**Name :- Jayesh Vasant Falak**

**Roll No :-57**

**Practical No 2:-implements the find-s inductive learning algorithm**

Code:

import pandas as pd  
print(pd.\_\_version\_\_)

import numpy as np  
print(np.\_\_version\_\_)  
  
data = pd.read\_csv("C:/Users/Khemangi/Desktop/dataset1.CSV")  
print("Given Data set")  
print(data,"n")  
  
d=np.array(data)[:,:-1]  
print("n the attributes are:",d)  
  
target=np.array(data)[:,-1]  
print("n the target is :",target)  
  
def train(c,t):  
 for i,val in enumerate(t):  
 if val=='yes':  
 sp\_hp = c[i].copy()  
 break  
 print("initially hypothesis=")  
 print(sp\_hp,"\n")  
  
 for i,val in enumerate(c):  
 if t[i]=='yes':  
 for x in range(len(sp\_hp)):  
 if val[x]!=sp\_hp[x]:  
 sp\_hp[x]='?'  
 else:  
 pass  
 print("hypothesis is ",i,"=",sp\_hp)  
 return sp\_hp  
print("\n the final hypothesis is :",train(d,target))

**output:-**

1.5.3

1.24.2

Given Data set

sky temp humidity water wind forecast Enj\_sport

0 sunny warm normal warm strong same yes

1 sunny warm high warm strong same yes

2 rainy cold high warm strong change no

3 sunny warm high cool strong change yes

4 sunny cold high warm weak same no

5 sunny cold normal warm weak same yes n

n the attributes are: [['sunny' 'warm' 'normal' 'warm' 'strong' 'same']

['sunny' 'warm' 'high' 'warm' 'strong' 'same']

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

['sunny' 'warm' 'high' 'cool' 'strong' 'change']

['sunny' 'cold' 'high' 'warm' 'weak' 'same']

['sunny' 'cold' 'normal' 'warm' 'weak' 'same']]

n the target is : ['yes' 'yes' 'no' 'yes' 'no' 'yes']

initially hypothesis=

['sunny' 'warm' 'normal' 'warm' 'strong' 'same']

hypothesis is 0 = ['sunny' 'warm' 'normal' 'warm' 'strong' 'same']

hypothesis is 1 = ['sunny' 'warm' '?' 'warm' 'strong' 'same']

hypothesis is 2 = ['sunny' 'warm' '?' 'warm' 'strong' 'same']

hypothesis is 3 = ['sunny' 'warm' '?' '?' 'strong' '?']

hypothesis is 4 = ['sunny' 'warm' '?' '?' 'strong' '?']

hypothesis is 5 = ['sunny' '?' '?' '?' '?' '?']

the final hypothesis is : ['sunny' '?' '?' '?' '?' '?']

**Practical No:-3**

**Practical Name:-Implement the Candidate-Elimination Inductive Learning algorithm.**

**Name:-Jayesh Vasant Falak**

**Roll No:-057**

import numpy as np  
import pandas as pd  
data = pd.read\_csv('C:/Users/comp/Desktop/datafile.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 general\_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 i in range(len(specific\_h)):  
 if h[x]!= specific\_h[x]:  
 specific\_h[x] ='?'  
 general\_h[x][x] ='?'  
 else:  
 print("Instance is Negative")  
 for x in range(len(specific\_h)):  
 if h[x]!= specific\_h[x] and specific\_h[x]!='?':  
 general\_h[x][x] = specific\_h[x]  
 else:  
 general\_h[x][x] = '?'  
 print("Specific Boundary 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")

C:\Users\comp\AppData\Local\Programs\Python\Python310\python.exe C:/Users/comp/Desktop/ml.py/ml2.py

**OUTPUT:**

Instances are:

[['sunny' 'warm' 'high' 'cool' 'strong' 'same']

['sunny' 'warm' 'high' 'warm' 'strong' 'same']

['rainy' 'cold' 'low' 'warm' 'weak' 'change']

['rainy' 'cold' 'high' 'warm ' 'weak' 'cahnge']

['sunny ' 'cold' 'low' 'warm' 'strong' 'same']]

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

Initialization of specific\_h and general\_h

Specific Boundary: ['sunny' 'warm' 'high' 'cool' 'strong' 'same']

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

Instance 1 is ['sunny' 'warm' 'high' 'cool' 'strong' 'same']

Instance is Negative

Specific Boundary after 1 Instance is ['sunny' 'warm' 'high' 'cool' 'strong' 'same']

Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['sunny' 'warm' 'high' 'warm' 'strong' 'same']

Instance is Negative

Specific Boundary after 2 Instance is ['sunny' 'warm' 'high' 'cool' 'strong' 'same']

Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', 'cool', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is ['rainy' 'cold' 'low' 'warm' 'weak' 'change']

Instance is Negative

Specific Boundary after 3 Instance is ['sunny' 'warm' 'high' 'cool' 'strong' 'same']

Generic Boundary after 3 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', 'high', '?', '?', '?'], ['?', '?', '?', 'cool', '?', '?'], ['?', '?', '?', '?', 'strong', '?'], ['?', '?', '?', '?', '?', 'same']]

Instance 4 is ['rainy' 'cold' 'high' 'warm ' 'weak' 'cahnge']

Instance is Negative

Specific Boundary after 4 Instance is ['sunny' 'warm' 'high' 'cool' 'strong' 'same']

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

Instance 5 is ['sunny ' 'cold' 'low' 'warm' 'strong' 'same']

Instance is Negative

Specific Boundary after 5 Instance is ['sunny' 'warm' 'high' 'cool' 'strong' 'same']

Generic Boundary after 5 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', 'high', '?', '?', '?'], ['?', '?', '?', 'cool', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

['sunny' 'warm' 'high' 'cool' 'strong' 'same']

Final General\_h:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', 'high', '?', '?', '?'], ['?', '?', '?', 'cool', '?', '?']]

Process finished with exit code 0

**Name:- Jayesh Vasant Falak**

**Roll no: 057**

**Practical Name: Implementation Linear Regression Algorithm.**

**Practical No:-04**

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

import numpy as np

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y \* x) - n \* m\_y \* m\_x

SS\_xx = np.sum(x \* x) - n \* m\_x \* m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1 \* m\_x

return (b\_0, b\_1)

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12, 15])

# x = np.array([1, 2, 4, 6, 8, 10])

# y = np.array([2, 5, 8, 12, 15, 21])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

print(b)

# plotting regression line

#plot\_regression\_line(x, y, b)

y\_pred = b[0] + b[1] \* x

print("x input :", x)

print("Original Y:", y)

print("Y\_pred:", y\_pred)

er = np.mean(np.square(y-y\_pred))/2

print("Error=", er)

y\_prr = y\_pred

for i in range(len(y\_pred)):

y\_prr[i] = y\_pred[i] + er

print("Improved Y\_pred adding error:", y\_prr)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT:-**

C:\Users\comp\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/comp/PycharmProjects/pythonProject/regressiondemo.py

Estimated coefficients:

b\_0 = 0.9545454545454541

b\_1 = 1.2636363636363637

(0.9545454545454541, 1.2636363636363637)

x input : [ 0 1 2 3 4 5 6 7 8 9 10]

Original Y: [ 1 3 2 5 7 8 8 9 10 12 15]

Y\_pred: [ 0.95454545 2.21818157 3.48181818 4.74545455 6.00909091 7.27272727

8.53636364 9.8 11.06363636 12.32727273 13.59090909]

Error= 0.38801652892561994

Improved Y\_pred adding error: [ 1.34256198 2.60619857 3.86983471 5.13347107 6.39710744 7.6607438

8.92438017 10.18801653 11.45165289 12.71528926 13.97892562]

Process finished with exit code 0

**Name: Jayesh Vasant Falak**

**Roll No. :- 057**

**Practical Name : Write A Program To Implement Decision Tree Using Python/R/Programming Language Of Your Choice**

**Practical No :- 05.1**

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

import matplotlib.pyplot as plt  
import pandas as pd  
import sklearn.datasets  
data\_b = sklearn.datasets.load\_iris()  
df=pd.DataFrame(data\_b.data,columns=data\_b.feature\_names)  
df['target'] = data\_b.target  
#df['target']  
print(df)  
#print(data\_b)  
print("Dataset Labels=",data\_b.target\_names)  
from sklearn.tree import DecisionTreeClassifier  
from sklearn import metrics  
from sklearn import tree  
from sklearn.model\_selection import train\_test\_split  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(df[data\_b.feature\_names], df['target'])  
print(x\_train)  
print(x\_test)  
print(y\_train)  
print(y\_test)  
clf = DecisionTreeClassifier(max\_depth = 5,random\_state=1, criterion='gini') #'gini'  
clf = clf.fit(x\_train, y\_train)  
y\_pred = clf.predict(x\_test)  
print(y\_test, y\_pred)  
print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))  
fn=['sepal length (cm)','sepal width (cm)', 'petal length (cm)', 'petal width (cm)']  
cn=['setosa', 'versicolor', 'virginica']  
  
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4, 4), dpi = 300)  
tree.plot\_tree(clf, feature\_names = fn, class\_names = cn,filled = True); fig.savefig('dstimq.png')

OU sepal length (cm) sepal width (cm) ... petal width (cm) target

0 5.1 3.5 ... 0.2 0

1 4.9 3.0 ... 0.2 0

2 4.7 3.2 ... 0.2 0

3 4.6 3.1 ... 0.2 0

4 5.0 3.6 ... 0.2 0

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

145 6.7 3.0 ... 2.3 2

146 6.3 2.5 ... 1.9 2

147 6.5 3.0 ... 2.0 2

148 6.2 3.4 ... 2.3 2

149 5.9 3.0 ... 1.8 2

[150 rows x 5 columns]

Dataset Labels= ['setosa' 'versicolor' 'virginica']

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

13 4.3 3.0 1.1 0.1

123 6.3 2.7 4.9 1.8

11 4.8 3.4 1.6 0.2

31 5.4 3.4 1.5 0.4

40 5.0 3.5 1.3 0.3

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

146 6.3 2.5 5.0 1.9

99 5.7 2.8 4.1 1.3

148 6.2 3.4 5.4 2.3

43 5.0 3.5 1.6 0.6

97 6.2 2.9 4.3 1.3

[112 rows x 4 columns]

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

44 5.1 3.8 1.9 0.4

103 6.3 2.9 5.6 1.8

133 6.3 2.8 5.1 1.5

96 5.7 2.9 4.2 1.3

89 5.5 2.5 4.0 1.3

17 5.1 3.5 1.4 0.3

57 6.6 3.0 4.4 1.4

7 5.0 3.4 1.5 0.2

121 5.6 2.8 4.9 2.0

16 5.4 3.9 1.3 0.4

128 6.4 2.8 5.6 2.1

100 6.3 3.3 6.0 2.5

66 5.6 3.0 4.5 1.5

138 6.0 3.0 4.8 1.8

94 5.6 2.7 4.2 1.3

33 5.5 4.2 1.4 0.2

84 5.4 3.0 4.5 1.5

83 6.0 2.7 5.1 1.6

68 6.2 2.2 4.5 1.5

53 5.5 2.3 4.0 1.3

51 6.4 3.2 4.5 1.5

22 4.6 3.6 1.0 0.2

39 5.1 3.4 1.5 0.2

57 5.0 3.2 1.2 0.2

23 5.1 3.3 1.7 0.5

126 6.2 2.8 4.8 1.8

98 5.1 2.5 3.0 1.1

69 5.6 2.5 3.9 1.1

143 6.8 3.2 5.9 2.3

58 6.6 2.9 4.6 1.3

93 5.0 2.3 3.3 1.0

36 5.5 3.5 1.3 0.2

139 6.9 3.1 5.4 2.1

5 5.4 3.9 1.7 0.4

90 5.5 2.6 4.4 1.2

136 6.3 3.4 5.6 2.4

113 5.7 2.5 5.0 2.0

28 5.2 3.4 1.4 0.2

13 0

123 2

11 0

31 0

40 0

..

146 2

99 1

148 2

43 0

97 1

Name: target, Length: 112, dtype: int32

44 0

103 2

133 2

96 1

89 1

17 0

57 1

7 0

121 2

16 0

128 2

100 2

66 1

138 2

94 1

33 0

84 1

83 1

68 1

53 1

51 1

22 0

39 0

57 0

23 0

126 2

98 1

69 1

143 2

58 1

93 1

36 0

139 2

5 0

90 1

136 2

113 2

28 0

Name: target, dtype: int32

44 0

103 2

133 2

96 1

89 1

17 0

57 1

7 0

121 2

16 0

128 2

100 2

66 1

138 2

94 1

33 0

84 1

83 1

68 1

53 1

51 1

22 0

39 0

57 0

23 0

126 2

98 1

69 1

143 2

58 1

93 1

36 0

139 2

5 0

90 1

136 2

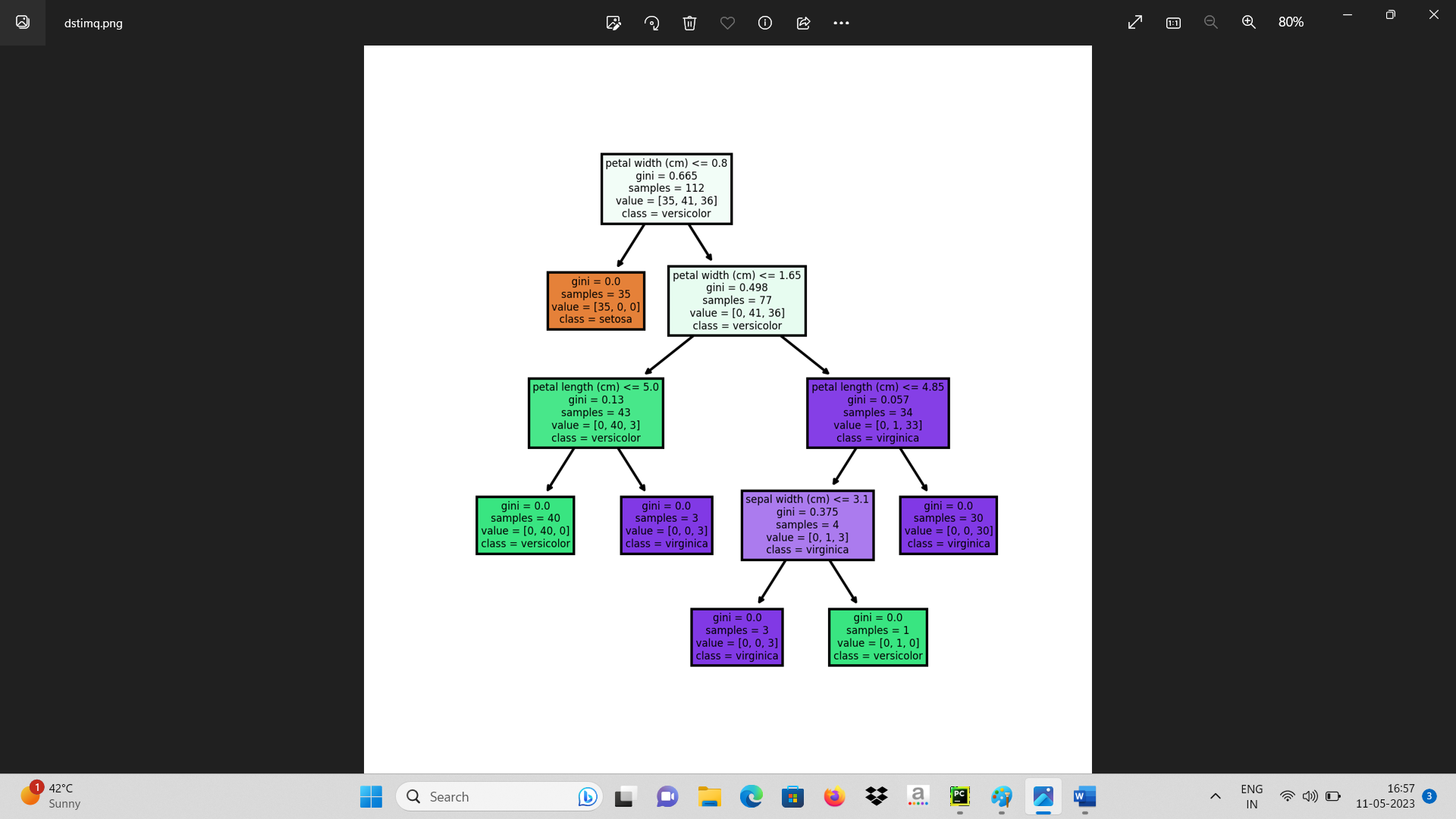
113 2

28 0

Name: target, dtype: int32 [0 2 2 1 1 0 1 0 2 0 2 2 1 2 1 0 1 2 1 1 1 0 0 0 0 2 1 1 2 1 1 0 2 0 1 2 2

0]

Accuracy: 0.9736842105263158TPUT:



**Name:- Jayesh Vasant Falak**

**Roll No:- 057**

**Practical No. :- 06**

**Practical Name :- . Implement Simple KNN Using Euclidean Distance In Python.**

from pandas import DataFrame  
from sklearn.datasets import load\_iris  
data\_b = load\_iris()  
df= DataFrame(data\_b.data, columns=data\_b.feature\_names)  
df['target'] = data\_b.target  
#print(df)  
#print(data\_b.DESCR)  
print("Dataset Labels=",data\_b.target\_names)  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, Y\_train, y\_test = train\_test\_split(df[data\_b.feature\_names], df['target'], random\_state=1)  
print(X\_train.head(6))  
print(Y\_train.head(6))  
print(X\_test.head())  
clf = KNeighborsClassifier(n\_neighbors=6)  
clf.fit(X\_train, Y\_train) # model is trained  
y\_pred=clf.predict(X\_test)  
#print(y\_test, y\_pred)  
print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))  
cm = confusion\_matrix(y\_test, y\_pred)  
print("Confusion Matrix:")  
print(cm)

**OUTPUT:**

Dataset Labels= ['setosa' 'versicolor' 'virginica']

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

54 6.5 2.8 4.6 1.5

108 6.7 2.5 5.8 1.8

112 6.8 3.0 5.5 2.1

17 5.1 3.5 1.4 0.3

119 6.0 2.2 5.0 1.5

103 6.3 2.9 5.6 1.8

54 1

108 2

112 2

17 0

119 2

103 2

Name: target, dtype: int32

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm)

14 5.8 4.0 1.2 0.2

98 5.1 2.5 3.0 1.1

57 6.6 3.0 4.4 1.4

16 5.4 3.9 1.3 0.4

131 7.9 3.8 6.4 2.0

Accuracy: 1.0

Confusion Matrix:

[[13 0 0]

[ 0 16 0]

[ 0 0 9]]

**Name:-Jayesh Vasant Falak**

**Roll No:- 057**

**Practical No. :- 07**

**Practical Name :- . Write A Program To Implement K-Nearest Neighbour Algorithm To Classify The Iris Dataset. Print Both Correct And Wrong Predictions. Java/Python Ml Library Classes Can Be Used For This Problem..**

**CODE: FOR BREAST CANCER DATA SET**

from pandas import DataFrame  
# from sklearn.datasets import load\_iris  
from sklearn.datasets import load\_breast\_cancer  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn import metrics  
from sklearn.metrics import confusion\_matrix  
data\_b = load\_breast\_cancer()  
df= DataFrame(data\_b.data, columns=data\_b.feature\_names)  
df['target'] = data\_b.target  
#print(df)  
#print(data\_b.DESCR)  
print("Dataset Labels=",data\_b.target\_names)  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, Y\_train, y\_test = train\_test\_split(df[data\_b.feature\_names], df['target'], random\_state=1)  
print(X\_train.head(6))  
print(Y\_train.head(6))  
print(X\_test.head())  
clf = KNeighborsClassifier(n\_neighbors=6)  
clf.fit(X\_train, Y\_train) # model is trained  
y\_pred=clf.predict(X\_test)  
#print(y\_test, y\_pred)  
print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))  
cm = confusion\_matrix(y\_test, y\_pred)  
print("Confusion Matrix:")  
print(cm)

**OUTPUT:**

Dataset Labels= ['malignant' 'benign']

mean radius mean texture ... worst symmetry worst fractal dimension

562 15.22 30.62 ... 0.4089 0.14090

291 14.96 19.10 ... 0.2962 0.08472

16 14.68 20.13 ... 0.3029 0.05716

546 10.32 16.57 ... 0.2681 0.07399

293 11.85 17.46 ... 0.3101 0.07007

570 11.66 17.07 ... 0.2731 0.06575

[6 rows x 30 columns]

562 0

291 1

16 0

546 1

293 1

570 1

Name: target, dtype: int32

mean radius mean texture ... worst symmetry worst fractal dimension

421 14.69 13.98 ... 0.2577 0.09208

47 13.17 18.66 ... 0.3900 0.11790

292 12.95 16.02 ... 0.3380 0.09584

186 18.31 18.58 ... 0.3206 0.06938

414 15.13 29.81 ... 0.3233 0.06165

[5 rows x 30 columns]

Accuracy: 0.9370629370629371

Confusion Matrix:

[[51 4]

[ 5 83]]

**Name: -Jayesh Vasant Falak**

**Roll No: 057**

**Practical No:0 8**

**Practical Name: Write a Program for Confusion Matrix and calculate Precision, Recall, F-Measure**

**------------------------------------------------------------------------------------**

**Code:**

from sklearn.datasets import load\_iris, load\_breast\_cancer

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

from sklearn.neighbors import KNeighborsClassifier

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

# Load the Irish dataset

iris = load\_iris()

X\_iris = iris.data

y\_iris = iris.target

# Split the Irish dataset into training and testing sets

X\_train\_iris, X\_test\_iris, y\_train\_iris, y\_test\_iris = train\_test\_split(X\_iris, y\_iris, test\_size=0.2, random\_state=42)

# Train the KNN classifier on the Irish dataset

knn\_iris = KNeighborsClassifier()

knn\_iris.fit(X\_train\_iris, y\_train\_iris)

# Make predictions on the Irish testing set

y\_pred\_iris = knn\_iris.predict(X\_test\_iris)

# Calculate the confusion matrix for Irish dataset

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

print("Confusion Matrix (Irish Dataset):")

print(cm\_iris)

# Calculate precision, recall, and F-measure for Irish dataset

precision\_iris = precision\_score(y\_test\_iris, y\_pred\_iris, average='macro')

recall\_iris = recall\_score(y\_test\_iris, y\_pred\_iris, average='macro')

f1\_iris = f1\_score(y\_test\_iris, y\_pred\_iris, average='macro')

print("Precision (Irish Dataset):", precision\_iris)

print("Recall (Irish Dataset):", recall\_iris)

print("F-measure (Irish Dataset):", f1\_iris)

# Load the Breast Cancer dataset

cancer = load\_breast\_cancer()

X\_cancer = cancer.data

y\_cancer = cancer.target

# Split the Breast Cancer dataset into training and testing sets

X\_train\_cancer, X\_test\_cancer, y\_train\_cancer, y\_test\_cancer = train\_test\_split(X\_cancer, y\_cancer, test\_size=0.2, random\_state=42)

# Train the KNN classifier on the Breast Cancer dataset

knn\_cancer = KNeighborsClassifier()

knn\_cancer.fit(X\_train\_cancer, y\_train\_cancer)

# Make predictions on the Breast Cancer testing set

y\_pred\_cancer = knn\_cancer.predict(X\_test\_cancer)

# Calculate the confusion matrix for Breast Cancer dataset

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

print("\nConfusion Matrix (Breast Cancer Dataset):")

print(cm\_cancer)

# Calculate precision, recall, and F-measure for Breast Cancer dataset

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

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

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

print("Precision (Breast Cancer Dataset):", precision\_cancer)

print("Recall (Irish Dataset):", recall\_cancer)

print("F-measure (Irish Dataset):", f1\_cancer)

**Name :- Jayesh Vasant Falak**

**Roll No. :- 57**

**Practical No.:- 9**

**Practical Name:- Write a program for linear regression and find parameters like Sum of Squared Errors (SSE), Total Sum of Squares (SST), R2, Adjusted R2, etc.**

import numpy as np

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

# Input data

X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

y = np.array([3, 4, 5, 6])

model = LinearRegression() # Create a linear regression model

model.fit(X, y) # Fit the model to the data

y\_pred = model.predict(X) # Predict the output

sse = np.sum((y\_pred - y) \*\* 2) # Calculate SSE (Sum of Squared Errors)

sst = np.sum((y - np.mean(y)) \*\* 2) # Calculate SST (Total Sum of Squares)

r2 = r2\_score(y, y\_pred) # Calculate R2 score

# Calculate adjusted R2

n = X.shape[0] # Number of samples

p = X.shape[1] # Number of predictors

adjusted\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

# Print the results

print("Sum of Squared Errors(SSE):- ", sse)

print("Total Sum of Squares(SST):- ", sst)

print("R Square(R2):- ", r2)

print("Adjusted Square(R2):- ", adjusted\_r2 )

**OUTPUT:**

Sum of Squared Errors(SSE):- 0.0

Total Sum of Squares(SST):- 5.0

R Square(R2):- 1.0

Adjusted Square(R2):- 1.0

**Name :- Jayesh Vasant Falak**

**Roll No.:-057**

**Practical No:-10 Practical Name:-Write the program to implement the naive Bayesian Classifier for a sample training dataset stored as a .CSV file. Compute the accuracy of the classifier considering a few test dataset.**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn import datasets  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.metrics import confusion\_matrix  
iris = datasets.load\_iris() *#load dataset*x = iris.data *#input*y = iris.target *#traget*print("Features :", iris['feature\_names'])  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)  
NB = GaussianNB()  
NB.fit(x\_train, y\_train)  
y\_pred = NB.predict(x\_test)  
cm = confusion\_matrix(y\_test,y\_pred)  
print("Confusion Matrix")  
print(cm)

**OUTPUT:**

Features : ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Confusion Matrix

[[13 0 0]

[ 0 16 0]

[ 0 0 9]]

**Name :- Jayesh Vasant Falak**

**Roll No :- 57**

**Program No. :- 11.2**

**Practical Name :- Write a Program for Fuzzy c-means clustering in python.**

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

# Generate some example data

np.random.seed(0)

data = np.random.rand(100, 2)

# Define the number of clusters

n\_clusters = 3

# Apply fuzzy c-means clustering

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

data.T, n\_clusters, 2, error=0.005, maxiter=1000, init=None)

# Predict cluster membership for each data point

cluster\_membership = np.argmax(u, axis=0)

# Print the cluster centers

print('Cluster Centers:', cntr)

# Print the cluster membership for each data point

print('Cluster Membership:', cluster\_membership)

**Output :-**

Cluster Centers: [[0.22645397 0.71840176]

[0.52083891 0.18668653]

[0.76252289 0.60239021]]

Cluster Membership: [2 2 0 0 2 2 2 1 0 2 2 0 0 0 1 0

0 0 2 2 1 1 2 1 1 2 1 1 1 1 1 1 0 1 1 2 2

1 1 1 1 0 1 1 2 0 0 1 1 1 1 2 0 2 0 0 1 2 2 2 2 2 0

0 1 2 1 2 2 2 2 0 2 0

2 0 0 0 2 1 2 2 2 0 1 1 1 1 0 1 0 1 2 2 1 1 0 2 1 0]

**Name :- Jayesh Vasant Falak**

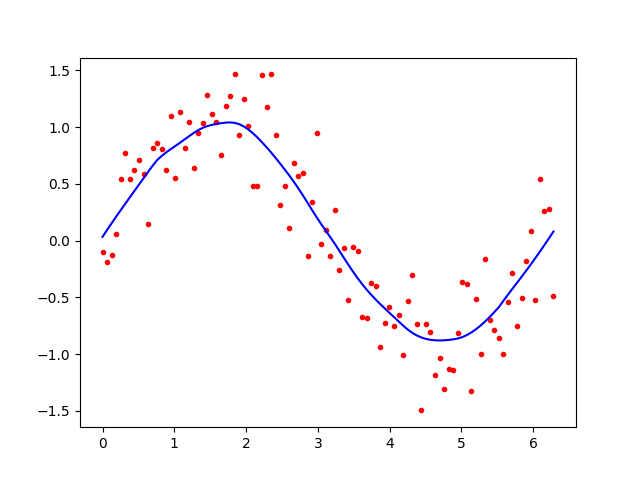
**Roll No :- 57**

**Practical No. :- 12**

**Practical Name :- Implement the non-parametric locally weighted regression algorithm in order to fit data points. select the appropriate data set for your experiment and draw graphs.**

from math import ceil  
import numpy as np  
from scipy import linalg  
  
  
def lowess(x, y, f, iterations):  
 n = len(x)  
 r = int(ceil(f \* n))  
 h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]  
 w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)  
 w = (1 - w \*\* 3) \*\* 3  
 yest = np.zeros(n)  
 delta = np.ones(n)  
 for iteration in range(iterations):  
 for i in range(n):  
 weights = delta \* w[:, i]  
 b = np.array([np.sum(weights \* y), np.sum(weights \* y \* x)])  
 A = np.array([[np.sum(weights), np.sum(weights \* x)], [np.sum(weights \* x), np.sum(weights \* x \* x)]])  
 beta = linalg.solve(A, b)  
 yest[i] = beta[0] + beta[1] \* x[i]  
  
 residuals = y - yest  
 s = np.median(np.abs(residuals))  
 delta = np.clip(residuals / (6.0 \* s), -1, 1)  
 delta = (1 - delta \*\* 2) \*\* 2  
  
 return yest  
  
  
import math  
  
n = 100  
x = np.linspace(0, 2 \* math.pi, n)  
y = np.sin(x) + 0.3 \* np.random.randn(n)  
f = 0.25  
iterations = 3  
yest = lowess(x, y, f, iterations)  
  
import matplotlib.pyplot as plt  
plt.plot(x, y, "r.")  
plt.plot(x, yest, "b-")  
plt.show()

**OUTPUT:**



**Name :- Jayesh Vasant Falak**

**Roll no :- 057**

**Practical No.: 13.1**

**Practical Name: Construction Of simple Neural Network using Python**

**Code:**

import numpy as np

from scipy.special import expit as activation\_function

from scipy.stats import truncnorm

# define the network

# generate numbers within a truncated (bounded)

# normal Distribution

def truncated\_normal(mean=0, sd=1, low=0, upp=10):

return truncnorm((low - mean) / sd, (upp - mean) / sd, loc=mean, scale=sd)

# creat the Network class and define the arguments:

# set the no. of neurons/nodes for each layer

# and initialize the weight matrices

class Nnetwork:

def \_\_init\_\_(self, no\_of\_in\_nodes, no\_of\_out\_nodes, no\_of\_hidden\_nodes, learning\_rate):

self.no\_of\_in\_nodes = no\_of\_in\_nodes

self.no\_of\_out\_nodes = no\_of\_out\_nodes

self.no\_of\_hidden\_nodes = no\_of\_hidden\_nodes

self.learning\_rate = learning\_rate

self.create\_weight\_matrices()

def create\_weight\_matrices(self):

"""A method to initialize the weight matrices of the neural network"""

rad = 1 / np.sqrt(self.no\_of\_in\_nodes) # rad = 0.2707

x = truncated\_normal(mean=0, sd=1, low=-rad, upp=rad)

self.weight\_in\_hidden = x.rvs((self.no\_of\_hidden\_nodes, self.no\_of\_in\_nodes))

print("weights\_in\_hidden = ", self.weight\_in\_hidden)

rad = 1/np.sqrt(self.no\_of\_hidden\_nodes)

x = truncated\_normal(mean=0, sd=1, low=-rad, upp=rad)

self.weight\_in\_hidden\_out = x.rvs((self.no\_of\_out\_nodes, self.no\_of\_hidden\_nodes))

print("weights\_in\_hidden\_out = ", self.weight\_in\_hidden\_out)

def train(self, input\_vector, target\_vector):

pass

def run(self, input\_vector):

input\_vector = np.array(input\_vector, ndmin=2).T

print("Input = ", input\_vector)

input\_hidden = activation\_function(self.weight\_in\_hidden @ input\_vector)

print("Hidden = ", input\_hidden)

output\_vector = activation\_function(self.weight\_in\_hidden\_out @ input\_hidden)

print("Output = ", output\_vector)

return output\_vector

simple\_network = Nnetwork(no\_of\_in\_nodes=2, no\_of\_out\_nodes=2, no\_of\_hidden\_nodes=4, learning\_rate=0.6)

#run simple network for arrays, lists and tuples with shape (2):

y = simple\_network.run([2,3])

print("Y = ", y)

**OUTPUT”:**

weights\_in\_hidden = [[-0.68798443 0.29425766]

[ 0.57363879 -0.64646032]

[-0.38809421 0.07104818]

[-0.23288421 0.26427463]]

weights\_in\_hidden\_out = [[ 0.12718945 -0.15067287 -0.36574728 0.3725497 ]

[-0.09102931 -0.22077172 0.40025881 -0.32165789]]

Input = [[2]

[3]]

Hidden = [[0.37915865]

[0.31171721]

[0.36284346]

[0.58104257]]

Output = [[0.52124119]

[0.46381691]]

Y = [[0.52124119]

[0.46381691]]

**Name: Jayesh Vasant Falak**

**Roll No: -57**

**Practical No: 13.2**

**Practical Name: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

**------------------------------------------------------------------------------------------**

# classification of iris data set by aplying artificial neural network using Back-propogation algorithm

**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn.datasets **import** load\_iris

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

**import** matplotlib.pyplot **as** plt

# load dataset

data = load\_iris()

# Get features and target

x = data.data

y = data.target

print(**"Y="**, y)

y = pd.get\_dummies(y).values

print(y[:3])  
# split data into train and test data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=20, random\_state=4)

# initialize variable

learning\_rate = 0.1

iteration = 6000

N = y\_train.size

# number of input features

input\_size = 4

# number of hidden layers neurons

hidden\_size = 2

# mo. of neurons at output layers

output\_size = 3

results = pd.DataFrame(columns=[**"mse"**, **"accuracy"**])

# initialize weights

np.random.seed(10)

# initialiizing weight for the hidden layers

W1 = np.random.normal(scale=0.5, size=(input\_size, hidden\_size))

print(**"weight 1"**, W1)

# initializing weight for the output layers

W2 = np.random.normal(scale=0.5, size=(hidden\_size, output\_size))

print(**"weight 2"**, W2)

**def** sigmoid(x):

**return** 1/(1 + np.exp(-x))

**def** mean\_squared\_error(y\_pred, y\_true):

**return** (((y\_pred - y\_true) \*\* 2).sum()) / (2 \* y\_pred.size)

**def** accuracy(y\_pred, y\_true):

acc = y\_pred.argmax(axis=1) == y\_true.argmax(axis=1)

**return** acc.mean()

**for** itr **in** range(iteration):

# feedforward propagation

# on hidden layer

Z1 = np.dot(x\_train, W1)

A1 = sigmoid(Z1)

# on output layer

Z2 = np.dot(A1, W2)

A2 = sigmoid(Z2)

# calculating error

mse = mean\_squared\_error(A2, y\_train)

acc = accuracy(A2, y\_train)

results = results.\_append({**"mse"**: mse, **"accuracy"**: acc}, ignore\_index=**True**)

# backpropagation

E1 = A2 - y\_train

dw1 = E1 \* A2 \* (1 - A2)

E2 = np.dot(dw1, W2.T)

dw2 = E2 \* A1 \* (1 - A1)

# weight updates

W2\_update = np.dot(A1.T, dw1) / N

W1\_update = np.dot(x\_train.T, dw2) / N

W2 = W2 - learning\_rate \* W2\_update

W1 = W1 - learning\_rate \* W1\_update

results.mse.plot(title=**"Mean squared Error"**)

results.accuracy.plot(title=**"Accuracy"**)

# feedforward

Z1 = np.dot(x\_test, W1)

A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)

A2 = sigmoid(Z2)

acc = accuracy(A2, y\_test)

print(**"Accuracy: {}"**.format(acc))

**OUTPUT:**

C:\Users\MCA-I\_ML\Scripts\python.exe C:/Users/PycharmProjects/MCA-I\_ML/nural\_network\_Backpropa\_algo.py

Y= [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

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

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

[[ True False False]

[ True False False]

[ True False False]]

weight 1 [[ 0.66579325 0.57763949]

[-0.77270015 -0.00419192]

[ 0.31066799 -0.36004278]

[ 0.13257579 0.05427426]]

weight 2 [[ 0.00214572 -0.08730011 0.21651309]

[ 0.60151869 -0.45753284 0.51413704]]