\_\_Machine Learning Algorithms\_\_

**1.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 of Find-S Algorithm:

The Find-S algorithm is a machine learning algorithm used for finding a hypothesis that best fits a set of positive training examples. It is a basic algorithm for concept learning in the field of artificial intelligence and is used in supervised learning. The algorithm is used to find a maximally specific hypothesis from a set of training data examples. The algorithm starts with an empty hypothesis and then iteratively adds attributes to it based on the positive examples it encounters during training. The algorithm examines each example and compares it to the current hypothesis. If the example is consistent with the hypothesis, then the algorithm moves on to the next example. If the example is not consistent with the hypothesis, then the algorithm adds the attributes of the example to the hypothesis. The algorithm stops when all of the positive examples have been examined and a maximally specific hypothesis has been constructed. The hypothesis is then used to classify new examples.

Algorithm:

1. Initilize *h* to the most specific hypothesis in *H*

2. For each positive training instance *x*

For each attribute contraint *ai* in *h*

If the contraint *ai* is satisfied by *x*

then do nothing

Else

replace *ai* in *h* by the next more general constraint that is satisfied by *x*

3. Output the hypothesis *h*

Dataset:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forecast** | **EnjoySport** |
| Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| Sunny | Warm | High | Strong | Warm | Same | Yes |
| Rainy | Cold | High | Strong | Warm | Change | No |
| Sunny | Warm | High | Strong | Cool | Change | Yes |

Pythoncode:

import numpy as np

# Define the training dataset

train\_data = np.array([['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', True],

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

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

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', True]])

# Define the hypothesis space

hypothesis = ['0', '0', '0', '0', '0', '0']

# Implement the Find-S algorithm

for data in train\_data:

if data[-1] == True:

for i in range(len(data) - 1):

if hypothesis[i] == '0':

hypothesis[i] = data[i]

elif hypothesis[i] != data[i]:

hypothesis[i] = '?'

# Print the final hypothesis

print(“Final Hypothesis is: ”hypothesis)

Output:

Final Hypothesis is: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

**2. 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 is a machine learning algorithm that operates on instances represented as a conjunction of attribute-value pairs. It is used to learn a hypothesis that can correctly classify all positive and negative instances in a given training set.The algorithm then iterates over the training instances and eliminates hypotheses that do not correctly classify the instances. For each instance, the algorithm updates the most specific and most general hypotheses as follows:

- If the instance is positive, the algorithm removes any hypothesis that does not include the instance's attribute-value pairs and updates the most specific hypothesis to include any attribute-value pairs that are present in the instance but not in the current most specific hypothesis.

- If the instance is negative, the algorithm removes any hypothesis that includes the instance's attribute-value pairs and updates the most general hypothesis to include any attribute-value pairs that are not present in the instance but are present in the current most general hypothesis.

The Candidate Elimination algorithm is a simple and efficient algorithm that can handle noisy data and incremental learning. However, it has some limitations, such as not being able to handle attribute dependencies or interactions between attributes.

Algorithm:

For each training example d, do:

If d is positive example

Remove from G any hypothesis h inconsistent with d

For each hypothesis s in S not consistent with d:

Remove s from S

Add to S all minimal generalizations of s consistent with d and having a generalization in G

Remove from S any hypothesis with a more specific h in S

If d is negative example

Remove from S any hypothesis h inconsistent with d

For each hypothesis g in G not consistent with d:

Remove g from G

Add to G all minimal specializations of g consistent with d and having a specialization in S

Remove from G any hypothesis having a more general hypothesis in G

Dataset:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forecast** | **EnjoySport** |
| Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| Sunny | Warm | High | Strong | Warm | Same | Yes |
| Rainy | Cold | High | Strong | Warm | Change | No |
| Sunny | Warm | High | Strong | Cool | Change | Yes |

Pythoncode:

import numpy as np

import pandas as pd data = pd.DataFrame(data=pd.read\_csv(' C:\Users\pc\Downloads\ENJOYSPORT.csv'))

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

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

def learn(concepts, target):

specific\_h = concepts[0].copy()

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

for i, h in enumerate(concepts):

if target[i] == "yes":

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

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

specific\_h[x] ='?'

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

if target[i] == "no":

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] = '?'

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:

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

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

**3. Aim: Implement Linear and multi Linear Regression on any dataset.**

Description:

Linear Regression:-

Linear regression is a type of supervised learning algorithm in machine learning that is used to predict a continuous outcome variable (also known as the dependent variable) based on one or more input predictor variables (also known as independent variables). It assumes a linear relationship between the input variables and the output variable, and attempts to find the best linear relationship that fits the data. The goal of linear regression is to create a model that can accurately predict the value of the dependent variable for new input data.

Equation: y= a0+a1x+ ε

Here,

Y= Dependent Variable (Target Variable)  
X= Independent Variable (predictor Variable)  
a0= intercept of the line (Gives an additional degree of freedom)  
a1 = Linear regression coefficient (scale factor to each input value).  
ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Dataset:

|  |  |
| --- | --- |
| **Attendance** | **Marks** |
| 89 | 90 |
| 88 | 90 |
| 90 | 93 |
| 94 | 95 |
| 92 | 94 |
| 78 | 81 |
| 87 | 88 |
| 85 | 87 |
| 64 | 67 |
| 66 | 67 |
| 74 | 76 |
| 84 | 87 |
| 94 | 95 |
| 78 | 80 |
| 73 | 76 |
| 80 | 81 |
| 82 | 84 |
| 79 | 82 |
| 86 | 87 |
| 89 | 91 |

PythonCode:

import pandas as pd

att=pd.read\_csv("E:\\21761A4205\\attendence.csv")

print(att)

from sklearn.linear\_model import LinearRegression

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

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

X=att[["Attendance"]]

Y=att[['Marks']]

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

print("\nX\_train:\n")

print(X\_train)

print("\nX\_train shape:\n")

print(X\_train.shape)

print("\nX\_test shape:\n")

print(X\_test.shape)

print("\nY\_train")

print(Y\_train)

print("\nY\_train shape:\n")

print(Y\_train.shape)

print("\nY\_test shape")

print(Y\_test.shape)

model=LinearRegression()

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

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

print(y\_prediction)

R2score=r2\_score(Y\_test,y\_prediction)

print(R2score)

m=model.score(X\_test,Y\_test)

print(m)

Output:

Attendance Marks

0 89 91

1 88 85

2 90 92

3 94 95

4 92 93

5 78 80

6 87 90

7 85 88

8 64 75

9 66 69

10 74 70

11 84 80

12 94 96

13 78 75

14 73 70

15 80 79

16 82 80

17 79 80

18 86 88

19 89 90

X\_train:

Attendance

18 86

0 89

7 85

16 82

2 90

12 94

4 92

3 94

9 66

8 64

17 79

15 80

5 78

11 84

X\_train shape:

(14, 1)

X\_test shape:

(6, 1)

Y\_train

Marks

18 88

0 91

7 88

16 80

2 92

12 96

4 93

3 95

9 69

8 75

17 80

15 79

5 80

11 80

Y\_train shape:

(14, 1)

Y\_test shape

(6, 1)

[[88.78856307]

[77.21525348]

[89.61522804]

[76.38858851]

[87.9618981 ]

[80.52191336]]

0.6843986841164922

0.6843986841164922

Multi-Linear Regression:

Multilinear regression, also known as multiple regression, is an extension of linear regression in machine learning where two or more independent variables are used to predict the outcome variable. It involves identifying the linear relationship between the input variables and the output variable, and fitting a linear equation that best represents this relationship. The goal of multilinear regression is to create a model that can accurately predict the value of the dependent variable for new input data based on multiple input variables. It is commonly used in business and social science research to analyze and predict the relationship between multiple variables.

Equation: MLR formula look like : y = a + bx1 + cx2 + dx3 + …….

The coefficients tell you exactly how much each independent variable contributes to the dependent variable and how much each independent variable contributes in isolation.

Dataset:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attendance** | **Marks** | **No\_of\_certifications** | **No\_of\_bls** |
| 89 | 90 | 1 | 0 |
| 88 | 90 | 1 | 1 |
| 90 | 93 | 1 | 0 |
| 94 | 95 | 1 | 0 |
| 92 | 94 | 1 | 0 |
| 78 | 81 | 1 | 1 |
| 87 | 88 | 1 | 0 |
| 85 | 87 | 1 | 0 |
| 64 | 67 | 1 | 3 |
| 66 | 67 | 1 | 2 |
| 74 | 76 | 1 | 1 |
| 84 | 87 | 1 | 0 |
| 94 | 95 | 1 | 0 |
| 78 | 80 | 1 | 1 |
| 73 | 76 | 1 | 2 |
| 80 | 81 | 1 | 1 |
| 82 | 84 | 1 | 0 |
| 79 | 82 | 1 | 1 |
| 86 | 87 | 1 | 0 |
| 89 | 91 | 1 | 0 |

PythonCode:

import pandas as pd

att=pd.read\_csv("E:\\21761A4205\\attendence.csv")

print(att)

from sklearn.linear\_model import LinearRegression

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

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

X=att[["Attendance","No\_of\_certifications","No\_of\_bls"]]

Y=att['Marks']

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

print("\nX\_train:\n")

print(X\_train)

print("\nX\_train shape:\n")

print(X\_train.shape)

print("\nX\_test shape:\n")

print(X\_test.shape)

print("\nY\_train")

print(Y\_train)

print("\nY\_train shape:\n")

print(Y\_train.shape)

print("\nY\_test shape")

print(Y\_test.shape)

model=LinearRegression()

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

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

print(y\_prediction)

R2score=r2\_score(Y\_test,y\_prediction)

print(R2score)

print(model.coef\_)

print(model.intercept\_)

m=model.score(X\_test,Y\_test)

print(m)

y\_prediction

mae=mean\_absolute\_error(Y\_test,y\_prediction)

print(mae)

m=model.predict([[79,1,1]])

print(m)

df=pd.DataFrame({'actual': Y\_test, 'predicted': y\_prediction})

print(df)

import matplotlib.pyplot as plt

p=plt.plot(Y\_test,y\_prediction)

print(p)

Output:

Attendance No\_of\_certifications No\_of\_bls Marks

0 89 1 0 90

1 88 1 1 90

2 90 1 0 93

3 94 1 0 95

4 92 1 0 94

5 78 1 1 81

6 87 1 0 88

7 85 1 0 87

8 64 1 3 67

9 66 1 2 67

10 74 1 1 76

11 84 1 0 87

12 94 1 0 95

13 78 1 1 80

14 73 1 2 76

15 80 1 1 81

16 82 1 0 84

17 79 1 1 82

18 86 1 0 87

19 89 1 0 91

X\_train:

Attendance No\_of\_certifications No\_of\_bls

11 84 1 0

2 90 1 0

3 94 1 0

18 86 1 0

10 74 1 1

9 66 1 2

16 82 1 0

0 89 1 0

6 87 1 0

8 64 1 3

7 85 1 0

13 78 1 1

5 78 1 1

19 89 1 0

X\_train shape:

(14, 3)

X\_test shape:

(6, 3)

Y\_train

11 87

2 93

3 95

18 87

10 76

9 67

16 84

0 90

6 88

8 67

7 87

13 80

5 81

19 91

Name: Marks, dtype: int64

Y\_train shape:

(14,)

Y\_test shape

(6,)

[95.72931686 75.21266735 81.01579128 93.7492276 89.92619293 82.00583591]

0.9893788626217219

[0.99004463 0. 0.13714384]

2.6651218674905834

0.9893788626217219

array([95.72931686, 75.21266735, 81.01579128, 93.7492276 , 89.92619293,

82.00583591])

0.6385456001830837

[81.01579128]

actual predicted

12 95 95.729317

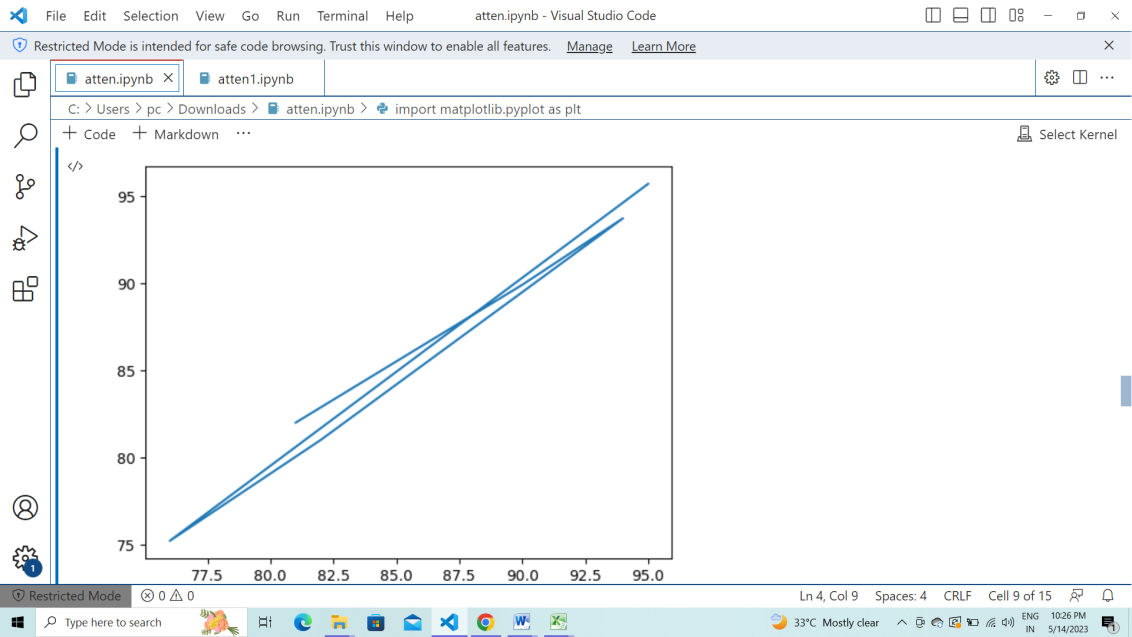
14 76 75.212667

17 82 81.015791

4 94 93.749228

1 90 89.926193

15 81 82.005836



**4.Aim:Implement Polynomial Regression on any dataset.**

Description:

Polynomial regression is a type of regression analysis in machine learning where the relationship between the independent variable and the dependent variable is modeled as an nth degree polynomial. Unlike linear regression, which assumes a linear relationship between the input variables and the output variable, polynomial regression can capture more complex non-linear relationships. It involves fitting a polynomial function to the data by minimizing the sum of squared errors between the predicted values and the actual values. The degree of the polynomial is determined by the degree of the highest order term in the equation. Polynomial regression is useful in situations where a linear relationship between the input and output variables is not appropriate, and there may be curvature or other non-linear patterns in the data.

Equation of the Polynomial Regression Model

Any linear equation is a polynomial regression that has a degree of 1. The very common and usual equation used to define the regression is;

y = mx+b

In this equation, m is the slope, and b is the y-intercept. One can easily write this as

f(x) = c0 + c1 x where c1 is the slope and the c0 is the y-intercept.

Dataset:

|  |  |
| --- | --- |
| **Attendance** | **Marks** |
| 89 | 90 |
| 88 | 90 |
| 90 | 93 |
| 94 | 95 |
| 92 | 94 |
| 78 | 81 |
| 87 | 88 |
| 85 | 87 |
| 64 | 67 |
| 66 | 67 |
| 74 | 76 |
| 84 | 87 |
| 94 | 95 |
| 78 | 80 |
| 73 | 76 |
| 80 | 81 |
| 82 | 84 |
| 79 | 82 |
| 86 | 87 |
| 89 | 91 |

Pythoncode:

from sklearn.linear\_model import LinearRegression

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

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

x=att[["Attendance"]]

y=att['Marks']

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

print("\nX\_train:\n")

print(x\_train)

print("\nX\_train shape:\n")

print(x\_train.shape)

print("\nX\_test shape:\n")

print(x\_test.shape)

print("\nY\_train")

print(y\_train)

print("\nY\_train shape:\n")

print(y\_train.shape)

print("\nY\_test shape")

print(y\_test.shape)

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)

print(poly)

poly\_features1 = poly.fit\_transform(x\_train)

poly\_reg\_model = LinearRegression()

poly\_reg\_model.fit(poly\_features1,y\_train)

poly\_features2 = poly.fit\_transform(x\_test)

r=poly\_reg\_model.predict(poly\_features2)

print(r)

Output:

X\_train:

Attendance

2 90

13 78

19 89

8 64

18 86

4 92

16 82

6 87

14 73

3 94

15 80

1 88

17 79

7 85

X\_train shape:

(14, 1)

X\_test shape:

(6, 1)

Y\_train

2 93

13 80

19 91

8 67

18 87

4 94

16 84

6 88

14 76

3 95

15 81

1 90

17 82

7 87

Name: Marks, dtype: int64

Y\_train shape:

(14,)

Y\_test shape

(6,)

PolynomialFeatures()

[80.16480593 95.65769281 90.74372157 85.8956086 76.39695742 68.98770827]

**5.Aim: Implement Logistic Regression on any dataset and visualize the confusion matrix for the dataset.**

Description:

Logistic regression is a type of regression analysis in machine learning that is used to predict a binary outcome variable (i.e., a variable that can take on one of two possible values, such as 0 or 1) based on one or more input predictor variables (also known as independent variables). It models the relationship between the input variables and the probability of the binary outcome using a logistic function. The logistic function transforms the output of a linear equation into a probability value between 0 and 1. The goal of logistic regression is to create a model that can accurately predict the probability of the binary outcome for new input data. Logistic regression is widely used in fields such as healthcare, finance, and marketing to predict outcomes such as disease diagnosis, credit risk, and customer churn.

Dataset:

https://github.com/plotly/datasets/blob/master/diabetes.csv

Pythoncode:

import pandas as pd

data = pd.read\_csv("E:\\21761A4205\\diabetes.csv")

print(data)

#data.drop(['label'],inplace =True,axis=1)

print(data.shape)

print(data['Outcome'])

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

y = data['Outcome']

print(X)

print(y)

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)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

data.isnull().values.any()

from sklearn.linear\_model import LogisticRegression

lg = LogisticRegression(random\_state=16)

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

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

print(y\_pred)

from sklearn.metrics import confusion\_matrix

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

print(cnf\_matrix)

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

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

print(acc)

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

print(prec)

rec = recall\_score(y\_test,y\_pred,average='weighted')

print(rec)

f1sc = f1\_score(y\_test,y\_pred)

print(f1sc)

import seaborn as sns

ax = sns.heatmap(cnf\_matrix,annot=True,cmap="Blues")

ax.set\_title('Seaborn Confusion Matrix with labels\n\n');

ax.set\_xlabel('\nPredicted Values')

ax.set\_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order

ax.xaxis.set\_ticklabels(['False','True'])

ax.yaxis.set\_ticklabels(['False','True'])

print(data['Outcome'].value\_counts())

X\_test

Output:

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 Outcome

0 0.627 50 1

1 0.351 31 0

2 0.672 32 1

3 0.167 21 0

4 2.288 33 1

.. ... ... ...

763 0.171 63 0

764 0.340 27 0

765 0.245 30 0

766 0.349 47 1

767 0.315 23 0

[768 rows x 9 columns]

(768, 9)

0 1

1 0

2 1

3 0

4 1

..

763 0

764 0

765 0

766 1

767 0

Name: Outcome, Length: 768, dtype: int64

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

...

765 0

766 1

767 0

Name: Outcome, Length: 768, dtype: int64

(614, 8)

(154, 8)

(614,)

(154,)

False

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

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

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

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

0 0 1 0 1 0]

[[95 7]

[20 32]]

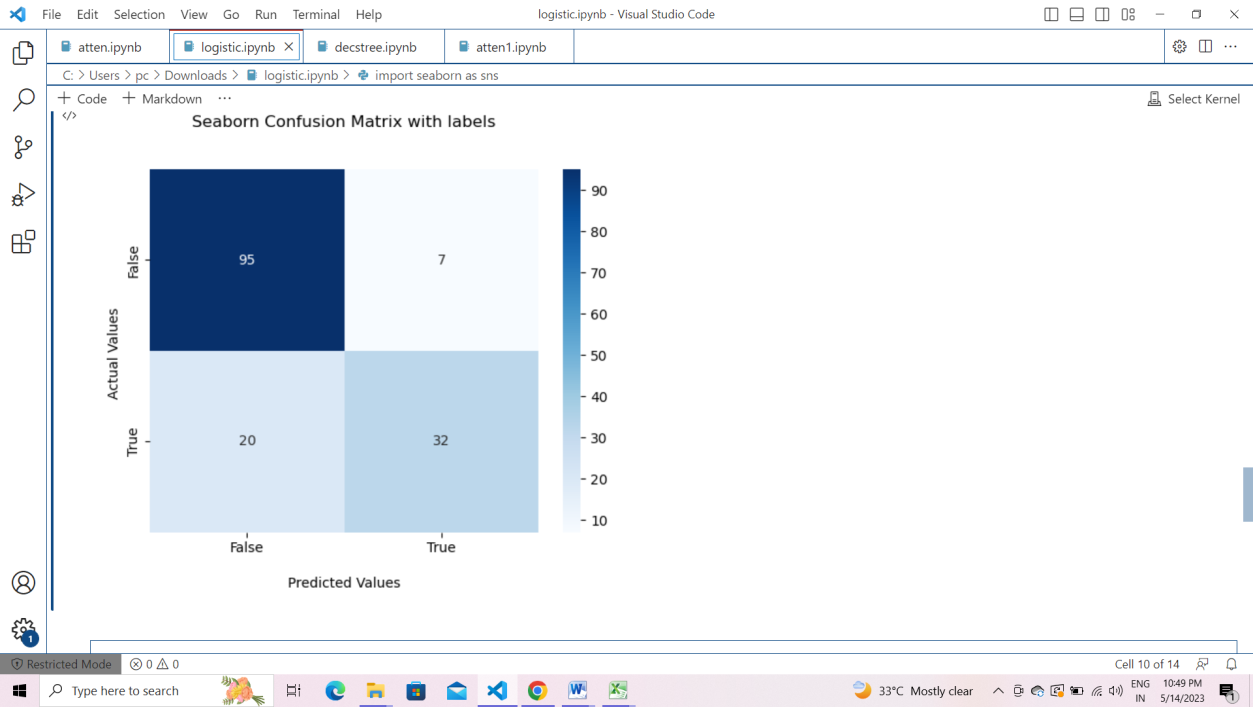
0.8246753246753247# accuracy\_score

0.8205128205128205#precision\_score

0.8246753246753247# recall\_score

0.7032967032967034

[Text(0, 0.5, 'False'), Text(0, 1.5, 'True')]



0 500

1 268

Name: Outcome, dtype: int64

\_\_\_\_\_\_

r=lg.predict([[5,130,76,44,280,30.1,0.401,24]])

#r=lg.predict([[7,187,68,39,304,37.7,0.254,41]])

if r==0:

    print("no diabetes")

else:

    print("has diabetes")

Output: no diabetes

**6.Aim: Take any dataset and Implement Descision Tree Classifier on taken data.**

Description:

A decision tree classifier is a type of supervised learning algorithm in machine learning that is used for classification tasks. It is a tree-like structure where each internal node represents a test on a feature or attribute, each branch represents the outcome of the test, and each leaf node represents a class label. The decision tree classifier works by recursively splitting the input data based on the values of different features, until it reaches a point where it can predict the class label of the input data with high accuracy.

The decision tree is built using a top-down approach, where at each step, the algorithm selects the feature that results in the highest information gain or the highest reduction in impurity, based on a pre-defined criterion such as entropy or Gini index. Decision tree classifiers are commonly used in applications such as medical diagnosis, credit risk assessment, and fraud detection.

Dataset:

https://github.com/plotly/datasets/blob/master/diabetes.csv

Python code:

#import pandas library

import pandas as pd

*#loading dataset*

df=pd.read\_csv("/content/drive/My Drive/Colab Notebooks/diabetes\_dataset.csv")

df.head()*#feature variables*

x=df.drop(['Outcome'], axis=1)

x

768 rows × 8 columns

*#target variable*

y=df.Outcome

y

*# Import Decision Tree Classifier*

from sklearn.tree import DecisionTreeClassifier from sklearn.model\_selection import train\_test\_split *# Import train\_test\_split function*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=1)

*# Create Decision Tree classifer object*

model = DecisionTreeClassifier()

*# Train Decision Tree Classifer*

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

*#Predict the response for test dataset*

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

*#Evaluation using Accuracy score*

from sklearn import metrics *#Import scikit-learn metrics module for accuracy calculation*

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred)\*100)

*#Evaluation using Confusion matrix*

from sklearn.metrics import confusion\_matrix

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

*#Evaluation using Classification report*

from sklearn.metrics import classification\_report

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

*#checking prediction value*

model.predict([[6,148,72,35,0,33.6,0.627,50]])

*#Import modules for Visualizing Decision trees*

from sklearn.tree import export\_graphviz

from sklearn.externals.six import StringIO

from IPython.display import Image

import pydotplus

features=x.columns

dot\_data = StringIO()

export\_graphviz(model,out\_file=dot\_data,filled=True, rounded=True,special\_characters=True,feature\_names = features,class\_names=['0','1'])

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png('diabetes\_set.png')

Image(graph.create\_png())

Output:

| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | o/p |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |  |
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 |  |
| 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 |  |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 |  |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 |  |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 |  |
| ... | ... | ... | ... | ... | ... | ... | ... |  |
| 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 |  |
| 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 |  |
| 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 |  |
| 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 |  |
| 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 |  |

768 rows × 8 columns

0 1

1 0

2 1

3 0

4 1

..

763 0

764 0

765 0

766 1

767 0

Name: Outcome, Length: 768, dtype: int64

Accuracy: 67.53246753246754

array([[76, 23],

[27, 28]])

precision recall f1-score support

0 0.74 0.77 0.75 99

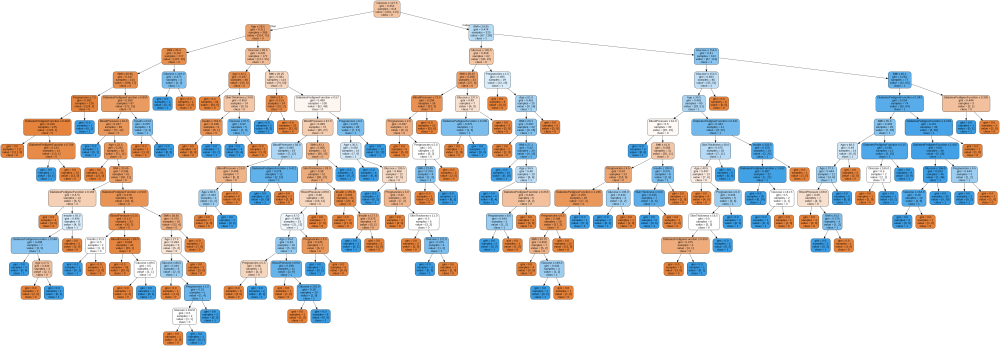
1 0.55 0.51 0.53 55

accuracy 0.68 154

macro avg 0.64 0.64 0.64 154

weighted avg 0.67 0.68 0.67 154

array([1])



**7.Aim: Implement Decision-Tree Regressor on any example dataset and plot a tree with predicted values.**

Description:

A decision tree regressor is a machine learning algorithm used for regression tasks. It is a tree-like model where each internal node represents a feature or attribute, each branch represents a possible value or range of values for that feature, and each leaf node represents a predicted continuous value. The decision tree regressor works by recursively partitioning the input data based on the values of different features, and predicting the output value at each leaf node.

During training, the decision tree regressor uses a top-down approach to build the tree. At each step, it selects the feature and corresponding threshold that result in the best split of the data, aiming to minimize the variance or the mean squared error of the predictions. This splitting process continues until a stopping criterion is met, such as reaching a maximum depth or a minimum number of samples at a leaf node.

To make predictions with the trained decision tree regressor, new input data is passed down the tree, and the predicted value is obtained by averaging the target values of the training samples that reach the corresponding leaf node.

Decision tree regressors have several advantages. They can handle both numerical and categorical features, handle missing values, and are interpretable, as the decision paths can be easily understood. However, decision tree regressors can suffer from overfitting, especially when the trees become too deep or complex. To mitigate this, techniques like pruning, setting a maximum depth, or using ensemble methods such as random forests can be applied.

Overall, decision tree regressors are versatile and widely used algorithms for solving regression problems, providing a balance between interpretability and predictive performance.

Dataset:

|  |  |
| --- | --- |
| **Attendance** | **Marks** |
| 89 | 90 |
| 88 | 90 |
| 90 | 93 |
| 94 | 95 |
| 92 | 94 |
| 78 | 81 |
| 87 | 88 |
| 85 | 87 |
| 64 | 67 |
| 66 | 67 |
| 74 | 76 |
| 84 | 87 |
| 94 | 95 |
| 78 | 80 |
| 73 | 76 |
| 80 | 81 |
| 82 | 84 |
| 79 | 82 |
| 86 | 87 |
| 89 | 91 |

Pythoncode:

import pandas as pd

data=pd.read\_csv("E:\\21761A4205\\attendence.csv")

data

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)

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:

Attendance Marks

0 89 91

1 88 85

2 90 92

3 94 95

4 92 93

5 78 80

6 87 90

7 85 88

8 64 75

9 66 69

10 74 70

11 84 80

12 94 96

13 78 75

14 73 70

15 80 79

16 82 80

17 79 80

18 86 88

19 89 90

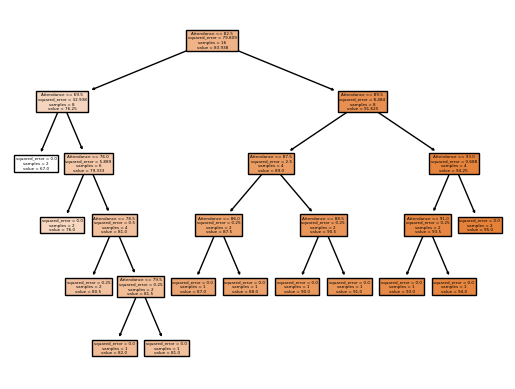
(16, 1)

(4, 1)

(16,)

(4,)

array([91., 81., 87., 87.])



**8. Aim: Perform Data Pre-Processing on a bottle.csv dataset and plot graph between salnity and temperature.**

Description:

Data preprocessing is an essential step in machine learning that involves preparing and transforming the raw data into a format suitable for training and evaluating machine learning models. It is an important part of the overall machine learning pipeline and can significantly impact the performance and accuracy of the models.

The goal of data pre-processing is to address common issues in the data, such as missing values, outliers, inconsistent formats, and irrelevant features, and to transform the data into a format that can be effectively used by the machine learning algorithms.

Here are some common techniques used in data pre-processing:

1. Data Cleaning: This involves handling missing values, which can be done by either removing the instances with missing values, filling in the missing values with a reasonable estimate (e.g., mean, median, or mode), or using more advanced techniques like interpolation or imputation.

2. Data Transformation: This includes normalizing or standardizing the data to bring all features to a similar scale, which helps prevent certain features from dominating the learning process. Common techniques include min-max scaling or z-score normalization.

3. Handling Outliers: Outliers can have a significant impact on the model's performance. They can be detected and dealt with by either removing them, transforming them to a more reasonable value, or using robust statistical methods that are less affected by outliers.

4. Feature Selection: It involves selecting the most relevant features that have the most significant impact on the target variable. This can be done through various techniques like statistical tests, correlation analysis, or using algorithms that evaluate the importance of features.

5. Feature Encoding: Categorical variables are often encoded into numerical representations to be used by machine learning algorithms. Common techniques include one-hot encoding, label encoding, or ordinal encoding.

6. Dimensionality Reduction: If the dataset has a large number of features, dimensionality reduction techniques like principal component analysis (PCA) or feature extraction methods can be used to reduce the number of features while retaining the most relevant information.

7. Splitting Data: The dataset is typically divided into training, validation, and testing sets. The training set is used to train the model, the validation set is used for hyperparameter tuning and model selection, and the testing set is used for evaluating the final model's performance.

These are just some of the common techniques used in data preprocessing. The specific preprocessing steps and techniques depend on the nature of the data, the problem at hand, and the requirements of the machine learning algorithm being used. Proper data preprocessing can lead to more accurate and robust models, improving

the overall performance of the machine learning system.

Dataset:

C:\Users\ML Lab\Downloads\bottle.csv

Python code:

import pandas as pd

b=pd.read\_csv("C:\\Users\\ML Lab\\Downloads\\bottle.csv\\bottle.csv")

print(b)

Output:

Cst\_Cnt Btl\_Cnt Sta\_ID Depth\_ID \

0 1 1 054.0 056.0 19-4903CR-HY-060-0930-05400560-0000A-3

1 1 2 054.0 056.0 19-4903CR-HY-060-0930-05400560-0008A-3

2 1 3 054.0 056.0 19-4903CR-HY-060-0930-05400560-0010A-7

3 1 4 054.0 056.0 19-4903CR-HY-060-0930-05400560-0019A-3

4 1 5 054.0 056.0 19-4903CR-HY-060-0930-05400560-0020A-7

Depthm T\_degC Salnty O2ml\_L STheta O2Sat ... R\_PHAEO R\_PRES \

0 0 10.50 33.440 NaN 25.649 NaN ... NaN 0

1 8 10.46 33.440 NaN 25.656 NaN ... NaN 8

2 10 10.46 33.437 NaN 25.654 NaN ... NaN 10

3 19 10.45 33.420 NaN 25.643 NaN ... NaN 19

4 20 10.45 33.421 NaN 25.643 NaN ... NaN 20

R\_SAMP DIC1 DIC2 TA1 TA2 pH2 pH1 DIC Quality Comment

0 NaN NaN NaN NaN NaN NaN NaN NaN

1 NaN NaN NaN NaN NaN NaN NaN NaN

2 NaN NaN NaN NaN NaN NaN NaN NaN

3 NaN NaN NaN NaN NaN NaN NaN NaN

4 NaN NaN NaN NaN NaN NaN NaN NaN

[5 rows x 74 columns]

\_\_\_\_\_\_\_

b=b.dropna(axis=1,thresh=800000)

print(b)

Output:

Cst\_Cnt Btl\_Cnt Sta\_ID Depth\_ID \

0 1 1 054.0 056.0 19-4903CR-HY-060-0930-05400560-0000A-3

1 1 2 054.0 056.0 19-4903CR-HY-060-0930-05400560-0008A-3

2 1 3 054.0 056.0 19-4903CR-HY-060-0930-05400560-0010A-7

3 1 4 054.0 056.0 19-4903CR-HY-060-0930-05400560-0019A-3

4 1 5 054.0 056.0 19-4903CR-HY-060-0930-05400560-0020A-7

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

864858 34404 864859 093.4 026.4 20-1611SR-MX-310-2239-09340264-0000A-7

864859 34404 864860 093.4 026.4 20-1611SR-MX-310-2239-09340264-0002A-3

864860 34404 864861 093.4 026.4 20-1611SR-MX-310-2239-09340264-0005A-3

864861 34404 864862 093.4 026.4 20-1611SR-MX-310-2239-09340264-0010A-3

864862 34404 864863 093.4 026.4 20-1611SR-MX-310-2239-09340264-0015A-3

Depthm T\_degC Salnty STheta RecInd T\_prec ... DarkAq \

0 0 10.500 33.4400 25.64900 3 1.0 ... 9.0

1 8 10.460 33.4400 25.65600 3 2.0 ... 9.0

2 10 10.460 33.4370 25.65400 7 2.0 ... 9.0

3 19 10.450 33.4200 25.64300 3 2.0 ... 9.0

4 20 10.450 33.4210 25.64300 7 2.0 ... 9.0

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

864858 0 18.744 33.4083 23.87055 7 2.0 ... 9.0

864859 2 18.744 33.4083 23.87072 3 2.0 ... 9.0

864860 5 18.692 33.4150 23.88911 3 2.0 ... 9.0

864861 10 18.161 33.4062 24.01426 3 2.0 ... 9.0

864862 15 17.533 33.3880 24.15297 3 2.0 ... 9.0

MeanAq R\_Depth R\_TEMP R\_POTEMP R\_SALINITY R\_SIGMA R\_SVA \

0 9.0 0 10.50 10.50 33.440 25.640 233.0

1 9.0 8 10.46 10.46 33.440 25.650 232.5

2 9.0 10 10.46 10.46 33.437 25.650 232.8

3 9.0 19 10.45 10.45 33.420 25.640 234.1

4 9.0 20 10.45 10.45 33.421 25.640 234.0

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

864858 9.0 0 18.74 18.74 33.408 23.871 402.4

864859 9.0 2 18.74 18.74 33.408 23.871 402.5

864860 9.0 5 18.69 18.69 33.415 23.889 400.8

864861 9.0 10 18.16 18.16 33.406 24.014 389.1

864862 9.0 15 17.53 17.53 33.388 24.153 376.0

R\_DYNHT R\_PRES

0 0.000 0

1 0.010 8

2 0.020 10

3 0.040 19

4 0.040 20

... ... ...

864858 0.000 0

864859 0.008 2

864860 0.020 5

864861 0.040 10

864862 0.059 15

[864863 rows x 24 columns]

\_\_\_\_\_\_

b.info()

Output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 864863 entries, 0 to 864862

Data columns (total 24 columns):

# Column Non-Null Count Dtype

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

0 Cst\_Cnt 864863 non-null int64

1 Btl\_Cnt 864863 non-null int64

2 Sta\_ID 864863 non-null object

3 Depth\_ID 864863 non-null object

4 Depthm 864863 non-null int64

5 T\_degC 853900 non-null float64

6 Salnty 817509 non-null float64

7 STheta 812174 non-null float64

8 RecInd 864863 non-null int64

9 T\_prec 853900 non-null float64

10 S\_prec 817509 non-null float64

11 NH3q 808299 non-null float64

12 C14A1q 848605 non-null float64

13 C14A2q 848623 non-null float64

14 DarkAq 840440 non-null float64

15 MeanAq 840439 non-null float64

16 R\_Depth 864863 non-null int64

17 R\_TEMP 853900 non-null float64

18 R\_POTEMP 818816 non-null float64

19 R\_SALINITY 817509 non-null float64

20 R\_SIGMA 812007 non-null float64

21 R\_SVA 812092 non-null float64

22 R\_DYNHT 818206 non-null float64

23 R\_PRES 864863 non-null int64

dtypes: float64(16), int64(6), object(2)

memory usage: 158.4+ MB

\_\_\_\_\_\_\_

import numpy as np

mean\_value=b["T\_degC"].mean()

b["T\_degC"].fillna(value=mean\_value, inplace=True)

e=b["T\_degC"].isnull().sum()

print(e)

Output:

0

\_\_\_\_\_\_\_

import numpy as np

mean\_value=b["Salnty"].mean()

b["Salnty"].fillna(value=mean\_value, inplace=True)

a=b["Salnty"].isnull().sum()

print(a)

Output:

0

\_\_\_\_\_\_\_

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

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

x=b[["Salnty"]]

y=b['T\_degC']

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

print(x\_test)

model=LinearRegression()

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

y\_prediction=model.predict(x\_test)

print(y\_prediction)

import matplotlib.pyplot as plt

p=plt.plot(y\_test,y\_prediction)

print(p)

Output:

Salnty

683571 34.13300

415965 33.84035

615594 33.98800

188585 33.89000

685504 33.42600

... ...

619379 33.99100

771395 34.18900

737511 33.16000

521783 34.35700

473635 33.84035

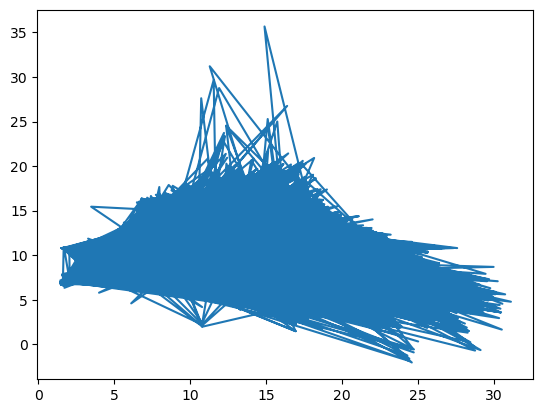
[259459 rows x 1 columns]

[ 9.45279419 10.79847488 10.11954251 ... 13.92690535 8.42278299

10.79847488]

[<matplotlib.lines.Line2D object at 0x000001E38E652490>]

[14.79137904]



\_\_\_\_\_\_\_

m=model.predict([[34.13300]])

print(m)

Output:

[9.45279419]

\_\_\_\_\_\_

co=b.corr()

print(co[co>0])

Output:

Cst\_Cnt Btl\_Cnt Depthm T\_degC Salnty O2ml\_L STheta \

Cst\_Cnt 1.000000 0.999345 NaN 0.089329 NaN 0.156096 NaN

Btl\_Cnt 0.999345 1.000000 NaN 0.090118 NaN 0.154219 NaN

Depthm NaN NaN 1.000000 NaN 0.572630 NaN 0.582710

T\_degC 0.089329 0.090118 NaN 1.000000 NaN 0.795700 NaN

Salnty NaN NaN 0.572630 NaN 1.000000 NaN 0.603470

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

DIC2 NaN NaN 0.572374 NaN 0.955942 NaN 0.965546

TA1 NaN NaN 0.786873 NaN 0.895124 NaN 0.779899

TA2 NaN NaN 0.866185 NaN 0.937147 NaN 0.887778

pH2 NaN NaN 0.223304 0.547436 0.583807 0.174204 NaN

pH1 0.020778 0.020315 NaN 0.743735 0.106417 0.663591 NaN

O2Sat Oxy\_µmol/Kg BtlNum ... R\_CHLA R\_PHAEO R\_PRES \

Cst\_Cnt 0.136510 0.147493 NaN ... 0.005807 NaN NaN

Btl\_Cnt 0.134732 0.145498 NaN ... 0.009648 NaN NaN

Depthm NaN NaN NaN ... NaN NaN 0.999994

T\_degC 0.850596 0.795543 0.750413 ... 0.108925 0.014987 NaN

Salnty NaN NaN NaN ... NaN NaN 0.571405

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

DIC2 NaN NaN NaN ... 0.024819 0.220101 0.570633

TA1 NaN NaN NaN ... 0.006186 NaN 0.785334

TA2 NaN NaN NaN ... 0.265152 0.235116 0.865140

pH2 0.732628 0.180058 NaN ... 0.047099 0.049127 0.223304

pH1 0.913807 0.666715 0.333808 ... NaN NaN NaN

R\_SAMP DIC1 DIC2 TA1 TA2 pH2 pH1

Cst\_Cnt NaN NaN NaN NaN NaN NaN 0.020778

Btl\_Cnt NaN NaN NaN NaN NaN NaN 0.020315

Depthm NaN 0.567855 0.572374 0.786873 0.866185 0.223304 NaN

T\_degC 0.006503 NaN NaN NaN NaN 0.547436 0.743735

Salnty NaN 0.938153 0.955942 0.895124 0.937147 0.583807 0.106417

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

DIC2 NaN 0.991091 1.000000 0.885735 0.888669 NaN NaN

TA1 0.025613 0.853682 0.885735 1.000000 0.994924 0.616844 0.260047

TA2 0.020177 0.883278 0.888669 0.994924 1.000000 0.578090 0.577613

pH2 NaN NaN NaN 0.616844 0.578090 1.000000 0.995913

pH1 0.333808 NaN NaN 0.260047 0.577613 0.995913 1.000000

[70 rows x 70 columns]

**9. Aim: Random-Forest Classifier**

Description:

Random Forest Classifier is a popular machine learning algorithm that belongs to the ensemble learning family. It combines multiple decision trees to create a powerful classification model. Here are some key points to understand about Random Forest Classifier:

1. Ensemble Method: Random Forest is an ensemble method that combines multiple decision trees to make predictions. It builds a forest of decision trees and aggregates their predictions to make the final prediction.

2. Bagging: Random Forest uses a technique called bagging (bootstrap aggregating) to create different subsets of the training data by sampling with replacement. Each decision tree in the forest is trained on a different subset of the data.

3. Random Feature Selection: In addition to using random subsets of the data, Random Forest also randomly selects a subset of features at each split in a decision tree. This helps to introduce further randomness and prevent overfitting.

4. Decision Tree Combination: The predictions of individual decision trees in the forest are combined to make the final prediction. For classification tasks, it uses majority voting, where the class that receives the most votes from the decision trees is chosen. For regression tasks, it takes the average of the predicted values.

5. Feature Importance: Random Forest provides a measure of feature importance, indicating which features had the most significant impact on the predictions. This can be helpful for feature selection and understanding the importance of different features in the dataset.



6. Robustness to Overfitting: Random Forest is less prone to overfitting compared to individual decision trees. The combination of multiple decision trees helps to reduce the variance and improve the generalization ability of the model.

7. Handling of Missing Values and Outliers: Random Forest can handle missing values and outliers in the data. It is robust to noisy data and can provide reasonably accurate predictions even in the presence of missing values.

8. Versatility: Random Forest can be used for both classification and regression tasks. It can handle categorical and numerical features and can work well with large datasets.

9. Interpretability: While individual decision trees are interpretable, the interpretation of the Random Forest as a whole is more challenging. However, feature importance measures can provide insights into the relative importance of different features.

10. Hyperparameter Tuning: Random Forest has several hyperparameters that can be tuned to optimize the model's performance, such as the number of trees in the forest, the maximum depth of each tree, and the number of features considered at each split.

These key points highlight the main characteristics and advantages of the Random Forest Classifier in machine learning. It is a powerful and widely used algorithm known for its accuracy, robustness, and ability to handle various types of data.

Dataset:

https://www.kaggle.com/datasets/uciml/iris.csv

PythonCode:

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

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

from sklearn.metrics import accuracy\_score

from sklearn import tree

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2, random\_state=42)

# Create a Random Forest Classifier object

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

# Train the classifier on the training data

rf\_classifier.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = rf\_classifier.predict(X\_test)

# Calculate the accuracy of the classifier

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

print("Accuracy:", accuracy)

# Get one of the decision trees from the forest

tree\_to\_visualize = rf\_classifier.estimators\_[0]

# Visualize the decision tree

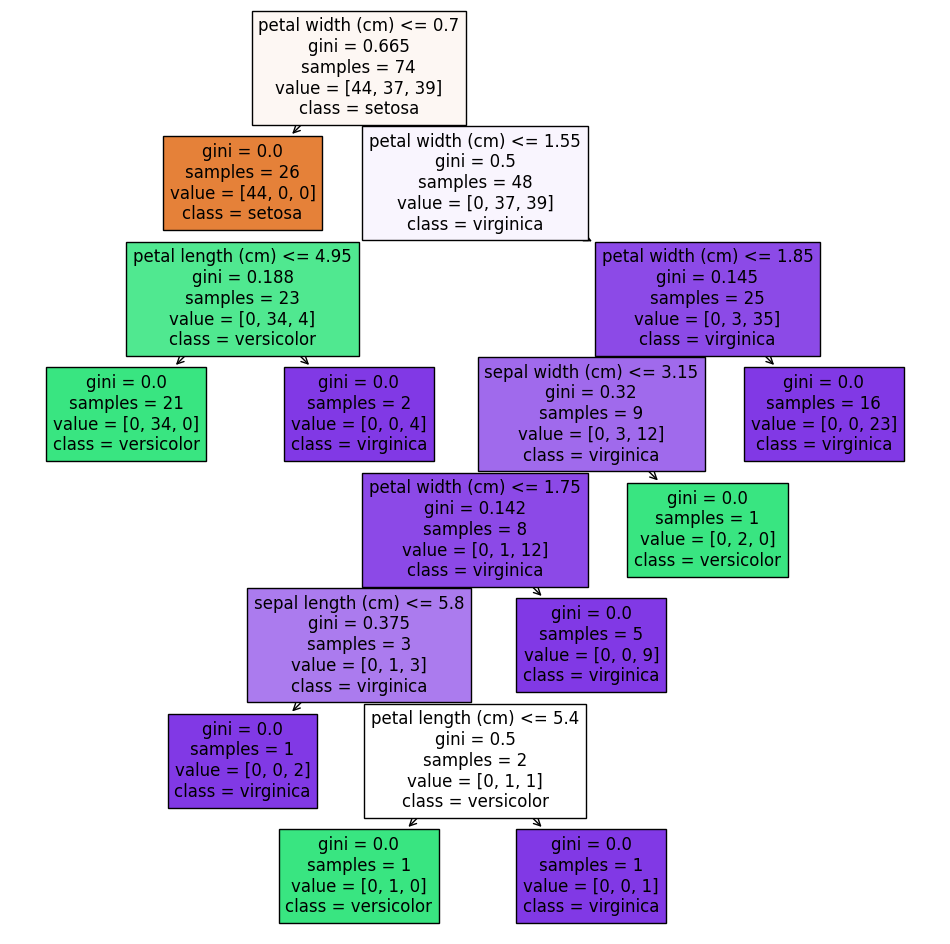
fig, ax = plt.subplots(figsize=(12, 12))

tree.plot\_tree(tree\_to\_visualize, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)

plt.show()

Output:

Accuracy: 1.0



**10.Aim: Random-Forest Regressor**

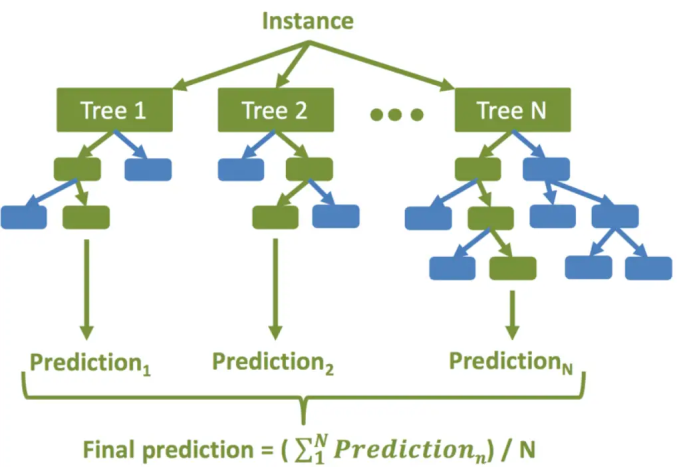
Description:

In Sklearn, random forest regression can be done quite easily by using **RandomForestRegressor** module of sklearn.ensemble module.

**Random Forest Regressor Hyperparameters (Sklearn)**

Hyperparameters are those parameters that can be fine-tuned for arriving at better accuracy of the machine learning model. Some of the main hyperparameters that RandomForestRegressor module of Sklearn provides are as follows –

* **n\_estimators:** It denotes the number of decision trees to be created in the random forest model. By default, it is 100.
* **criterion:** This denotes the criteria to be used to assess the quality of the split in decision trees. The supported values are ‘squared\_error’ (default), ‘absolute\_error’, ‘friedman\_mse’, ‘poisson’.
* **max\_depth:** It denotes the maximum depth of the tree. By default is None in which case nodes are expanded till all leaves become pure or until all leaves contain less than min\_samples\_split samples.
* **min\_samples\_split:** It denotes the minimum number of samples needed to split an internal node. By default, it is 2.
* **min\_samples\_leaf:** It denotes the minimum number of samples required to be at the leaf node. By default, it is 1.
* **max\_features:** It denotes the number of features to be considered for the best split. It can have values of ‘auto’, ‘sqrt’, ‘log2’, ‘None’, int, or float value. By default, it is 1.0
* **max\_samples:** It denotes the number of samples to be drawn from training data in bootstrap sampling.



The high-level steps for random forest regression are as followings –

1. Decide the number of decision trees N to be created.
2. Randomly take K data samples from the training set by using the bootstrapping method.
3. Create a decision tree using the above K data samples.
4. Repeat steps 2 and 3 till N decision trees are created.
5. For the new unseen data predict regression results using each of the N decision trees. Then take an average of these results to arrive at the final regression output.

Dataset:

Download from chrome: salary\_dataset.csv

Pythoncode:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.metrics import r2\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

df = pd.read\_csv('/content/salary\_dataset.csv')

df

plt.scatter(x = df['YearsExperience'], y = df['Salary'])

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

y = df.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

rf\_regressor = RandomForestRegressor(n\_estimators = 10, random\_state = 0)

rf\_regressor.fit(X\_train, y\_train)

y\_pred = rf\_regressor.predict(X\_test)

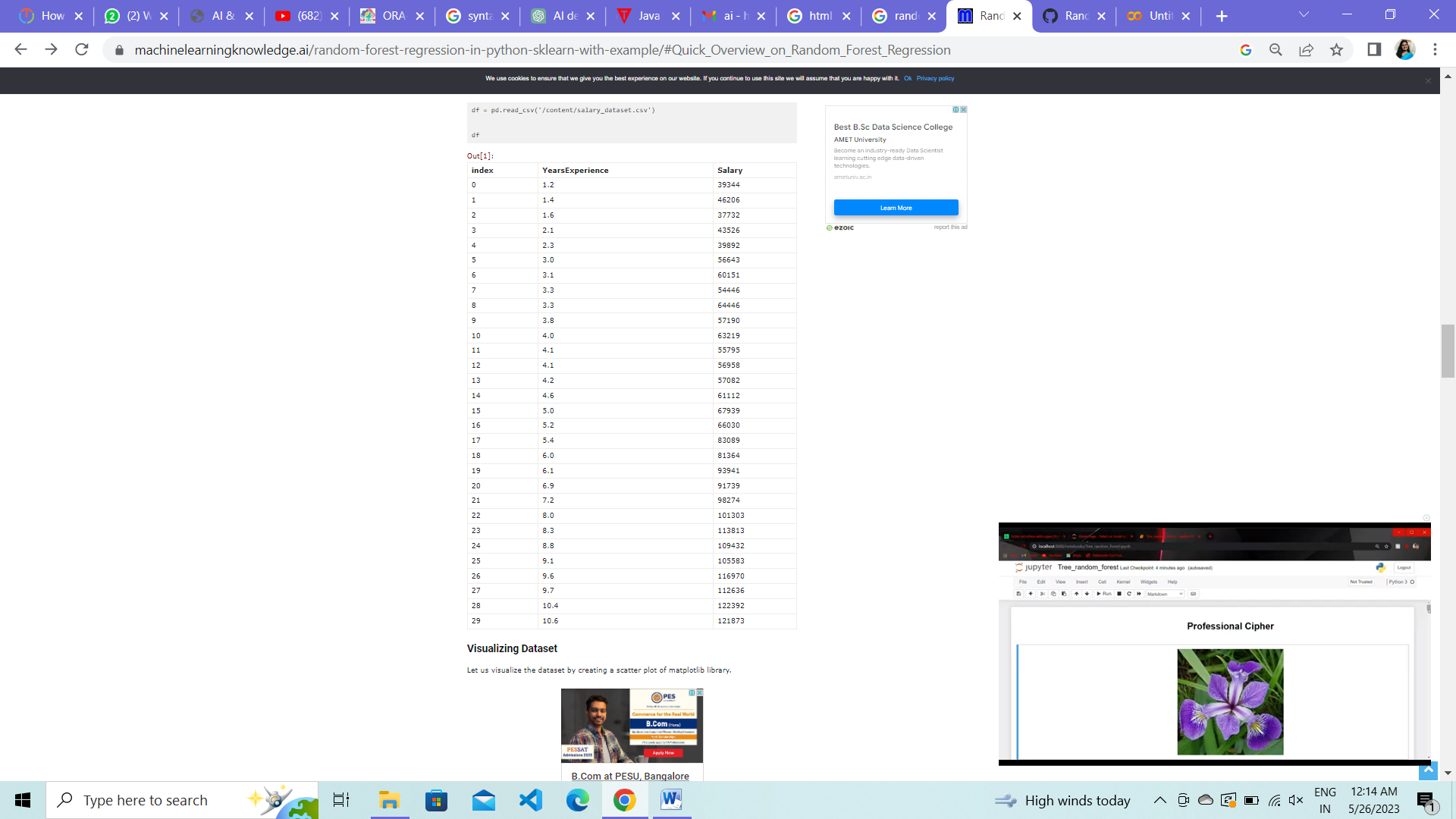
r2\_score(y\_test, y\_pred)

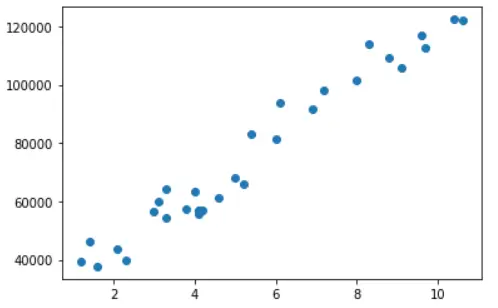
fig, ax = plt.subplots()

ax.scatter(X\_test,y\_test, color = "red")

ax.scatter(X\_test,y\_pred, color = "blue")

Output:





0.9815329041236582

