VDDBD



ARTIFICIALINTELLIGENCE AND MACHINE LEARNING



21761A4226,21761A4259,21761A4264

II CSE (AI &Ml)

**INDEX**

**1.Machine Learning**

i. Find-S :1-4

ii. CEA :5-8

iii. Linear and Multi Linear Regression :9-14

iv. Polynomial Regression :15-16

v. Logistic Regression :17 - 21

vi. Decision Tree Regressor :22-25

vii. Decision Tree Classifier :26-31

viii. Random Forest Classifier :31-37

xi. Random Forest Regressor :38- 46

**2.Artifiacial Intelligence:**

i. Water Jug Problem :47-56

ii. Tic Tac Toe :57-62

iii. Monkey Banana Problem :62-69

iv. Towers of Hanoi :69-71

**AIM:**

To Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .csv file.

**Description:**

The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.

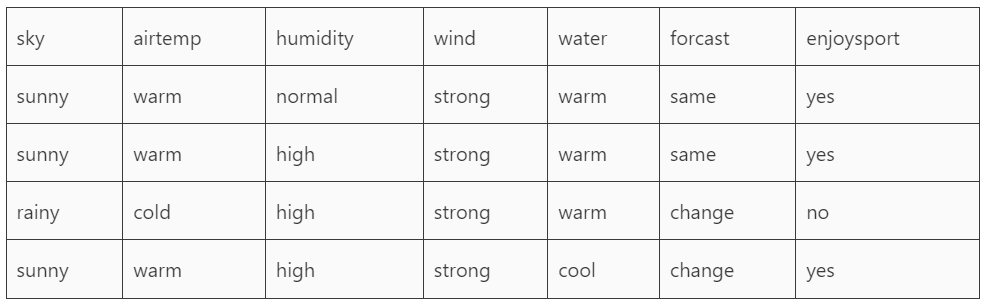
**Important Representation:**

1. **?**indicates that any value is acceptable for the attribute.
2. specify a single required value (e.g., Cold) for the attribute.
3. **ϕ** indicates that no value is acceptable.
4. The most **general hypothesis** is represented by: **{?, ?, ?, ?, ?, ?}**
5. The most **specific hypothesis** is represented by: **{ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}**

**Steps Involved In Find-S :**

1. Start with the most specific hypothesis.   
   **h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}**
2. Take the next example and if it is negative, then no changes occur to the hypothesis.
3. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.
4. Keep repeating the above steps till all the training examples are complete.
5. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples.

**Dataset:**



**Python Program:**

import pandas as pd

import numpy as np

d=pd.read\_csv("find.csv")

print(d)

a=np.array(d)[:,1:-1]

print("The attribute are:",a)

t=np.array(d)[:,-1]

print("The target is:",t)

print("The no.of instances",len(a))

n\_att=len(a[0])-1

print("the initial hypothesis is:")

hypothesis=['0']\*n\_att

print(hypothesis)

for i in range(0,len(a)):

    if t[i]=='Yes':

        print("instance",i+1,"is",a[i],"and is positive instances")

        for j in range(0,n\_att):

            if hypothesis[j]=='0' or hypothesis[j]==a[i][j]:

                hypothesis[j]=a[i][j]

            else:

                hypothesis[j]='?'

        print("hypothesis for training instance",i+1,"is:",hypothesis,"\n")

    if t[i]=='No':

        print("instance",i+1,"is",a[i],"and negative instances are ignored")

        print("The hypothesis for the training instances",i+1,"is",hypothesis,"\n")

print("\n The maximally specific hypothesis for the training instance is",hypothesis)

**Output:**

S.NO Sky AirTemp Humidity Wind Water Forecast EnjoySport

0 Sunny Warm Normal Strong Warm Same Yes

1 Sunny Warm High Strong Warm Same Yes

2 Rainy Cold High Strong Warm Change No

3 Sunny Warm High Strong Cool Change Yes

The attribute are: [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

The target is: ['Yes' 'Yes' 'No' 'Yes']

The no.of instances 4

the initial hypothesis is:

['0', '0', '0', '0', '0']

instance 1 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same'] and is positive instances

hypothesis for training instance 1 is: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm']

instance 2 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same'] and is positive instances

hypothesis for training instance 2 is: ['Sunny', 'Warm', '?', 'Strong', 'Warm']

instance 3 is ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change'] and negative instances are ignored

The hypothesis for the training instances 3 is ['Sunny', 'Warm', '?', 'Strong', 'Warm']

instance 4 is ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change'] and is positive instances

hypothesis for training instance 4 is: ['Sunny', 'Warm', '?', 'Strong', '?']

The maximally specific hypothesis for the training instance is ['Sunny', 'Warm', '?', 'Strong', '?']

**AIM:**

To Implement and demonstrate CEA algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .csv file.

**Description:**

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

You can consider this as an extended form of the Find-S algorithm.

Consider both positive and negative examples.

Actually, positive examples are used here as the Find-S algorithm (Basically they are generalizing from the specification).

While the negative example is specified in the generalizing form.

**Algorithm:**

**Step1:** Load Data set

**Step2:** Initialize General Hypothesis and Specific Hypothesis.

**Step3:** For each training example

**Step4:** If example is positive example

if attribute\_value == hypothesis\_value:

Do nothing

else:

replace attribute value with '?' (Basically generalizing it)

**Step5:** If example is Negative example

Make generalize hypothesis more specific.

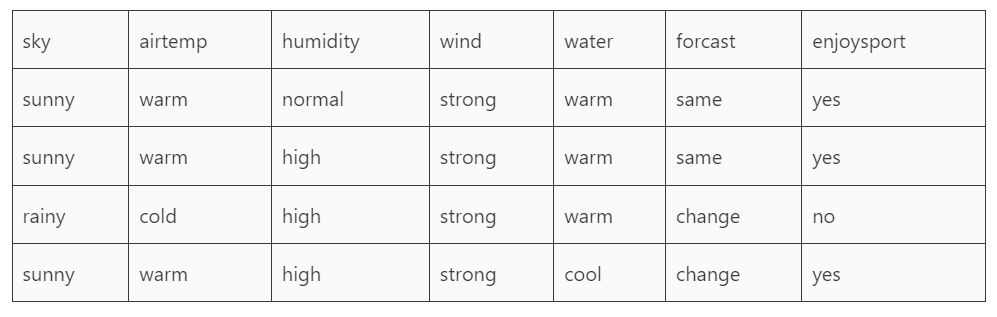
**Advantages of CEA over Find-S:**

1. Improved accuracy: CEA considers both positive and negative examples to generate the hypothesis, which can result in higher accuracy when dealing with noisy or incomplete data.
2. Flexibility: CEA can handle more complex classification tasks, such as those with multiple classes or non-linear decision boundaries.
3. More efficient: CEA reduces the number of hypotheses by generating a set of general hypotheses and then eliminating them one by one. This can result in faster processing and improved efficiency.
4. Better handling of continuous attributes: CEA can handle continuous attributes by creating boundaries for each attribute, which makes it more suitable for a wider range of datasets.

**Disadvantages of CEA in comparison with Find-S:**

1. More complex: CEA is a more complex algorithm than Find-S, which may make it more difficult for beginners or those without a strong background in machine learning to use and understand.
2. Higher memory requirements: CEA requires more memory to store the set of hypotheses and boundaries, which may make it less suitable for memory-constrained environments.
3. Slower processing for large datasets: CEA may become slower for larger datasets due to the increased number of hypotheses generated.
4. Higher potential for overfitting: The increased complexity of CEA may make it more prone to overfitting on the training data, especially if the dataset is small or has a high degree of noise.

**Dataset:**

****

**Python Program:**

import numpy as np

import pandas as pd

data = pd.read\_csv(path+'/enjoysport.csv')

concepts = np.array(data.iloc[:,0:-1])

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1])

print("\nTarget Values are: ",target)

def learn(concepts, target):

    specific\_h = concepts[0].copy()

    print("\nInitialization of specific\_h and genearal\_h")

    print("\nSpecific Boundary: ", specific\_h)

    general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

    print("\nGeneric Boundary: ",general\_h)

    for i, h in enumerate(concepts):

        print("\nInstance", i+1 , "is ", h)

        if target[i] == "yes":

            print("Instance is Positive ")

            for x in range(len(specific\_h)):

                if h[x]!= specific\_h[x]:

                    specific\_h[x] ='?'

                    general\_h[x][x] ='?'

        if target[i] == "no":

            print("Instance is Negative ")

            for x in range(len(specific\_h)):

                if h[x]!= specific\_h[x]:

                    general\_h[x][x] = specific\_h[x]

                else:

                    general\_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific\_h)

        print("Generic Boundary after ", i+1, "Instance is ", general\_h)

        print("\n")

    indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

    for i in indices:

        general\_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h: ", s\_final, sep="\n")

print("Final General\_h: ", g\_final, sep="\n")

**Output:**

Instances are:  
[[‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]  
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’]  
[‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’]  
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’]]

Target Values are: [‘yes’ ‘yes’ ‘no’ ‘yes’]

Initialization of specific\_h and genearal\_h

Specific Boundary: [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary: [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

**AIM:**

Implement Linear and Multi Linear Regression.

**Description:**

Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. The goal of linear regression is to find the linear equation that best describes the relationship between the variables.

In a simple linear regression model, there is only one independent variable that is used to predict the dependent variable. The equation for a simple linear regression model is:

y = b0 + b1\*x

where y is the dependent variable, x is the independent variable, b0 is the intercept, and b1 is the slope. The intercept represents the value of y when x is zero, and the slope represents the change in y for every one unit change in x.

In multiple linear regression models, there are multiple independent variables that are used to predict the dependent variable. The equation for a multiple linear regression model is:

y = b0 + b1\*x1 + b2\*x2 + ... + bn\*xn

where y is the dependent variable, x1, x2, ..., xn are the independent variables, b0 is the intercept, and b1, b2, ..., bn are the slopes.

The process of finding the best linear equation involves estimating the values of the intercept and slopes using a method called least squares. In this method, the sum of the squared differences between the predicted values and the actual values is minimized.

Linear regression models can be used for various applications, such as predicting future values of a dependent variable based on historical data, determining the relationship between two variables, and identifying which independent variables are the most important predictors of the dependent variable.

**Performance Metrics:**

Performance metrics are used to evaluate the accuracy and effectiveness of a regression model. Here are some of the commonly used performance metrics for regression:

1. Mean Squared Error (MSE): This metric measures the average of the squared differences between the predicted values and the actual values. It is calculated by taking the sum of the squared errors and dividing it by the number of observations.

2. Root Mean Squared Error (RMSE): This is the square root of the MSE and provides a measure of the average deviation of the predicted values from the actual values.

3. Mean Absolute Error (MAE): This metric measures the average of the absolute differences between the predicted values and the actual values.

4. R-squared (R2): This metric measures the proportion of the variance in the dependent variable that is explained by the independent variables. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data.

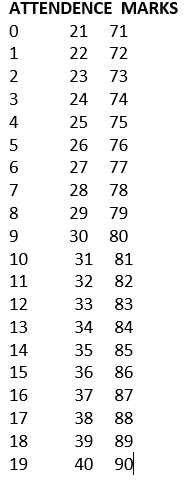
5. Adjusted R-squared: This metric is a modification of R-squared that adjusts for the number of independent variables in the model. It penalizes models that include unnecessary variables.

6. Mean Absolute Percentage Error (MAPE): This metric measures the average percentage difference between the predicted values and the actual values. It is calculated as the average of the absolute differences between the predicted values and the actual values, divided by the actual values, multiplied by 100.

7. Coefficient of Determination (COD): It is similar to R-squared, but it is normalized. COD is a measure of the strength of correlation between the observed and predicted values.

Choosing the appropriate performance metric depends on the specific problem being addressed and the goals of the model. For example, MSE and RMSE are useful for evaluating the accuracy of predictions, while MAE is useful for evaluating the magnitude of errors. R-squared and adjusted R-squared are useful for evaluating the overall fit of the model, while MAPE and COD are useful for evaluating the performance of the model in the context of a particular application.

**Dataset:**

****

**Python Progran:**

import pandas as pd

#Load the Dataset.

df=pd.read\_csv("D:/attendence.csv")

print(df)

x=df[["ATTENDENCE"]]

y=df["MARKS"]

print(x)

print(x.shape)

print(y)

print(y.shape)

#Fit the linear regression model.

from sklearn.linear\_model import LinearRegression

model=LinearRegression()

model.fit(x\_train,y\_train)

result=model.predict(x\_test)

print(result)

#Find the accuracy.

from sklearn.metrics import r2\_score

r=r2\_score(y\_test,result)

print(r)

#Find the error rate.

from sklearn.metrics import mean\_absolute\_error

r1=mean\_absolute\_error(y\_test,result)

print(r1)

#Plotting

import matplotlib.pyplot as plt

plt.scatter(y\_test,result)

plt.plot(y\_test,result)

plt.show

**Output:**

(14, 1)

(14,)

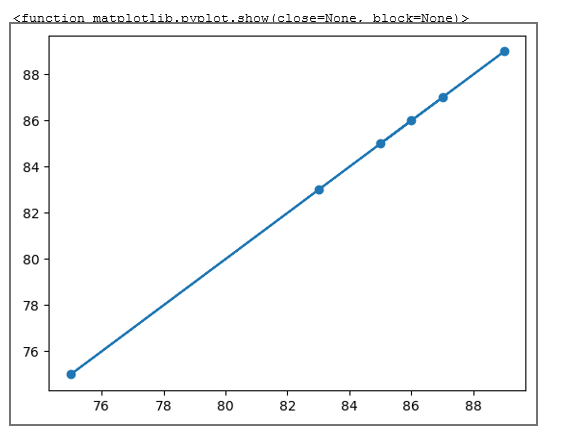
(6, 1)

(6,)

[86. 85. 89. 83. 75. 87.]

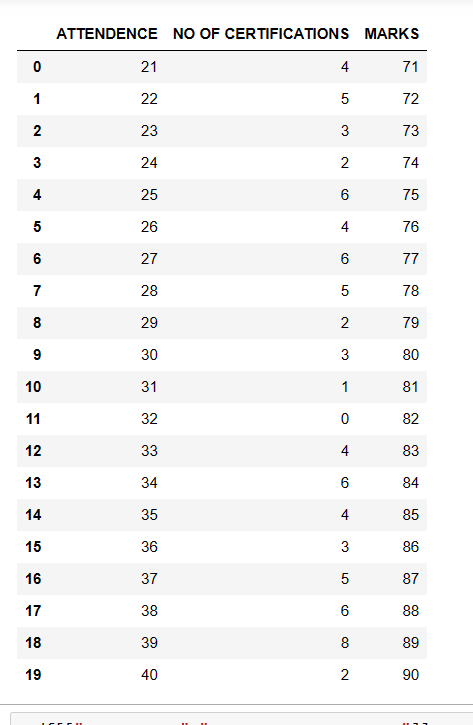
1.0

0.0

****

**ii. Multi Linear Regression:**

**Dataset:**



**Python Program:**

#Import the required libraries.

import pandas as pd

from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

#Load the dataset

df=pd.read\_csv("Attendence.csv")

df

x=df[["ATTENDENCE","NO OF CERTIFICATIONS"]]

y=df["MARKS"]

print(x.shape)

print(y.shape)

OUTPUT:

(20, 2)

Name: MARKS, dtype: int64

(20,)

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3)

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

#Fit the linear regression model.

from sklearn.linear\_model import LinearRegression

model=LinearRegression()

model.fit(x\_train,y\_train)

result=model.predict(x\_test)

print(result)

#Find the accuracy.

from sklearn.metrics import r2\_score

r=r2\_score(y\_test,result)

print(r)

#Find the error rate.

from sklearn.metrics import mean\_absolute\_error

r1=mean\_absolute\_error(y\_test,result)

print(r1)

#Plotting

import matplotlib.pyplot as plt

plt.scatter(y\_test,result)

plt.plot(y\_test,result)

plt.show

**Output:**

(14, 2)

(14,)

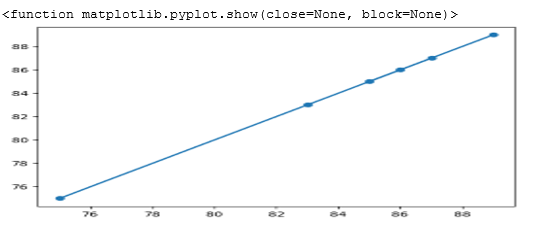
(6, 2)

(6,)

[86. 85. 89. 83. 75. 87.]

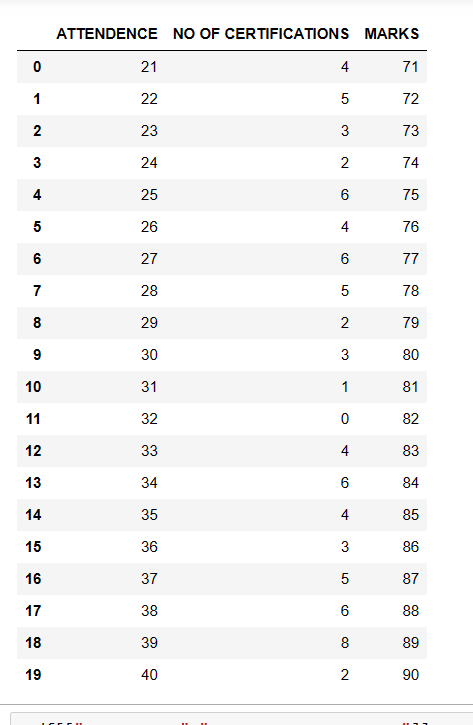
1.0

0.0



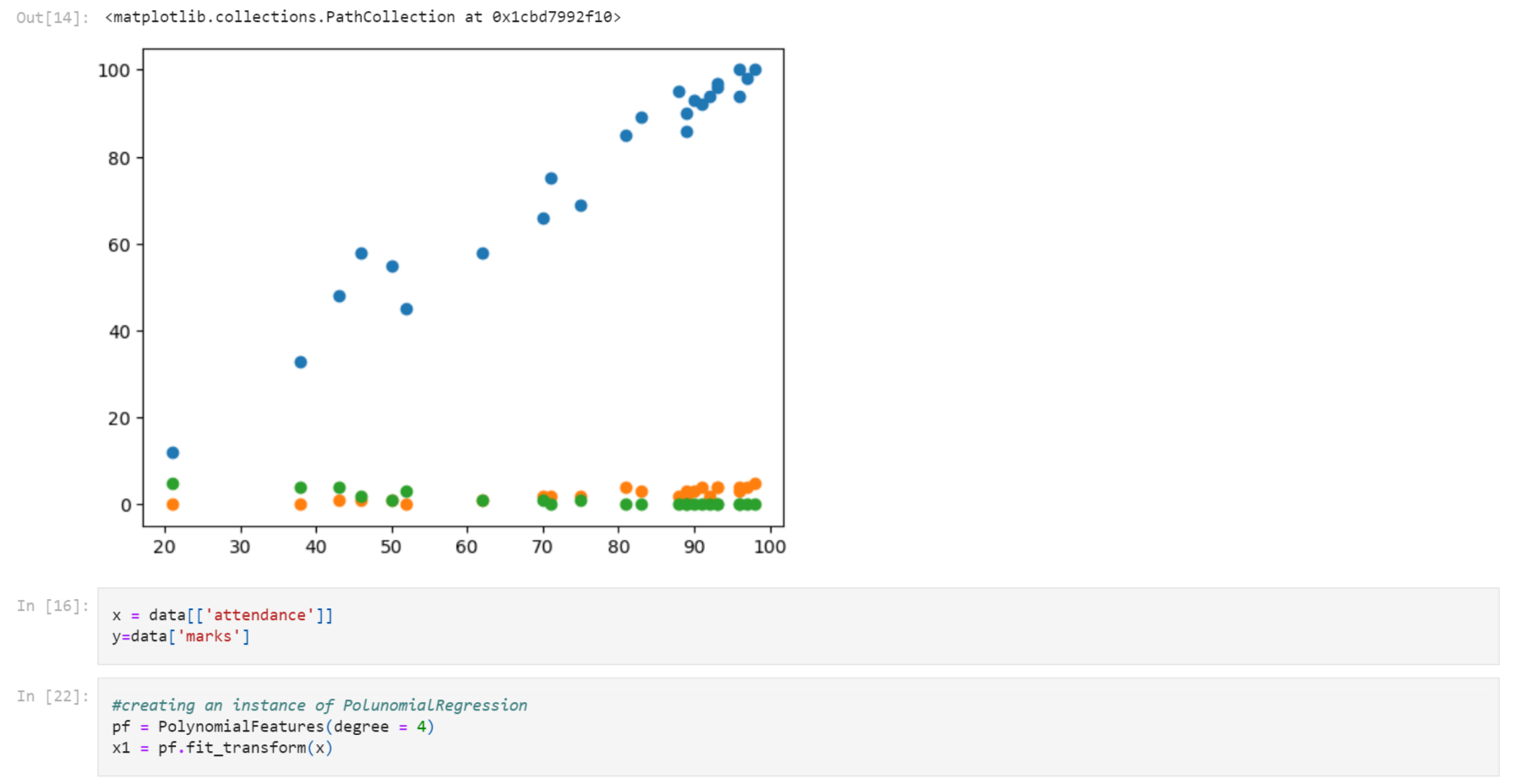
**Aim:** ImplementationPolynomial Regression using Python.

**Dataset:**



**Python Program And Output:**







**AIM:**Write a python program to demonstrate Logistic Regression using an appropriate dataset.

**Description:**

Logistic regression is a statistical method used to model the relationship between a binary dependent variable and one or more independent variables. The dependent variable can take only two possible values, such as "yes" or "no", "true" or "false", "0" or "1". Logistic regression is widely used in various fields, including healthcare, marketing, and social sciences.

The goal of logistic regression is to estimate the probability that the dependent variable is equal to 1 (or "true") given the values of the independent variables. The logistic regression model estimates the probability of the dependent variable using the logistic function (also called the sigmoid function), which maps any real-valued number to a value between 0 and 1. The logistic function has an S-shaped curve, which allows the model to model non-linear relationships between the dependent and independent variables.

The logistic regression equation is:

P(Y=1|X) = e^(β0 + β1X1 + β2X2 + ... + βnXn) / (1 + e^(β0 + β1X1 + β2X2 + ... + βnXn))

where P(Y=1|X) is the probability of the dependent variable (Y) being 1 given the values of the independent variables (X), β0 is the intercept, β1, β2, ..., βn are the slopes, and X1, X2, ..., Xn are the values of the independent variables.

To estimate the coefficients (slopes) of the logistic regression equation, a maximum likelihood estimation (MLE) method is used. The MLE method finds the values of the coefficients that maximize the likelihood of the observed data.

Logistic regression models can be evaluated using various performance metrics, including accuracy, precision, recall, F1 score, and receiver operating characteristic (ROC) curve. These metrics are used to evaluate the model's ability to correctly classify the dependent variable and to assess the model's performance for different thresholds.

**Evaluation Parameters for Logistic Regression(Classification Problems):**

Logistic regression models can be evaluated using various performance metrics, including:

1. **Confusion Matrix:** A confusion matrix is a table that summarizes the performance of a classification algorithm. It shows the number of true positives, true negatives, false positives, and false negatives.

2. **Accuracy:** Accuracy measures the proportion of correctly classified instances. It is calculated as the number of correctly classified instances divided by the total number of instances.

3. **Precision:** Precision measures the proportion of true positives among the instances that were classified as positive. It is calculated as the number of true positives divided by the sum of true positives and false positives.

4. **Recall (Sensitivity):** Recall measures the proportion of true positives among the instances that are actually positive. It is calculated as the number of true positives divided by the sum of true positives and false negatives.

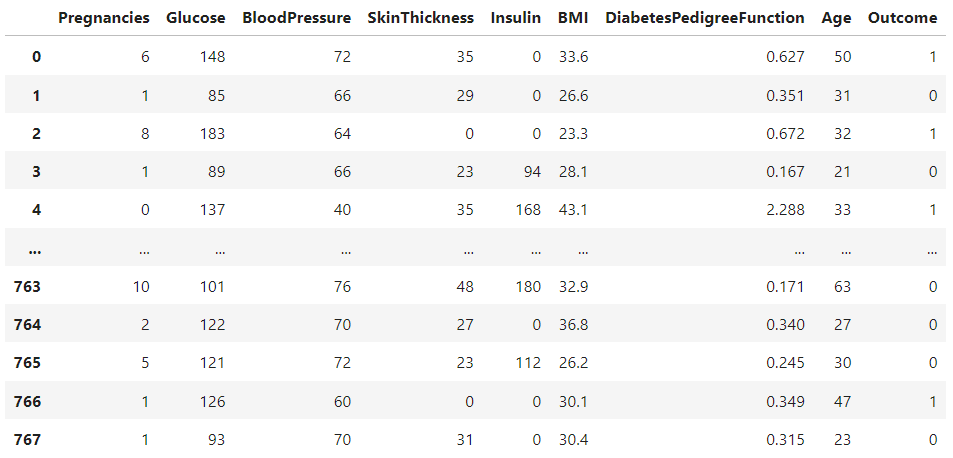
5. **F1 Score:** F1 score is the harmonic mean of precision and recall. It is a single metric that balances both precision and recall. It is calculated as 2 \* (precision \* recall) / (precision + recall).

6. **Receiver Operating Characteristic (ROC) Curve:** ROC curve is a graphical representation of the performance of a binary classifier. It shows the trade-off between the true positive rate and the false positive rate for different threshold values.

7. **Area Under the Curve (AUC):** AUC is the area under the ROC curve. AUC is a single metric that summarizes the overall performance of the classifier.

The choice of evaluation metric depends on the specific problem being addressed and the goals of the model. For example, accuracy may be appropriate if the classes are balanced, while recall may be more important if the goal is to identify all positive instances. The ROC curve and AUC are useful for evaluating the overall performance of the model across different threshold values.

**Dataset:**



**Correlation Between Attributes using data.corr():**



**Python Program:**

#Import Required Libraries

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

#Load the Data Set and Display

data=pd.read\_csv(r"C:\Users\Harsha\Downloads\diabetes.csv")

# Split the dataset into features and target

x=data.drop("Outcome",axis=1)

y=data["Outcome"]

# Split the dataset into training and testing data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2)

x\_train.shape

y\_train.shape

# Create the Logistic Regression Model and fit the model using the training data

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression()

classifier.fit(x\_train, y\_train)

# Make predictions on the test data

y\_pred = classifier.predict(x\_test)

y\_pred

# Print the confusion matrix

from sklearn.metrics import confusion\_matrix

c=confusion\_matrix(y\_test,y\_pred)

c

# Print the classification report

from sklearn.metrics import classification\_report

print(classification\_report(y\_test,y\_pred))

#Evaluation Parameters

from sklearn.metrics import precision\_score,accuracy\_score,f1\_score,recall\_score

print("precision\_score : ",precision\_score(y\_test,y\_pred))

print("accuracy\_score : ",accuracy\_score(y\_test,y\_pred))

print("f1\_score",f1\_score(y\_test,y\_pred))

print("recall\_score",recall\_score(y\_test,y\_pred))

**Output:**

(614,)

LogisticRegression()

array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,

1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,

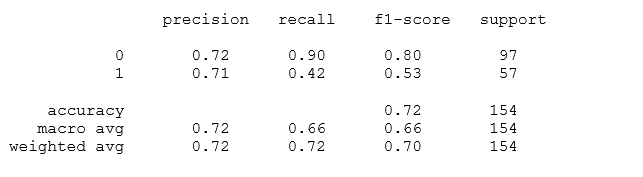
1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,

0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],

dtype=int64)

array([[87, 10],

[33, 24]], dtype=int64)



precision\_score : 0.7058823529411765

accuracy\_score : 0.7207792207792207

f1\_score 0.5274725274725274

recall\_score 0.42105263157894735

**AIM:** Write a program to demonstrate the working of the decision tree Regressor. Use appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.

**Description:**

Decision tree is a popular supervised machine learning algorithm used for both regression and classification tasks. It is a non-parametric model that works by recursively partitioning the data into smaller subsets based on the values of the input features. At each step of the partitioning process, the algorithm selects the feature that provides the most information gain, which is the measure of how much the feature contributes to reducing the impurity or uncertainty in the data.

The decision tree consists of a tree-like structure, where each internal node represents a decision based on the value of a feature, and each leaf node represents the output class or the predicted value. The goal of the algorithm is to create a tree that can classify or predict the output variable accurately.

The decision tree algorithm has several advantages, such as being easy to interpret, able to handle both categorical and continuous input features, and able to capture non-linear relationships between the features and the output variable. However, decision trees can be prone to overfitting, especially when the tree is too deep or when there are noisy or irrelevant features in the data.

There are several variations of the decision tree algorithm, including:

1. Classification Tree: A decision tree used for classification tasks, where the output variable is a categorical variable.

2. Regression Tree: A decision tree used for regression tasks, where the output variable is a continuous variable.

3. Random Forest: A variation of decision tree algorithm that creates an ensemble of decision trees and aggregates their outputs to improve the overall performance and reduce overfitting.

4. Gradient Boosting Tree: A variation of decision tree algorithm that creates an ensemble of decision trees sequentially, where each subsequent tree is trained to correct the errors of the previous tree.

The performance of a decision tree model can be evaluated using various metrics, including accuracy, precision, recall, F1 score, and mean squared error (MSE) for regression tasks. The choice of evaluation metric depends on the specific problem being addressed and the goals of the model.

**There are several algorithms that can be used to create a decision tree. Some of the most popular algorithms are:**

1. ID3 (Iterative Dichotomiser 3): ID3 is a classic decision tree algorithm that was introduced by Ross Quinlan in 1986. The algorithm works by recursively selecting the feature that provides the most information gain and partitioning the data into subsets based on the values of that feature.

2. C4.5: C4.5 is an improved version of the ID3 algorithm that was introduced by Ross Quinlan in 1993. The algorithm works by using the concept of gain ratio instead of information gain to select the feature to split on.

3. CART (Classification and Regression Trees): CART is a decision tree algorithm that was introduced by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone in 1984. The algorithm works by recursively selecting the feature that provides the most reduction in impurity, which can be measured using either the Gini index or entropy.

4. CHAID (Chi-square Automatic Interaction Detection): CHAID is a decision tree algorithm that was introduced by Kass in 1980. The algorithm works by recursively selecting the feature that provides the most significant association with the output variable, which can be measured using the chi-square test.

5. MARS (Multivariate Adaptive Regression Splines): MARS is a decision tree algorithm that was introduced by Jerome Friedman in 1991. The algorithm works by recursively splitting the data into subsets based on the values of the input features and fitting a linear regression model in each subset.

**Criterion for creating a Decision Tree:**

Entropy, Gini Impurity, and Information Gain are all measures used to determine the quality of a split in a decision tree. Here is an overview of each measure:

1. Entropy: Entropy is a measure of the impurity of a set of examples. It is calculated as the sum of the negative logarithm of the probability of each class in the set, multiplied by the probability of that class. Entropy is maximum when all classes are equally likely, and minimum when the set contains only examples of a single class.

Entropy (H): H(S) = -∑(i=1 to n) p(i) \* log2 p(i)

where S is the set of examples, n is the number of classes, and p(i) is the probability of class i in the set S.

1. Gini Impurity: Gini Impurity is a measure of the probability of misclassifying a randomly chosen example in a set. It is calculated as the sum of the probabilities of each class squared. Gini Impurity is maximum when all classes are equally likely, and minimum when the set contains only examples of a single class.

Gini Impurity (G):

G(S) = 1 - ∑(i=1 to n) [p(i)]^2

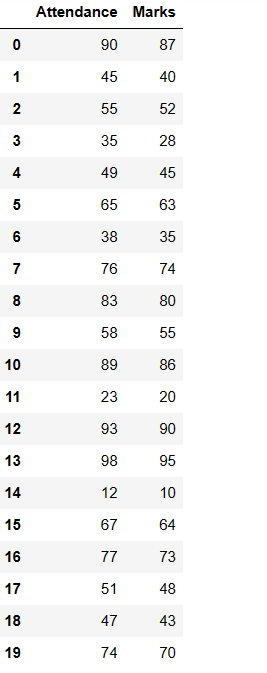
where S is the set of examples, n is the number of classes, and p(i) is the probability of class i in the set S.

1. Information Gain: Information Gain is the reduction in entropy or Gini Impurity achieved by splitting a set of examples based on a specific attribute. It is calculated as the difference between the entropy or Gini Impurity of the original set and the weighted average of the entropy or Gini Impurity of the subsets resulting from the split.

Information Gain (IG):

IG(S, A) = H(S) - ∑(v ∈ Values(A)) [(|Sv| / |S|) \* H(Sv)]

**Dataset:**



**Python Program:**

import pandas as pd

data=pd.read\_csv("D:/21761A4241/marks.csv")

x=data[['Attendance']]

y=data['Marks']

from sklearn.model\_selection import train\_test\_split

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.2)

print(xtrain.shape)

print(xtest.shape)

print(ytrain.shape)

print(ytest.shape)

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

model.fit(xtrain,ytrain)

pred=model.predict(xtest)

print(pred)

#from sklearn.tree import export\_graphviz

import matplotlib.pyplot as plt

from sklearn import tree

#fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)

tree.plot\_tree(model,feature\_names =['Attendance'],class\_names=['Marks'],filled=True)

plt.show('imagename.png')

**Output:**

(16, 1)

(4, 1)

(16,)

(4,)

array([48., 40., 86., 73.])



**AIM:**

Write a program to demonstrate the working of the decision tree classifier. Use appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.

**Description:**

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

**Pseudo code :**

* 1. Find the best attribute and place it on the root node of the tree.
  2. Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.
  3. Find leaf nodes in all branches by repeating 1 and 2 on each subset.

While implementing the decision tree we will go through the following two phases:

* 1. Building Phase
     + Preprocess the dataset.
     + Split the dataset from train and test using Python sklearn package.
     + Train the classifier.
  2. Operational Phase
     + Make predictions.
     + Calculate the accuracy.

**Syntax:-**

*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)*

**Parameters:-**

1. **criterion*{“gini”, “entropy”, “log\_loss”}, default=”gini”***

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “log\_loss” and “entropy” both for the Shannon information gain, see [Mathematical formulation](https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation).

1. **splitter*{“best”, “random”}, default=”best”***

The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

1. **max\_depth*int, default=None***

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

1. **min\_samples\_split*int or float, default=2***

The minimum number of samples required to split an internal node:

* If int, then consider min\_samples\_split as the minimum number.
* If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.

*Changed in version 0.18:*Added float values for fractions.

1. **min\_samples\_leaf*int or float, default=1***

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

* If int, then consider min\_samples\_leaf as the minimum number.
* If float, then min\_samples\_leaf is a fraction and ceil(min\_samples\_leaf \* n\_samples) are the minimum number of samples for each node.

*Changed in version 0.18:*Added float values for fractions.

1. **min\_weight\_fraction\_leaf*float, default=0.0***

The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample\_weight is not provided.

1. **max\_features*int, float or {“auto”, “sqrt”, “log2”}, default=None***

The number of features to consider when looking for the best split:

* If int, then consider max\_features features at each split.
* If float, then max\_features is a fraction and max(1, int(max\_features \* n\_features\_in\_)) features are considered at each split.
* If “auto”, then max\_features=sqrt(n\_features).
* If “sqrt”, then max\_features=sqrt(n\_features).
* If “log2”, then max\_features=log2(n\_features).
* If None, then max\_features=n\_features.

*Deprecated since version 1.1:*The "auto" option was deprecated in 1.1 and will be removed in 1.3.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.

1. **random\_state*int, RandomState instance or None, default=None***

Controls the randomness of the estimator. The features are always randomly permuted at each split, even if splitter is set to "best". When max\_features < n\_features, the algorithm will select max\_features at random at each split before finding the best split among them. But the best found split may vary across different runs, even if max\_features=n\_features. That is the case, if the improvement of the criterion is identical for several splits and one split has to be selected at random. To obtain a deterministic behaviour during fitting, random\_state has to be fixed to an integer. See [Glossary](https://scikit-learn.org/stable/glossary.html#term-random_state) for details.

1. **max\_leaf\_nodes*int, default=None***

Grow a tree with max\_leaf\_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

1. **min\_impurity\_decrease*float, default=0.0***

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

N\_t / N \* (impurity - N\_t\_R / N\_t \* right\_impurity

- N\_t\_L / N\_t \* left\_impurity)

where N is the total number of samples, N\_t is the number of samples at the current node, N\_t\_L is the number of samples in the left child, and N\_t\_R is the number of samples in the right child.

N, N\_t, N\_t\_R and N\_t\_L all refer to the weighted sum, if sample\_weight is passed.

*New in version 0.19.*

1. **class\_weight*dict, list of dict or “balanced”, default=None***

Weights associated with classes in the form {class\_label: weight}. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y))

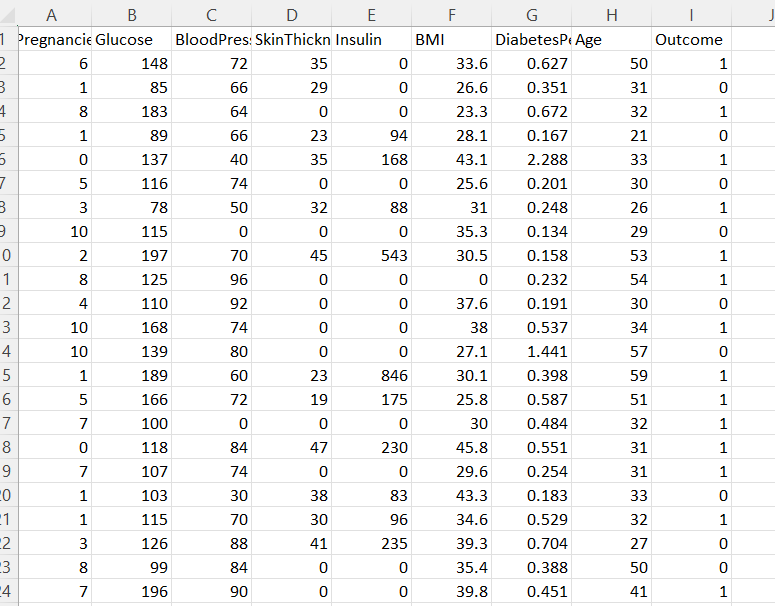
For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample\_weight (passed through the fit method) if sample\_weight is specified.

1. **ccp\_alpha*non-negative float, default=0.0***

Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp\_alpha will be chosen. By default, no pruning is performed.

**Dataset:**



**Python Program:**

import pandas as pd

data=pd.read\_csv("C:/Users/ML Lab/Downloads/diabetes.csv")

x=data.drop(["Outcome"],axis=1)

y=data['Outcome']

print(x)

from sklearn.model\_selection import train\_test\_split

xtrain, xtest, ytrain, ytest= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

print(xtrain.shape)

print(xtest.shape)

print(ytrain.shape)

print(ytest.shape

from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

xtrain= st\_x.fit\_transform(xtrain)

xtest= st\_x.transform(xtest)

from sklearn.tree import DecisionTreeClassifier

classifier= DecisionTreeClassifier()

classifier.fit(xtrain, ytrain)

ypred=classifier.predict(xtest)

print(ypred)

from sklearn.metrics import confusion\_matrix

cm=confusion\_matrix(ytest,ypred)

print(cm)

from sklearn.metrics import accuracy\_score

acc=accuracy\_score(ytest,ypred)

acc

0.7135416666666666

#from sklearn.tree import export\_graphviz

import matplotlib.pyplot as plt

from sklearn import tree

#fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)

tree.plot\_tree(model,feature\_names =['BMI'],class\_names=['Outcome'],filled=True)

plt.show('imagename1.png')

**Output:**

(16, 1)

(4, 1)

(16,)

(4,)

array([48., 40., 86., 73.])



**AIM :**

Write a program to demonstrate the working of Random Forest classifier. Use appropriate dataset for Random Forest Classifier.

## **Introduction:**

Random Forest is a supervised machine learning algorithm which is based on ensemble learning. In this kernel, I build two Random Forest Classifier models to predict the safety of the car, one with 10 decision-trees and another one with 100 decision-trees. The expected accuracy increases with number of decision-trees in the model. I have demonstrated the **feature selection process** using the Random Forest model to find only the important features, rebuild the model using these features and see its effect on accuracy.

Random forest is a supervised learning algorithm. It has two variations – one is used for classification problems and other is used for regression problems. It is one of the most flexible and easy to use algorithm. It creates decision trees on the given data samples, gets prediction from each tree and selects the best solution by means of voting. It is also a pretty good indicator of feature importance.

Random forest algorithm combines multiple decision-trees, resulting in a forest of trees, hence the name Random Forest. In the random forest classifier, the higher the number of trees in the forest results in higher accuracy.

Random forest algorithm intuition can be divided into two stages.

In the first stage, we randomly select “k” features out of total m features and build the random forest. In the first stage, we proceed as follows:-

1. Randomly select k features from a total of m features where k < m.
2. Among the k features, calculate the node d using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat 1 to 3 steps until l number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for n number of times to create n number of trees.

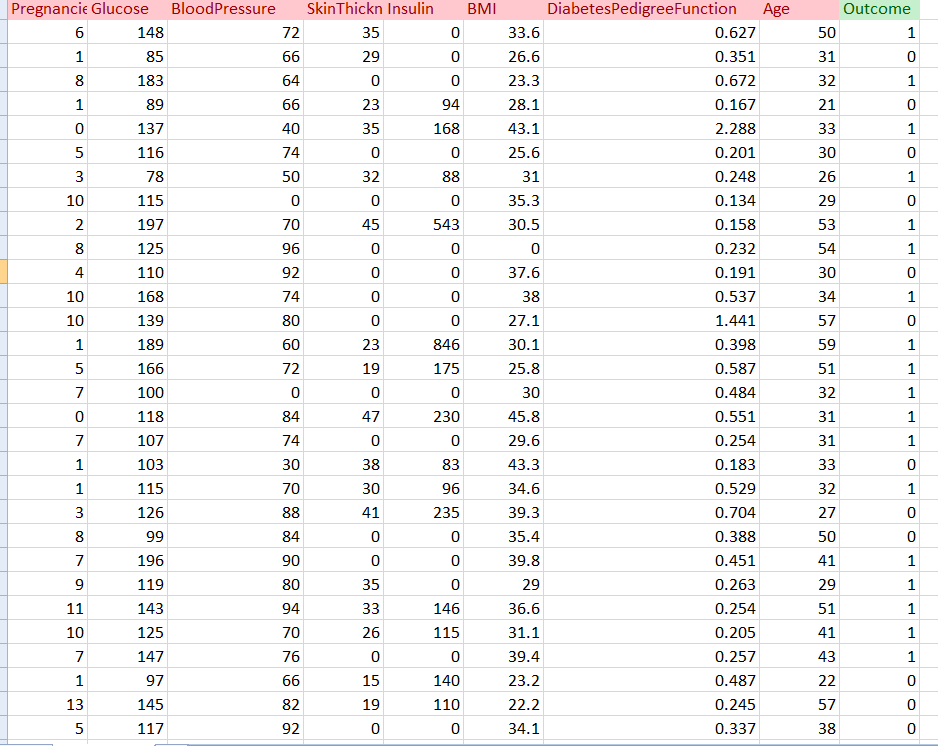
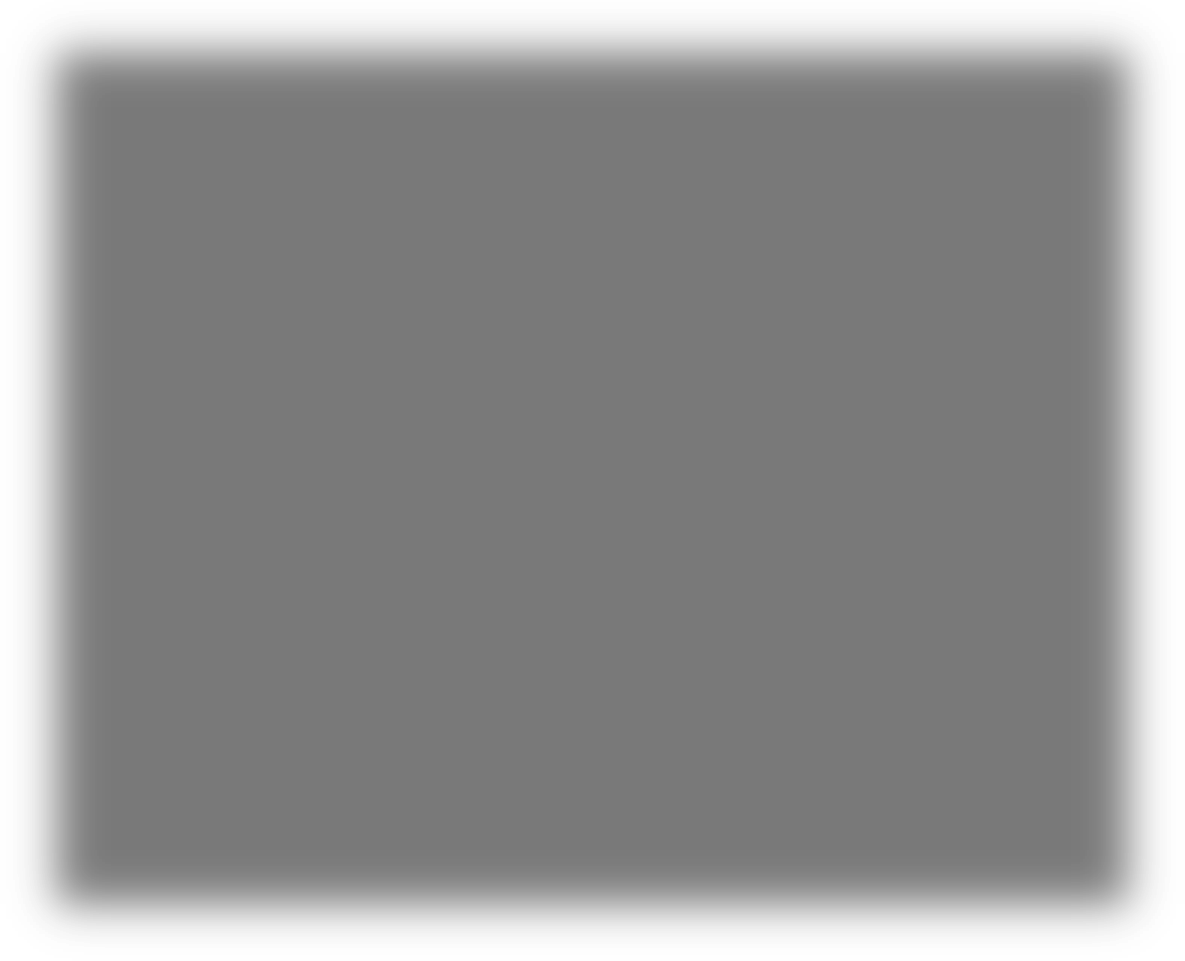
In the second stage, we make predictions using the trained random forest algorithm.

1. We take the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome.
2. Then, we calculate the votes for each predicted target.
3. Finally, we consider the high voted predicted target as the final prediction from the random forest algorithm.

### **Random Forest algorithm intuition**



**Dataset:**



**Pythoncode:**

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

df = pd.read\_csv("diabetes.csv")

X = df.drop("Outcome", axis=1)

y = df["Outcome"]

data.corr()



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

**O/P;**

(576, 8)

(192, 8)

(576,)

(192,)

classifier = RandomForestClassifier(n\_estimators=100, random\_state=0)

classifier.fit(X\_train, y\_train)

**O/P:**

RandomForestClassifier()

y\_pred = classifier.predict(X\_test)

print(y\_pred)

[1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0

0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1

0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0

0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1

1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0

0 0 0 1 0 0 0 1 0]

cm = confusion\_matrix(y\_test, y\_pred)   
print("Confusion Matrix:")

cm

**O/P:**

Confusion Matrix:

Out[22]:

array([[115, 12],

[ 31, 34]], dtype=int64)

from sklearn.metrics import classification\_report

res = classification\_report(y\_test, y\_pred)

print("\nClassification Report:\n", res)

**O/P:**

Classification Report:

precision recall f1-score support

0 0.79 0.91 0.84 127

1 0.74 0.52 0.61 65

accuracy 0.78 192

macro avg 0.76 0.71 0.73 192

weighted avg 0.77 0.78 0.76 192

**Create the heatmap using seaborn**

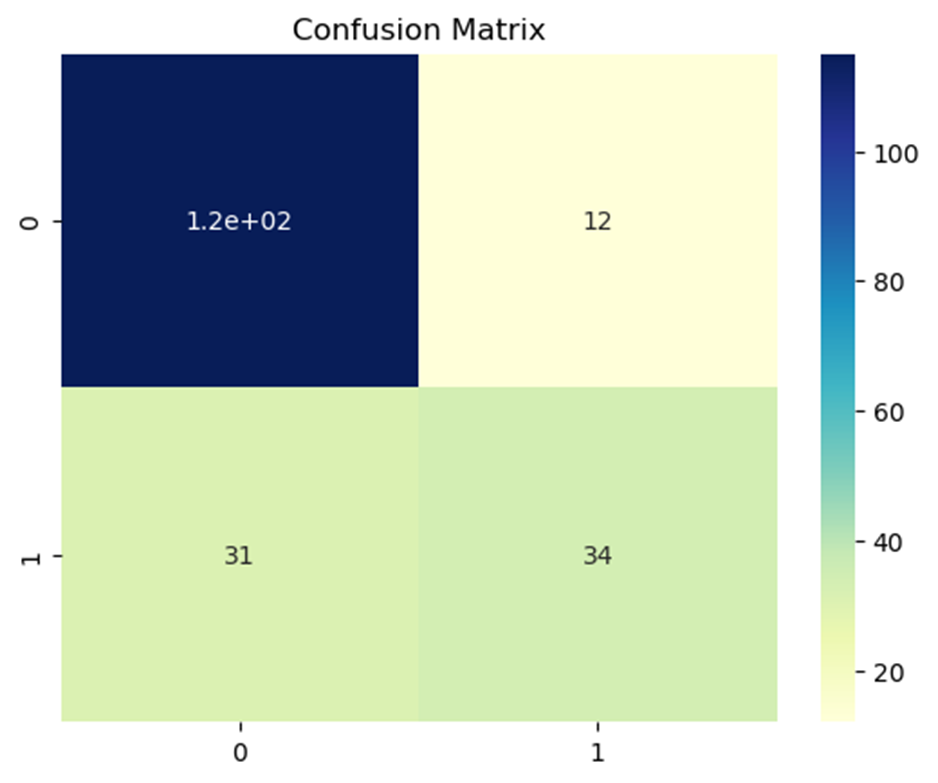
import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

sns.heatmap(cm,annot=True,cmap="YlGnBu")

plt.title("Confusion Matrix")

****

**Evaluation Parameters**

from sklearn.metrics import precision\_score,recall\_score, f1\_score, accuracy\_score,confusion\_matrix

print("Accuracy:", accuracy\_score(y\_test,y\_pred))

Accuracy: 0.7760416666666666

print("Precision:", precision\_score(y\_test,y\_pred,average="weighted"))

Precision: 0.7712381501886044

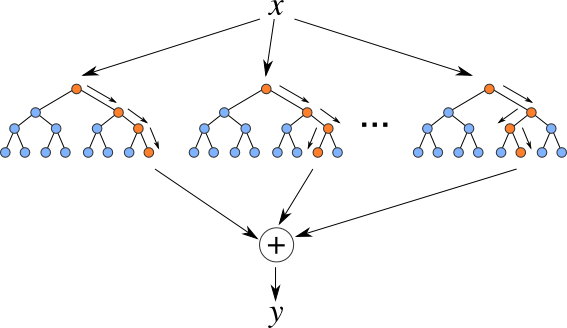
print('Recall:', recall\_score(y\_test,y\_pred,average="weighted"))

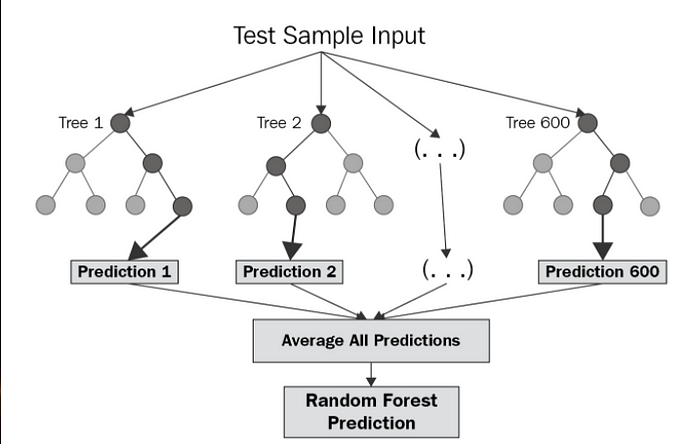
Recall: 0.776041666666666

**Aim: Write a program to demonstrate the working of Random Forest Regressor. Use appropriate dataset for Random Forest Regressor**

## **Introduction:**

# **Random Forest Regressor:**



**Random Forest Regression** is a supervised learning algorithm that uses **ensemble learning** method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model. 

The diagram above shows the structure of a Random Forest. You can notice that the trees run in parallel with no interaction amongst them. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees. To get a better understanding of the Random Forest algorithm, let’s walk through the steps:

1. Pick at random k data points from the training set.
2. Build a decision tree associated to these k data points.
3. Choose the number N of trees you want to build and repeat steps 1 and 2.
4. For a new data point, make each one of your N-tree trees predict the value of y for the data point in question and assign the new data point to the average across all of the predicted y values.

A Random Forest Regression model is powerful and accurate. It usually performs great on many problems, including features with non-linear relationships. Disadvantages, however, include the following: there is no interpretability, overfitting may easily occur, we must choose the number of trees to include in the model.

# Let’s see Random Forest Regression in action!

Now that we have a basic understanding of how the Random Forest Regression model works, we can assess its performance on a real-world dataset. Similar to my previous posts, I will be using data on House Sales in King County, USA.

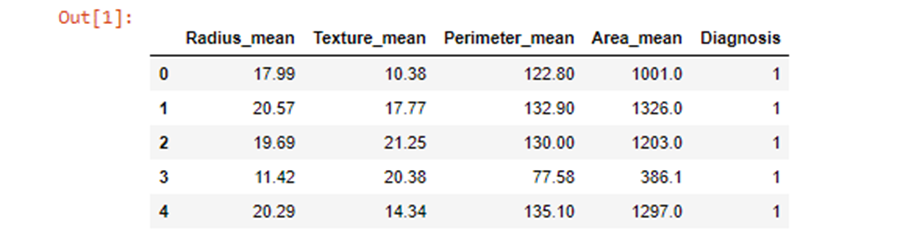
After importing the libraries, importing the dataset, addressing null values, and dropping any necessary columns, we are ready to create our Random Forest Regression model!

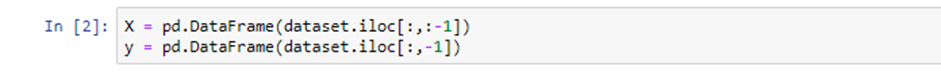
**Algorithm:**

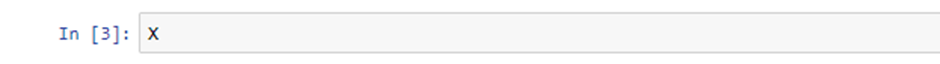
* Design a specific question or data and get the source to determine the required data.
* Make sure the data is in an accessible format else convert it to the required format.
* Specify all noticeable anomalies and missing data points that may be required to achieve the required data.
* Create a machine-learning model.
* Set the baseline model that you want to achieve
* Train the data machine learning model.
* Provide an insight into the model with test data
* Now compare the performance metrics of both the test data and the predicted data from the model.
* If it doesn’t satisfy your expectations, you can try improving your model accordingly or dating your data, or using another data modeling technique.
* At this stage, you interpret the data you have gained and report accordingly.

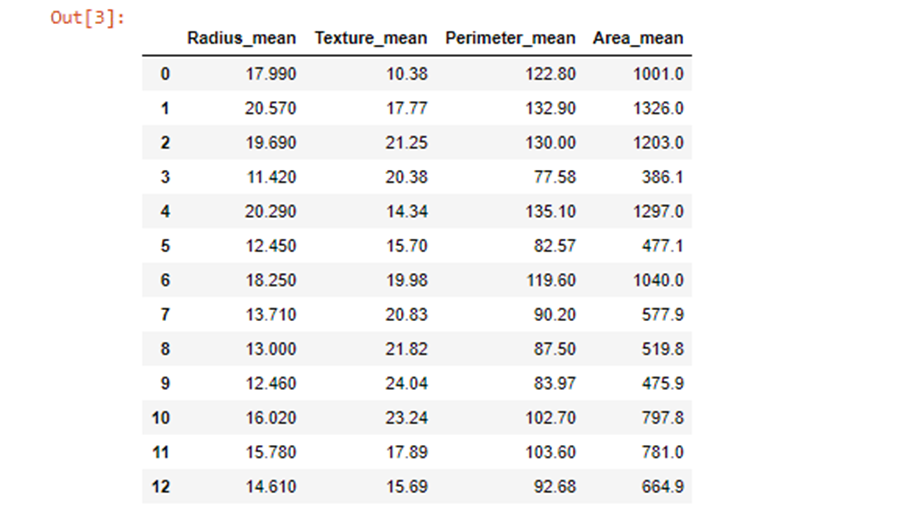
**Python code:**

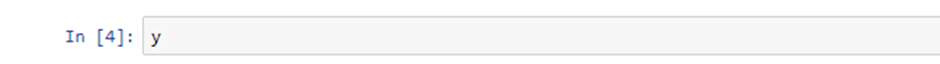
****

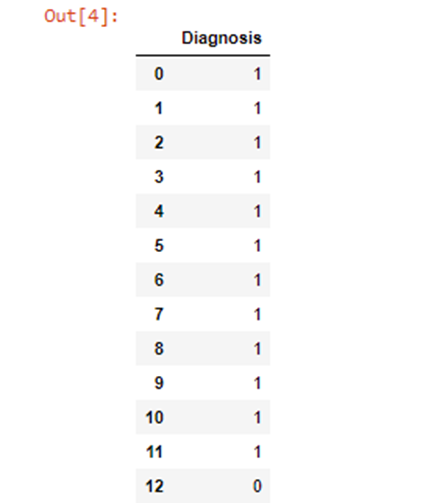


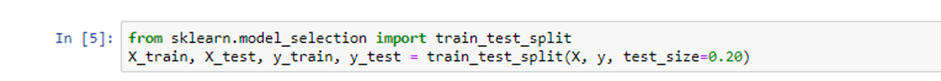


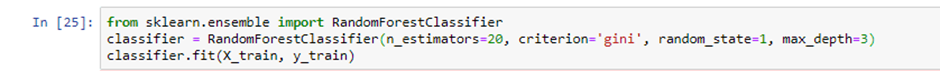




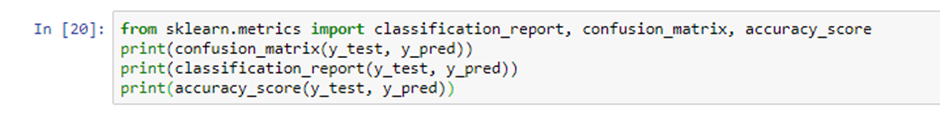


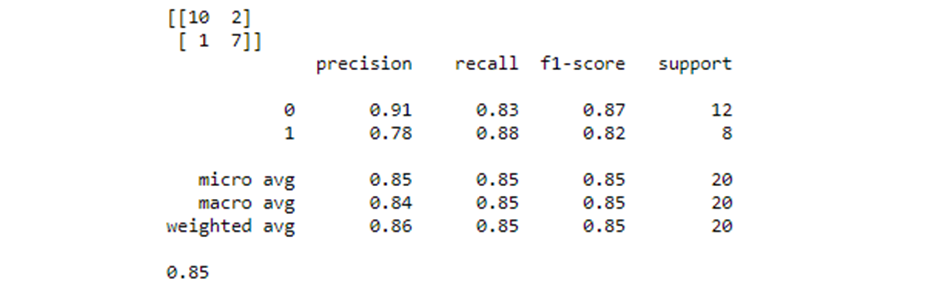




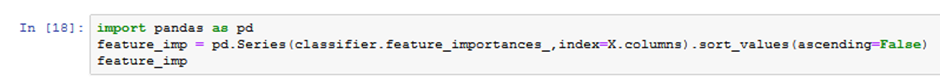


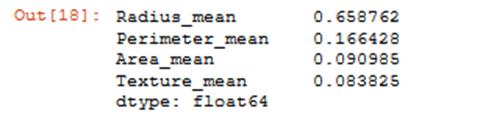


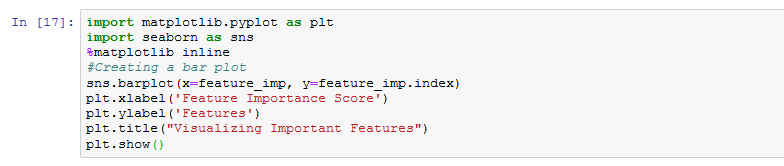


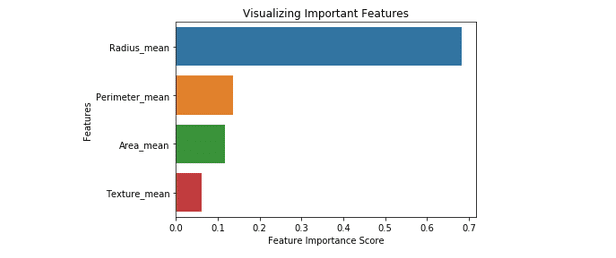


**.Let us find out important features and visualize them using Seaborn**

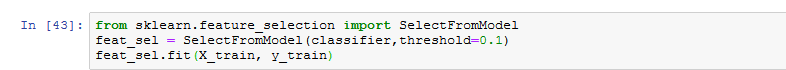


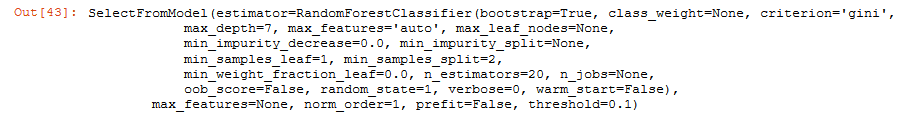


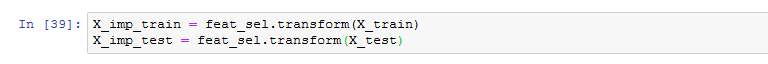




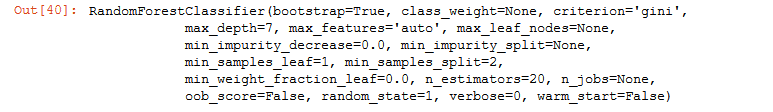
**Import the SelectFromModel function. We will pass the classifier object we’ve created above.**

















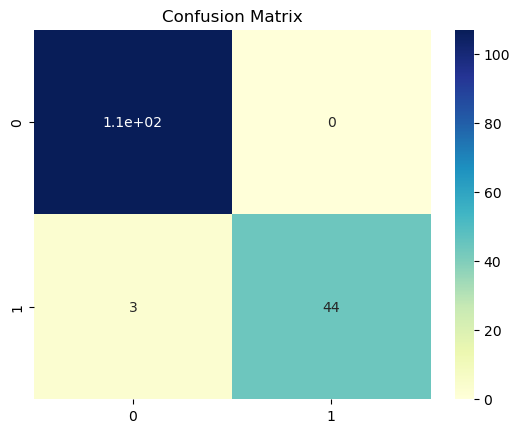


**import** seaborn **as** sns

sns**.**heatmap(y,annot**=True**,cmap**=**"YlGnBu")

plt**.**title("Confusion Matrix")

Out[72]:

Text(0.5, 1.0, 'Confusion Matrix') 

**ARTIFICIAL INTELLIGENCE**

**AIM: Implementation of DFS for water jug problem:**

## **Introduction:**

# **DFS Algorithm:**

Depth first search is another way of traversing graphs, which is closely related to preorder traversal of a tree. Recall that preorder traversal simply visits each node before its children. It is most easy to program as a recursive routine:

    preorder(node v)

    {

    visit(v);

    for each child w of v

        preorder(w);

    }

To turn this into a graph traversal algorithm, we basically replace “child” by “neighbour”. But to prevent infinite loops, we only want to visit each vertex once. Just like in BFS we can use marks to keep track of the vertices that have already been visited, and not visit them again. Also, just like in BFS, we can use this search to build a spanning tree with certain useful properties.

    dfs(vertex v)

    {

    visit(v);

    for each neighbour w of v

        if w is unvisited

        {

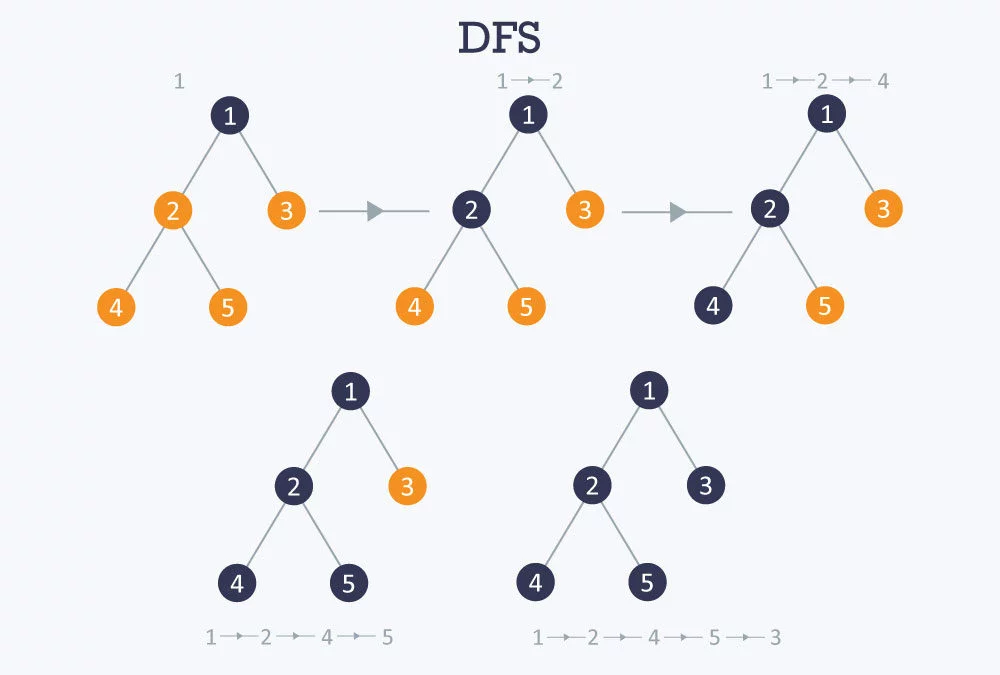
        dfs(w);

        add edge vw to tree T

        }

    }

The overall depth first search algorithm then simply initializes a set of markers so we can tell which vertices are visited, chooses a starting vertex x, initializes tree T to x, and calls dfs(x). Just like in breadth first search, if a vertex has several neighbours it would be equally correct to go through them in any order. I didn’t simply say “for each unvisited neighbour of v” because it is very important to delay the test for whether a vertex is visited until the recursive calls for previous neighbours are finished.



**Problem:** There are two jugs of **volume A litre** and **B litre.** Neither has any **measuring mark** on it.There is a pump that can be used to fill the jugs with water.How can you get exactly**x litre** of water into the **A litre jug.**Assuming that we have unlimited supply of water.

**Note:**Let's assume we have **A=4 litre and B= 3 litre jugs**. And **we want exactly 2 Litre water into jug A (i.e 4 litre jug)** how we will do this.

**Solution:**

The state space for this problem can be described as the set of ordered **pairs of integers (x,y)**

Where,

x represents the quantity of  water in the 4-gallon jug  x= 0,1,2,3,4

y represents the quantity of water in 3-gallon jug y=0,1,2,3

**Start State: (0,0)**

**Goal State: (2,0)**

Generate production rules for the water jug problem

We basically perform three operations to achieve the goal.

1. **Fill water jug.**
2. **Empty water jug**
3. and**Transfer water jug**

| **Rule** | **State** | **Process** |
| --- | --- | --- |
| **1** | (X,Y | X<4) | (4,Y)  {Fill 4-gallon jug} |
| **2** | (X,Y |Y<3) | (X,3)  {Fill 3-gallon jug} |
| **3** | (X,Y |X>0) | (0,Y)  {Empty 4-gallon jug} |
| **4** | (X,Y | Y>0) | (X,0)  {Empty 3-gallon jug} |
| **5** | (X,Y | X+Y>=4 ^ Y>0) | (4,Y-(4-X))  {Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full} |
| **6** | (X,Y | X+Y>=3 ^X>0) | (X-(3-Y),3)  {Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full} |
| **7** | (X,Y | X+Y<=4 ^Y>0) | (X+Y,0)  {Pour all water from 3-gallon jug into 4-gallon jug} |
| **8** | (X,Y | X+Y <=3^ X>0) | (0,X+Y)  {Pour all water from 4-gallon jug into 3-gallon jug} |
| **9** | (0,2) | (2,0)  {Pour 2 gallon water from 3 gallon jug into 4 gallon jug} |

**Initialization:**

Start State: (0,0)

**Apply Rule 2:**

Fill 3-gallon jug  
Now the state is (x,3)

**Iteration 1:**

Current State: (x,3)

**Apply Rule 7:**

Pour all water from 3-gallon jug into 4-gallon jug

Now the state is (3,0)

**Iteration 2:**

Current State : (3,0)

**Apply Rule 2:**

Fill 3-gallon jug  
Now the state is (3,3)

**Iteration 3:**

Current State:(3,3)

**Apply Rule 5:**

Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full

Now the state is (4,2)

**Iteration 4:**

Current State : (4,2)

**Apply Rule 3:**

Empty 4-gallon jug  
Now state is (0,2)

**Iteration 5:**

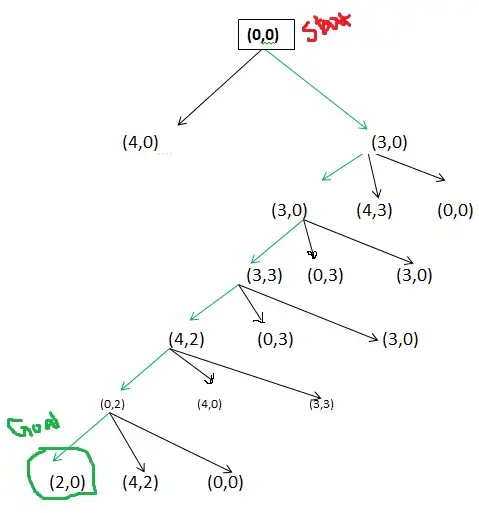
Current State : (0,2)

**Apply Rule 9:**

Pour 2 gallon water from 3 gallon jug into 4 gallon jug

**Now the state is (2,0)-- Goal Achieved.**

**Water Jug Solution using DFS (Depth First Search)**



**Python code:**

print("Water Jug Problem")

x,y = 0,0

c = int(input("Enter te goal state:"))

while True:

    r = int(input("Enter the rule no:"))

    if r == 1:

        if x < 4:

            x,y = 4,y

    if r == 2:

        if y <3:

            x,y = x,3

    if r == 3:

        if x > 0:

            x,y = 0,y

    if r == 4:

        if y > 0:

            x,y = x,0

    if r == 5:

        if y>0 and 0 < x+y >= 4:

            x,y = 4,(y-(4-x))

    if r == 6:

        if 0 < x+y >= 3 and x>0:

            x,y = (x-(3-y)),3

    if r == 7:

        if 0 <x+y <= 4 and y>=0:

            x,y = (x+y),0

    if r == 8:

        if 0 <= x+y <=3 and x>=0:

            x,y = 0,x+y

    print("current state is : (%d,%d)"%(x,y))

    if x == c:

        print("Goal State is reached")

        break

**output:**

Water Jug Problem

Enter te goal state:2

Enter the rule no:1

current state is : (4,0)

Enter the rule no:6

current state is : (1,3)

Enter the rule no:4

current state is : (1,0)

Enter the rule no:8

current state is : (0,1)

Enter the rule no:1

current state is : (4,1)

Enter the rule no:6

current state is : (2,3)

Goal State is reached

**water jug problem Using Java:**

import java.util.\*;

public class Main

{

    public static void main(String[] args)

    {

            int x=0,y=0,g,c1,c2,r,c;

            Scanner read = new Scanner(System.in);

            System.out.println("Enter the capacity of jug 1:");

            c1 = read.nextInt();

            System.out.println("Enter the capacity of jug 2:");

            c2 = read.nextInt();

            System.out.println("Enter the goal state:");

            c = read.nextInt();

            while(true)

            {

                System.out.println("Enter the Rule:");

                r = read.nextInt();

                if(r == 1)

                {

                    if(x < c1)

                    {

                        x = c1;

                    }

                }

                if(r == 2)

                {

                    if(y < c2)

                    {

                        y = c2;

                    }

                }

                if(r == 3)

                {

                    if(x > 0)

                    {

                        x = 0;

                    }

                }

                if(r == 4)

                {

                    if(y > 0)

                    {

                        y = 0;

                    }

                }

                if(r == 5)

                {

                    if((0 < x+y && x+y >= c1) && y >0)

                    {

                        y = y - (c1 - x);

                        x = c1;

                    }

                }

                if(r == 6)

                {

                    if(((0 < x+y) && (x+y >= c2)) && x >0)

                    {

                        x = x - (c2 - y);

                        y = c2;

                    }

                }

                if(r == 7)

                {

                    if((0 < x+y && x+y <= c1) && y >=0)

                    {

                        x = x+y;

                        y = 0;

                    }

                }

                if(r == 8)

                {

                    if((0 < x+y && x+y <= c1) && x >= 0)

                    {

                        y = x+y;

                        x = 0;

                    }

                }

                System.out.println("Current state is :"+"("+x+","+y+")");

                if(x == c)

                {

                    System.out.println("Goal State is reached");

                    break;

                }

            }

    }

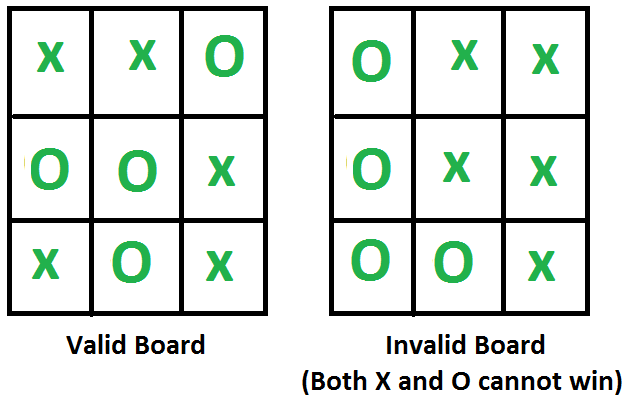
}

**Aim: Implement and demonstrate the Tic-Tac-Toe problem in python code**.

## **Introduction:**

A Tic-Tac-Toe board is given after some moves are played. Find out if the given board is valid, i.e., is it possible to reach this board position after some moves or not.  
Note that every arbitrary filled grid of 9 spaces isn’t valid e.g. a grid filled with 3 X and 6 O isn’t valid situation because each player needs to take alternate turns.

**Steps**  
1. Identify the outcomes and instructional focus of a unit of study.  
2. Use assessment data and student profiles to determine student readiness, learning styles, or interests.3. Design nine different tasks.  
4. Arrange the tasks on a choice board.  
5. Select one required task for all students. Place it in the center of the board.  
6. Students complete three tasks, one of which must be the task in the middle square.  
The three tasks should complete a Tic-Tac-Toe row.



Input is given as a 1D array of size 9.

**Examples:**

Input: board[] = {'X', 'X', 'O',

'O', 'O', 'X',

'X', 'O', 'X'};

Output: Valid

Input: board[] = {'O', 'X', 'X',

'O', 'X', 'X',

'O', 'O', 'X'};

Output: Invalid

(Both X and O cannot win)

Input: board[] = {'O', 'X', ' ',

' ', ' ', ' ',

' ', ' ', ' '};

Output: Valid

Basically, to find the validity of an input grid, we can think of the conditions when an input grid is invalid. Let no. of “X”s be countX and no. of “O”s be countO. Since we know that the game starts with X, a given grid of Tic-Tac-Toe game would be definitely invalid if following two conditions meet

1. countX != countO AND
2. countX != countO + 1
   * Since “X” is always the first move, second condition is also required.
   * Now does it mean that all the remaining board positions are valid one? The answer is NO. Think of the cases when input grid is such that both X and O are making straight lines. This is also not
   * valid position because the game ends when one player wins. So we need to check the following condition as well
3. If input grid shows that both the players are in winning situation, it’s an invalid position.
4. If input grid shows that the player with O has put a straight-line (i.e. is in win condition) and countX != countO, it’s an invalid position. The reason is that O plays his move only after X plays his
   * move. Since X has started the game, O would win when both X and O has played equal no. of moves.
5. If input grid shows that X is in winning condition than xCount must be one greater that oCount.
   * Armed with above conditions i.e. a), b), c) and d), we can now easily formulate an algorithm/program to check the validity of a given Tic-Tac-Toe board position.

**Tic-tac-Toe using python code:**

tic = [[1,2,3,],

       [4,5,6],

       [7,8,9]]

p=[0,2,3]

p[1] = input("Enter player 1 name = ")

p[2] = input("Enter player 2 name = ")

print("Tic Tac Toe Board:")

for i in tic:

    print(i)

#insertion function

def insertAt(place,i):

    for j in range(len(tic)):

        for k in range(len(tic[j])):

            if tic[j][k] == place:

                if i == 1:

                    tic[j][k] = "X"

                else:

                    tic[j][k] = "O"

#verification of places

def verify(i):

    if ((1 and 2 and 3) not in tic[0]) and ((4 and 5 and 6) not in tic[1]) and ((7 and 8 and 9) not in tic[2]):

        print("Draw!! Try again")

        exit(0)

    #horizantal lines

    elif tic[0][0] == tic[0][1] == tic[0][2]:

        print(f"Hurrah! player {p[i]} won the match")

        exit(0)

    elif tic[1][0] == tic[1][1] == tic[1][2]:

        print(f"Hurrah! player {p[i]} won the match")

        exit(0)

    elif tic[2][0] == tic[2][1] == tic[2][2]:

        print(f"Hurrah! player {p[i]} won the match")

        exit(0)

    #Diagonal lines

    elif tic[0][0] == tic[1][1] == tic[2][2]:

        print(f"Hurrah! player {p[i]} won the match")

        exit(0)

    elif tic[0][2] == tic[1][1] == tic[2][0]:

        print(f"Hurrah! player {p[i]} won the match")

        exit(0)

    #vertical lines

    elif tic[0][0] == tic[1][0] == tic[1][0]:

        print(f"Hurrah! player {p[i]} won the match")

        exit(0)

    elif tic[0][1] == tic[1][1] == tic[2][1]:

        print(f"Hurrah! player {p[i]} won the match")

        exit(0)

    elif tic[0][2] == tic[1][2] == tic[2][2]:

        print(f"Hurrah! {p[i]} won the match")

        exit(0)

#Displaying the tic-tac-toe panel

def disp():

    for i in tic:

        print(i)

i=1

for itr in range(9):

    place = int(input(f"{p[i]},Enter a position = "))

    if i == 1:

        insertAt(place,i)

        disp()

        verify(i)

        i=2

    else:

        insertAt(place,i)

        disp()

        verify(i)

        i=1

**output:**

Enter player 1 name = ravi

Enter player 2 name = siva

Tic Tac Toe Board:

[1, 2, 3]

[4, 5, 6]

[7, 8, 9]

ravi,Enter a position = 1

['X', 2, 3]

[4, 5, 6]

[7, 8, 9]

siva,Enter a position = 5

['X', 2, 3]

[4, 'O', 6]

[7, 8, 9]

ravi,Enter a position = 3

['X', 2, 'X']

[4, 'O', 6]

[7, 8, 9]

siva,Enter a position = 7

['X', 2, 'X']

[4, 'O', 6]

['O', 8, 9]

ravi,Enter a position = 6

['X', 2, 'X']

[4, 'O', 'X']

['O', 8, 9]

siva,Enter a position = 2

['X', 'O', 'X']

[4, 'O', 'X']

['O', 8, 9]

ravi,Enter a position = 4

['X', 'O', 'X']

['X', 'O', 'X']

['O', 8, 9]

Hurrah! player ravi won the match

# **AIM:** Introduction Of Prolog

## **Introduction:**

Prolog is a logic programming language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the **logic** being applied. Formulation or Computation is carried out by running a query over these relations.

**Syntax and Basic Fields :**

In prolog, We declare some facts. These facts constitute the Knowledge Base of the system. We can query against the Knowledge Base. We get output as affirmative if our query is already in the knowledge Base or it is implied by Knowledge Base, otherwise we get output as negative. So, Knowledge Base can be considered similar to database, against which we can query. Prolog facts are expressed in definite pattern. Facts contain entities and their relation. Entities are written within the parenthesis separated by comma (, ). Their relation is expressed at the start and outside the parenthesis. Every fact/rule ends with a dot (.). So, a typical prolog fact goes as follows :

Format : relation(entity1, entity2, ....k'th entity).

Example :

friends(raju, mahesh).

singer(sonu).

odd\_number(5).

Explanation :

These facts can be interpreted as :

raju and mahesh are friends.

sonu is a singer.

5 is an odd number.

**Key Features :**  
**1. Unification :** The basic idea is, can the given terms be made to represent the same structure.  
**2. Backtracking :** When a task fails, prolog traces backwards and tries to satisfy previous task.  
**3. Recursion :** Recursion is the basis for any search in program.

**Running queries :**  
A typical prolog query can be asked as :

Query 1 : ?- singer(sonu).

Output : Yes.

Explanation : As our knowledge base contains   
the above fact, so output was 'Yes', otherwise   
it would have been 'No'.

Query 2 : ?- odd\_number(7).

Output : No.

Explanation : As our knowledge base does not   
contain the above fact, so output was 'No'.

**Advantages :**  
**1.**Easy to build database. Doesn’t need a lot of programming effort.  
**2.**Pattern matching is easy. Search is recursion based.  
**3.**It has built in list handling. Makes it easier to play with any algorithm involving lists.

**Disadvantages :**  
**1.** LISP (another logic programming language) dominates over prolog with respect to I/O features.  
**2.** Sometimes input and output is not easy.

**Applications :**

Prolog is highly used in artificial intelligence(AI). Prolog is also used for pattern matching over natural language parse trees.

**Symbols:**

Prolog expressions are comprised of the following truth-functional symbols, which have the same interpretation as in the predicate calculus.

|  |  |  |
| --- | --- | --- |
| **English** | **Predicate Calculus** | **Prolog** |
| If | --> | :- |
| Not | ~ | Not |
| Or | V | ; |
| and | ^ | , |

**Queries:**In Prolog, the query is the action of asking the program about the information which is available within its database. When a Prolog program is loaded, we will get the query prompt,

1. ?-

After this, we can ask about the information to the run time system. Using the above simple database, we can ask a question to the program like

**?-‘it is summer’**.

and it will give the answer

1. yes
2. ?-

The system responds to the query with yes if the database information is consistent to answer the query. Using the available database information, we can also check that the program is capable of proving the query true. No indicates that the fact is not deducible based on the available information.

The system answers no to the query if the database does not have sufficient information.

1. **?-‘it is cold’**.
2. no
3. ?-

**Rules:**

Rules extend the logic program capabilities. Rules are used to provide the decision-making process in Prolog. Rules are specified in the form:

1. head:- t1, t2, t3,….., tk. Where k>=1

The head is known as the clause of the head.

:- is known as the clause neck. It is read as 'if'. The body of the clause is specified by t1, t2, t3,…, tk. It contains one or more components, and it can be separated using the commas. A rule will read as 'head is true if t1, t2, t3,…., tk are all true'.

In the following program, first two lines indicate the facts and last two lines indicate the rules:

1. dog(rottweiler).    large(rottweiler).
2. cat(siamese).         large(siamese).
3. large\_animal(A) :- dog(A),large(A).
4. large\_animal(C) :- cat(C),large(C).

The above rules mean that 'large\_animal(A) is true if dog(A) is true, and large(A) is true, etc.'

The last line means that 'large\_animal(C) is true if cat(C) is true, and large(C) is true.

**AIM:** **Implementation of Monkey Banana Problem using PROLOG**

## **Introduction:**

In this prolog example, we will see one very interesting and famous problem, The Monkey and Banana Problem.

## **Problem Statement**

Suppose the problem is as given below −

* A hungry monkey is in a room, and he is near the door.
* The monkey is on the floor.
* Bananas have been hung from the center of the ceiling of the room.
* There is a block (or chair) present in the room near the window.
* The monkey wants the banana, but cannot reach it.

****

**So how can the monkey get the bananas?**

So if the monkey is clever enough, he can come to the block, drag the block to the center, climb on it, and get the banana. Below are few observations in this case −

* Monkey can reach the block, if both of them are at the same level. From the above image, we can see that both the monkey and the block are on the floor.
* If the block position is not at the center, then monkey can drag it to the center.
* If monkey and the block both are on the floor, and block is at the center, then the monkey can climb up on the block. So the vertical position of the monkey will be changed.
* When the monkey is on the block, and block is at the center, then the monkey can get the bananas.

Now, let us see how we can solve this using Prolog. We will create some predicates as follows −

We have some predicates that will move from one state to another state, by performing action.

* When the block is at the middle, and monkey is on top of the block, and monkey does not have the banana (i.e. ***has not*** state), then using the ***grasp*** action, it will change from ***has not*** state to ***have*** state.
* From the floor, it can move to the top of the block (i.e. ***on top*** state), by performing the action ***climb***.
* The **push** or **drag** operation moves the block from one place to another.
* Monkey can move from one place to another using **walk** or **move** clauses.

Another predicate will be canget(). Here we pass a state, so this will perform move predicate from one state to another using different actions, then perform canget() on state 2. When we have reached to the state ‘**has>**’, this indicates ‘**has banana**’. We will stop the execution.

## **Program:**

move(state(middle,onbox,middle,hasnot),

grasp,

state(middle,onbox,middle,has)).

move(state(P,onfloor,P,H),

climb,

state(P,onbox,P,H)).

move(state(P1,onfloor,P1,H),

drag(P1,P2),

state(P2,onfloor,P2,H)).

move(state(P1,onfloor,B,H),

walk(P1,P2),

state(P2,onfloor,B,H)).

canget(state(\_,\_,\_,has)).

canget(State1) :-

move(State1,\_,State2),

canget(State2).

## **Output:**

?- canget(state(atdoor, onfloor, atwindow, hasnot)).

true ?

yes

| ?- trace

.

The debugger will first creep -- showing everything (trace)

yes

{trace}

| ?- canget(state(atdoor, onfloor, atwindow, hasnot)).

1 1 Call: canget(state(atdoor,onfloor,atwindow,hasnot)) ?

2 2 Call: move(state(atdoor,onfloor,atwindow,hasnot),\_52,\_92) ?

2 2 Exit:move(state(atdoor,onfloor,atwindow,hasnot),walk(atdoor,\_80),state(\_80,onfloor,atwindow,hasnot)) ?

3 2 Call: canget(state(\_80,onfloor,atwindow,hasnot)) ?

4 3 Call: move(state(\_80,onfloor,atwindow,hasnot),\_110,\_150) ?

4 3 Exit: move(state(atwindow,onfloor,atwindow,hasnot),climb,state(atwindow,onbox,atwindow,hasnot)) ?

5 3 Call: canget(state(atwindow,onbox,atwindow,hasnot)) ?

6 4 Call: move(state(atwindow,onbox,atwindow,hasnot),\_165,\_205) ?

6 4 Fail: move(state(atwindow,onbox,atwindow,hasnot),\_165,\_193) ?

5 3 Fail: canget(state(atwindow,onbox,atwindow,hasnot)) ?

4 3 Redo: move(state(atwindow,onfloor,atwindow,hasnot),climb,state(atwindow,onbox,atwindow,hasnot)) ?

4 3 Exit: move(state(atwindow,onfloor,atwindow,hasnot),drag(atwindow,\_138),state(\_138,onfloor,\_138,hasnot)) ?

5 3 Call: canget(state(\_138,onfloor,\_138,hasnot)) ?

6 4 Call: move(state(\_138,onfloor,\_138,hasnot),\_168,\_208) ?

6 4 Exit: move(state(\_138,onfloor,\_138,hasnot),climb,state(\_138,onbox,\_138,hasnot)) ?

7 4 Call: canget(state(\_138,onbox,\_138,hasnot)) ?

8 5 Call: move(state(\_138,onbox,\_138,hasnot),\_223,\_263) ?

8 5 Exit: move(state(middle,onbox,middle,hasnot),grasp,state(middle,onbox,middle,has)) ?

9 5 Call: canget(state(middle,onbox,middle,has)) ?

9 5 Exit: canget(state(middle,onbox,middle,has)) ?

7 4 Exit: canget(state(middle,onbox,middle,hasnot)) ?

5 3 Exit: canget(state(middle,onfloor,middle,hasnot)) ?

3 2 Exit: canget(state(atwindow,onfloor,atwindow,hasnot)) ?

1 1 Exit: canget(state(atdoor,onfloor,atwindow,hasnot)) ?

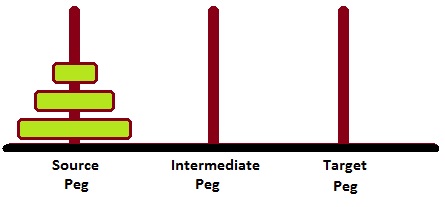
true ?

# **AIM: Implementation of Towers of Honoi using Prolog**

Towers of Hanoi Problem is a famous puzzle to move N disks from the source peg/tower to the target peg/tower using the intermediate peg as an auxiliary holding peg. There are two conditions that are to be followed while solving this problem −

* A larger disk cannot be placed on a smaller disk.
* Only one disk can be moved at a time.

The following diagram depicts the starting setup for N=3 disks.



To solve this, we have to write one procedure move(N, Source, Target, auxiliary). Here N number of disks will have to be shifted from Source peg to Target peg keeping Auxiliary peg as intermediate.

For example – move(3, source, target, auxiliary).

* Move top disk from source to target
* Move top disk from source to auxiliary
* Move top disk from target to auxiliary
* Move top disk from source to target
* Move top disk from auxiliary to source
* Move top disk from auxiliary to target
* Move top disk from source to target

**Rules:**

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

* Only one disk can be moved among the towers at any given time.
* Only the "top" disk can be removed.
* No large disk can sit over a small disk.



### **Program:**

move(1,X,Y,\_) :-

write('Move top disk from '), write(X), write(' to '), write(Y), nl.

move(N,X,Y,Z) :-

N>1,

M is N-1,

move(M,X,Z,Y),

move(1,X,Y,\_),

move(M,Z,Y,X).

**Output:**

?- move(4,source,target,auxiliary).

Move top disk from source to auxiliary

Move top disk from source to target

Move top disk from auxiliary to target

Move top disk from source to auxiliary

Move top disk from target to source

Move top disk from target to auxiliary

Move top disk from source to auxiliary

Move top disk from source to target

Move top disk from auxiliary to target

Move top disk from auxiliary to source

Move top disk from target to source

Move top disk from auxiliary to target

Move top disk from source to auxiliary

Move top disk from source to target

Move top disk from auxiliary to target

true ?