# 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-FindS :**

The Find-S algorithm is a machine learning algorithm used for supervised learning of classification tasks. It is a concept learning algorithm that learns a hypothesis that can be used to classify new examples based on their features.

The algorithm starts with the most specific hypothesis, which assumes that all attributes of the instance are negative. It then iteratively updates the hypothesis by generalizing it based on positive examples until it covers all positive examples.

# The Find-S algorithm works as follows:

Initialize the hypothesis h to the most specific hypothesis. This hypothesis states that all attributes of an instance are negative.

For each positive training example, update the hypothesis by making the most specific generalization that still includes the positive example.

Return the hypothesis.

The Find-S algorithm can be applied to problems with discrete-valued attributes. It produces a hypothesis that is consistent with the training data, but it may not be the most accurate hypothesis.

# DataSet :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Wind | Play Tennis |
| Overcast | Hot | High | Weak | Yes |
| Rain | Mild | High | Weak | Yes |
| Rain | Cool | Normal | Strong | No |
| Overcast | Cool | Normal | Weak | Yes |

**Pythoncode:**

import csv num\_attributes = 4 a = []

print("\n The Given Training Data Set \n") with open('finds.csv', 'r') as csvfile:

reader = csv.reader(csvfile) for row in reader:

a.append (row) print(row)

print("\n The initial value of hypothesis: ") hypothesis = ['0'] \* num\_attributes print(hypothesis)

for j in range(0,num\_attributes): hypothesis[j] = a[0][j];

print("\n Find S: Finding a Maximally Specific Hypothesis\n") for i in range(0,len(a)):

if a[i][num\_attributes]=='Yes':

for j in range(0,num\_attributes): if a[i][j]!=hypothesis[j]:

hypothesis[j]='?'

else :

hypothesis[j]= a[i][j]

print(" For Training instance No:{0} the hypothesis is".format(i),hypothesis) print("\n The Maximally Specific Hypothesis for a given TrainingExamples :\n") print(hypothesis)

# Output:

The Given Training Data Set

['Overcast', 'Hot', 'High', 'Weak', 'Yes']

['Rain', 'Mild', 'High', 'Weak', 'Yes']

['Rain', 'Cool', 'Normal', 'Strong', 'No']

['Overcast', 'Cool', 'Normal', 'Weak', 'Yes']

['Overcast', 'Hot', 'High', 'Weak', 'Yes']

The initial value of hypothesis:

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

Find S: Finding a Maximally Specific Hypothesis

For Training instance No:0 the hypothesis is

['Overcast', 'Hot', 'High', 'Weak']

For Training instance No:1 the hypothesis is

['?', '?', 'High', 'Weak']

For Training instance No:2 the hypothesis is

['?', '?', 'High', 'Weak']

For Training instance No:3 the hypothesis is

['?', '?', '?', 'Weak']

For Training instance No:4 the hypothesis is

['?', '?', '?', 'Weak']

The Maximally Specific Hypothesis for a given TrainingExamples :

['?', '?', '?', 'Weak']

# 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 (CEA) is a machine learning algorithm used for concept learning in supervised learning tasks. The algorithm is used to find the most specific hypothesis based on a given set of training data samples.

The algorithm starts by initializing the most specific hypothesis S0, which is the set of all possible hypotheses in the hypothesis space. It then initializes the most general hypothesis G0, which is the set of all possible hypotheses in the hypothesis space.

The algorithm then iterates through each training example and updates the most specific and most general hypotheses based on the example. If an example is positive, the algorithm updates the most specific hypothesis by removing any inconsistent hypotheses. If an example is negative, the algorithm updates the most general hypothesis by removing any inconsistent hypotheses.

The algorithm continues to iterate through the training examples until it converges on a single hypothesis or a set of hypotheses. The final hypothesis is the most specific hypothesis that is consistent with all the positive examples and none of the negative examples.

The steps of the Candidate-Elimination Algorithm are as follows:

Initialize the most specific hypothesis S0, which includes all the attributes and values of the training examples.

Initialize the most general hypothesis G0, which includes all possible attributes and values in the hypothesis space.

For each positive training example, update the most specific hypothesis by removing any inconsistent hypotheses.

For each negative training example, update the most general hypothesis by removing any inconsistent hypotheses.

Return the most specific hypothesis that is consistent with all the positive examples and none of the negative examples.

The Candidate-Elimination Algorithm is useful for finding the most specific hypothesis in a hypothesis space, but it may not always converge to a single hypothesis if there are multiple hypotheses that are consistent with the training data.

# DataSet

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Outlook | Temperature |  | Humididty | Wind |  | Play |
| Sunny | Warm |  | Normal | Strong |  | Yes |
| Sunny | Warm |  | High | Strong |  | Yes |
| Sunny | Cold |  | High | Strong |  | No |

**Python code:**

import numpyas np

import pandasaspd

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

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

print("\nInstancesare:\n",concepts)

target =np.array(data.iloc[:,-1]) print("\nTarget Valuesare: ",target)

deflearn(concepts, target): specific\_h=concepts[0].copy()

print("\nInitializationof specific\_hand 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)

fori, hinenumerate(concepts): print("\nInstance", i+1 , "is", h) if target[i] =="yes": print("Instance is Positive") forx in range(len(specific\_h)):

ifh[x]!=specific\_h[x]: specific\_h[x] ='?' general\_h[x][x] ='?'

if target[i] =="no":

print("Instance is Negative ") forx in range(len(specific\_h)): ifh[x]!=specific\_h[x]:

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

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

print("Specific Bundaryafter", i+1,"Instanceis", specific\_h) print("Generic Boundaryafter", i+1,"Instance is", general\_h) print("\n")

indices=[i for i, val inenumerate(general\_h) ifval ==['?','?','?', '?', '?', '?']] fori inindices:

general\_h.remove(['?', '?', '?', '?', '?', '?']) returnspecific\_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', 'high', 'strong', 'warm', 'same'],

['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'high', 'strong', 'cool', 'change']

]

Target values are: ['yes', 'yes', 'no', 'yes']

Initialization of specific\_h and general\_h:

Specific Boundary: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

Generic Boundary: [ ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same'] and is positive. Specific Boundary after 1 instance is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same'] Generic Boundary after 1 instance is [ ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same'] and is positive. Specific Boundary after 2 instances is ['sunny', 'warm', '?', 'strong', 'warm', 'same'] Generic Boundary after 2 instances is [ ['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change'] and is negative. Specific Boundary after 3 instances is ['sunny', 'warm', '?', 'strong', 'warm', 'same'] Generic Boundary after 3 instances is [ ['sunny', '?', '?', '?', '?', '?'],

['?', 'warm', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

]

Instance 4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change']. Instance is Positive.

Specific Boundary after 4 Instance is ['sunny', 'warm', '?', 'strong', '?', '?']

Generic Boundary after 4 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h: ['sunny', 'warm', '?', 'strong', '?', '?']

Final General\_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

# AIM:Implement Linear and multi Linear Regression Desccription:Model/equation/R2,MAE

**Linear regression** is a statistical method used to study the relationship between two continuous

variables, where one variable (called the dependent variable or response 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 term, β1 is 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 squared residuals between the predicted values of y and the actual values of y.

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

y = β0 + β1x1 + β2x2 + ... + βpxp + ε

Where y is the dependent variable, x1, x2, ..., xp are the independent 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, ..., βp that minimize the sum of the squared residuals between the predicted values of y and the actual values of y.

**R-squared (R2) score** measures the proportion of the variance in the dependent variable (y) that is explained by the independent variable(s) (x) in the model. R2 score ranges from 0 to 1, where a value of 1 indicates that the model explains all the variability in the dependent variable, and a value of 0 indicates that the model does not explain any variability in the dependent variable.

The formula for R2 score is:

R2 = 1 - (SS\_res / SS\_tot)

Where SS\_res is the sum of squared residuals (the difference between the predicted values and the actual values of y) and SS\_tot is the total sum of squares (the difference between the actual values of y and the mean value of y).

**Mean absolute error (MAE)** is a measure of the average magnitude of the errors between the predicted values and the actual values of y. MAE is calculated as the average of the absolute differences between the predicted values and the actual values of y. The formula for MAE is:

MAE = (1/n) \* Σ|yi - ŷi|

Where n is the number of observations, yi is the actual value of y, and ŷi is the predicted value of y.

Dataset:Attendance No of certificattions Marks

# Python:

LinearRegression:- import pandas as pd

from sklearn.linear\_model import LinearRegression from sklearn.metrics import r2\_score a=pd.read\_csv("C:\\Users\\ML Lab\\Desktop\\inputfile.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()

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 70 80

1 71 81

2 72 82

3 73 83

4 74 84

5 75 85

6 76 86

7 77 87

8 78 88

9 79 89

10 80 90

11 81 91

12 82 92

13 83 93

14 84 94

15 85 95

16 86 96

17 87 97

18 88 98

19 89 99

Y predicted values:

Predicted values

|  |  |
| --- | --- |
| 0 | 86.0 |
| 1 | 99.0 |
| 2 | 95.0 |
| 3 | 88.0 |
| 4 | 80.0 |
| 5 | 84.0 |

R2score:-1.0

Y predicted value:-

[[70.]]

# Multi LinearRegression:-

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)

attendance 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 |

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 | 89.882069 |

# Aim:Polynomial Regrssion Description :

Polynomial Regression is a type of regression analysis where the relationship between the independent variable and dependent variable is modeled as an nth degree polynomial. This is used when the relationship between the variables is nonlinear, and a linear regression cannot capture the relationship.

In polynomial regression, the independent variable is raised to different powers, such as x^2, x^3, x^4, etc., and the coefficients for these powers are calculated using least squares regression. The degree of the polynomial is usually selected based on the data, with higher degrees allowing for more complex relationships to be modeled, but also increasing the risk of overfitting.

Once the model is fit to the data, it can be used to predict the dependent variable for new values of the independent variable. Polynomial regression has a wide range of applications, such as predicting stock prices, analyzing trends in data, and modeling physical systems. It is a useful tool in data analysis and machine learning.

# Python 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 Description:

Logistic Regression is a statistical method used for binary classification, which means it is used to predict the probability of a binary outcome (0 or 1). It is a type of generalized linear model that is used when the dependent variable is categorical.

The goal of logistic regression is to find the best-fit parameters of a function that describes the relationship between the independent variables and the probability of a specific outcome. The function used in logistic regression is called the sigmoid function, which maps any real-valued number to a probability value between 0 and 1.

The sigmoid function takes the form:

P(y=1|X) = 1 / (1 + exp(-z))

where P(y=1|X) is the probability of the binary outcome (y=1) given the independent variables (X), z is the linear combination of the independent variables and their associated coefficients, and exp() is the exponential function.

To find the best-fit parameters, logistic regression uses a technique called maximum likelihood estimation, which involves finding the parameter values that maximize the likelihood of the observed data given the model.

Logistic regression can be used for a variety of tasks, such as predicting whether a customer will buy a product or not, whether a patient has a disease or not, or whether a user will click on an ad or not. It is widely used in industries such as finance, healthcare, marketing, and more.

# Python 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: 1

# Description: Accuracy, Precision, Sensitivity, Recall, TP, FP, TN, FN

In the context of classification problems, several metrics are used to evaluate the performance of a model. Here are the most common ones:

Accuracy: It measures the proportion of correctly classified samples out of the total number of samples. It can be calculated as follows:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

Precision: It measures the proportion of true positives among the samples that the model predicted as positive. It can be calculated as follows:

Precision = TP / (TP + FP)

Sensitivity (also called recall or true positive rate): It measures the proportion of true positives among the samples that are actually positive. It can be calculated as follows:

Sensitivity = TP / (TP + FN)

Specificity (also called true negative rate): It measures the proportion of true negatives among the samples that are actually negative. It can be calculated as follows:

Specificity = TN / (TN + FP)

False Positive (FP): It is the number of samples that are actually negative but the model predicted them as positive.

False Negative (FN): It is the number of samples that are actually positive but the model predicted them as negative.

True Positive (TP): It is the number of samples that are actually positive and the model correctly predicted them as positive.

True Negative (TN): It is the number of samples that are actually negative and the model correctly predicted them as negative.

These metrics are used to evaluate the performance of a model on a test set. A good model should have high accuracy, precision, and sensitivity, and low false positive and false negative rates. However, there may be a trade-off between these metrics, and the best metric to use depends on the specific problem and the business goals.

# Decison Tree-Regressor Description:

A Decision Tree Regressor is a type of decision tree algorithm used for regression problems, where the goal is to predict a continuous target variable.

The Decision Tree Regressor works by recursively splitting the data into subsets based on the feature that provides the most information gain for the target variable. At each internal node, the algorithm selects the feature that provides the most information gain, which is calculated using a specific criterion such as mean squared error or mean absolute error. Once the data is split into subsets, the algorithm fits a regression model to each subset based on the values of the target variable.

In practice, Decision Tree Regressors are often used for their interpretability, as the resulting tree can be visualized and easily understood by non-experts. However, they can be prone to overfitting, especially when the tree is deep and complex. Therefore, techniques such as pruning or ensemble methods like Random Forests can be used to improve the performance of Decision Tree Regressors.

# Python code:

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

# Method:Parameters- 7.Descision Tree Classifier Python code:-

import pandas as pd

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

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix from sklearn.datasets import load\_iris

# load iris dataset iris = load\_iris()

X = pd.DataFrame(iris.data, columns=iris.feature\_names) y = pd.Series(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 decision tree classifier model and fit to training data model = DecisionTreeClassifier(random\_state=42) model.fit(X\_train, y\_train)

# make predictions on test data and calculate accuracy, precision, and confusion matrix

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

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

precision = precision\_score(y\_test, y\_pred, average='weighted') cm = confusion\_matrix(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}') print(f'Precision: {precision:.2f}') print('Confusion Matrix:') print(cm)

# Output:-

Accuracy: 1.00

Precision: 1.00 Confusion Matrix: [[10 0 0]

[ 0 9 0]

[ 0 0 11]]

# RandomForest-Regressor Description:

Random Forest is an ensemble machine learning algorithm that combines multiple decision trees to generate more accurate and stable predictions. Each decision tree in a Random Forest is built using a random subset of the training data and a random subset of the input features, which helps to reduce overfitting.

To make a prediction using a Random Forest, each decision tree in the ensemble is evaluated, and the output of all the trees is combined to produce the final prediction. The most common approach is to use majority voting for classification problems and averaging for regression problems.

Random Forests have several advantages over individual decision trees, including improved accuracy, reduced overfitting, and increased robustness to outliers and noisy data. They are also relatively easy to use and require minimal feature engineering.

However, Random Forests can be computationally expensive and memory-intensive, especially when dealing with large datasets or high-dimensional feature spaces. They can also be difficult to interpret compared to simpler models like linear regression.

# Python code:

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 Description:

Data preprocessing is a crucial step in any machine learning project. It involves transforming raw data into a format that can be easily understood by machine learning models. The goal of data preprocessing is to improve the quality of data, eliminate inconsistencies, and transform the data into a format that is suitable for analysis.

Some common data preprocessing techniques include data cleaning, feature selection, feature engineering, normalization, and standardization. Data cleaning involves removing or fixing missing or incorrect data, while feature selection involves selecting the most relevant features to be used in the analysis. Feature engineering involves creating new features that can be used in the analysis.

Correlation refers to the degree of association between two variables. It is a statistical measure that indicates the extent to which two or more variables are related. Correlation can be either positive or negative. Positive correlation means that two variables move in the same direction, while negative correlation means that two variables move in opposite directions.

Correlation analysis is a technique used to study the relationship between two or more variables. It helps to identify which variables are most strongly related to each other. Correlation analysis is often used in data preprocessing to identify which features are most relevant to the analysis. By identifying the most relevant features, data preprocessing can help to improve the accuracy of machine learning models.

# Python code:

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 50 0

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 0.893

# List of Experiments (Artificial Intelligence)

# Implementation of Monkey Banana Problem using LISP/PROLOG

# The Monkey and Banana problem is a classic problem in Artificial Intelligence. It involves a monkey in a room trying to reach a banana hanging from the ceiling, with the additional challenge of the monkey needing to move a box to reach the banana.

# Initial State: The monkey is in a specific position, the banana is hanging from the ceiling, and the box is in a specific position.

# Actions: The monkey can perform the following actions:

# "walk": Moves the monkey from one position to an adjacent position.

# "climb": Allows the monkey to climb the box.

# "push": Pushes the box from one position to an adjacent position.

# Goal State: The goal is to reach the banana while standing on the box.

# Path Cost: The cost associated with each action is typically uniform, so each action has the same cost.

**Using prolog:-**

on(floor,monkey).

on(floor,box).

in(room,monkey).

in(room,box).

at(ceiling,banana).

strong(monkey).

grasp(monkey).

climb(monkey,box).

push(monkey,box):-

strong(monkey).

under(banana,box):-

push(monkey,box).

canreach(banana,monkey):-

at(floor,banana);

at(ceiling,banana),

under(banana,box),

climb(monkey,box).

canget(banana,monkey):-

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

write("monkey can reach the banana").

OUTPUT:-

?- trace

| .

true.

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

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

Call: (11) at(floor, banana) ? creep

Fail: (11) at(floor, banana) ? creep

Redo: (10) canreach(banana, monkey) ? creep

Call: (11) at(ceiling, banana) ? creep

Exit: (11) at(ceiling, banana) ? creep

Call: (11) under(banana, box) ? creep

Call: (12) push(monkey, box) ? creep

Call: (13) strong(monkey) ? creep

Exit: (13) strong(monkey) ? creep

Exit: (12) push(monkey, box) ? creep

Exit: (11) under(banana, box) ? creep

Call: (11) climb(monkey, box) ? creep

Exit: (11) climb(monkey, box) ? creep

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

true.

2**.Implementation of DFS for water jug problem using LISP/PROLOG/python**

**Water Jug Problem**

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallonone, a pump which has unlimited water which you 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?

**State Representation and Initial State** –

we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note 0 ≤ x ≤ 4, and 0 ≤ y ≤ 3.

**Our initial state: (0,0)**

**Goal Predicate – state** = (2,y) where 0 ≤ y ≤ 3.

**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

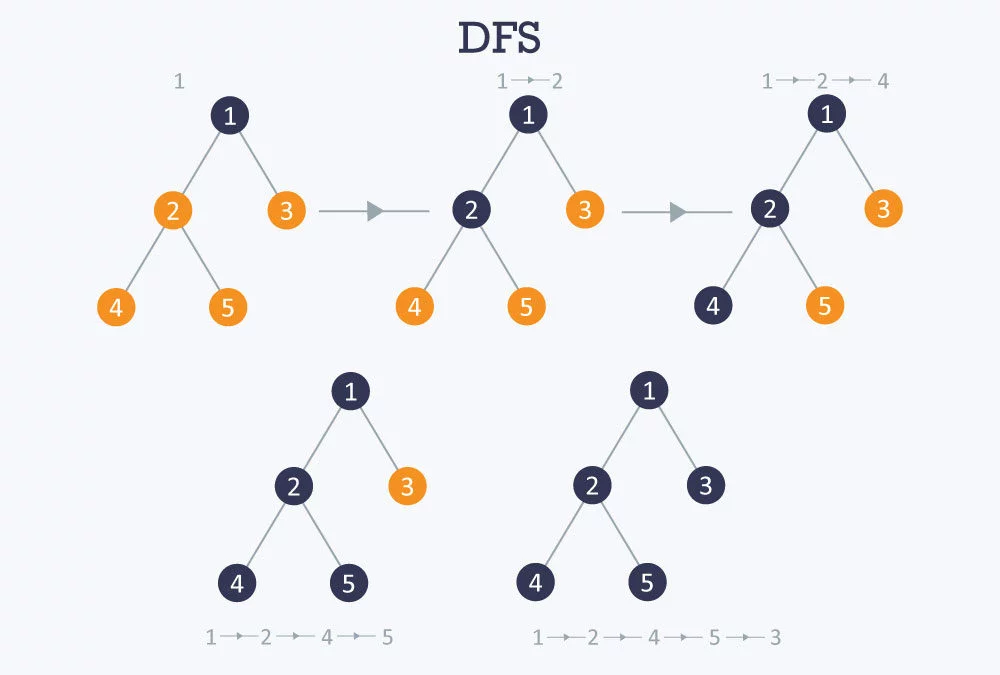
7. Pour all of water from 3-gal jug (x,y) → (x+y, 0)

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 :



Python code:-

print("water jug problem")

x=int(input("enter the number"))

y=int(input("enter the number"))

while True:

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

if rule==1:

if x<4:

x=4

if rule==2:

if y<3:

y=3

if rule==3:

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 state reached")

break;

output:-

water jug problem

enter the number0

enter the number0

enter the number1

x :- 4

y :- 0

enter the number6

x :- 1

y :- 3

enter the number4

x :- 1

y :- 0

enter the number8

x :- 0

y :- 1

enter the number1

x :- 4

y :- 1

enter the number6

x :- 2

y :- 3

Goal state reached

3.**Implementation of BFS for tic-tac-toe problem using LISP/PROLOG/python**.

from tkinter import \*

import random

def next\_turn(row, column):

global player

if buttons[row][column]['text'] == "" and check\_winner() is False:

if player == players[0]:

buttons[row][column]['text'] = player

if check\_winner() is False:

player = players[1]

label.config(text=(players[1]+" turn"))

elif check\_winner() is True:

label.config(text=(players[0]+" wins"))

elif check\_winner() == "Tie":

label.config(text="Tie!")

else:

buttons[row][column]['text'] = player

if check\_winner() is False:

player = players[0]

label.config(text=(players[0]+" turn"))

elif check\_winner() is True:

label.config(text=(players[1]+" wins"))

elif check\_winner() == "Tie":

label.config(text="Tie!")

def check\_winner():

for row in range(3):

if buttons[row][0]['text'] == buttons[row][1]['text'] == buttons[row][2]['text'] != "":

buttons[row][0].config(bg="green")

buttons[row][1].config(bg="green")

buttons[row][2].config(bg="green")

return True

for column in range(3):

if buttons[0][column]['text'] == buttons[1][column]['text'] == buttons[2][column]['text'] != "":

buttons[0][column].config(bg="green")

buttons[1][column].config(bg="green")

buttons[2][column].config(bg="green")

return True

if buttons[0][0]['text'] == buttons[1][1]['text'] == buttons[2][2]['text'] != "":

buttons[0][0].config(bg="green")

buttons[1][1].config(bg="green")

buttons[2][2].config(bg="green")

return True

elif buttons[0][2]['text'] == buttons[1][1]['text'] == buttons[2][0]['text'] != "":

buttons[0][2].config(bg="green")

buttons[1][1].config(bg="green")

buttons[2][0].config(bg="green")

return True

elif empty\_spaces() is False:

for row in range(3):

for column in range(3):

buttons[row][column].config(bg="yellow")

return "Tie"

else:

return False

def empty\_spaces():

spaces = 9

for row in range(3):

for column in range(3):

if buttons[row][column]['text'] != "":

spaces -= 1

if spaces == 0:

return False

else:

return True

def new\_game():

global player

player = random.choice(players)

label.config(text=player+" turn")

for row in range(3):

for column in range(3):

buttons[row][column].config(text="",bg="#F0F0F0")

window = Tk()

window.title("Tic-Tac-Toe")

players = ["x","o"]

player = random.choice(players)

buttons = [[0,0,0],

[0,0,0],

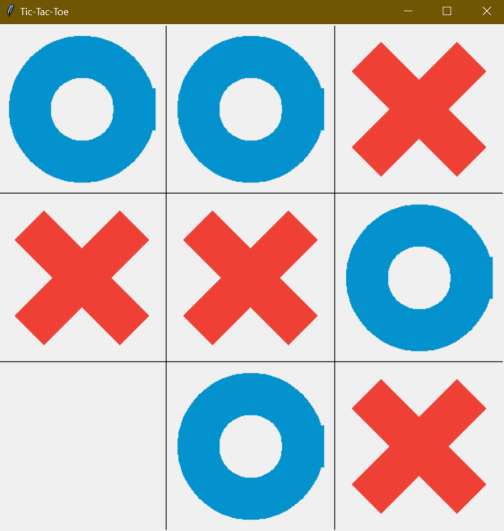
[0,0,0]]

label = Label(text=player + " turn", font=('consolas',40))

label.pack(side="top")

reset\_button = Button(text="restart", font=('consolas',20), command=new\_game)

OUTPUT:-



**4.Prolog example:-**

got(devi,first).

went(devi,kulumanali).

went(rahul,kulumanali).

happy(rahul):-

got(rahul,first);

went(rahul,kulumanali).

Output:-

Got(devi,X).

X=first

Happy(Rahul).

True

**5**.**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.

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:-

| ?- [towersofhanoi].

compiling D:/TP Prolog/Sample\_Codes/towersofhanoi.pl for byte code...

D:/TP Prolog/Sample\_Codes/towersofhanoi.pl compiled, 8 lines read - 1409 bytes written, 15 ms

yes

| ?- 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 ?

(31 ms) yes