Machine learning Algorithms

STUDY GUIDE

SUBIN VASANTHAN

This guide is prepared by Subin Vasanthan as a personal knowledge reference and learning companion for understanding on different Machine learning Algorithms.

Table of Contents

Book: Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow by O'Reilly

1. Overview of Machine Learning

Machine Learning

→ field of making machines learn from data ×

→ field of writing algorithms that enable Computers to learn from the data and make a decision using it !!

Agenda

Tuesday, 28 September 2021 8:05 AM

- Overview/Introduction to Machine Learning
- Machine Learning Algorithms
  - Linear Regression - Supervised Reg
  - Logistic Regression - Sup Class
  - Decision Trees - Sup Reg&Class
  - Random Forest - Sup Reg&Class
  - Support vector Machines - Sup Reg&Class
  - K-Means - UnSup
  - DB Scan - UnSup
  - Hierarchical Clustering - UnSup

Tuesday, 28 September 2021 9:07 AM

Types of data

→ Training data

→ Validation data

→ Testing data

Types of problems

→ Regression problem

→ Classification problem

[the type of target column]

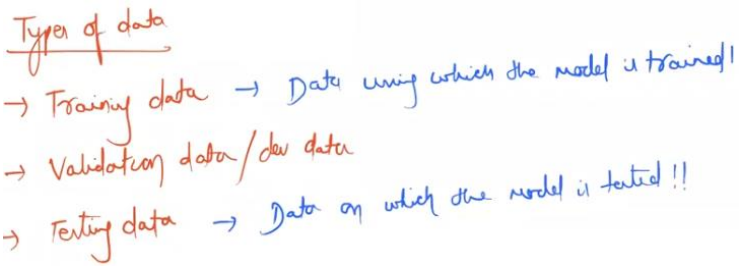
Types of approaches

→ Supervised approach

→ Unsupervised approach

[Presence / absence of target column in the training dataset.]

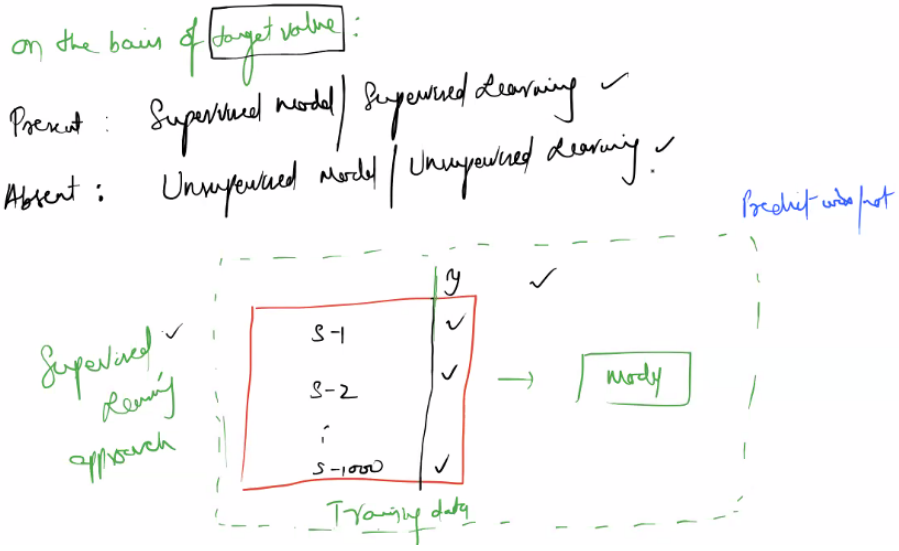
2. Types of Data



3. Supervised Learning / Model & Un- Supervised Learning / Model

a. Supervised Learning / Model

If the training data contains some target results for the scenarios given it is called supervised learning



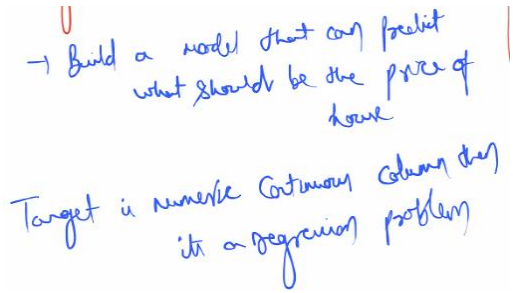
b. Un supervised Learning

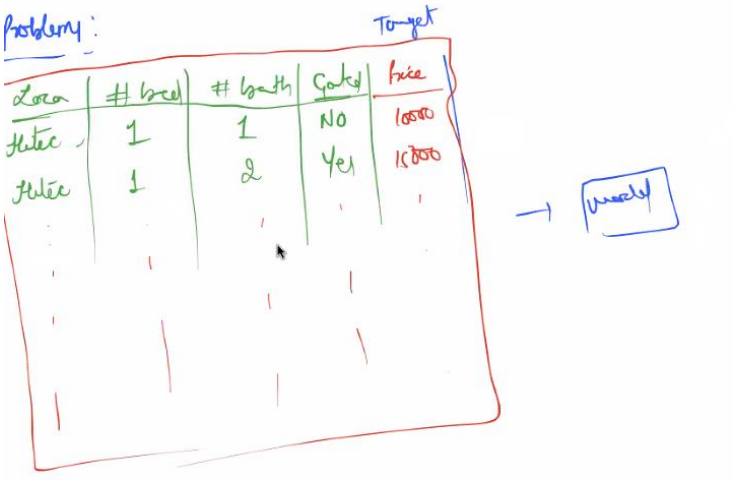
If the training data does not contain any target results for the scenarios given it is called unsupervised learning

4. Regression problem and Classification problem

Machine Learning can solve 2 kinds of Problems

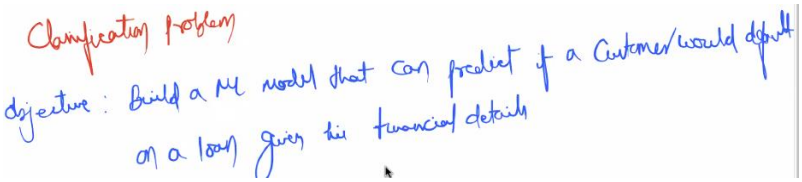
a. Regression problem





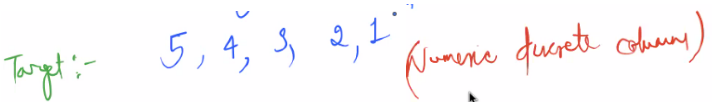
Given the above scenarios the model will learn the Target values.

b. Classification problem

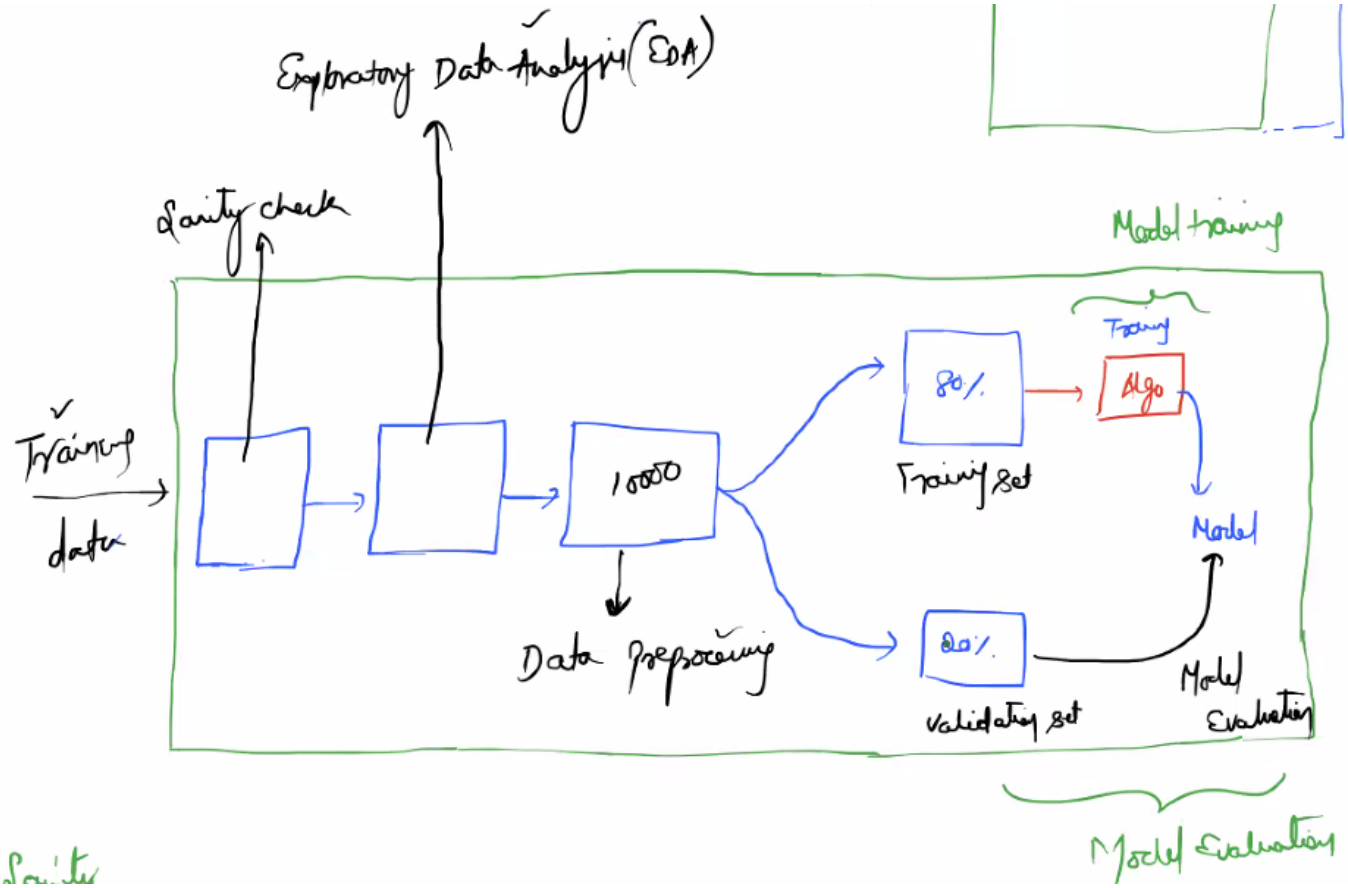


The target values will be categorical that is 1's or 0's whether he will default or not default. This is called classification problem

The classification problem can also have more than 2 values.
Like ratings



5. Building a Model



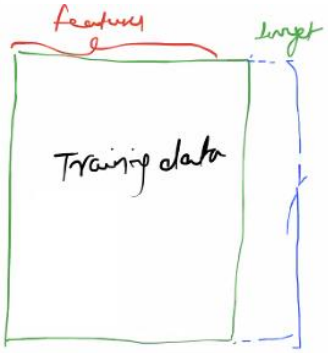
Sanity

Sanity

→ Do I have enough observ/ good quantity of data to build a model

→ Is the target column present in the training data & what kind of column it is

→ Are the features are locally related to the target !!



Exploratory Data Analysis

→ Are there missing values present in the data

→ Feature composition of the data

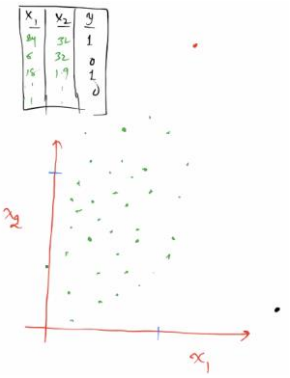
→ Are there any outliers in the data

→ Distribution of each of the features

→ Scales of the features

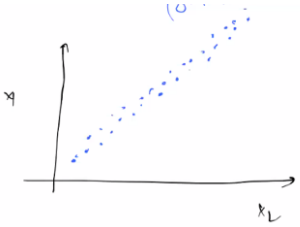
→ Correlation b/w features & the target

Outliers



The points that are significantly different are called outliers like the one red and blue in the above plot

Positive correlation



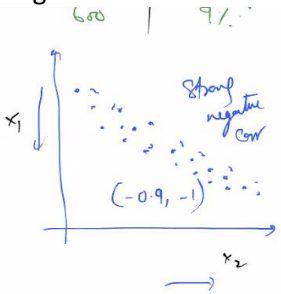
Correlation is a method to measure the linear relationship between two numeric continuous columns

$-1 \leq \text{Correlation} < 1$

Correlation = 1 (very strongly positively related)

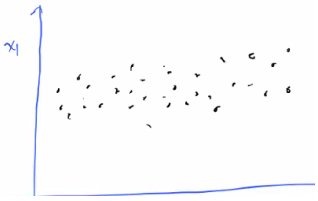
Correlation = -1 (very & strongly negatively related)

Negative correlation



Correlation = 0 (there is no relationship b/w the columns)

Zero correlation



Between Feature ? Low correlation is desired

Between Feature and Target ? High Correlation is desired

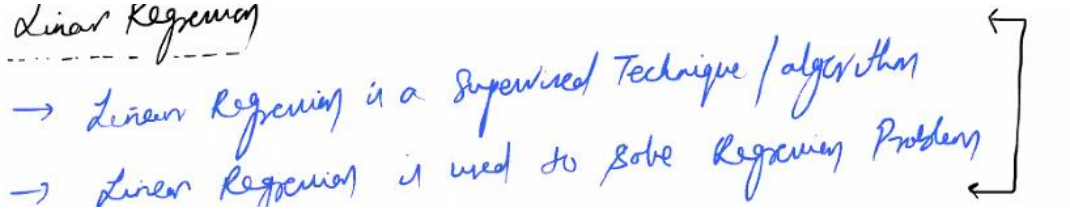
Data Preprocessing:

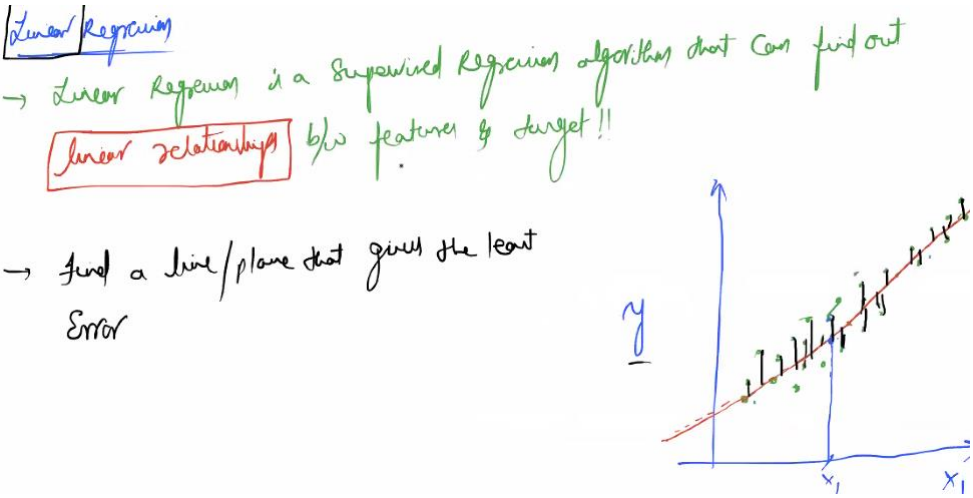
At this stage we treat all the problems like outliers, missing data etc. encountered at the EDA

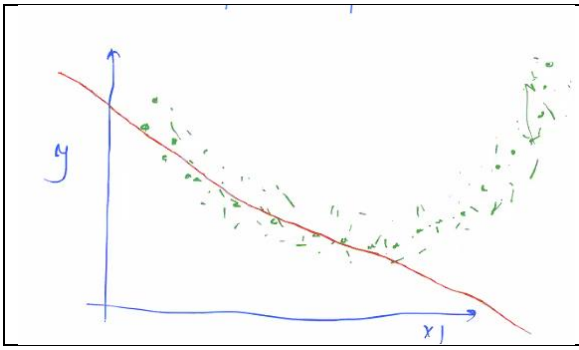
6. Machine Learning Algorithms

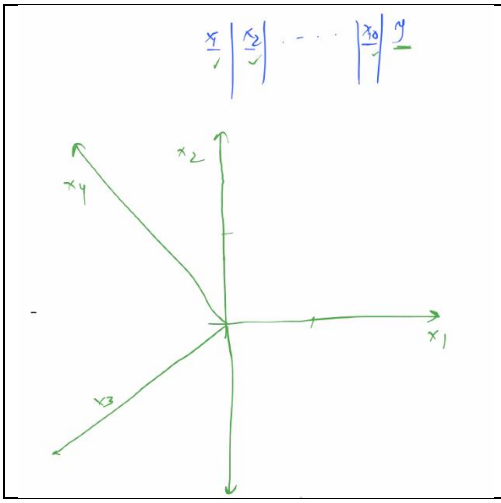
a. SUPERVISED MODELS

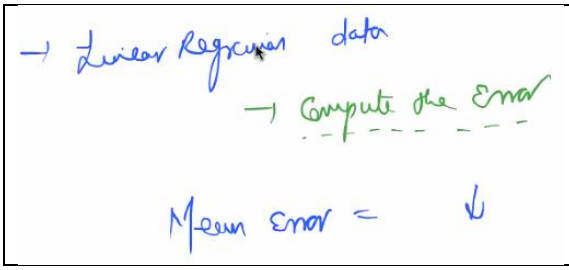
i. Linear Regression Algorithm – (Supervised Regression)



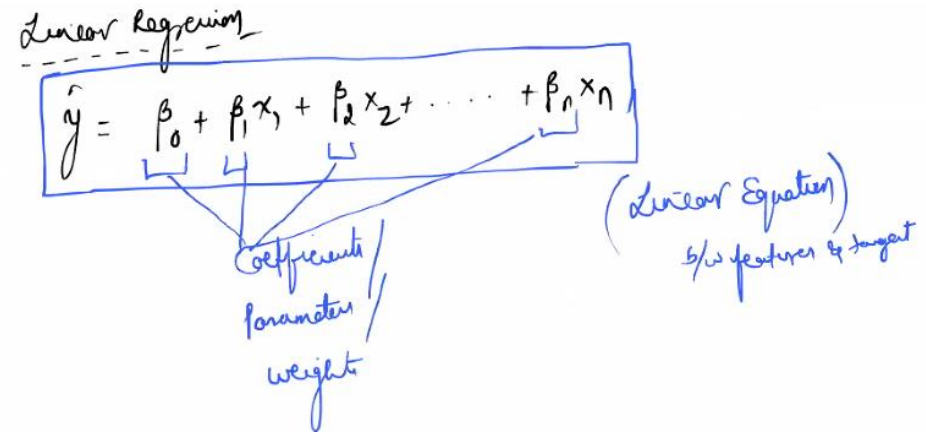


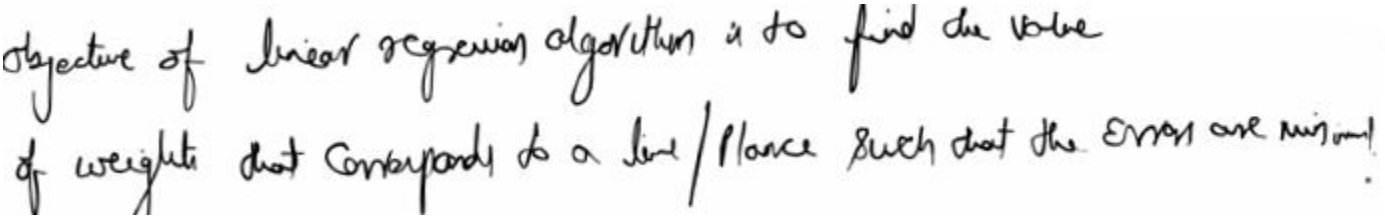
	The below is not good for Linear regression. SO we have to go for another algorithm

	In real time there will be many dimensions that is many inputs for a target.

	If the error is low, we will use linear regression other wide if the error is big we will choose other algorithms

Linear Regression Equation





Learning Steps

Linear Reg. Equation $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

Learning Steps

Step-0: Randomly assign the values of $\beta_0, \beta_1, \beta_2$

$\beta_0 = 2, \beta_1 = 10, \beta_2 = 3$ $(\hat{y} = 2 + 10x_1 + 3x_2)$

Step-1: Using the Current Equation, Predict on the training set

x_1	x_2	y	$\hat{y}$
1	1	6	15
2	2	8	28
3	3	9	41

Step-2: Compute the loss (error) due to the current predictions

x_1	x_2	y	$\hat{y}$	Error
1	1	6	15	9
2	2	8	28	20
3	3	9	41	32

Loss func = Mean squared Error $= \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$

x_1	x_2	y	$\hat{y}$	Sq Error
1	1	6	12	$(12-6)^2 = 36$
2	2	8	20	$(20-8)^2 = 144$
3	3	9	30	$(30-9)^2 = 441$

Mean ✓

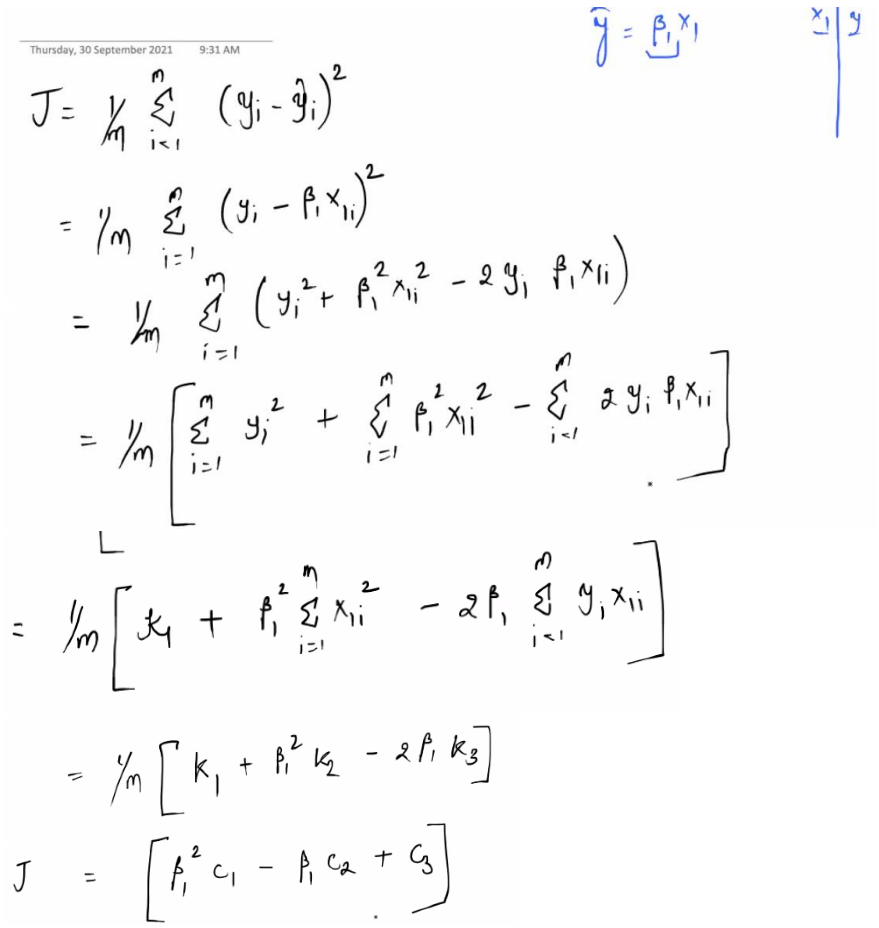
Step-3: Magic it will see the current error (due to the current set of β's) and update the values of β such that the Error reduces

$(\beta_0 = 1.8, \beta_1 = 5, \beta_2 = 2.3)$

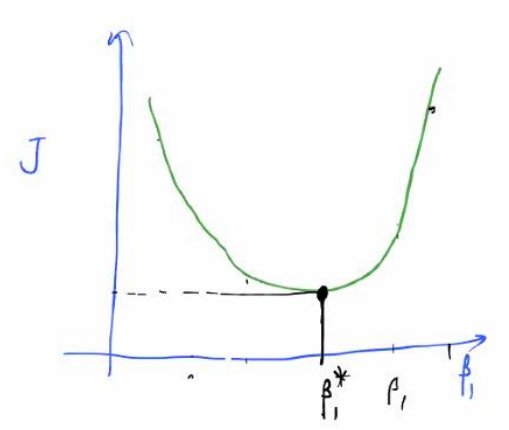
After this it will go to step-1 and repeat this until the model is improved to give a least value of error.

The Word Magic used in Step 3 is called Optimizers

Below is how we optimize



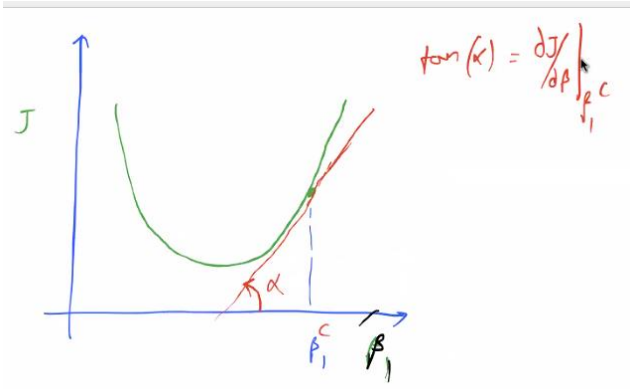
This equation is similar to y= ax^2+bx+c which is the equation of a parabola



So here we have the best value of B, where the error is low.

The formulae to get this is
Optimizer: Gradient Descent Algorithm

In the above dj/db is the derivative of J. Derivate is the angle that a value makes on the x axis.



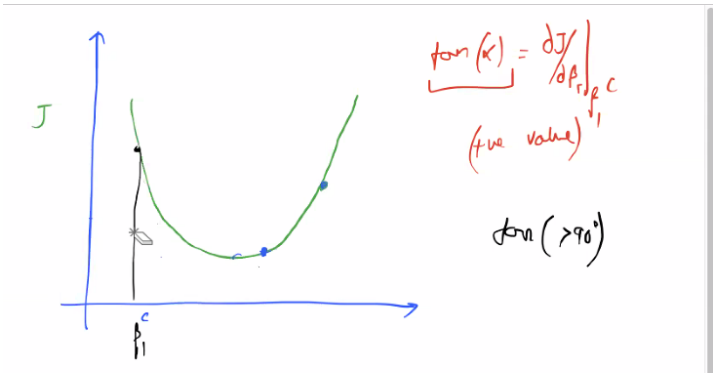
$\beta_1^u = \beta_1^c - \eta$ (very small value)

$\beta_1^u = \beta_1^c - \eta(0)$

$\beta_1^u = \beta_1^c$

We update the values until the tan of the value is close to zero.

If the B value chosen is on the left of the parabola, tan of the angle will be –ve. So in the equation the B will be increased to reach the bottom of the curve.



$\tan(\alpha) = \frac{dJ}{d\beta_1}\Big|_{\beta_1^c}$ (+ve value)

$\tan(>90°)$

$\beta_1^c = \beta_1^u - \eta\left(\frac{dJ}{d\beta_1}\right)\Big|_{\beta_1^c}$

$\beta_1^u = \beta_1^c - \eta$ (slope)

$\beta_1^u = \beta_1^c - \eta(-ve) = \beta_1^u = \beta_1^c + ve$

The above is explained using 2 parameters for the ease of explaining

But in real time we will have many parameters so in this case we will use gradient descent algorithm for all the values.

$$\beta_0^u = \beta_0^c - \eta \left(\frac{\partial J}{\partial \beta_0}\right)\Big|_{\beta_0^c}$$

$$\beta_1^u = \beta_1^c - \eta \left(\frac{\partial J}{\partial \beta_1}\right)\Big|_{\beta_1^c}$$

$$\beta_2^u = \beta_2^c - \eta \left(\frac{\partial J}{\partial \beta_2}\right)\Big|_{\beta_2^c}$$

Optimizer → Gradient descent

$$\beta_j^u = \beta_j^c - \eta \left(\frac{\partial J}{\partial \beta_j}\right)\Big|_{\beta_j^c}$$

Slope of the Cost function w.r.t β_j

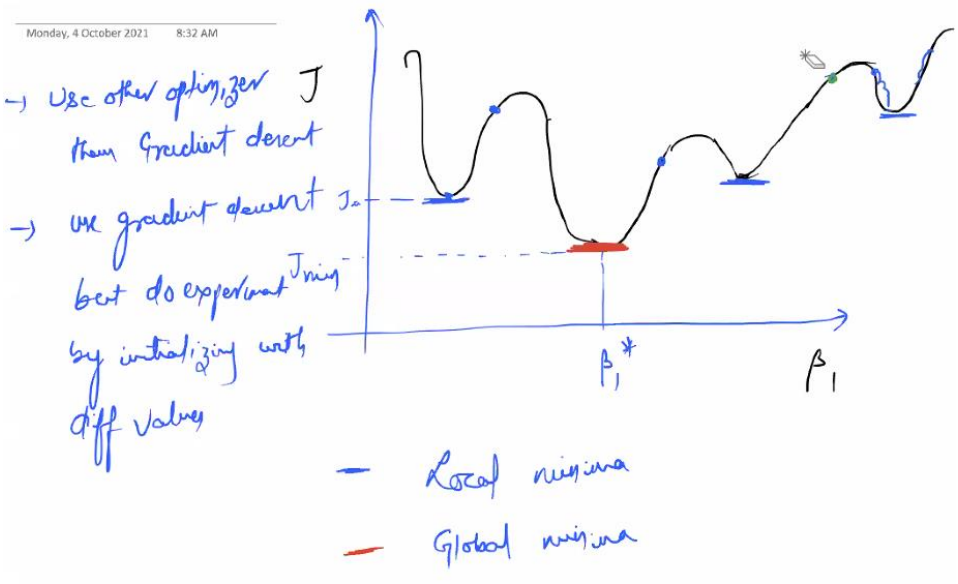
Learning rate (typically η is around 10^{-3})

η → Learning rate → Controls the update that is made to the Parameter

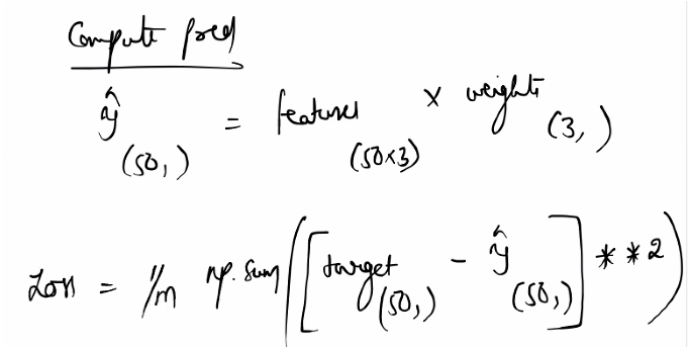
Advantage of η → we gradually can do minima

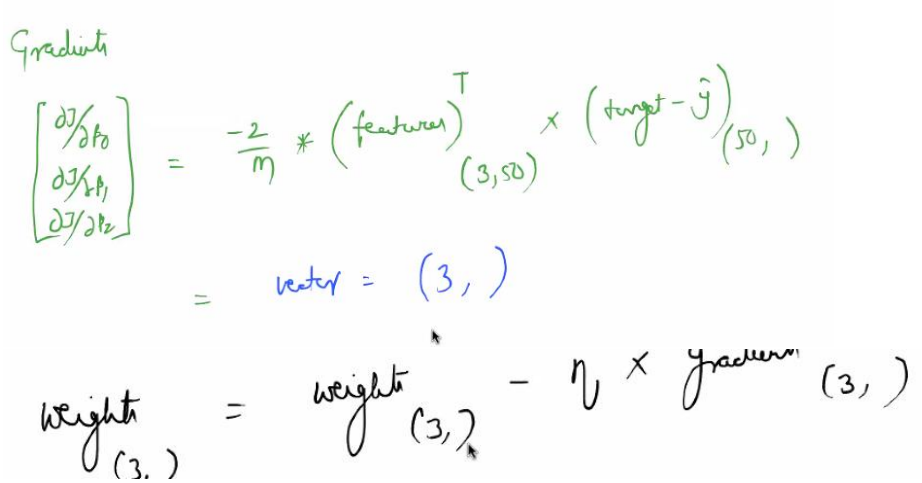
Disadvantage of η → we will have to do multiple iterations before minima is reached!!

The above is good for a linear regression equation, but in case of other functions the curve may not be always a inverted parabola. It can be wriggly curve as below



Below is just an extra

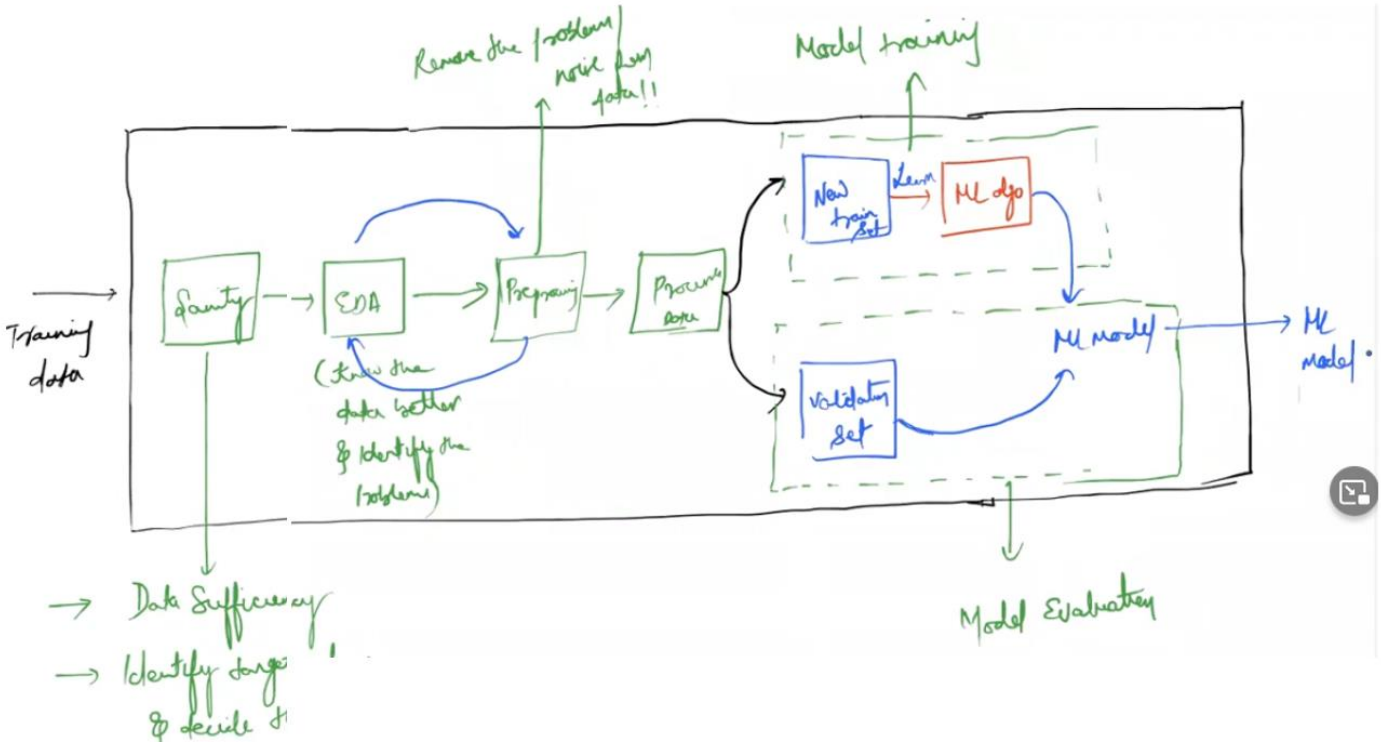




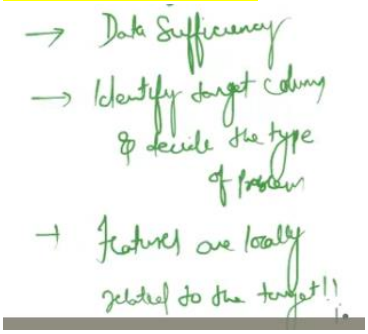
```
features_df = features.copy(deep = True) ### (50,3)
targets = y ### (50,)
weights = np.random.random(size = (3,)) #### Step-0 :random initialization ### (3,)

def prediction(weights, features_df):
    predicted_vals = np.dot(features_df,weights) ## This will be an vector which contains predicted values for all obs in features_df, shape = (50,)
    return (predicted_vals)


def error (targets,predicted):
    error_val = (targets-predicted)  ### (50,)
    return (error_val)

def cost_val(error_val):
    cost = np.mean(error_val**2)  #### mean squared error value  ### scalar
    return (cost)


#### pred_y = wo*xo + w1*x1 + w2*x2

##### cost_funct = (y - (wo*xo +w1*x1 +w2*x2))^2

##### gradient (cost_fucnt)/wo = 2 * (y- wo*xo + w1*x1 + w2*x2) * xo


def gradients(error_val,features_df):
    temp_gradients = np.dot(features_df.T, error_val)  ### vector of shape (3,)
    #error_val = np.reshape(error_val,newshape = (1,50))
    #temp_gradients = np.dot(error_val,features_df) ### (1,3) ###[grad(wo), grad(w1), grad(w2)]
    #temp_gradients = temp_gradients.T #### (3,1)
    temp_gradients = temp_gradients*(-2)
```

```
    temp_gradients = temp_gradients/(features_df.shape[0]) ## (3,1)
    grads = temp_gradients
    #grads = np.reshape(temp_gradients,-1) ###(3,)
    return(grads)


def linearReg(features_df,targets,weights,learning_rate,n_iter):

  cost_list= [] ### Store error after every upadte of weights

  for i in range(n_iter):
    ##### Step-1 : Compute prediction ###

    predict_vector = prediction(weights,features_df) ### shape (50,)

    ##### Step -2 : Compute cost #####

    error_vector = error(targets,predict_vector)

    cost_value = cost_val(error_vector) ## Mean squared error for the current predictions

    #### Step -3 : Gradient descent ######

    weights = weights - learning_rate * gradients(error_vector,features_df)

    cost_list.append(cost_value)

    #print("iteration : {} Cost : {}".format(i,cost_value))

  return(weights,error_list)
```

Implementing linear Regression by solving an End to End Problem



Step -1: Sanity Check



Step -2: EDA

EDA

→ Whether missing values present in the data

→ Whether outliers are present in the data

→ Look at the distribution of the features

→ What is the composition of data

→ Correlation between features -- feature & target !!

Step -3: Pre-processing

Preprocessing

→ Treat the missing values → Mandatory

→ Convert all columns into numeric types → Mandatory

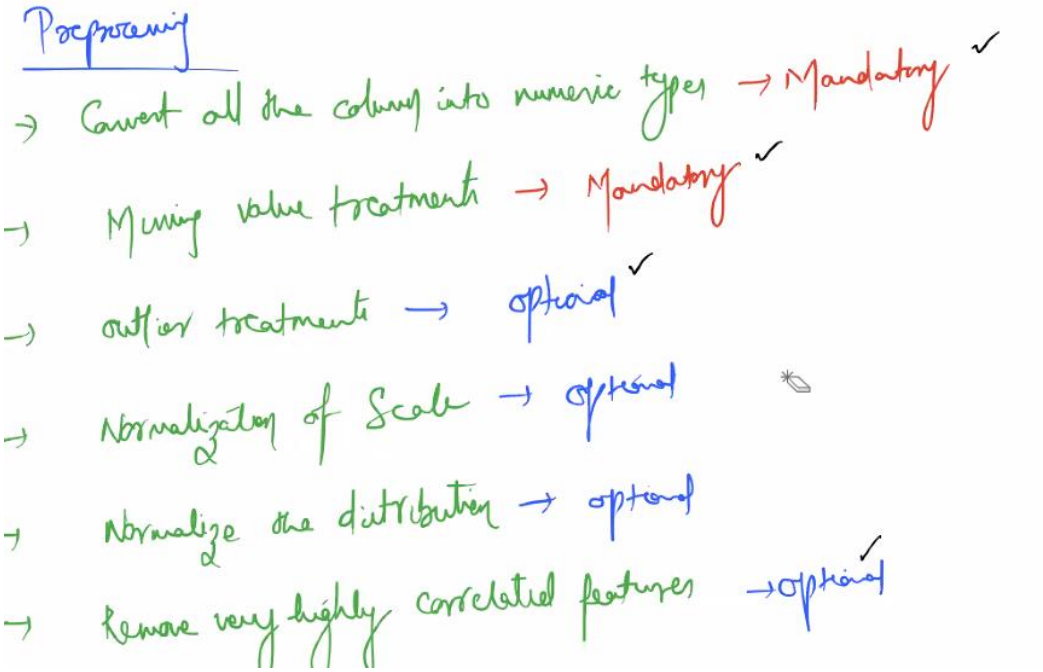
→ Treat outliers → optional

→ Normalize the scale of the data → optional

→ Remove very highly correlated features (>0.95/0.90) → optional

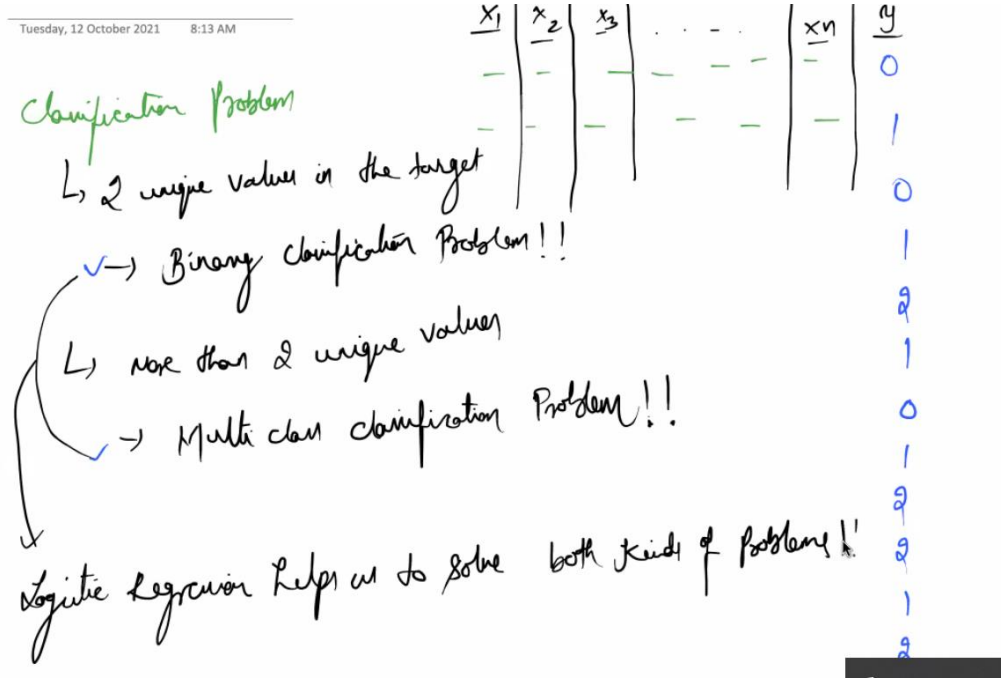
→ Normalize the data distribution → optional

1 → 0.95

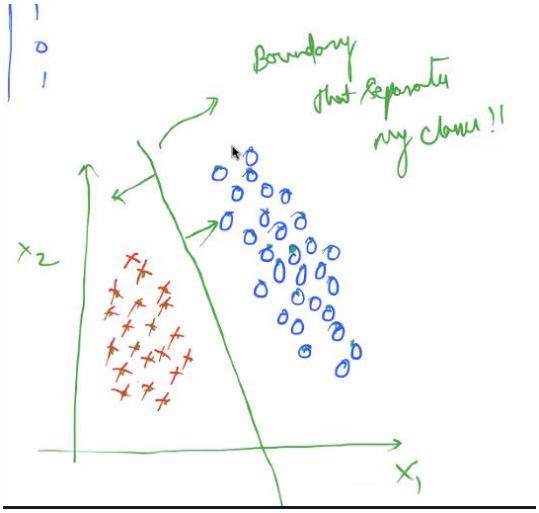


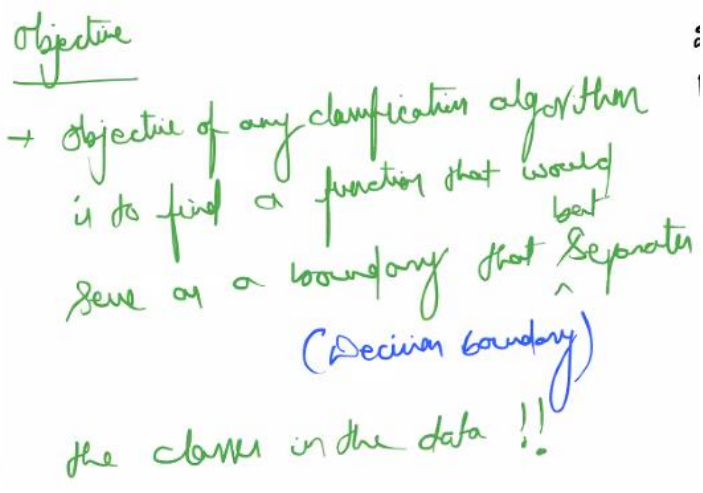
Step -4: Model Evaluation

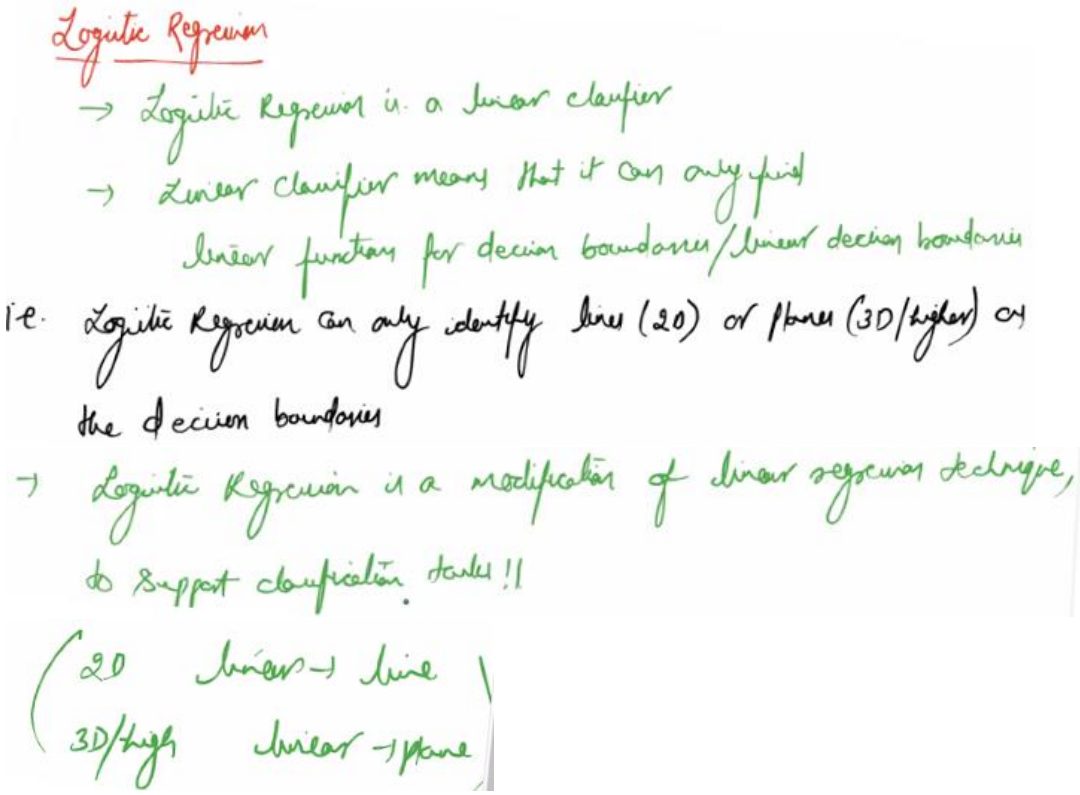
ii. Logistics Regression

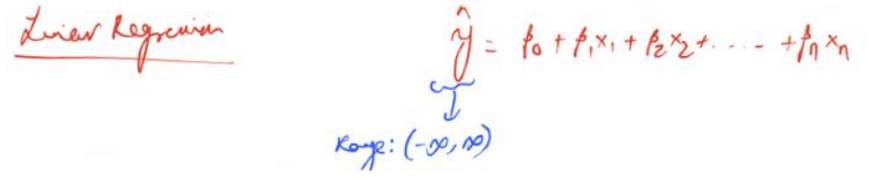


i. Binary Classification

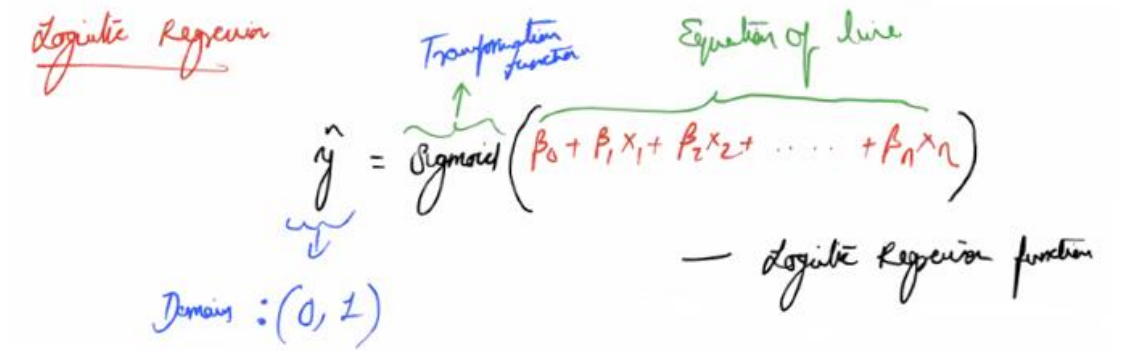








a. Equation



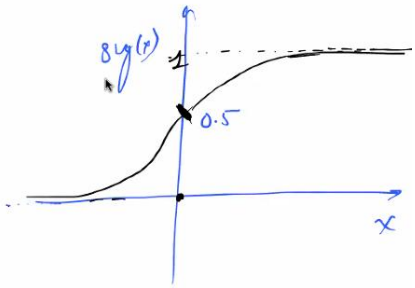
Y lies between 0 and 1

Now we will find out the values between 0 and 1 be classified to either 0 or 1.

Tuesday, 12 October 2021 8:42 AM

* $\hat{y} = Sig(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)$

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



$$\hat{y} = Sig(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)$$

$$\hat{y} = \frac{1}{1+e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

* $Sig(x) \rightarrow 0.5$

→ Probability that the model thinks the observation is class 1

$$\hat{y} = Sig(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)$$

$$\hat{y} = P(y=1 \mid x_1, \cdots, x_n) = Sig(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)$$

↓

Probability of given obs belonging to class 1

* Logistic Regression will output probabilities ! !

* once we get the probabilities, we can apply some threshold ...

$\hat{y} = Sig(x) > 0.5 \rightarrow 1$

$\leq 0.5 \rightarrow 0$

→ ideal/default threshold

But we can change the thresholds as per the experience

b. Learning Steps

$$\hat{y} = sig(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)$$

Step-0: Random Initialization of weight
ie $\beta_0, \beta_1, \beta_2 \cdots, \beta_n$ are randomly chosen

Step-1: Predict the probabilities using the current set of β's

Step-2: Compute the Loss/Error in the prediction

$$\text{Loss} = \text{Mean log loss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

Step-3: Apply an optimizer (Gradient descent) to optimize the values of β's

Linear Regression

Loss func: Mean Sq. Error

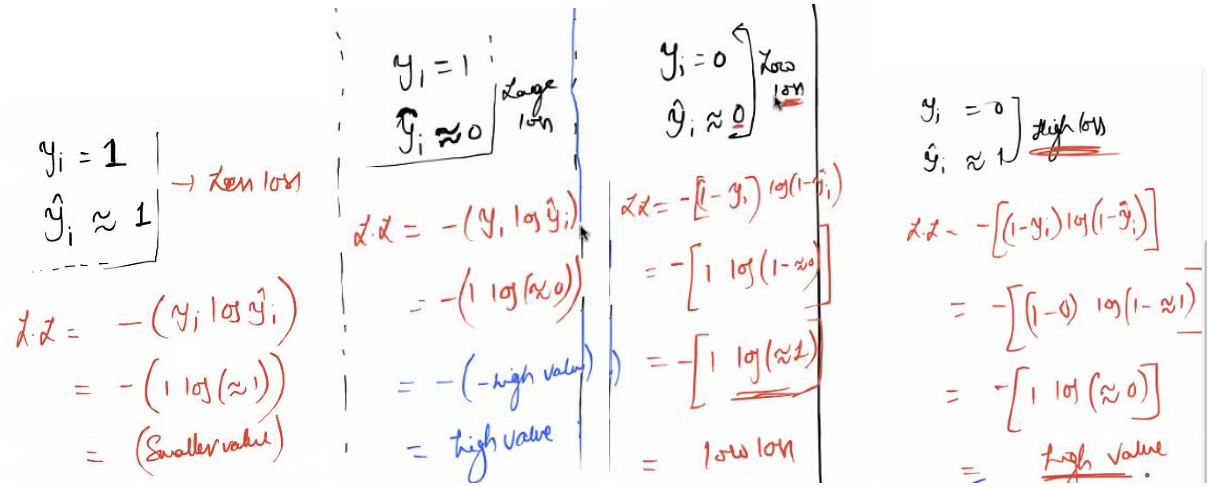
Logistic Regression

$$\text{Mean Log loss} = \frac{1}{n}\sum_{i=1}^{n} -\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$$

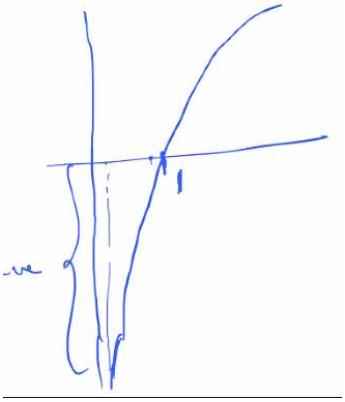
↓

Binary Cross Entropy loss

$$\text{Log loss} = -\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$$

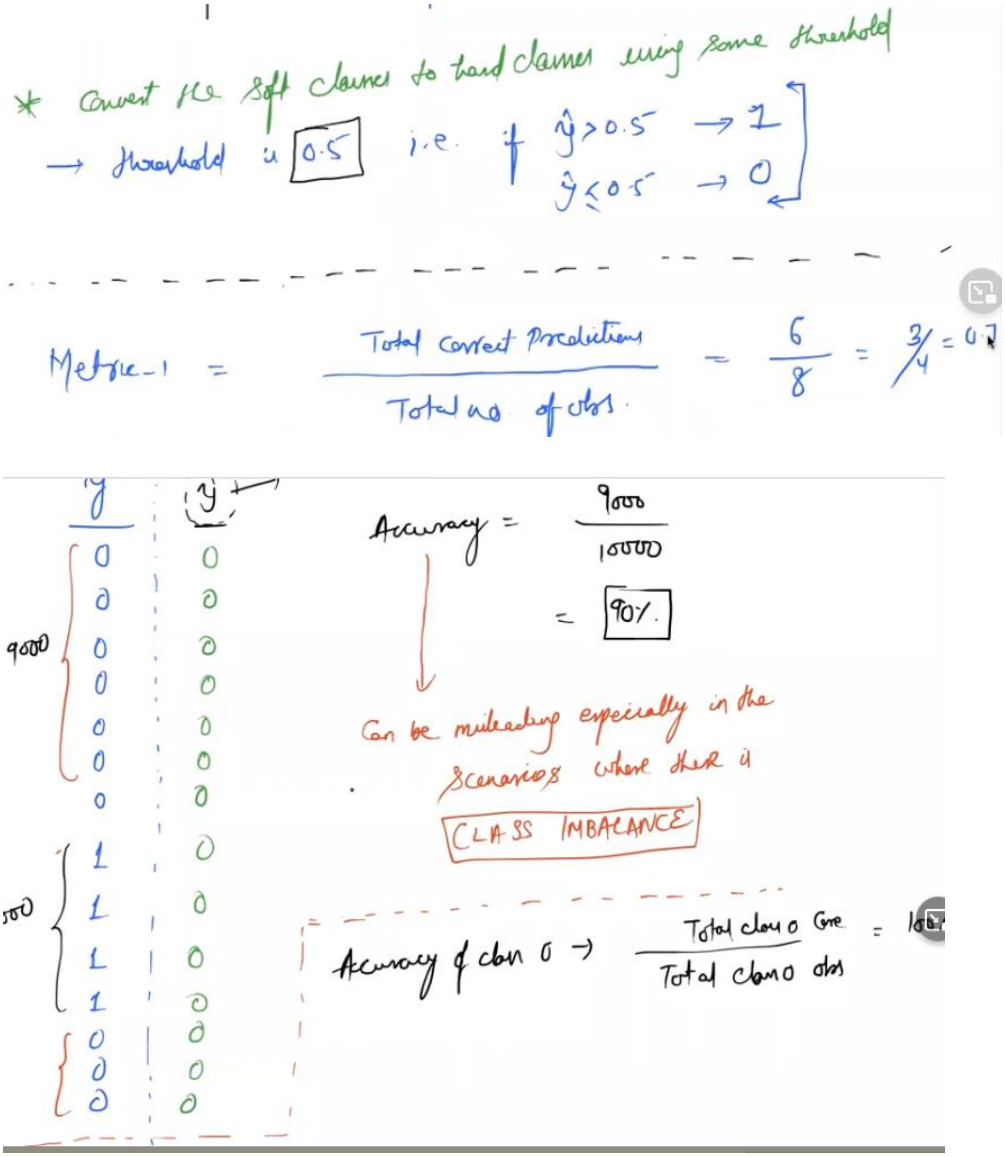


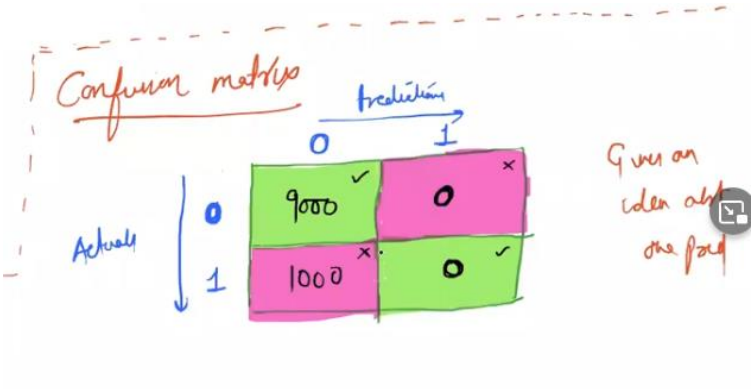
Log x graph

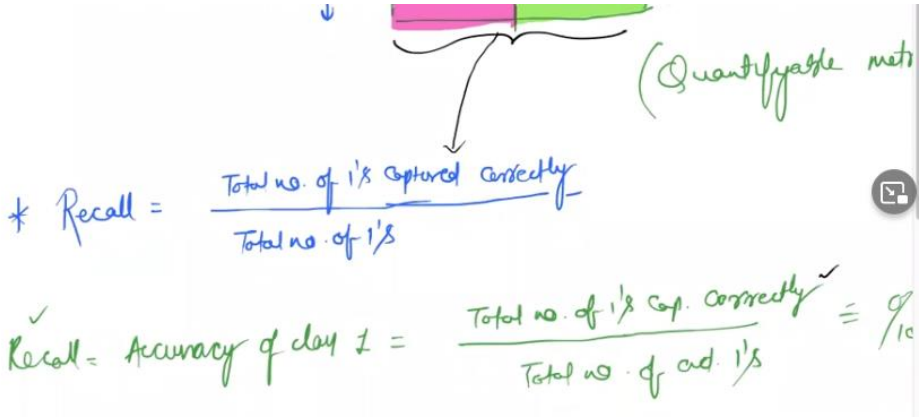


Evaluation metrics

y_{actual}	$\hat{y}_{pred}$	
0	1	F
1	1	T
0	0	T
1	1	T
1	0	F
1	1	T
0	0	T
1	1	T

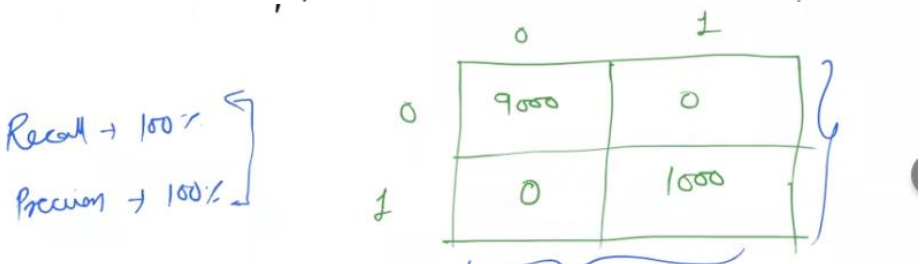

* Convert the soft classes to hard classes using some threshold
→ threshold is 0.5 i.e. if ŷ > 0.5 → 1
ŷ ≤ 0.5 → 0
Metric-1 = Total correct predictions / Total no of obs. = 6/8 = 3/4 = 0.7
y
ŷ
9000
Accuracy = 9000/10000
= 90%.
Can be misleading especially in the scenarios where there is
CLASS IMBALANCE
Accuracy of class 0 → Total class 0 Corr / Total class 0 obs


Confusion matrix
Predictions
0
1
Actuals
9000
0
1000
0
Given an



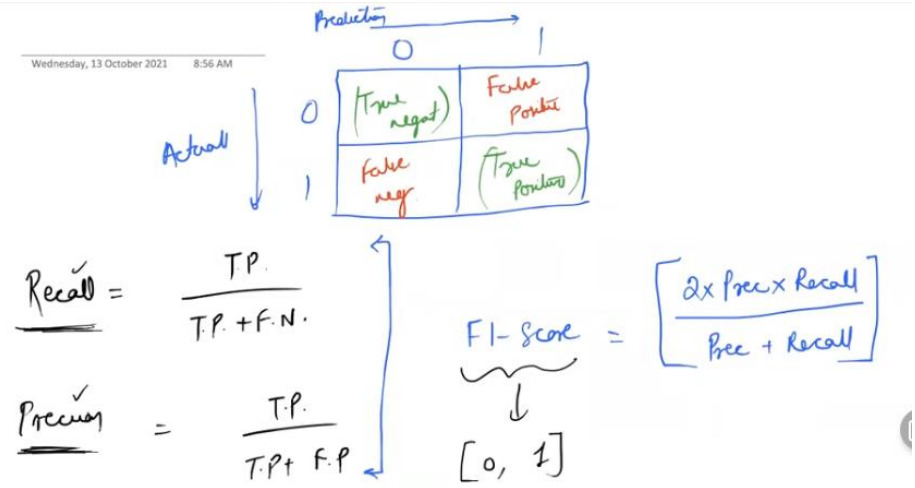
Recall → out of all 1's .. .. How many 1's are Captured Correctly

Precision → out of all predictions of 1's → How many are correct



(A good model is a model that high values of both Precision & Recall)

Wednesday, 13 October 2021 8:56 AM



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

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

$$\text{F1-Score} = \left[\frac{2 \times \text{Prec} \times \text{Recall}}{\text{Prec} + \text{Recall}}\right]$$

$$\downarrow$$

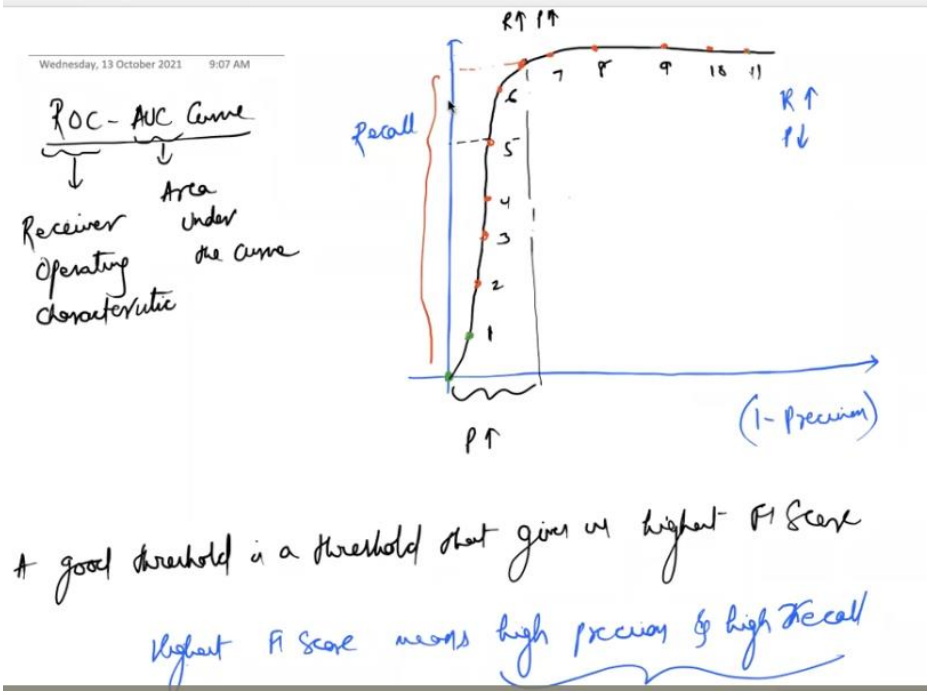
$$[0, 1]$$

Any F1 score closer to 1 represents good model, any F1 score closer to 0 represents a bad model!!

Higher the F1 score better the model!!

0.1 → F1 score
0.2 → F1 score
0.3 → F1 score
0.4 →
0.5 →
0.6 →
0.7 →
0.8 →
0.9 →
1.0 →

Consider that value as the threshold that gives you highest F1 score!!



ii. Parameters for logistics regression

class sklearn.linear_model.LogisticRegression(*penalty='l2'*, *, *dual=False*, *tol=0.0001*, *C=1.0*, *fit_intercept=True*, *intercept_scaling=1*, *class_weight=None*, *random_state=None*, *solver='lbfgs'*, *max_iter=100*, *multi_class='auto'*, *verbose=0*, *warm_start=False*, *n_jobs=None*, *l1_ratio=None*)

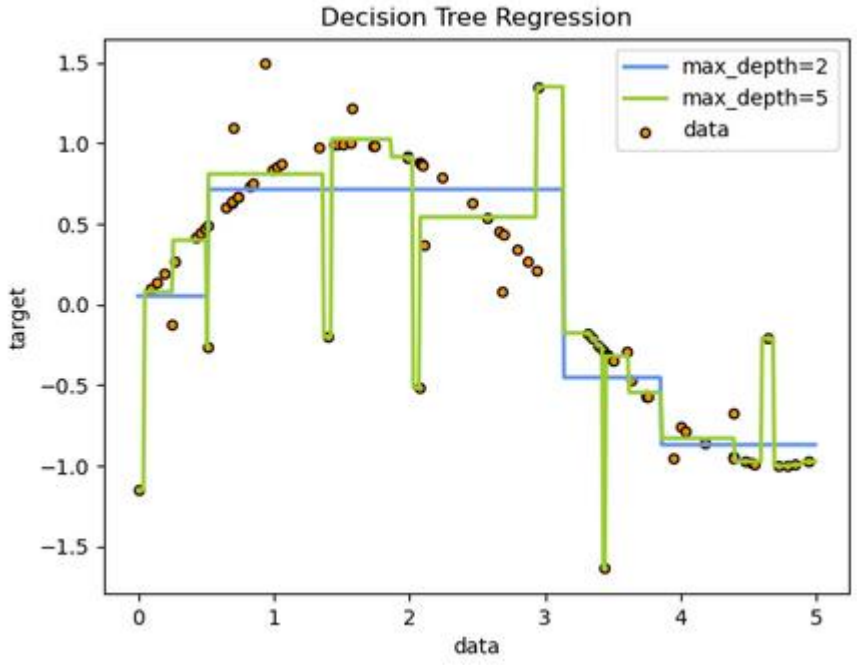
Refer to https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
For all the defenitions

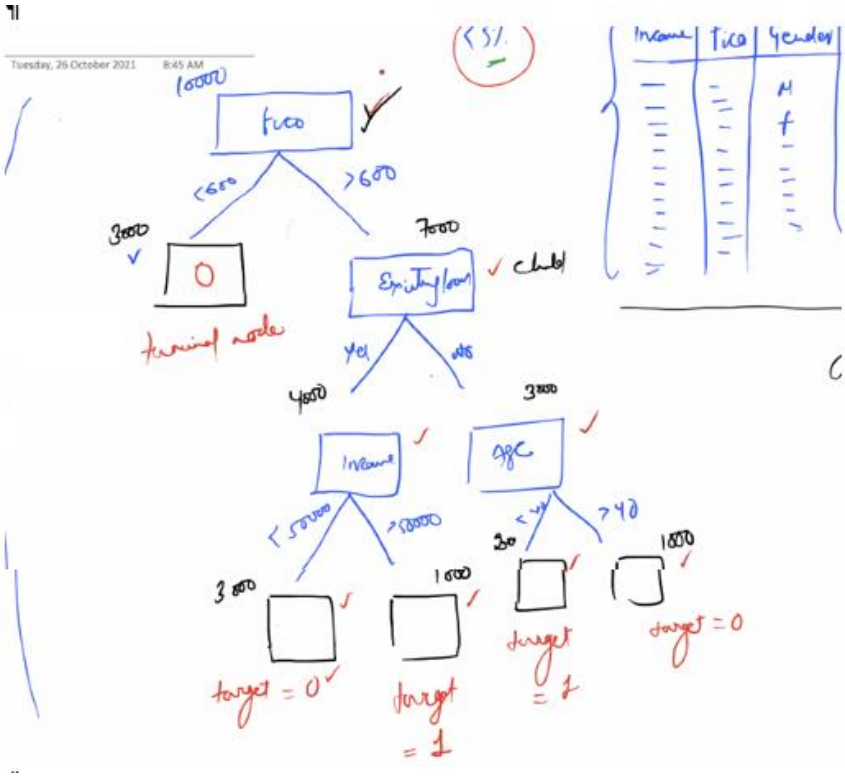
iii. Decision Trees

Decision Trees are supervised learning algorithms which can be used for solving both classification and regression tasks.

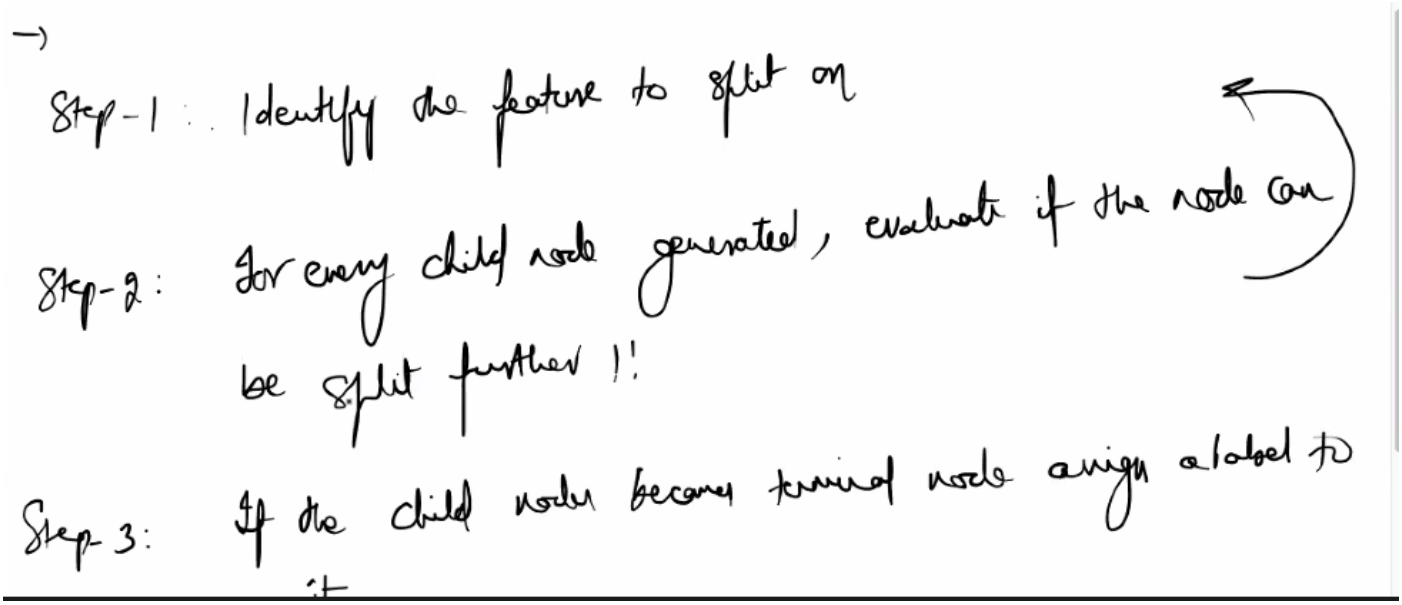
•Unlike logistic and linear regressions, decision trees can find both non linear and linear functions.

Decision Trees is a powerful algorithm when compared to linear and logistic regression.





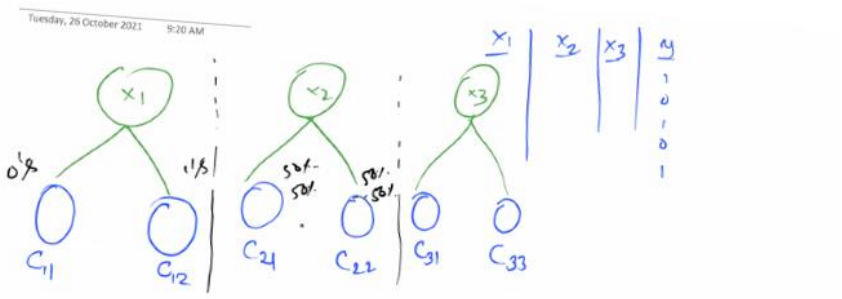
i. Learning Steps



ii. Learning Questions

* How does the algorithm decide the feature to split on ? ✓
* What are the different criterias that can help decide whether to split the node further or not ? ✓
* How does a splitting rule get decided for a numeric feature ?? ✓
* How does decision tree work for Regression problems ?? ✓

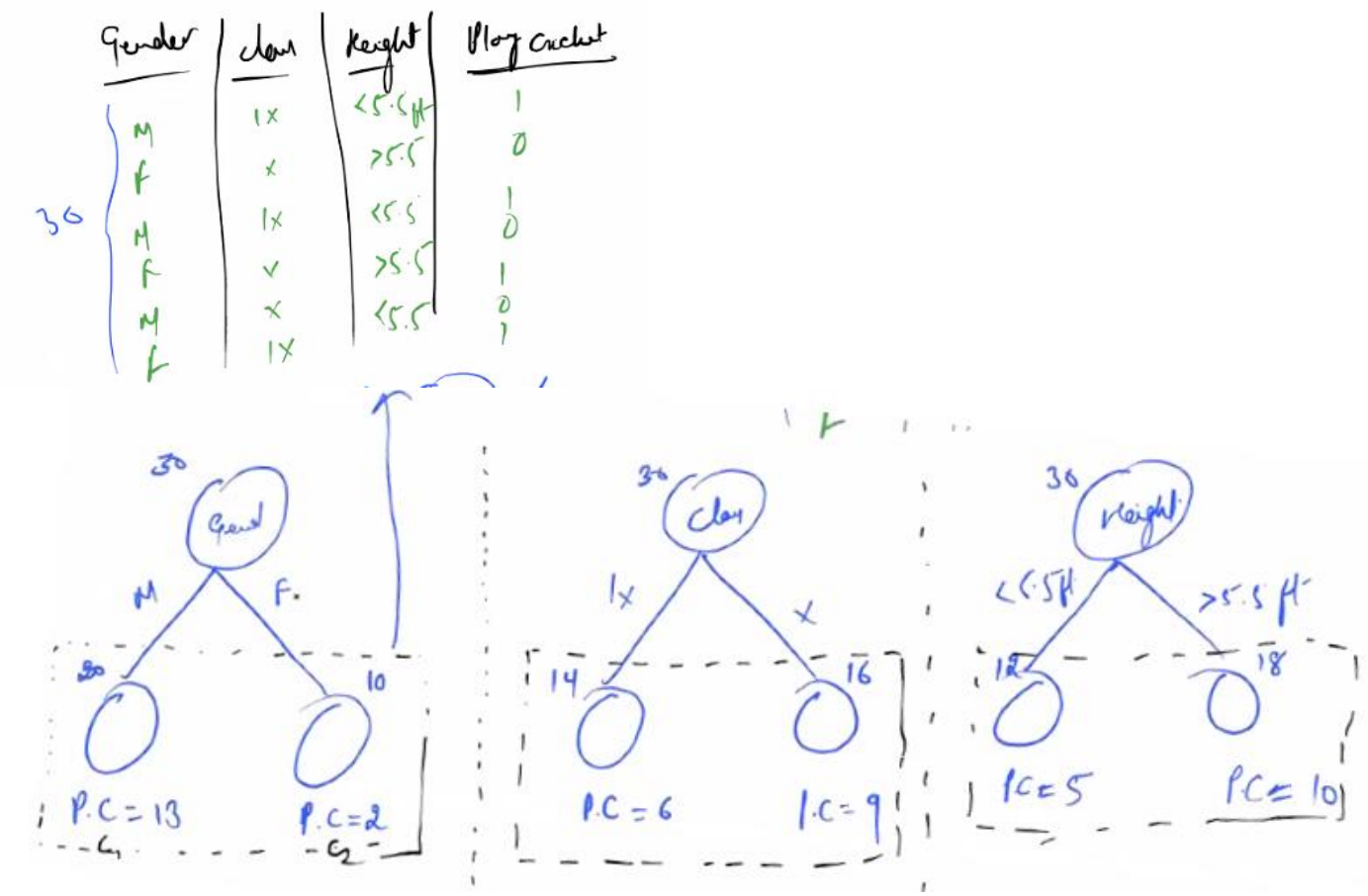
a. Splitting

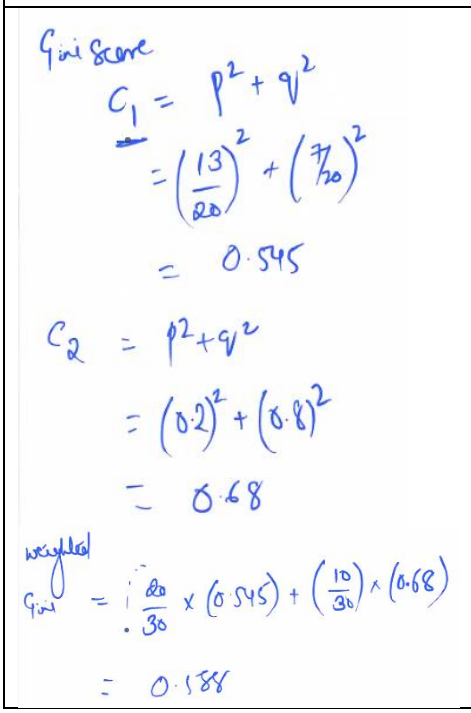
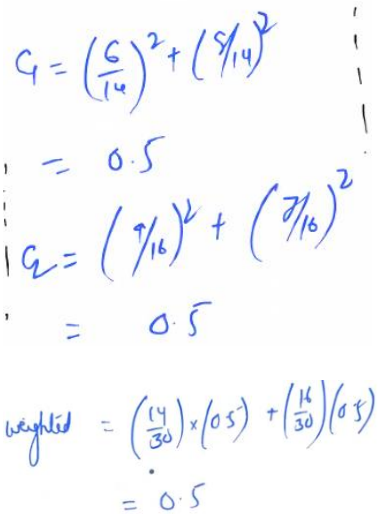
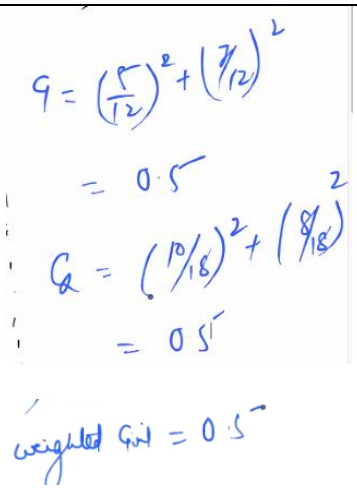


* It will select that split that separates out the positive & negative classes the best
* In other words, it will split on that features that gives the most homogenous child nodes!!

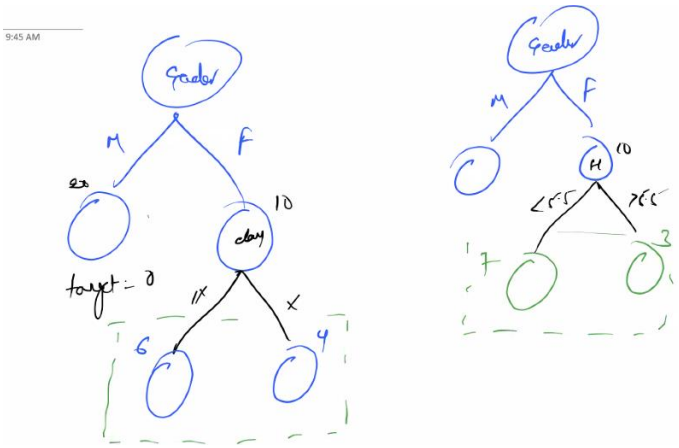
Gini Score

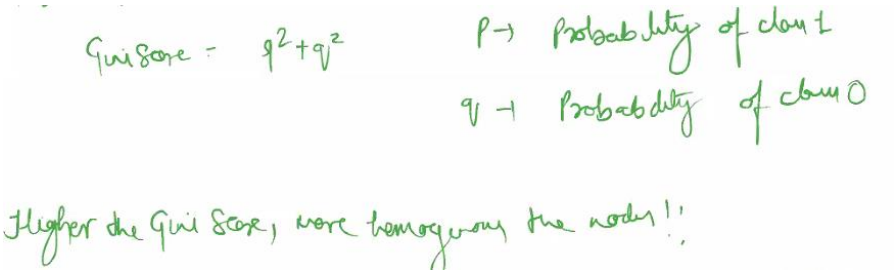
Gini score is used for determining which is the best feature to split on first
An example is given below



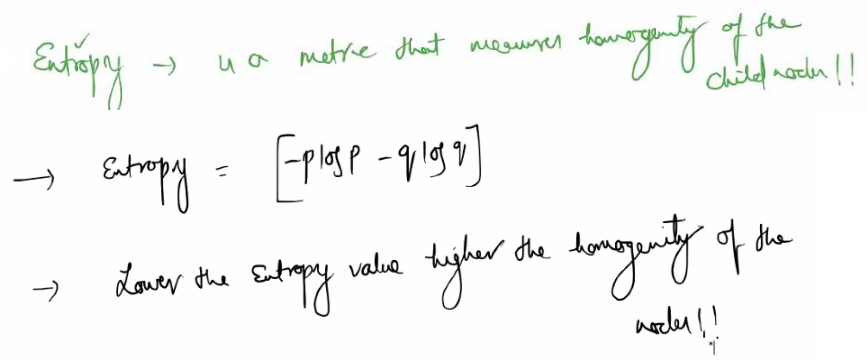
Gene Score for gender who plays cricket	Gene Score for classr who plays cricket	Gene Score for classr who plays cricket
		

Since the gene score is higher for gender, we know where to start the split and then go to the other branch to calculate the gene score and then so on



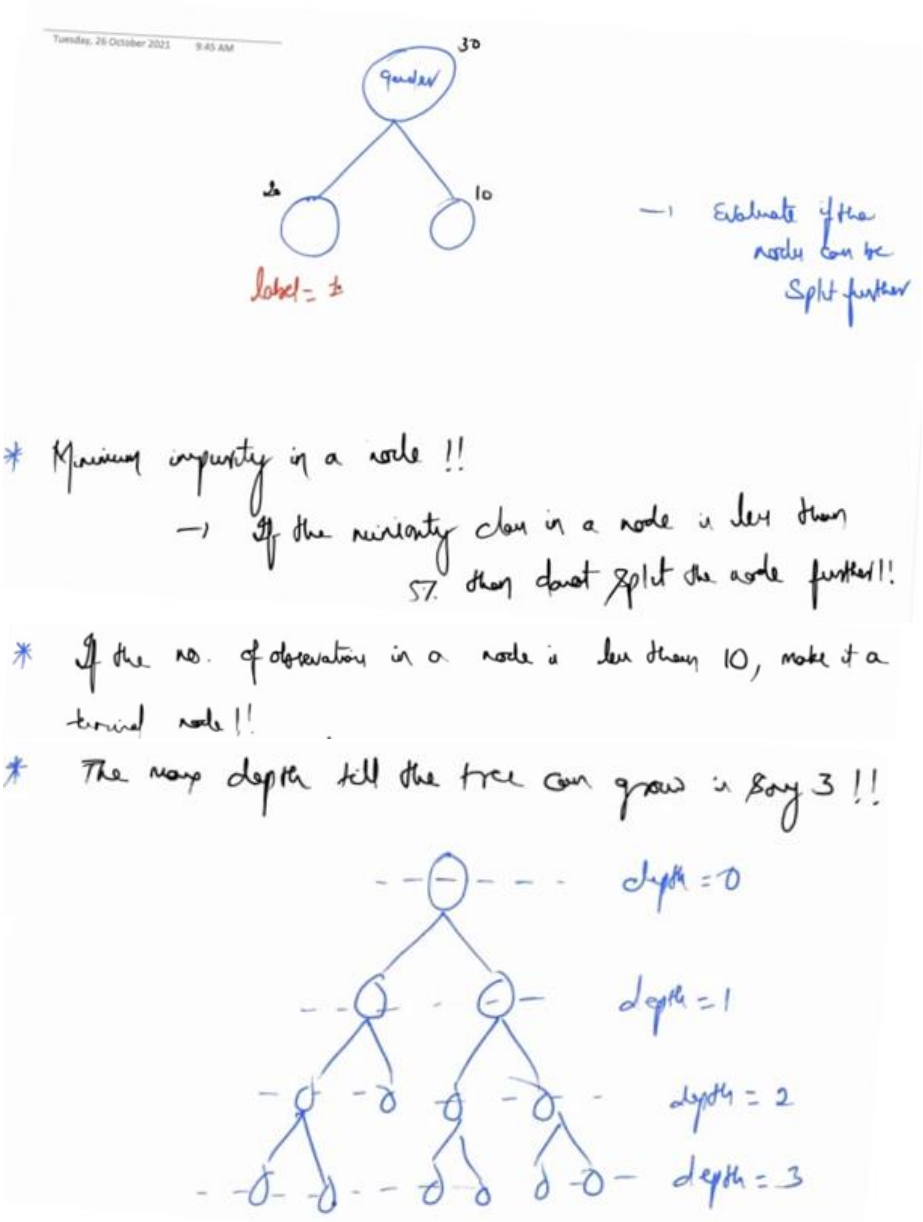


Entropy



Weighted average will calculate in the same way as the Gini score

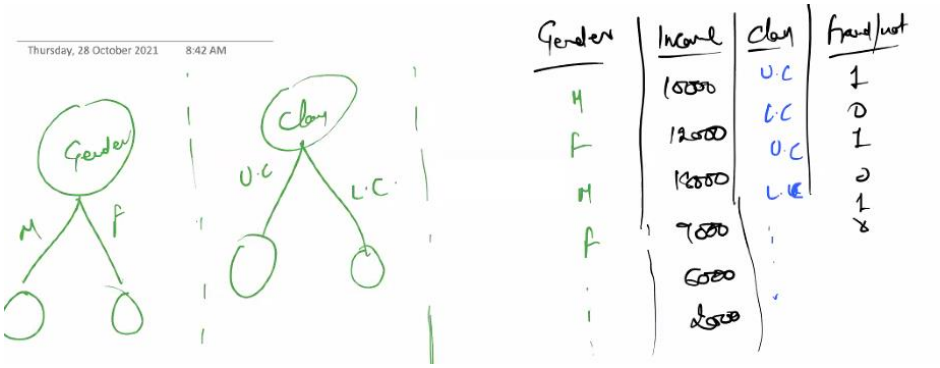
b. Stopping criteria's (Stop Splitting)



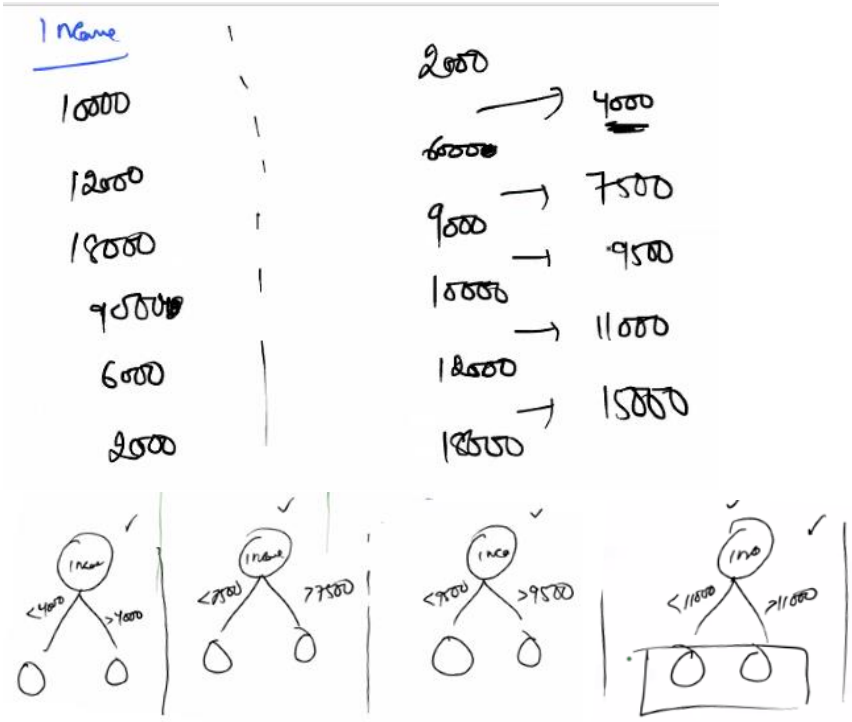
We should specify the depth in such a way that the depth will learn to the optimum.

Stopping Criteria are necessary to prevent Underfitting / overfitting of a tree !!

c. Splitting decision



For Gender and class, it a straight forward split, but for the Income the splitting will be difficult to decide on the range to split.
To split income, the value of income will be first sorted and then median of the values between will be taken for the range

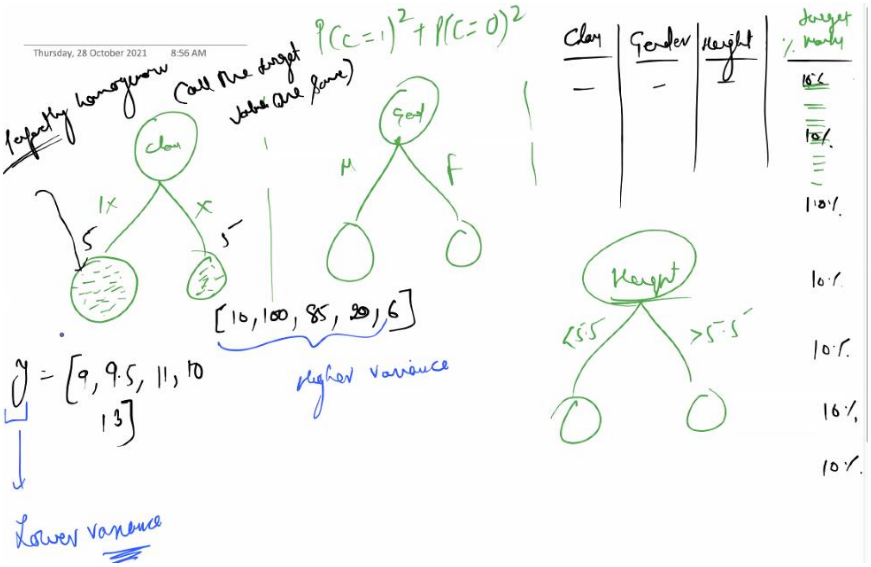


Now we calculate the Gini score for Gender, class and all the different Income ranges and from that we decide on which to split

→ If there are numeric continous columns, the decision tree algo- rithm has to evaluate many possible splits before deciding the best split to split the node!!

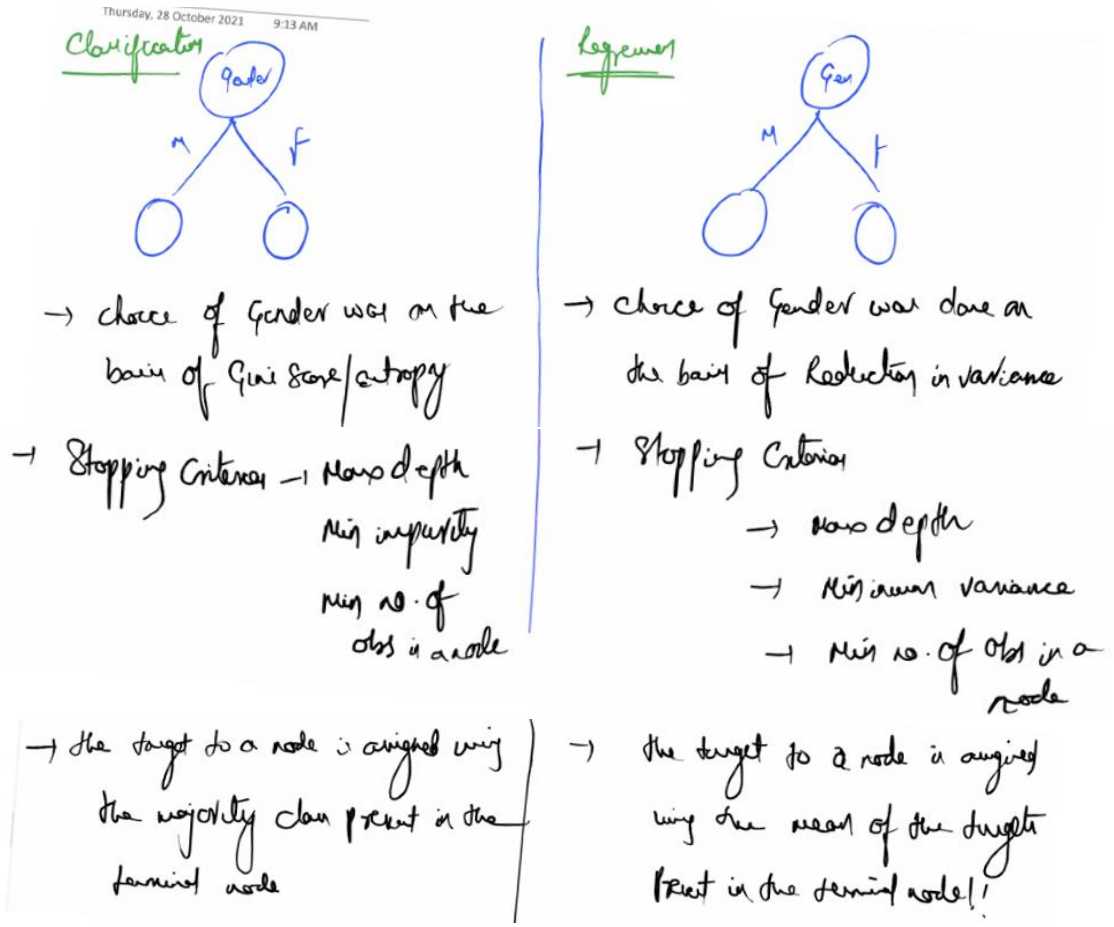
d. How does decision tree work for regression problems?

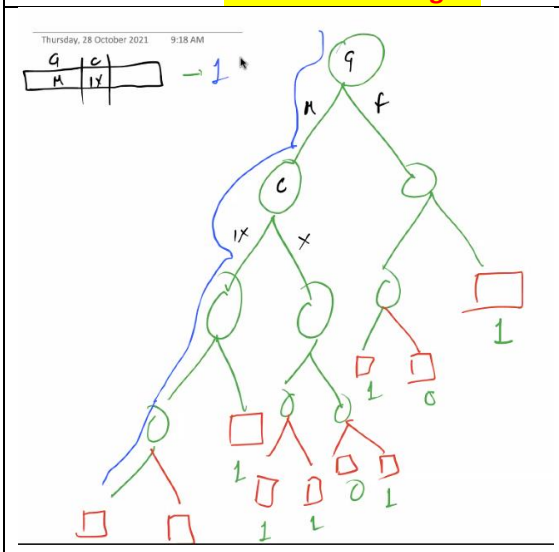
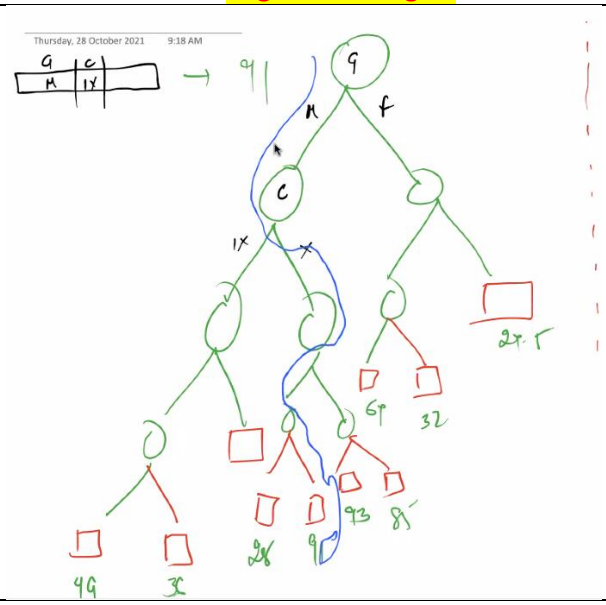
The below is a regression



So in case of a regression problem, variance is the key to find out the way to tree splitting

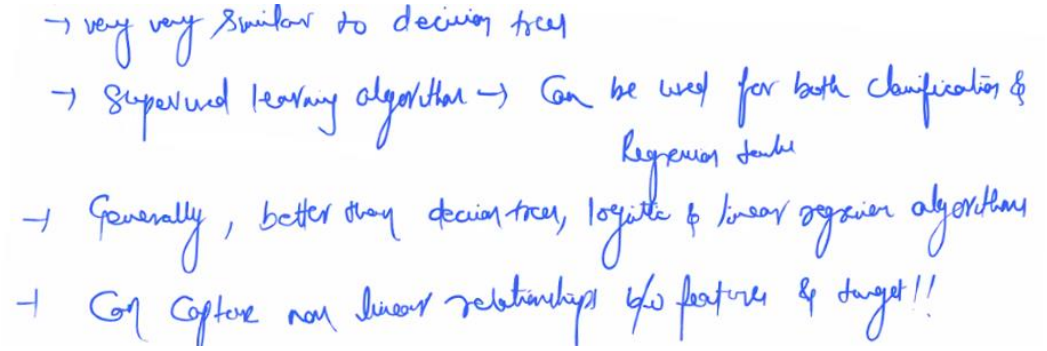
We calculate the weighted average on the variance and then we see which has the lowest weighted average and split on that node



Classification Targets	Regression Targets
	

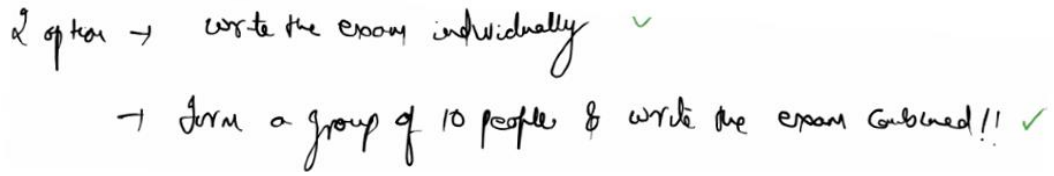
Decision trees are not widely used for regression problems.

A. Random Forest

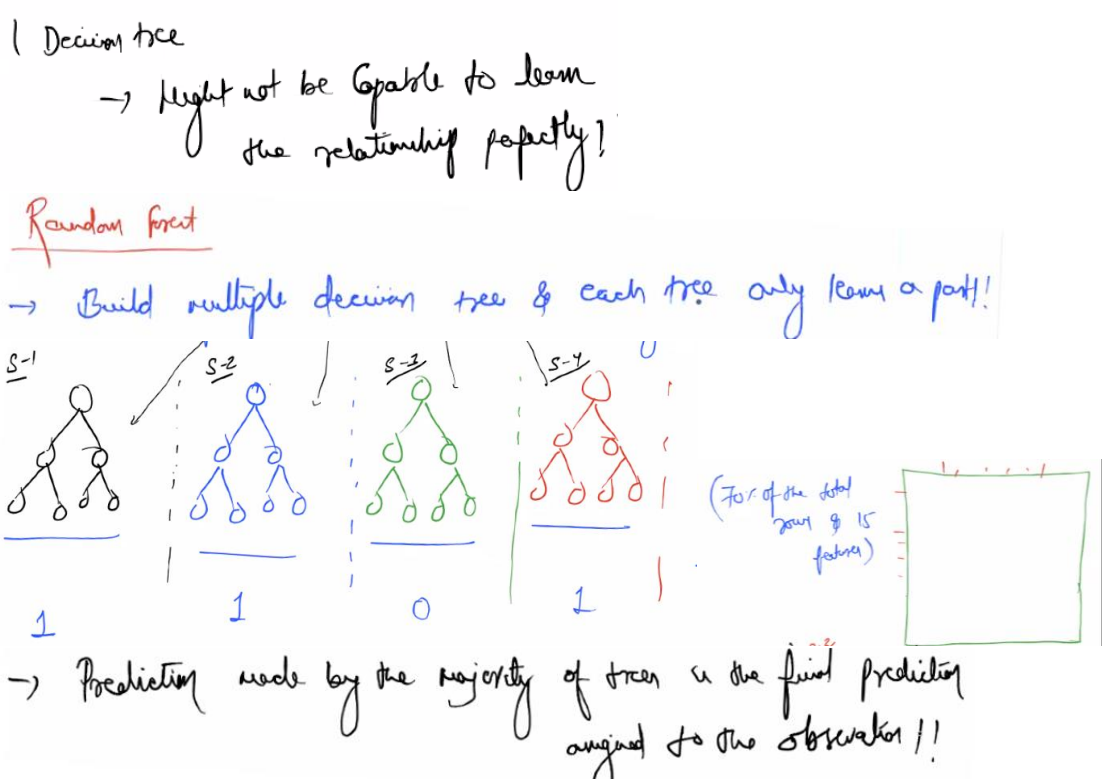


Example given below

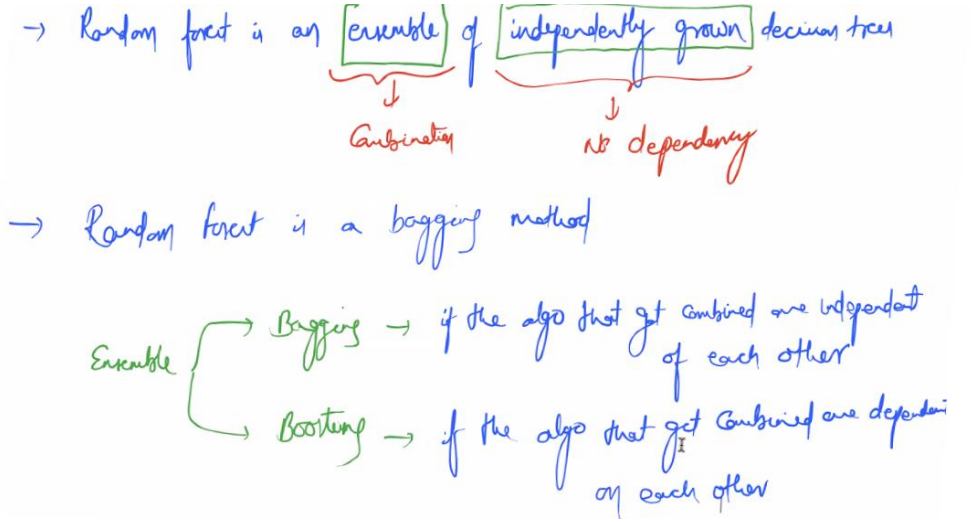
If there is an exam with 100 chapters



The second option is better as the chapters can be split into different individuals to save time



And if it a regression problem, then the average of the prediction will be the result



i. Implementation

a. Classification problems

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

sklearn.tree.DecisionTreeClassifier¶

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*) [source]

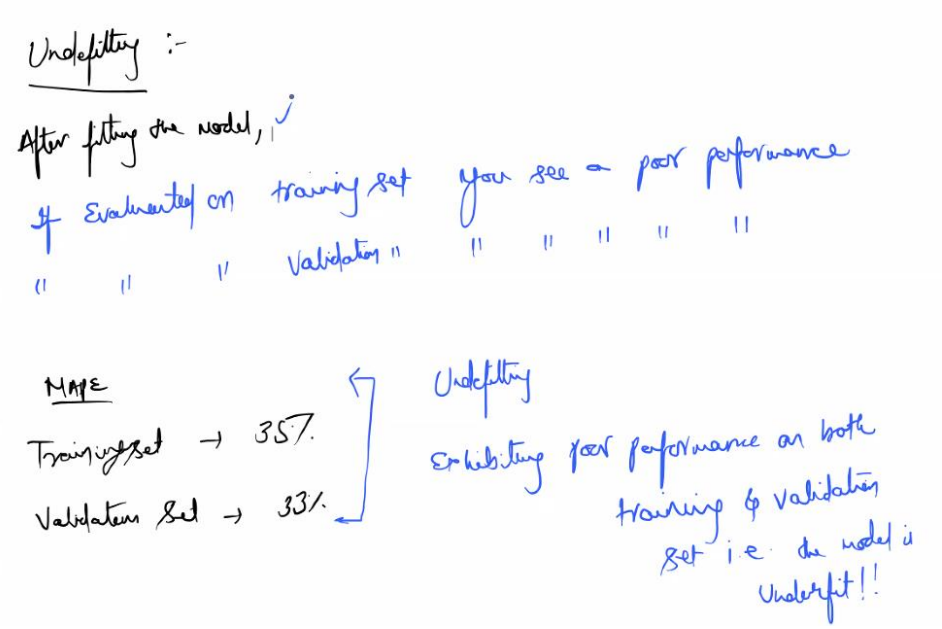
b. Regression problems

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

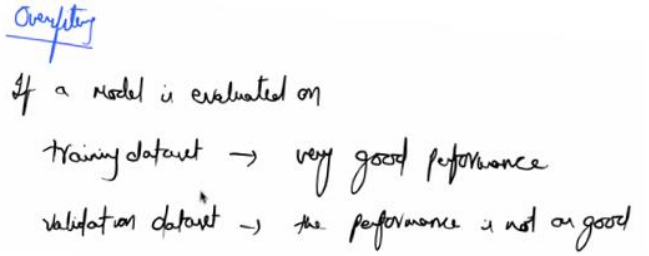
sklearn.tree.DecisionTreeRegressor¶

class sklearn.tree.DecisionTreeRegressor(*, *criterion='squared_error', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0*) [source]

ii. Under fitting



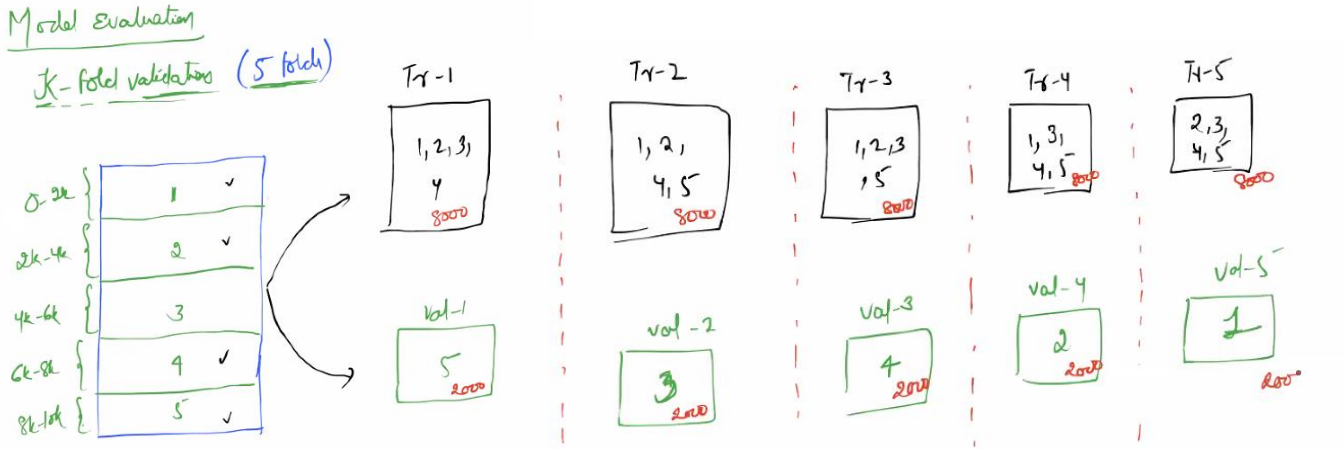
iii. Overfitting



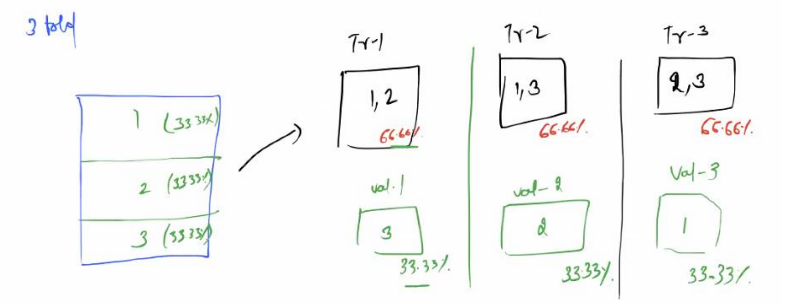
iv. K-Fold Validation for GridSearchCV Function

CV parameter of GridSearchCV the below is the explanation of the cv patamater of the GridSearchCV function

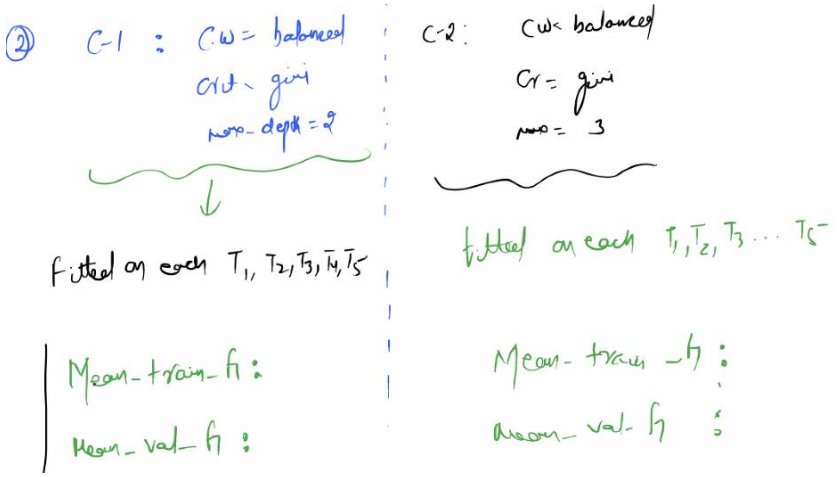
K-Fold is a means of splitting the data into n parts and training it on these n parts.
We are evaluating the parameter combination on different datasets to get a better evaluation
In the below we are splitting the data into 5 parts.



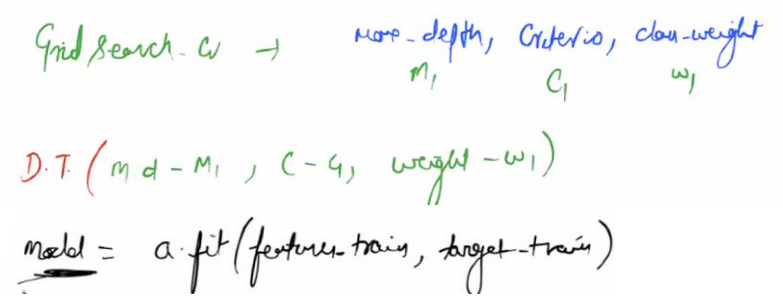
For each we would get a train f1 and validation f1 score



Note: We will use either 5 or 10 folds for this K-folds as we are dividing the data into 2 sets in case of 2-fold. The train data would contain only 50% of the combination which would be an under fit model



Now that we did a GridSearchCV and found the best combination we can now fit a decision tress with the these parameters



v. SVM-Support Vector Machines

→ Support vector Machines (SVM) is a supervised learning algorithm

→ It can be used to solve both classification & regression tasks

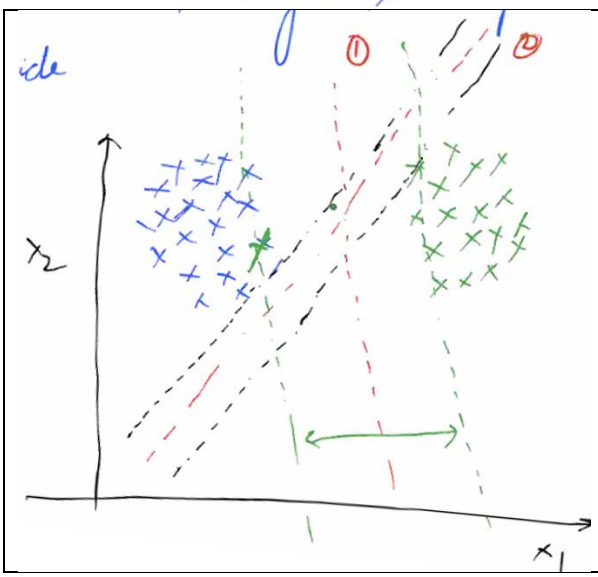
Similar to logistic & linear regression SVM is a linear classifier / regressor

i.e - SVM can only find linear decision boundaries / linear relationship between features & targets !!

→ SVM's are also known as large margin classifiers / large margin regressors !!

→ SVM algorithm will always find a decision boundary that has highest margin !!

Margin :- distance of the decision boundary with closet point on either side



In this example the classifier labeled 1 has the higher margin.

* High margin decision boundaries are more robust to the changes in the data

→ one major advantage of SVM

SVM algorithm is a linear algorithm

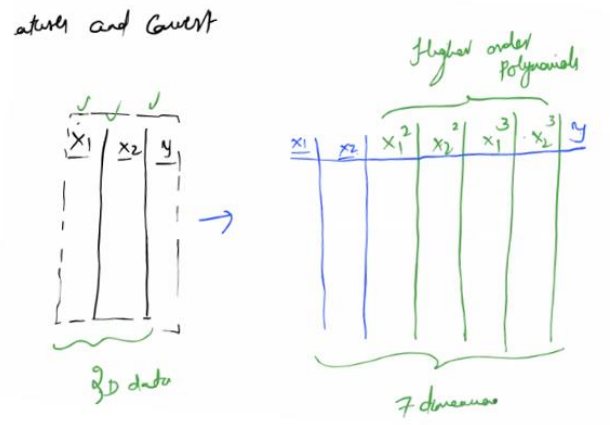
SVM + Kernel → can help to capture non linear decision boundaries

Kernel :- Transformation that can be applied on features and convert the data to a high dimension data

* Polynomial Kernel
* Gaussian Kernel
* Radial Kernel
* Tree based Kernel
* Custom Kernel

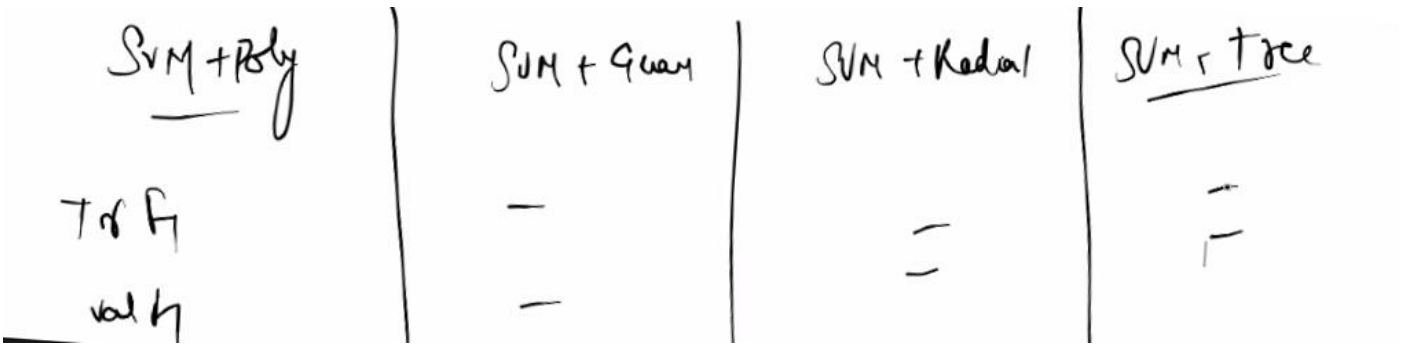
i. Polynomial Kernel

Polynomial kernel with degree 3. What this would do is, it will create additional feature with higher order polynomials



On this newly formed dimension we will apply the SVM.

Note: We have to check which kernel is good for our model by passing it to the gridsearchCV



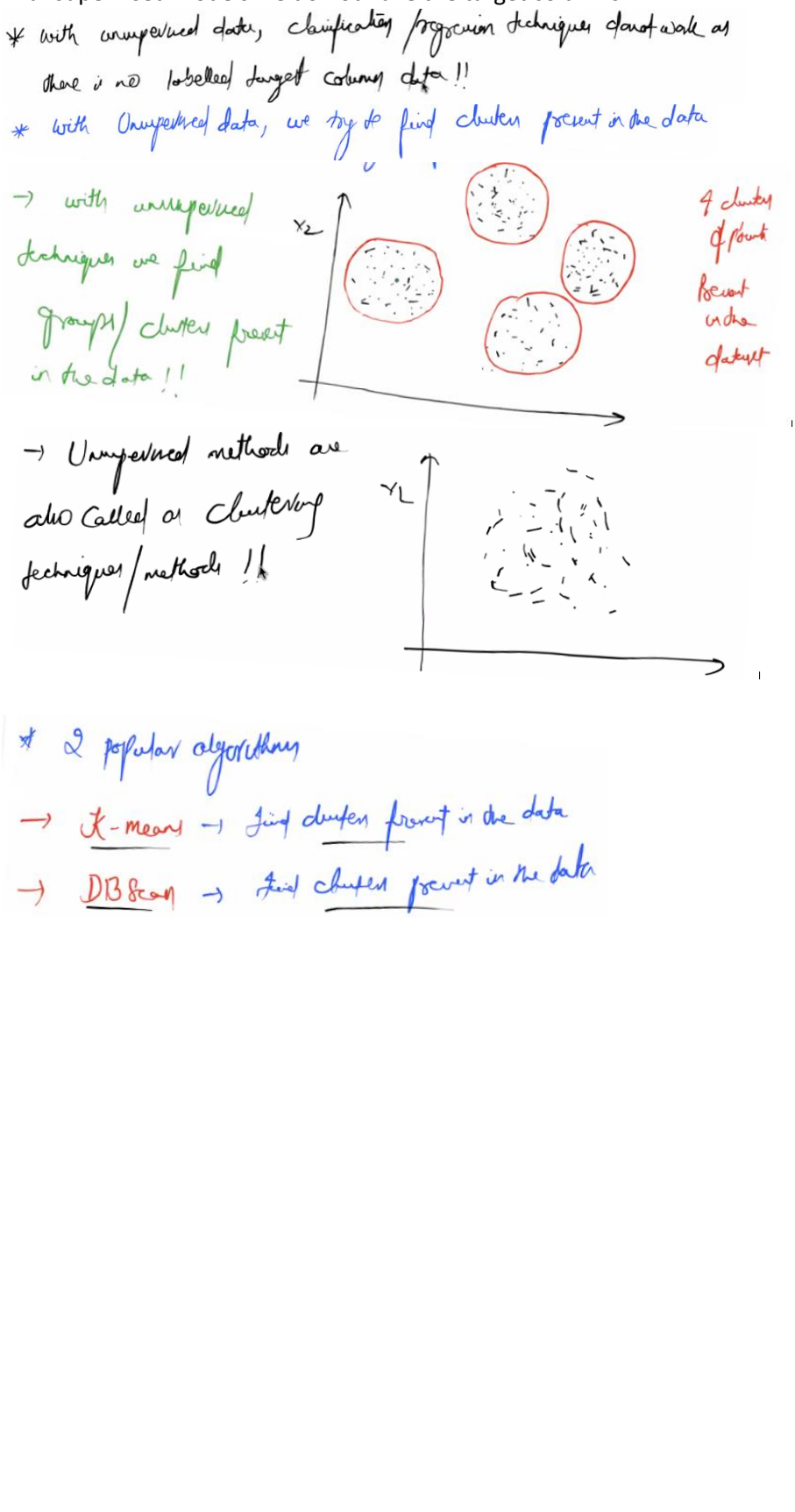
All the earlier steps will be done as the same for all the MLs.

Only the parameter differs below is the way to fit for SVM

```
from sklearn.svm import SVC
clf = SVC()
#clf = LogisticRegression()
params = {'C':[0.1,0.01,0.001,1,2,5,10],
          'kernel' : ['linear', 'rbf', 'poly', 'sigmoid'],
          'degree' : [2,3,4],
          'class_weight':['balanced']}
```

b. UN-SUPERVISED MODELS

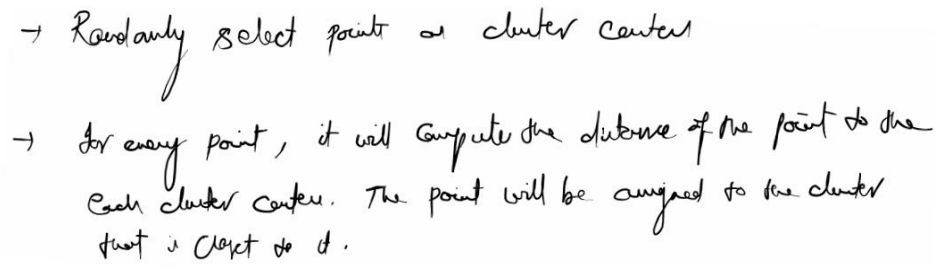
In unsupervised models we do not have the target columns



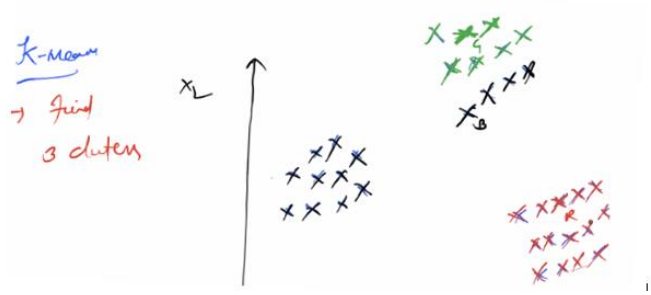
i. K- Means

This is the most popular algorithm
For visual please check this website
http://shabal.in/visuals/kmeans/2.html
http://shabal.in/visuals/kmeans/3.html

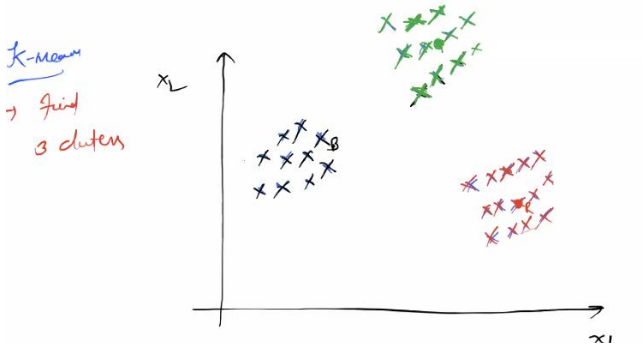
We have to specify the number of clusters. This is a limitation in the K-means.
K means method will find randomly pic up cluster centers in the data
With the clusters centers, it will calculate the distance of the points to these cluster center points and assigns as



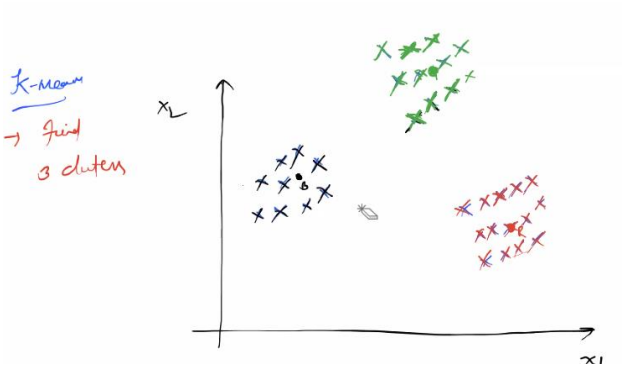
First Iteration



Second Iteration
Now it will find the mid of the cluster it has formed and again assigns its nearest neighbors to its cluster



3rd Iteration
In the 3rd iteration the points will align itself to the cluster center and from here on there will be no more changes in the clusters center points.



i. Learning Steps

Step-0 : Randomly select 'k' points as the cluster centers !!

Step-1 : Assign a cluster label to all other points on the basis of the distance of the point from the cluster center

Step-2 : Recalculate the cluster centers

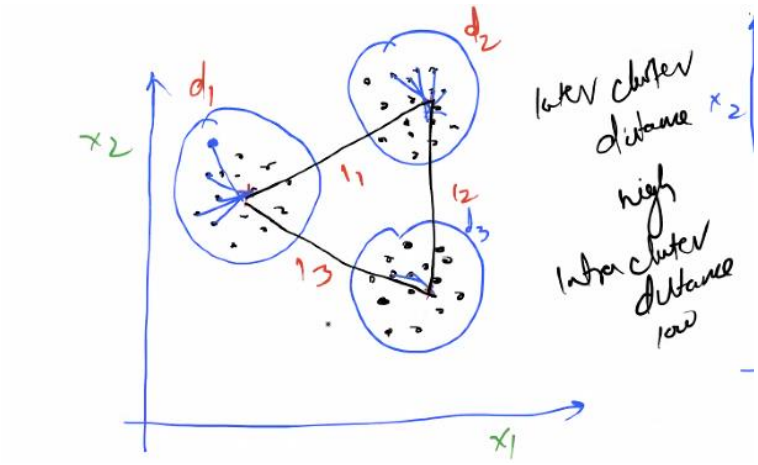
The Step 1 & Step 2 will repeat till the time no points are changing their existing cluster !!

ii. Limitations of K-Means

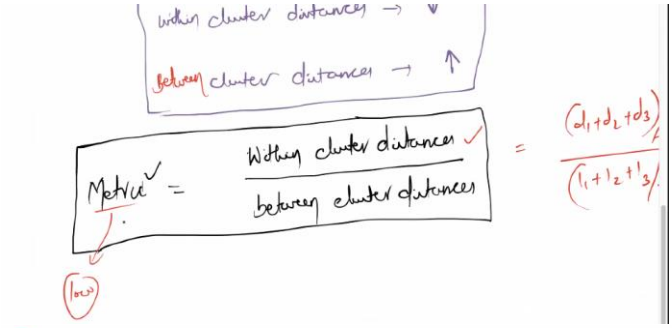
Limitation of K-means

* It needs the number of clusters to find as an input & this is something that we will not know beforehand !!
* K-means algo is sensitive to outliers
  → outlier treatment before applying K-means !!
* K-means algo is impacted by scale difference in the features
  → Scaling before implementing K-means algo

iii. Finding the robustness of a cluster



Silhouette score



Silouhette Score is a metric to measure the goodness/robustness of the clusters formed

$-1 < \text{Silouhette} < 1$

Higher the Silouhette score more the robustness of the clusters.

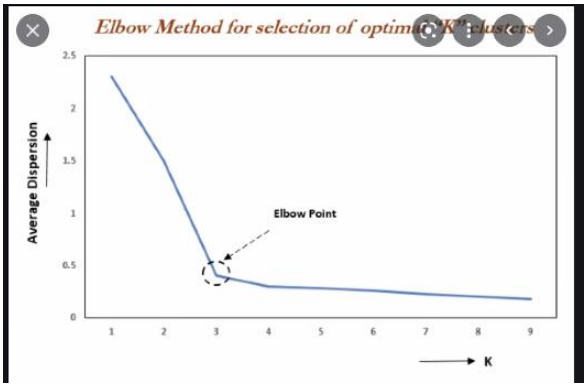
Inertia

Inertia is a metric

Elbow curve → is a plot that helps us to decide on number of clusters!!

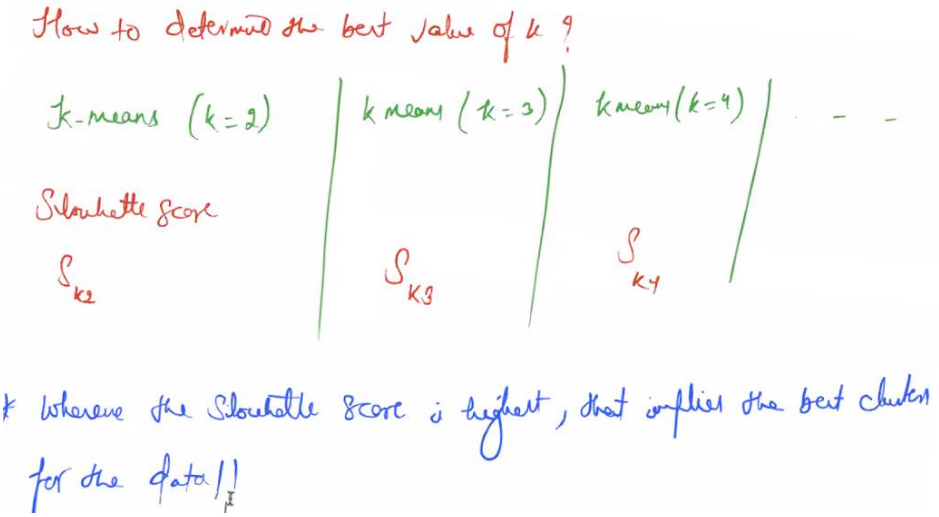
Inertia also uses inter cluster and intra cluster average distances to calculate the number of k's

In the elbow curve we plot the inertia vs the number of clusters and the point where we can see an elbow is the best value of K



iv. How to determine the best value of K

We can never know before hand is the answer ☺
So we fit the k-Means for k values from 2 to 10
Then we calculate the Silhouette score is close to 1, we will choose that cluster k

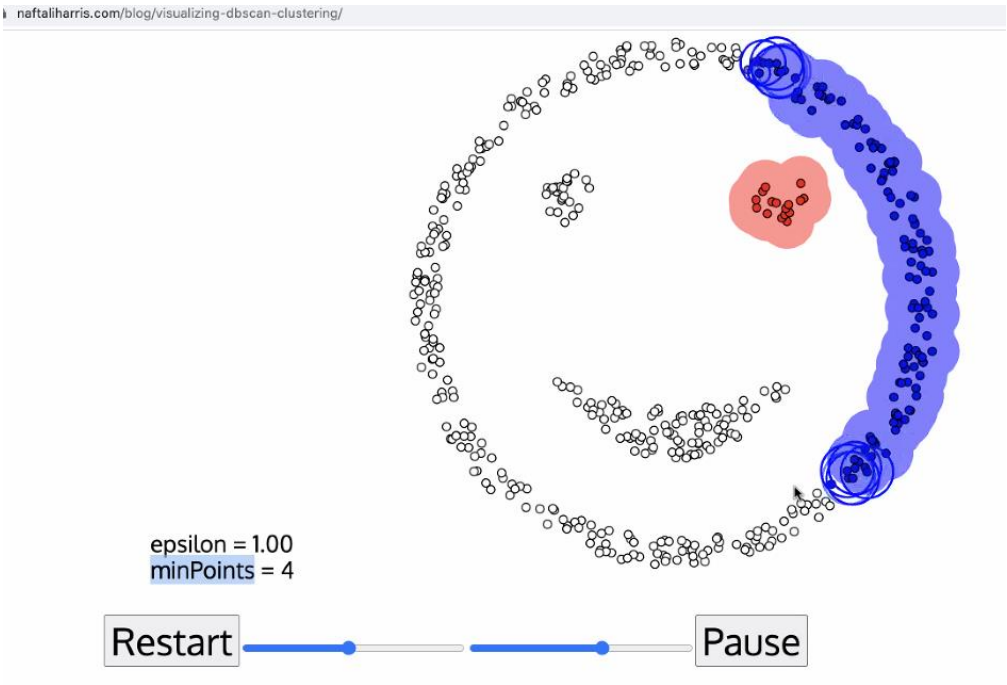


v. Implementation in Python

check it yourself

ii. DB-Scan

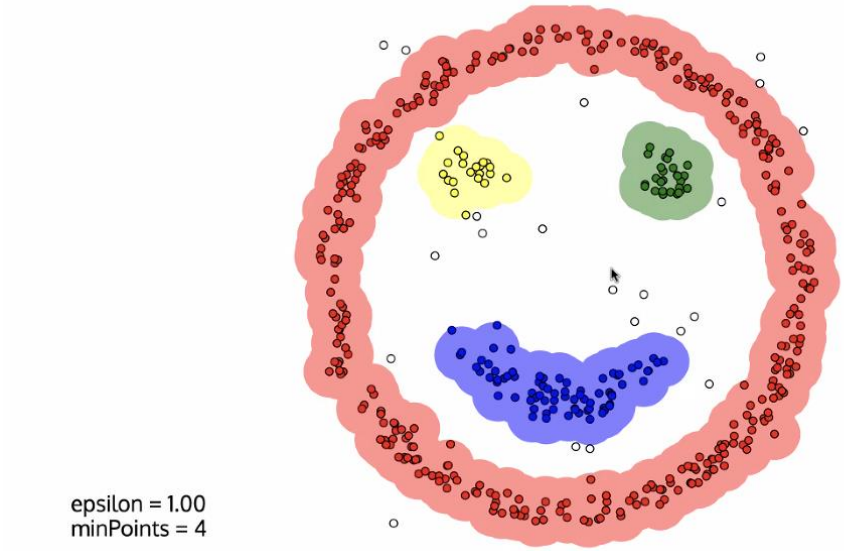
https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/



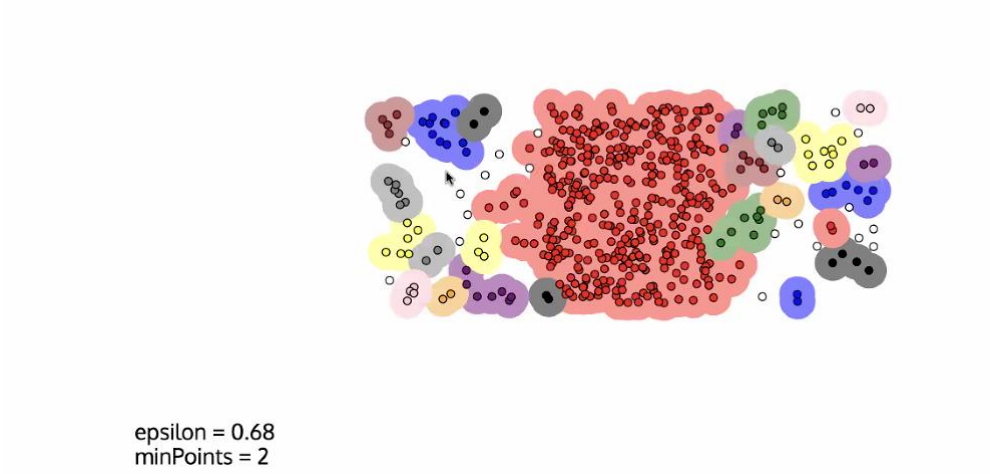
DB scan works on 2 parameters

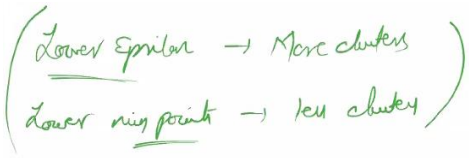
1. Epsilon is the radium of circle of to be created
2. minPoints is the minimum number of points required to form the cluster around the epsilon radius formed

In the below we can see that there are points which can be thought as outliers. For the outlier point when DB scan forms a circle there will be no minPoints of a given number and DBscan thus treats them as outliers.

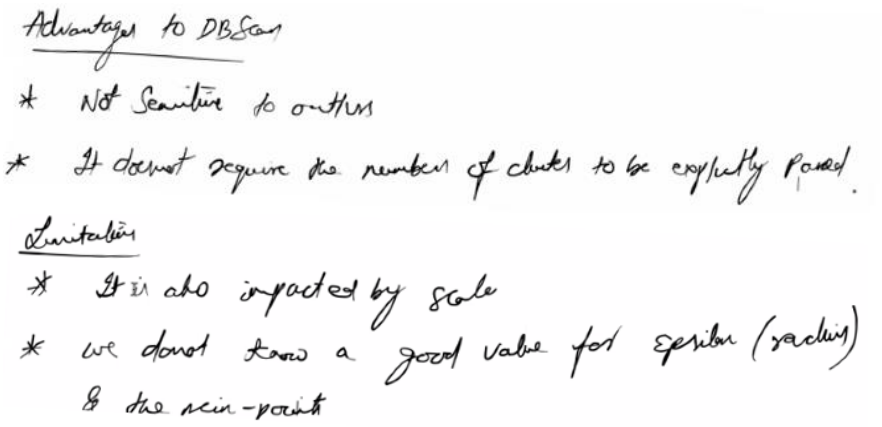


Changing the value of epsilon and minPoints we get differing results. So it is important to set the right values for epsilon and minPoints



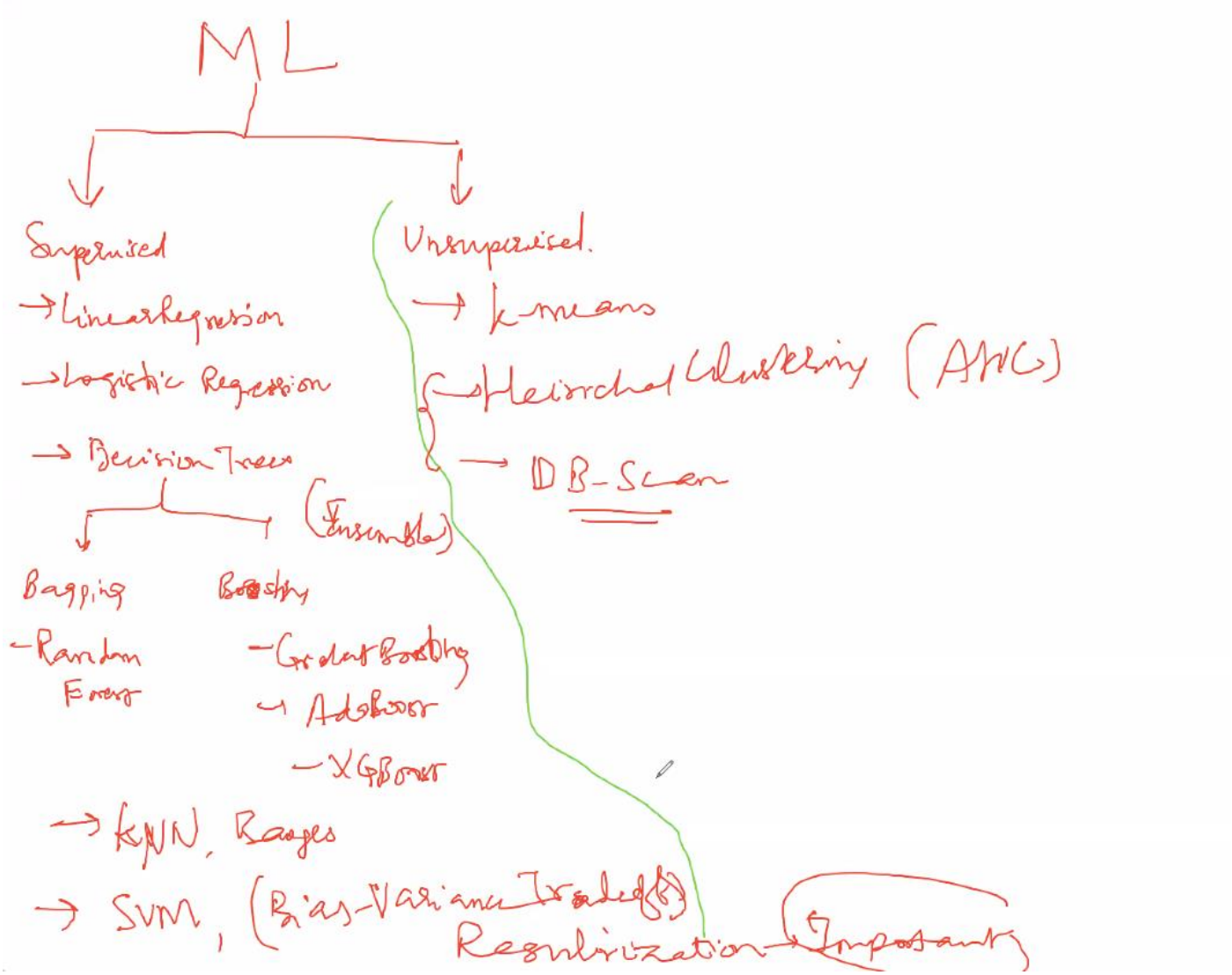


i. Advantages and limitation of DB Scan



iii. Hierarchical Clustering

Check it yourself



Resources

- Towards Data science
- Analytics Vidya
- Kaggle solutions- Machine learning
- Machine learning mastery