

|  |
| --- |
|  |
| SUPERVISED MACHINE LEARNING  [Rohith Somella](http://www.linkedin.com/in/rohithsomella) |
| |  |  |  | | --- | --- | --- | |  |  |  | |

|  |  |
| --- | --- |
| **Content** | **Page No.** |
| [ML Intro](#ML_Introduction) | 1 |
| [Linear Regression](#Linear_recgession) | 3 |
| [Multi Linear Regression](#Multi_linear_regression) | 5 |
| [One Hot Encoding](#One_hot_encoding) | 6 |
| [Standardisation](#standerdisation) | 8 |
| [Normalisation](#Normalising) | 9 |
| [Logistic Regression](#Logistic_regression) | 10 |
| [Confusion Matrix](#confussion_matrix) | 12 |
| [Classification Report](#clssification_report) | 13 |
| [KNN](#knn) | 14 |
| [Multi Collinearity](#multicollinearity) | 15 |
| [Decision Tree](#decision_tree) | 17 |
| [Hyper Parameter Tuning](#Hyper_parameter_tuning) | 21 |
| [Ensemble Learning](#ensemble_learning) | 23 |
| [Random Forest](#random_forest) | 23 |
| [AdaBoost](#AdaBoost) | 25 |
| [Naïve Bayes](#Naive_Bayes) | 27 |

**Machine Learning**

1. **Supervised Machine Learning**

*In this supervised machine Learning we give both Input and output data to train the model and test the model so that we can compare the accuracy of the original values of the output column and the predicted values of the model.*

**Train Test Split**

|  |  |
| --- | --- |
| Import | from sklearn.model\_selection import train\_test\_split |
| Assigning | X\_train,X\_test,y\_train,y\_test=  train\_test\_split  (X,y,train\_size=0.70,random\_state=0) |

**Regression: -** here the values are continuous which means has no fixed category for the values

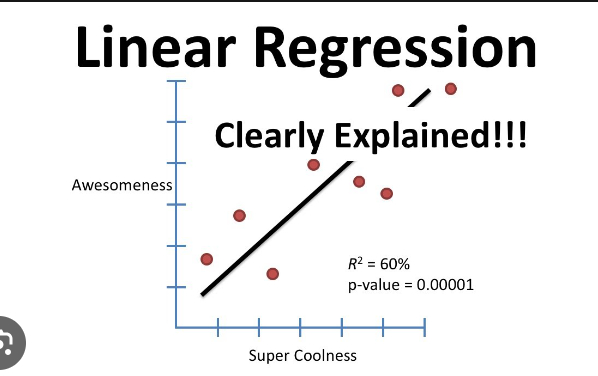
Ex: - price, age etc…

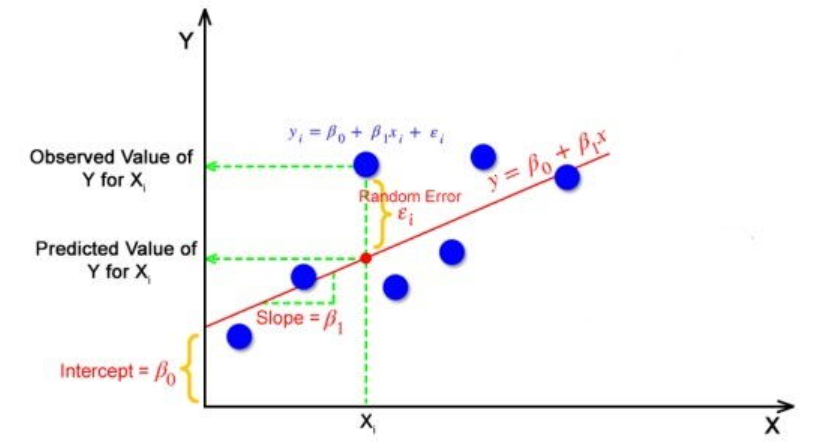
**Classification: -** here the values are discreet which means has a fixed category for the values

Ex: - yes/no, true/false etc…

**Linear Regression**

|  |  |
| --- | --- |
| Import | from sklearn.linear\_model import LinearRegression |
| Assigning variable as Model | model1=LinearRegression() |
| fit 🡪Train the Data | model1.fit(X\_train,y\_train) |
| test | y\_pred=model1.predict(X\_test) |
| Coefficient | model1.coef\_ |
| Intercept | model1.intercept\_ |
| Import r2 score  only for Regression type output data | from sklearn.metrics import r2\_score |
| r2 score | r2\_score(y\_test,y\_pred) |
| Import MSE  loss or error | from sklearn.metrics import mean\_squared\_error |
| MSE | mean\_squared\_error(y\_test,y\_pred) |
| RMSE | RMSE=np.sqrt(MSE) |





Fit: - it is used to Train the data for the Machine Learning model, here we give only training data which include both input and output data.

r2\_score: - it is used for regression type output data and used to find the accuracy between the y\_test (output test data) and y\_pred (output predicted data).

(MSE) Mean Squared Error: - which means How far the points from the predicted line in the scatter plot is called Error (distance between the test values to the predicted values in the scatter plot) called as error. (y\_test – y\_pred)2

**MSE: Mean Square Error**

**Loss Function:** (Actual value – predicted value)2

**MSE = *(y- ŷ)2*** 🡪 (Actual – Predicted)2

**Cost Function:** Sum(Actual – Predicted)2

**MSE =**

The loss function measures the discrepancy between predictions and true values for individual data points, while the cost function provides an overall evaluation of the model's performance across the entire dataset. Minimizing the cost function helps optimize the model's parameters to improve its generalization capabilities.

In many cases, the terms "loss function" and "cost function" are used interchangeably, especially in the context of optimization algorithms, where the objective is to minimize the discrepancy between predictions and true values.

**RMSE:** Root Mean Squared Error: - it is a square root of MSE value, and it is the most considering value in real time.

**Formula: - (*y = mx + c)*** can be written as

M = coefficient

X = value (magnitude)

Mx = vector

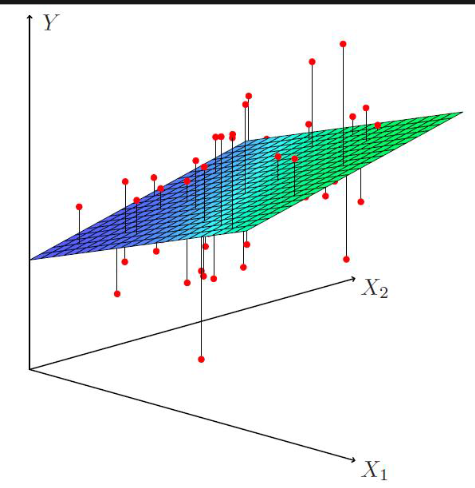
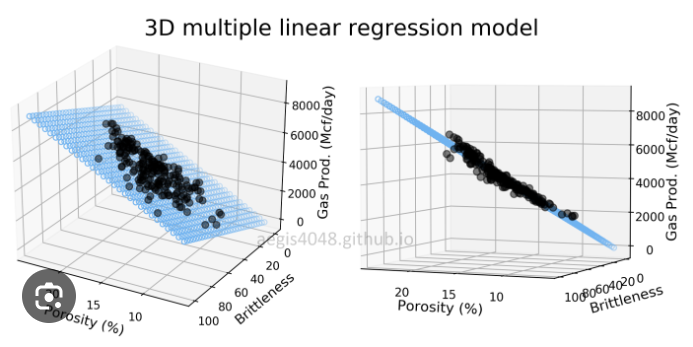
C = Intercept

**y = B0 + B1X**

y = Intercept + coefficient \* X

**Multi Linear Regression**

|  |  |
| --- | --- |
| X – variable | X=dataset.drop([‘output\_column’], axis = 1) |
| y – variable | y = dataset.loc[‘output\_column’] |
| plotting the regression line | plt.scatter(X\_test,y\_test)  plt.plot(X\_test,y\_pred, color = 'red') |



X – variable: - in this variable contains the Input columns or independent values.

Y – variable: - in this variable contains the output column or dependent variable.

y = predicted values

B0: - Intercept or bias space from point 0 in line plot

Bn: - coefficient or weight or cross line points in the line plot

Train\_size = 0.75: - which means the training data contains 75%, and train data is should be more then test data.

Test\_size = 0.25: - which means the test data contains 25%, and it should be less then train data and we can give any one of this both in the syntax.

Random\_state = 0: - here it means values should be selected randomly but it has a patten and pattern code is 0 here, we can take any code here I choose 0.

**Formula: -**

y = B01 + B1X1 + B2X2 + … + Bn Xn

y = Intercept + coefficient \* X1 + coefficient \* X2 + … + coefficient(n) \* Xn

**while in fit function the model is trained based on this formula**

**One Hot Encoding**

|  |  |
| --- | --- |
| Creating dummies | data\_dummy = pd.get\_dummies(Dataset) |
| auto removing 1st column dummies | data\_dummy2 = pd.get\_dummies(Dataset, drop\_first = True) |
| assigning values for Ordinal column | Label = {"Ex":4,"Gd":3,"TA":2, "Fa":1} |
| replacing numerical values in the column | Data['col\_name\_Oe'] = Data['col\_name']. map (Label) |
| shortcut with numpy | Data [' output\_col''] = np.where(Dataset['output\_col'] == 'yes',1,0) |

**Categorical: -**

1. **Nominal: -** In this there will be unique and equally Important categories

Male/Female, yes.no, section A, B, C, and D

1. **Ordinal: -** In this, the values will be in hieratical, here Importance vary to each other

Excellent - 10

Very Good - 8

Good - 6

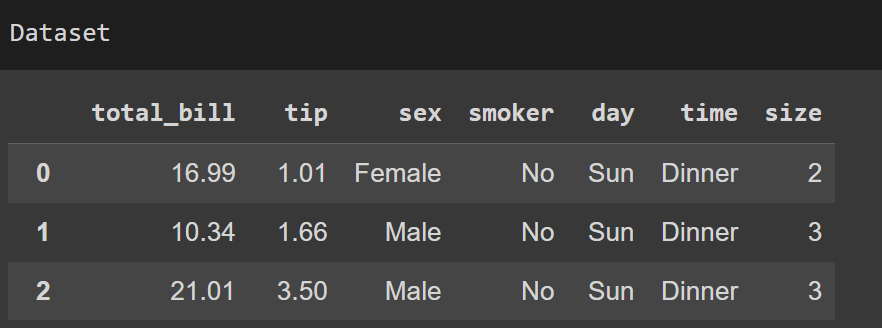
Average - 5

Poor - 4

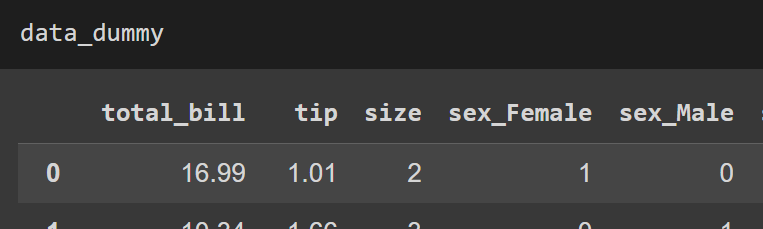
Fail – 3, 2, 1, 0

Encoding: - It is a part of EDA, and this method is used to convert the categorical column into respective Numerical column.

**One Hot encoding: -** this method is used to convert the Nominal column into numerical columns, by creating the dummy variable

****Ex: - if we have male and female in gender column so it converts into numerical columns by creating the dummy variables.

**Nominal: - 🡪**

Here in original dataset, we have the column sex which have categorical values Male and Female but when we apply dummies, we can see dummy variables or columns are created with name of sex\_Female and sex\_Male columns. In sex\_Male column “1” means male and “0” means female which means not male.

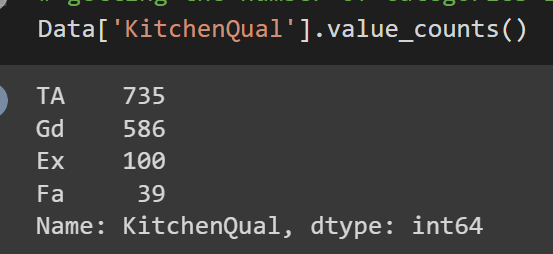
Note: - get dummies () only for Nominal column but Not for Ordinal column.

Shortcut method for **Nominal Encoding**

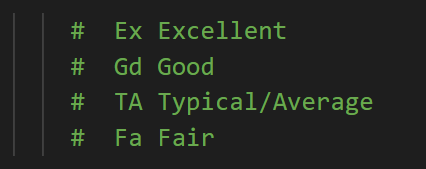
Dataset[output\_col'] = np.where(Dataset['output\_col'] == 'yes',1,0)

**Ordinal: - 🡪**

In this Ordinal column the values in the hierarches and those information and values we need to assign were given in the description of the dataset.

****

So, we just need to assign the numerical values by using dictionary to the Ordinal column. Because Machine Learning model only takes numerical values but not object type of data.



**Standard Scaling**

|  |  |
| --- | --- |
| Import | from sklearn.preprocessing import StandardScaler |
| model | sc = StandardScaler() |
| fit | sc.fit(X\_train) |
| transform | X\_train\_sc = sc.transform(X\_train)  X\_test\_sc = sc.transform(X\_test) |

**Standard Scalar: -**

**Basic Definition:** In standardization each. Value ranges from “0” zero of its mean from “1” one of standard deviation.

**Detail explanation:** Standard means fixed for all the values, here it means fixed importance for all the values. Scalar means measurement of the value Ex: kgs and grams, kilometres, and meters etc…

So, Standard Scalar means making all the values to maintain equal Importance in spite of any units or scale.

Here In standardisation, we use Standard scaling

Here we have the Input in different unit formats and degrees, range (scale) Ex: - 700gm, 5kgs. Here ML consider 700gm is more Importance because it has more value but we know that 700gm is less then 5kgs, but ML don’t understand that units it just looks at the numerical values.

Here In syntax, we give only X\_train and X\_test data for training in standard scaling because here X means Input data of the Training data.

Why we don’t give y\_train or y\_test because, here fit means training so we give only training data and in training data we have to convert all the values into float values. so that it will get the equal importance by ML Algorithm. But in y\_train there is no input data, so we don’t need to change or standardise the values, so we don’t need to add y\_train data to fit.

sc.fit(X\_train)

Here the data is trained by giving in fit

Next transform model: - here we give both X\_train, y\_train for the model

To standardise the values as mentioned above, and new values come in array format.

Formula for Standard Scaling: -

**New value = old value – mean / standard deviation**

X\_train\_sc=sc.transform(X\_train)

X\_test\_sc=sc.transform(X\_test)

Transformed the data to make all the values to get Equal Importance so that data is Standardised. And stored in new variables for safe side. X\_train and X\_test is normalized now and ready for prediction.

After standardization, the values we get are contains or holds. mean as zero and standard deviation as one for each value. In the output array.

**Normalisation:**

Formula = X Normalized =

**New value = Old value – min value / max value – min value**

Here min and max values are within the dataset for each individual columns take their own column values.

Library: Min-Max Scaler()

Each value ranges from zero to 1, after normalization

**Use case of both Standardisation and Normalisation:**

Standardization mostly used for machine learning algorithms and normalisation is used for CNN image processing. So, in this deep learning techniques like CNN and ANN Model will understand easily if the values are scaled from zero to one. So, it is easy for the model to understand and arrange the weights accordingly. If the values are from 0 to 1 range.

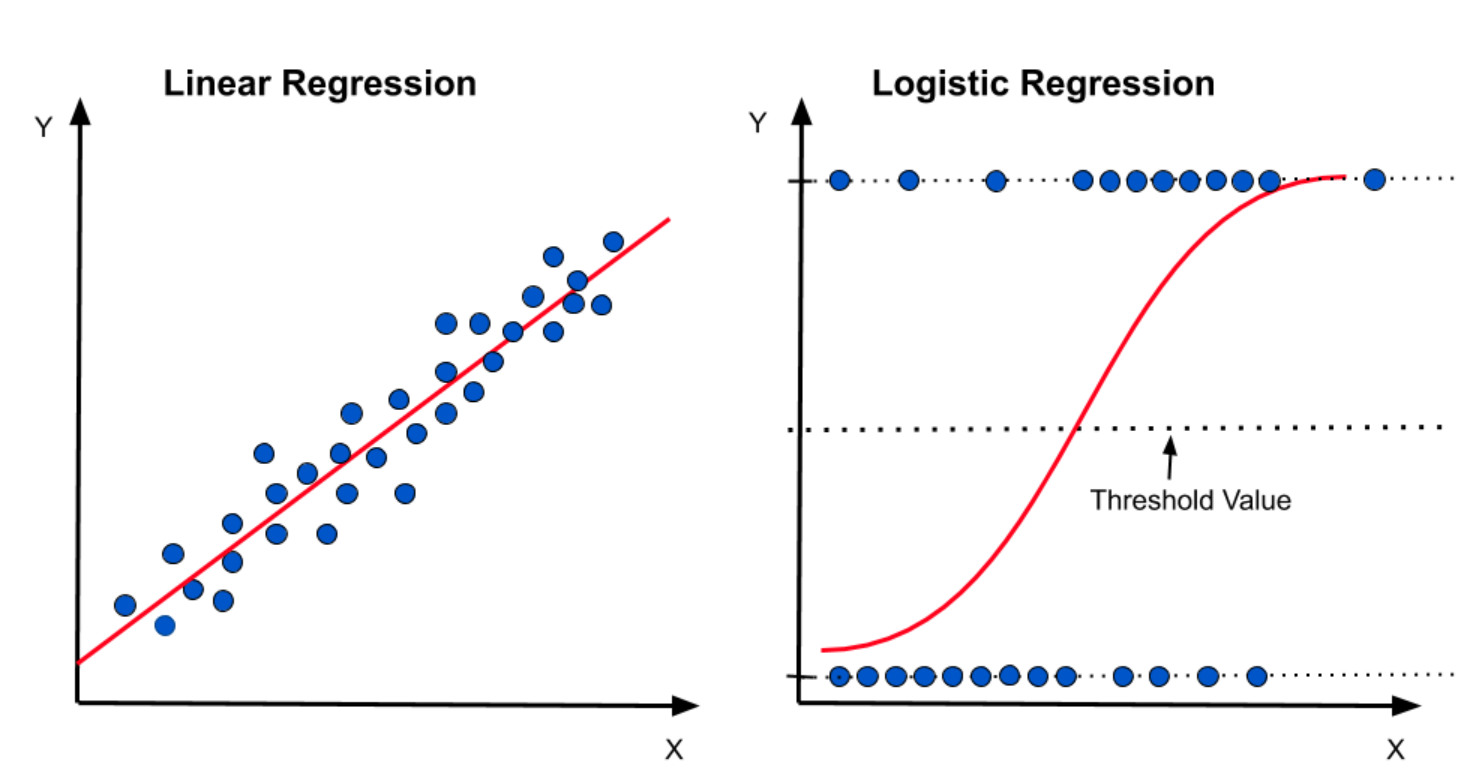
Standardization used where Euclidean distance or A gradient descent Takes place in the algorithm.

So, in some scenarios, like logistic regression, linear regression, K nearest neighbour, K means clustering. So in this we use gradient descent, or either we use Euclidean distance. So, in these cases, we need to scale down the values.

But in some cases, like creating a tree like decision tree, random forest, AdaBoost, there we are already creating a tree for each node for each value. So, we don't have to scale down the values we can simply go for model building without scale down the values.

**Logistic Regression**

|  |  |
| --- | --- |
| Import | from sklearn.linear\_model import LogisticRegression |
| assign calling with object | model1 = LogisticRegression() |
| fit | model1.fit(X\_train,y\_train) |
| predict | y\_pred = model1.predict(X\_test) |
| Import accuracy score | from sklearn.metrics import accuracy\_score |
| check accuracy | accuracy\_score(y\_test,y\_pred) |
| Import Confusion Matrix | from sklearn.metrics import confusion\_matrix |
| Confusion Matrix | confusion\_matrix(y\_test,y\_pred) |
| see heat map for confusion matrix | sns.heatmap(name\_of\_matrix, annot=True) |
| Import Classification Report | from sklearn.metrics import classification\_report |
| Classification Report | (classification\_report  (y\_test,y\_pred)) |



**Why it is called Logistic Regression:** even though it works for classification problem

* because its underlying technique is quite the same as Linear Regression.

Logistic Regression works on the categorical data which means both Nominal and Ordinal type data. Simply it can used in binary classification problem.

This Algorithm is Inspired from Sigmoid Function

Steps: - 1st It use the Multi Linear Regression, 2nd it applies Sigmoid Function to get (0 or 1)

1. **Linear Regression**

y = B01 + B1X1 + B2X2 + … + Bn Xn

1. **Sigmoid Function**

**Z = 1 / 1+ e-y** 🡪 y is from above formula, Z is predicted value

If Z >= 0.5 = 1

If Z <= 0.5 = 0

If Z = 0.5 = 0 🡪 Mostly preferred 🡪 it takes the threshold value ceil (0.5) and gives 0 as the result.

* + - If Z >= 0.5, you predict the positive class (1).
    - If Z < 0.5, you predict the negative class (0).

The value of Z being exactly 0.5 is an edge case and would generally be handled according to your particular use case. For many binary classifiers, when Z is exactly 0.5, it may be treated as the positive class (1).

For Yes and No binary classification yes is positive so Alternatively, you may decide to treat it as the negative class (0). The choice of how to handle the threshold value depends on the specific application and context.

Below are the ways to consider the value 0.5.

1. We assign Threshold Value to surpass this issue
2. By collecting more data
3. By adding weighted classes
4. By taking randomly

**Question: 1.** What is the difference between the Regression and Classification Problem? Give the example for each and also speak about the ML Algorithm respectively?

**Answer: -**

Regression and Classification are part of supervised learning problems in Machine Learning.

**Regression**

Regression problems used to predict the output or dependent variable based on independent variable called input values.

we mention input value columns in X variable output value column in y variable.

Example: - output values in different patterns have no common to each other like Price, Temperature etc

ML: - we use Linear and Multilinear Regression methods for this Problems. for accuracy we use r2\_score.

**Classification**

in these problems used to predict the values in classification values by taking the independent values and predicting the dependent values, some only works on binary classification problems but not on multi classification problems like Logistic regression

Examples: -output values are in categorical format like gender have male and female, yes/no, in/out etc...

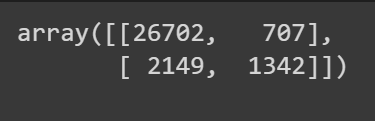
For multiclassification:

Decision tree, random forest and SVM

ML: - for this classification type we Use Logistic Regression methods, here we use accuracy\_score for accuracy and we do check with classification report

**Confusion Matrix: -**

Confusion Matrix is used to evaluate the accuracy of a classifier by comparing the actual values with the predicted values.

A confusion matrix typically has four entries: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN).

True: - same in both test and predicted data

False: - Not same in both test and predicted data

Positive: - 1 one (output catogorical value 1)

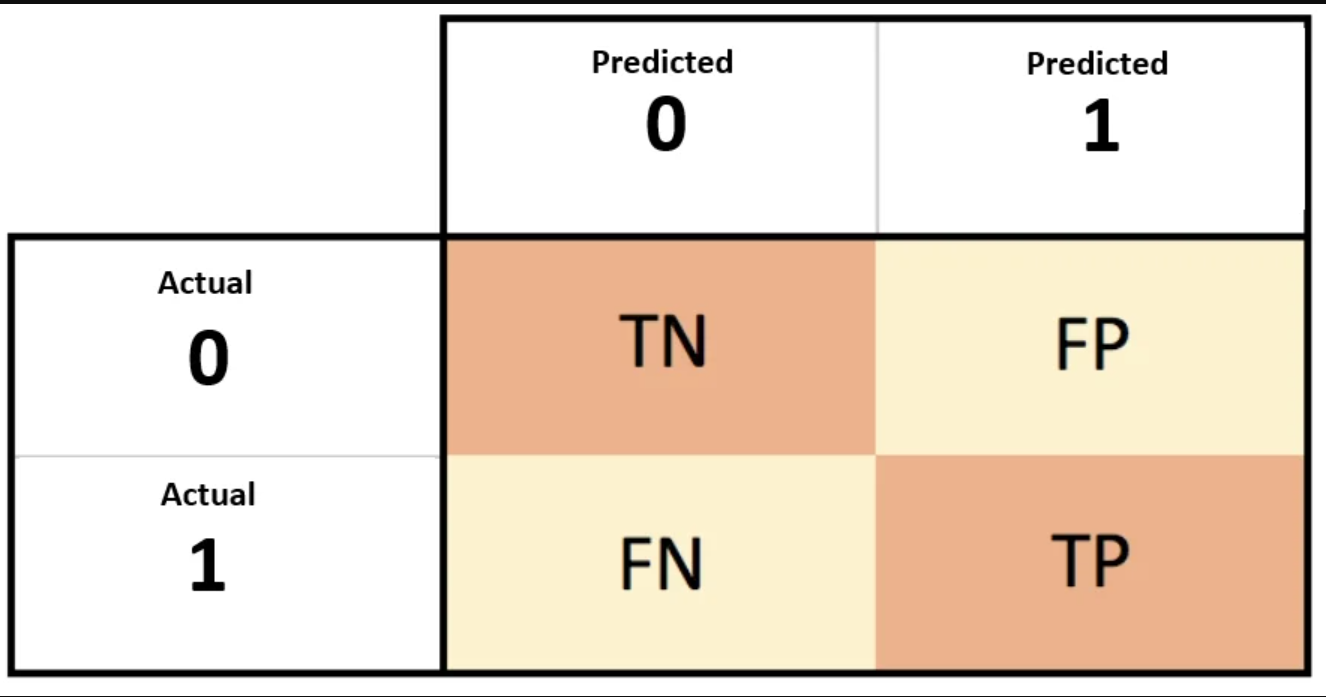
Negative: - 0 zero (output catogorical value 0)

TP: - both test and predicted values are same, and it is 1 (1,1)

FP: - both test and predicted values are Not same, and it is (0,1)

TN: - both test and predicted values are same, and it is 1 (1,0)

FN: - both test and predicted values are Not same, and it is (0,0)



|  |  |  |
| --- | --- | --- |
| CONFUSION MATRIX  **Total Values** | **Predicted**  **0** | **Predicted**  **1** |
| **Actual** 100 **0** | 90 TN | 10 FP |
| **Actual** 100 **1** | 11 FN | 89 TP |

|  |  |
| --- | --- |
| **Accuracy:** | (TP + TN) / (TP + TN + FP + FN) |
| **Precision:** | TP / (TP + FP) |
| **Recall (Sensitivity or True Positive Rate):** | TP / (TP + FN) |
| **Specificity (True Negative Rate):** | TN / (TN + FP) |
| **F1-Score:** | 2 \* (Precision \* Recall) / (Precision + Recall) |
| **False Positive Rate (FPR):** | FP / (FP + TN) |
| **False Negative Rate (FNR):** | FN / (FN + TP) |

**Classifier: - Classification Report**

A classification report is a performance evaluation metric that provides a summary of the performance of a classification model on a test dataset. It is commonly used in machine learning and data mining to evaluate the quality of a classifier's predictions.

The classification report typically includes the following metrics:

**Precision:** the ratio of true positives to the sum of true positives and false positives, i.e., how many of the predicted positive cases are actually positive.

**Recall:** the ratio of true positives to the sum of true positives and false negatives, i.e., how many of the actual positive cases are correctly identified by the model.

**F1-score:** the harmonic-mean of precision and recall, i.e., a single metric that combines both precision and recall.

**Support:** the number of occurrences of each class in the test dataset.

The classification report provides a concise summary of the model's performance, making it easier to compare different models and make informed decisions about which model to choose.

**Hormonic Mean:** number of observations, or entries in the series, by the reciprocal of each number in the series.

HM = 2ab ⁄ a+b

**Athematic mean:** sum of observations by number of (count of) observations

AM = a+b/count(ab); so, count is 2

AM = a+b/2

**KNN (K Nearest Neighbour)**

|  |  |
| --- | --- |
| Import | from sklearn.neighbors import KNeighborsClassifier |
| assign | model1 = KNeighborsClassifier() |
| assigning k value | model=NearestNeighbors(n\_neighbors=1) |
| fit | model1.fit(X\_train\_sc,y\_train) |
| Prediction | y\_pred = model1.predict(X\_test\_sc) |

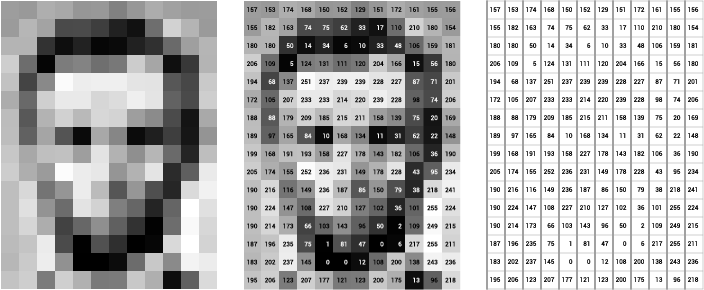
KNN: - it is mostly used for classification problems and also used sometime for regression problems. It is used when the two or more independent variables or column are corelated. And should contains less data or less columns do perform this model. KNN is used to give the values based on the number of nearest values we give k =5 as default k value, which means the model has to take 5 nearest values to give the predicted value.

Here It takes the nearest and neighbour values to give the predicted value by making the mode of the nearest values as we given k=5 so here this model or algorithm takes those nearest 5 values and gives the most frequent value as the predicted value.

It is mostly used for image processing, pixel arranging kind of data…

KNN can be used for a variety of applications, such as image recognition, recommendation systems, and anomaly detection.

KNN is simple to implement but it is complex to determine the k value.



In the above picture if we want to predict the output values It basically takes the nearest values and gives the predicted values which is the most frequent value so that the output or the overall picture wont effect. And looks same as before.

**Multicollinearity**

|  |  |
| --- | --- |
| 1st Correlation | Data.corr() |
| Import Multicollinearity  VIF Import | from statsmodels.stats.outliers\_influence import variance\_inflation\_factor |

There will be some situations where **Curse of Dimensionality** occurs which means the situation where the data has complexity and contains multiple dimensions and it became difficult to understand and identifying meaningful patterns and relationships within the data.

To overcome this situation, we have mainly 2 options they are

1. Feature Selection (Supervised Learning)
   1. Multicollinearity
2. Feature Extraction (Un Supervised Learning)
   1. PCA

**Feature Selection: -** it is used to remove the highly correlated column from the data. Columns are removed because if it is highly correlated then the prediction may not be containing good values or percentage of the r2 score.

**Multicollinearity: -** it is the situation where we cannot select one thing from the multiple values because of it contains same features and same importance. this situation can be handled by this Multicollinearity method.

Ex: - in front of you there are 5 toys looks same and have same dimensions, same color, same size and shape… so it is difficult to select your favourite one from all 5.

**Feature Extraction: -** in this method reducing the dimensionality by identifying the most relevant features which are contributed to the variance in the data and eliminating those features which are irrelevant features and increasing the accuracy of the prediction and improving the efficiency of the machine learning algorithm.

Identifying the Multicollinearity

1. By having the good Knowledge of the data

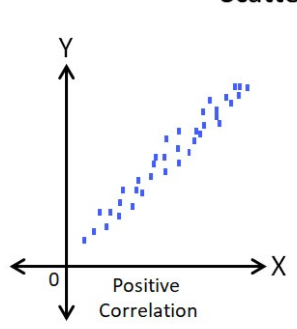
|  |  |  |
| --- | --- | --- |
| Grade | Experience | Salary |

Here the Grade and Experience Increases Salary also increases

If Grande and Experience is decreasing the salary also decreases

So here these Grade and Experience are correlated and it knows by having the good knowledge of the data

1. By scatter plot



All the points are pointed near to the fit line which shows all the values are close to each other which means features or columns are correlated

1. By plotting the correlation matrix

dataset.corr()

1. VIF (Variance Inflation factor) technique

if VIF value is more than it will be removed in the feature selection, Multicollinearity.

VIF formula: -

1/1 – r2\_score ; (1/1-R2) both are same

So here in this VIF method

1. Linear Regression which means Simple or Multi Linear Regression depends on data.
2. Finding r2\_score
3. VIF

In VIF method

Suppose we have X1, X2, X3, X4, and X5. There are 5 columns

So, 1st it takes X1 as y means output (dependent variable) others as Input and finds the VIF value. Next same 2nd time it takes X2 as y and rest as Input then 3rd, 4th, and 5th. So, we will get 5 different VIF values for each column or feature then removes the highest value and do the same process with those 4 features or columns then we have 4 VIF values then again removes the highest value column from the dataset and do the same process again.

We will remove the columns from the dataset when we find the values is grater then the required value Ex: - required value may 6 8 or 10 depends on data and the choice of the data analyst, if Data Analyst chooses the required number as 6.

Then every time the highest value column is removed and processed again until all the values are less than 6.

If r2\_score is high then the VIF value also go high

The r2\_score ranges between 0 to 1

Note: - it deals with numerical data only so remove the object type columns and output columns

**Process: -**

Appling the filter to remove the object type columns and the output columns from the dataset to perform VIF method

col\_list = []

for i in col:

if((Data[i].dtype!='object') & (i!='charges')):

col\_list.append(i)

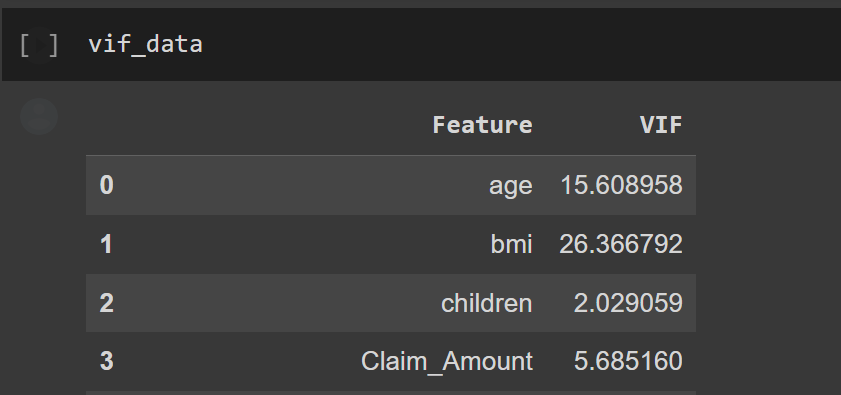
X = Data[col\_list]

vif\_data = pd.DataFrame()

vif\_data['Feature'] = X.columns

vif\_data['VIF'] = [variance\_inflation\_factor(X.values,i) for i in range(len(X.columns))]

**Output: -**



Remove the highest value contains column

Ex: - drop bmi column from the dataset

Then again perform the same above step to make the data frame then again remove or drop the highest value contained column then perform the same steps do repeat until all values reach with in the required value Ex: - 6 here

Repeating the process means simply droping the column and copy the above code and paste it in the next cell simply use these two steps to reach the required value.

**Decision Tree**

|  |  |
| --- | --- |
| Importing the data from sklearn | from sklearn.datasets import load\_iris |
| Importing the data from seaborn | iris1 = sns.load\_dataset('iris') |
| assigning variable to the data | iris = load\_iris() |
| Decision Tree for classification problems | from sklearn.tree import DecisionTreeClassifier |
| Decision Tree for regression problems | sklearn.tree import DecisionTreeRegressor |
| model assigning | model1 = DecisionTreeClassifier() |
| Plotting the decision tree | from sklearn import tree  plt.figure(figsize = (15,10))  tree.plot\_tree(model1, filled = True)  plt.show() |
| post pruning  it is done after the Decision tree | model2 = DecisionTreeClassifier(max\_depth = 2)  model2.fit(X\_train,y\_train) |

It is basically used in classification problems; it works like IF ELSE conditions and it form tree like structure by taking the decisions on if else conditions.

There is a website having a game which works on decision tee: - <https://en.akinator.com>

**1. Purity: -** it is used to predict the node is a leaf node or not, by the bases of Entropy or Gini Index. Simply by entropy or Gini it gives the tree where to stop and continue.

**2. Gini or Entropy: -**

**Entropy: -** it is ranges from 0 to 1

0: - then it is a leaf node means end of this side (pure split)

1: - then it is a decision node means it must continue the tree (Impure split)

H(s) = -P+ Log2 P+ - P- Log2 P-

H(s) = Entropy

P+ = Yes

P- = No

P = Probability of being positive

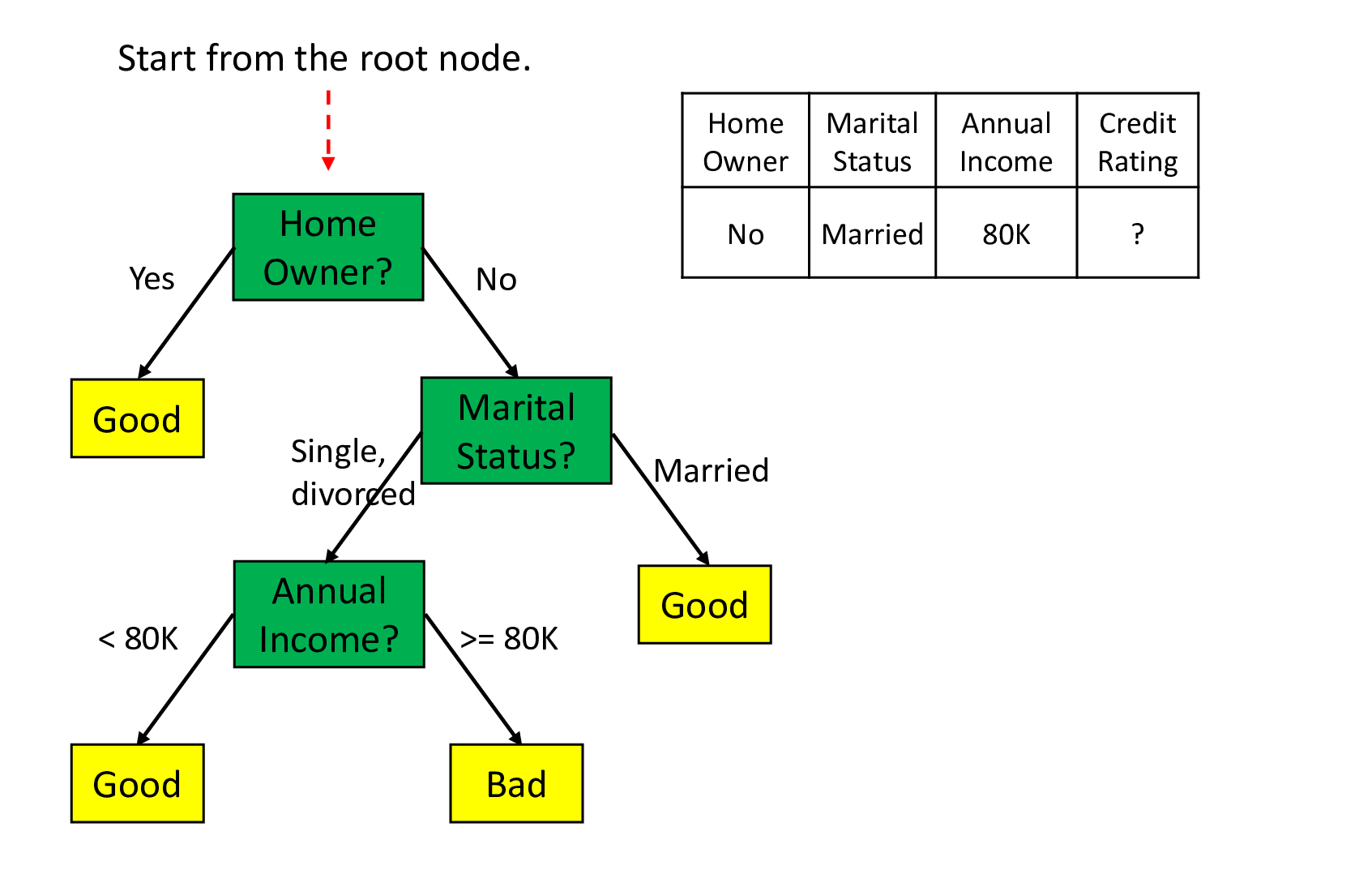
Simply:

H(s) = -Pyes Log2 Pyes – Pno Log2 Pno

**Gini Index: -** it ranges from 0 to 0.5

0: - pure split

0.5: - impure split



**3. Information Gain: -** it calculates how much information can provided by the column

According to the information it takes the tree split the nodes into leaf node and decision node.

It selects the highest values in the information as its priority

Ex: - 0.5 and 0.7 from these two values it takes 0.7 as its priority

How trees made in the beginning Here it takes the output variable as the road node. So, in that root node, we’ll be having yes and no elements. Then it split into multiple decision notes here. Each node represents each independent variable, which is input variable. In this there will be values Which represent yes and no for output

|  |  |  |  |
| --- | --- | --- | --- |
| Col1 | Col2 | Col3 | Output |
| 5 | 6 | 7 | Yes |
| 7 | 4 | 8 | no |

Here above is the example data we can see in each column. There are values which represent yes and no So in this way, it counts the number of values from independent variable is representing “yes” and number of values which representing “no” in independent variables.

From the above tree:

Here F1 is output variable, which contains. 9 yes and 5 no.

Nest nodes are from independent variables, input variables. So input variables representing 6 yes and no respectively.

Formula: -

IG = entropy (parent) – [(entropy (child 1) + entropy (child 2)]

Including Entropy:

IG = …

H(s) = -P+ Log2 P+ - P- Log2 P-  🡪 (entropy of parent)

Node1 = 6+2

Node2 = 5+3

H(c1) = entropy of Node 1

H(c2) = entropy of Node 2

F1 = Feature 1 or (column 1) like for all columns

H(C1) = (-(Yes) / (Full) Log2 (Yes) / (Full)) - ((No) / (Full) Log2 (No) / (Full))

Including Gini:

IG = …

**4. Pruning: -** Pruning is of two types

* **Pre-Pruning: -** It is done before the Decision Tree made
  + - * Grid Search CV
* **Post-Pruning: -** It is done after the Decision Tree made
  + - * Max depth = 2

Pruning is done It the data makes the Decision Tree too large simply if the Decision Tree is too large to understand then we will do pruning to make the Decision tree look small.

In Decision Tree 1st step: -

Selecting the column with the help of the Information Gain (IG)

Splitting the Nodes into leaf node or decision node according to the entropy or Gini

Then Post pruning if the tree is too large

And pruning helps to avoid over fitting.

**Pruning**

**Hyper Parameter Tuning**

|  |  |
| --- | --- |
| Hyper Parameters | parameter = {  'criterion' : ["gini", "entropy", "log\_loss"],  'splitter' : ["best","random"],  'max\_depth' : [1,2,3,4,5],  'max\_features' : ["auto","sqrt","log2"]  } |
| Import Grid Search CV | from sklearn.model\_selection import GridSearchCV |
| calling Grid Search CV with details | model2 = GridSearchCV(model1, param\_grid = parameter, cv=5, scoring = "accuracy") |
| Training the model | model2.fit(X\_train,y\_train) |
| choice of the parameter selection by the model | model2.best\_params\_ |
| Prediction | y\_pred = model2.predict(X\_test) |

**Pruning: -**

Pruning is used to cut down the over fitting and under fitting problems in Decision Tree

And makes the model to train and gives the better accuracy.

It is used to cut down the extra leaf nodes from the decision Tree.

Over Fitting: - problem contains low Bias but high Variance

Under Fitting: - Problems contains high Bias and high Variance

Bias = Error in Training the model

Variance= Error in Testing the model

**High bias:** Underfitting, poor performance on both training and test data.

**High variance:** Overfitting, good performance on training data but poor performance on test data.

Pruning is of Two types

1. Post-Pruning
2. Pre-Pruning

Post-Pruning: - it is used after the Decision Tree formed and here model is trained with Hyper parameter called max depth

Max depth: - it is used to display the levels of the decision tree by cutting the Extra nodes.

Pre-Pruning: - it is used before the Decision Tree formed and here model is trained with Hyper Parameter tuning or simply, we can say Hyper parameter Optimisation. And the hyper parameter called Grid Search CV

Parameters: - parameters are of two types they are

1. Model parameters
2. Hyper parameters

Model parameters: - it is usually applied directly by the model

Ex: - in logistic regression its 1st applies linear regression formula then sigmoid function to train the data and predict the values and those formulas are the parameters of the model.

Hyper parameters: - here in this parameter we need to give the parameters 1st Externally.

Ex: - In decision tree in pruning method we will give max depth for post pruning and Gini or entropy for pre-pruning method.

parameter = {

'criterion' : ["gini", "entropy", "log\_loss"],

'splitter' : ["best","random"],

'max\_depth' : [1,2,3,4,5],

'max\_features' : ["auto","sqrt","log2"]

}

Grid Search CV: -

model2 = GridSearchCV(model1, param\_grid = parameter, cv=5, scoring = "accuracy")

**param grid**: - is used to Identify the best parameter,

**cv**: - cross validation, we use cv for hyper parameter tuning.

Here cv = 5 means here the given Input data is split into 5 different components

Ex: - Training data = 8000

So, if cv = 5 than 8000/5 = 1600

Validation data = 1600

Leftover data = 6400 = training data

Validation data is marked in green colour in each component.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0-1600 | 1600-3200 | 3200-4800 | 4800-6400 | 6400-8000 |
| 0-1600 | 1600-3200 | 3200-4800 | 4800-6400 | 6400-8000 |
| 0-1600 | 1600-3200 | 3200-4800 | 4800-6400 | 6400-8000 |
| 0-1600 | 1600-3200 | 3200-4800 | 4800-6400 | 6400-8000 |
| 0-1600 | 1600-3200 | 3200-4800 | 4800-6400 | 6400-8000 |

Here the Training data is split into 5 different components and each component the values are different in validation, in 1st component from 0 to 1600 values are in validation then in 2nd component from 1600 to 3200 values are in validation and in 3rd component 3200 to 4800 and 4th component 4800 to 6400 and last in 5th component 6400 to 8000 values are in validation.

So, if **cv = 5** the total training data is split in to 5 component this time.

And algorithm trained with multiple sub data to overcome the overfitting problem.

**Scoring = ‘accuracy’**

the accuracy of the model will be used as the metric to evaluate the performance of the model. Accuracy is a common metric used in classification problems, which calculates the proportion of correct predictions among all predictions made by the model.

**Ensemble Learning**

**Ensemble Learning: -** It is a Learning Technique in which multiple Individual models come together to create a Super Model. It works for both regression and classification problems.

Ex: - there are around 100 decision trees are trained by taking the samples of the dataset and then it gets combine and give the output.

There are two Types

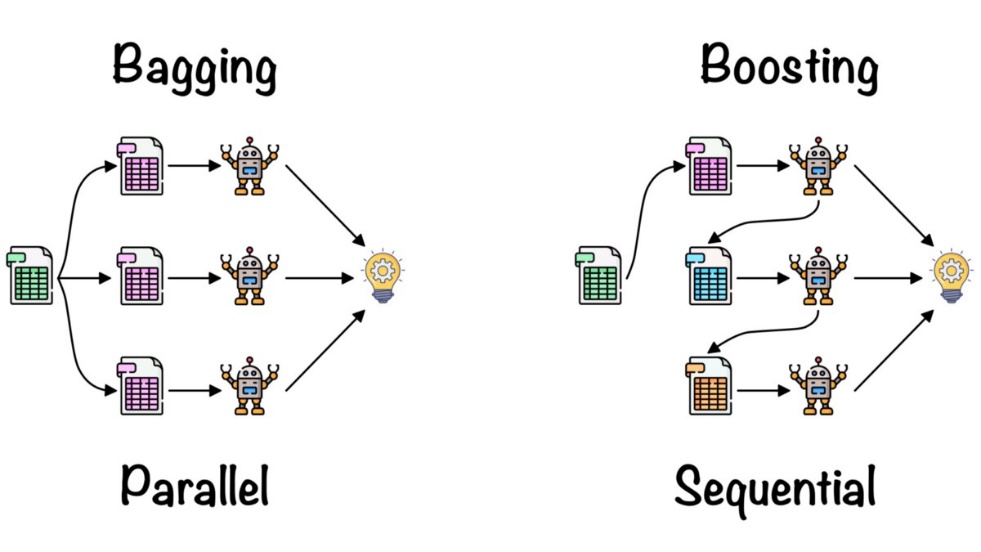
1. Bagging

Ex: - **Random Forest**

1. Boosting

Ex: - **adaboost**

Both are used for Regression and Classification problems.



1. **Bagging: -** It is a Technique part of Ensemble Learning to build a super model based on multiple sub models and here all the sub models are built parallelly. Here all the models are Independent and Equally distributed by each sub models.

Ex: - **Random Forest**

**Random Forest**

|  |  |
| --- | --- |
| Import regression problem | from sklearn.ensemble import RandomForestRegressor |
| Import Classification Problem | from sklearn.ensemble import RandomForestClassifier |
| Model assigning | model2 = RandomForestRegressor  (n\_estimators = 150) |
| fit (train) | model2. fit(X\_train\_sc, y\_train) |
| Prediction | y\_pred2 = model2.predict(X\_test\_sc) |

Random Forest is a Supervised Machine Learning Algorithm and refers to Bagging Technique. Used for both regression and classification problems. It is based on Ensemble Learning Concept. It contains n number of Fully grown Decision Trees as the sub models.

**Dataset**

**M = columns**

**D = rows**

If Regression

Mean/median

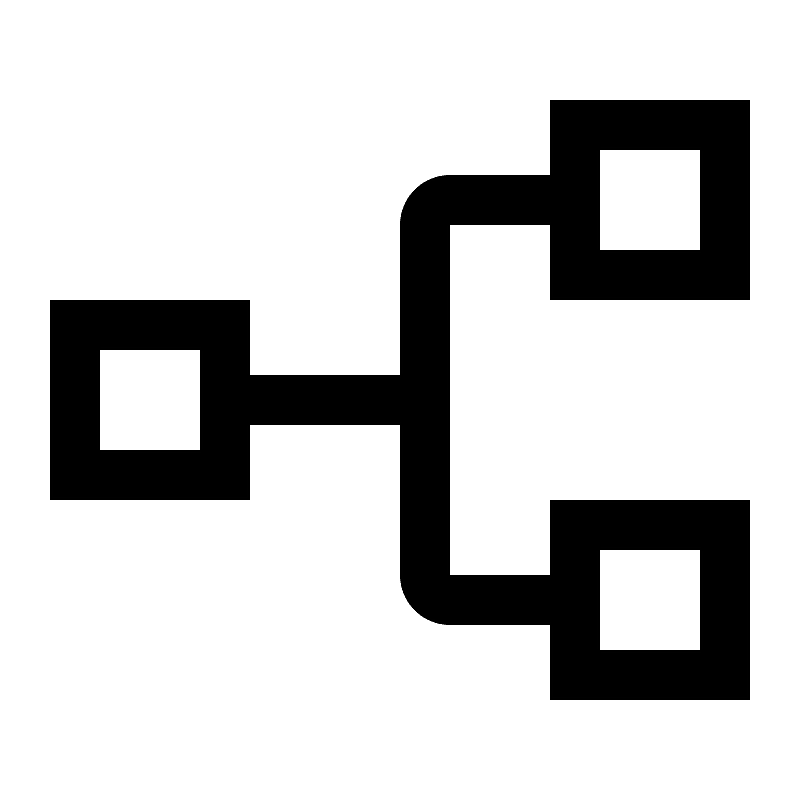
If classification

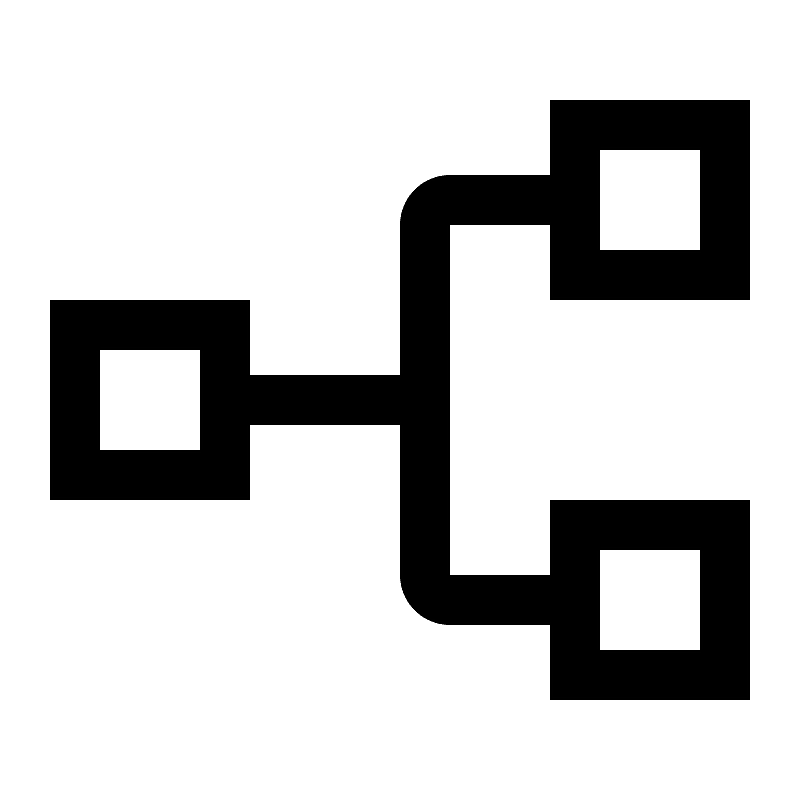
mode

**Dataset**

**Decision Tree models** ModelsMomodels

**Output**

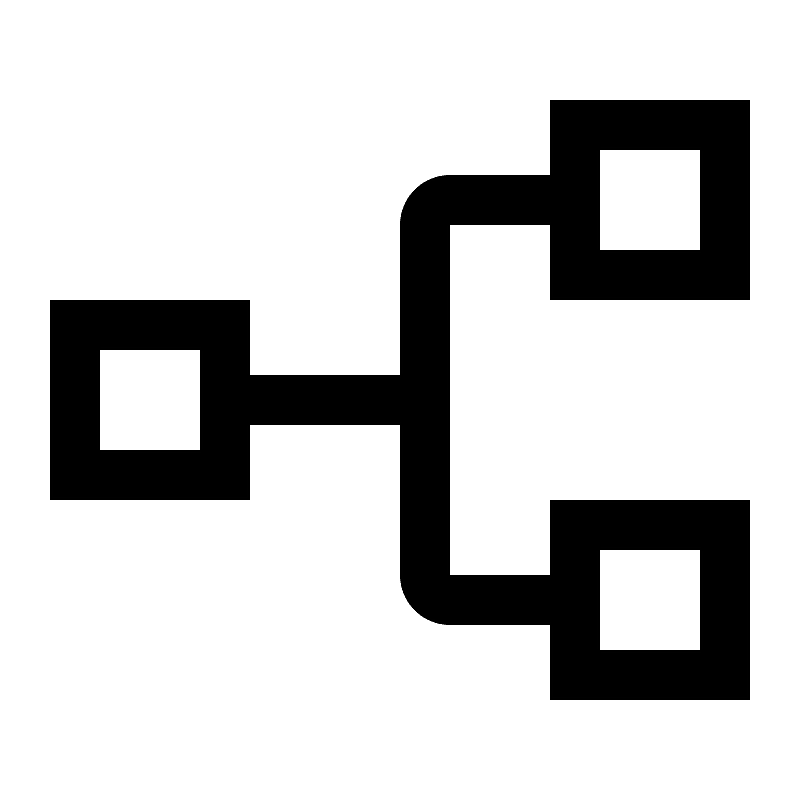




M3 D3

M2 D2

M1 D1



Total dataset contains all the rows and columns are sub divided and form a decision tree.

Those are sub divided by taking the rows and columns randomly with respect to formulas given below, and the selection or split happens randomly so there may be a chance of selecting a row or a column multiple times or in multiple sub models and also there is a chance of not selecting a row or a column which it selecting randomly. This is called as “with replacement”.

M = columns and D = rows in the main dataset and it split into multiple datasets and form decision trees as sub models where it takes the m1 and d1 as the columns and rows.

the division happens for rows 2/3 \* rows is chosen for each row with replacement.

Ex: - total 9 rows are split into 2/3 rows = 6 rows and it splits into sub modules until it reaches the n estimator to form decision trees.

The division happens for columns ; N = number of columns with replacement

Ex: - total 9 columns are split in to of columns in sub modules until it reaches the n estimator to form decision trees choose with replacement.

n estimator: - it is basically giving the hyper parameter to the random forest. It indicates the number of decision trees as sub model to form in random forest super model. Default

n estimator = 100.

**OOB: -** Out of Bagdata, data is never selected in the process of split or selection for the sub models of decision tree. There may be a column or a row or multiple columns or rows.

After formation of all the sub models the output of the sub models are collected and done.

Regression problem: - taking the mean or median for those outputs make one output for the model.

Classification problem: - taking the mode or majority vote of the all-outputs sub models to form one output from it.

1. **Boosting: -** it is a part of Ensemble Learning, in this all the sub models are built sequentially by taking the previous output as the input of the next model to reduce the error. And all the models have unique contribution.

Ex: **adaboost**

Base

Model

Base

Model

Base

Model

Base

Model

**Op as Ip**

**Op as Ip**

**Op as Ip**

**Adaboost**

|  |  |
| --- | --- |
| base model | base\_model2 = DecisionTreeClassifier() |
| adaboost model building | new\_model = AdaBoostClassifier(base\_estimator = base\_model2, n\_estimators = 150,learning\_rate=1.0) |
| Best Parameters | param\_grid = {  'n\_estimators': [50, 100, 200],  'learning\_rate': [0.1, 0.5, 1],  } |
| fit | new\_model.fit(X\_train\_sc,y\_train) |
| predict | y\_pred = new\_model.predict(X\_test\_sc) |
| classification report | print(classification\_report(y\_test,y\_pred)) |

Adaboost: -It works on boosting technique in Ensemble Learning. In this if we given a base model as decision tree here the sub models are not fully grown decision trees instead it just grows in to stumps which means there is one root node and multiple leaf nodes according to the data but you cannot see any decision nodes.

You can change the base model to any model for both regression and classification data using the base\_estimator parameter and we can give the number of sub models to form by giving the n\_estimator just like in random forest.

Ex: - Decision tree, Logistic regression, Random Forest etc…

Sample weight: - sample weight is assigned equally to each and every row of the dataset and the weight is assigned by the formula 1 / (no. of rows)

Ex: - there are 10 rows so it takes as 1/10 as the sample weight for every row.

Weights are assigned to each row to determined its significance in the training dataset, it’s a kind of hyper parameter tuning.

-------------------------------------------- From YouTube ------------------------------------------------

Total error: - sum of Unclassified rows sample weights. Let say Unclassified rows are 2 so 2 \* sample weight which is 1/10 so, 2\* 1/10 (or) 1/10 + 1/10…

Performance of stumps: - simply called it as performance say.

Formula: -

Performance say =

updated sample weight: -

For Unclassified row

For remaining rows

By adding the values into this formula, we will get the updated weights of each row.

Normalised values: -here in this dividing all the updated values with sum of the updated values wee will get the Normalised values.

Bucket range: - the values of each row are in a bucket range to make the values to fit in the rows with equal importance, so the data values will be replaced with these new normalised values with in the bucket range.

|  |  |  |
| --- | --- | --- |
| Normalised values | Normalised weights | math in side of it |
| 0.1 | 0 – 0.1 | from 0 to 0.1 |
| 0.1 | 0.1 – 0.2 | 0.1 + 0.1 = 0.2 |
| 0.3 | 0.2 – 0.5 | 0.2 + 0.3 = 0.5 |

the normalised bucket range would be start from 0 to the 1sr row updated weight and the 2nd rows normalised weight would be start from the end of the 1st row value and ends with the sum of previous row updated weight and the current row updated weight.

This process is continued by taking the less entropy value maintaining the decision tree and fill all the values in the dataset, so that if the value contains error will also be selected and taken as an input to the next model and continue to reduce the error.

-----------------------------Or simply use other algorithms-----------------------------

**Naive Bayes**

|  |  |
| --- | --- |
| Import | from sklearn.naive\_bayes import GaussianNB |
| model assigning | model = GaussianNB() |
| fit | model.fit(X\_train\_sc, y\_train) |
| predict | y\_pred = model.predict(X\_test\_sc) |

Naïve Bayes is a Supervised machine learning algorithm and mostly used for binary Classification problems. This algorithm used to predict based on predictions. This method mostly used on object type data by converting the data into numerical by encoding method.

Input data or columns as X1, X2, X3, … Xn (Independent values or columns or variables)

Ex: -

Let say a dice is an independent event contains {1, 2, 3, 4, 5, 6}

Probability of getting one 6 is 1/6

Probability of getting one 3 is 1/6 same

But in dependent variable as y (output column)

Here output of one event is affected by the output or past or other events. Basically, it is on binary classification

Ex: -

Let say one bag contains 2 red balls 3 blue balls. So, total 5 balls in one bag

Taking 1 red ball is 2/5

Taking 1 blue ball is 3/4 but its not 3/5 because we already removed 1 red ball from the bag.

So, for this we can say (blue given red) is ¾

Blue given red: -

P(B/R) 🡪 condition probability.

P (R and B) = P(R) \* P(B/R)

P (R and B) 🡪 Dependent event.

Note: - P (A and B) = P(A) \* P(B/A)

Where A and B are dependent events.

Note: - P (A and B) = P (B and A)

P(A) \* P(B/A) = P(B) \* P(A/B)

**Bayes Theorem: -**

P(B/A) =

Analysing: - let say X = A and y = B

X1, X2, X3, … Xn = A

y = B

applying the X and y, in Bayes Theorem

P (y / X1, X2, X3, … Xn) =

Now

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X1 | X2 | X3 | X4 | X5 | Xn | Output |
| 65 | 34 | 34 | 64 | 75 | 57 | 0 |
| 45 | 56 | 12 | 34 | 65 | 78 | 1 |

Basically, the binary classification output contains o and 1 which means yes and no in the dependent column.

So, y = Yes as case 1, and y = No as case 2.

Case 1: -

P (Yes / X1, X2, X3, … Xn) =

Case 2: -

P (No / X1, X2, X3, … Xn) =

So, in both the cases the denominator is common so we will consider it as constant and ignore it in the next step. Then applying the numerical values to those formulas, we will get the values in decimal points than do multiply with 100 we will get the values in percentage if the values of yes percentage is more than it takes the values as Yes, if not than takes the values as No. values takes as 1 and 0 in the output column so for Yes, its 1 and for No its 0. we must assign these 1 and 0 values accordingly by using encoding method.