# LAKIREDDY BALI REDDYCOLLEGE OF ENGINEERING

(AUTONOMOUS)



Department of Computer Science& Engineering

(Artificial Intelligence and Machine Learning)

Introduction to Artificial Intelligence and Machine Learning Lab (20AM51)

Name of the Student:

Registered Number:

**Branch&Section:**

AcademicYear: 2022 23

# LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)



**CERTIFICATE**

Certificate that this is a bonafied record of the practical work done in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Laboratory by \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ with Regd. No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_B.Tech Course\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Semester in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Branch during the Academic Year2022-23

No.of Experiments held:\_\_\_\_\_\_\_\_

No.of Experiments Done: \_\_\_\_\_\_\_\_

 2023 Signature of the Faculty

INTERNAL EXAMINER EXTERNAL EXAMINER

**AI&ML LAB**

**---Artificial Intelligence:-**

1. Implementation of DFS for water jug problem using LISP/PROLOG.

**Aim:-**  water jug problem

**Example**: Water Jug Problem

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which y can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?  
Operators – we must define a set of operators that will take us from one state to another:  
1. Fill 4-gal jug (x,y) → (4,y)  
x < 4  
2. Fill 3-gal jug (x,y) → (x,3)  
y < 3  
3. Empty 4-gal jug on ground (x,y) → (0,y)  
x > 0  
4. Empty 3-gal jug on ground (x,y) → (x,0)  
y > 0  
5. Pour water from 3-gal jug (x,y) → (4, y - (4 - x))  
to fill 4-gal jug 0 < x+y ≥ 4 and y > 0  
6. Pour water from 4-gal jug (x,y) → (x - (3-y), 3)  
to fill 3-gal-jug 0 < x+y ≥ 3 and x > 0  
7. Pour all of water from 3-gal jug (x,y) → (x+y, 0)  
into 4-gal jug 0 < x+y ≤ 4 and y ≥ 0  
8. Pour all of water from 4-gal jug (x,y) → (0, x+y)  
into 3-gal jug 0 < x+y ≤ 3 and x ≥ 0  
Through Graph Search, the following solution is found :

**Program:-**

print('Water jug problem')

x=int(input('enter the number for x'))

y=int(input('enter the number for y'))

while True:

rule=int(input("rule number"))

if rule==1:

if x<4:

x=4

if rule==2:

if y<3:

y=3

if rule==3:#empty

if x>0:

x=0

if rule==4:

if y>0:

y=0

if rule==5:

if x+y>=4 and y>0:

x,y=4,y-(4-x)

if rule==6:

if x+y>=3 and x>0:

x,y=x-(3-y),3

if rule==7:

if x+y<=4 and y>=0:

x,y=x+y,0

if rule==8:

if x+y<=3 and x>=0:

x,y=0,x+y

print("x :",x)

print("y :",y)

if x==2:

print('Goal is reached')

break

**output:-**

Water jug problem

enter the number for x0

enter the number for y0

rule number2

x : 0

y : 3

rule number3

x : 0

y : 3

rule number45

x : 0

y : 3

rule number8

x : 0

y : 3

rule number9

x : 0

y : 3

rule number3

x : 0

y : 3

rule number2

x : 0

y : 3

rule number3

x : 0

y : 3

2.Implementation of BFS for tic-tac-toe problem using LISP/PROLOG/Java

**Aim:-** tic-tac-toe problem

There will be two players in a game. Two signs represent each player. The general signs used in the game are **X**and **O**. Finally, there will be a board with **9** boxes.

The gameplay will be as follows.

* First, one user will place their sign in one of the available empty boxes.
* Next, the second user will place their sign in one of the available empty boxes.
* The goal of the players is to place their respective signs completely row-wise or column-wise, or diagonally.
* The game goes on until a player wins the game or it ended up in a draw by filling all boxes without a winning match.

### Algorithm

We will now discuss the algorithm to write the code. This algorithm will help you to write code in any [programming language](https://geekflare.com/new-programming-languages/) of your choice. Let’s see how it’s done.

* Create a board using a 2-dimensional array and initialize each element as empty.
  + You can represent empty using any symbol you like. Here, we are going to use a hyphen. '-'.
* Write a function to check whether the board is filled or not.
  + Iterate over the board and return false if the board contains an empty sign or else return true.
* Write a function to check whether a player has won or not.
  + We have to check all the possibilities that we discussed in the previous section.
  + Check for all the rows, columns, and two diagonals.
* Write a function to show the board as we will show the board multiple times to the users while they are playing.
* Write a function to start the game.
  + Select the first turn of the player randomly.
  + Write an infinite loop that breaks when the game is over (either win or draw).
    - Show the board to the user to select the spot for the next move.
    - Ask the user to enter the row and column number.
    - Update the spot with the respective player sign.
    - Check whether the current player won the game or not.
    - If the current player won the game, then print a winning message and break the infinite loop.
    - Next, check whether the board is filled or not.
    - If the board is filled, then print the draw message and break the infinite loop.
  + Finally, show the user the final view of the board.

**Program:**-

# Define the Tic Tac Toe board

board = [' ' for x in range(9)]

# Define a function to print the Tic Tac Toe board

def print\_board():

print('--------------')

print('| ' + board[0] + ' | ' + board[1] + ' | ' + board[2] + ' |')

print('-------------')

print('| ' + board[3] + ' | ' + board[4] + ' | ' + board[5] + ' |')

print('-------------')

print('| ' + board[6] + ' | ' + board[7] + ' | ' + board[8] + ' |')

print('-------------')

# Define a function to check if there is a winner

def check\_winner():

global game\_running

winner = False

# Check rows

if board[0] == board[1] == board[2] != ' ':

winner = True

elif board[3] == board[4] == board[5] != ' ':

winner = True

elif board[6] == board[7] == board[8] != ' ':

winner = True

# Check columns

elif board[0] == board[3] == board[6] != ' ':

winner = True

elif board[1] == board[4] == board[7] != ' ':

winner = True

elif board[2] == board[5] == board[8] != ' ':

winner = True

# Check diagonals

elif board[0] == board[4] == board[8] != ' ':

winner = True

elif board[2] == board[4] == board[6] != ' ':

winner = True

if winner:

print\_board()

print('Congratulations! ' + turn + ' has won the game!')

game\_running = False

# Define a function for a player's turn

def player\_turn():

global turn

position = input('It is ' + turn + '\'s turn. Choose a position from 1-9:')

valid\_move = False

while not valid\_move:

while position not in ['1', '2', '3', '4', '5', '6', '7','8','9']:

position = input('Invalid input. Choose a position from 1-9:')

position = int(position) - 1

if board[position] == ' ':

valid\_move = True

else:

print('That position is already occupied. Choose another position.')

position = input('Choose a position from 1-9: ')

board[position] = turn

print\_board()

check\_winner()

# Define the main game loop

game\_running = True

turn = 'X'

while game\_running:

print\_board()

player\_turn()

if game\_running:

if turn == 'X':

turn = 'O'

else:

turn = 'X'

**output:-** 

--------------

| | | |

-------------

| | | |

-------------

| | | |

-------------

It is X's turn. Choose a position from 1-9: 1

--------------

| X | | |

-------------

| | | |

-------------

| | | |

-------------

--------------

| X | | |

-------------

| | | |

-------------

| | | |

-------------

It is O's turn. Choose a position from 1-9: 2

--------------

| X | O | |

-------------

| | | |

-------------

| | | |

-------------

--------------

| X | O | |

-------------

| | | |

-------------

| | | |

-------------

It is X's turn. Choose a position from 1-9: 5

--------------

| X | O | |

-------------

| | X | |

-------------

| | | |

-------------

--------------

| X | O | |

-------------

| | X | |

-------------

| | | |

-------------

It is O's turn. Choose a position from 1-9: 6

--------------

| X | O | |

-------------

| | X | O |

-------------

| | | |

-------------

--------------

| X | O | |

-------------

| | X | O |

-------------

| | | |

-------------

It is X's turn. Choose a position from 1-9: 9

--------------

| X | O | |

-------------

| | X | O |

-------------

| | | X |

-------------

--------------

| X | O | |

-------------

| | X | O |

-------------

| | | X |

-------------

Congratulations! X has won the game!

1. Implementation of Monkey Banana Problem using LISP/PROLOG

**Aim:-** Monkey 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.

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.

**Program:**-

on(floor,monkey).

on(floor,monkey).

on(floor,chair).

in(room,monkey).

in(room,chair).

in(room,banana).

at(ceiling,banana).

strong(monkey).

grasp(monkey).

climb(monkey,chair).

push(monkey,chair):-

strong(monkey).

under(banana,chair):-

push(monkey,chair).

canreach(banana,monkey):-

at(floor,banana);

at(ceiling,banana),

under(banana,chair),

climb(monkey,chair).

canget(banana,monkey):-

canreach(banana,monkey),grasp(monkey).

**Output:-**

?- ['E:/monkey.pl'].

true.

?- canget(banana,monkey).

true.

?- trace.

true.

[trace] ?- canget(banana,monkey).

Call: (10) canget(banana, monkey) ? creep

Call: (11) canreach(banana, monkey) ? creep

Call: (12) at(floor, banana) ? creep

Fail: (12) at(floor, banana) ? creep

Redo: (11) canreach(banana, monkey) ? creep

Call: (12) at(ceiling, banana) ? creep

Exit: (12) at(ceiling, banana) ? creep

Call: (12) under(banana, chair) ? creep

Call: (13) push(monkey, chair) ? creep

Call: (14) strong(monkey) ? creep

Exit: (14) strong(monkey) ? creep

Exit: (13) push(monkey, chair) ? creep

Exit: (12) under(banana, chair) ? creep

Call: (12) climb(monkey, chair) ? creep

Exit: (12) climb(monkey, chair) ? creep

Exit: (11) canreach(banana, monkey) ? creep

Call: (11) grasp(monkey) ? creep

Exit: (11) grasp(monkey) ? creep

Exit: (10) canget(banana, monkey) ? creep

Exit: (10) canget(banana, monkey) ? creep

true.

**--Machine Learning:-**

**1.** Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis

based on agiven set of training data samples. Read the training data from a .csv file

**Aim:-**Find-s Algorithm.

Find-salgorithm :- The find-S algorithm finds the most specific hypothesis that fits all thepositive examples. We have to note here that the algorithm considers only those positivetraining 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.

1. Initialize h to the most specific hypothesis in H

2. For each positive training instance x

For each attribute constraint a, in h

If the constraint a, is satisfied by x

Then do nothing

Else replace a, in h by the next more general constraint that is satisfied by x

3. Output hypothesis h

**Program:-**

import pandas as pd

import numpy asnp

#read datasets

Dataset=pd.read\_csv(“ weathercon.csv” )Dataset

m=list()

data=np.asarray(Dataset)

fori inrange(4):

if(data[i][6]=='Yes'):

k=data[i]

m.append(k)

h=[‘ NULL’ ]\*(len(m[0])-1)

fori inrange(6): #initialize firsttrainingsampleinh[0]

if h[i]==m[0][i]:

pass

else:

h[i]=m[0][i]

for j in range(3):

for i in range(6):

if h[i]==m[j][i]:

pass

else:

h[i]= '?'

print(h)

**output:-**

[array(['sunny','Warm','Normal','Strong','Warm','Same','Yes'],

[array(['sunny','Warm','High','Strong','Warm','Same','Yes'],

[array(['sunny','Warm','High','Strong','Cool','Change','Yes'],

['NULL','NULL','NULL','NULL','NULL','NULL']

[‘ sunny’,’ warm’,’ ?’,’ strong’,’ warm’,’ same’ ]

[‘ sunny’,’ warm’,’ ?’,’ strong’,’ warm’,’ same’ ]

[‘ sunny’,’ warm’,’ ?’,’ strong’,’ ?’,’ ?’ ]

**2.** For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypotheses consistent withthe training examples

**Aim:-**Candidate Elimination Algorithm

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

**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

**Program:-**

**Output:-**

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

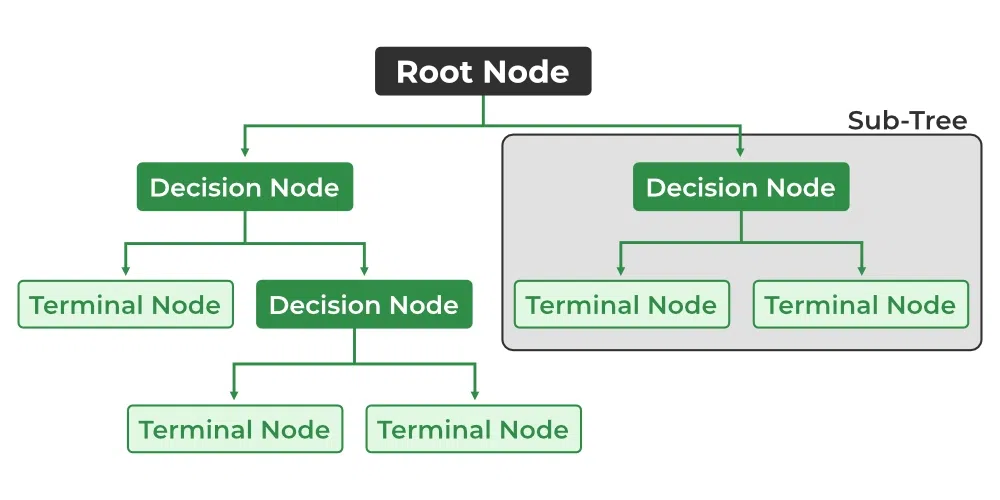
Specific Boundary: [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

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

**3.**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.

**Aim:-** decision tree classifier

* Decision Tree is a **Supervised learning technique**that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where**internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**



**Program:-**

**->** import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

df

print(df.isnull().sum())

**o/p:**

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome 0

dtype: int64

df.corr()

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

y=df[['Outcome']]

print(X.shape)

print(y.shape)

print(X)

print(y)

**o/p:-**

(768, 8)

(768, 1)

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \

0 6 148 72 35 0 33.6

1 1 85 66 29 0 26.6

2 8 183 64 0 0 23.3

3 1 89 66 23 94 28.1

4 0 137 40 35 168 43.1

.. ... ... ... ... ... ...

763 10 101 76 48 180 32.9

764 2 122 70 27 0 36.8

765 5 121 72 23 112 26.2

766 1 126 60 0 0 30.1

767 1 93 70 31 0 30.4

DiabetesPedigreeFunction Age

0 0.627 50

1 0.351 31

2 0.672 32

3 0.167 21

4 2.288 33

.. ... ...

763 0.171 63

764 0.340 27

765 0.245 30

766 0.349 47

767 0.315 23

...

766 1

767 0

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

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

print(X.shape)

print(y.shape)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

**o/p:-**

(768, 8)

(768, 1)

(576, 8)

(192, 8)

(576, 1)

(192, 1)

->from sklearn.preprocessing import StandardScaler

st=StandardScaler()

X\_train=st.fit\_transform(X\_train)

X\_test=st.fit\_transform(X\_test)

from sklearn.tree import DecisionTreeClassifier

classifier=DecisionTreeClassifier(criterion='entropy',random\_state=0)

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

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

print(y\_pred)

**o/p:-**

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

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

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

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

0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0]

from sklearn.metrics import confusion\_matrix

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

cm

**o/p:-**

array([[103, 27],

[ 19, 43]], dtype=int64**)**

->from sklearn.metrics import classification\_report,accuracy\_score

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

print("Classification Report:")

print(res)

**o/p:-**

Classification Report:

precision recall f1-score support

0 0.84 0.79 0.82 130

1 0.61 0.69 0.65 62

accuracy 0.76 192

macro avg 0.73 0.74 0.73 192

weighted avg 0.77 0.76 0.76 192

->result=accuracy\_score(y\_test,y\_pred)

print("Accuracy:",result)

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

Accuracy=accuracy\_score(y\_test,y\_pred)

Accuracy

**o/p:-**

Accuracy: 0.7604166666666666

0.7604166666666666

->from sklearn.metrics import confusion\_matrix

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

print("Confusion Matrix:")

print(result)

**o/p:-**

Confusion Matrix:

[[103 27]

[ 19 43]]

->import seaborn as sns

import matplotlib.pyplot as plt

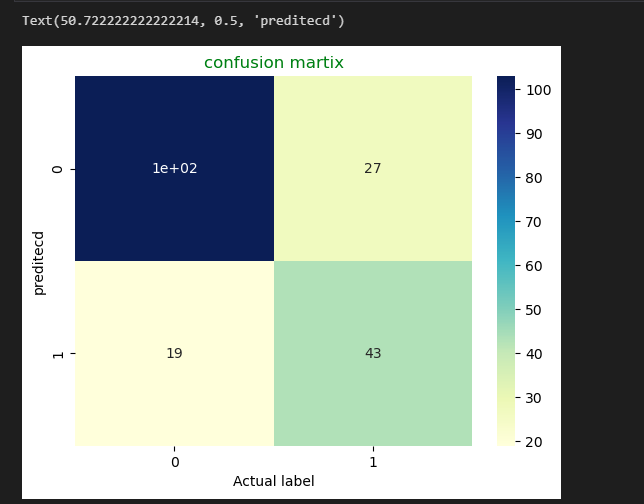
sns.heatmap(cm,annot=True,cmap="YlGnBu")

plt.title("confusion martix",color="green")

plt.xlabel("Actual label")

plt.ylabel("preditecd")

**o/p:-**

****

**4.** Write a program to demonstrate the working of Decision tree regressor. Use appropriate dataset for decision tree regressor

**Aim:-** Decision tree regressor

Decision tree is a supervised learning algorithm used for both classification and regression tasks. In decision tree regression, the algorithm creates a model by recursively splitting the data into smaller subsets based on the input features, until it reaches a stopping criterion, such as a maximum depth or a minimum number of samples per leaf. At each split, the algorithm chooses the feature and the value of that feature that result in the greatest reduction in the variance of the target variable (i.e., the sum of squared differences between the actual and predicted values of the target variable). This process creates a tree-like model where each node represents a split on a feature and each leaf node represents a predicted value.

**Program:-**

import pandas as pd

from sklearn.tree import DecisionTreeRegressor

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

from sklearn.metrics import mean\_absolute\_error # load data

data = pd.read\_csv('attendance\_marks.csv')

X = data.drop('marks' , axis=1)

y = data['marks'] # split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # create decision tree regressor model and fit to training data

model = DecisionTreeRegressor(random\_state=42)

model.fit(X\_train, y\_train) # make predictions on test data and calculate mean absolute error

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

mae = mean\_absolute\_error(y\_test, y\_pred)

print(f'Mean Absolute Error: {mae:.2f}')

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

**Output:-**

Mean Absolute Error: 5.42

Accuracy: 79.36

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

**Aim:**- Random Forest classifier

Random Forest is a popular machine learning algorithm used for both classification and regression tasks. It belongs to the family of ensemble methods, which combine multiple models to improve their accuracy and robustness. Random Forest is a type of decision tree ensemble method that uses multiple decision trees to make a prediction.

The basic idea behind Random Forest is to create a large number of decision trees and combine their predictions to make a final prediction. Each decision tree in the Random Forest is trained on a randomly selected subset of the training data, and at each node of the tree, a random subset of features is considered for splitting. This helps to reduce overfitting and improve the generalization performance of the model

**Program:**-

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

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

from sklearn.metrics import r2\_score

from sklearn.datasets import load\_boston # load boston housing dataset

boston = load\_boston()

X = pd.DataFrame(boston.data, columns=boston.feature\_names)

y = pd.Series(boston.target) # split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # create random forest regressor model and fit to training data

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train) # make predictions on test data and calculate accuracy

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

accuracy = r2\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

**Output**:-

Accuracy: 0.87

**Random Forest Classifier :-**

**Python code:-**

from sklearn.datasets import load\_iris

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score # load iris dataset

iris = load\_iris()

X = iris.data

y = iris.target # split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # create random forest classifier model and fit to training data

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train) # make predictions on test data and compute accuracy and precision

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

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted') # compute confusion matrix

conf\_mat = confusion\_matrix(y\_test, y\_pred) # print results

print(f"Accuracy: {accuracy:.2f}")

print(f"Precision: {precision:.2f}")

print("Confusion Matrix:") print(conf\_mat)

**Output:-**

Accuracy: 1.00

Precision: 1.00

Confusion Matrix: [[10 0 0] [ 0 9 0] [ 0 0 11]]

**Data preprocessing and correlation example:-**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt # read cricket dataset from CSV file

df = pd.read\_csv("cricket.csv") # display the first five rows of the dataset

print(df.head()) # check for missing values

print(df.isna().sum()) # remove unnecessary columns

df.drop(["PLAYER" , "Pos" , "HS" , "Avg" , "100" , "50"], axis=1, inplace=True) # convert columns to numeric df["Inns"] = pd.to\_numeric(df["Inns"], errors="coerce")

df["Runs"] = pd.to\_numeric(df["Runs"], errors="coerce")

df["BF"] = pd.to\_numeric(df["BF"], errors="coerce")

df["SR"] = pd.to\_numeric(df["SR"], errors="coerce") # check for missing values after conversion print(df.isna().sum()) # compute correlation matrix

corr = df.corr() # plot heatmap of correlation matrix

sns.heatmap(corr, annot=True, cmap="YlGnBu") # display the correlation coefficients for each pair of features

print(corr)

**Output:-**

PLAYER Span Mat Inns NO Runs HS Ave BF SR 100 500

0 SR Tendulkar 1989-2013 463 452 41 18426 200\* 44.83 21367 86.23 49 96

20

1 RT Ponting 1995-2012 375 365 39 13704 164 42.03 17046 80.39 30 82

20

2 JH Kallis 1996-2014 328 314 53 11579 139\* 44.36 15885 72.89 17 86

17

3 ST Jayasuriya 1989-2011 445 433 18 13430 189 32.36 14725 91.20 28

68 34

4 DPMD Jayawardene 1998-2015 448 418 39 12650 144\* 33.37 16020 78.96

19 77 28

PLAYER 0

Span 0

Mat 0

Inns 0

NO 0

Runs 0

HS 0

Ave 0

BF 0

SR 0

100 0

50 0

0 0

dtype: int64

PLAYER 0

Span 0

Mat 0

Inns 6

NO 6

Runs 6

HS 6

Ave 6

BF 6

SR 6

100 6

50 6

0 0

dtype: int64

Mat Inns NO Runs BF SR 100

50 0

Mat 1.000000 0.996146 0.905562 0.910104 0.882194 0.468139 0.757196

0.765968 -0.181809

Inns 0.996146 1.000000 0.893116 0.897532 0.876428 0.473458 0.750869

0.758170 -0.180510

NO 0.905562 0.890

**6.** Write a program to demonstrate the working of Logistic Regression classifier. Use appropriate dataset for Logistic Regression.

**Aim:-** Logistic Regression,Linear Regression,MultiLine Linear Regression.

**Linear Regression:-**

Sastatistical method used to study the relationship between two continuous variables,whereon variable(called the dependent variable corresponse variable)is predicted from,the other variable (called the independent variable or predictor variable)through a linear equation. The equation for simple linear regression can be expressed as :

y=β0+β1x+ε

where

y is the dependent variable,

x is the independent variable,

β0 is the intercept tterm,

β1is the slope coefficient,and ε is the error term.

The goal of linear regression is to find the values of the coefficients β0 and β1 that minimize the sum of the square dresiduals between the predicted values of y and the actual values of y

**Program:-**

LinearRegression:-

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

a=pd.read\_csv("Attedance.csv")

df=pd.DataFrame(a)

print(df)

x=df[['attendence']]

y=df[["marks"]]

print(y.head())

print(x.head())

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)

model=LinearRegression() ss

model.fit(x\_train,y\_train) y\_predict=model.predict(x\_test)

print(y\_predict)

r2=r2\_score(y\_test,y\_predict)

print(r2)

print(model.predict([[62]]))

Output:-

attendence marks

0 1 70 80

1 2 71 81

2 3 72 82

3 4 73 83

4 5 74 84

5 6 75 85

6 7 76 86

7 8 77 87

8 9 78 88

9 10 79 89

10 11 80 90

11 12 81 91

12 13 82 92

13 14 83 93

14 15 84 94

15 16 85 95

16 17 86 96

17 18 87 97

18 19 88 98

19 20 89 99

marks

0 80

1 81

2 82

3 83

4 84

attendence

0 70

1 71

2 72

3 73

4 74

Y predicted values

[[86.]

[99.]

[95.]

[88.]

[80.]

[84.]]

R2score:-1.0

Y predicted value:-

[[70.]]

**Multi LinearRegression:**

Is an extension of linear regression that allows for the analysis of more than one independent variable .The equation for multiple linearregression can be expressed as:

y=β0+β1x1 +β2x2+...+β pxp+ε

where y is the dependent variable,

x1,x2,...,x paretheindependent variables,

β0 is the intercept term,

β1, β2, ...,βp are the slope coefficients ,and ε is the error term. The goal of multiple linear regression is to find the values of the coefficients β0,β1,β2,...,β pthat minimize the sum of the squared residuals between the predicted values of y and the actual values of y.

**Code:-**

import numpy as np

import pandas as pd

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

df=pd.read\_csv(r"/content/Multi.csv")

df=pd.DataFrame(df)

print(df)

x=df.iloc[:,:-1]

y=df.iloc[:,-1]

print(f'size of x: {x.shape} and y: {y.shape}')

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

print(f'size of x\_test: {x\_test.shape} and x\_train: {x\_train.shape}')

print(f'size of y\_test: {y\_test.shape} and y\_train: {y\_train.shape}')

from sklearn.linear\_model import LinearRegression

model=LinearRegression()

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

y\_predict=a.predict(x\_test)

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

print(f'r2 score fit {r2\_score(y\_test,y\_predict)}')

print(f'mean score error {mean\_squared\_error (y\_test,y\_predict)}')

a=pd.DataFrame({'Actual':y\_test,'Predict':y\_predict})

print(a)

**OUTput:-**

attandence courses backlogs marks

0 71 3 0 80

1 72 2 0 75

2 73 1 0 70

3 74 0 2 40

4 75 4 0 90

5 76 2 0 76

6 77 0 2 43

7 78 1 1 55

8 79 4 0 91

9 80 2 0 76

10 81 3 0 80

11 82 2 0 76

12 83 1 2 43

13 84 0 3 34

14 85 4 0 95

15 86 3 0 87

16 87 2 0 76

17 88 1 2 51

18 89 3 0 98

19 90 0 0 85

sizeof x:(20,3)andy:(20,)

sizeof x\_test:(4,3)andx\_train:(16,3)

sizeof y\_test:(4,)andy\_train:(16,)

r2 score fit 0.9342747169332467

mean score error 28.24133256774555

Actual Predict

10 80 86.266664

13 34 28.757983

11 76 82.157044

15 87 89.88206

**PolynomialRegrssion:-**

Polynomial regression is a type of regression analysis in which the relationship between the independent variable x and the dependent variable y is modele dasannth degree polynomial.The aim of polynomial regression is to find the best-fit curve that represents the relationship between the variables. Polynomial regression can be useful in situations where the relationship between the variables is not linear , but rather has a curve dornonlinear shape.It can be used in various fields such as finance, economics ,physics ,and engineering.

**Code:-**

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error,accuracy\_score

from sklearn.preprocessing import PolynomialFeatures

# Load the dataset

data =pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality

/winequality-white.csv')

# Split the dataset into training and testing sets

X = data.drop(['quality'], axis=1)

y = data['quality']

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

# Create polynomial features

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

# Fit the polynomial regression model to the training data

model = LinearRegression()

model.fit(X\_train\_poly, y\_train)

# Make predictions on the testing set

y\_pred = model.predict(X\_test\_poly)

# Calculate the accuracy of the model

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

**Output:-**

Accuracy: 85

**Logistic Regression:-**

Logistic Regression is a type of classification algorithm used to predict a binary outcome (i.e., a value of 0 or 1) based on one or more input variables. The algorithm models the probability of the binary outcome using a logistic function, which is a mathematical function that maps any input value to a probability between 0 and 1. The logistic function is an S-shaped curve that starts at 0 when the input is very negative, rises steeply in the middle, and levels off at 1 when the input is very positive. The logistic regression algorithm works by fitting a line to the input variables that maximizes the likelihood of the observed outcomes. The line is called the decision boundary, and it separates the two classes (i.e., 0 and 1) in the input space. The algorithm then uses the decision boundary to predict the class of new input data. P(Y=1|X) = 1 / (1 + exp(-z)) where z = b0 + b1X1 + b2X2 + ... + bn\*Xn Here, b0 is the intercept term, and b1, b2, ..., bn are the coefficients associated with the input variables X1, X2, ..., Xn. The logistic function, 1 / (1 + exp(-z)), maps the linear combination of input variables to a probability between 0 and 1

**Code:-**

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.linear\_model import LogisticRegression

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

confusion\_matrix

# Load the dataset

data = load\_iris()

X = data['data']

y = data['target']

# Split the dataset into training and testing sets

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

# Create a logistic regression model

model = LogisticRegression()

# Fit the model to the training data

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

# Make predictions on the testing set

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

# Calculate the accuracy of the model

accuracy = accuracy\_score(y\_test, y\_pred)

# Calculate the precision of the model

precision = precision\_score(y\_test, y\_pred)

# Calculate the sensitivity (recall) of the model

sensitivity = recall\_score(y\_test, y\_pred)

# Calculate the confusion matrix of the model

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

tn, fp, fn, tp = cm.ravel()

# Print the performance metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Sensitivity:", sensitivity)

print("TP:"], tp)

print("FP:", fp)

print("TN:", tn)

print("FN:", fn)

**Output:-**

Accuracy: 0.9777777777777777

Precision: 0.9809523809523809

Sensitivity: 0.9777777777777777

TP: 14 FP: 0 TN: 15 FN: 11