessing, preparing your data to be modelled

① Feature imputation:-

(Filling missing values)

→ A ML model can't learn on data that isn't there so,

i) single imputation: Fill with mean/median of column

ii) multiple imputation:- model other missing values and fill with what your model finds.

iii) KNN (k-nearest neighbour) :- Fill data with a value from another example which is similar.

iv) many more such as random imputation, last observation carried forward (for time series), moving window, most frequent.

∴ there are ways to handle missing values.

② Feature encoding:-

(Turning values into numbers)

→ A ML model requires all values to be numeric

i) OneHotEncoding:-

Turn all unique values into list of 0's & 1's where the target value is 1 & the rest are 0's.

Eg:- car colors green, red, blue,

a green car's color feature would be [1,0,0]

& a red one would be [0,1,0]

& a blue one [0,0,1].

ii) LabelEncoder :-

Turn labels into distinct numerical values

Eg:
If your target variables are diff animals as dog, cat, bird then 0, 1, 2 respectively

iii) Embedding encoding :-

Learn a representation amongst all the diff data points.

Eg:
A language model is a representation of how diff words relate to each other. Embeddings are also becoming more widely available for structured data.

---

(3) Feature normalization (scaling) or standardization :-

When your numerical variables are on diff scales. (like size of land b/w 1000 - 20000 sq.ft).
In such case some ML models don't perform well. Scaling / standardization help to fix this.

i) Feature scaling (normalization) :-

Shifts your values so they always appear b/w 0 & 1.

* This is done by subtracting min value & divide with range. (value b/w 0 & 1 but numerical relationship is still there)

ii) Feature standardization :-

Standardized all values so they have a mean of 0 & variance of 1.

* this is done by subtracting mean & divide with SD.
* values doesn't end up b/w 0-1 (if variable).
* Standardization is more robust to outliers than scaling.

④ Feature Engineering :-

→ Transform data into potentially more meaningful representations by adding in domain knowledge.

i) Decompose

Like, turn a date (such as 2020-06-18 to 20:16: into hour-of-day, day-of-week, day-of-month, is-holi etc.

ii) Discretization

(Turning larger groups to smaller groups)

- For numerical variables,
  Like age you may want to move it into bucket such as over-50 (under-50, (21-30, 31-40 etc.) this process is known as (BINNING).
- For categorical variables,
  such as car color, this may mean combining colors such as (light-green), (dark-green), (lime green) into a single green

iii) Crossing & Interaction features

(combining two/more features)

→ depends on situation. Sometime adding / subtracting two features could help.

iv) Indicator features :-

like using is-paid-traffic for paid

[using other parts of data to indicate something potentially significa

→ create a x-missing feature for wherever a column x contains a missing value.

⑤ Feature selection :-

→ selecting the most valuable features of your dataset to model.

→ Potentially reducing overfitting & training time (less overall data & less redundant data to train on) & improving accuracy.

i) Dimensionality reduction :-

A common dimensionality reduction method, PCA takes a larger no. of dimension (features) & uses linear algebra to reduce them to less dimensions. For example, say you have 10 numerical features you could run PCA to reduce it to 3.

ii) Feature importance :-

Fit a model to a set of data, then inspect which features were most important to the results, remove the least important ones. (can use "TPOT" for this)

⑥ Dealing with imbalances :-

Does your data have 10,000 examples of one class, but only 100 examples of another?

→ collect more data (if possible)

→ use scikit-learn-contrib imbalanced-learn Package.

→ use SMOTE (synthetic minority over-sampling technique) which creates synthetic samples of your minor class to try & level the playing field.

→ so this is all about Data Preprocessing.

c) Data splitting:

1) training set (usually 90-80% of data):
model learns on this.

2) validation set (typically 10-15% of data):
model hyperparameters are tuned on this.

3) Test set (usually 10-15%): (Dev set)
models final performance is evaluated on this
* Do not use this dataset to tune the model

Q. why shouldn't we tune on test set?

A.
To do the regularization we have to take 100 diff models & try on the test data. And so last we finds (suppose) the best hyperparameter that produces a model with low generalization error, say just 5%.

And you launch it to production & found it produces 15% error.

* Now, here is what happened as you trying every hyperparameter on the same test data eventually your model learned the test-data & fitted to it thus reducing generalization errors but not improving model.
this is called "overfitting"

→ So to avoid we take validation set / cross-validation

Step ③

Train model on dataset

It has 3 main process.

a) choose an algorithm
b) overfit the model
c) Reduce overfitting with Regularization.

a) Algorithm can be choosed by understanding supervised, unsupervised (explained previous)

b) Types of learning :-

i) Batch Learning :-

All of your data exists on a big static warehouse & you train a model on it. You may train a new model once per month once you get new data. Learning may take a while & isn't done often. Runs in production without learning (can be retrained).

ii) Online Learning :-

Your data is constantly being updated & you constantly train new models on it. Each learning step is usually fast & cheap.

Runs in production & learns continuously.

iii) Transfer learning:-

Take the knowledge one model has learned & use it with your own. Gives you the ability to leverage SOTA (state of the art) models for your own problems.
Helpful if you dont have much data / vast compute resources.

iv) Active learning:-

Also referred to as "human in the loop" learning. A human expert interacts with a model & provides updates to labels for samples which the model is most uncertain about.

v) Ensembling:-

Not really a form of learning, more combining algorithms which have already learned in some way to get better results.

Underfitting:-

Happens when your model doesn't perform as well as you'd like on your data. Try training for a longer & more advance model.

---

overfitting :-

Happens when your validation loss (how your model is performing on the validation dataset, lower is better) starts to increase. or if you don't have a validation set,

(*) Happens when the model performs far better on the "training set" than on the "test set".

(eg:- 99% accuracy on training set, 67% accuracy on test set) → overfitting the training set

Fix through various regularization techniques.

Regularization :- [techniques to prevent/reduce overfitting.]

i) L1 (Lasso) & L2 (ridge) regularization :-

→ L1 regularization sets unneeded feature coefficients to 0.

→ L2 contains a model's features (won't set them to 0)

ii) Dropout :-

Randomly remove parts of your model. So the rest of it has to become better.

iii) Early Stopping :-

Stop your model from training before the validation loss starts to increase too much / more generally, any other metric has stopped improving. Early stopping usually implemented in the form of a model callback.

iv) Data Augmentation :-

manipulate your dataset in artificial ways to make it harder to learn.

→ For example, if you're dealing with images, randomly rotate, skew, flip & adjust the height or your images.

→ this makes your model have to learn similar patterns across diff styles or the same image (harder).

→ use functions like ImageDataGenerator in keras or transforms in torchvision.

v) Batch Normalization :-

Standardize inputs as well as adding two parameters (beta, how much to offset the parameter for each layer & epsilon to avoid division by zero) before they go into next layer this often results in faster training speed since the optimizer has less parameters to update.

maybe a replacement for dropout in some networks.

Hyperparameter Tuning

→ Run a bunch of experiments with diff model settings & see which works best.

a) Setting a learning rate:-

(often the most important hyperparameter)

Generally

High learning rate = algorithm rapidly adapts to new data

Low learning rate = algorithm adapts slower to new data (est- for transfer learning)

i) Finding the optimal learning rate:-

Train the model for a few hundred iterations starting with a very low learning rate (10^{-6}) & slowly increase it to very large value (e^{1}) then plot the loss vs learning rate (using a log-scale for learning rate), you should see a U-shaped curve, the optimal learning rate is about 1-2 notches to the left of the bottom of U-curve. (P326 - Hands on ML)

ii) Learning rate scheduling involves decreasing the learning rate slowly as model learns more (gets closer to convergence). [use Adam optimizer]

iii) Cyclic learning rates

Dynamically change the LR up & down b/w some thresholds & potentially speed-up training.

Other hyperparameter you can tune

1) No. of layers (Deep learning networks)
2) Batch size [How many eg or data your model sees at once.]
3) No. of trees (Decision tree algo)
4) No. of iterations [How many times model goes through the data]
.. Many more ..
Depends on algo used. (can use early-stopping)

Step ④

Analysis / Evaluation:- to estimate how well a model will generalize to out-of-sample data.

a) Evaluation metrics:-

i) classification
- ROC & AUC curve
- Accuracy
- Precision
- Recall
- F1 score
- confusion matrix.

ii) Regression
- MSE (mean squared error)
- MAE (mean absolute error)
- R2 (r-squared)

iii. Task-based metric (create one based on your specific problem).

---

1) **Classification accuracy:-**

Percentage of correct predictions. $\left(\frac{Y\_pred}{Y\_test}\right)$

Null accuracy:-

Accuracy that could be achieved by always predicting the most frequent class.

→ It's a good way to know the min. we should achieve with our models.

→ It gives a baseline to compare our model's accuracy.

∴ classification accuracy is easy, but it doesn't tell you the underlying dist of response class.

→ And it doesn't tell you what type of error your classifier is making.

a) confusion matrix

FP - False Positive
FN - False negative.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

TP: which are actually correct

TN: which are actually wrong

FP: they are actually wrong, but predicted as true. ← they actually don't have corona. But we predicte they have.

FN: they are actually true, but predicted as false ← they actually have, but we say they don't

→ Classification accuracy ← $\frac{TP+TN}{TP+TN+FP+FN}$ talks about correct.

How often is the classifier correct?

→ misclassification rate ← $\frac{FP+FN}{TP+TN+FP+FN}$

Recall / sensitivity [True positive rate]

→ How "sensitive" is the classifier to detecting positive instances?

→ when the actual positive is tested, How often it is correct?

Recall = $\frac{TP}{TP+FN}$

Out of total positive ones, How many positive I am able to (Recall)

→ It depends on FN, which were actually true, but predicted false.

Actual Pos - TP+FN

Predicted Pos. (TP)

Precision & Recall linked with TP

Precision = Really Pos / Total Pos: actually

$$\frac{TP}{TP+FP}$$

$$\frac{TP}{TP+FN} = \frac{\text{Really Pos}}{\text{Predicted Pos}}$$

→ when a positive value is predicted, How often is the prediction correct?

→ How **"precise"** is the classifier when predicting positive instances.

out of predicted pos, how many really pos.

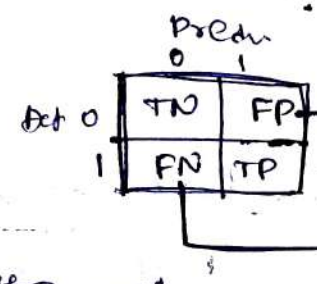
$$\text{Precision} = \frac{TP}{TP+FP}$$

→ It depends on FP, which were actually negative, but predicted true.

Q.) which metrics to focus on?

Eg ①

1 = COVID 19 (+)
0 = − (Healthy)



→ Now analyze the cost

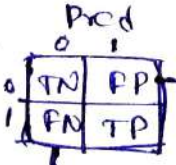
→ we know a sick predicted as healthy could spread a lot more cases.

So, **cost of FN > cost of FP**

Hence it depends on FN, we go for "Recall".

Eg ②

1 = Spam
0 = Not Spam



→ we know if a not spam (imp. mail) goes to spam, that could cause lot of damage.

So, **cost of FP > cost of FN**

Hence, it depends on FP, we go for "precision"

Specificity :-

$$\frac{TN}{TN+FP}$$

False positive rate :-

$$\frac{FP}{TN+FP}$$

FB-Score :-

when you want to consider both FP & FN like we need both sensitivity (recall) & precision, in such cases we choose F-Beta.

General Formula

$$F_\beta = (1+\beta^2) \cdot \frac{\text{Precision} \times \text{recall}}{(\beta^2 \cdot \text{Precision}) + \text{recall}}$$

$$F_\beta = \frac{(1+\beta^2) \times TP}{(1+\beta^2) \times TP + \beta^2 \cdot FN + FP}$$

In beta, there is posibility for 3 ranges.

1) when FP & FN are equally important, like you can't compromise even a bit both are crucial, then we set beta = 1 which is F1-score.

$$F1\text{-Score} = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

2) when both are worse, but FP is drastic than FN. then we go for beta = 0.5 (usually 0-1)

3) when both are worse, but FN is drastic than FP. then we go for beta = 2 (usually 1-10).

---

note

we can also adjust classification threshold to modify performance of classifier.

Roc curve

can see how steadily & sensitivity are affected by various thresholds, without actually changing the threshold.

AUC curve (Area under curve)

If you randomly choose one + & one -, AUC represents the likelihood that your classifier will assign a higher predicted probability to the positive observation.

Analysis of model

1) Feature importance

Which feature contributed most to model
Should some be removed?

2) Training / Inference time

How long does a model take to train?
Is this feasible?

How long does inference take?
Is it suitable for production?

3) Using "what-if" tool

what-if changed thing in data?
How does this effect the outcome?

4) Bias / variance trade-off

→ High bias results in underfitting &
a lack of generalization to new samples

→ High variance results in overfitting due
to the model finding patterns in the
data which is actually random
noise.

Step (5)

Deploying a model

Put the model into production & see how it goes. Evaluation metrics in notebooks are great but until it's production, you won't know how it performs for real.

Step (6)

Retrain model

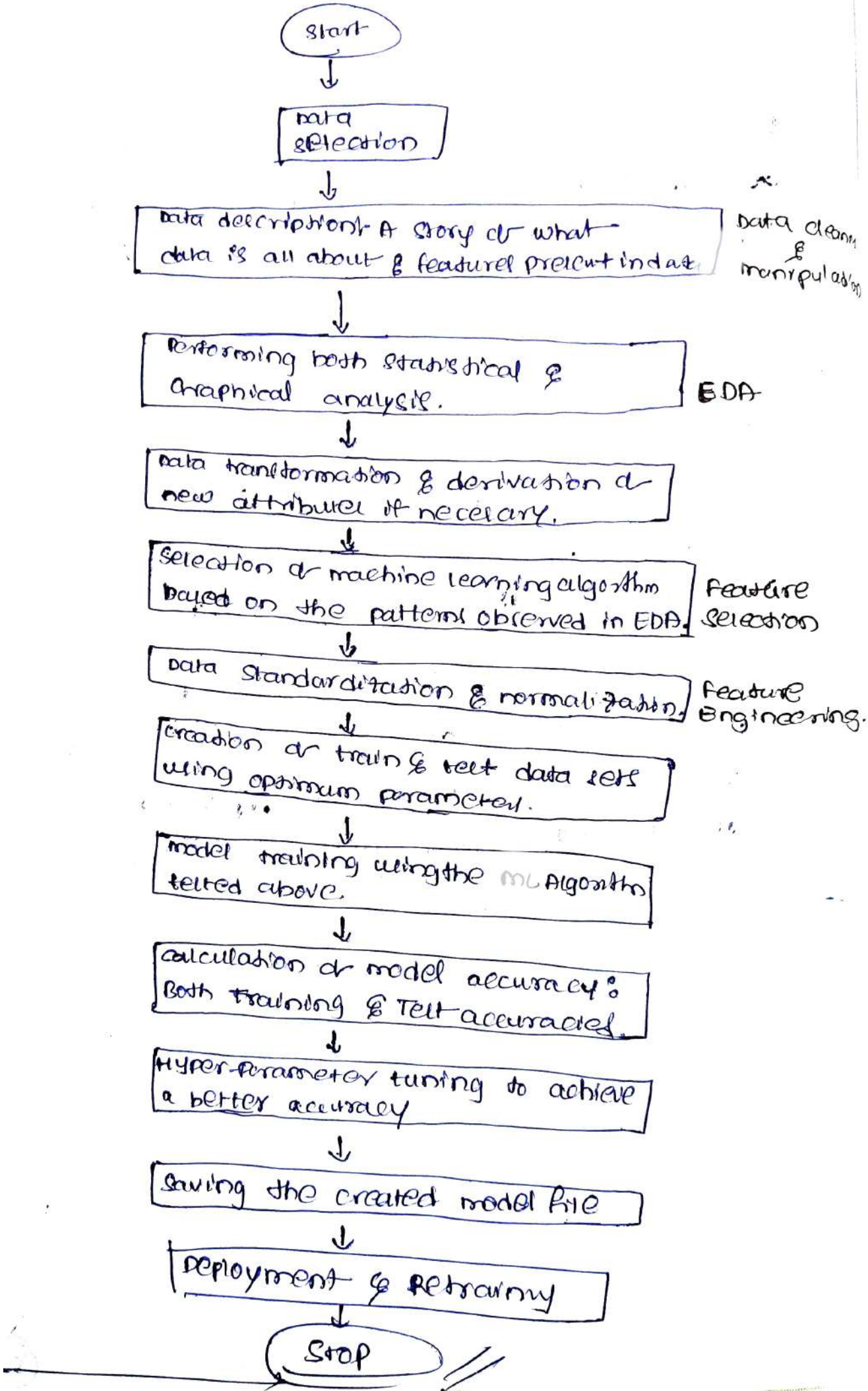
→ see how the model performs after serving based on various evaluation metrics & revisit the above steps as req.

→ you will find your models prediction start to cages or (drift), as in when data sources change or upgrade. this is when you will want to retrain it.

(0,0)

Stop / NO END

So, basically the flowchat is ---



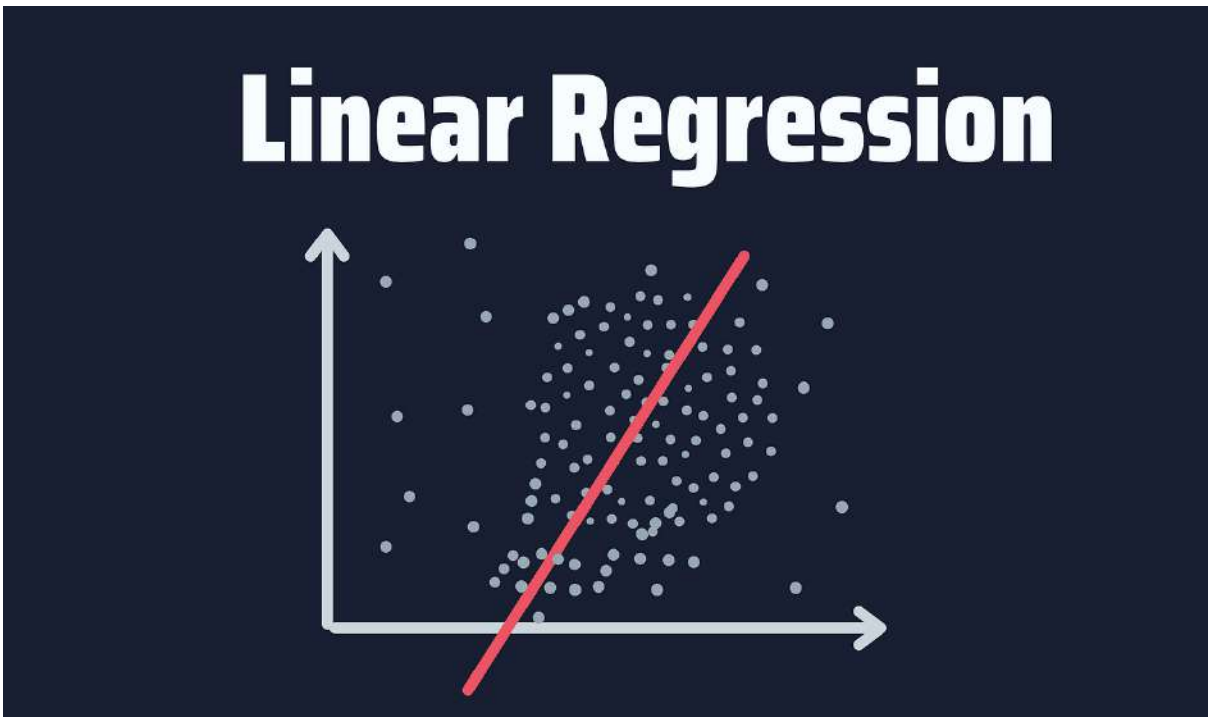


Table of Contents

Understanding the algorithms

① Simple Linear Regression

Linear model in Regression (supervised learning)

Simple linear regression assumes that a linear relationship exists b/w the response variable & the explanatory variable, It models this relationship with a linear surface called a "hyperplane"

→ This will be clear with an eg.
Let's take the height & weight sample of few people

Height	weight
5.6	60
5.7	62
5.8	63
5.9	62
6.1	64
6.2	75

$H \propto W$

$W \propto H$

Now, If I want to know what would be the weight of a person with 6.0 Height?

→ By observation it seems like

$H \propto W$ → to remove proportionality we can multiply a constant

$W = mH$ → But, it may also differ by some value, so, $\pm b$

$W = mH \pm b$ → Now, this is the relation of height & weight.

constants

$w = mH \pm b$ → Now, here we know the value of 'm' & 'b'. then we can predict the weight.

given height

Now, main aim is to find out value of 'm' & 'b'.

→ Let's take 2 samples, (5.6, 60), (5.7, 62),

$60 = m(5.6) + b$ — (1)

$62 = m(5.7) + b$ — (2)

→ By this we get $m = 20$, $b = -52$.

So, our equation $w = 20H - 52$

Let's try with a known height 5.9.

$w = 20(5.9) - 52 = 66$ kg.

Here we got '66', but we are expecting 62.

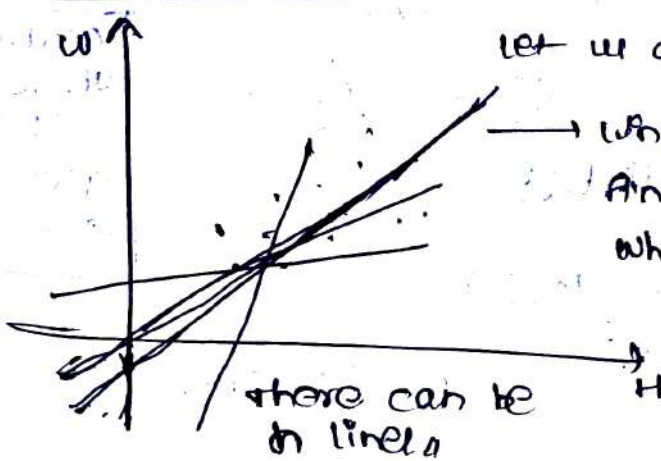
(*) So, may be if we change 'm' & 'b' then we could get a exact value / approx.

Now, Here we get the definition of ML as **establishing a mathematical relationship** of data. So, that it can predict a new data.

→ So, when we say a ML model,

(*) It is nothing but a mathematical relationship which can predict new data.

Plotting 'H' & 'w'



let us assume we have a best fit line.

→ Linear regression always tries to find the linear relation b/w variable which is a straight line.

Hence, $y = mx \pm c$

→ we could find n diff lines.

But that one line which can predict all the values approx. is the "Best fit line"

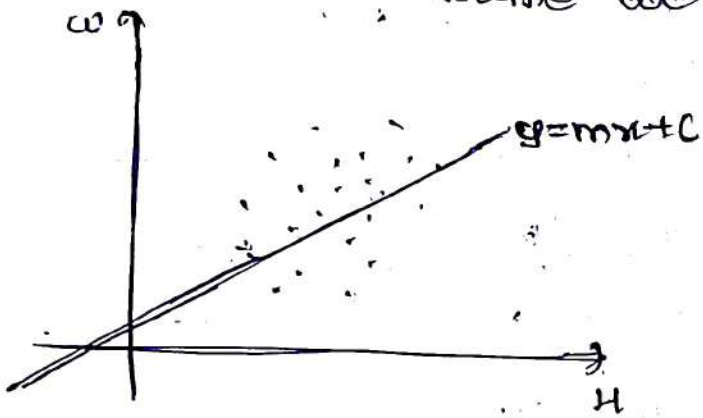
$y = mx + c$ → And this best fit line is called model

→ we have to find the 'm' & 'c'.

→ m-slope, c-Intercept

(*) Evaluating the fitness of the model with a cost function

→ Firstly, let us assume we have a best fit line.



But, now we have doubts.

How did we get this line?
why this line?
what about other lines?

→ Now, if it's the best fit line, then for future if I want the weight of '6.0'.

→ It is nothing but the value of y-coordinate $y = mx + c$ line with x coordinate '6.0'

→ If this line is $W = 20H - 52$ and at $x = 5.9$,

I got $\hat{w} = 66$, but actual $w = 62$.

So, I can see there is a loss / error.

$$\text{loss} = |(y - \hat{y})|$$

(y known, $\hat{y}$ predicted)

Always there will be a slight error

Q.) why did we got this loss?

A may be best of 'm' & 'c' we taken.

Gradient Descent - 48

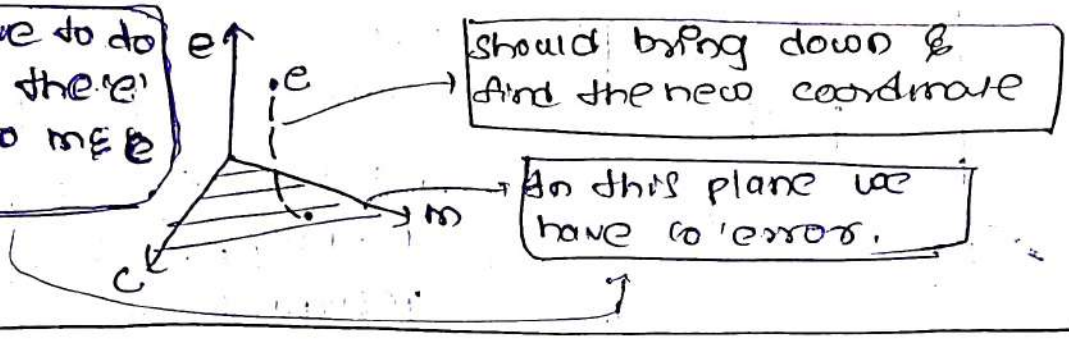
→ Now, to improvize my model strength I have to 'reduce the loss'.

and this loss depends on 'm' & 'c' taken.

→ So, I am supposed to shift m & c such that I end up reducing the loss.

we have loss (e), slope (m), constant (c).

So I have to do is bring the 'e' down to m & c plane



Slowly, by examining each input m & c gets changed and eventually loss decreased. this process is called (learning).

So,

$$m_{new} = m_{old} - \eta \Delta E_m$$

$$c_{new} = c_{old} - \eta \Delta E_c$$

m_{new} - changed one, new

m_{old} - old m value.

η - Learning rate

ΔE_m - Error in m $\left(\frac{de}{dm}\right)$

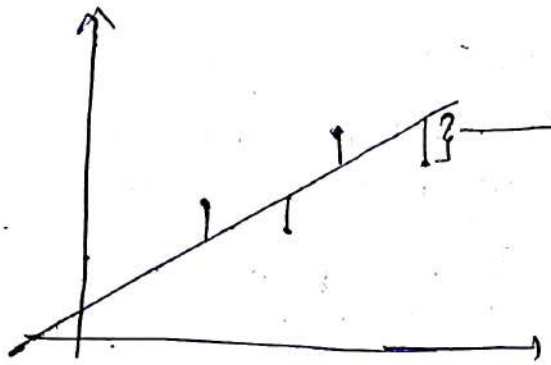
$\eta \rightarrow (0.0001 - 10)$ range.

→ So, here we are taking a new m.

By subtracting a fraction of the error rate from old m that fraction is learning rate.

So, now I am supposed to choose the m & c, which gives a least possible error then it will be the best fit line.

→ A cost function also called a loss func, is used to define & measure the error of a model. the diff. b/w the weights predicted by model & the observed weights in training set are called "residuals" or training error / loss.



→ this is the residual / loss. which is $|y-\hat{y}|$

→ we have to reduce this residual Actually, we have to reduce residual of all data points.

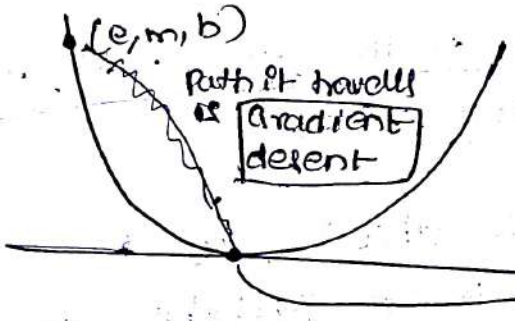
→ we can produce best weight predictor model by minimizing the 'sum of residuals'.

(Sum of residuals) $\sum_{i=1}^{n} r_i = \sum_{i=1}^{n} [y_i - (mx_i + c)]$ → $\hat{y}$

So, to cancel the negative value, we take square

(*) $R(x) = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} [y_i - (mx_i + c)]^2$ sum of squares of residuals.

Now, it is a quadrant in 3D.



→ So, we are supposed to shift the (e, m, b) to the origin (0, m, b).

where $\frac{de}{dm} = 0$, $\frac{de}{db} = 0$.

As we can see residual is both a func. of m & b, so, differentiating partially w.r.t m & b will give us

$$\frac{\partial E}{\partial m} = \sum_{i=0}^{n} x_i (b + m x_i - y_i)$$

$$\frac{\partial E}{\partial b} = \sum_{i=0}^{n} 2 (b + m x_i - y_i)$$

So, we know for best line, residual should be min. minima of func. occurs where derivative = 0. i.e,

$$\sum_{i=0}^{n} x_i (c + m x_i - y_i) = 0$$

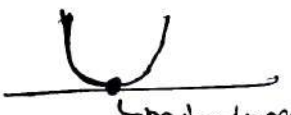
$$\sum_{i=0}^{n} 2 (b + m x_i - y_i) = 0$$

$$\boxed{\begin{aligned} \sum_{i=0}^{n} x_i c + \sum_{i=0}^{n} m x_i^2 + \sum_{i=0}^{n} x_i y_i = 0 \\ \sum_{i=0}^{n} c + \sum_{i=0}^{n} m x_i - \sum_{i=0}^{n} y_i = 0 \end{aligned}}$$

The same eqn. can be written in matrix form as:

$$\begin{bmatrix} \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 \\ n & \sum_{i=0}^{n} x_i \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} x_i y_i \\ \sum_{i=0}^{n} y_i \end{bmatrix}$$

Ideally, if we'd have an eqn. of one dependent & one independent variable the minima would look like



dependent var $\boxed{y = mx + c}$ independent var

→ the new values for slope & intercept are calculated.

repeat until convergence()

{

$$m_{new} = m_{old} - \eta \left[\frac{1}{n} \sum_{i=1}^{n} (h_0 \cdot x_i - y_i) x_i \right] \quad \leftarrow \frac{dE}{dm}$$

$h_0 = m$

$$c_{new} = c_{old} - \eta \left[\frac{1}{n} \sum_{i=1}^{n} (h_0 \cdot x_i - y_i) \right] \quad \leftarrow \frac{dE}{dc}$$

}

Accuracy → this is how LR works.

(*) measuring model strength

RSS - Residual Sum of Squares

TSS - Total sum of Squares

The R^2-squared (R^2) statistic provides a measure of fit.

It takes the (form of proportion) − (proportion explained).

$$\boxed{R^2 = 1 - \frac{RSS}{TSS}}$$

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad \leftarrow \text{Predicted}$$

because changing

$$TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2 \quad \leftarrow \text{mean.}$$

If $R^2 = 0.75$ it says that our model fits 75% of dataset.

Adjusted R^2 Statistic

→ As R^2 is just a linear eqn., as we increase the no. of independent variables in our eqn., the R^2 increases as well.

But, that doesn't mean that the new independent variables have any correlation with the output variable.

i.e; R^2 will increase, but is not necessary model yields better results.

→ To rectify this we use adjusted R^2 value which penalises excessive use of such features which don't correlate with the output data.

$$R^2_{adj} = 1 - \frac{(1-R^2)(N-1)}{N-P-1}$$

where

P - no. of predictors

N - total sample size.

If $P=0$, $R^2_{adj} = R^2$

Note:-

→ using training data to learn the values of the parameters for simple linear regression that produce the best fitting model is called ordinary least squares (OLS) (or) linear least square.

Another approach to find m & C.

(To solve OLS)

→ variance is a measure of how far a set of values are spread out.

→ covariance is a measure of how much two variables change together. If the variables increase together, their cov is positive. 0 - no relation, -ve → one increase & one decrease

$$var = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

→ variance of explanatory variable.

$$covar = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

→ covariance of the response, explanatory variables.

$$\therefore \beta = \frac{cov(x,y)}{var(x)}$$

$y = \beta x + \alpha$.

$\alpha = \bar{y} - \beta \bar{x}$ → we can get α

Notes

→ The independent variables are uncorrelated with the residual term, also known as "Exogeneity". This in layman term generalises that in no way should the error term be predicted given the value of independent variables.

the error terms have a constant variance.
i.e) Homoscedasticity.

→ the error terms are normaly distributed.

→ No multicollinearity, i.e; no independent variables should be correlated with each other or affect another.

→

model confidence

Q) Is linear regression a low bias / high variance model (or) a high bias / low variance model?

A) It's a "High bias / low variance" model

→ Even after repeated sampling, the best fit line will stay roughly in the same pos. (low variance)

→ But the avg. of the models created after repeated sampling won't do a great job in capturing the perfect relationship. (High bias)

→ Low variance is helpful when we don't have real training data.

```
import statsmodels.formula.api as smf
model_name = smf.ols(formula='y ~ x', data=df).fit()
model_name.conf_int()
```

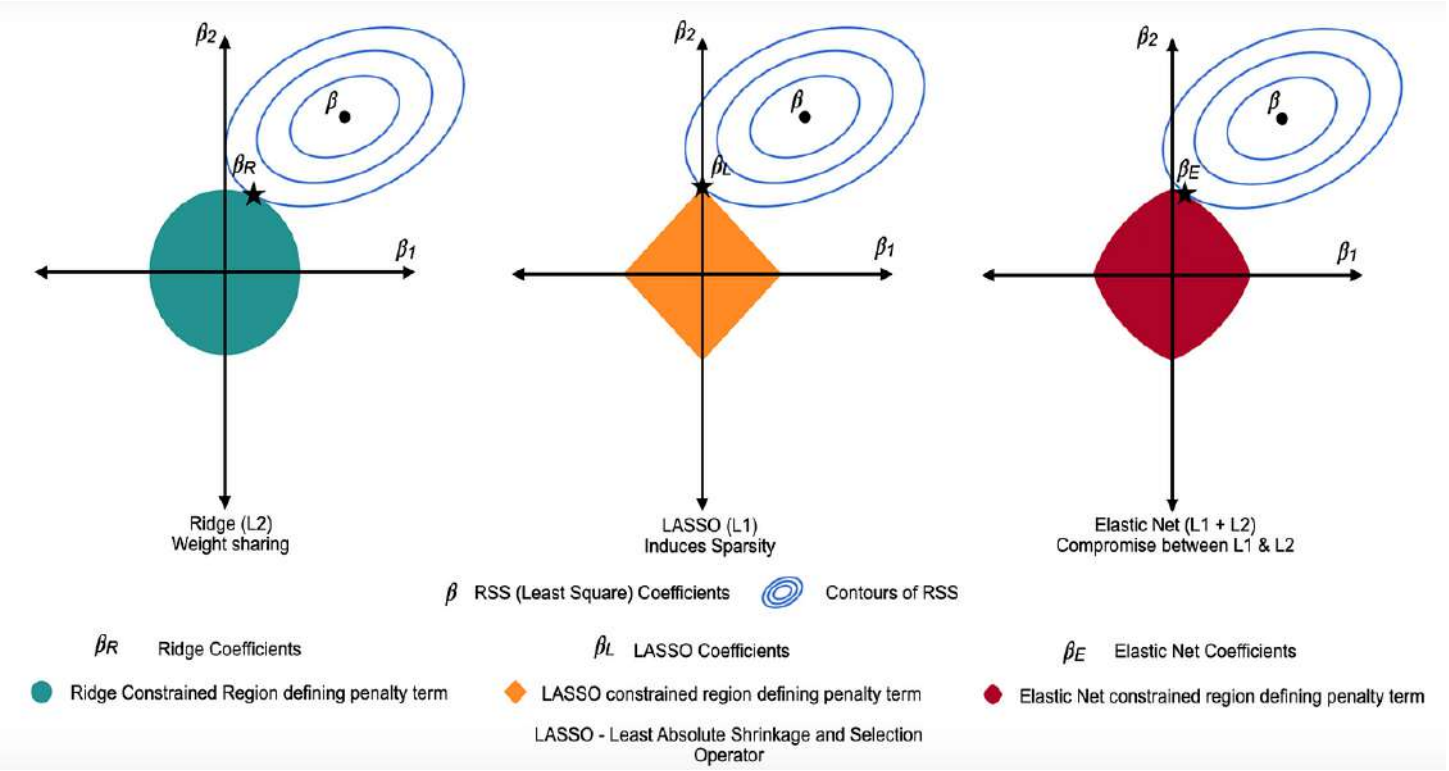


Table Of Contents

(*) Multi collinearity

→ we can define multi-collinearity as the situation where the independent var have strong correlation among themselves.

→ the coeff. in a linear Reg model represent the extent of change in 'y' when a certain $x(x_1, x_2, x_3 \ldots)$ is changed keeping other constant. But if x_1 & x_2 are dependent then this assumption itself is wrong that we are changing one var keeping other constant as the dependent var will also be changed.

It means model becomes a bit flawed.

$y = mx_1 + nx_2 + c$

while,

$x_2 = ax_1 + d$

Independent var (x_2, x_1) are related.

More than one when eqn,

Hence called **multi collinearity** //

→ we have redundancy in our model as two variables (or more than two) are trying to convey the same information.

→ As the extent of collinearity increases, there is a chance that we might produce an "overfitted model".

→ An overfitted model works well with the test data but its accuracy fluctuates when applied to other datasets.

© can result in a dummy variable trap.

→ By the heatmap we can check collinearity.
Generally a correlation greater than 0.9 (or) less than -0.9 are to be avoided.

(#) variance Inflection Factor (VIF) :-

Regression of one X var against other X variables.

$$VIF = \frac{1}{1-R^2}$$

VIF < 5 ✓ perfect

→ If VIF > 5, extreme correlation is avoided.

Remedies for multi-collinearity

(1) Do-nothing :- If corr not extreme / the var not used the ignore

(2) Remove one-var :- like in dummy var trap (onehotencoding)

(3) combine correlated var :- combine variables.

(4) Principle componet Analysis (PCA)

(*) Regularization :-

→ When we use Regr. models to train some data, there is a good chance that the model will overfit the given training dataset.

→ Regularization helps sort this overfitting problem by restricting the degree or freedom of given eq.

Eg)
Simply reducing the no. of degree of a polynomial by reducing their corresponding weight.

Reason for Regularization

→ In a linear eqn, we don't want huge coeff, as a small change in coeff can make a large diff. So regular constraints weights or such features to avoid overfitting.

$$y = B_0 + B_1 x_1 + B_2 x_2 + \cdots B_p \cdot x_p$$

$$RSS = \sum_{i=1}^{n} \left(y_i - B_0 - \sum_{j=1}^{p} B_j \cdot x_{ij} \right)^2 \quad (y_i - \hat{y})$$

* To Regularize a model, Shrinkage penalty is added to cost function.

LASSO (Least Absolute Shrinkage & Selection Operator) (L1 form)

→ LASSO regression penalized the model based on the sum of magnitude of the coefficients.

The regularization term is given by, $\text{regularization} = \lambda * \sum |B_j|$

(λ — shrinkage factor)

→ Loss after regularization is:-

New loss func. $RSS + \lambda \cdot \sum_{j=1}^{p} |B_j| = \sum_{i=1}^{n} (y_i - B_0 - \sum_{j=1}^{p} B_j \cdot x_{ij})^2 + \lambda \cdot \sum_{j=1}^{p} |B_j|$

Here, loss is not calculated just by previous values $(y_i - \hat{y})$, now it has been split into several parts which decreases loss.

Ridge Regression (L2 form)

→ Ridge Regression penalized the model based on the sum of squared of mag of coeff.

$\text{regularization} = \lambda * \sum |B_i^2|$

λ — can be calculated by cross validation

→ loss after regularization,

$$RSS + \lambda \cdot \sum_{j=1}^{p} B_i^2 = \sum_{i=1}^{n} (y_i - B_0 - \sum_{j=1}^{p} B_i \cdot x_{ij}) + \lambda \cdot \sum_{j=1}^{p} B_i^2$$

Note:- consider B_1 & B_2 be coeff of LR & $\lambda = 1$.

(1) For LASSO, $|B_1| + |B_2| \leq S$

(2) for Ridge, $B_1^2 + B_2^2 \leq S$

→ where 'S' is the max value the eqn can achieve. At

Elastic net

→ middle ground b/w Ridge & Lasso.

→ the regularization term is a simple mix of both Ridge & Lasso, and you can control mix ratio α.

$$\text{Lenet}(\vec{\beta}) = \frac{\sum_{i=1}^{n}(y_i - x_i\vec{\beta})^2}{2n} + \lambda\left[\left(\frac{1-\alpha}{2}\right)\cdot\sum_{j=1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j=1}^{m}|\beta_j|\right]$$

→ when 'α' is the mixing parameter.

Ridge ($\alpha=0$)

Lasso ($\alpha=1$) //

(*) difference between Ridge & LASSO

→ Ridge Regression shrinks the coeff for those predictors which contribute very less in the model but have huge weights, very close to 0.

$y = 0.04x + 1e4$

$R^2 = 0.0016$ — very less

thus final model will still contain all those predictors, though with less weights.

→ where as in LASSO, the L1 penalty does reduce some coeff exactly to zero, when we use a sufficiently large dummy parameter λ.

So, in addition to regularizing, LASSO also performs feature selection.

Q- Why use Regularization?

A → It helps to reduce the variance of the model, without a substantial increase in the bias.

→ If there's variance in the model that means the model won't fit well for dataset other than training data (which is called overfitting).

→ the tuning parameter (λ) controls this bias & variance trade off. selected using cross-validation

→ when (λ) is increased to certain limit it reduces the variance without losing any important data.

→ But after certain limit model will start losing important prop which will increase (bias) in the data.

Q) what should be used plain linear, Ridge, LASSO or Elastic net?

A)

→ It is always preferable to have atleast a little bit of Regularization, so generally avoid plain linear Reg.

→ Ridge is a good default,

→ But if you suspect that only a few features are actually useful, you should prefer LASSO/EN, since they tend to reduce the useless features weights down to zero.

→ In general Elastic net is preferred over LASSO since LASSO may behave erratically when the no. of features is greater than no. of training instances or when several features are strongly correlated.

whenever, mean is not 0.
it's better to do StandardScaler

Note:-

① StandardScaler() → $\frac{x-\bar{x}}{SD}$ $\bar{x}=0$ $SD=1$

② Two ways to find multicollinearity → correlation & VIF

③ 4 ways to remove multicollinearity.

④ train set, high accuracy; test set, less accuracy.
i.e. overfitting, so regularize it.

→ Even after checking Lasso and Ridge, if it gives the same accuracy values, then you can't is not over-fitted.

i.e. just by having huge difference in train accuracy & test accuracy than it doesn't just mean overfitted. test before judge. on this case model is not problem, more data is needed.

⑤ The linear in Linear Regression does not talk about the degree or polynomial eq in terms of dependent variables.

instead, it talks about the degree of coeff.

$y = a + bx + cx^2 + \cdots nx^n$

it's not talking about power of x, but the power of a, b, c etc. And their powers 1.

Hence it is linear Regression.

Polynomial Regression

→ It is a mechanism to predict a dependent var based on the polynomial relationship with the independent variable.

In the eq, $y = a + bx + cx^2 + \dots nx^n + \dots$

max. power of x is called degree of polynomial.

∴ Deg ① → $y = a + bx$

Deg ② → $y = a + bx + cx^2$

When to use?

⊛ If data is scattered in a polynomial curve shape (not linear) then keep trying different degree until the line fit to data.

Codes

```
from sklearn.preprocessing import PolynomialFeatures
Poly_reg = PolynomialFeatures(degree=2)
X_Poly = Poly_reg.fit_transform(X)
```

→ Keep changing degree until it fit to data.

✱ Logistic regression (it not Regression model)

⑧ Logistic Regression

Linear model for classification (supervised learning)

✱ (Probability based model)

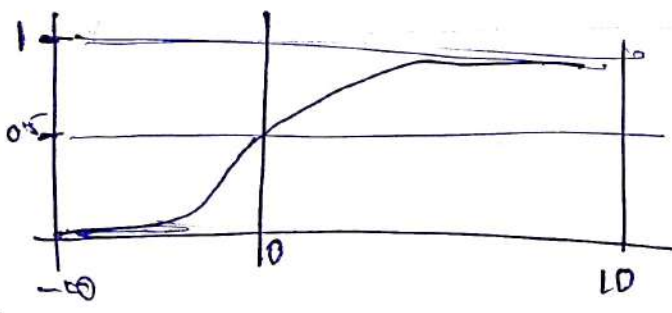
→ Logistic Regression is used for performing "classification problems".

→ It calculates the probability that a given value belongs to a specific class. If the probability is more than 50%, it assigns the value to that particular class,
else if probability is less than 50%, the value is assigned to the other class.

∴ we can say that logistic Regr. acts as a binary classifier.

✱ Here, As we give input x, we get y, where y is the probability or given var x belonging to a certain class.
thus, it is obvious that the value of y should we between 0 & 1.

So, we use Sigmoid function as the underlying function in Logistic Regression.



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

Q. Why we sigmoid?

① the sigmoid function's range is bounded between 0 & 1, which is what we need.

② Easy to calculate than other functions which is useful during gradient descent calculation.

③ It is a simple way to introduce non-linearity to model

→

the logistic func is given as,

$$P(x) = \frac{e^{B_0 + B_1 x}}{1 + e^{B_0 + B_1 x}}$$

we have $B_0 + B_1 x$

$$P(x) = \frac{1}{1 + e^{-(B_0 + B_1 x)}} = \frac{e^{B_0 + B_1 x}}{1 + e^{B_0 + B_1 x}} \quad -①$$

$$1 - P(x) = 1 - \frac{e^{h(\theta)}}{1 + e^{h(\theta)}} \quad (h(\theta) = B_0 + B_1 x)$$

$$1 - P(x) = \frac{1}{1 + e^{h(\theta)}} \quad -②$$

Now Divide ①, ②

$$\frac{①}{②} \Rightarrow \frac{P(x)}{1 - P(x)} = e^{h(\theta)} = e^{B_0 + B_1 x}$$

$$B_0 + B_1 x = \log\left(\frac{P(x)}{1 - P(x)}\right)$$ → logarithmic function.

Now we have,

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x$$

(logit function)

If we input (x), we will get $p(x)$ (probability)

→ And we get $p(x)$, based on the line with β_0, β_1 constants.

This is how logistic regression works

Here, if we have two classes $(0, 1)$

$$y = \begin{cases} 0 & , \text{if } p(x) < 0.5 \\ 1 & , \text{if } p(x) \geq 0.05 \end{cases}$$

0.5 – threshold

we can change threshold

→ Based on the scenario, we should change threshold.

Cost function / loss function

For a single training instance,

$$\text{cost}\left(p(x), y\right) = \begin{cases} -\log(p(x)) & , \text{if } y = 1 \\ -\log(1-p(x)) & , \text{if } y = 0 \end{cases}$$

If $p(x) = 1$ & $y = 1$, cost / loss $= 0$

If $p(x) = 0$ & $y = 0$, cost / loss $= 0$

But,

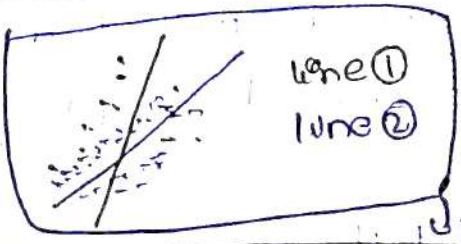
If $p(x) = 0$ & $y = 1$, cost / loss $\to \infty$

If $p(x) = 1$ & $y = 0$, cost / loss $\to \infty$

→ cost func for whole training set is given as

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} \left[y_i \cdot \log(\hat{p}_i) + (1-y_i) \cdot \log(1-\hat{p}_i) \right]$$

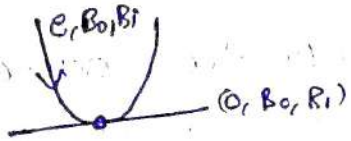
where, m - rows
y_i - expected y
$\hat{p}_i$ - predicted y



→ In logit func, RHS we have a line, And we will keep on adjusting the line w.r.t error so that we'll get correct probability // for from data

→ the values of parameter (θ) for which the cost func is (min) is calculated using the gradient decent algorithm (as showed in LR).
the partial derivative for above cost func is,

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[\sigma(\theta^T \cdot x_i) - y_i \right] \cdot x_j$$



multinomial logistic Regression

→ we can extend Logistic Regression for multi-class classification. the logic is, we train our model for each class and calculate the probability that a specific feature belongs to that class. once we have trained the model for all the classes, we predict a new values class by choosing that class for which prob (class) is maximum.

* we rarely use it this way, because we have many other classification models for these scenarios.

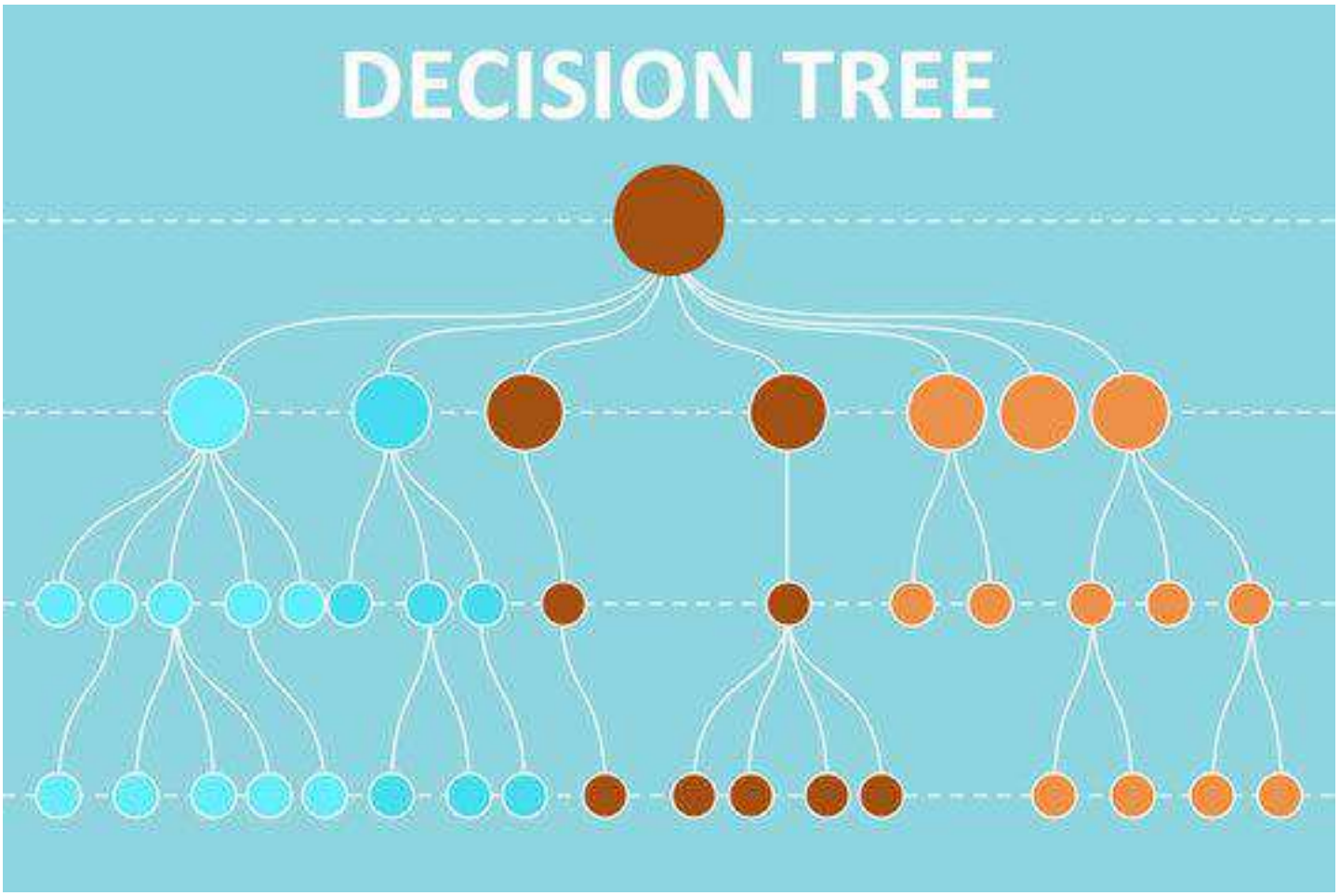


Table Of Contents

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)

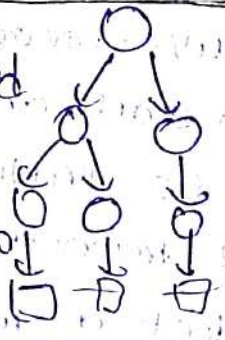
→ But in case of classification models. we have this logistic regression which can only classify by using lines.

* 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

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

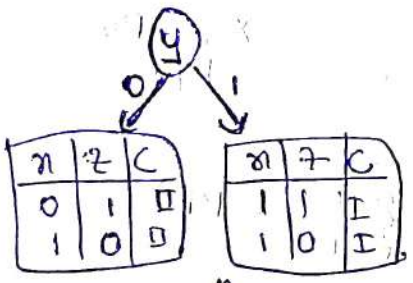
(S(1): no. of 1's; S(0): no. of 0's; S: total no. of values in x)

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

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

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

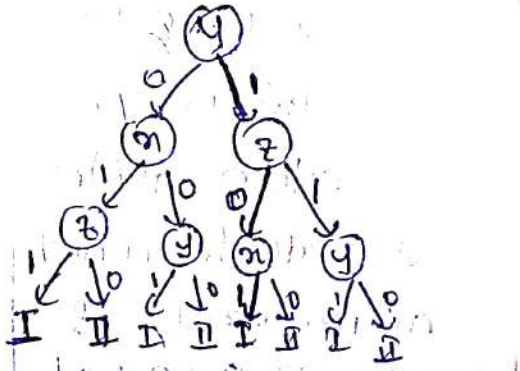
$G_y = 1$, is the highest info gain so local entropy. Hence it is 'Root Node: (y)'



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		

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

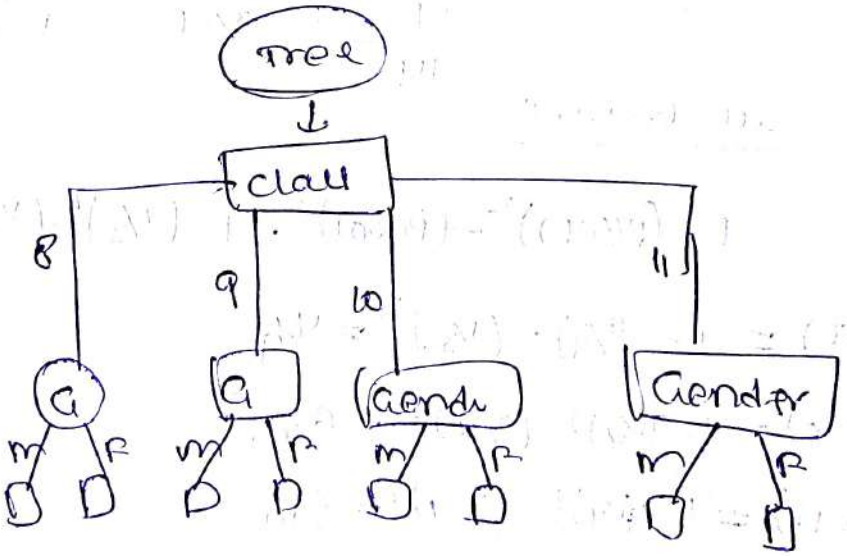
$$G(F) = 1 - (1/2)^2 - (1/2)^2 = 0.5$$

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

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

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.

→ This 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 To.

'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'

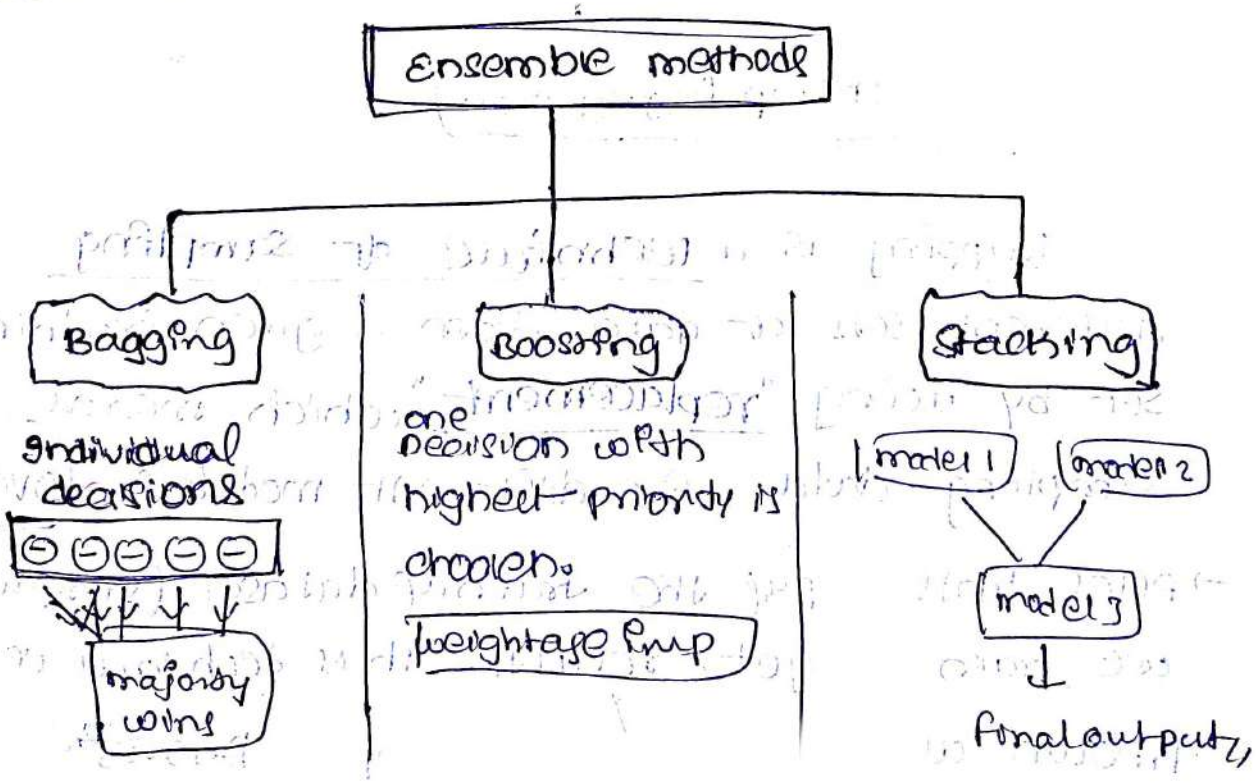
multiple decision makers

(*) 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

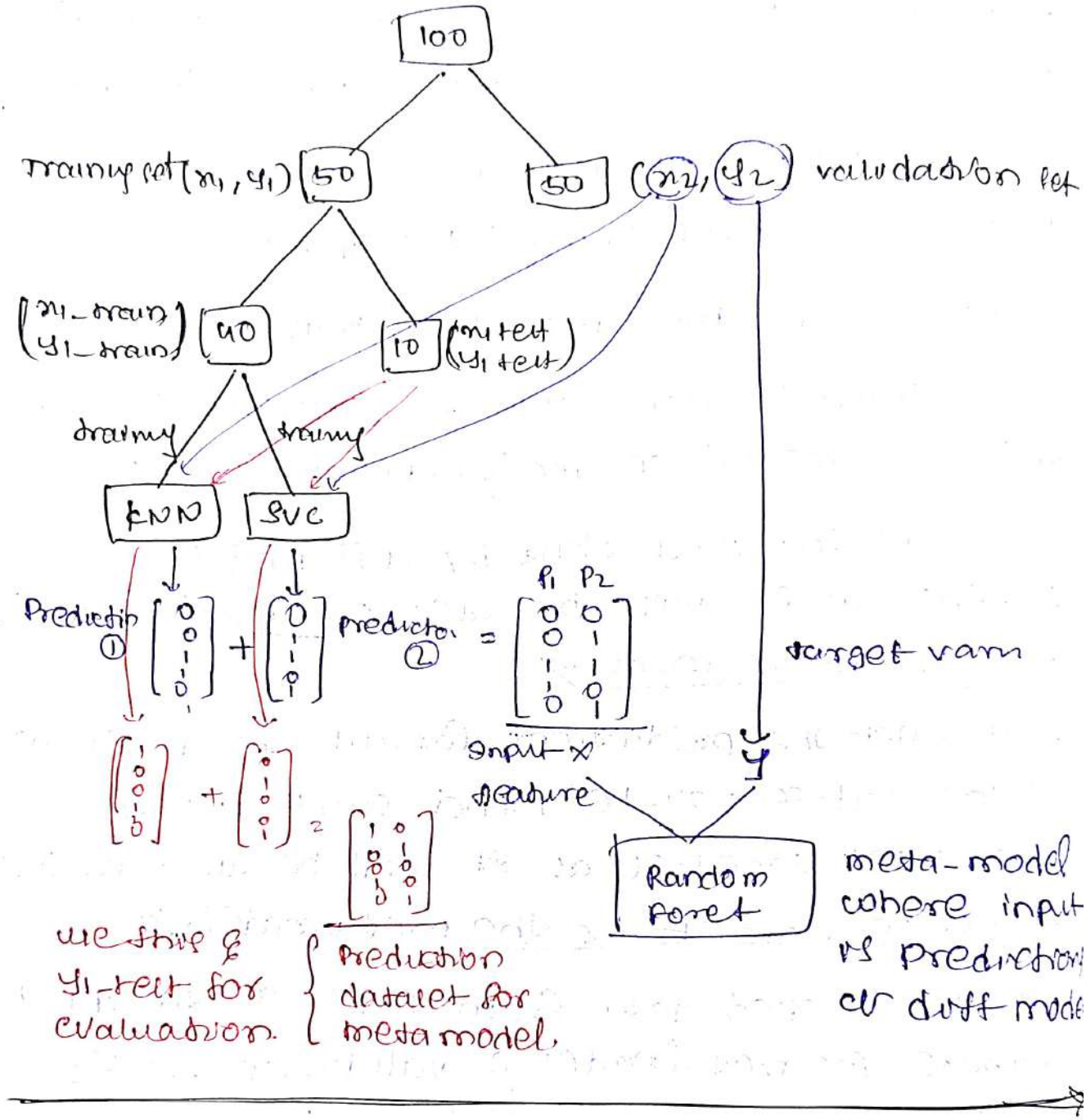
⑦ Prediction dataset for the metamodel.

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.

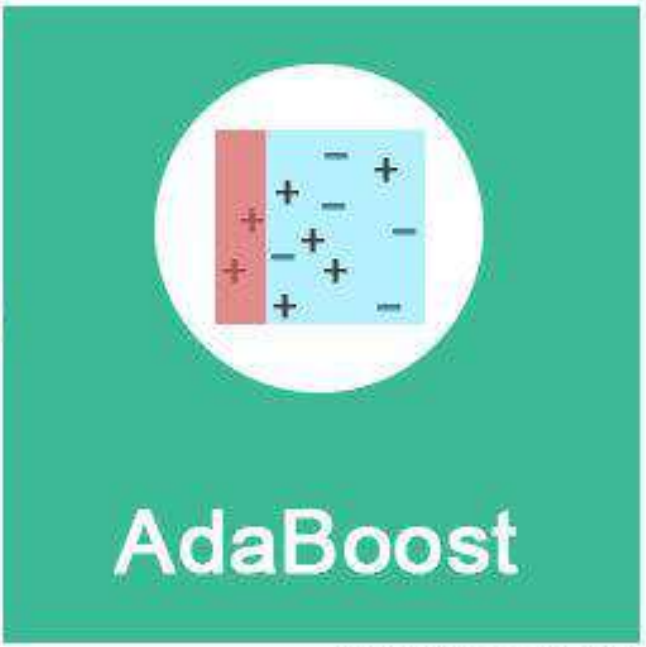
② since base model is a tree, handling of missing values is easy.

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

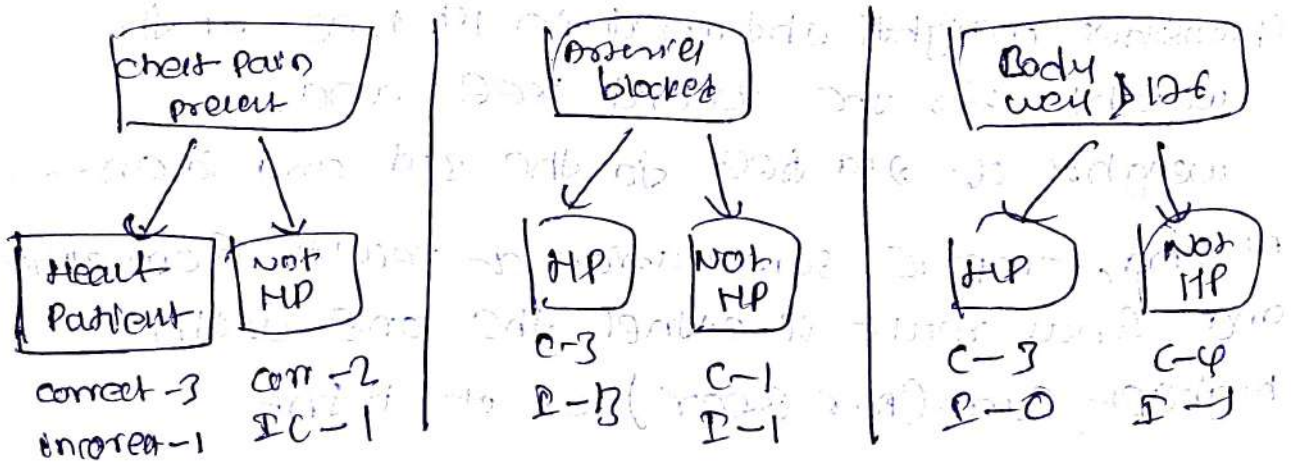
	is chest pain	Arteries blocked	weight of person	is Heart Patient	new weight
0	yes	yes	205	yes	0.05
1	no	yes	180	yes	0.05
2	yes	no	210	yes	0.05
3	yes	yes	167	yes	0.33
4	no	yes	156	no	0.05
5	no	yes	125	no	0.05
6	yes	no	168	no	0.05
7	yes	yes	172	no	0.05

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 weight.

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

then combination 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 total error

② calculate pseudo residuale.

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 : $F_0(x) = 1414$

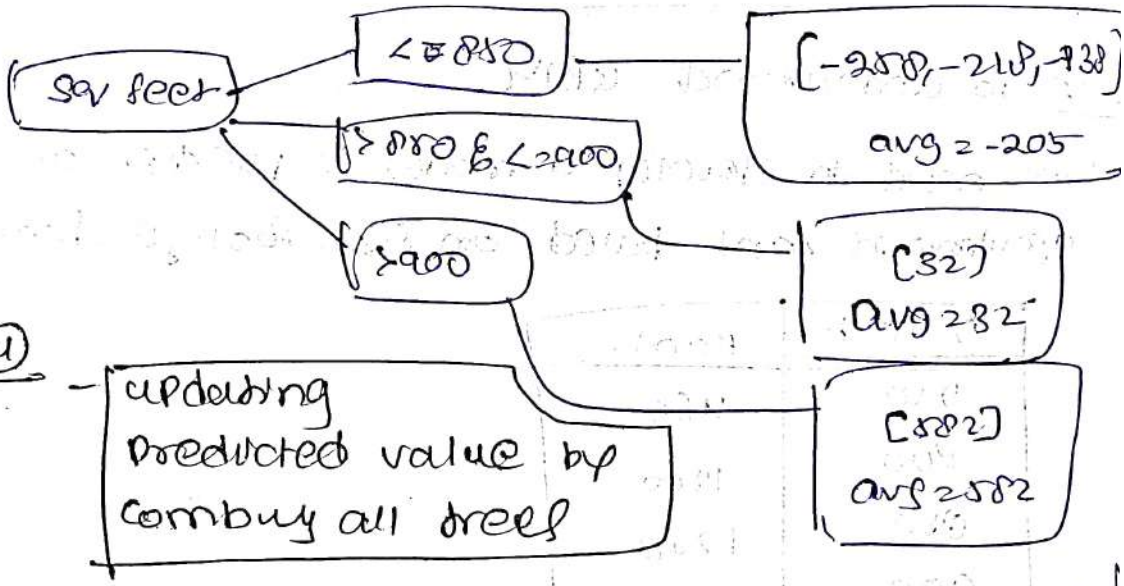
Step ① — calculate the residuals

→ 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.

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

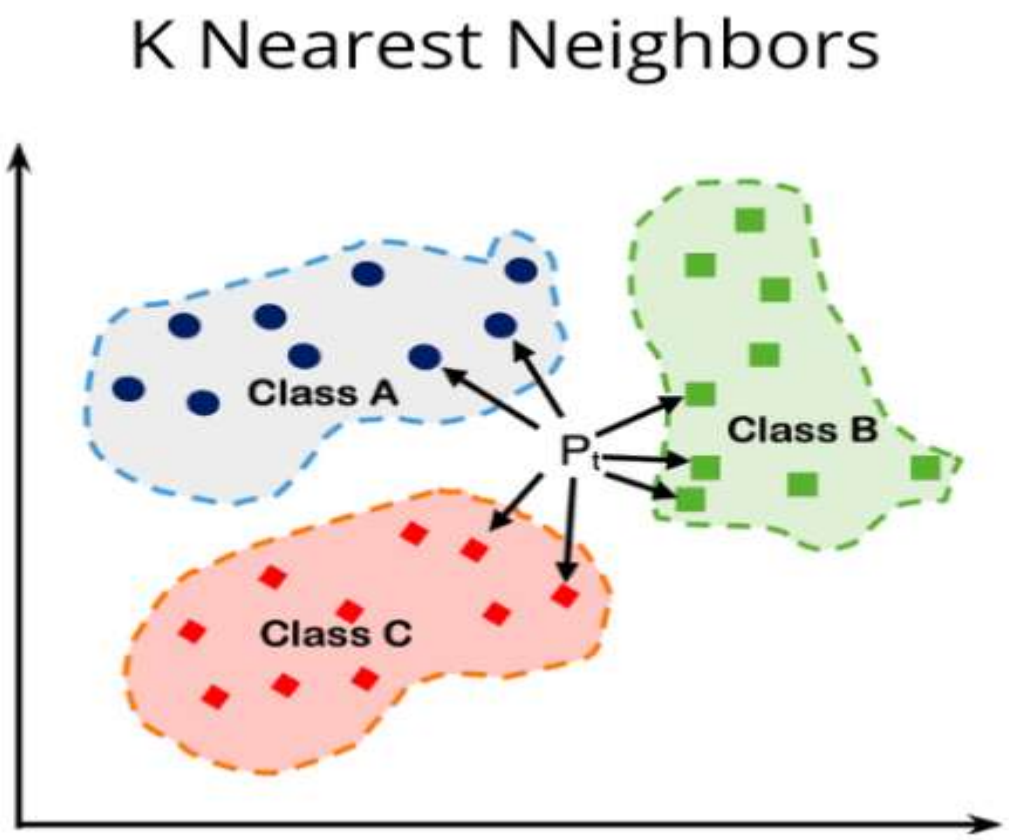


Table Of Contents

(9) K-Nearest neighbour (KNN)

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

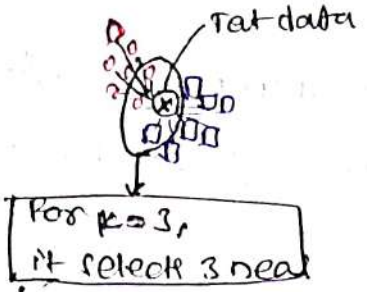
→ Given a dataset with different classes, KNN tries to predict the correct class for test data by calculating the distance between the test data and ALL THE TRAINING POINTS!! (ALL)
Each element in test is measured with all the training points.

→ 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"

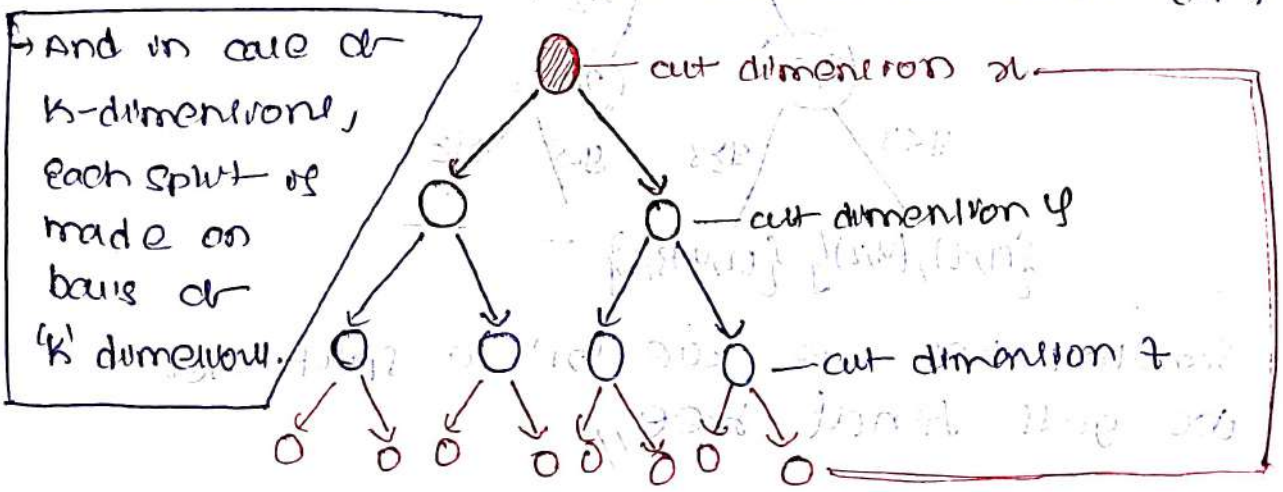
⊛ K-dimensional Tree (Kd Tree)

→ 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.

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

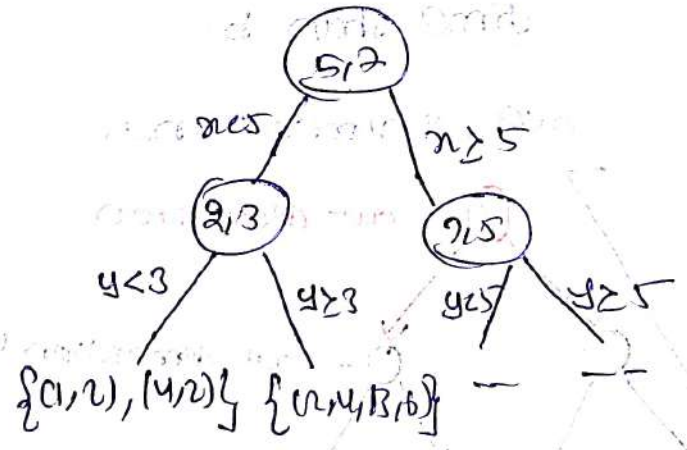
↳ median.

∴ 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.

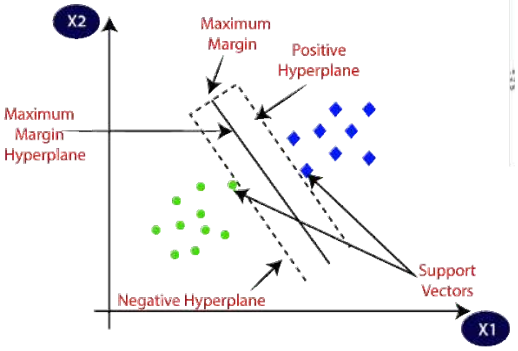
→ the main problem of kd tree is that it gives probable nearest neighbors but can miss out actual nearest neighbors.

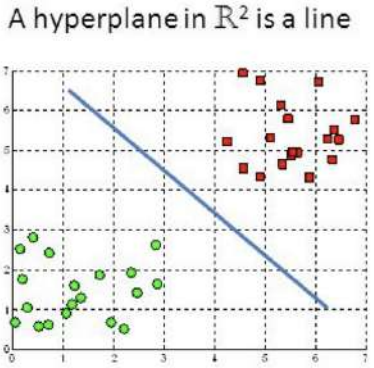
ii) Ball-Tree

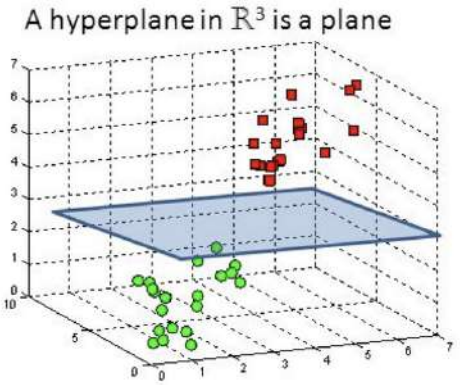
→ similar to kd trees, Ball trees are also hierarchical data structures. these are very efficient specially in case of higher dimensions.

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

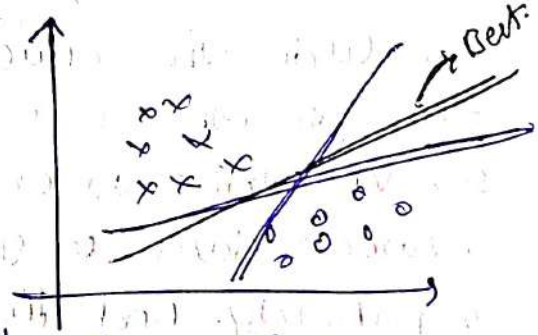
Table Of Contents

(*) 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

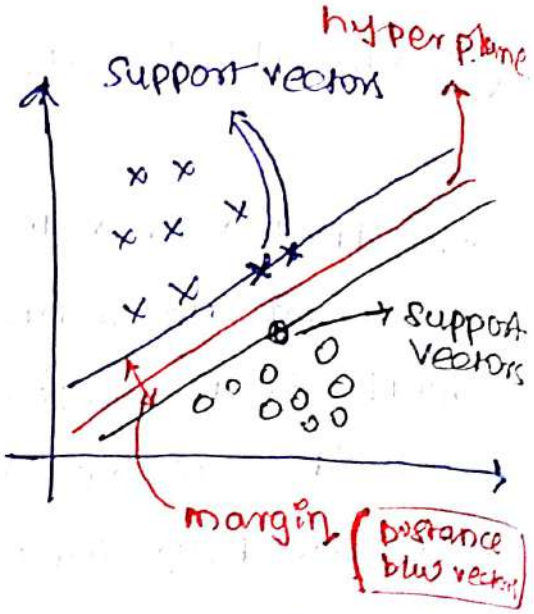
→ similar to logistic Regression, we get a plane / hyper plane in SVM, but how do we decide on the best hyperplane. There can be several planes passing through the datapoints.



→ 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.

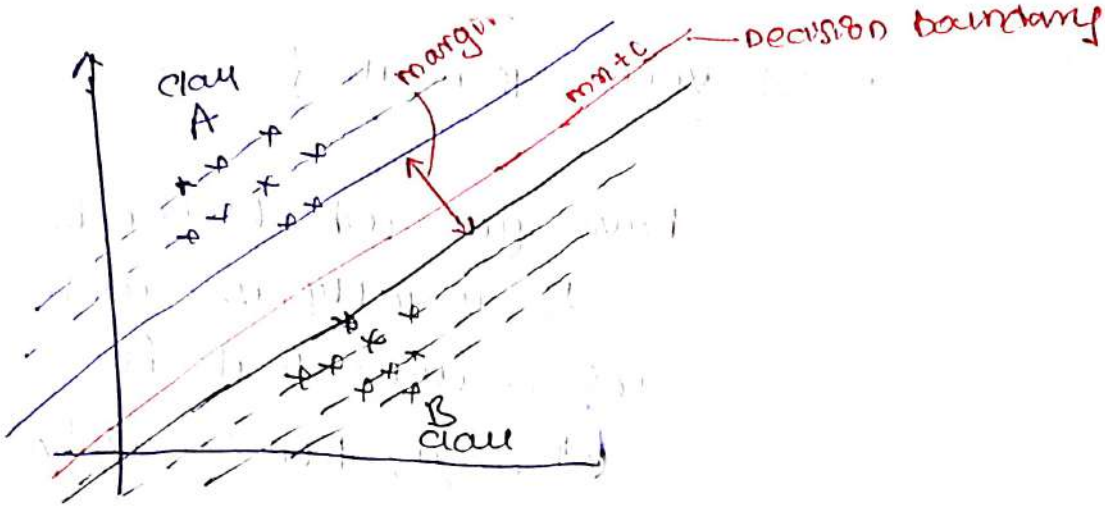


→ the points that are close to decision boundary are called "support vectors". And these support vectors are crucial for defining hyperplane so these support vectors help us find optimal plane

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!

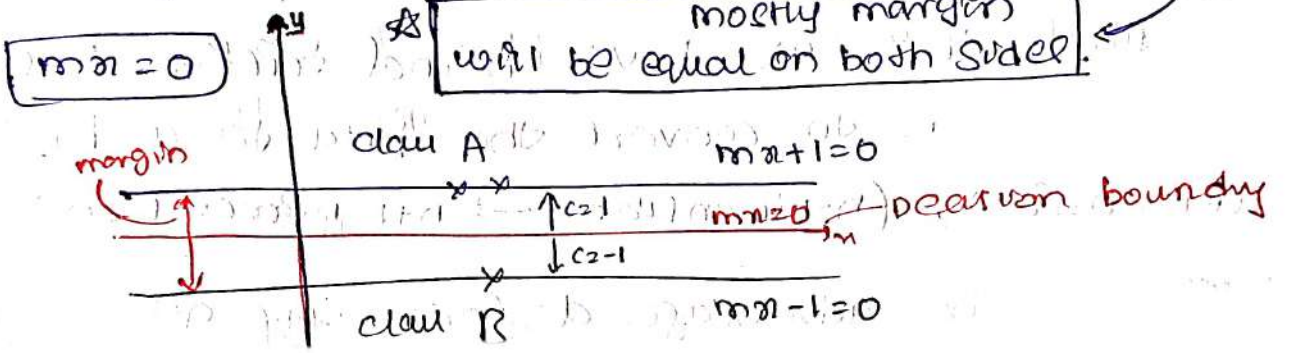
So, SVM tries to create maximum seperation between data to get rid of the confusion of Probability. And the maximum seperation can be achieved through the distance between two closest vector lines from opp. sides.



→ 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

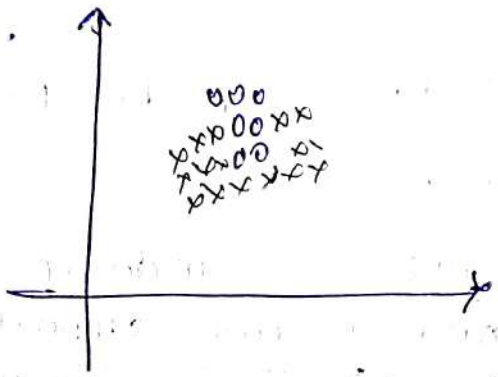
→ SM allows for some margin violations & misclassifications, thus providing flexibility when dealing with noisy / overlapping data. It introduces a slack variable (ξ_i) to handle margin violations, allowing some points to fall within the margin / on wrong side of boundary.

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

↳ where $\xi_i \geq 0$ for all points

⊙ Kernelized support vector machines

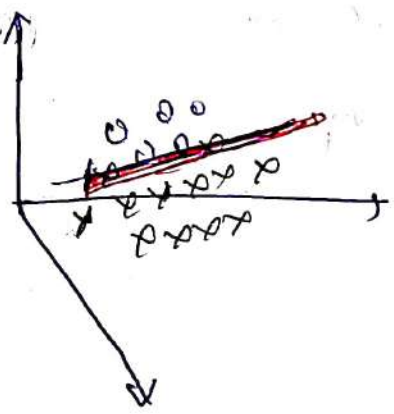
+ It's not always we have get which can be easily seperable by line!! usually, we will be having data in such a way that it cannot be seperable through lines easily! Such as below.



→ 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:

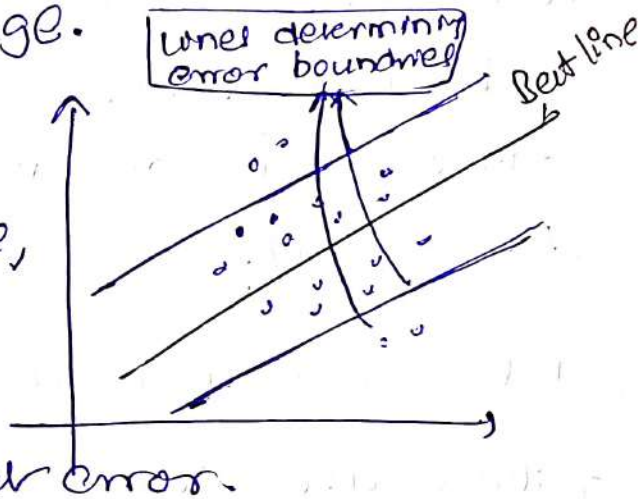
RBF kernel measures the similarity between two samples using a Gaussian radial function.

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

⊙ 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

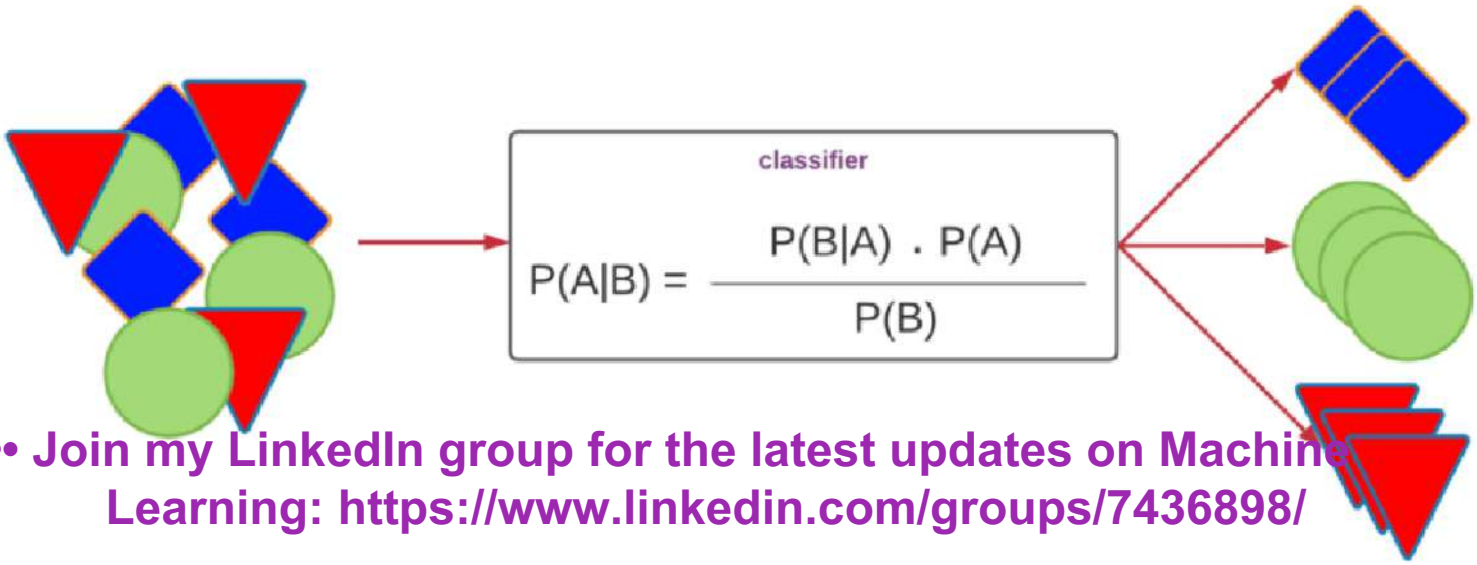


Table Of Contents

⊙ 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

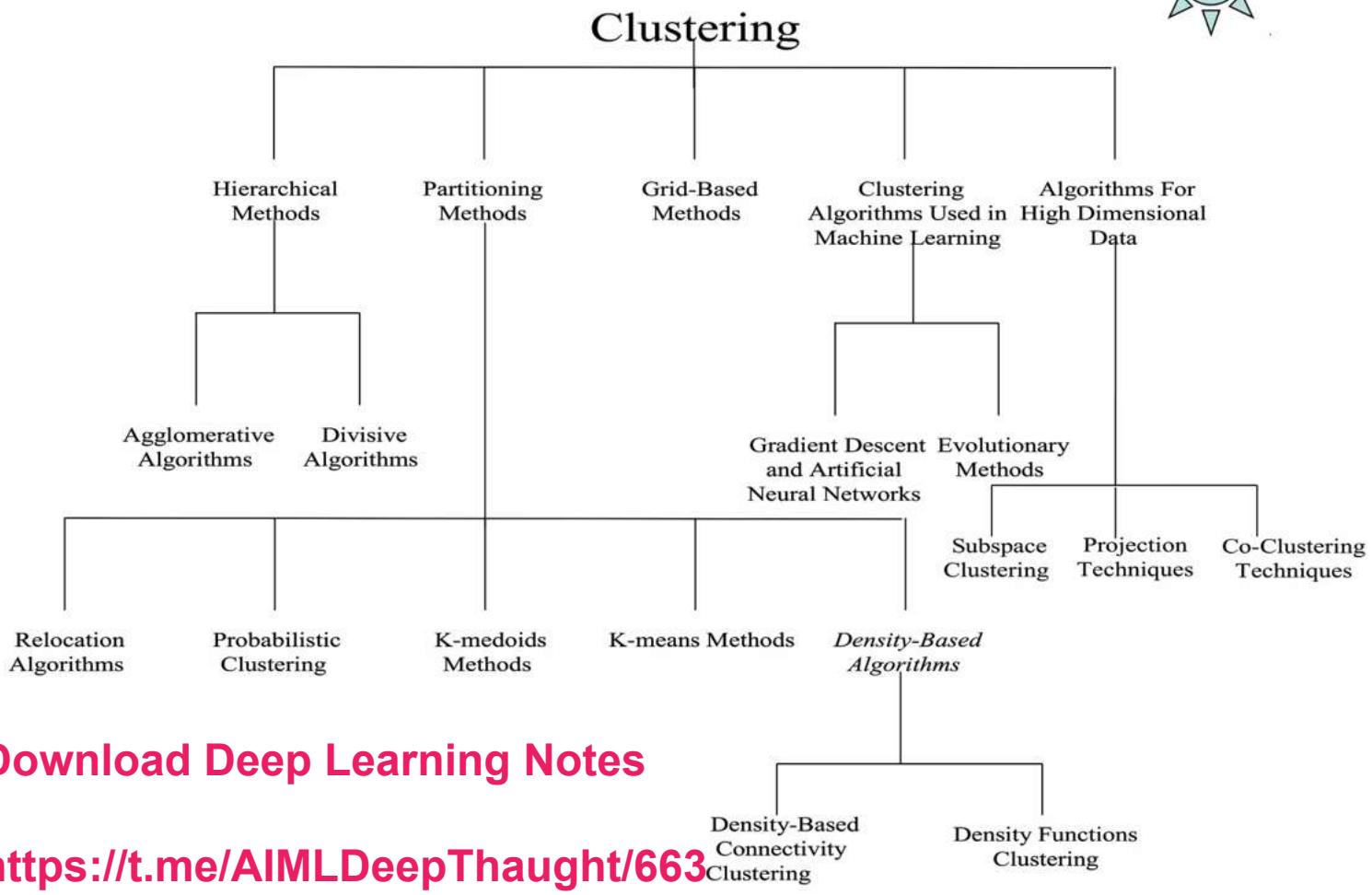


Table of Contents

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.

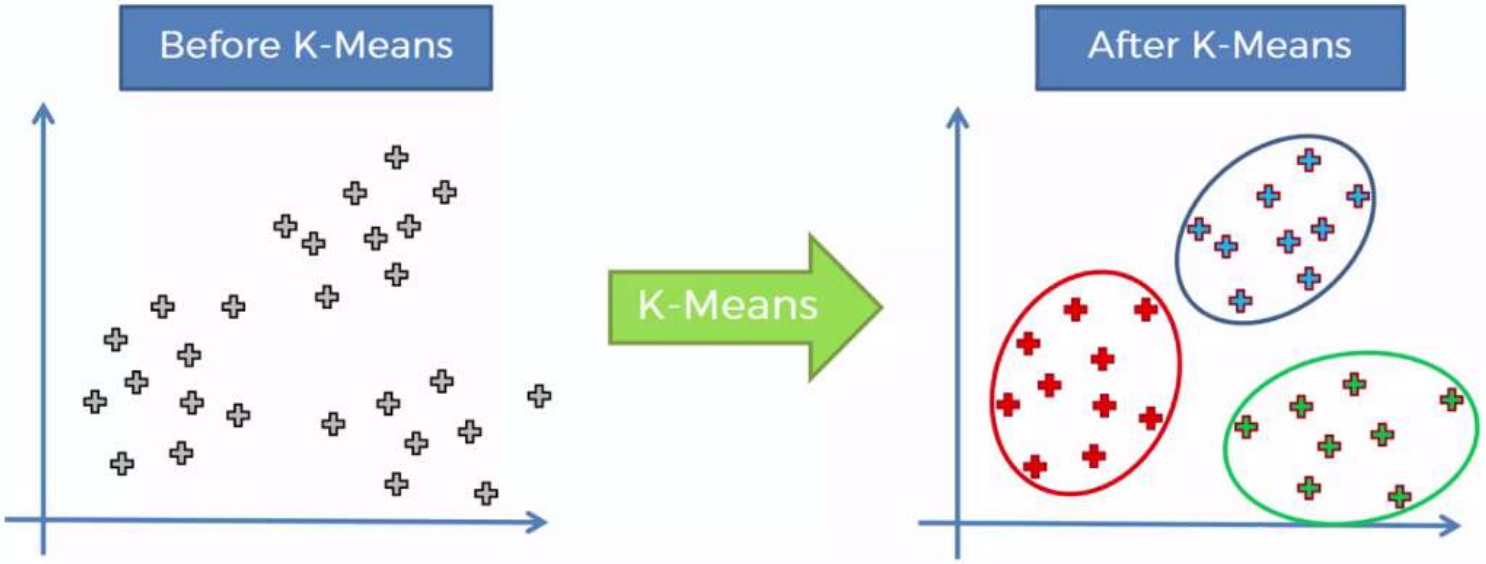


Table Of Contents

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.

10 K-means

Old Algorithm

Proposed in 1957 by Stuart Lloyd, AUO
In 1965 by Edward W. Forgy. so
Sometimes called Lloyd-Forgy Algo.

→ K-means is a clustering approach in which data is grouped into K distinct non overlapping clusters based on their distances from the K centres.

→ The value of 'K' needs to be specified first & then the algorithm allocates the points to exactly one cluster.

Theory of K-means

→ Let $C_1, C_2, \ldots C_k$ be the K-cluster.

→ then we can write :-

$C_1 \cup C_2 \cup C_3 \cup \ldots \cup C_k = \{1, 2, 3, \ldots n\}$ → i.e, Each data point has been assigned to a cluster

And also, $C_k \cap C_{k'} = \phi$ ⟶

this means, cluster are non-overlapping.

→ the idea behind the K-means approach is that within-cluster variation amongst the point should be minimum.

The within-cluster variance is denoted by :- $W(C_k)$

→ Hence, according to statement above, we need to minimize the variance for all cluster mathematically,

$$\underset{C_1, C_2, \ldots C_k}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}$$

WCSS :- within cluster summation of squares

→ the next step is to define the criterion for measuring the within cluster variance. Generally, the criterion is the Euclidian distance between two datapoints.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

→ the above formula says that we are ① calculating the distances b/w all the points in a cluster ② then we are repeating it for all the K-clusters (so two summations)

Eg:-

Let's take 4 points

Case ①:-

4 Points
4 clusters

→ Now, each point acts as a cluster and so it is centroid or itself.

∴ Distance b/w the point & centroid (same point) is zero (0).

→ And it is same for all the 4 clusters on this.

i.e) $w(C_4) = 0+0+0+0 = 0$

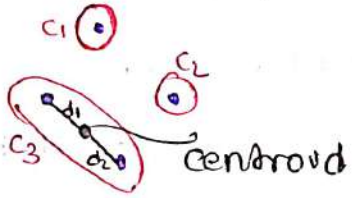
$C_1\ C_2\ C_3\ C_4$

↓

(Summation of distances)

Case ②:-

4 Points
3 clusters



→ Now we need distance b/w points & the centroid of cluster it is present in.

$w(C_3) = C_1 + C_2 + C_3$

$w(C_3) = 0 + 0 + (d_1 + d_2)$

(w) (w) (summation of distances in a cluster)

① Individual sum of distances of a cluster

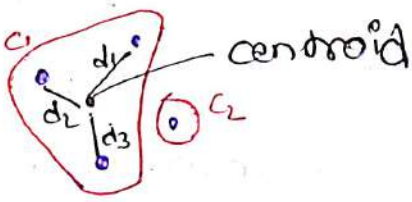
$\sum_{i=1}^{n} (x - c)^2$ ← centroid

② All the clusters sum

$\sum_{k=1}^{k} \sum_{i=1}^{n} (x - c)^2$ //

Case ③:-

4 Points
2 clusters



$w(C_4) = C_1 + C_2$

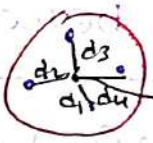
↓

$w(C_4) = (d_1 + d_2 + d_3) + 0$

(distance b/w point & centroid) (summation of distances in a cluster.)

Summation of the individual summations of distances in cluster

Case ④:- 4 Points, 1 cluster



$w(C_1) = C_1$

↓

$w(C_4) = (d_1 + d_2 + d_3 + d_4)$

Goal is to minimize $w(c_1)+w(c_2)+w(c_3)+w(c_4)$ → low variance for all clusters

So, we got ↓

$w(c_4) = 0$

$w(c_3) = d_1 + d_2$

$w(c_2) = d_1' + d_2' + d_3'$

$w(c_1) = d_1'' + d_2'' + d_3'' + d_4''$

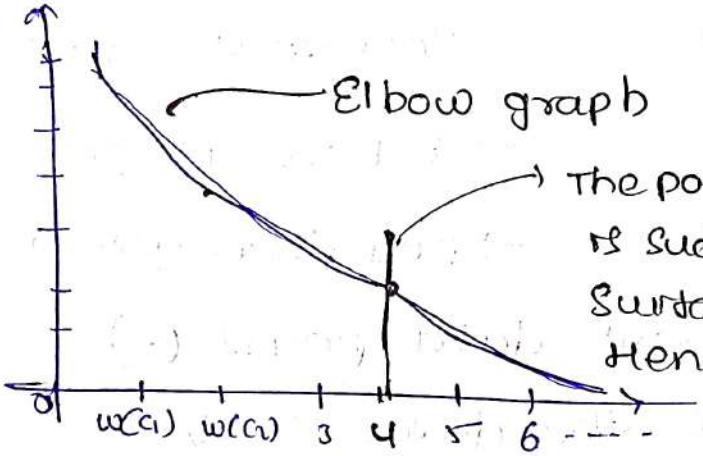
→ By observation, we can say that

As no. of cluster increases, $w(c_k)$ value decreases & tends to '0'

$$\therefore w(c_k) \propto \frac{1}{\text{no. of cluster}(k)}$$

$w(c_1) > w(c_2) > w(c_3) > \cdots > w(c_k)$

and the graph goes as -



within cluster summation of squares (WCSS)

→ this is how we find the value of 'k' in k-means.

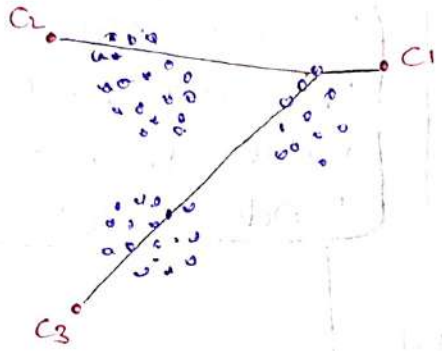
But, here is the problem in k-means,

(*) After selecting k value, it takes k-points as centroids and starts building the cluster normally.

But thing is where do we place those k-points ?

(*) cluster building process

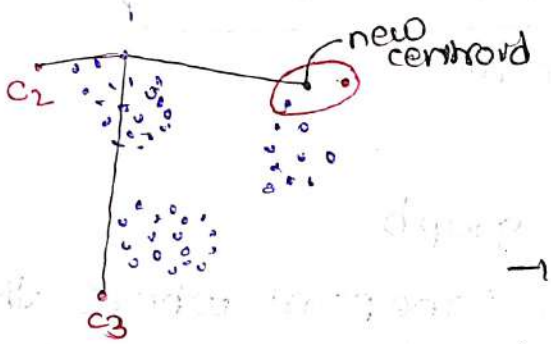
Let's say we got: (K=3)



→ After, randomly placing the K points,

→ It takes a point from our data (·) and measures the distance to all the K-centroid points (·),

And whichever distance is smaller, the point goes that cluster.



→ So, here it goes to C_1

→ And then gets a new centroid.

→ Now for next data point (·) we measure the distance of point to the "new centroids"

→ And in this way, cluster keeps building and centroid gets updated.

→ This process continues until all the data points are finished.

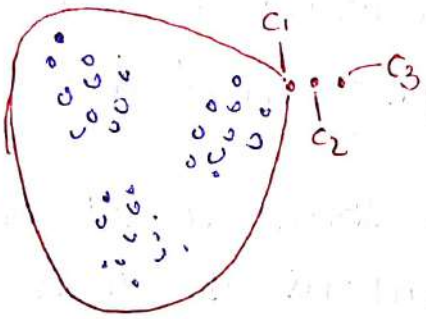
this is how grouping happens.

Edge case scenario

Problem with K-means

→ In the above example, we took K=3 and then it generated 3 random points and build cluster.

→ So, what if all the 3 points are close?



If all are close to only C_1.

And,
$C_2 = 0$
$C_3 = 0$
} elements

In this case, all data points would go to cluster ① (C_1)

which is not what we are expecting!!

Algorithm

① Find WCSS for K-groups possible (max K = n) (n = no. of elements)

② Build Elbow graph, and understand K-value (point where there's distinction)

③ Randomly assign K-centres.

④ calculate distance of all points from all K-centres and allocate the points to cluster based on shortest distance to "cluster centroid".
The model's inertia is the mean squared distance b/w each instance & closest centroid.
Goal is to have model with low inertia.

⑤ Once all points assigned to cluster recompute centroids.

(6) Repeat steps 4 & 5 until the location of the centroids stop changing and the cluster allocation of the points becomes constants.

Note:-

→ K-means is only recommended for spherical data (or) symmetrical data.

→ For, non spherical data there won't be fair chance for k clusters to form clusters, and it will become high biased cluster.

(*) Challanges and Improvements in K-means:-

(1) we need to specify the no. of clusters beforehand.

(2) It is required to run the algorithm multiple times to avoid a suboptimal solution.

(3) K-means does not behave very well when the clusters have varying sizes, different densities or non-spherical shapes.

(4) An improvement to K-means is propoled in 2003 by charles Elkan.

→ It accelerates the algorithm by many unnecessary distance calculations ; which is achieved by exploiting the "TRIANGLE INEQUALITY"

In a triangle with sides a, b & c i.e. $a+b>c$
the straight line is always shortest

⑤ Another important improvement to k-means algorithm called "k-means++", was proposed in 2006 by David Arthur & Sergei.

→ They introduced a smarter initialization step that tends to select centroids that are distant from one another, and this makes the k-means algorithm much less likely to converge to a suboptimal solution.

⑥ Yet another important variant of the k-means algorithm was proposed in a 2010 paper by David Sculley.

→ Instead of using the full dataset at each iteration, the algorithm is capable of using mini-batches, moving the centroids just slightly at each iteration. This speeds up the algorithm typically by a factor of 3 or 4 and makes it possible to cluster huge datasets that do not fit in memory. Scikit-Learn implements this algorithm in the "MiniBatchKMeans" class.

Initially, All points as seperate clusters

(11) Hierarchical Clustering

Bottom-up / Agglomerative approach

→ one main disadvantage of K-means is that it needs us to pre-enter the no. of clusters (k).

→ Hierarchical clustering is an alternative approach which does not need us to give the value of k beforehand and also, it creates a beautiful 'tree-based structure' for visualisation).

→ we start by defining any sort of similarity between the datapoints. Generally, we consider Euclidian distance. the points which are closer to each are more similar than the points which are farther away.

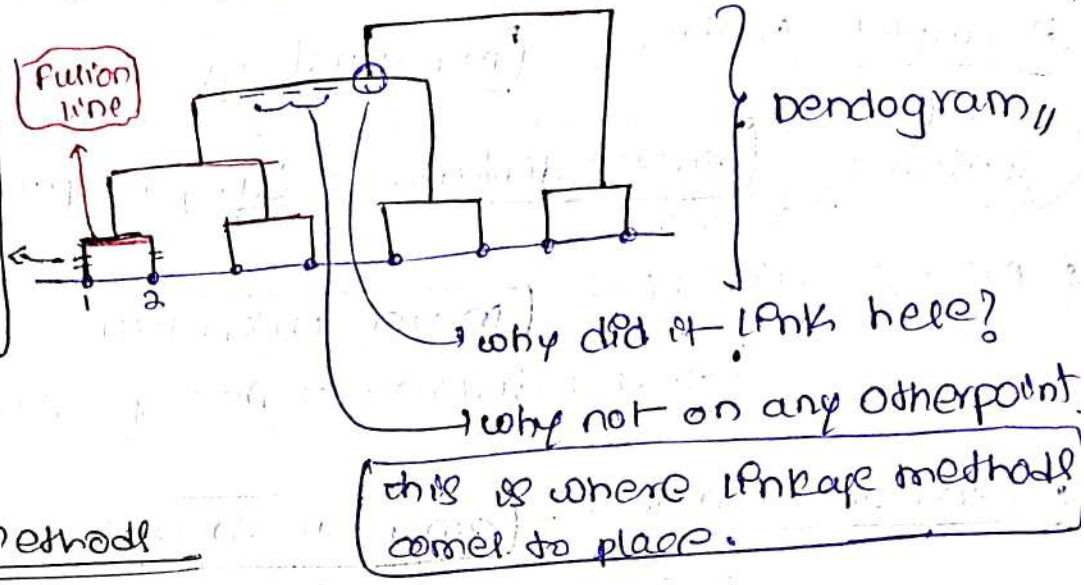
Algorithm

① Begin with n observations and a measure (such as Euclidian distance) of all the $n(n-1)/2$ pairwise dissimilarities (distances). Treat each observation as its own cluster. So, initially we have n clusters.

② compare all the distances and put two closest points/clusters in the same cluster. the dissimilarity between these two clusters indicates the height in the dendogram at which "fusion line" should be placed.

③ compute the new pairwise inter-cluster dissimilarities among the remaining clusters.

④ Repeat steps 2 & 3 till we have only one cluster left.



Linkage methods

→ Based on pairwise distances, we can now compute a linkage matrix.
the linkage matrix is simply a table listing which pairs of points are merged at what step & what distance.

→ we can cut dendogram to form flat clusters.

* we know, initially HC starts with clusters consisting of individual points.

→ later it compares the clusters with each other and merges the two "closest clusters".

→ since clusters are pair of points, there are many different kinds of linkage methods.

Notes

⊙ Single Linkage:-

cluster distance = smallest pairwise distance

⊙ complete Linkage:- maximal intercluster distance

less sensitive to outliers

cluster distance = Largest pairwise distance

⊙ Average Linkage:- mean intercluster dissimilarity

cluster distance = Average pairwise distance

⊙ centroid Linkage:- can result in undesirable inversions

cluster distance = distance between the centroids of clusters

⊙ ward's Linkage:- Before & after merging

cluster distance = minimize the variance in cluster.

Simple Linkage:-

minimal intercluster dissimilarity.

→ Single linkage can result in extended, trailing cluster in which single observations are fused one-at-a-time.

→ cluster distance is the smallest distance b/w any point in cluster ① & any point in cluster ②

→ High sensitive to outliers when forming flat clusters.

→ works well for low-noise data with unusual structures

(12) DBSCAN (Density Based Spatial Clustering of Applications with Noise.)

→ It is an unsupervised machine learning algorithm. This algorithm defines clusters as continuous regions of high density.

Key words :-

⊙ **Epsilon :-** (EPS)

This is the distance fn which we look for the neighbouring points.

⊙ **min-points :-**

The min no. of points specified by the user

⊙ **core point :-**

If the no. of points inside the epsilon radius of a point is greater than or equal the min_points then it is called a **core point**.

⊙ **Border point :-**

If the no. of points inside the epsilon radius of a point is less than the min points and it lies within the eps radius region of a core point, then it is called border point.

⊙ **Noise :-**

A point which is neither core nor a border point is a noise point.

Algorithm steps

① The algorithm starts with a random point in the dataset which has not been visited yet and its neighbouring points are identified based on the eps value.

② If the point contains ≥ points than min points then the cluster formation starts and this point becomes a core point;
else it's considered as Noise.

→ The point to note here is that a point initially classified as noise can later become a border point if it's in the eps radius of a core point.

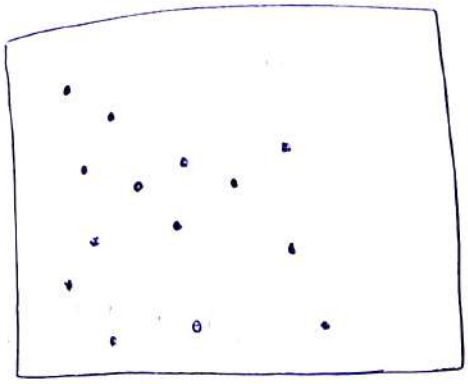
③ If the point is a core point, then all its neighbours become a part of cluster. If the points in the neighbourhood turn out to be core points then their neighbours are also part of cluster.

④ Repeat the steps above until all points are classified into different clusters or noises.

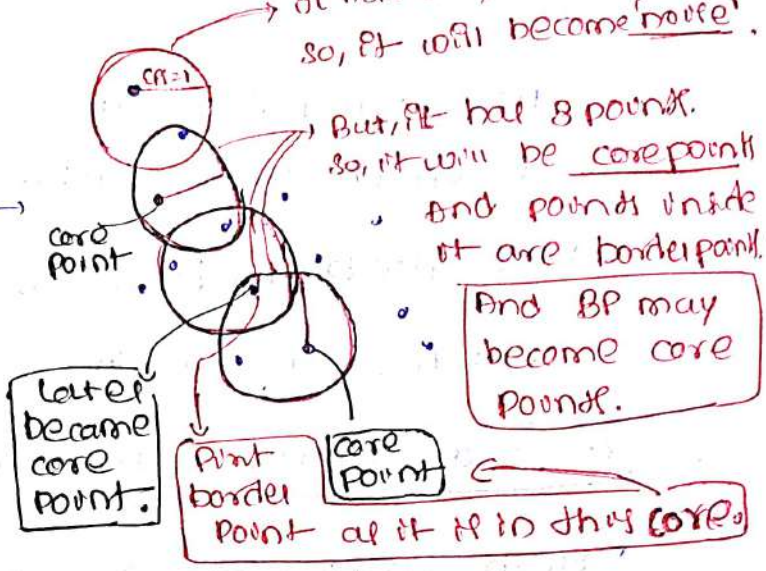
This algo works well if all clusters are dense enough, and they are well separated by low dense regions

Eg:-

Let's say eps = 1, min point = 3.



data.



→ In short, DBSCAN is a very simple yet powerful algorithm, capable of identifying any no. of clusters of any shape, it is robust to outliers, and it has only two hyperparameters.

(eps & min samples)

→ However, if the density varies significantly across the clusters, it can be impossible for it to capture all the clusters properly. Moreover its computational complexity is roughly $O(m \log m)$, making it pretty close to linear with regards to the no. of instances.

//

⊛ Evaluating of clustering:

Cluster validity:

The validation of cluster created is a troublesome task.

→ The problem here is "cluster are in the eyes of the beholder".

A good cluster will have:

- High inter class similarity
- Low inter class similarity.

Aspects of cluster validation:

- External: compare your cluster to ground truth.
- Internal: Evaluating the cluster without reference to the external data.
- Reliability: the clusters are not formed by chance (randomly) — Some statistical framework can be used

⊙External measures

→ No. of objects in the data P :- $P_1, P_2, \ldots P_m$

the set of ground truth clusters C :- $C_1, C_2 \ldots C_n$

the set of clusters formed by the algorithm.

the Incidence matrix N * N matrix.

→ $P_{ij} = 1$, if the two points O_i & O_j belong to the same cluster in the ground truth

else, $P_{ij} = 0$.

→ $C_{ij} = 1$, if the two points O_i & O_j belong to the same cluster in the ground truth

else, $C_{ij} = 0$

Now there can be following scenario

1) $C_{ij} = P_{ij} = 1$ ⟶ Both the points belong to the same cluster for both our algorithm & ground truth (Agree) — SD

2) $C_{ij} = P_{ij} = 0$ ⟶ Both the points don't belong to same cluster for both our algorithm & ground truth (Agree) — DD

3) $C_{ij} = 1$ but $P_{ij} = 0$ → the points belong in the same cluster for our algorithm but different cluster in ground truth (Disagree) — SD

4) $C_{ij} = 0$ but $P_{ij} = 1$ → the points don't belong in same cluster for our Algo but same cluster in ground truth (disagree) — DS

↑ Just like accuracy score

$$\text{Rand Index} = \frac{\text{total Agree}}{\text{total Disagree}} = \frac{(SS+DD)}{(SS+DD+DS+SD)}$$

→ the disadvantage of this is it can be dominated by DD.

$$\text{Jaccard coefficient} = \frac{SS}{(SS+DS+SD)}$$

∴ A higher value of Rand Index & Jaccard coefficient mean that the clusters generated by our algorithm mostly agree to the ground truth.

confusion matrix

n = no. of points
m_i = Points in cluster i
C_j = Points in class j
n_{ij} = Points in cluster i coming from class j.
$P_{ij} = \frac{n_{ij}}{m_i}$

(Probability of element from cluster i to be assigned to class j

	class 1	class 2	class 3	
cluster 1	n_{11}/P_{11}	n_{12}/P_{12}	n_{13}/P_{13}	m_1
cluster 2	n_{21}/P_{21}	n_{22}/P_{22}	n_{23}/P_{23}	m_2
cluster 3	n_{31}/P_{31}	n_{32}/P_{32}	n_{33}/P_{33}	m_3
	C_1	C_2	C_3	n

→ Entropy of cluster P_i

$$e_i = -\sum P_{ij} (\log P_{ij})$$

→ For entering clustering algorithm, entropy can be given as

$$e = \sum \frac{m_i}{n} e_i$$

4

→ Purity of cluster i, $p_i = \max(p_{ij})$

And for entire cluster it is $P(C) = \sum \frac{m_i}{n} \cdot p_i$

→ Purity, is the total percentage of data points "clustered correctly."

→ A high value of purity score means that our clustering algorithm performs well against the ground truth.

Note:-

→ In calculating External measures, most of time we don't have ground truth,

---

⊙ External measures:-

These are the methods we do measure the quality of cluster without external reference. there are two aspects of it.

Cohesion:-

How closely the objects in the same cluster are related to each other. It is the within-cluster sum of squared distances. It is the same metric that we used to calculate for K-means Algorithm.

$$WCSS = \sum \sum (x - m_i)^2$$

Separation:-

How different objects in different clusters are and how distinct a well separated cluster is from other clusters.

It is the between cluster sum of squared distance

$$BSS = \sum c_i (m - m_i)^2$$

where, c is the size of individual cluster & m is centroid of all data points

Note:-

✓ BSS + WSS is always a constant

→ the silhouette can be calculated as:-

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}$$

where,

a(x) is avg. distance of x from all other point in same cluster.

b(x) is avg. distance of x from all other points in other clusters.

And silhouette coefficient is,

$$SC = \frac{1}{N} \sum s(x)$$

Higher s(x) means that the ____ inter cluster similarity is less and the intra cluster dissimilarity is more (Good clustering)

(*) The curse of Dimensionality.

→ In the real world datasets, we often encounter data with hundreds or even thousands of features. Problems with more features/dimensions.

① As the majority of the machine learning algorithms rely on the calculation of distance for model building, and as the number of dimensions increases, it becomes more & more computationally intensive to create a model out of it.

Eg:

→ To calculate the distance b/w two points in just one dimension, we can just subtract the co-ordinates of one point from another.

→ For 2D, $= \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

→ And for ND, it becomes: $\sqrt{(a_1-a_2)^2+(b_1-b_2)^2+\cdots+(n_1-n_2)^2}$

→ This is the effort for just two points, just imagine the no. of calculations involved for all datapoints.

② Hard to visualize the relationship b/w features: If we have an n-Dimensional dataset, the only solution left is to create either a 2D/3D graph out of it. Suppose we have 1000 features in the dataset, if we plan to create 2D graphs, we could need $(1000 \times 999)/2 = 499500$ combinations!! And we know it is not possible to spend time to analyze that many graphs to understand the relationship b/w the variables.

→ And like this, when we have huge no. of features, we need to ask the questions like -

① Are all the features really contributing to decision making.

② Is there a way to come to the same conclusion using lesser no. of features.

③ Is there a way to combine features to create a new feature and drop the old ones.

④ Is there a way to remodel features in a way to make them visually comprehensible.

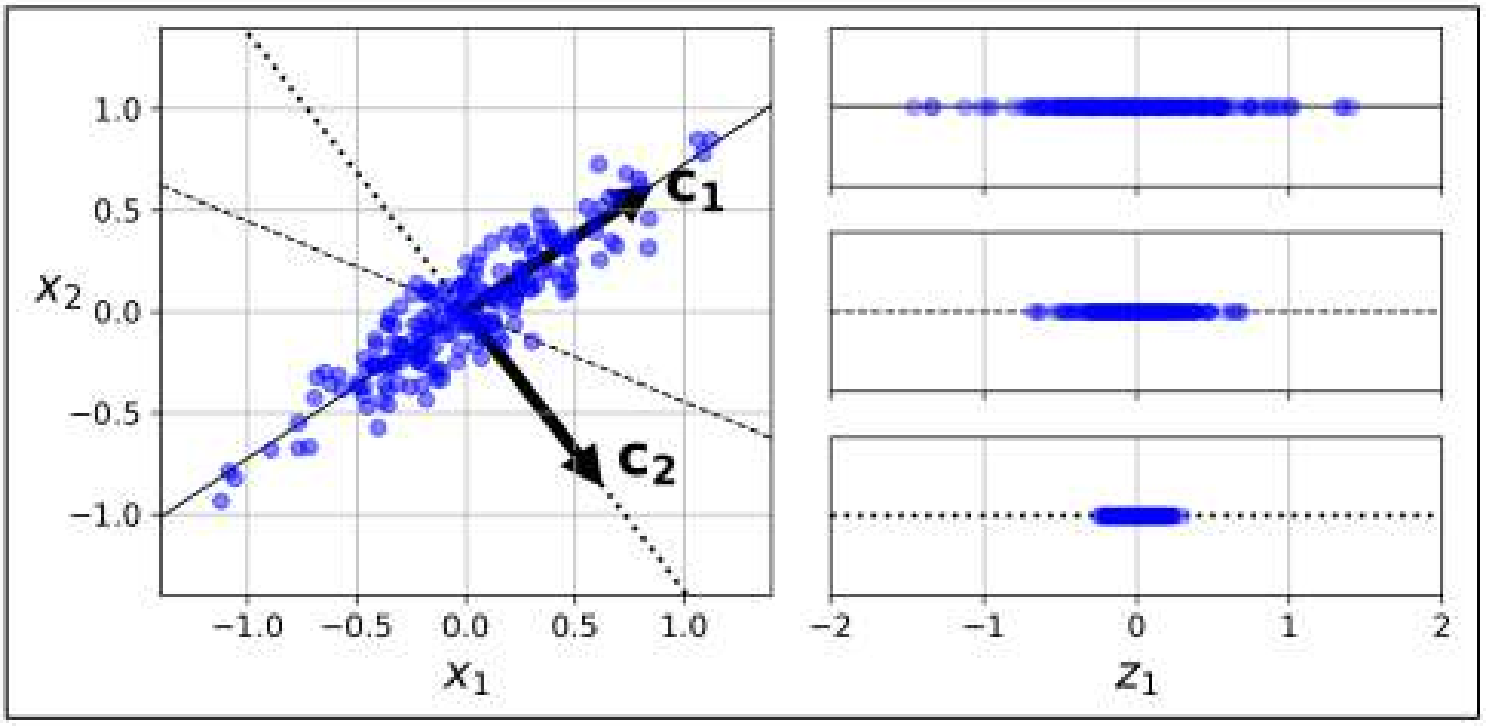
The answer to all the above questions is :-

⊙ Dimensionality Reduction Technique

→ Dimensionality reduction is a feature selection technique using which we reduce the no. of features to be used for making a model without losing significant amount of information compared to original dataset.

→ In other words, a dimensionality reduction technique projects a data of higher dimension to a lower dimension subspace.

→ DR technique should be used before feeding the data to a machine learning algorithm, it reduces the space in which the distances are calculated, thereby improving ML algorithm performance



Understanding Principal Component Analysis

Table Of Contents

(*) Principal component Analysis

→ The most popular dimensionality reduction technique is the principal component analysis, it is an unsupervised algorithm used for feature selection.

Idea behind PCA

→ PCA transforms and fits the data from a higher dimensional space to a lower dimensional subspace, this results into an entirely new co-ordinates system of the points where the first axis explains the most variance in the data (first principle component), this is the idea behind PCA.

what are principal components

→ Principal components are the axis, that explain the maximum variance in the data, the first principal component explains the most variance, the 2nd a bit less and so on..

→ Each of new axis / principal component found using PCA is a linear combination of the original features

→ All the principal component are uncorrelated and orthogonal to each other.

(*) Math Behind PCA

→ Principal component analysis can be implemented in two approaches, each with its unique focus and method. They are as follows:

① Eigen decomposition Method

② singular value decomposition Method,

Lets delve into them

① Eigen Decomposition Method

→ Eigen decomposition is a process that decomposes a square matrix A into a set of eigenvectors and eigenvalues.

→ For a given square matrix A, the eigenvectors are the non-zero vectors that satisfy the equation.

$$\boxed{A = \lambda V}$$ where,

V is the eigen vector and

λ is the corresponding eigenvalue.

Steps of Eigen-decomposition approach

① Standardize columns of A, so that each feature has a mean of zero, this ensures that each feature contributes equally to the analysis.

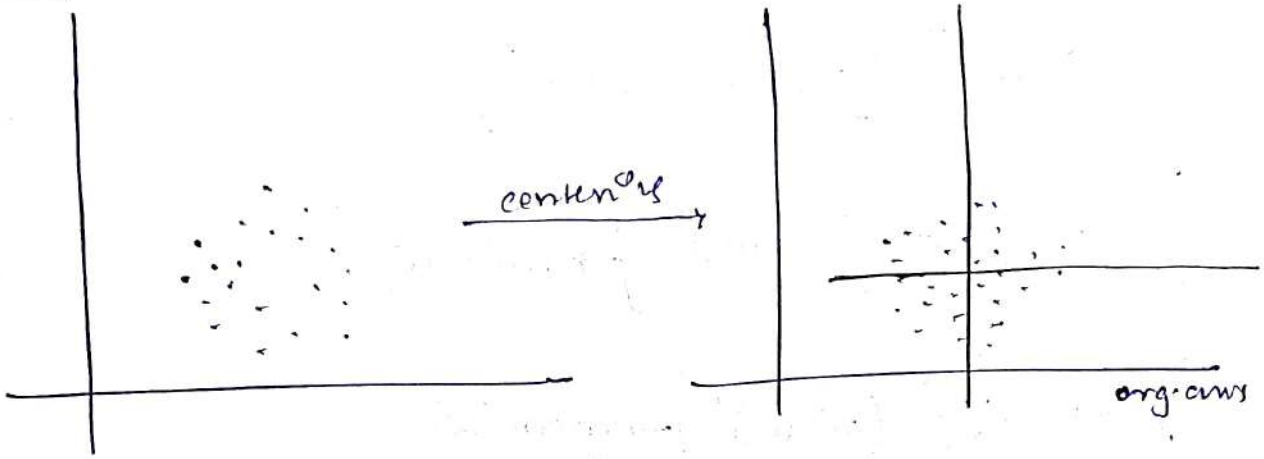
(2) compute co-variance matrix from the normalized data, $\Sigma = A^T A / (m-1)$

(3) perform eigen decomposition of Σ [covariance matrix], to obtain eigen values & eigenvectors.

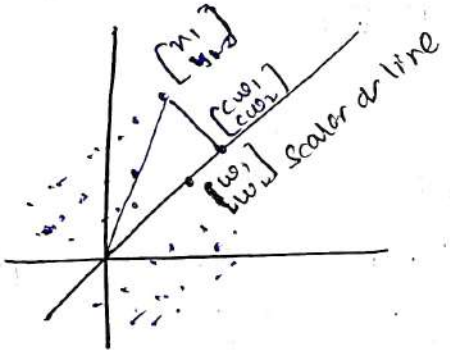
(4) choose the top K eigen vectors corresponding to the largest eigen values to construct the principal components as AX_K → Kth eigen vector.

Proof

Step 1



we have, $D = \{x_1, x_2, \ldots x_n\}$ $x_i \in \mathbb{R}^d$



the projection point will be, $(x^T w) \cdot w$, considering w is a scalar or the best fit line/ the line that has least reconstruction error.

$$\text{Error}(\text{line}, \text{dataset}) = \sum_{i=1}^{m} \text{error}(\text{line}, x_i)$$

$$= \sum_{i=1}^{m} \| x_i - (x^T w) w \|^2$$

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} \| x - (x^T w)\cdot w \|^2$$

This is the mean of the residuals, we can think of this as the avg. error.

$$= \frac{1}{n}\sum_{i=1}^{n} \left[(x - (x^T w)\cdot w)^T \cdot (x - (x^T w)\cdot w) \right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} \left[x_i^T x_i - (x_i^T w)^T - (x_i^T w)^2 + [x_i^T w^2] \right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} \left[x_i^T x_i - (x_i^T w)^2 \right]$$

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} -(x_i^T w)^2 \rightarrow \min w$$

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} (x_i^T w)^2 \rightarrow \max w$$

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} (x_i^T w)(x_i^T w) = \frac{1}{n}\sum_{i=1}^{n} w^T (x_i \cdot x_i^T) w$$

$$J(w) = w^T \underbrace{\left(\frac{1}{n}\sum_{i=1}^{n} x_i \cdot x_i^T \right)}_{\text{we know this is covariance matrix}} \cdot w$$

Each column corresponds to one data point.

$$\therefore \max(w) = w^T C \cdot w$$

where, C – covariance matrix
w – Eigen vector corresponding to max eigen value of C_{11}

→ so, Basically we can say that, the line which represents the data forms or minimum reconstruction error is same as the line which maximizes the $w^T C w$, as C - covariance matrix,
and the line is given by eigen vector corresponding to max. eigen value of C, and this line is the principal component!!

★ First principal component $(w_1) = \arg\max_w (w^T C \cdot w)$

→ Now, any line which minimizes the sum of errors / residuals, must also be orthogonal to w_1.
Hence, second principal component will be orthogonal to the first $w_2^T w_1 = 0$ ★

→ By, continuing this procedure, we get $\{w_1, w_2 \dots w_d\}$
such that,

$$\|w_k^2\| = 1 \quad \forall k \text{ and}$$
$$w_i^T w_j = 0 \quad \forall i \neq j$$

Residuals after d rounds

$$\forall i \rightarrow n_i - \left[(x_i^T w_1) w_1 + (x_i^T w_2) w_2 + \dots + (x_i^T w_k) w_d\right] = \vec{0}$$

$$\forall i \rightarrow x_i = (x_i^T w_1) w_1 + (x_i^T w_2) w_2 + \dots + (x_i^T w_d) \cdot w_d$$

→ If original dataset is of $(d \times n)$, then based on above eqn, after k rounds of PCA, we get $(d \times k) + (k \times n)$

Eg

→ Suppose we have an initial dataset with 50 features and 100 samples. $d=50$, $n=100$

Data dimensions $50 \times 100 = 1500$ data points to represent entire dataset

After 5 PCA rounds, $K=5$

data dimension $= (50 \times 5) + (5 \times 100) = 750$

So, now we only need half of datapoints to represent entire dataset

→ $\{w_1\ w_2\ w_3\ w_4\ w_5\}$ → Representation, common for dataset

→ $x_i \rightarrow [x_i^T w_1 \quad x_i^T w_2 \quad x_i^T w_3 \quad x_i^T w_4 \quad x_i^T w_5]$ → Data Point Specific

→ Hence, to reconstruct x_i we don't need 1500 numbers like naive way features

we only need 5 principal components

→ And through Eigen decomposition approach, we calculate the covariance matrix, and find its eigenvalues & eigenvectors. And define the principal components from eigen vectors. we can reconstruct by

$A \times k \times k^T$

② Singular value decomposition Method

→ SVD method factorises the dataset in such a way that it becomes a product of the multiplication of 3 individual matrices.

$$X(\text{original data}) = U * \Sigma * V^T$$

where, U is the matrix that contains the principal components.

Steps in SVD approach

① Standardize columns of A, to ensure each feature contributes equally to analysis.

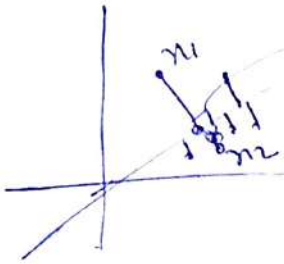
② compute SVD of the centered data matrix to obtain matrices U, Σ, and V^T, such that $X = U \Sigma V^T$ where U contains left singular vectors, Σ holds the singular values & V^T includes the right singular vectors.

③ select principal components: The principal components are obtained by selecting the columns of U corresponding to the largest singular values in Σ, these principal components represent the directions of maximum variance in the dataset.

Note:-

→ what does high variance mean?

when making a proxy,
what if two points get same proxy.



→ Now, How do we distinguish x_1 & x_2.

→ And How do we reconstruct x_1 & x_2.

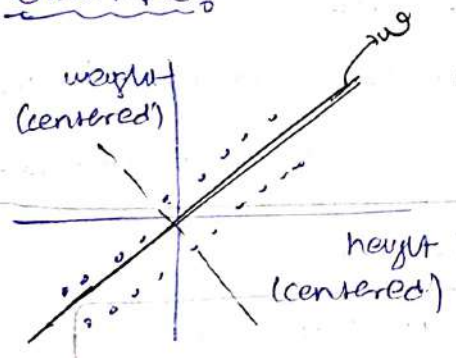
→ So, if all the datapoints are crowded, then we can't reconstruct and the information is lost.

This happens due to less variance / small variance.

Hence, we want directions where projections not crowded.

variance has to be high,
so the datapoints will be spread out
and reconstruction of points can be done properly.

example:



- height gives some information
- weight

→ we have two directions now w_1, w_2..

→ And these two directions are decorrelated, every direction is giving a new information from the other.

→ Hence, we can say these directions are de-correlated.

this's what the algorithm is essentially doing & it's called Principal Component Analysis.

* once we get principal components, Project the original data onto the subspace spanned by the selected principal components, effectively reducing the dimensionality of the dataset.

(*) what is the optimal no. of PC's needed?

* Before we ask this qn, we need to ask how much of the information in a given dataset is lost by projecting the observations onto first few principal components, ie,

> the proportion of variance explained by each principal component.

Explained variance ratio = $\frac{\text{variance by } m^{th} \text{ PC}}{\text{total variance}}$

EVR = $\frac{\sum_{i=1}^{n} \left(\sum_{j=1}^{p} w_{jm} x_{ij} \right)^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2}$

→ EVR of PC1 = $\frac{\text{Distance of PC1 Points}}{\text{Distance of PC1 Points} + PC2 + \cdots PCn}$

If EVR of PC1 = 0.91, then that means PC1 explains 91% of the variance of data.

→ Now that we have a metrics to get the variance explained, we can choose the no. of dimensions that add up to a sufficiently large proportion of the variance (eg- as %).

Eg:

EVR of PC1 = 0.84

EVR of PC2 = 0.10

EVR of PC3 = 0.05

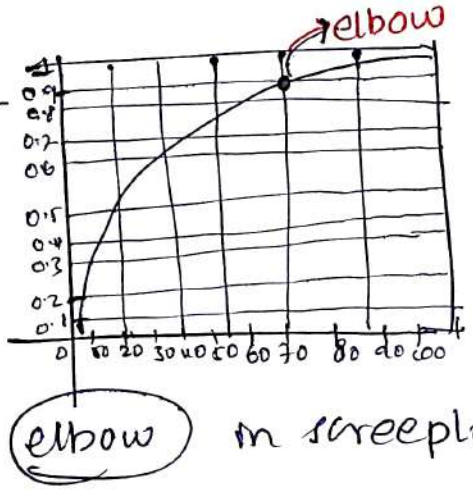
→ Now, if we want to preserve (>90%) variance, then we can just take PC1 & PC2, as they sum up to 0.94, 94%, >90%.

→ Instead if we want to preserve (>95%) of variance, then we should take PC1, PC2 & PC3. as the sum up to 0.99, 99%. >95%.

① Scree plots

→ Scree plots are the graphs that convey how much variance is explained by corresponding principal components.

→ So, by eyeballing the screeplot and looking for a point at which the proportion of variance explained by each subsequent PC drops off. This is often referred as elbow in screeplots.

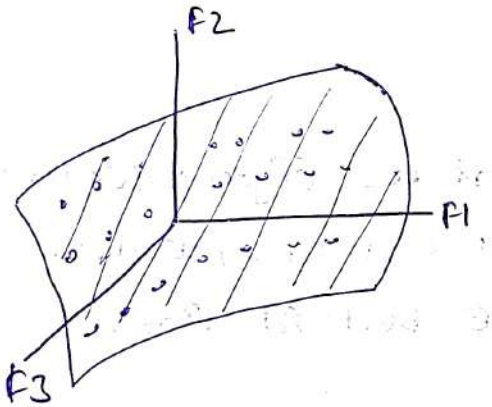


Issues / concerns with PCA

1. Time complexity
   - In particularly, covariance matrix.
   - Usually, it takes $O(d^3)$
   - If no. of features were large, (like 32000 for image) then d^3 is going to be very very large.

   Finding the eigen vectors and eigen values of covariance matrix is the time consuming step.

   Eg: Face recognition (Eigen faces)

2. 

   Data may not necessarily live in a low-dimensional Linear subspa

   → features will not be linearly related.

Notes

- we will have same solution for both the above issues, solving one will solve the other.

Kernel PCA

- Input — $\{x_1, \dots x_n\}$ $x_i \in \mathbb{R}^d$,

Kernel $K: \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$

Step ① compute $K \in \mathbb{R}^{n \times n}$,
where $K_{ij} = K(x_i, x_j)$ $\forall i, j$.

And then we need to center the kernel

Step ② compute $\beta_1, \dots \beta_l$ eigen vectors & $n\lambda_1 \geq \dots n\lambda_l$ eigen values.

~~step~~ and normalize to get, $\alpha_k = \frac{\beta_k}{\sqrt{n\lambda_k}}$ $\forall k$.

Step ③ $w_k = \phi(x) \cdot \alpha_k$

Eigen computation of kernel matrix. Hence, thus defeats the purpose, as it needs $\phi(x)$, which we tried to avoid all long!!

we cannot reconstruct the eigen vectors of the covariance matrix

→ But we can still compute the compressed representation.

$$\phi(x_i)^T \cdot w_k = \phi(x_i)^T \cdot \left(\sum_{j=1}^{n} \phi(x_j) \alpha_{kj} \right) = \sum_{j=1}^{n} \alpha_{kj} \phi(x_i)^T \phi(x_j)$$

$$\phi(x_i)^T w_k = \sum_{j=1}^{n} \alpha_{kj} \cdot K_{ij}$$

→ for downstream task, this is usually good enough.

→ Typically, the reason why we do kernel PCA is to get these projections and throw away original points and only retain these projections along each of these data points.

⚠ This is where, dimensionality Reduction happens.

(eg → original datapoint was 100 dimension, but then you only care about top 5.

→ So, we map the 100 dimensional data into a 5 dimensional projection onto these 5 most important directions.

→ That becomes a 5 dimensional representation of your 100 dimensional data

→ once, you learn these representation in a low dimensional space, as just the projection onto these eigen directions you can use these projection values as your data point and work this for a downstream task (supervised learning task).

modified

Step (3) compute $\sum_{j=1}^{n} \alpha_{kj} \cdot k_{ij} \quad \forall k$

$$x \in \mathbb{R}^d \longrightarrow \left[\sum_{j=1}^{n} \alpha_{1j} \cdot k_{ij}, \sum_{j=1}^{n} \alpha_{2j} \cdot k_{ij} \cdots \sum_{j=1}^{n} \alpha_{kj} \cdot k_{ij} \right]$$

→ It might increase the dimension, nevertheless it will allow you to capture more complicated relationships.

Note:

→ After step 2, we have to center the kernel.

→ Given, $K \in \mathbb{R}^{n \times n}$

$K_{ij} = k(x_i, x_j) \quad \forall i, j$

create a new kernel K^c — centered.

$$K^c_{ij} = K_{ij} - \theta_i 1_j^T - 1_i \theta_j^T + P 1_i \cdot 1_j^T$$

where,

$$\theta_i = \frac{1}{n} \sum_{k=1}^{n} K_{ik}$$

$$P = \frac{1}{n^2} \sum_{i,j} K_{ij}$$

→ Now, we can say we are able to solve both the issues by using kernel PCA.

– Supervised Algorithms –

Regression Models

ALGORITHM	DESCRIPTION & APPLICATION	ADVANTAGES	DISADVANTAGES
Linear Regression	Linear Regression models a linear relationship between input variables and a continuous numerical output variable. The default loss function is the mean square error (MSE).	1. Fast training because there are few parameters. 2. Interpretable/Explainable results by its output coefficients.	1. Assumes a linear relationship between input and output variables. 2. Sensitive to outliers. 3. Typically generalizes worse than ridge or lasso regression.
Polynomial Regression	Polynomial Regression models nonlinear relationships between the dependent, and independent variable as the n-th degree polynomial.	1. Provides a good approximation of the relationship between the dependent and independent variables. 2. Capable of fitting a wide range of curvature.	1. Poor interpretability of the coefficients since the underlying variables can be highly correlated. 2. The model fit is nonlinear but the regression function is linear. 3. Prone to overfitting.
Support Vector Regression	Support Vector Regression (SVR) uses the same principle as SVMs but optimizes the cost function to fit the most straight line (or plane) through the data points. With the kernel trick it can efficiently perform a non-linear regression by implicitly mapping their inputs into high-dimensional feature spaces.	1. Robust against outliers. 2. Effective learning and strong generalization performance. 3. Different Kernel functions can be specified for the decision function.	1. Does not perform well with large datasets. 2. Tends to underfit in cases where the number of variables is much smaller than the number of observations.
Gaussian Process Regression	Gaussian Process Regression (GPR) uses a Bayesian approach that infers a probability distribution over the possible functions that fit the data. The Gaussian process is a prior that is specified as a multivariate Gaussian distribution.	1. Provides uncertainty measures on the predictions. 2. It is a flexible and usable non-linear model which fits many datasets well. 3. Performs well on small datasets as the GP kernel allows to specify a prior on the function space.	1. Poor choice of kernel can make convergence slow. 2. Specifying specific kernels requires deep mathematical understanding

Classification Models

ALGORITHM	DESCRIPTION & APPLICATION	ADVANTAGES	DISADVANTAGES
SVM	In its simplest form, support vector machine is a linear classifier. But with the kernel trick. it can efficiently perform a non-linear classification by implicitly mapping their inputs into high-dimensional feature spaces. This makes SVM one of the best prediction methods.	1. Effective in cases with a high number of variables. 2. Number of variables can be larger than the number of samples. 3. Different Kernel functions can be specified for the decision function.	1. Sensitive to overfitting, regularization is crucial. 2. Choosing a "good" kernel function can be difficult. 3. Computationally expensive for big data due to high training complexity. 4. Performs poorly if the data is noisy (target classes overlap).
Negrest Neighbors	Nearest Neighbors predicts the label based on a predefined number of samples closest in distance to the new point.	1. Successful in situations where the decision boundary is irregular. 2. Non-parametric approach as it does not make any assumption on the underlying data.	1. Sensitive to noisy and missing data. 2. Computationally expensive because the entire set of n. points for every execution is required.
Logistic Regression (and its extensions)	The logistic regression models a linear relationship between input variables and the response variable. It models the output as binary values (o or rather than numeric values.	1. Explainable & Interpretable. 2. Less prone to overfitting using regularization. 3. Applicable for multi-class predictions.	1. Makes a strong assumption about the relationship between input and response variables. 2. Multicollinearity can cause the model to easily overfit without regularization.
Linear Discriminant Analysis	The linear decision boundary maximizes the separability between the classes by finding a linear combination of features.	1. Explainable & Interpretable. 2. Applicable for multi-class predictions.	1. Multicollinearity can cause the model to overfit. 2. Assuming that all classes share the same covariance matrix. 3. Sensitive to outliers. 4. Doesn't work well with small class sizes.

Both Regression and Classification Models

ALGORITHM	DESCRIPTION & APPLICATION	ADVANTAGES	DISADVANTAGES
Decision Trees	Decision Tree models learn on the data by making decision rules on the variables to separate the classes in a flowchart like a tree data structure. They can be used for both regression and classification.	1. Explainable and interpretable. 2. Can handle missing values.	1. Prone to overfitting. 2. Can be unstable with minor data drift. 3. Sensitive to outliers.
Random Forest	Random Forest classification models learn using an ensemble of decision trees. The output of the random forest is based on a majority vote of the different decision trees.	1. Effective learning and better generalization performance. 2. Can handle moderately large datasets. 3. Less prone to overfit than decision trees.	1. Large number of trees can slow down performance. 2. Predictions are sensitive to outliers. 3. Hyperparameter tuning can be complex.
Gradient Boosting	An ensemble learning method where weak predictive learners are combined to improve accuracy. Popular techniques include XGBoost, LightGBM and more.	1. Handling of multicollinearity. 2. Handling of non-linear relationships. 3. Effective learning and strong generalization performance. 4. XGBoost is fast and is often used as a benchmark algorithm.	1. Sensitive to outliers and can therefore cause overfitting. 2. High complexity due to hyperparameter tuning. 3. Computationally expensive.
Ridge Regression	Ridge Regression penalizes variables with low predictive outcomes by shrinking their coefficients towards zero. It can be used for classification and regression.	1. Less prone to overfitting. 2. Best suited when data suffers from multicollinearity. 3. Explainable & Interpretable.	1. All the predictors are kept in the final model. 2. Doesn't perform feature selection.
Lasso Regression	Lasso Regression penalizes features that have low predictive outcomes by shrinking their coefficients to zero. It can be used for classification and regression.	1. Good generalization performance. 2. Good at handling datasets where the number of variables is much larger than the number of observations. 3. No need for feature selection.	1. Poor interpretability/explainability as it can keep a single variable. from a set of highly correlated variables.
AdaBoost	Adaptive Boosting uses an ensemble of weak learners that is combined into a weighted sum that represents the final output of the boosted classifier.	1. Explainable & Interpretable. 2. Less need for tweaking parameters. 3. Usually outperforms Random Forest.	1. Less prone to overfitting as the input variables are not jointly optimized. 2. Sensitive to noisy data and outliers.

– Unsupervised Algorithms –

Clustering Algorithms

ALGORITHM	DESCRIPTION & APPLICATION	ADVANTAGES	DISADVANTAGES
K-Means	Most common clustering approach which assumes that the closer data points are to each other, the more similar they are It determines K clusters based on Euclidean distances.	1. Scales to large datasets 2. Interpretable & explainable results 3. Can generate tight clusters	1. Requires defining the expected number of clusters in advance. 2. Not suitable to identify clusters with non-convex shapes.
DBSCAN	Density-Based Spatial Clustering of Applications with Noise can handle non-linear cluster structures, purely based on density. It can differentiate and separate regions with varying degrees of density, thereby creating clusters.	1. No assumption on the expected number of clusters. 2. Can handle noisy data and outliers 3. No assumptions on the shapes and sizes of the clusters 4. Can identify clusters with different densities	1. Requires optimization of two parameters 2. Can struggle in case of very high dimensional data
HDBSCAN	Family of the density-based algorithms and has roughly two steps: finding the core distance of each point, and expands clusters from them. It extends DBSCAN by converting it into a hierarchical clustering algorithm.	1. No assumption on the expected number of clusters 2. Can handle noisy data and outliers. 3. No assumptions on the shapes and sizes of the clusters. 4. Can identify clusters with different densities	1. Mapping of unseen objects in HDBSCAN is not straightforward. 2. Can be computationally expensive
Agglomerative Hierarchical Clustering	Uses hierarchical clustering to determine the distance between samples based on the metric, and pairs are merged into clusters using the linkage type.	1. There is no need to specify the number of clusters. 2. With the right linkage, it can be used for the detection of outliers. 3. Interpretable results using dendrograms.	1. Specifying metric and linkages types requires good understanding of the statistical properties of the data 2. Not straightforward to optimize 3. Can be computationally expensive for large datasets
OPTICS	Family of the density-based algorithms where it finds core sample of high density and expands clusters from them. It operates with a core distance (e) and reachability distance.	No assumption on the expected number of clusters. 2. Can handle noisy data and outliers. 3. No assumptions on the shapes and sizes of the clusters. 4. Can identify clusters with different densities. 5 Not required to define fixed radius as in DBSCAN.	1. It only produces a cluster ordering. 2. Does not work well in case of very high dimensional data. 3. Slower than DBSCAN.

Dimensionality Reduction Techniques

ALGORITHM	DESCRIPTION & APPLICATION	ADVANTAGES	DISADVANTAGES
PCA	Principal Component Analysis (PCA) is a feature extraction approach that uses a linear function to reduce dimensionality in datasets by minimizing information loss.	1. Explainable Interpretable results. 2 New unseen datapoints can be mapped into the existing PCA space 3. Con be used as dimensionality reduction technique as preliminary step to other machine learning tasks 4 Helps reduce overfitting 5. Helps remove correlated features	1. Sensitive to outliers 2. Requires data standardization
t-SNE	t-distributed Stochastic Neighbor Embedding is a non-linear dimensionality reduction method that converts similarities between data points to joint probabilities using the Student t-distribution in the low-dimensional space	1. Helps preserve the relationships seen in high dimensionality 2. Easy to visualise the structure of high dimensional data in or 3 dimensions 3. Very effective for visualizing clusters or groups of data points and their relative proximities	1. The cost function is not convex: different initializations can get different results. 2. Computationally intensive for large datasets. 3. Default parameters do not always achieve the best results
UMAP	Uniform Manifold Approximation and Projection (UMAP) constructs a high-dimensional graph representation of the data then optimizes a low-dimensional graph to be as structurally similar as possible.	1. It can be used as general-purpose dimension reduction technique as o preliminary step to other machine learning tasks. 2. Can be very effective for visualizing clusters or groups of data points and their relative proximities. 3 Able to handle high dimensional sparse datasets	1. Default parameters do not always achieve the best results
ICA	Independent Component Analysis (ICA) is a linear dimensionality reduction method that aims to separate a multivariate signal into additive subcomponents under the assumption that independent components are non-gaussian. Where PCA "compresses" the data, ICA "separates" the information.	1. Can separate multivariate signals into its subcomponents 2 Clear aim of the method: only applicable if there are multiple independent generators of information to uncover. 3. Can extract hidden factors in the data by transforming a set of variables to new set that maximally independent.	1. Without any prior knowledge, determination of the number of independent components or sources can be difficult. 2. PCA is often required as a pre-processing step.

Association Rules

ALGORITHM	DESCRIPTION & APPLICATION	ADVANTAGES	DISADVANTAGES
Apriori algorithm	The Apriori algorithm uses the join and prune step iteratively to identify the most frequent itemset in the given dataset. Prior knowledge (apriori) of frequent itemset properties is used in the process.	1. Explainable & interpretable results. 2. Exhaustive approach based on the confidence and support.	1. Requires defining the expected number of clusters or mixture components in advance 2. The covariance type needs to be defined for the mixture of component
FP-growth algorithm	Frequent Pattern growth (FP-growth) is an improvement on the Apriori algorithm for finding frequent itemsets. It generates a conditional FP-Tree for every item in the data.	1. Explainable & interpretable results. 2. Smaller memory footprint than the Apriori algorithm	1. More complex algorithm to build than Apriori 2. Can result in many (incremental) overlapping/trivial itemsets
FP-Max Algorithm	A variant of Frequent pattern growth that is focused on finding maximal itemsets.	1. Explainable & Interpretable results. 2. Smaller memory footprint than the Apriori and FP-growth algorithms	1. More complex algorithm to build than Apriori