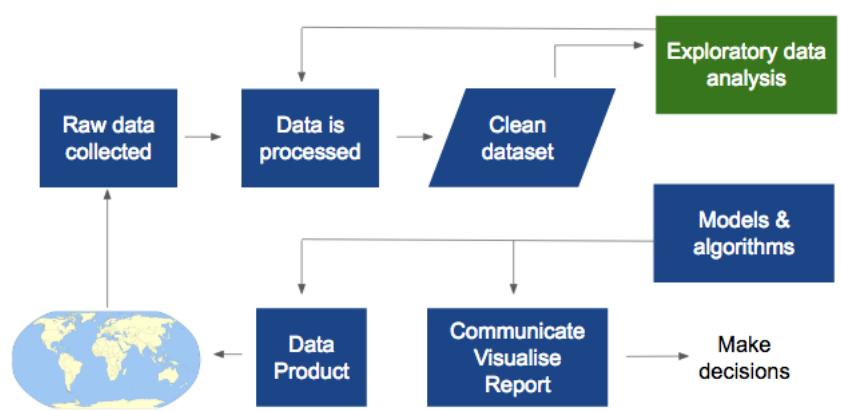
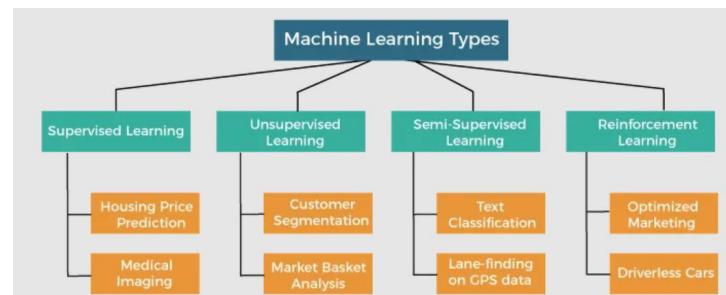


## - Understanding Machine Learning Algorithms: Handwritten Notes -

### Table of Contents

1. What is Machine Learning?
2. What are the Types of Machine Learning?
3. Supervised Machine Learning
4. Unsupervised Machine Learning
5. Reinforcement Learning
6. Semi-Supervised Learning
7. Steps in ML Project
8. Exploring Step 1 Data Collection
9. Exploring Step 2 Data Preparation
  - Exploratory Data Analysis
  - Data Preprocessing
  - Feature Engineering

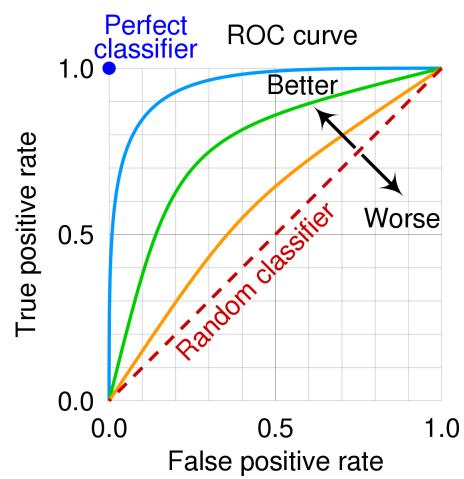


Notes by RaviTeja G



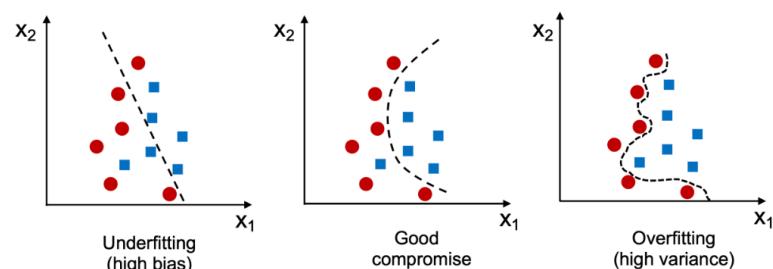
## 10. Exploring Step 3 - Train Model on Dataset

- Types of Learning
- Under Fitting and OverFitting
- Regularization techniques
- Hyperparameter Tuning



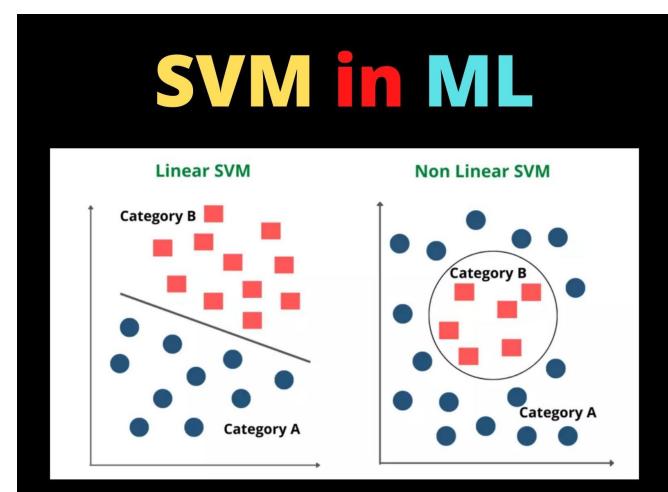
## 11. Exploring Step 4 - Evaluation of a Model

- Evaluation Metrics
- Confusion Matrix
- Recall/Sensitivity
- Precision
- Specificity
- F1 Score
- AUC and ROC Curve
- Analysis of a Model



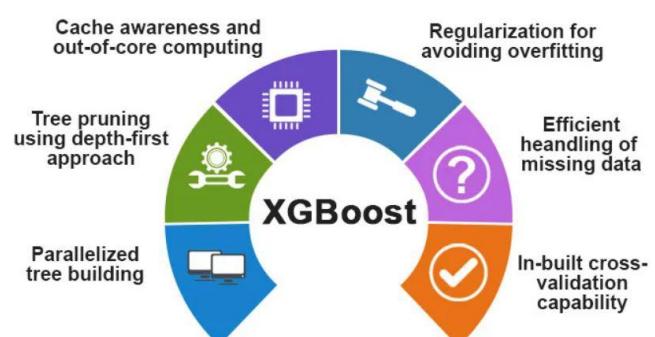
## 12. Supervised Learning

- 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



Download Deep Learning Notes

<https://t.me/AIMLDeepThaught/663>

## 14. Cheat Sheet of Supervised and Unsupervised Algorithms



# Machine Learning

## Introduction:-

→ At a high-level, machine learning is simply the study or teaching a computer program/algo. 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 three, 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' or 'machine learning', meaning it is the algorithms that turn a dataset into a model.

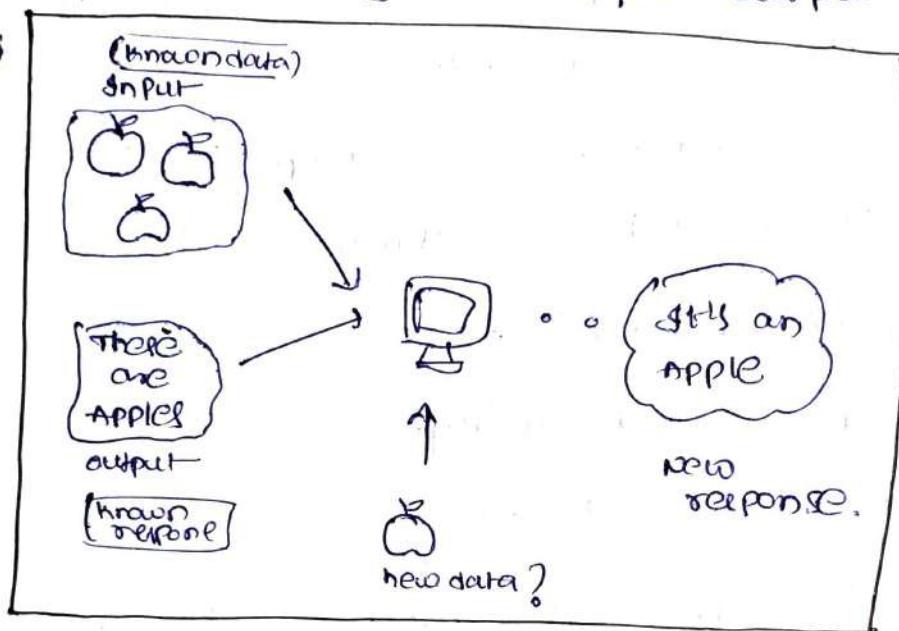
- \* which kind of algorithm works best and what to choose (supervised, unsupervised, classification, regression, etc...) depends on the kind of problem you're solving, the computing resources available & the nature of the data.

## Brickling 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 example 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 applications  
Face Recognition

Face look 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 classification

Spam filter is a supervised learning system. fed emails 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 ads you see as you browse are placed there because a learning algorithm said that those were of reasonable popularity (more clicked).

## 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

4) Logarithmic Regression

### Applications

① Risk Assessment

② Score prediction etc.

### Classification

→ we use classification algorithms for predicting 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

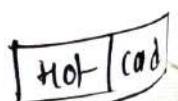
### Applications

① Fraud detection

② Email spam detection

③ Image classification

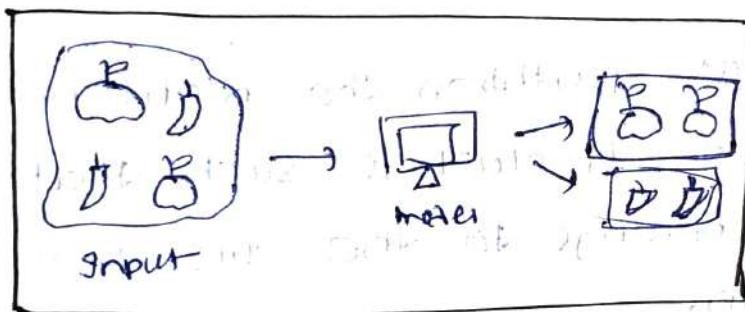
④ Diagnosis etc..



## ② unsupervised learning

[Based on Hebbian learning)

- To easily understand, in case of unsupervised learning algorithm, the data is not explicitly labeled into diff. classes, 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) Anomaly 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

### Download Deep Learning Notes

<https://t.me/AIMLDeepThaught/663>

In this we partition the n observations in the data into K clusters 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 & marks the outliers in low density regions.

### c) Hierarchical clustering

A hierarchy of clusters 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-dimensional 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 example are normal by observing the instance that fit the remainder dataset.

### one-class classification

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

Algorithms for doing so include -

- ① one-class k-means
- ② one-class SVM
- ③ Isolation Forest

#### 4) Association

we use association algorithms for allocating co-occurring items / events.  
Association algorithms

• Apriori

→ 5) Autoencoder (Dimensionality Reduction)  
Autoencoders take input data, compress data onto a code, then tries to recreate the input data from learned code.

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

#### Note

→ Autoencoder can also be used to find outliers (anomaly).

**Download Deep Learning Notes**

<https://t.me/AIMLDeepThaught/663>

### ③ Reinforcement Learning

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

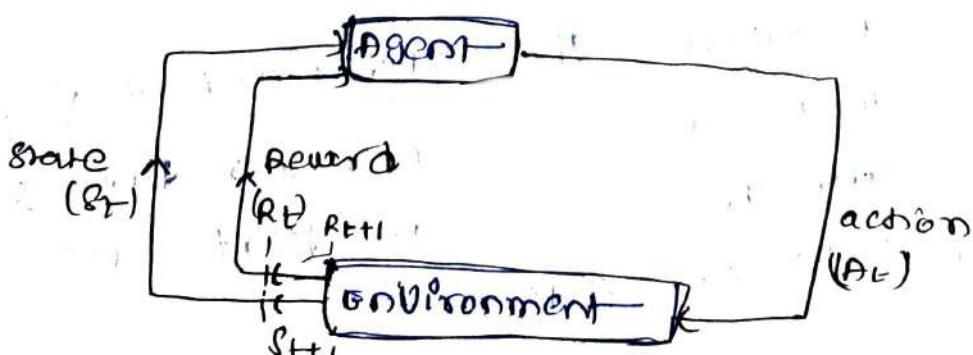
→ Use trial & error method. It finds in a trial then it learns & tries & it learns etc.

→ Games are a real example of RL.

Take a chess game model. All it has is the environment. First AI takes a step & feels the outcome & 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 tasks. As the agent takes action that goes toward the goal, it receives a reward.

overall aim → Product of best next step to earn the best reward.

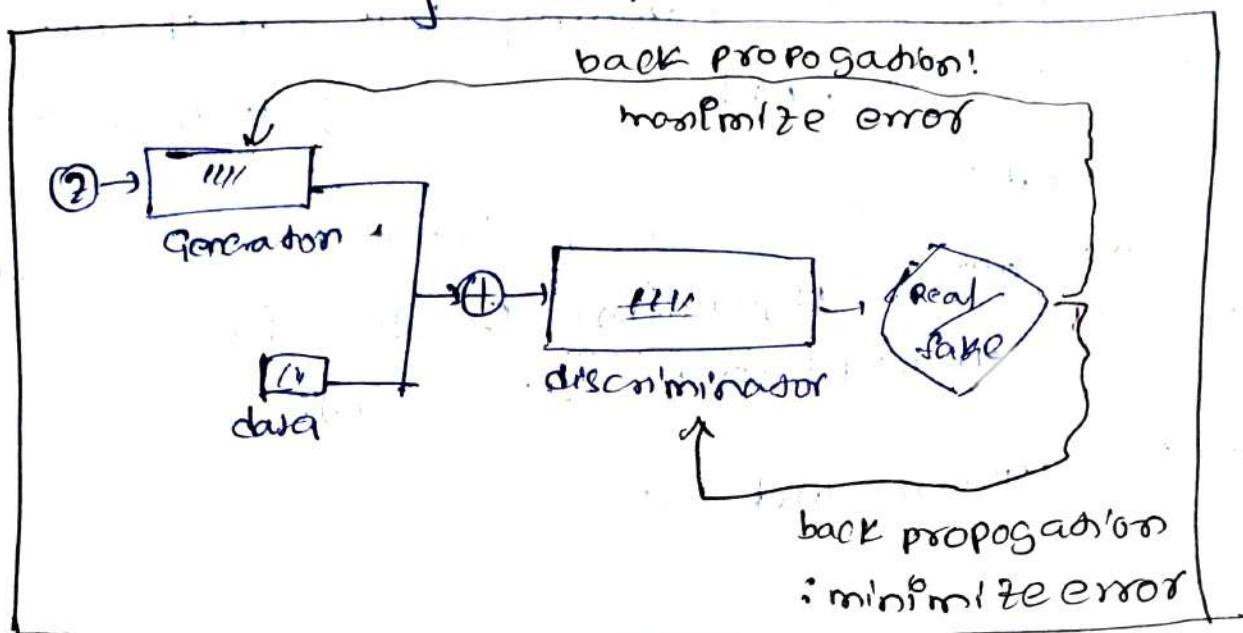


When there is no train. dataset, it learns from experience.

#### 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 'generative adversarial networks' / [GAN's]  
like two deep learning networks in competition, each trying to outsmart the 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, 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.



e.g.

→ common situations for this kind of learning are medical images like CT scans (or) MRI's.

A trained radiologist can go through & label a small subset of scans for tumors or diseases.

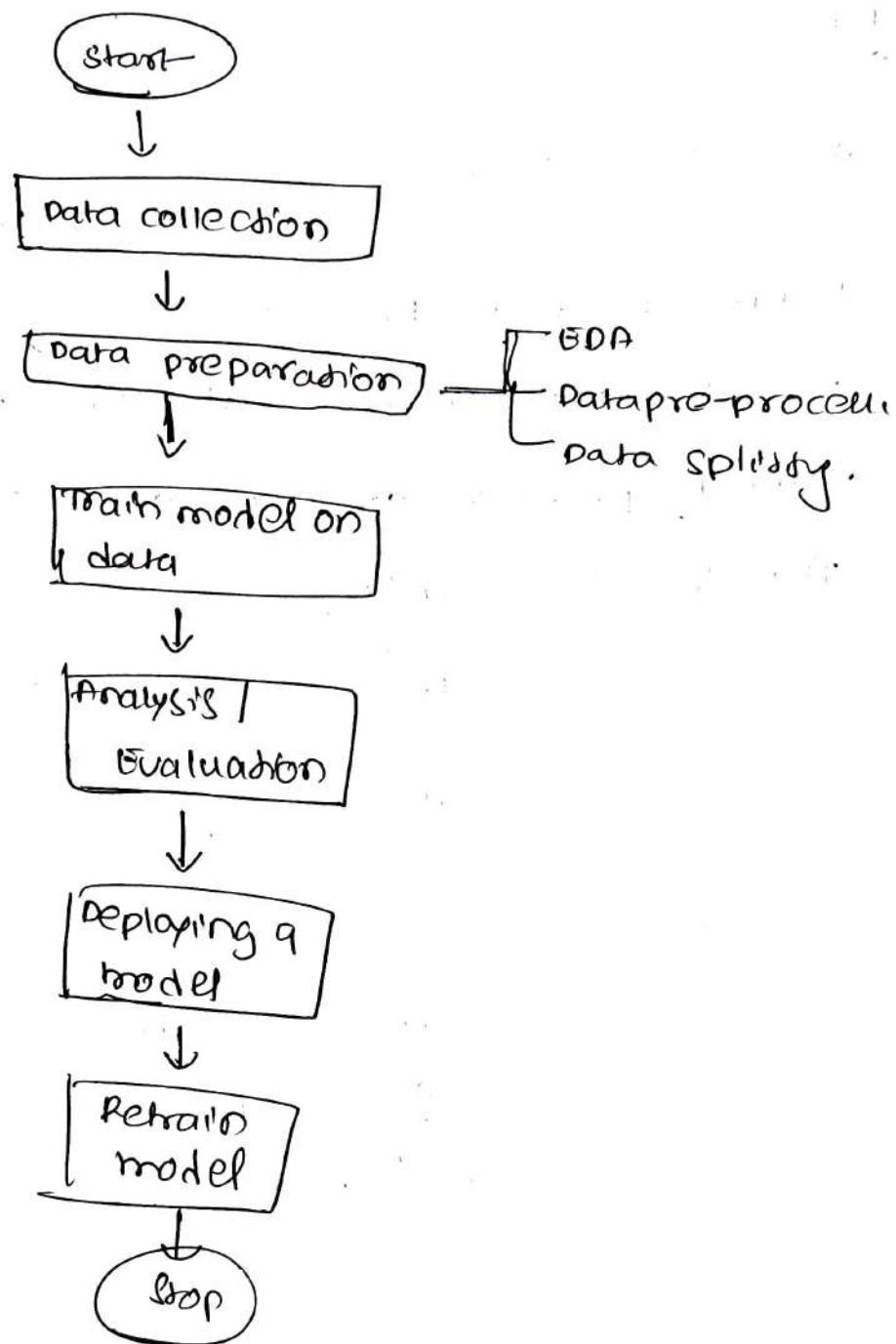
→ it would be too time-intensive & costly to manually label all the scans - but the deep learning network can still benefit from the small proportion of labeled data & improve its accuracy compared to fully unsupervised models.

→ 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 ~~some~~ a algorithm & see it.  
you will understand the path.  
so, let's understand the path.

## Download Deep Learning Notes

<https://t.me/AIMLDeepThaught/663>

## 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

###### i) structured data?

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

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

###### ii) unstructured data?

Data with no rigid structure.

(Image, video, natural language text, speech,

## Step②

### Data preparation

- It has 3 main process.
  - Exploratory data analysis (EDA), understanding data.
  - Data preprocessing, preparing your data to be modelled.
  - Data splitting.
- a) EDA, understanding the data

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

[Ex: 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]

- what kind of data do you have?  
structured, unstructured, categorical / numerical.
- Are there missing values? Should you remove them / fill them with feature imputation?
- what are outliers?  
How many of them are there?  
Are they out by much ( $3 + 2\sigma$ )?  
Why are they there?
- Are these questions you could ask a domain expert about the data?

Download Deep Learning Notes

## b) data preprocessing, preparing your data to be modeled

### ① Feature imputation

(filling missing values)

A ML model can't learn on data that isn't there.

i) single imputation: fill with mean / median or

ii) multiple imputation:  
model other missing values  
column

iii) KNN (K-nearest neighbor):  
and fill with what your model finds.

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 1's & 0's where the target value is 1 & the rest are 0's.

e.g. 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) Label Encoder :-

Turn labels into distinct numerical values

Eg:-

If your target variables are diff.

animal 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 (st. var.).

Standardization is more robust to outliers than scaling.

## ④ Feature Engineering

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

### i) Decompose

Like, turn a date (such as 2020-06-18 to 20:16: etc.) into hour-dr-day, day-dr-week, day-dr-month, ts-holt 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)

→ depending on situation sometimes 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 significant

+ Create a X-missing feature for whenever a column X contains a missing value.

## ⑤ Feature Selection :-

- Selecting the most valuable feature or 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 Imbalance :-

Does your data have 10,000 examples or one day, but only 100 examples the other?

- 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 <sup>data</sup> 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)

model's final performance is evaluated on this.

④ Do not use this dataset to tune the model

#### a. 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 produced a model with 10% generalization error, say just 5%.

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

④ 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 models.

This is called "overfitting".

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

④

### Step③

#### Train model on dataset

It has 3 main process.

a) choose an algorithm

b) overfit the model

c) Reduce overfitting with regularization.

i) Algorithm can be choosed by understanding supervised, unsupervised (Explained previous)

b) Type of learning

i) Batch Learning

All or your data exists in 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.

**Download Deep Learning Notes**

### iii) transfer learning

Take the knowledge one model has learned & use it with your own. Give you the ability to leverage SOTA (state of the art) models for your own problems.  
Helpful if you don't have much data / want compute resources.

### iv) Active learning

Also referred to as "human in the loop" learning. A human expert interacts with a model & provides updated 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".  
(e.g. 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.

ii) Dropout:  
→ L2 constrains a model's features (won't set them to 0)

Randomly remove parts of your model.  
So the network 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  
image, randomly rotate, skew, flip &  
adjust the height of your images.
  - this makes your model have to learn  
similar patterns across different styles of  
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 parameter to  
update.

may be a replacement for dropout in  
some networks.

## Hyperparameter Tuning

→ Run a bunch of experiments with different 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  
(e.g. 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 every large value (e.g. 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 the curve. (P326 - Handwritten)

ii) Learning rate scheduling involves decreasing the learning rate slowly as model learns more (get closer to convergence). (see Adam Optimizer)

#### iii) cyclic learning rates

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

other hyperparameters you can tune

- 1) No. of layers (Deep learning network)
- 2) Batch size [How many es or data your model sees at once.]
- 3) No. of trees (Decision tree algory)
- 4) No. of iterations [How many times model goes through the data depends on algory. (can use early-stopping)]

[Download Deep Learning Notes](https://t.me/AIMLDeepThought/663)

## Step ④

### Analysis / Evaluation

to estimate how well a model will generalize to out-of-sample data.

#### a) Evaluation metrics:-

##### i) Classification

ROC & AUC curves

Accuracy

Precision

Recall

F1 score

Confusion matrix

##### ii) Regression

MSE (mean squared error)

MAE (mean absolute error)

R<sup>2</sup> (r-squared)

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

#### i) Classification accuracy:-

Percentage of correct predictions. ( $\frac{Y_{\text{pred}}}{Y_{\text{true}}}$ )

##### 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 model.

It gives a baseline to compare our

model's accuracy.

Classification accuracy is easy, but it doesn't tell you the underlying distribution or response class.

And it doesn't tell you what type of error your classifier makes.

## a) confusion matrix

FP - False Positive  
FN - False Negative.

Predicted			
		0	1
Actual 0	TN	FP	
Actual 1	FN	TP	

they actually don't have corona. But we predicted they have.

they actually have, but we say they don't

TP% which are actually correct

TN% which are actually wrong

FP% they are actually wrong,

But predicted as true.

FN% they are actually true,  
But predicted as False

classification

accuracy

$$\rightarrow \frac{TP + TN}{TP + TN + FP + FN}$$

How often is the classifier correct?

talks about  
correct.

misclassification rate

$$\rightarrow \frac{FP + FN}{TP + TN + FP + FN}$$

Recall / sensitivity

[True positive rate]

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

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

Predicted Pos.

TP

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

→ It depends on FN,

which were actually true, but predicted false.

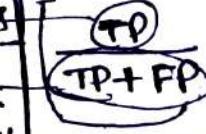
out of total positive ones,  
How many positive I am able to Recall

$$\text{Actual Pos} = TP + FN$$

## Precision & Recall linked with TP

### Precision

Really Pos  
Total Pos.  
actually



$$\frac{TP}{TP+FP} \rightarrow \text{Really Pos}$$

$\frac{TP}{TP+FN}$  → Predicted Pos

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

→ How "precise" is the classifier when  
Predicting positive individual.

out of all  
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)

Pred		
0	1	
0	TN	FP
1	FN	TP

Healthy predicted as sick.

Sick predicted as healthy.

→ now analyze the cost

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

So,  $\text{cost of FN} > \text{cost of FP}$

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

Eg ②

1 = Spam

0 = Not Spam

Pred		
0	1	
0	TN	FP
1	FN	TP

Not spam predicted as spam.

Spam got into mail box.

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

So,  $\text{cost of FP} > \text{cost of FN}$

Hence, it depends on FP, we go for "Precision".

(29)

Sensitivity P.

$$\boxed{\frac{TN}{TN + FN}}$$

False positive rate

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

FB-Score

when you want to consider both  
 PP & PN like we need both  
 Sensitivity (recall) & precision,  
 In such case we choose F-Beta.

General Formula

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

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

In beta, there is possibility for 3 ranges.

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

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

- 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 curves

can see how specificity & 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 model take to train?

Is this feasible?

How long does inference take?

Is it suitable for production?

### 3) using what-if tool

what-if rechanges bring in data?

How does this effect the outcome?

### 4) Bias / variance trade-off

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

→ High variance results in overfitting an  
the model finding patterns in the  
data which are actually random

**Download Deep Learning Notes**

Step 5

## Deploying a model

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

→ Step 6

## Retrain model

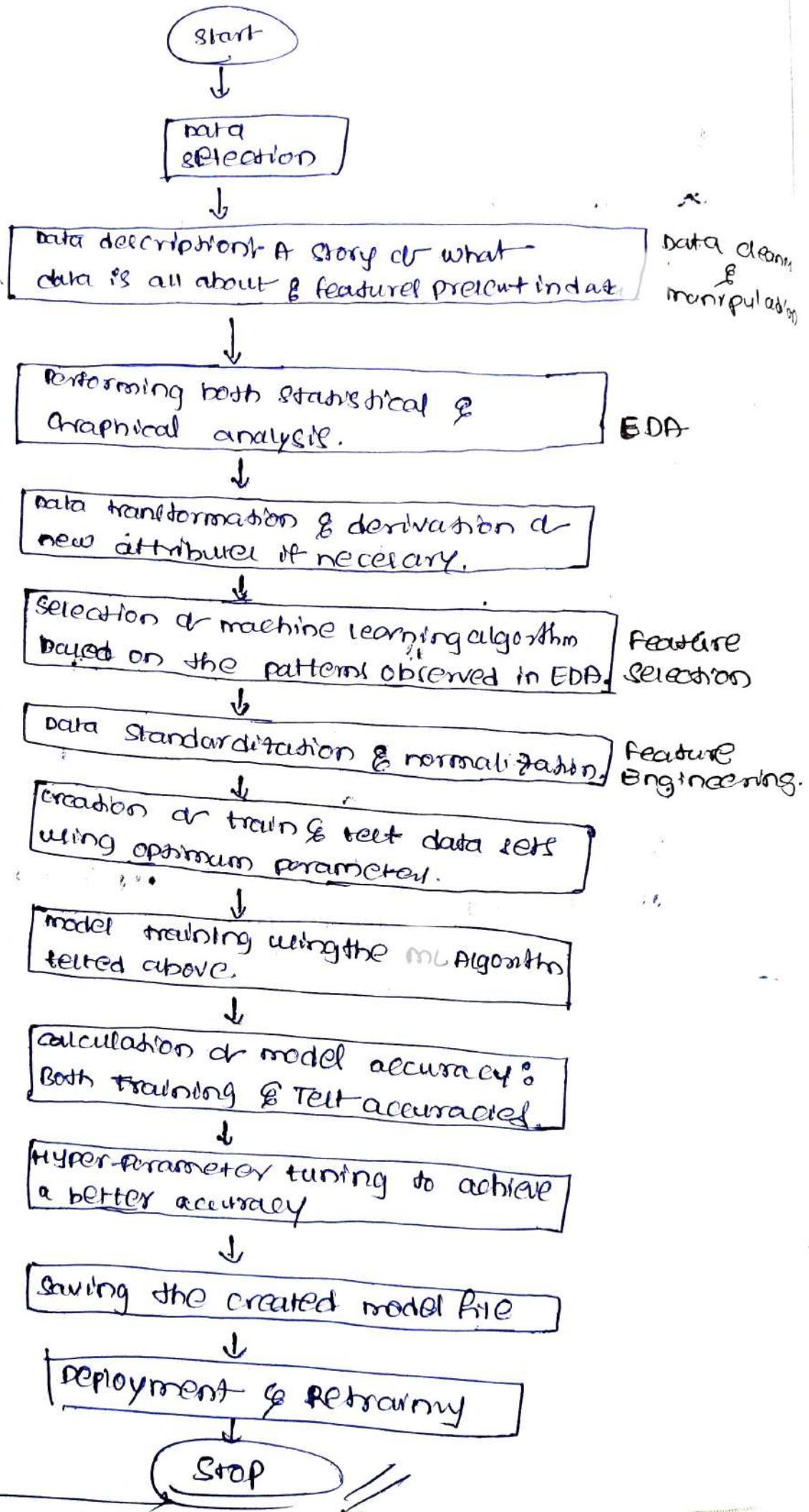
→ See how the model performs after serving based on various evaluation metrics & revisit the above steps as reqd.

→ You will find your model's prediction starts to lag or drift, as in when data source change or upgrade. This is when you will want to retrain it.

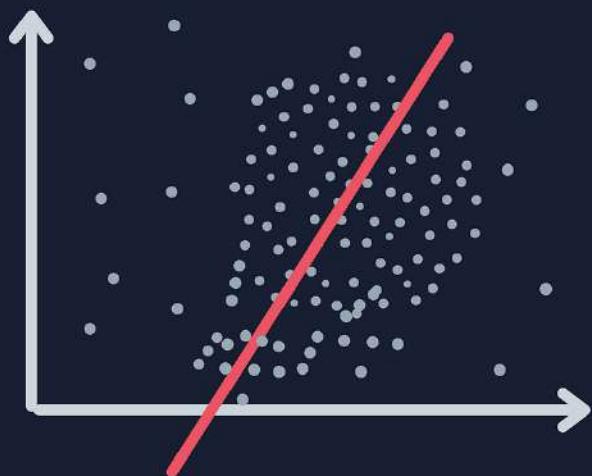
Stop / NO END

DO IT

so, basically the flowchart :-



# Linear Regression



## Table of Contents

1. What is Linear Regression
2. Understanding with an example
3. Evaluating the fitness of the model
4. Understanding Gradient descent
5. Understanding Loss Function
6. Measuring Model Strength
7. Another Approach for LR - OLS

[Download Deep Learning Notes](#)

<https://t.me/AIMLDeepThaught/663>

## 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 w/ a linear surface called a "hyperplane".

→ This will be clear with an ex.  
lets 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

$$\begin{matrix} H \propto W \\ \downarrow \\ W = aH \end{matrix}$$

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

→ By observation or second like

$$H \propto W$$

→ To remove proportionality we can multiply a constant

$$W = mH$$

→ But, it may also differ by some value,  $l_0, \pm b$

$$W = mH + b$$

→ Now, this is the relation of height & weight.

constants

$w = mH + b$  → now, here if I know  
the value of 'm' & 'b'.  
then I can predict the weight.

Given height

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

→ Let's take a sample, (5.6, 60), (5.7, 62).

$$\begin{aligned} 60 &= m(5.6) + b \quad \text{①} \\ 62 &= m(5.7) + b \quad \text{②} \end{aligned} \quad \left. \begin{array}{l} \text{By this we get} \\ m = 20, b = -52. \end{array} \right.$$

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

lets say with a known height 5.9.

$$w = 20(5.9) - 52 = 66 \text{ kg.}$$

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

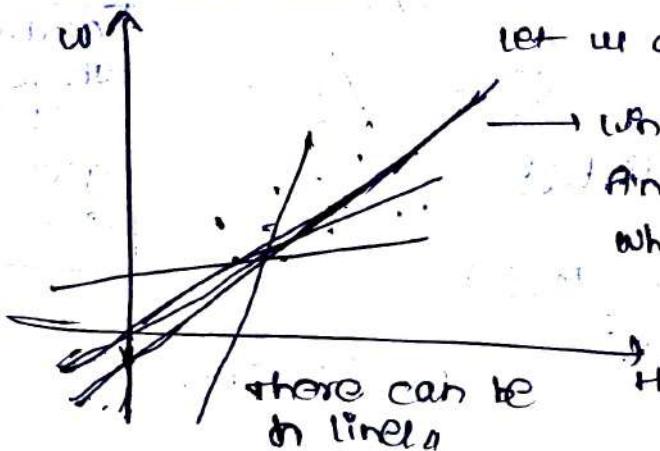
④ It may be or we change ['m' & 'b'] then  
we could get an exact value (approx.)

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

→ so, when we say a model,

④ 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 bw variable  
which is a straight line.

Hence,  $y = mx + c$

→ we could find n diff lines.

But that one line which can predict

all the values approx. is the "Best fit line"

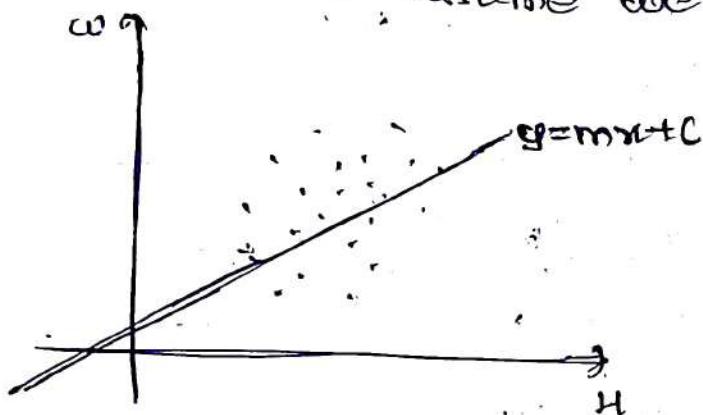
$y = mx + c$  → And this best fit line is called model  
→ we have to find the 'm' & 'c'.

m-slope, c-Intercept

\* Evaluating the fitness of the model with a

cost function

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



But, now we have doubt.

How did we opt this line?  
Why this line?  
What about other lines?

→ Now, if it's the best fit line, then procedure  
if I want the weight or 'w'.

→ It is nothing but the value of y-coordinates  
 $y = mx + c$  like worth of coordinate '6,0'

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

I got  $w = 66$ , but actually  $w = 62$ .

So, I can see there is a known error.

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

Q) Why did we get this loss?

A) may be best fit isn't being taken.

Always  
there  
will be  
a slight  
error

## Gradient Descent

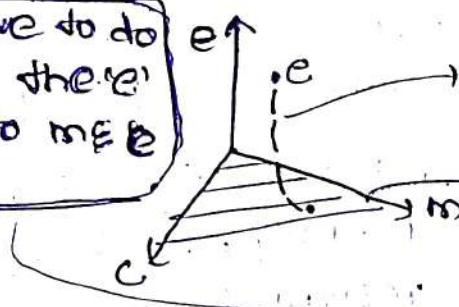
→ Now, to improve my model strength  
I have to reduce the loss.

and this loss depends on line "c" taken.

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

We have loss( $e$ ), slope( $m$ ), constant( $c$ ).

All I have to do  
if bring these  
down to  $m$  &  $c$   
plane



should bring down &  
find the new coordinate

On this plane we  
have no errors.

② Slowly, by examining each input  $m$  &  $c$  gets changed and eventually loss decreased. This process is called learning.

So,

$$m_{\text{new}} = m_{\text{old}} - \eta \Delta m$$

$$c_{\text{new}} = c_{\text{old}} - \eta \Delta c$$

$m_{\text{new}}$  - changed one, new

$m_{\text{old}}$  - old  $m$  value.

$\eta$  - learning rate

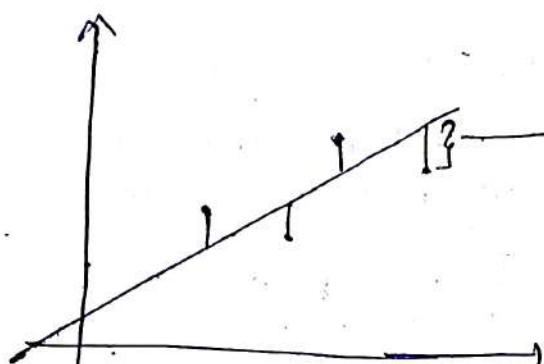
$\Delta m$  - Error in  $m$  (de/dm)

→ So, here we are taking a new  $m$ .

By subtracting a fraction of the error rate from old fraction, we get 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  $|y_i - \hat{y}_i|$

→ 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'.

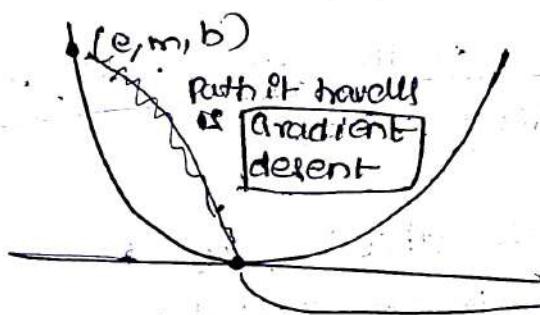
$$(\text{Sum of residuals}) \quad E_n = \sum_{i=1}^n [y_i - (m x_i + c)]$$

so, to cancel the negative value, we take square

$$R(x) = \sum_{i=1}^n y_i^2 = \sum_{i=1}^n [y_i - (m x_i + c)]^2$$

sum of squares  
of residuals,

now, it is a quadrature in 3D.



→ so, we are supposed to shift the  $(c, m, b)$  to the origin  $(0, m, b)$ .

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

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

$$\frac{\partial R}{\partial m} = \sum_{i=0}^n \alpha_i (b + mx_i - y_i)$$

$$\frac{\partial R}{\partial b} = \sum_{i=0}^n \alpha_i (b + mx_i - y_i)$$

so, we know for best line, residual should be min.  
minima or max. occurs where derivative = 0.  
i.e.,

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

$$\sum_{i=0}^n \alpha_i (b + mx_i - y_i) = 0$$

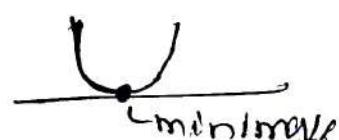
$$\sum_{i=0}^n \alpha_i c + \sum_{i=0}^n \alpha_i m x_i^2 + \sum_{i=0}^n \alpha_i y_i x_i = 0$$

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

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

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

ideally, if we have an eq. or one  
dependent & one independent variable  
the minima will look like



dependent var  $y = m x + c$  independent var

→ the new values for slope & intercept are calculated.

repeat until convergence

{

$$m_{\text{new}} = m_{\text{old}} - \eta \left[ \sum_{i=1}^n (h_0 \cdot x_i - y_i) x_i \right] \xrightarrow{\frac{\partial E}{\partial m}}$$

$$c_{\text{new}} = c_{\text{old}} - \eta \left[ \sum_{i=1}^n (h_0 \cdot x_i - y_i) \right] \xrightarrow{\frac{\partial E}{\partial c}}$$

}

Accuracy this is how LR works.

measuring model strength RSE - residual sum of squares RSS - total sum of square

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

It takes the form of proportion (proportion explained).

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

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

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

It  $R^2 = 0.75$

It says that our model fits 75% of data points.

Download Deep Learning Notes

<https://t.me/AIMLDeepThought/663>

## Adjusted R<sup>2</sup> Statistic

→ As R<sup>2</sup> is just a linear eq., as we increase the no. of independent variables in our eq., the R<sup>2</sup> increases as well.

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

i.e. R<sup>2</sup> will increase, but it is not necessarily model yields better results.

To rectify this, we use adjusted R<sup>2</sup> value which penalises excessive use of such features which do not 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 eqs)

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

→ covariance is a measure of how much two variables change together. If the variables increase together, their cov is positive.

o - no relation, -ve → one increase & one decrease

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

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

→ variance of explanatory variable.

→ covariance of the response, explanatory variables.

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

$$y = \beta x + \alpha$$

$$\alpha = \bar{y} - \beta \bar{x}$$

now can see

Note :-

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

the error terms have a constant variance.

### Homo<sup>s</sup>cedasticity

- the error terms are normally distributed.
- no multicollinearity, i.e., no independent variables should be correlated with each other or affect another.

### model confidence

(or) is linear regression a low bias / high variance model

(or) a high bias / low variance model?

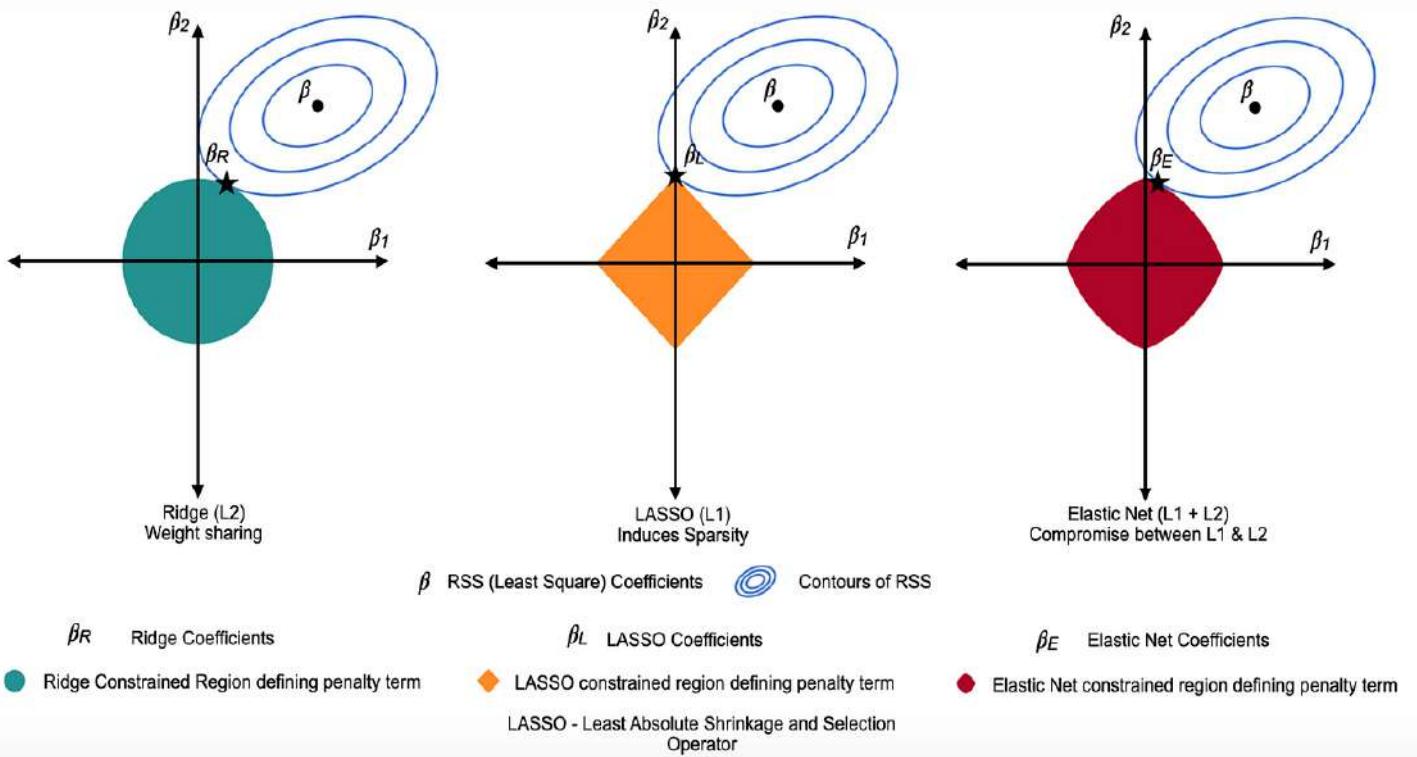
→ It's a "high bias / low variance" model.

→ Even after repeated sampling, the fit line won't stay roughly in the same pos. (low variance)

→ But, the avg'd the models created after repeated sampling won't do a great job in capturing the p'st relationship.

→ Low variance is helpful when we don't have well behaved data.

```
import statsmodels.formula.api as smf  
model_name = smf.ols(formula='y ~ b1, b2')  
model_name.conf_int()
```



## Table Of Contents

1. Understanding Multicollinearity
2. Variance Inflection Factor
3. Regularization
4. Lasso - L1 Form
5. Ridge - L2 Form
6. Elastic Net
7. Difference Between Ridge and Lasso
8. When to use Ridge/Lasso/Elastic Net
9. Polynomial Regression

[Download Deep Learning Notes](#)

<https://t.me/AIMLDeepThaught/663>

## Multi-collinearity

- we can define multi-collinearity at the situation where the independent var have strong correlation among themselves.
- the co-eff. in a linear Reg model represent the extent or change in  $y$  when a certain  $x_1(x_1, x_2, x_3, \dots)$  is changed keeping other constant. But if  $x_1 \& x_2$  are dependent then still assumption itself is wrong that (we are) changing one vars keeping others constant all the dependent var will also be changed.

It means model becomes a bit flawed.

$$y_2 = m_1x_1 + m_2x_2 + c$$

while,

$$m_2 = a_1 + d$$

Independent var ( $m_1, m_2$ )  
are related.

more than one linear 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.

④ 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.

#### ⑤ Variance Inflation Factor (VIF)

Regression on one X var against other X variables.

$$\text{VIF} = \left( \frac{1}{1-R^2} \right)$$

$$\text{VIF} < 5$$

✓ perfect

→ If  $\text{VIF} > 5$ , extreme correlation is avoided.

#### Remedies for multi-collinearity

- ① de-coeffing: if corr not extreme / the var not used the ignore
- ② remove one-var like in dummy var trap (one hot encoding)
- ③ combine correlated vars - combine variables.
- ④ Principle component 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.  
for,

simply reducing the order degree or a polynomial  
by reducing their corresponding weight.

### Reform for Regularization

→ in a linear eq, 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.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$$

$$RSS = \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j \cdot x_{ij})^2$$

To regularize a model,  
shrinkage penalty is added to cost function,

**Download Deep Learning Notes**

<https://t.me/AIMLDeepThought/663>

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

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

The regularization term is

given by,  $\boxed{\text{regularization} = \lambda * \sum |B_j|}$ .

shrinkage factor

→ Loss after regularization is -

new loss func.

$$\boxed{\text{RSS} + \lambda \cdot \sum_{j=1}^p |B_j| = \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j \cdot x_{ij})^2 + \lambda \cdot \sum_{j=1}^p |B_j|}$$

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

## Ridge Regression (L2 form)

→ Ridge Regression penalizes the model based on the sum of squares of mag of coeff.

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

$\lambda$  can be calculated by cross validation

→ loss after regularization,

$$\boxed{\text{RSS} + \lambda \cdot \sum_{j=1}^p B_j^2 = \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j \cdot x_{ij})^2 + \lambda \cdot \sum_{j=1}^p B_j^2}$$

Note :- consider  $B_1 \in R_2$  be coeff of LR if  $\lambda = 1$ .

- ① for LASSO,  $\boxed{\beta_1 + \beta_2 \leq S}$  → where 'S' is the max value the eq. can achieve.  $\beta_1$
- ② for Ridge,  $\boxed{\beta_1^2 + \beta_2^2 \leq S}$

## Elastic net

→ middle ground b/w Ridge & Lasso

→ the regularization term is a simple mix of both Ridge & Lasso, and you can control how much  $\alpha$ .

$$\text{Lenet}(\hat{\beta}) = \frac{\sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1})^2}{2n} + \lambda \left[ \left( \frac{1-\alpha}{2} \right) \cdot \sum_{j=1}^m \hat{\beta}_j^2 + \alpha \sum_{j=1}^m |\hat{\beta}_j| \right]$$

→ where  $\alpha$  is the mixing parameter

Ridge ( $\alpha=0$ )

Lasso ( $\alpha=1$ )

## ④ Difference between Ridge & Lasso

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

$y = 0.0016 - \text{very less}$   
 $R^2 = 0.0016 - \text{very less}$

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

→ whereas in Lasso, the L1 penalty does reduce some coefs exactly to zero, when we use a sufficiently large tuning parameter  $\lambda$ .

so, in addition to regularizing, Lasso also performs feature selection.

## → why use regularization?

- 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 ( $\lambda$ ) controls this bias & variance trade off, selected via cross-validation.
- when ' $\lambda$ ' is increased to certain point, it reduces the variance without losing any imp prop of data.
- But after certain point model will start lossing imp prop, which will increase bias in the data.

as what should be used plain linear, Ridge, Lasso or Elasticnet?

- Ans
- 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 prefer that only a few features are actually useful, you should prefer Lasso/EN, since they tend to reduce the useless feature 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 standard scalar

Notes 1

① standardScaler()

$$\frac{x - \bar{x}}{s_x}$$

$$\bar{x} = 0$$
  
$$s_x^2 = 1$$

② two ways to find multicollinearity →

correlation  
VIF

③ 4 ways to remove multicollinearity,

④ train set, high accuracy; test set, low accuracy.  
→ overfitting, so regularize it.

→ even after checking Lasso and Ridge,

it still gives the same accuracy values,  
then you can't tell if it is over-fitted.

e.g. just by having huge difference in  
train accuracy & test accuracy than it  
doesn't just mean overfitted. Test before  
judging! In this case model is not problem,  
more data is needed.

⑤

The Linear & Non Linear Regression does not talk  
about the degree or polynomial eqn. in  
terms of dependent varn.

Instead, it talks about the degree of coeff.

$$y = a + b_1x_1 + b_2x_2 + \dots + b_nx_n$$

It's not talking about power of x<sub>1</sub>, x<sub>2</sub>, ...

but the power of a, b, c etc. And their powers<sup>n</sup>

Hence it is linear regression

## Polynomial Regression

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

on 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 do we?

④ If data is scattered in a polynomial curve shape (not linear) then keep trying different degrees until the line fits the data.

### Code

```
from sklearn.preprocessing import PolynomialFeatures
```

```
Poly-reg = PolynomialFeatures(degree=2)
```

```
X_Poly = Poly-reg.fit_transform(X)
```

→ fit the quadratic function  
→ keep changing degree until it fits the data

\* Logistic regression is not Regression analysis

## ② Logistic Regression

\* Probability based model

Linear model for classification  
(supervised learning)

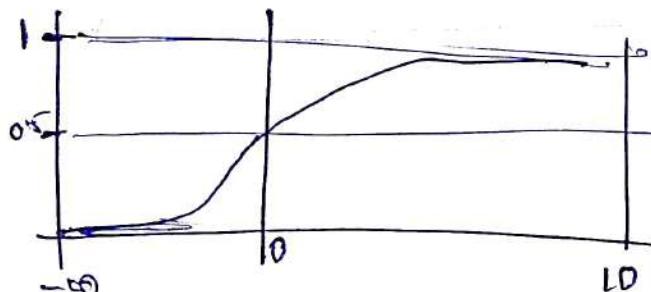
→ Logistic Regression is used for performing  
'classification problems'.

→ 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 of given variable  
belonging to a certain class.  
Thus, it is obvious that the value of  
 $y$  should be between 0 & 1.

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



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

## so why we sigmoid?

- ① the sigmoid function's range is bounded between 0 & 1, which is what we need.
- ② easy to calculate other functions which is useful during gradient descent calculation.
- ③ or is a simple way to introduce non-linearity to model

the logistic func is given as

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

we have  $B_0 + B_1 n$

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

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

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

$$\frac{\text{from, } \frac{P(n)}{1 - P(n)}}{\text{from, } \frac{1}{1 + e^{h(\theta)}}}, \frac{\text{divide ①, ②}}{\text{from, } \frac{e^{h(\theta)}}{1 + e^{h(\theta)}}}$$

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

$$B_0 + B_1 n = \log\left(\frac{P(n)}{1 - P(n)}\right) \rightarrow \text{(logarithmic function),}$$

Now we have,

$$\log\left(\frac{p(m)}{1-p(m)}\right) = B_0 + B_1 x \quad \text{(logit function)}$$

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

→ And we get  $p(m)$ , based on the line with  $B_0, B_1$  constants.

This is how logistic regression works

eg, if we have two classes (0,1)

$$y_2 \begin{cases} 0, & \text{if } p(m) \leq 0.5 \\ 1, & \text{if } p(m) \geq 0.5 \end{cases}$$

0.5-threshold

we can change thresholds

→ Based on the scenario

we should change thresholds

cost function / loss function

for a single training instance,

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

if  $p(m)=1 \& y=1$ ,  $\text{cost/loss}=0$

if  $p(m)=0 \& y=0$ ,  $\text{cost/loss}=0$

But,

if  $p(m)=0 \& y=1$ ,  $\text{cost/loss} \rightarrow \infty$

if  $p(m)=1 \& y=0$ ,  $\text{cost/loss} \rightarrow \infty$

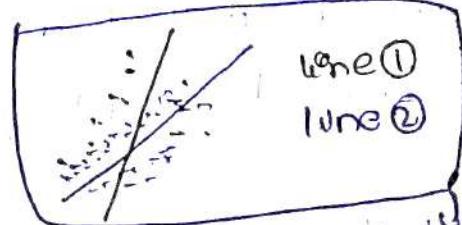
a cost func for whole training set is given as

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

where, m - rows

$y_i$  - expected y

$\hat{p}_i$  - Predicted y

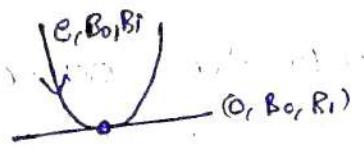


In logit func, RHS we have a line, and we will keep on adjusting the line w.r.t error so that we get correct probability

for from data

→ the values of Parameter ( $\theta$ ) for which the cost func is min is calculated using the gradient descent algorithm (As showed in LR).  
the partial derivative for above cost func is,

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

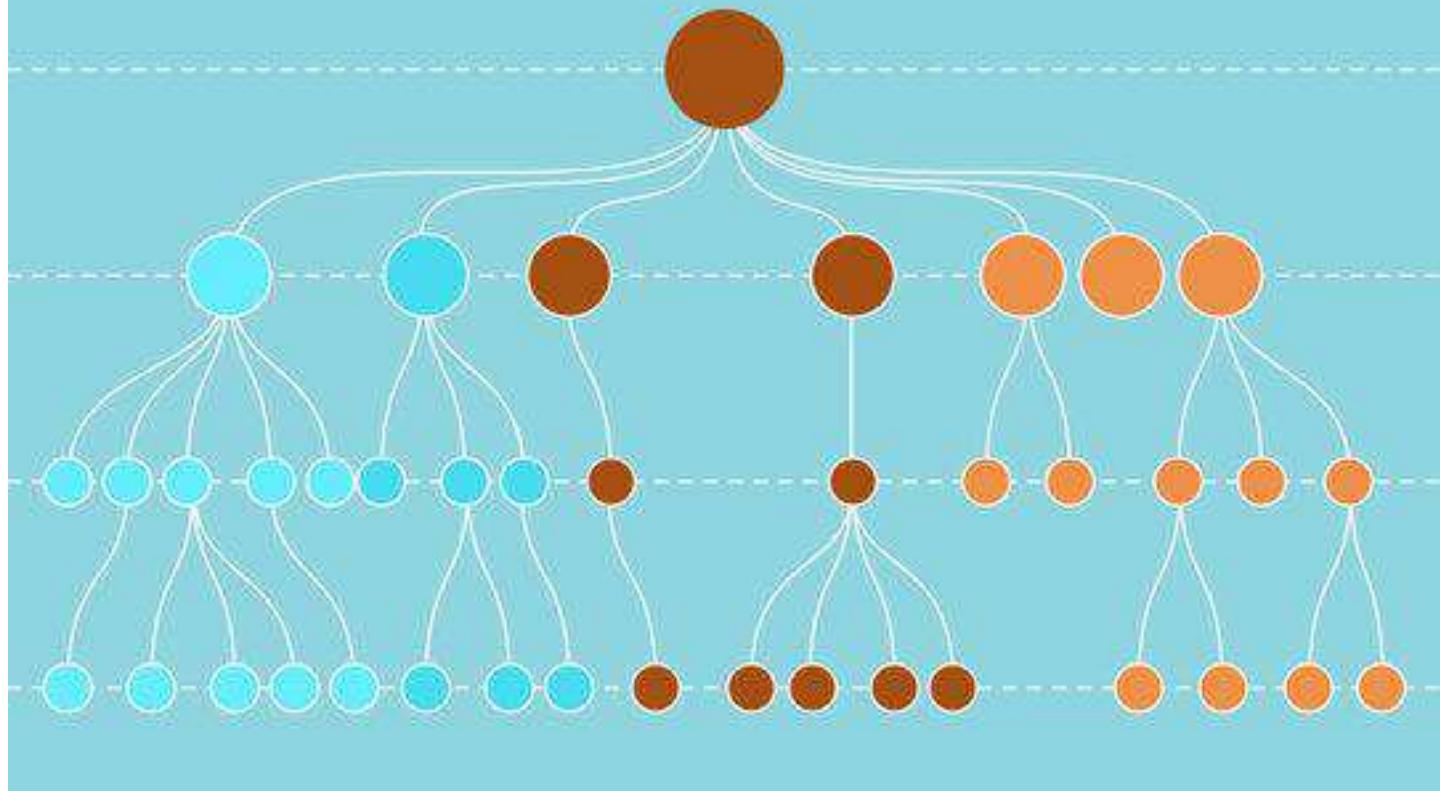


### multinomial Logistic Regression

→ we can extend Logistic Reg 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 value's 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.

# DECISION TREE



## Table Of Contents

1. Why do we need Decision Trees
2. How it works [Download Deep Learning Notes](https://t.me/AIMLDeepThaught/663)
3. How do we select a root node <https://t.me/AIMLDeepThaught/663>
4. Understanding Entropy, Information Gain
5. Solving an Example on Entropy
6. Understanding Gini Impurity
7. Solving an Example on Gini Impurity
8. Decision tree for Regression
9. Why Decision Trees are Greedy Apporach
10. Understanding Pruning

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 model,  
we have this logistic regression  
which can only classify by using line.

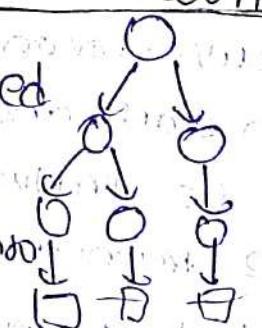
\* what if our data is classified in polynomial like

 } Logistic Regression  
can't be used here

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 give predictions based  
on those conditions. Let's see how it works.

- 1 → First it selects 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 full fill the cond., then it again, splits into new cond.
- 4 → continues till all the cond. are met or if you have pre-defined the depth of your tree



Q) But, how do we select a root node?

Ans: This is where mathematical induction comes in.

→ usually Regression tree are used for quantitative data.

For scale or qualitative/categorical data we use classification trees.

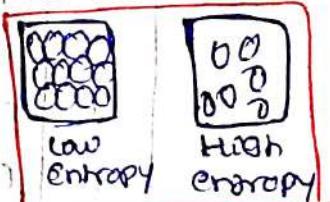
→ In regression tree, we split the nodes based on RSS (Residual sum of squares) criteria.

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 u.

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

P = Probability,

$$E = -P \cdot \log_2 P = (P \cdot \log_2 1/P)$$

\* Information gain (G<sub>n</sub>) 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.

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

$$G(n) = \text{Entropy}(T) - \text{Entropy}(T^n)$$

Q) Let's take three data,

n	x	y	class
①	1	1	I
①	1	0	I
0	0	1	II
①	0	0	II

At we introduced new data,

new data  
n x y class?  
1 0 1 , what is class?

First we need individual column entropy.

(E<sub>n</sub>, E<sub>x</sub>, E<sub>y</sub>)

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

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

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

'1' ① out of 3 is II class.

I

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

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

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

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

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

### Information Gain

↑ No. of 1's  
↑ No. of 0's.

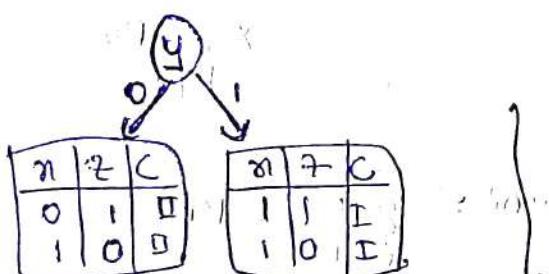
$$G_I = 1 - \left[ \frac{S_0}{S} \cdot E_I(1) + \frac{S_1}{S} \cdot E_I(0) \right] \quad \text{Total no. of evaluation}$$

$$G_I = 1 - \left[ \frac{3}{4} \times 0.276 + \frac{1}{4} \times 0 \right] \approx 0.27$$

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

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

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

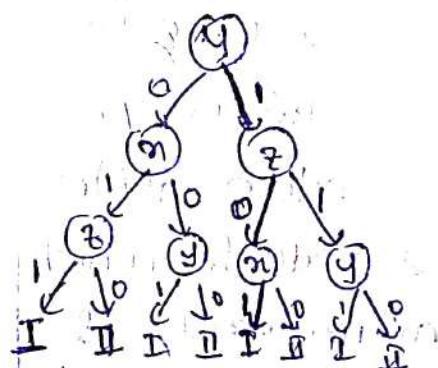


Again we have to find  $E_{I0}, E_{I1}, G_{I0}, G_{I1}$ .

Find largest if parent node.

until leaf node gives a class.

Finally it can be written:



So,  $1 - 0 = 1 \rightarrow \text{I class}$

## 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 observation is classified into the wrong class.

→ Let's suppose there are  $k$  number of classes and an observation belongs to the class  $C_i$ , & its probability is given by  $P_i$ .

then probability that observation belongs to any other class other than  $C_i$  is,

$$\sum_{k \neq i} P_k = 1 - P_i$$

then gini impurity =  $\sum_{i=1}^k P_i \times \sum_{k \neq i} P_k$

gini impurity value

will be 0 & 1

0 - no impurity

1 - Random distn.  
the node for which

gini impurity is least is selected  
as Root node  
to split,

$$Gini = \sum_{i=1}^k P_i (1 - P_i)$$

$$Gini = \sum_{i=1}^k P_i - P_i^2$$

$$Gini = \sum_{i=1}^k P_i - \sum_{i=1}^k P_i^2$$

$$Gini = 1 - \sum_{i=1}^k P_i^2$$

b) cover data

<u>class</u>	<u>Gender</u>	<u>Stay in hotel</u>
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
10	M	Yes
10	M	No

Let's understand how root node is selected by calculating impurity.

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

Step 1

<u>class</u>	<u>stay</u>	<u>Total</u>	<u>P(class)</u>	<u>P(No)</u>
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

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

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

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

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

Weighted sum of gini for class

$$G(class) = \frac{\text{no of class 8}}{\text{Total}} \times G(8) + \frac{\text{no of class 9}}{\text{Total}} \times G(9) + \dots$$

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

$$G(class) = 0.404$$

## ii) Gini for gender

Gender	stain	n	D(G)	P(N)
m	4el>5, No>3	8	5/8	3/8
F	4el>3, No>3	6	4/6	1/2

$$a(m) = 1 - (5/8)^2 - (3/8)^2 = 0.167$$

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

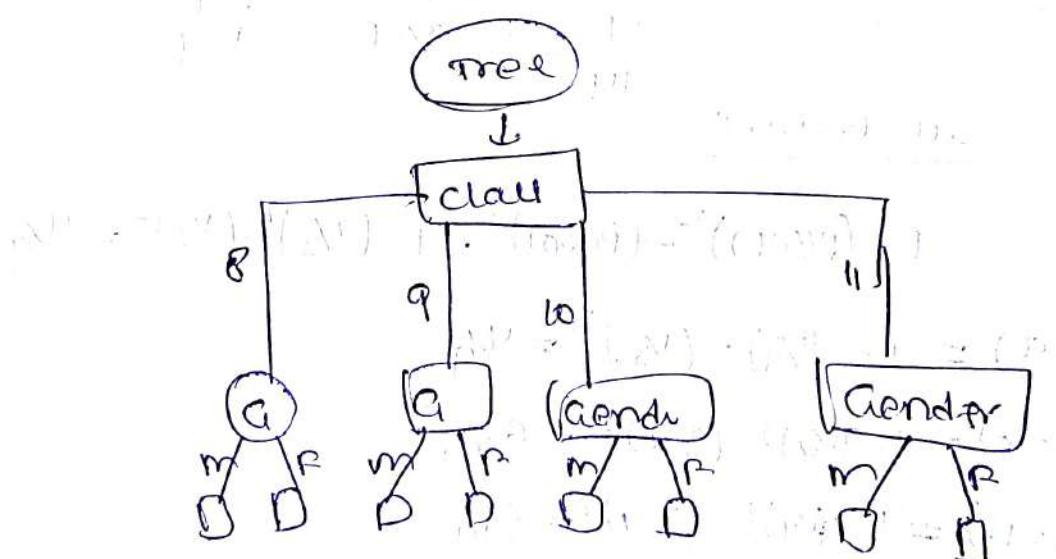
$$a(\text{Gender}) = \frac{8}{14} \times 0.167 + \frac{6}{14} \times 0.5$$

$a(\text{Gender}) = 0.16822$

we can see

$a(\text{class}) < a(\text{Gender})$

thus, root node  
is class



this is how decision tree node is selected by calculating gini impurity for each node individually. If other features, then we need to repeat same process after selecting each node.

## Note F

- ① we should control growth of tree by "min-sample-split" for which we will not, because a infinitely grown tree with depth ( $\approx 100$ ) also has a high chance at some point to overfit. It may go "overfitting".

## Decision Tree for Regression

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

Ex: For a set of possible values,  $x_1, x_2, \dots, x_p$ , we will try to divide them into  $J$  distinct & non-overlapping regions  $R_1, R_2, \dots, R_J$ . 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  $R_j$ .

- ② Regions are selected in a way to reduce residual sum.

$$\sum_{j=1}^J E[(y_i - \hat{y}_{R_j})^2]$$

$y_i$  = mean draw the response var. in the region  $R_j$ !

## 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 Take more time to train a DT model, |
|---|--|---------------------------------------|

## Recursive Binary splitting (Greedy Approach)

- As mentioned, we try to divide the  $x$  values into  $S$  regions, but is very expensive in terms of computational time to try to fit every set of  $x$  values into  $S$  regions.
- Thus decision tree opt for a greedy approach in which nodes are divided into regions based on the given condition is called greedy, but 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 decides threshold to divide others into different regions.

$$\sum_{i \in R_1(S)} (y_i - \hat{y}_{R_1})^2 + \sum_{i \in R_2(S)} (y_i - \hat{y}_{R_2})^2$$

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

**Download Deep Learning Notes**

<https://t.me/AIMLDeepThought/663>

## 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.

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

where,

'T' is the subtree which is a subset of full tree  $T_0$ . ' $\alpha$ ' is non-negative tuning parameter, which penalizes the MSE with an increase in tree len.

## Post-Pruning (Backward pruning)

→ It is the process where DT is generated first & then the non-significant branches are removed. Cross-validation set or data is used to check the effect pruning & test 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 branch from generating, or uses a condition to decide when should it terminate splitting of some of the branch prematurely at tree is generated.

## \* Different algorithms for Decision tree

① ID3 (Iterative Dichotomiser) :-  
It is one of the algorithms used to construct decision trees 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) & Decision tree

It is the most popular algorithm for constructing decision tree. 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.

Ans of Ques to calculate, entropy complicated

which is why CART uses Gini

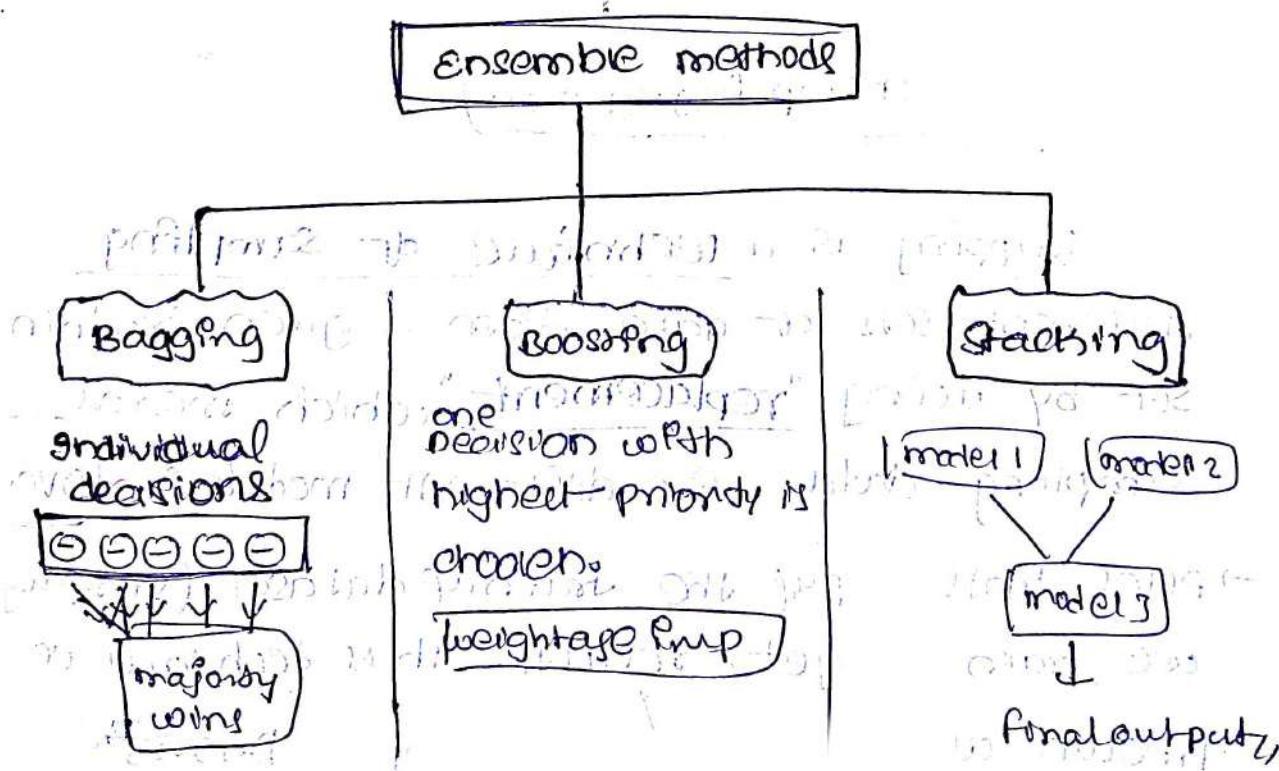
multiple  
decision maker

## \* Ensemble Techniques & Random Forest

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

→ we regularly come across the option of "Audience poll". And constant, mostly win when goel for option which had highest vote from audience. So, we can say, taking opinions from a majority or 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".



④ Let's suppose we have  $n$  predictors (models).

→  $z_1, z_2, \dots, z_n$  with a standard deviation of  $\sigma$

$$\text{var}(z) = \sigma^2 \quad \text{var of single predictor}$$

$$\text{Avg}(u) = \frac{(z_1 + z_2 + \dots + z_n)}{n} \quad \begin{array}{l} \text{Average of Predictors,} \\ (\text{Expected val}) \end{array}$$

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

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

This is why taking mean is preferred over single predictor.

\* So Ensemble methods use 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 parallelly and combine results at the end.

#### Advantages

- ① Bagging significantly decreases variance without bias.
- ② It works as well bootstrap diversely in training data.
- ③ Works well with small 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.

#### Disadvantages

- It improves the accuracy of model on the expense of interpretability.
- We can't interpret all those will be so many models.

**Sampling** → Similar to bagging but no replacement is done. This causes less diversity in the sampled datasets and data ends up being correlated. That's why Bagging is more preferable than sampling.

## 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 models.

The weights are assigned based on performance of individual tree.

→ In 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 -

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

Pros	Cons
1) Robust to outliers	1) Inability to extract a useful combination of features
2) Feature scaling not required	2) High variance leading to small computational power
3) Handles missing values	
4) can deal with irrelevant inputs	
5) computational scalability	

→ Boosting minimizes the variance by taking into consideration the result from small trees.

## \$ General understanding eg. of boost

① you wanna travel to different place.

→ And you know a friend who is a traveller, so you ask him about the place (give 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 ask another friend who live 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. (Bad weightage)

→ Grandpa thinks and says what model should be bought consider grand children also wants car. (Good weightage)

→ consider son & grandpa, mom says what to buy, first 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 models, also called base models and use the combination of the input for a new model

(re)

(meta model)

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

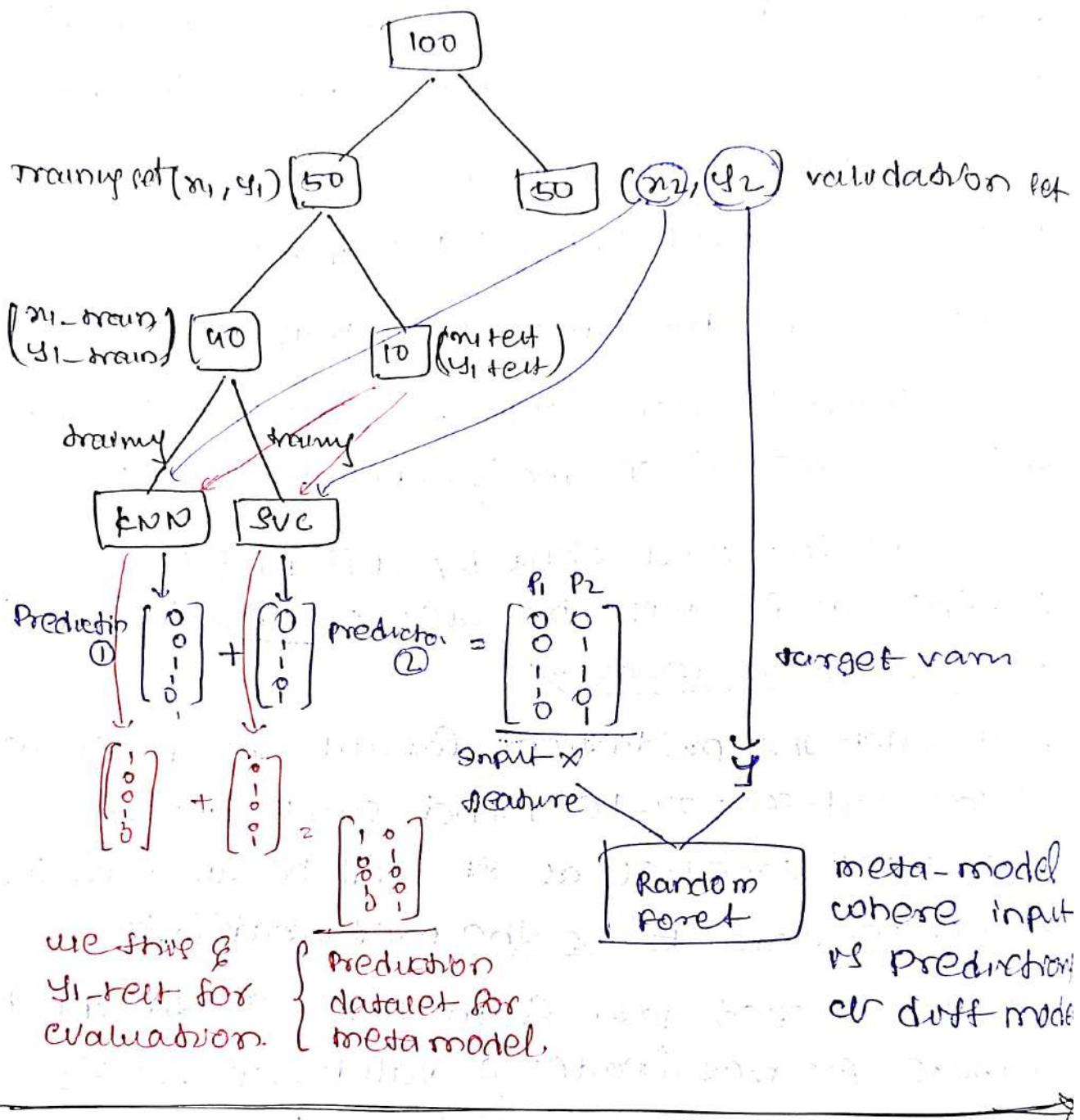
- suppose you have a classification problem & you can use several models like logitReg, svm, KNN, Random Forest etc.  
the idea is to use few models like KNN, SVM as the base model & make prediction using that
- Now predictions made by these models are used as an input feature for Random Forest to train on & give prediction.
- Stacking can be multi-level; using base model at level 1 the pass predictions onto 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 & do 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 =  $x_1, y_1$  ; Holdoutset =  $x_2, y_2$  [validation set]
- ② split the training set again into training & test sets.  
like,  $x_1\text{-train}, y_1\text{-train}, x_1\text{-test}, y_1\text{-test}$
- ③ train all the base models on training set  $x_1\text{-train}$ ,  $y_1\text{-train}$
- 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  $x_1\text{-test}$ , 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  $x_2$   
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 &  $y_1\text{-test}$  for evaluation
- ⑩ end

## visual interpretation



Download Deep Learning Notes

<https://t.me/AIMLDeepThaught/663>

## 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 data, they can be used for evaluating the accuracy of model by using these data for prediction.

## ④ Random Forest

- Decision tree 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 tree.
- 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 to working algorithm of bagging model to decrease the correlation in trees. The idea is to introduce more randomness while creating tree which will help in reducing correlation.

## working of RF model

- ① different samples are collected from training dataset using bootstrapping.
- ② on each sample we train our tree model & we allow the tree to grow depth.
- ③ once the trees are 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.

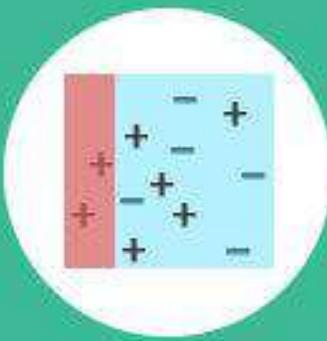
## adv & disadv of Random Forest

- ① it can be used for both regres & classif 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 DTs.
- ⑤ higher computational time than other recursive models.
- ⑥ should be used when accuracy is upmost priority.

Monday, Jan 15, 2018



**Gradient  
boosting**



**AdaBoost**

www.educba.com

## Table of Contents

1. Understanding Boosting
2. Understanding AdaBoost
3. Solving and Example on AdaBoost
4. Understanding Gradient Boosting
5. Solving an Example on Gradient Boosting
6. AdaBoost Vs Gradient Boosting

**Download Deep Learning Notes**

<https://t.me/AIMLDeepThaught/663>

## 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 decisions made.

the weights are assigned based on performance of individual tree

→ in 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 -

→ we mostly used decision tree as weak classifier.  
Any other algorithm can be used as base;

but reasons for choosing tree are -

Pros	Cons
1) Robust to outliers	1) Susceptible to noise
2) Feature scaling not required	2) Inability to extract a useful combination of features
3) Handles missing values	3) High variance leading to small computational power
4) can deal with irrelevant inputs	
5) Computational scalability	

→ Boosting minimize the variance by taking into consideration the result from random tree.

## ⑧ general understanding eg dr boost

① you wanna travel to different place.

→ And you know a friend who is a traveller, so you ask him about the place (give 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 ask another friend who live there.

First, we have to boost other person's worth into we have, and he will boost next person with into he have & finally get result.

## ② dad wants to buy car.

→ He asks you & without any knowledge you say Yes! (Bad weightage)

→ Grandpa thinks and says what model should be bought consider grand children also wants car. (Good weightage)

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

## (6) AdaBoost (Adaptive Boosting)

$$\begin{matrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{matrix}$$

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

① Initially, total weightage should be equal "1".

so. Initially consider observation weight as  $w_i = \frac{1}{N}$

② for  $m = 1$  to  $M$ ,

→ fit a classifier  $G_m(m)$  to the training data with  $w_i$

→ compute  $\text{err}_m = \sum_{i=1}^N w_i \cdot I(y_i \neq G_m(m))$  → Prediction

→ compute  $\alpha_m = \frac{1}{2} \log \frac{(1 - \text{err}_m)}{\text{err}_m}$  → This is the contribution of that tree to final result

→ calculate new weights using the formula,

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

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 of either the one with highest sum (for Regr) or the one with most weightage (for classification),

Eg

- ① understand ~~read~~ the Adaboost by simple dataset for heart patient prediction.

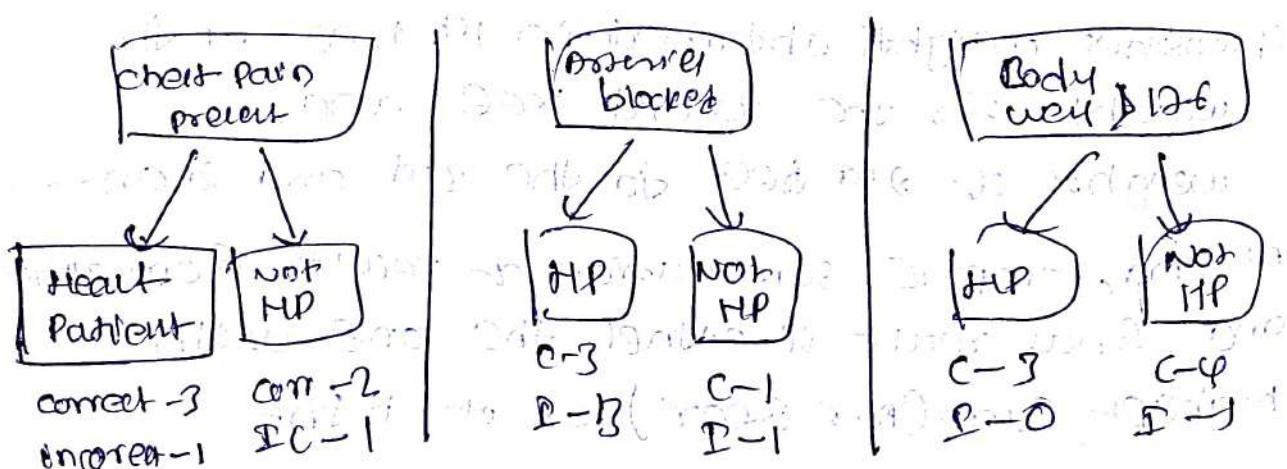
Ex	is chest pain	Arteries blocked	weight or person	is Heart Patient	new weight
0	Yes	Yes	205	Yes	0.05
1	No	Yes	180	Yes	0.05
2	Yes	No	210	Yes	0.05
3	Yes	Yes	162	Yes	0.33
4	No	Yes	156	No	0.05
5	No	Yes	125	No	0.05
6	Yes	No	160	No	0.05
7	Yes	Yes	172	No	0.05

### Step ① - Generalize weights

→ there are 8 rows in our dataset.

$$\therefore w = \frac{1}{N} = \frac{1}{8} \quad (\text{samples are equally important})$$

Step ② - consider columns to create a weak decision model and then try to figure out what all correct & incorrect predictions based on that column



→ we will calculate "Gini Index" of each column.

And we select tree with the lowest GI.

& this will be the first decision maker for our model.

over 100

Step ③ Now, we calculate the contribution of tree to our final decision (column)

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

$$\text{error} = \frac{1}{8}$$

$$\text{contribution} = \frac{1}{8} \log\left(\frac{1-\text{err}}{\text{err}}\right) \approx 0.97$$

→ Step ④ → modify the weights

→ increase the sample weight for

$$\text{incorrectly predicted, new weight} = \text{old weight} \cdot e^{0.97}$$

$$w_{10} = \frac{1}{8} \cdot e^{0.97} = 0.33$$

→ decrease the sample weight for

$$\text{correctly predicted, new weight} = \text{old weight} \cdot e^{-0.97}$$

$$w_7 = \frac{1}{8} \cdot e^{-0.97} = 0.08$$

And add weights & divide by summation to

normalize it.

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

## Step ①

- 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 contribution of all m tree & all n-tree  
are added to

Higher is taken

## ②

### \* 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. over the label column of initially this avg shall minimize the total loss
- ② calculate pseudo residual.

$$\text{pseudo residual} = \text{actual label} - \text{predicted (avg) result}$$

[Download Deep Learning Notes](#)

<https://t.me/AIMLDeepThaught/663>

mathematically,

$$\textcircled{4} \text{ derivative of pseudo residual} = \frac{\delta L(y_i, p_{mi})}{\delta (f_m))}$$

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

\textcircled{3} create a tree to predict the pseudo residuals instead of a tree to predict for actual val. of target variable.

$$\text{new result} = (\text{previous result}) + (\text{learning rate} \times \text{residual})$$

$$f_t(n) = f_0(n) + v \cdot \delta$$

v - learning rate  
δ - residual.

\textcircled{4} Repeat these steps until residual stop decreasing.

EBM 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	1200
900	1400
950	1600

STEP 0 - set initial model

lets set avg rent price in our model:  $f_0(n) \geq 1160$

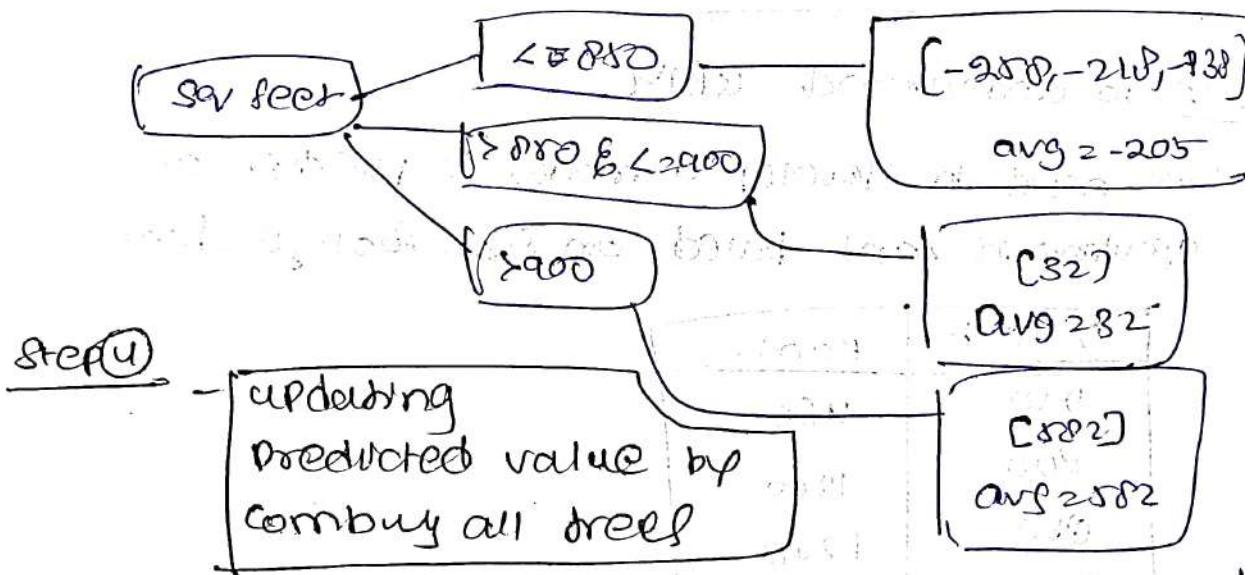
### Step ② - calculate the residuals

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

<u>sq feet</u>	<u>actual rent</u>	<u>Predicted rent (avg)</u>	<u>Residual</u>
750	1160	1018	-288
800	1200	1018	-218
850	1280	1018	-138
900	1400	1018	32
950	2000	1018	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 combining all trees

$$\text{from updated Predicted val} = P_0(n) + (\text{learning rate} \times \text{Residual})$$

<u><math>P_0(n)</math></u>	<u>Residual</u>	<u>Learn rate</u>	<u><math>P_1(n)</math></u>
1018	-205	1	1213
1018	-205	1	1213
1018	-205	1	1213
1018	32	1	1050
1018	582	1	2000

step 6 — 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 prediction.

### GBM vs AdaBoost

→ essentially both AdaBoost & GBM have similar intuition, i.e., to convert a set of weak learners 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 focused more on the difficult ones and therefore, contributed 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 minimized the overall error of the model.

## ⑧ XG Boost (eXtreme Gradient Boosting)

- XG Boost regularizes data better than normal gradient boosted tree.
- XG Boost's objective function is the sum of loss function evaluated over all the predictions & regularization func for all predictions (of tree).

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

- loss function depends on the task being performed and a regularization term is demanded,

$$L(f) = YT + \frac{\lambda}{2} \sum_{j=1}^J w_j^2$$

watcher over score.

For controlling overall model complexity by not creating leaves.

- unlike other tree-building algorithms, XG Boost doesn't use entropy or Gini index. Instead, it utilizes gradient/error term and a "heuristic" for measuring the tree.

- Heuristic for a Regr. Problem is residual and for classification problem, Heuristic is a second order derivative of the loss at the current estimate.

$$h(m) = \frac{\partial^2 L(y, f(m))}{\partial f(m)^2}$$

$m = f^{(m-1)}(m)$

## Tree Buildup in xG Boost

- ① Initialize the tree with one leaf.
- ② compute the "similarity" using,

$$\text{similarity} = \frac{\text{gradient}^2}{\text{heuristic} + \lambda}$$

where,  $\lambda$  - Regularization term.

- ③ Now, for splitting data into a tree form, calculate,

$$\text{gain} = (\text{left similarity}) + (\text{right similarity}) - (\text{similarity for Root})$$

- ④ For tree pruning, the parameter  $\gamma$  is used. The algorithm starts from the lowest level of the tree & then starts pruning based on the value of  $\gamma$ .

If  $(\text{Gain} - \gamma) \leq 0$ , then remove that branch.  
Else, keep the branch.

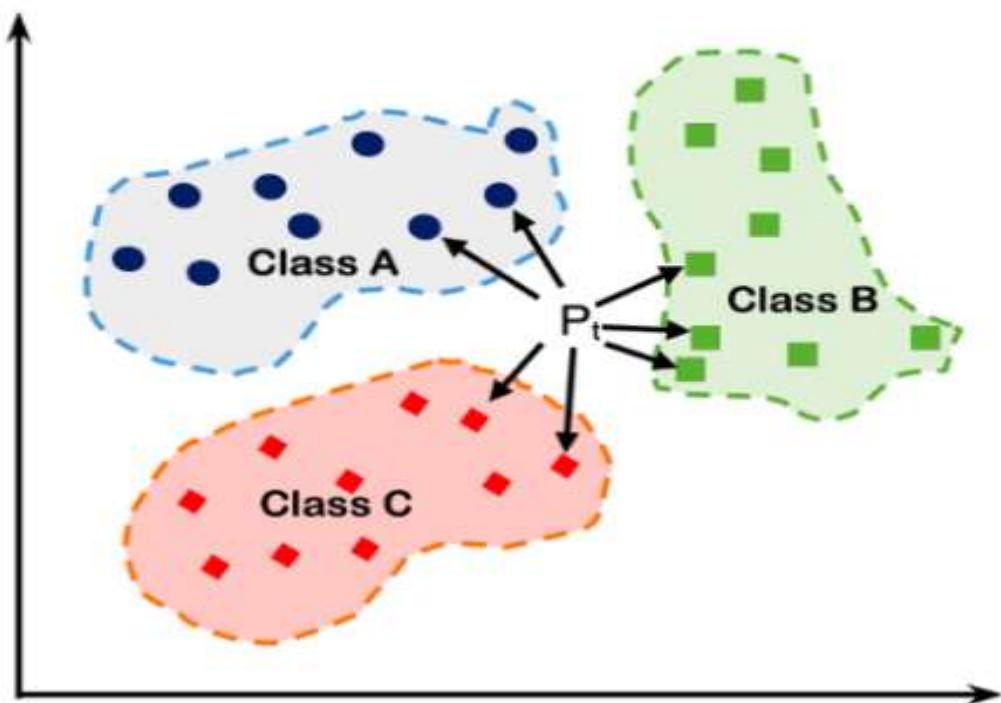
- ⑤ learning or done using new eq.

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

**Download Deep Learning Notes**

<https://t.me/AIMLDeepTaught/663>

# K Nearest Neighbors



## Table Of Contents

1. How does K-Nearest Neighbours work
2. How is Distance Calculated
  - Eculidean Distance
  - Hamming Distance
  - Manhattan Distance
3. Why is KNN a Lazy Learner
4. Effects of Choosing the value of K
5. Different ways to perform KNN
6. Understanding KD-Tree
7. Solving an Example of KD Tree
8. Understanding Ball Tree

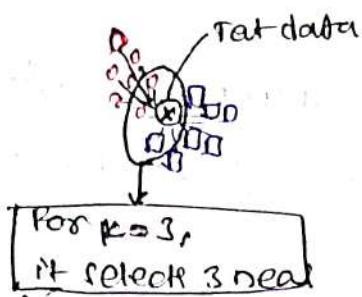
[Download Deep Learning Notes](#)

<https://t.me/AIMLDeepThaught/663>

## (a) 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 of test data by calculating the distance between the test data and ALL THE TRAINING POINTS!! 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 KNN 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 there are red, 1 blue. Hence, it is Red.

→ If it is Regr prob, the predicted value will be 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 mismatched b/w two vectors. It is most widely used on categorical data.

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

e.g.)  $(A, B, C, D) - (A, D, C, B) = (0, 1, 0, 1) = 2$ ,

## \* Manhattan distance

- also known as L<sub>1</sub> norm, taxicab form, rectilinear dist, city block distance.

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

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

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

## \* Lazy learners

- KNN algorithm is often termed as lazy learner.
- And most of the algorithms like Bayesian classifier, logistic regression & SVM etc. are called lazy learner. These algorithms generalize over the training set before receiving the test data.

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

- But this is not the case with KNN algorithm, it does not 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, waits for test set.

→ So, a lazy learner just stores training data &

## \* Weighted nearest neighbour

- As name suggest, we assign weight to the K-nearest neighbour. The weights are typically assigned on the basis of distance.
- Sometimes rest of data are assigned '0' also.
- The main intuition is that the point in neighbor should have more weight 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 low, then model has high variance & low bias.  
As 'K' value is increased, then variance decreased & bias increased.
- 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 classified the data by calculating the distance of test data from each of the observations and selecting K-value, this approach is known as "Brute Force KNN" [which is computationally very expensive (time taking)]

→ so, the idea behind using other algorithm 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 clusters".

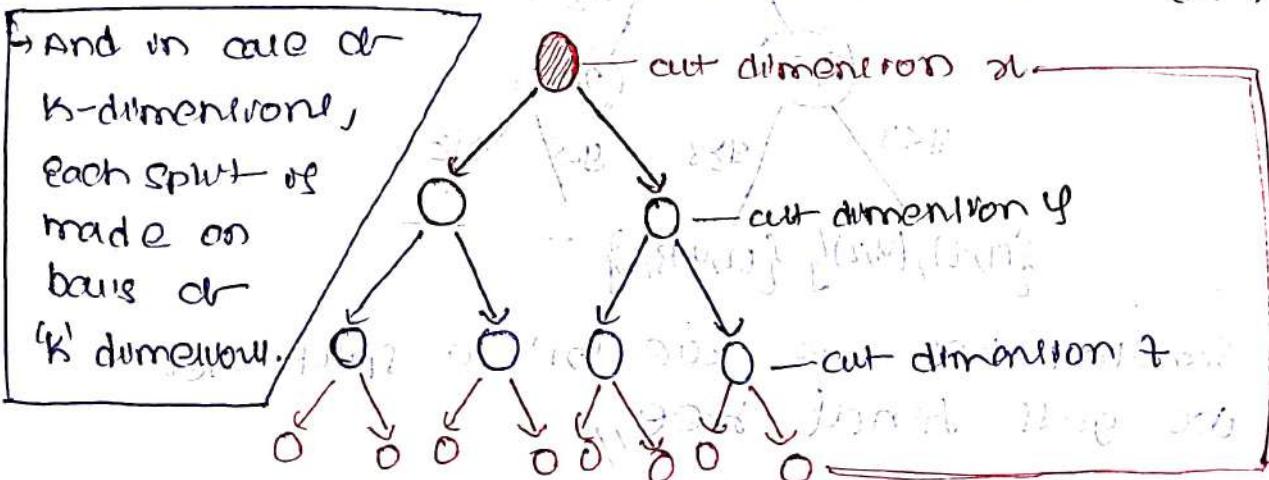
## \* K-dimensional tree (Kd Tree)

→ 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.

Ex

let's say we have 3 dimensional data (2, 4, 8).



Eg)

Q1 training data  $\Rightarrow \{ (1,2), (2,3), (2,4), (3,6), (4,2), (5,7), (6,8), (2,5), (8,5), (9,1), (9,3) \}$

H

Here,  $k=2$

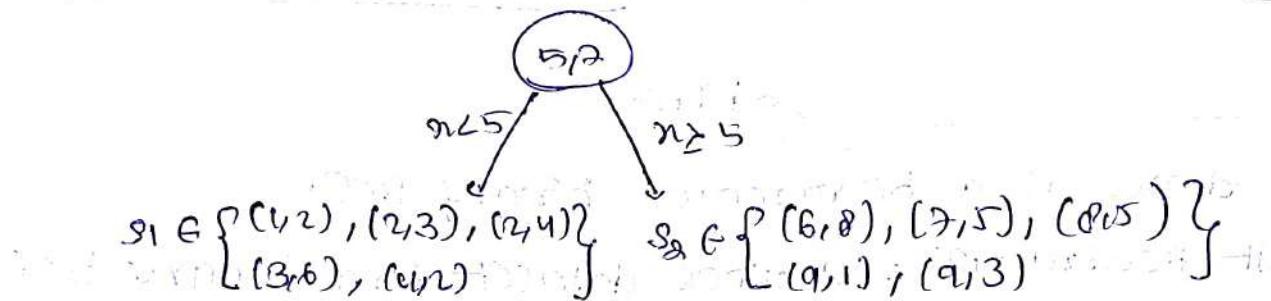
lets build our ad-tree

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

ie,  $\{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 9 \}$

median.

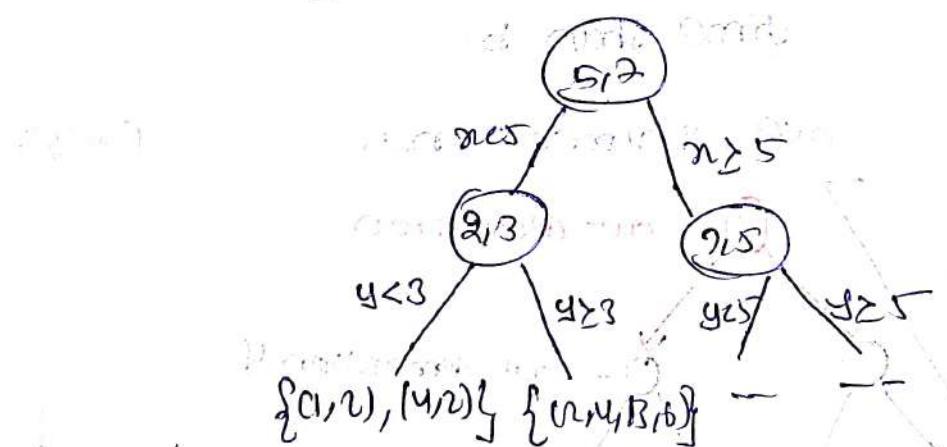
our Root node will be  $(5,7)$ ;  $x \geq 5$  (SPLIT cond)



(ii) splitting of right side nodes into further

split  $S_1, S_2$  on condition of  $y$ .

ie,  $y_1 \in \{ 2, 4, 6 \}$  &  $y_2 \in \{ 1, 3, 5, 7, 8 \}$



Similarly, here we can go to split and we get final tree.

## Note (kd-tree)

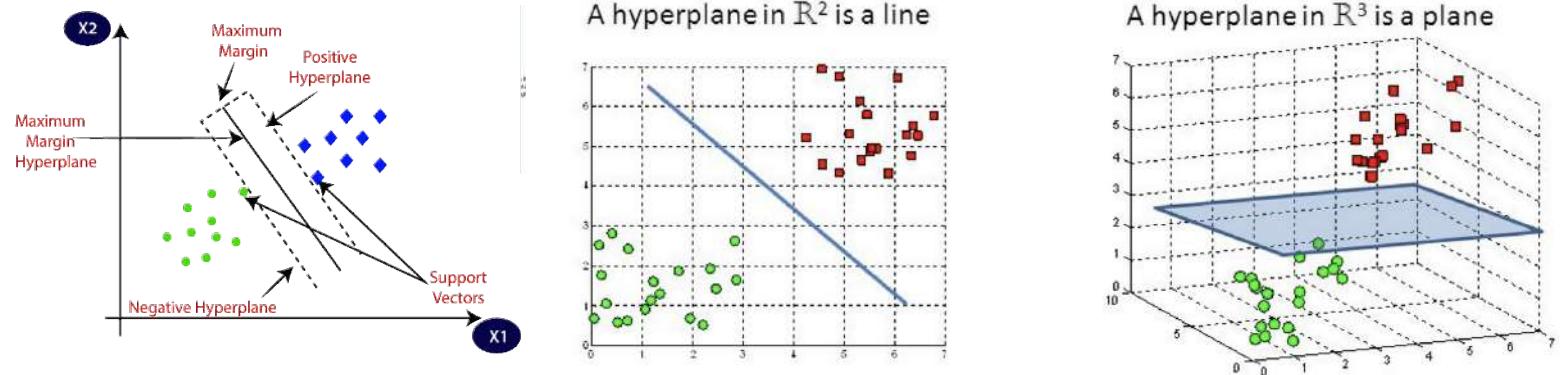
- once the tree is formed, it is easy for algorithm to search for the probable nearest neighbor just by traversing tree.
- the main problem of kd tree is that it gives probable nearest neighbors but can miss out actual nearest neighbors.

## ii) Ball-tree

- similar to kd tree, ball tree are also hierarchical data structures. they 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 cluster.
- one point cannot be in both clusters.
- distance of point is calculated from centroid of each cluster. the point closer to centroid 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.
- thus, this is how clusters are kept divided till a certain depth.



## Understanding Support Vector Machines

### Table Of Contents

1. Understanding Concept of SVC
2. What are Support Vectors
3. What is Margin
4. Hard Margin and Soft Margin
5. Kernelized SVC
6. Types of Kernels
7. Understanding SVR

[Download Deep Learning Notes](#)

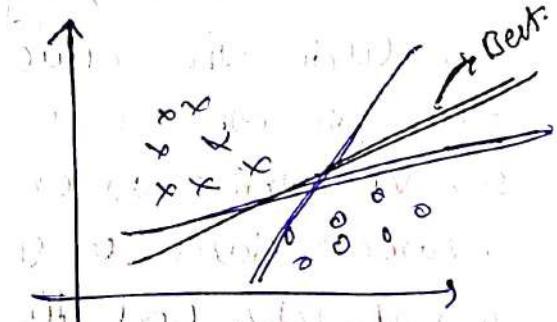
<https://t.me/AIMLDeepThaught/663>

## \* 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 side of the line, and given a new point based on the probability, we can say which part or plane it belongs. Now, what about SVM.

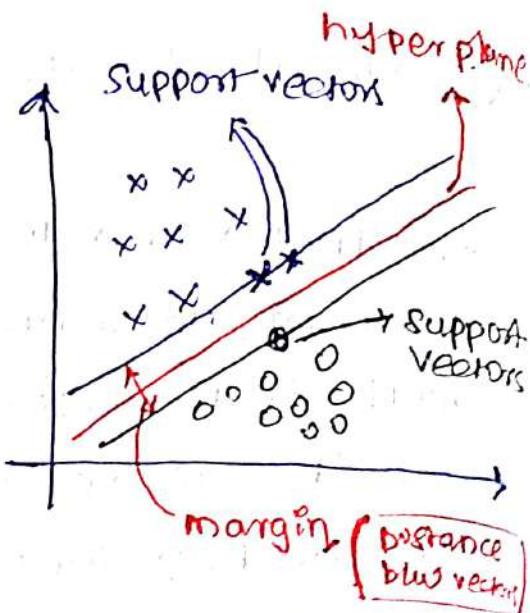
### Concept of SVM

- Similar to logistic regression, we get a plane/hyperplane in SVM, but how do we decide on the best hyperplane. There can be several planes passing through the data points.
- So, the key principle behind SVM is to find the hyperplane / decision boundary, that maximally separates data points belonging to distinct classes, that is, maximizing 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.

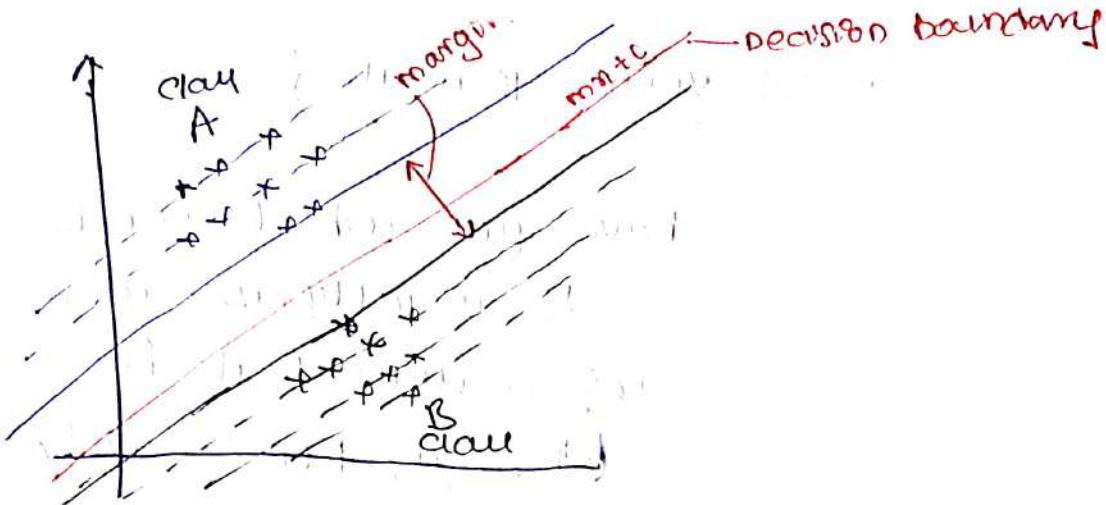


→ the points that are close to decision boundary are called "support vectors". And these support vectors are crucial for defining hyperplane so these support vectors help us find optimal plane.

## ② what is Margin

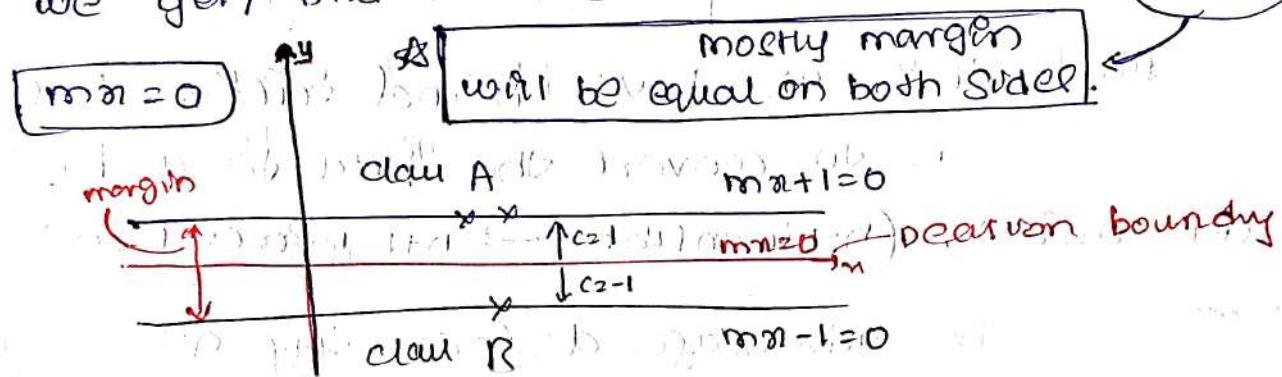
→ If we consider logistic Regression, based on the probability we will decide the class for a new point, but there are chances for things to go wrong here, say probability is 0.9 & so we conclude the class, but it can be wrong!

so, SVM tries to create maximum separation between data to get rid off the confusion or probability. And the maximum separation 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  $c=1$ .



+ Now, if we want to predict a new datapoint, then we first checks the y-value of  $f(x)$ . 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 probabilities. Just by the range of  $y$ , we are predicting the new datapoint.

## ① 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. OR
- it works well when the data is linearly separable & free from noise.

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

→ signifies all data points are correctly classified.

### Soft Margin

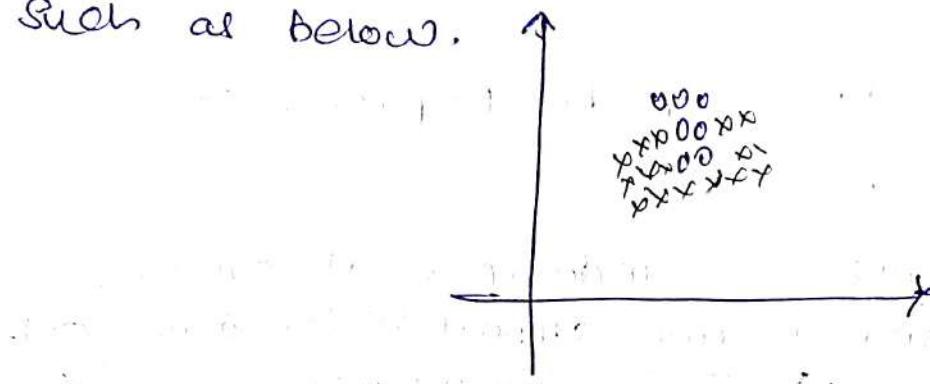
- SM allows for some margin violations & misclassifications, thus providing flexibility when dealing with noisy / overlapping data.
- OR introduced a slack variable ( $\epsilon_i$ ) to handle margin violations, allowing some points to fall within the margin / on wrong side of boundary.

$$y_i^*(w^T n_i + b) \geq 1 - \epsilon_i$$

→ where  $\epsilon_i \geq 0$  for all points,

## ⑥ Kernelized support vector machines

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



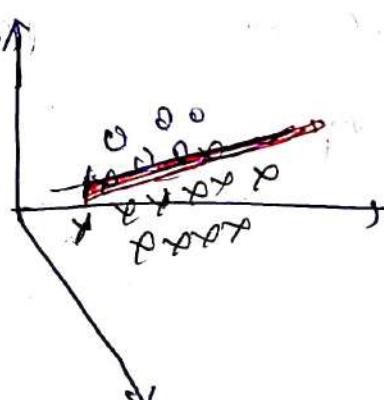
→ To deal this, we have kernel tricks. Basically, the idea is to convert the data to a higher dimension, ( $n$ -dimension  $\rightarrow$   $n+1$  dimension).

→ 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).

(The data is linearly separable in the next slide)

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



① How do we increase the dimension?

→ say we have a 2D dataset as  $(n, y)$ .

Now if we want to add another dimension, we can simply use some mathematical function to find  $z$ .

Eg  $z = n + y$ ,  $z = n - y$ ,  $z = n^2 + y^2$ ,  $z = n^2 - y^2$  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.

### ② Types of Kernels In sum

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

1) Linear kernel → compute dot product between two feature vectors.

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

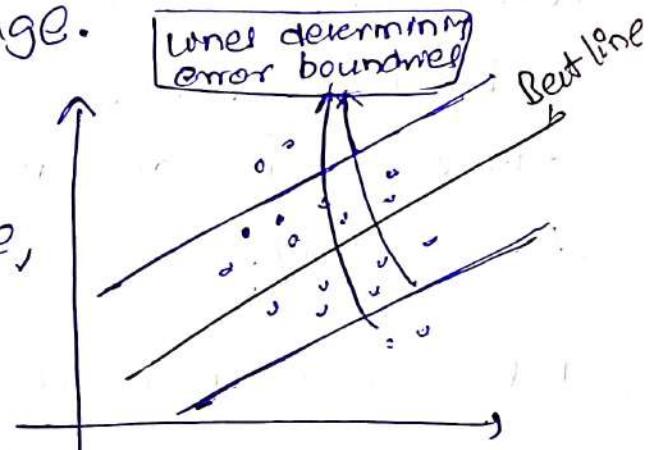
2) Polynomial kernel →  $K(n, y) = (n^T y + c)^d$

3) Radial Basis Function (RBF) kernel →  
RBF kernel measures the similarity between two samples using a Gaussian radial function.

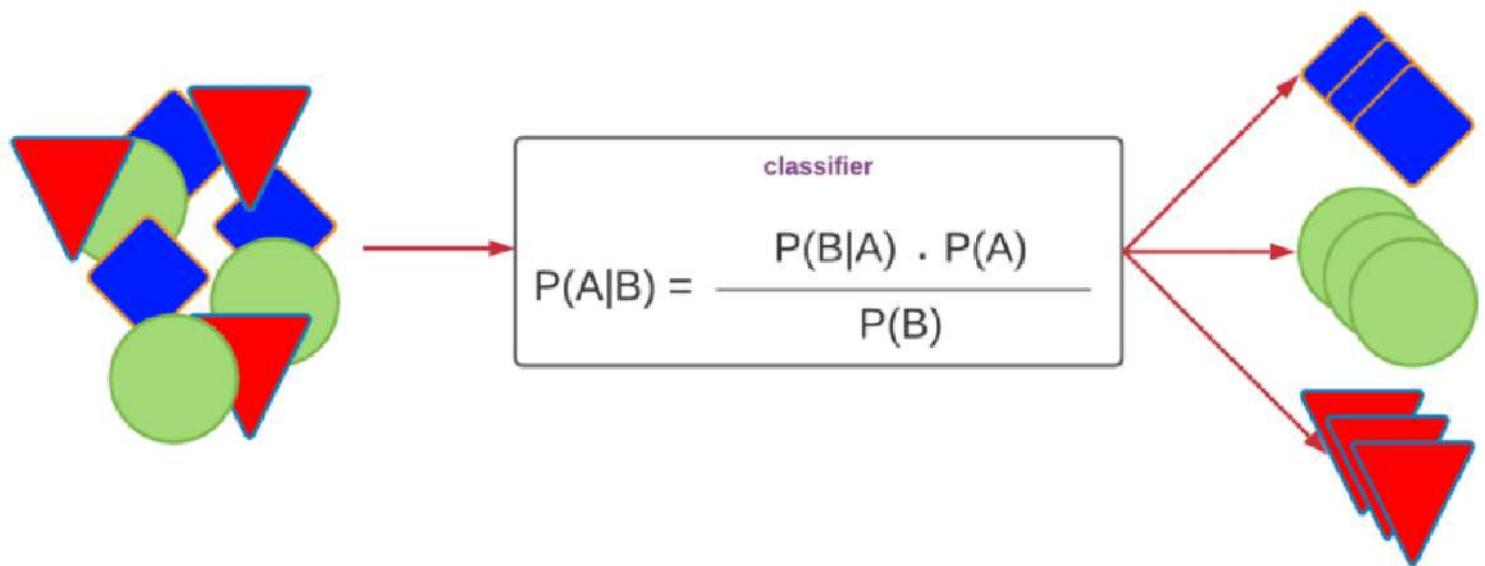
$$K(n, y) = \exp(-\gamma \|n - y\|^2)$$

## ⑥ 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 bounding 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

1. Why do we need Naive Bayes
2. Concept of how it works
3. Mathematical Intuition of Naive Bayes
4. Solving an Example on Naive Bayes
5. Other Bayes Classifiers
  - Gaussian Naive Bayes Classifier
  - Multinomial Naive Bayes Classifier
  - Bernoulli Naive Bayes Classifier

[Download Deep Learning Notes](#)

<https://t.me/AIMLDeepThaught/663>

## ① Naive Bayes classifier

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

Ex:-

- 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.

## ② This is where Naive Bayes helps!

→ Naive Bayes will not try to generalize 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.)

## ③ 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 occurring given the probability of another event that has already occurred.

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).

→  $P(A|B)$  is the posterior probability, that is the probability of event after evidence is seen.

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

	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 person 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 independent relationship of each feature with output label by leveraging Naive Bayes.

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

★ ① The fundamental 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 all features contribute equally to outcome. we have, Bayes' theorem as

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

y - all variable  
x - dependent feature.

→ And by taking naive assumption do bayes theorem, which is independence among the features, we get.

$$P(A, B) = P(A) \cdot P(B)$$

$$P(y | n_1, n_2, \dots, n_n) = \frac{P(n_1, n_2, \dots, n_n | y) \cdot P(y)}{P(n_1, n_2, \dots, n_n)}$$

Based on naive assumption, or independence,

$$= \frac{[P(n_1 | y) \cdot P(n_2 | y) \cdots P(n_n | y)] \cdot P(y)}{P(n_1) \cdot P(n_2) \cdots P(n_n)}$$

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

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.

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

so, to solve this problem we need to calculate  $P(y) \& P(n_i | y)$   $\rightarrow$  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.  $[P(n_i | y_j)]$

we get,

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

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

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

wind Feature			
	Y	N	P(Y)
false	6	2	6/9
true	3	3	3/9
Total	9	5	100%

play		$P(Y yes) / P(N no)$
yes	9	9/14
No	5	5/14
Total	14	100%

→ Here we calculated  $P(n_i^j | y_j)$  for each  $n^j$  &  $y_j$ , and also  $P(y_j)$  for each class variable  $y_j$ .

### Prediction

→ now, if we have a new datapoint as

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

$$P(Y | \text{today climate}) = \frac{P(\text{Sunny} | \text{Yes}) \cdot P(\text{Hot} | \text{Yes}) \cdot P(\text{Normal} | \text{Yes}) \cdot P(\text{False} | \text{Yes}) \cdot P(\text{Yes})}{P(\text{today})}$$

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

$$P(N \mid \text{today}) \propto \frac{2}{5} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{2}{5} \cdot \frac{5}{11} \approx 0.1142$$

However, we know,

$$P(Y \mid \text{today}) + P(N \mid \text{today}) = 1$$

APPLY reciprocal,

$$1 = \frac{1}{P(Y \mid \text{today}) + P(N \mid \text{today})} \quad \text{multiply } P(Y \mid \text{today})$$

$$P(Y \mid \text{today}) = \frac{P(Y \mid \text{today})}{P(Y \mid \text{today}) + P(N \mid \text{today})}$$

$$P(Y \mid \text{today}) = \frac{0.024}{0.0211 + 0.1142} = \frac{0.024}{0.1353} \approx 0.155$$

$$P(N \mid \text{today}) = \frac{0.1142}{0.0211 + 0.1142} = \frac{0.1142}{0.1353} \approx 0.844$$

So, clearly

$$\boxed{P(N \mid \text{today}) > P(Y \mid \text{today})}$$

Hence, "play golf  $\rightarrow$  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  $P(x \mid y)$

## ① Gaussian Naïve Bayes classifier

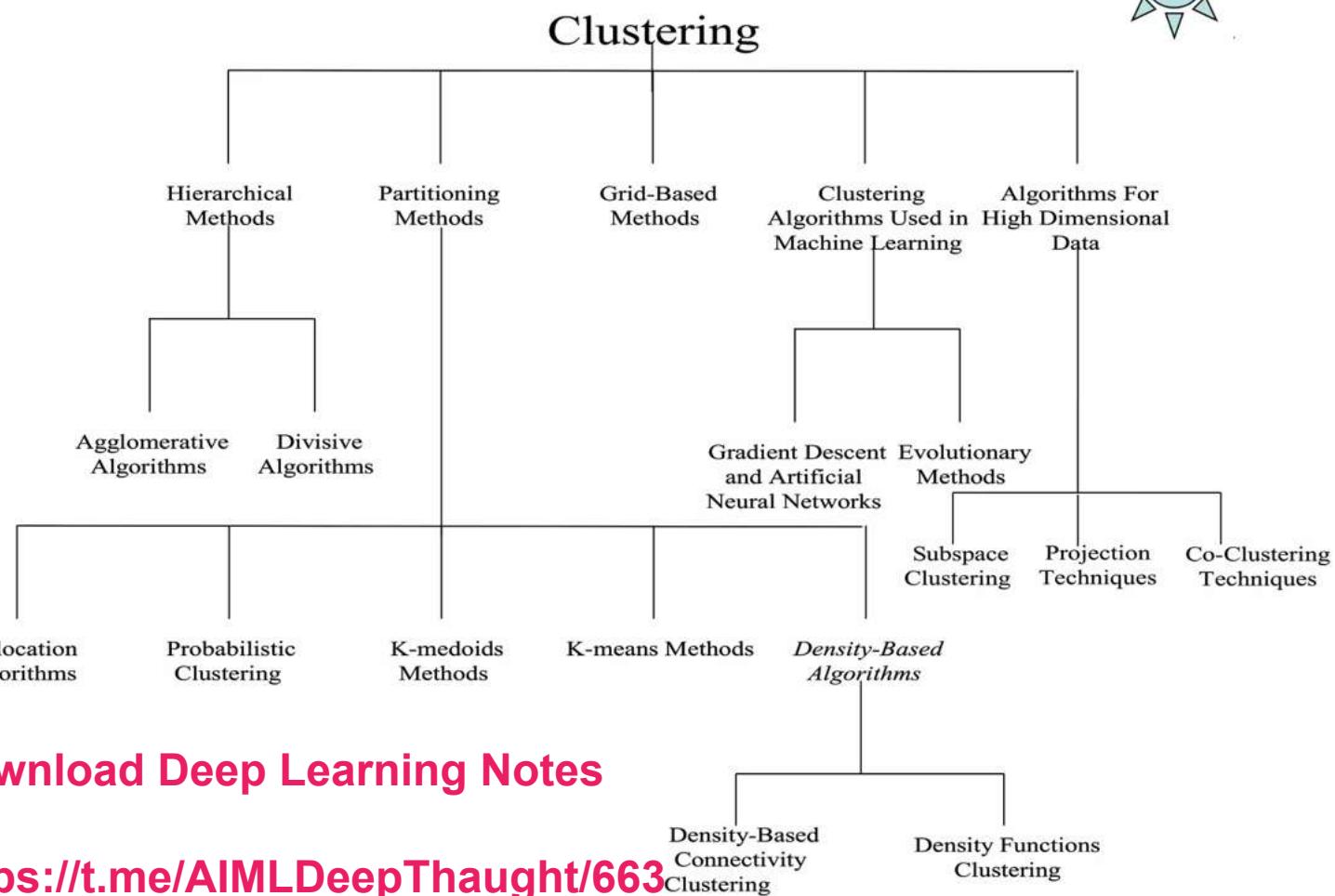
- If the feature variables are continuous variables, then we cannot use the Naïve Bayes, and instead we should be using Gaussian Naïve Bayes Classifier.
- In Gaussian Naïve Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution, also called "Normal distribution".
- The likelihood of the features is assumed to be Gaussian, hence conditional probability is,

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

Other popular Naïve Bayes classifier are -

- ① multinomial Naïve Bayes: feature vector represent the frequencies with which certain events have been generated by multinomial distribution. This is typically used for document classification.
- ② Bernoulli Naïve 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 features.

# Types of Clustering Algorithms



## Download Deep Learning Notes

<https://t.me/AIMLDeepThaught/663>

## Table of Contents

1. How clustering is different from classification
2. Applications of Clustering
3. What are density based methods
4. What are Hierarchical based methods
5. What are partitioning methods
6. What are Grid Based methods
7. Main Requirements for Clustering Algorithms

## Unsupervised Learning Algorithms. (pg 100 & 5 References)

- i) k-Means
- ii) Hierarchical clustering
- iii) DBSCAN
- iv) Performance measurement
- v) Principal component Analysis
- vi) Dimensionality Reduction.

} clustering -

Evaluation metrics for Clustering are Important



### 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 <u>similarity</u> <u>without</u> the help of <u>class labels</u> .
3) complexity	more complex	less complex
4) Example Algo	Logistic Regression, naive Bayes, SVM etc	K-means clustering, Hierarchical clustering, DBSCAN, BIRCH Algorithm, etc

## Application of clustering

### → for customer segmentation

You can cluster your customer based on their purchase, their activity on website and so on. And this can be used in recommendation system to suggest content that other user in same cluster enjoyed (Chirag)

Eg:- marketing, insurance, librairie etc.

### → Earthquake prediction

By learning old data it can make cluster and determine dangerous zone.

### → for search engines

As you search, similar image 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 system.

### → for Anomaly detection (outlier)

→ As a dimensionality Reduction Technique.

→ For Data Analysis

## clustering methods

### ① density-Based methods

these methods consider the clusters as dense region having some similarity and different from the lower dense region or Spurious.

These methods have good accuracy & ability to merge two clusters.

ESL

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

ii) OPTICS [Ordering Points to Identify Clustering Structure.]

### ② Hierarchical-Based methods

The clusters 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 -

form a single cluster & expand

ii) divisive

Top down approach -

form a big cluster & divide

iii) CURE

[Clustering Using Representatives]

iv) BIRCH

[Balanced Iterative Reducing Clustering and using Hierarchical]

### 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]

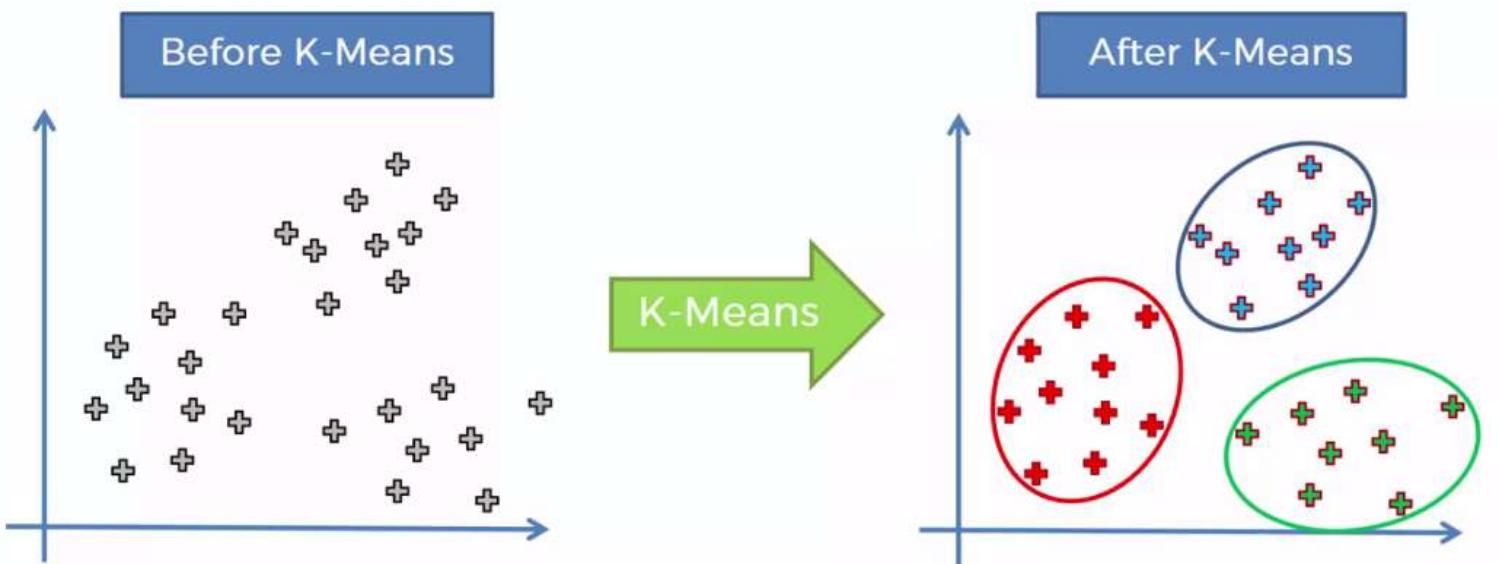
iii) Wave, cluster, etc - all grid methods

**Download Deep Learning Notes**

<https://t.me/AIMLDeepThought/663>

## 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 clusters should not vary with the order of input records.
- ⑥ It should be able to handle data of high dimension.
- ⑦ It should be easy to interpret & use.



**Download Deep Learning Notes**

<https://t.me/AIMLDeepThaught/663>

## Table Of Contents

1. Concept of K-Means Clustering
2. Math Intuition Behind K-Means
3. Cluster Building Process
4. Edge Case Scenarios of K-Means
5. Challenges and Improvements in K-Means

## 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 clusters should not vary with the order of input records.
- ⑥ It should be able to handle data of high dimension.
- ⑦ It should be easy to interpret & use.

### 10 K-means

old algorithm

Proposed in 1957 by Stuart Lloyd, also

In 1965 by Edward W. Fuzzy. 10

Sometimes called

Lloyd-Fuzzy 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 assigns the points to exactly one cluster.

## Theory of K-means

→ Let  $C_1, C_2, \dots, C_k$  be the  $k$ -clusters.

→ Then we can write

$$C_1 \cup C_2 \cup C_3 \cup \dots \cup C_k = \{1, 2, 3, \dots, n\} \rightarrow$$

$$\text{and also, } C_k \cap C_{k'} = \emptyset \rightarrow$$

i.e.,  
Each data point  
has been assigned  
to a cluster

This means, clusters are non-overlapping.

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

The within-cluster variance is denoted by  $\underline{w(C_k)}$

→ Hence, according to statement above,

we need to minimize the variance for all clusters mathematically,

$$\text{minimize}_{C_1, C_2, \dots, C_k} \left\{ \sum_{k=1}^K w(C_k) \right\}$$

WCSS →  
within cluster  
summation or variance

→ The next step is to define the criterion for measuring the within cluster variance.

Generally, the criterion is the Euclidean distance between two data points.

$$w(C_k) = \frac{1}{|C_k|} \sum_{i \in C_k} \sum_{j \in C_k} (x_{ij} - \bar{x}_{kj})^2$$

$$P = (1) w$$

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

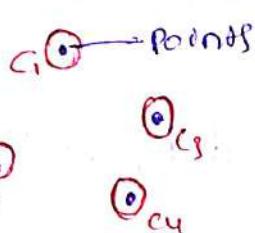
Eg:-

Let's take 4 points

case ① for

4 points

4 clusters



→ now, each point acts as a cluster and so if it is centroid or not.

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

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

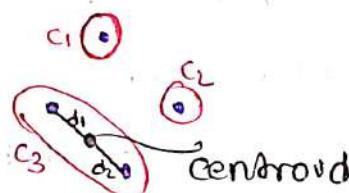
$$\text{e.g., } w(c_1) = 0 + 0 + 0 + 0 = 0 \\ c_1 \ c_2 \ c_3 \ c_4$$

(Summation of distances.)

case ② for

4 points

3 clusters



→ Now we need distance b/w Point & the centroid of cluster if present in.

$$w(c_3) = c_1 + c_2 + c_3$$

$$w(c_3) = 0 + 0 + (d_1 + d_2)$$

Individual

sum of distance in a cluster

$$\sum_{i=1}^3 (x - c)^2$$

Centroid

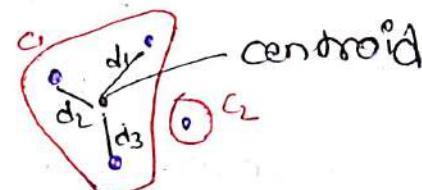
③ All the clusters sum

$$\sum_{i=1}^3 \sum_{j=1}^3 (x_j - c_i)^2$$

case ③ for

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 distance in a cluster.

Summation of the individual summations of distances in cluster.

case ④ for 4 points

1 cluster



$$w(c_1) = c_1$$

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

Goal is to minimize

$$w(c_1) + w(c_2) + w(c_3) + w(c_4)$$

↳ low

Variance  
for all  
clusters,

So, we got

$$w(c_4) = 0$$

$$w(c_3) = d_1 + d_2$$

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

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

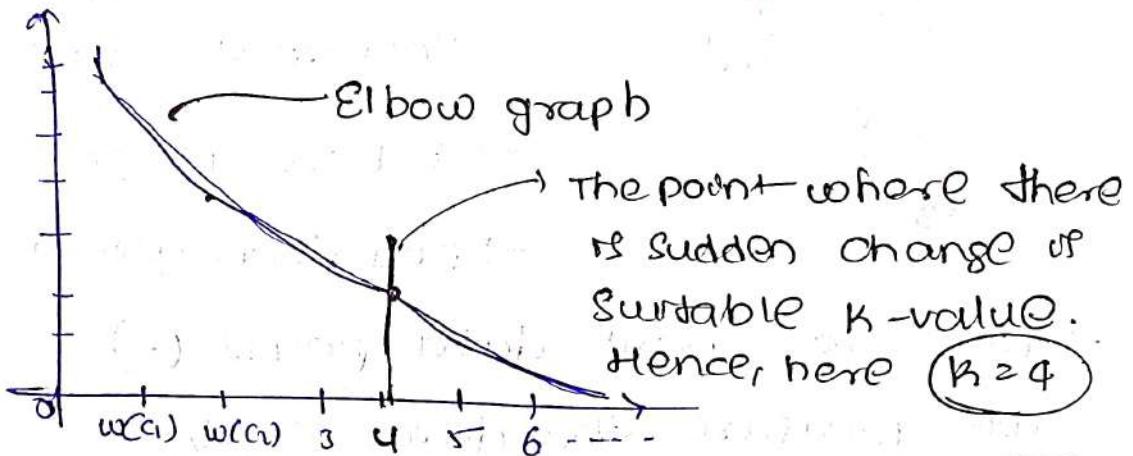
→ By observation, we  
can say that

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

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

$$w(c_1) \succ w(c_2) \succ w(c_3) \succ \dots \succ w(c_n)$$

and the graph goes as -



within cluster summation or square (WCSS)

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

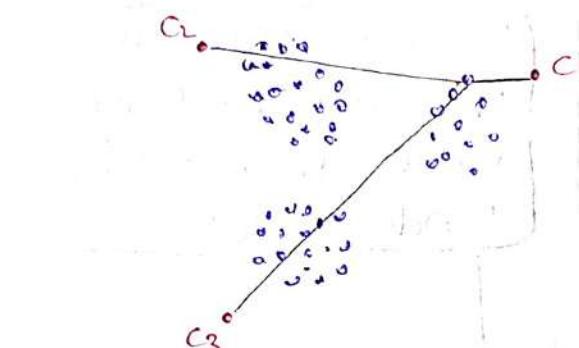
But, here is the problem in k-means,

\* After selecting k value, it takes  
k-points as centroids and starts  
building the clusters normally.

But the question is where do we  
place those k-points?

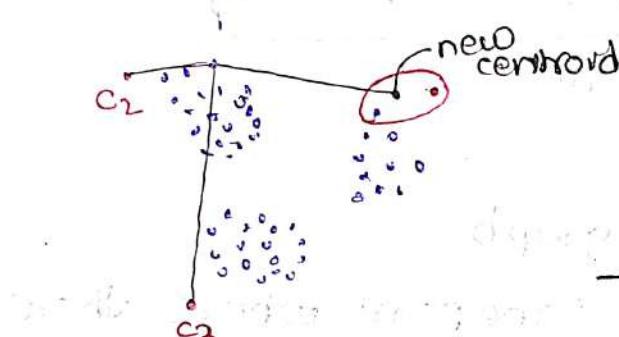
## \* cluster building process

let's say we got:  $K=3$



→ after, randomly placing the  $K$  points,

→ get taken a point from our data ( $\cdot$ ) and measured the distance to all the  $K$ -centroid point ( $\bullet$ ), and whichever distance is smaller, the point goes that cluster.



→ so, here it goes to  $c_1$ .

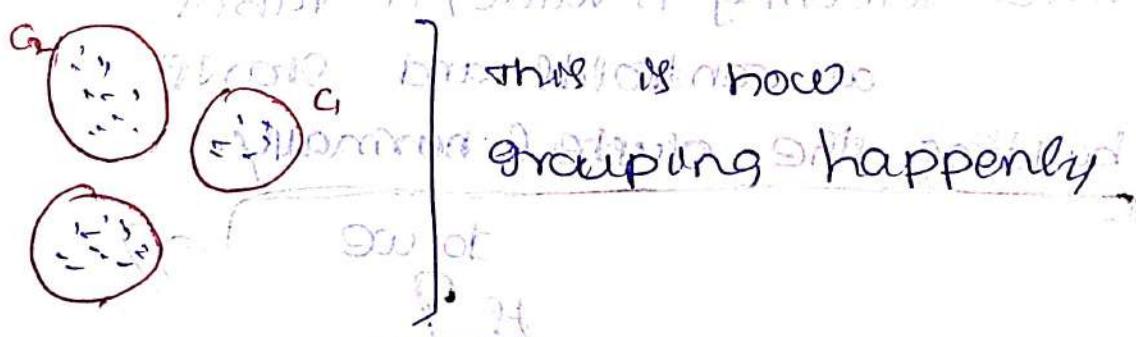
→ and then gets a new centroid.

→ now for next data point ( $\cdot$ ) 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.

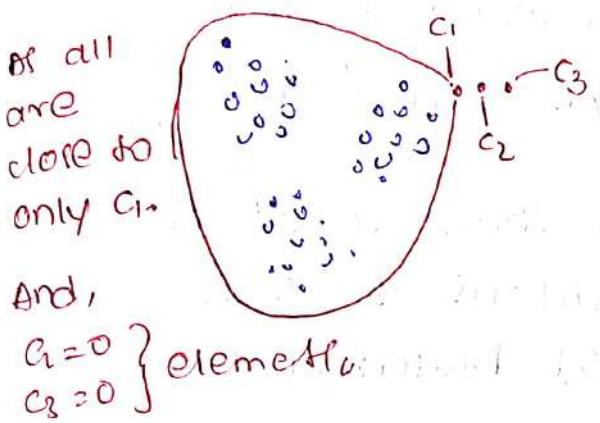


## edge case scenario

### 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?



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 ( $\max K=n$ )
- 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".
- ⑤ The model's inertia is the mean squared distance b/w each instance & closest centroid.
- ⑥ Goal is to have model with low inertia.
- ⑦ Once all points assigned to cluster recompute centroid.

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 highly biased clusters.

### \* Challenges 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 proposed in 2003 by Charles Elkan. It uses Euclidean distance calculations which is achieved by employing the TRIANGLE INEQUALITY.

On a triangle with sides  $a, b, c$  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.

**Download Deep Learning Notes**

<https://t.me/AIMLDeepThought/663>

## II Hierarchical Clustering

Bottom-up / Agglomerative approach

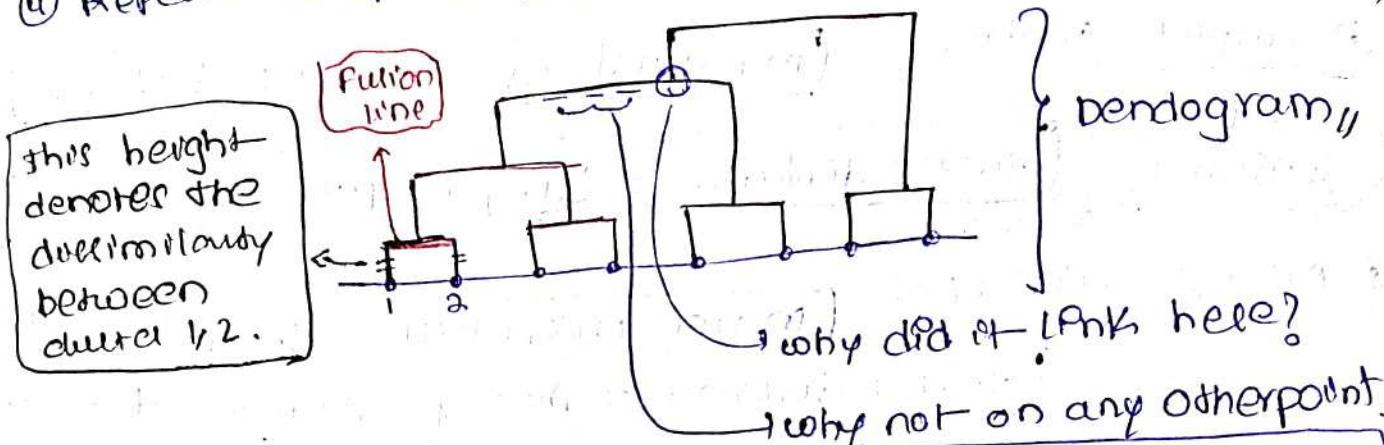
- Initially, all points are separate clusters.
- one main disadvantage of K-means is that we need us to pre-enter the number of clusters ( $k$ ).
  - Hierarchical clustering is an alternative approach which does not need us to give the value of  $k$  beforehand and also, it creates a beautiful tree-based structure for visualization.
  - we start by defining any sort of similarity between the datapoints. Generally, we consider Euclidean distance. The points which are closer to each other are more similar than the points which are farther away.

### Algorithm

- ① Begin with  $n$  observations and a measure (such as Euclidean 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 indicate the height in the dendrogram at which "fusing line" should be placed.

③ compute the new pairwise inter-cluster dissimilarity among the remaining cluster.

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 dendrogram to form flat clusters.

④ we know, arbitrary HC starts with clusters consisting of individual points.

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

→ since clusters are pair of points,

there are many different kinds of linkage methods.

↳ most common are Dissimilarity

## Notelet

### ① Single Linkage

cluster distance = smallest pairwise distance

### ② complete Linkage

minimal intercluster distance

less sensitive  
to outliers

cluster distance = largest pairwise distance

### ③ Average Linkage

mean intercluster dissimilarity

cluster distance = average pairwise distance

### ④ centroid linkage

can result in undesirable envelopes

cluster distance = distance between the centroids of clusters

### ⑤ ward's Linkage

[Before & after merging]

cluster distance = minimize the variance in cluster.

### simple linkage

minimal intercluster dissimilarity

→ single linkage can result in extended,

drawn-out clusters 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 sensitivity to outliers when forming flat clusters.

→ works well for low-noise data with unusual structures

## ⑩ 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 :-

① Epsilon :- (EPS)  
This is the distance till which we look for the neighbouring points.

② min-points :-  
The min no. of points specified by the user.

③ core point :-  
If the no. of points inside the epsilon radius of a point is greater than or equal to the min-points then it is called a core point.

④ Border point :-  
It is the no. of points inside the epsilon radius of a point is less than the min points and it lies within the epsilon radius region of a core point, then it is called border point.

⑤ Noise :-

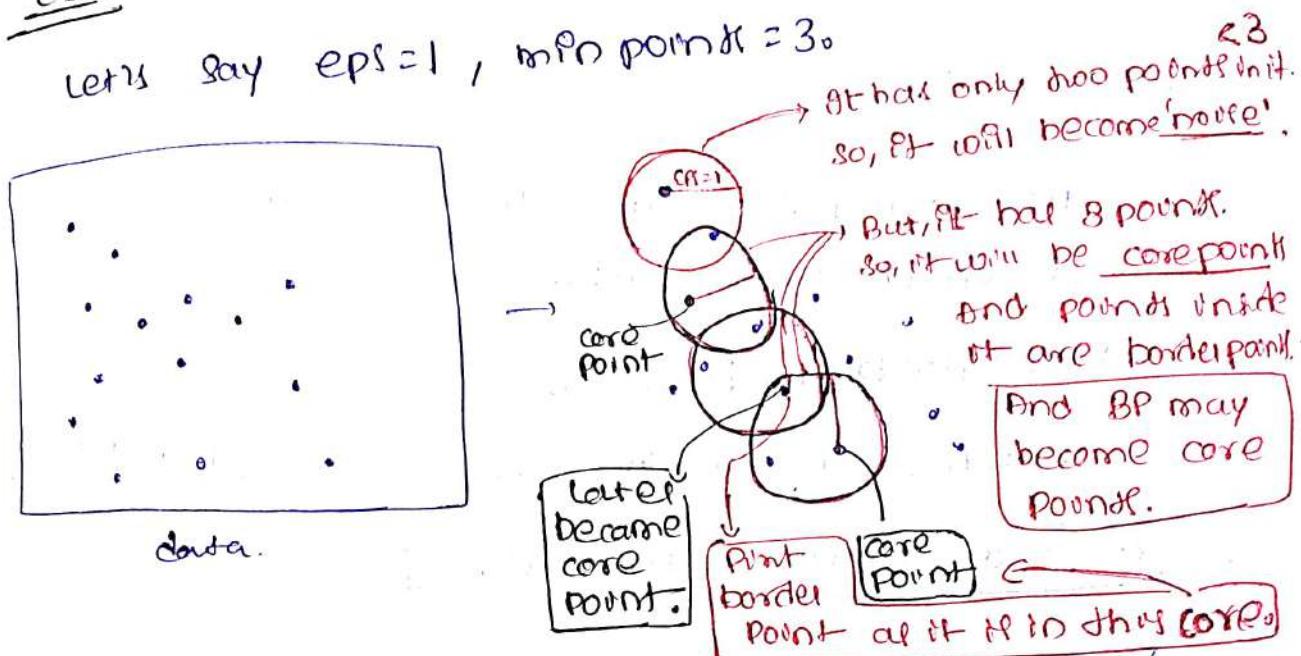
A point which is neither core nor a border point is a noise point.

## 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  $\epsilon$  value.
- ② If the point contains  $\geq$  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 or it's in the  $\epsilon$  radius of a core point.
- ③ If the point is not 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 noise.

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

Ex 2



→ 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 just two hyperparameters. (In practice, it needs to be supplied  $\text{eps}$  &  $\text{min samples}$ )

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

→ //

Download Deep Learning Notes

<https://t.me/AIMLDeepThought/663>

## \* Evaluating or clustering

### Cluster validity

The validation of clusters created us a troublesome task.

→ the problem here is

cluster are in the eyes of the beholder.

→ good cluster will have

### Problems in tool

- High inter class similarity

- low inter class similarity.

→ high similarity in different, dissimilar clusters

### Approach 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 to evaluate the quality of the clusters

**Download Deep Learning Notes**

<https://t.me/AIMLDeepThaught/663>

## ① External measures

→ no. of objects in the data  $P^o$  or  $P_1, P_2, \dots, P_m$   
the set of ground truth clusters :-  $C_1, C_2, \dots, C_n$   
the set of cluster formed by the algorithm.  
the incidence matrix  $N \times N$  matrix.

→  $P_{ij} = 1$ , if the two points  $O_i^o$  &  $O_j^o$  belong to  
the same cluster in the ground truth  
else,  $P_{ij} = 0$ .

→  $C_{ij} = 1$ , if the two points  $O_i^o$  &  $O_j^o$  belong to  
the same cluster in the ground truth  
else,  $C_{ij} = 0$

now there can be following scenarios

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

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

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

4)  $C_{ij} = 0$  but  $P_{ij} = 1 \rightarrow$  the points don't belong to  
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	$n_{11}/P_{11}$	$n_{12}/P_{12}$	$n_{13}/P_{13}$
cluster 2	$n_{21}/P_{21}$	$n_{22}/P_{22}$	$n_{23}/P_{23}$
cluster 3	$n_{31}/P_{31}$	$n_{32}/P_{32}$	$n_{33}/P_{33}$
	$c_1$	$c_2$	$c_3$
			$n$

→ Entropy of cluster $i$ ,

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

→ For entering clustering algorithm, entropy can be generalized

$$C = \sum m_i e_i^o$$

→ purity or cluster  $i$ ,  $P_i^o = \max(P_{ij}^o)$

and for entire cluster it is  $P(c) = \frac{1}{n} \cdot \sum P_i^o$

→ Purity, is the overall 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 measure, most of time we don't have ground truths.

## ① Internal measures

There are the methods we use to measure the quality of clusters 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.

$$WCSE = \sum_{i=1}^k \sum_{j=1}^{n_i} (x_j - \mu_i)^2$$

## Separation

How different objects in different clusters are and how different 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 or all data points.

## Note

$BSS + WSS$  is always a constant

→ the silhouette can be calculated as

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

where,

$a(m)$  is avg. distance of  $x$  from all other points in same cluster.

$b(m)$  is avg. distance of  $x$  from all other points in other cluster.

And silhouette coefficient is,

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

**Higher SC** means that the inter cluster similarity is less and the intra cluster similarity is more. (Good clustering)

inter cluster similarity is less and the intra cluster similarity is more.

## \* 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 coordinates of one point from another.

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

→ And for ND, it becomes  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \dots + (z_1 - z_n)^2}$

→ 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 would 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.

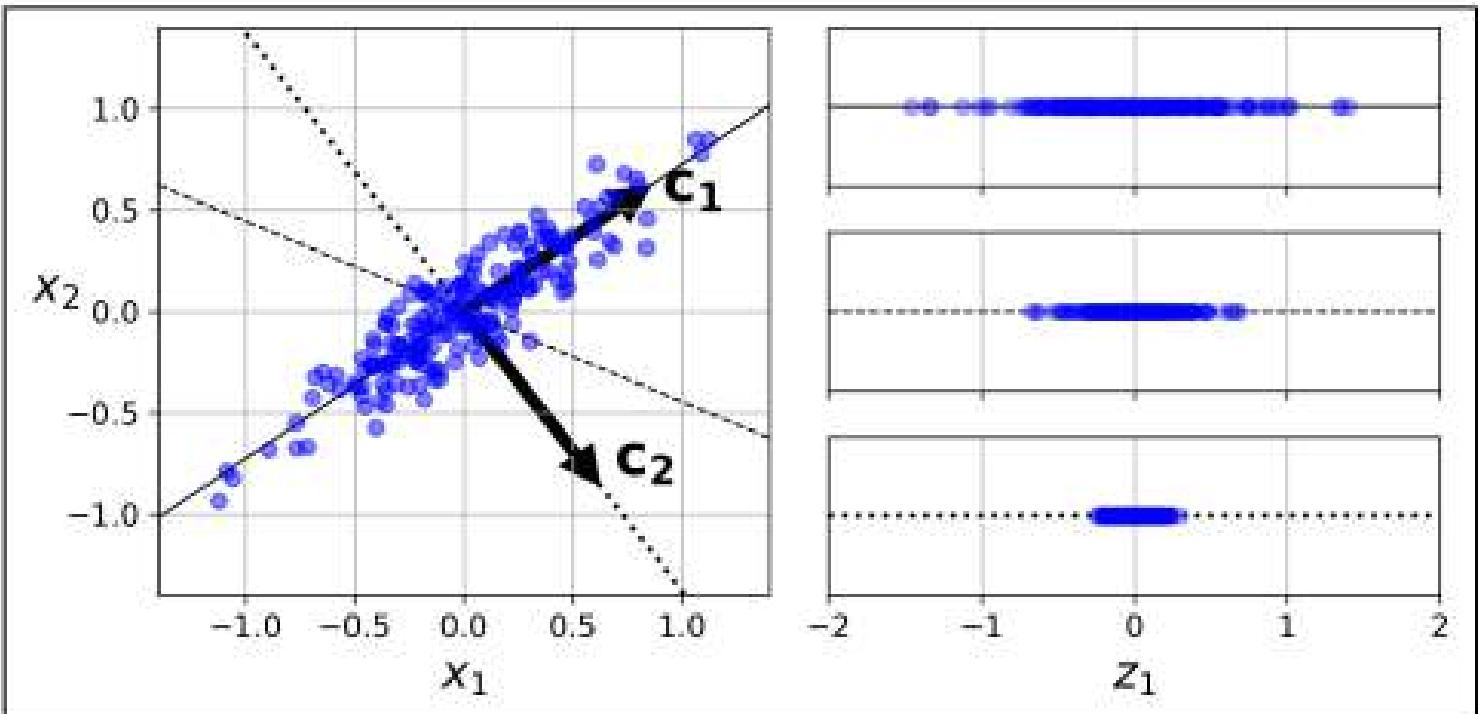
The answer to all the above questions is:

## 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 shall be used before feeding the data to a machine learning algorithm, as it reduces the space in which the distances are calculated, thereby improving ML algorithm performance.



## Understanding Principal Component Analysis

[Download Deep Learning Notes](#)

### Table Of Contents

1. Idea Behind PCA
  2. What are Principal Components
  3. Eigen Decomposition Approach
  4. Singular Value Decomposition Approach
  5. Why do we maximize Variance
  6. What is Explained Variance Ratio
  7. How to select optimal no.of Prinicpal Components
  8. Understanding Scree plot
  9. Issues with PCA
  10. Understanding Kernel PCA
- <https://t.me/AIMLDeepThaught/663>

## \* 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 coordinate system or 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 second a bit less and so on..
- Each new axis / principal component found using PCA is a linear combination of the original features
- All the principal components 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,

Let's derive both 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,

$$A = \lambda V$$

where,  $V$  is the eigen vector and  $\lambda$  is the corresponding eigen value.

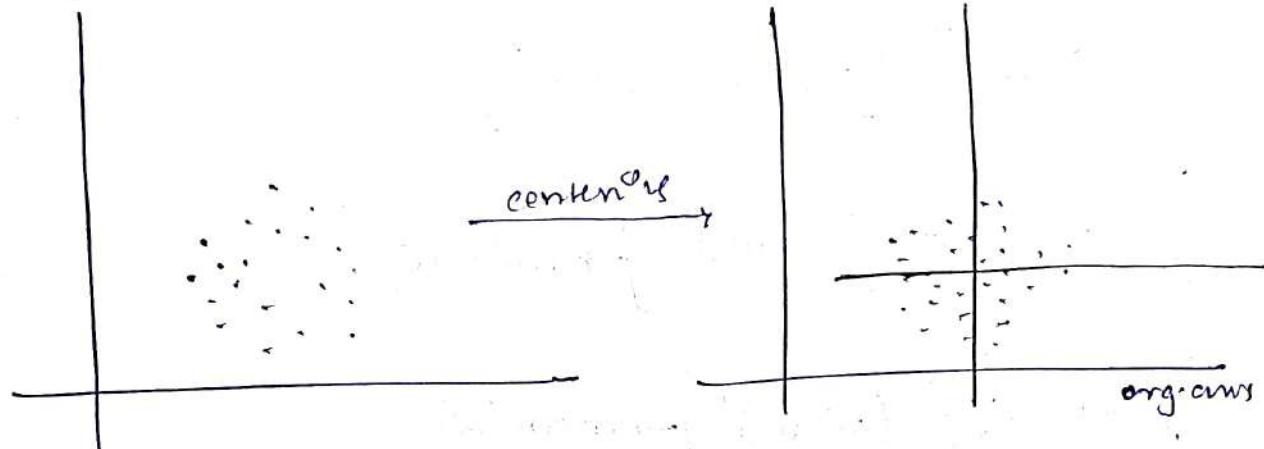
### Steps for Eigen-decomposition approach

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

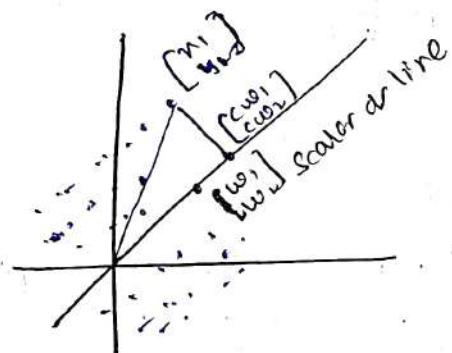
- ② compute co-variance matrix from the normalized data,  $\Sigma = A^T A / (m-1)$
- ③ Perform eigen decomposition of  $\Sigma$  (covariance matrix), to obtain eigenvalues & eigenvectors.
- ④ choose the top  $K$  eigen vectors comput associated with the largest eigenvalues to construct the principal components as  $A X_K$ ,  $K$ th eigen vector.

Proof

Step 1



we have,  $D = \{n_1, n_2, \dots, n_m\} \quad n_i \in \mathbb{R}^d$



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

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

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

$$F(\omega) = \frac{1}{n} \sum_{i=1}^n \| n - (n^T \omega) \cdot \omega \|^2$$

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

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

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

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

$$F(\omega) = \frac{1}{n} \sum_{i=1}^n [n^T n_i - (n^T \omega)^2] \rightarrow \min \omega$$

$$F(\omega) = \frac{1}{n} \sum_{i=1}^n (n^T \omega)^2 \rightarrow \max \omega$$

$$F(\omega) = \frac{1}{n} \sum_{i=1}^n (n^T \omega)(n^T n_i) = \frac{1}{n} \sum_{i=1}^n \omega^T (n_i \cdot n_i^T) \omega$$

$$F(\omega) = \omega^T \left( \frac{1}{n} \sum_{i=1}^n n_i \cdot n_i^T \right) \omega$$

we know  
this is covariance matrix

Each column corresponds  
to one data point.

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

\* where, C - covariance matrix  
 $\omega$  - eigen vector  
 corresponding to  
 max eigen value  
 of C,

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

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

→ 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

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

Residuals after d rounds

$$\forall i \rightarrow n_i - [(n_i^T w_1) w_1 + (n_i^T w_2) w_2 + \dots + (n_i^T w_d) w_d] = r_i$$

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

→ if original dataset is of  $(d \times n)$ , then based on above  
 so, 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 dimension =  $50 \times 100 = 1500$  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

+  $\{w_1, w_2, w_3, w_4, w_5\} \rightarrow$  Representation,

Common for  
dataset

+  $n^o \rightarrow [n_i^T w_1 \ n_i^T w_2 \ n_i^T w_3 \ n_i^T w_4 \ n_i^T w_5] \rightarrow$

Data  
Point  
Specific

+ Hence, to reconstruct  $n^o$  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 eigenvectors. we can reconstruct by

A<sup>T</sup>XK<sup>T</sup>

## ② singular value decomposition method

→ SVD method factorizes 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.

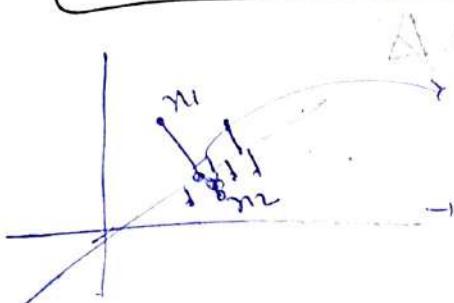
### steps in SVD approach

- ① Standardize columns of  $A$ , to ensure each feature contributes equally to analysis.
- ② compute SVD of the centered data matrix to obtain matrices  $U$ ,  $\Sigma$ , and  $V^T$ , such that,  
$$X = U \Sigma V^T$$
 where  $U$  contains left singular vectors,  $\Sigma$  holds the singular values &  $V^T$  includes the right singular vectors.
- ③ select principal components; the principal components are obtained by selecting the columns for  $U$  corresponding to the largest singular values in  $\Sigma$ , these principal components represent the directions of maximum variance in the data, etc.

## Note:-

→ what does high variance mean?

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



Now, how do we distinguish  
 $m_1 \& m_2$ .

→ And how do we reconstruct  
 $m_1 \& m_2$ .

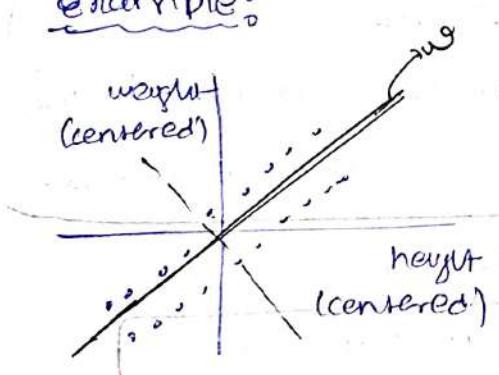
→ So, if all the datapoints are crowded,  
then we can't reconstruct and the  
information is lost.

This happens due to low 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 new directions are  
de-correlated, 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 data.

(\*) What is the optimal no. of PCs 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, i.e.,

the proportion of variance explained by each principal component

Explained variance ratio = variance by m<sup>th</sup> PC / total variance

## Download Deep Learning Notes

<https://t.me/AIMLDeepThought/663>

$$\text{EV}_k = \frac{\left( \sum_{j=1}^n w_{jk} x_{ij} \right)^2}{\sum_{j=1}^n \sum_{i=1}^p x_{ij}^2}$$

$$\sum_{j=1}^p \sum_{i=1}^n x_{ij}^2$$

$$\rightarrow \text{EV}_k \text{ or } \text{PC}_k = \frac{\text{Distance of PC}_k \text{ point}}{\text{Distance of PC}_k \text{ point} + \text{PC}_1 + \dots + \text{PC}_{k-1}}$$

If EV<sub>k</sub> or PC<sub>k</sub> = 0.91, then that means PC<sub>k</sub> 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- 95%).

Eg:-

$$\text{EV}R \text{ or } PC_1 = 0.88$$

$$\text{EV}R \text{ or } PC_2 = 0.10$$

$$\text{EV}R \text{ or } PC_3 = 0.05$$

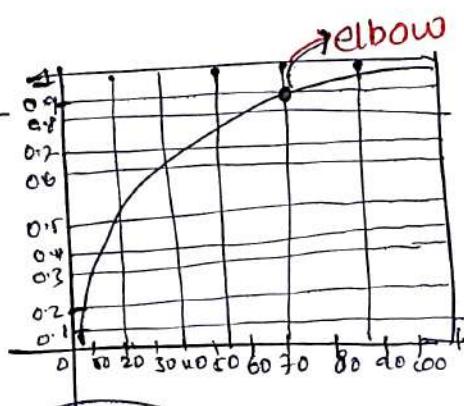
→ Now, if we want to preserve (99%) variance, then we can just take  $PC_1 \& PC_2$ , as they sum up to 0.94, 94% variance.

→ Instead if we want to preserve (99.5%) of variance, then we should take  $PC_1, PC_2 \& PC_3$ . as the sum up to 0.99, 99%. 99.5%.

### 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.

## Issue/concern with PCA

1. Time complexity

→ in particular,  
covariance matrix.

→ usually, it takes

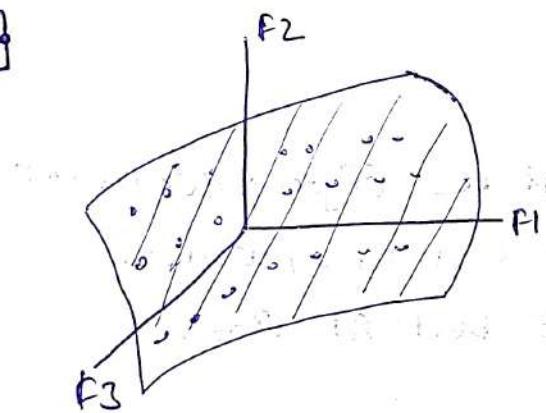
$$O(d^3)$$

→ 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.

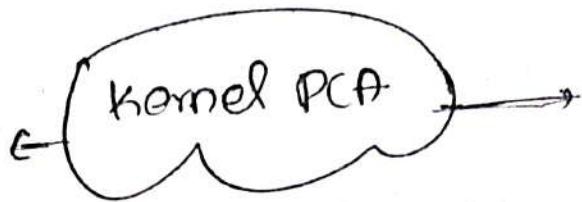


Data may not necessarily lie in a low-dimensional linear subspace

→ features will not be linearly related.

### Notes

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



• Input —  $\{n_1, \dots, n_n\}$   $n_i \in \mathbb{R}^d$ ,

kernel  $K: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

Step ① compute  $K \in \mathbb{R}^{n \times n}$ ,

where  $K_{ij} = K(n_i, n_j)$   $\forall i, j$ .

And then we need to center the kernel

Step ② compute  $\beta_1, \dots, \beta_d$  eigen vectors &

$n_{1,1}, \dots, n_{d,d}$  eigen values.

Step and normalize to get,

$$\alpha_k = \frac{\beta_k}{\sqrt{n_{kk}}}$$

Step ③

$$w_k = \phi(n) \cdot \alpha_k$$

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

we cannot  
reconstruct the  
eigen vectors or  
the covariance  
matrix

→ But we can still compute the  
compressed representation.

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

$$\phi(m_i)^T w_k = \sum_{j=1}^n \alpha_{kj} \cdot k_{pj}$$

→ 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, or these data points.

→ this is where dimensionality reduction happens.

(eg)

original datapoint was 100 dimensions,  
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 representations  
in a low dimensional space, as put the  
projections onto these eigen directions.  
you can use these projection values  
at your data point and work this for a  
downstream task, (supervised learning task)

$$(\phi(m_i)^T w_k) = (\phi(m_i) \cdot \frac{w}{\|w\|}) \cdot \frac{w}{\|w\|} = \text{scalar}$$

modified

Step ③

compute  $\sum_{j=1}^n \alpha_{kj} \cdot k_{ij} + k$

$$\begin{matrix} n \\ \in \mathbb{R}^d \end{matrix} \rightarrow \left[ \sum_{j=1}^n \alpha_{1j} \cdot k_{1j}, \sum_{j=1}^n \alpha_{2j} \cdot k_{2j}, \dots, \sum_{j=1}^n \alpha_{Lj} \cdot k_{Lj} \right]$$

→ 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,  $k \in \mathbb{R}^{n \times n}$

$$k_{ij} = k(n_i^o, n_j^o) + i_j^o$$

create a new kernel  $k^c$  — centered.

$$k_{ij}^c = k_{ij} - \theta_i l_j^T - l_i \theta_j^T + p l_i \cdot l_j^T$$

where,

$$\theta_i^o = \frac{1}{n} \sum_{p=1}^n k_{ip} \cdot k_p$$

$$p = \frac{1}{n} \sum_{p,j} k_{pj}$$

→ 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).	<ul style="list-style-type: none"><li>1. Fast training because there are few parameters.</li><li>2. Interpretable/Explainable results by its output coefficients.</li></ul>	<ul style="list-style-type: none"><li>1. Assumes a linear relationship between input and output variables.</li><li>2. Sensitive to outliers.</li><li>3. Typically generalizes worse than ridge or lasso regression.</li></ul>
Polynomial Regression	Polynomial Regression models nonlinear relationships between the dependent, and independent variable as the n-th degree polynomial.	<ul style="list-style-type: none"><li>1. Provides a good approximation of the relationship between the dependent and independent variables.</li><li>2. Capable of fitting a wide range of curvature.</li></ul>	<ul style="list-style-type: none"><li>1. Poor interpretability of the coefficients since the underlying variables can be highly correlated.</li><li>2. The model fit is nonlinear but the regression function is linear.</li><li>3. Prone to overfitting.</li></ul>
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.	<ul style="list-style-type: none"><li>1. Robust against outliers.</li><li>2. Effective learning and strong generalization performance.</li><li>3. Different Kernel functions can be specified for the decision function.</li></ul>	<ul style="list-style-type: none"><li>1. Does not perform well with large datasets.</li><li>2. Tends to underfit in cases where the number of variables is much smaller than the number of observations.</li></ul>
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.	<ul style="list-style-type: none"><li>1. Provides uncertainty measures on the predictions.</li><li>2. It is a flexible and usable non-linear model which fits many datasets well.</li><li>3. Performs well on small datasets as the GP kernel allows to specify a prior on the function space.</li></ul>	<ul style="list-style-type: none"><li>1. Poor choice of kernel can make convergence slow.</li><li>2. Specifying specific kernels requires deep mathematical understanding</li></ul>

# 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 (0 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.

**Download Deep Learning Notes**

**<https://t.me/AIMLDeepThaught/663>**

# 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.	<ul style="list-style-type: none"> <li>1. Scales to large datasets</li> <li>2. Interpretable &amp; explainable results</li> <li>3. Can generate tight clusters</li> </ul>	<ul style="list-style-type: none"> <li>1. Requires defining the expected number of clusters in advance.</li> <li>2. Not suitable to identify clusters with non-convex shapes.</li> </ul>
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.	<ul style="list-style-type: none"> <li>1. No assumption on the expected number of clusters.</li> <li>2. Can handle noisy data and outliers</li> <li>3. No assumptions on the shapes and sizes of the clusters</li> <li>4. Can identify clusters with different densities</li> </ul>	<ul style="list-style-type: none"> <li>1. Requires optimization of two parameters</li> <li>2. Can struggle in case of very high dimensional data</li> </ul>
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.	<ul style="list-style-type: none"> <li>1. No assumption on the expected number of clusters</li> <li>2. Can handle noisy data and outliers.</li> <li>3. No assumptions on the shapes and sizes of the clusters.</li> <li>4. Can identify clusters with different densities</li> </ul>	<ul style="list-style-type: none"> <li>1. Mapping of unseen objects in HDBSCAN is not straightforward.</li> <li>2. Can be computationally expensive</li> </ul>
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.	<ul style="list-style-type: none"> <li>1. There is no need to specify the number of clusters.</li> <li>2. With the right linkage, it can be used for the detection of outliers.</li> <li>3. Interpretable results using dendograms.</li> </ul>	<ul style="list-style-type: none"> <li>1. Specifying metric and linkages types requires good understanding of the statistical properties of the data</li> <li>2. Not straightforward to optimize</li> <li>3. Can be computationally expensive for large datasets</li> </ul>
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.	<ul style="list-style-type: none"> <li>No assumption on the expected number of clusters.</li> <li>2. Can handle noisy data and outliers.</li> <li>3. No assumptions on the shapes and sizes of the clusters.</li> <li>4. Can identify clusters with different densities.</li> <li>5. Not required to define fixed radius as in DBSCAN.</li> </ul>	<ul style="list-style-type: none"> <li>1. It only produces a cluster ordering.</li> <li>2. Does not work well in case of very high dimensional data.</li> <li>3. Slower than DBSCAN.</li> </ul>

## 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.	<ul style="list-style-type: none"> <li>1. Explainable Interpretable results.</li> <li>2. New unseen datapoints can be mapped into the existing PCA space</li> <li>3. Can be used as dimensionality reduction technique as preliminary step to other machine learning tasks</li> <li>4. Helps reduce overfitting</li> <li>5. Helps remove correlated features</li> </ul>	<ul style="list-style-type: none"> <li>1. Sensitive to outliers</li> <li>2. Requires data standardization</li> </ul>
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	<ul style="list-style-type: none"> <li>1. Helps preserve the relationships seen in high dimensionality</li> <li>2. Easy to visualise the structure of high dimensional data in 2 or 3 dimensions</li> <li>3. Very effective for visualizing clusters or groups of data points and their relative proximities</li> </ul>	<ul style="list-style-type: none"> <li>1. The cost function is not convex: different initializations can get different results.</li> <li>2. Computationally intensive for large datasets.</li> <li>3. Default parameters do not always achieve the best results</li> </ul>
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.	<ul style="list-style-type: none"> <li>1. It can be used as general-purpose dimension reduction technique as a preliminary step to other machine learning tasks.</li> <li>2. Can be very effective for visualizing clusters or groups of data points and their relative proximities.</li> <li>3. Able to handle high dimensional sparse datasets</li> </ul>	<ul style="list-style-type: none"> <li>1. Default parameters do not always achieve the best results</li> </ul>
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.	<ul style="list-style-type: none"> <li>1. Can separate multivariate signals into its subcomponents</li> <li>2. Clear aim of the method: only applicable if there are multiple independent generators of information to uncover.</li> <li>3. Can extract hidden factors in the data by transforming a set of variables to new set that maximally independent.</li> </ul>	<ul style="list-style-type: none"> <li>1. Without any prior knowledge, determination of the number of independent components or sources can be difficult.</li> <li>2. PCA is often required as a pre-processing step.</li> </ul>

## 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

**Download Deep Learning Notes**

**<https://t.me/AIMLDeepThaught/663>**

**• Join me on LinkedIn for the latest updates on Machine Learning: <https://www.linkedin.com/groups/7436898/>**

**• Join my WhatsApp Channel for the latest updates on Machine Learning:**  
**<https://www.whatsapp.com/channel/0029VavNSDO9mrGWYirxz40G>**