INDEX

Subject:- CA LAB-VII(A): LAB on Machine Learning

Practical – 1: Introduction to pycharm, Pandas Library, DataFrames, And Loading CSV File in DataFrame

---

```
import pandas as pd
'''pd.__version__'''

df1 = pd.DataFrame({"A": [1, 2, 3], "B": [2, 3, 4]}, index=[0, 1, 2])
print("df1:\n", df1)

df2 = pd.DataFrame({"B": [4, 5, 7], "C": ["x", "y", "z"]}, index=[4, 5, 6])
print("\ndf2:\n", df2)

df3 = df1.combine_first(df2)
print("\n combination of df1 and df2:\n", df3)


classes = pd.Series(["mathematics", "chemistry", "physics", "history", "geography",
"german"])
grades = pd.Series([90, 54, 77, 22, 25, 40])
year = pd. Series([2015, 2016, 2017, 2018, 2019, 2020])
df4 = pd. DataFrame({"Classes": classes, "Grades": grades, "Year": year})
print("\n", df4)

# upload a csv file  in sample_data section
# load the .csv in data frame

data_frame = pd.read_csv("C:/Users/sejal/PycharmProjects/dataset.csv")
print("\n", data_frame)
```

OUTPUT :

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-I_ML/1_prat.py
df1:
   A  B
0  1  2
1  2  3
2  3  4

df2:
   B  C
4  4  x
5  5  y
6  7  z


 combination of df1 and df2:
```

	A	B	C
0	1.0	2	NaN
1	2.0	3	NaN
2	3.0	4	NaN
4	NaN	4	x
5	NaN	5	y
6	NaN	7	z

	Classes	Grades	Year
0	mathematics	90	2015
1	chemistry	54	2016
2	physics	77	2017
3	history	22	2018
4	geography	25	2019
5	german	40	2020

	sky	temp	humidity	water	wind	forcast	enjoy-sport
0	sunny	warm	high	cool	strong	same	yes
1	sunny	warm	high	warm	strong	same	yes
2	rainy	cold	low	warm	weak	change	no
3	rainy	cold	high	warm	weak	change	no
4	sunny	warm	high	warm	strong	same	yes
5	sunny	cold	high	warm	strong	same	no

Practical - 2.: Implement the find-S inductive learning algorithm.

```
import pandas as pd
import numpy as np

# To read the data in csv file
data = pd.read_csv("C:/Users/comp273/Desktop/pract1ML.csv")
print("The Data-set For Enjoy Sport Example is:- ")
print(data)

# Making an array of all the attributes
d = np. array(data)[:, :-1]
print("\nThe Attributes are :- ")
print(d)

# Segragating the target that has positive and negative example
target = np.array(data)[:, -1]
print("\nThe Target is :- ")
print(target)

# Find S-algorithm - initial and f hypothesis
def train(c, t):
    for i, val in enumerate(t):
        if val == "yes":
            sp_hp = d[i].copy()
            break
    print("\nInitial Hypothesis:- ")
    print(sp_hp, "\n")


    for i, val in enumerate(c):
        if target[i] == "yes":
            for x in range(len(sp_hp)):
                if sp_hp[x] != val[x]:
                    sp_hp[x] = "?"
                else:
                    pass
        print("Hypothesis is:- ", i, "= ", sp_hp)
    return sp_hp

print("\nFinal Hypothesis is :- ", train(d, target))
```

OUTPUT:

```
C:\Users\comp273\PycharmProjects\ML_107\venv\Scripts\python.exe
C:/Users/comp273/PycharmProjects/ML_107/find_s_algo.py

The Data-set For Enjoy Sport Example is:-
     Sky AirTemp Humidity    Wind Water  Forcast EnjoySport
0  sunny    warm   normal  strong  warm     same        yes
1  sunny    warm     high  strong  warm     same        yes
```

```
2  sunny    cold     high  strong   warm  change        yes
3  rainy    cold   normal  strong   cool  change        no
4  sunny    cold     high    weak   warm  change        no
5  sunny    cold   normal    weak   warm    same        yes
6  rainy    warm     high    weak   cool  change        no

The Attributes are :-
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['sunny' 'cold' 'high' 'strong' 'warm' 'change']
 ['rainy' 'cold' 'normal' 'strong' 'cool' 'change']
 ['sunny' 'cold' 'high' 'weak' 'warm' 'change']
 ['sunny' 'cold' 'normal' 'weak' 'warm' 'same']
 ['rainy' 'warm' 'high' 'weak' 'cool' 'change']]

The Target is :-
['yes' 'yes' 'yes' 'no' 'no' 'yes' 'no']

Initial Hypothesis:-
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Hypothesis is:-  0 =  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Hypothesis is:-  1 =  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Hypothesis is:-  2 =  ['sunny' '?' '?' 'strong' 'warm' '?']
Hypothesis is:-  3 =  ['sunny' '?' '?' 'strong' 'warm' '?']
Hypothesis is:-  4 =  ['sunny' '?' '?' 'strong' 'warm' '?']
Hypothesis is:-  5 =  ['sunny' '?' '?' '?' 'warm' '?']
Hypothesis is:-  6 =  ['sunny' '?' '?' '?' 'warm' '?']

Final Hypothesis is :-  ['sunny' '?' '?' '?' 'warm' '?']

Process finished with exit code 0
```

Practical - 2.: Implement the Candidate-Elimination Inductive Learning algorithm.

```
import numpy as np
import pandas as pd
data = pd.read_csv("C:/Users/sejal/OneDrive/Desktop/FyMca Sem II Notes/"
                   "Practical Practice/CA LAB-VII(A) ML/Enjoy-sportExample.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_Hypothesis and General_Hypothesis")
    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("Instance", i+1, "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        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_Hypothesis: ", s_final, sep="\n")
print("Final General_Hypothesis: ", g_final, sep="\n")
```

OUTPUT:

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-I_ML/candidate_elimination.py

Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['sunny' 'cold' 'high' 'strong' 'warm' 'change']
 ['rainy' 'cold' 'normal' 'strong' 'cool' 'change']
 ['sunny' 'cold' 'high' 'weak' 'warm' 'change']
 ['sunny' 'cold' 'normal' 'weak' 'warm' 'same']
 ['rainy' 'warm' 'high' 'weak' 'cool' 'change']]

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

Initialization of Specific_Hypothesis and General_Hypothesis

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

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance 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']
Instance is Positive
Specific Boundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 3 is  ['sunny' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Positive
Specific Boundary after  3 Instance is  ['sunny' '?' '?' 'strong' 'warm' '?']
Generic Boundary after  3 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 4 is  ['rainy' 'cold' 'normal' 'strong' 'cool' 'change']
Instance is Negative
Specific Boundary after  4 Instance is  ['sunny' '?' '?' 'strong' 'warm' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'warm', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 5 is  ['sunny' 'cold' 'high' 'weak' 'warm' 'change']
```

Instance is Negative
Specific Boundary after 5 Instance is ['sunny' '?' '?' 'strong' 'warm' '?']
Generic Boundary after 5 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', 'strong', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 6 is ['sunny' 'cold' 'normal' 'weak' 'warm' 'same']
Instance is Positive
Specific Boundary after 6 Instance is ['sunny' '?' '?' '?' 'warm' '?']
Generic Boundary after 6 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 7 is ['rainy' 'warm' 'high' 'weak' 'cool' 'change']
Instance is Negative
Specific Boundary after 7 Instance is ['sunny' '?' '?' '?' 'warm' '?']
Generic Boundary after 7 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'warm', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_Hypothesis:
['sunny' '?' '?' '?' 'warm' '?']
Final General_Hypothesis:
[['sunny', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'warm', '?']]

Process finished with exit code 0

Practical - 4.: Finding the Estimated coefficient and regression coeficiant

```
import numpy as np
def estimated_coef(x, y):
    # number of observation\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])
    # estimating coefficients
    b = estimated_coef(x, y)
    print("Estimated coefficients :-\n b_0 = {} \n b_1 = {}".format(b[0], b[1]))
    y_pred = b[0] + b[1] * x
    print("x input :", x)
    print("original y : ", y_pred)
    e = y - y_pred
    merror = np.sum(e*e)
    n = np.size(x)
    print("mean square error = ", merror/(2 * n))

if __name__ == "__main__":
    main()
```

OUTPUT:

```
C:\Users\comp\mca107\venv\Scripts\python.exe C:/Users/comp/mca107/ml_pract4.py
Estimated coefficients :-
 b_0 = 0.9545454545454541
 b_1 = 1.2636363636363637
x input : [ 0  1  2  3  4  5  6  7  8  9 10]
original y :  [ 0.95454545  2.21818182  3.48181818  4.74545455  6.00909091  7.27272727
  8.53636364  9.8        11.06363636 12.32727273 13.59090909]
mean square error =  0.38801652892561994
```

Practical - 5.1: Write a program to implement Decision tree using the Python/R/Programming language of your choice

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris  # load_iris
data_b = load_iris()  # lo
df = pd.DataFrame(data_b.data, columns=data_b.feature_names)
df['target'] = data_b.target
# df['target']
print(df)
print("Dataset Labels=", data_b.target_names)

from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
# import numpy as np
from sklearn import tree
X_train, X_test, Y_train, y_test = train_test_split(df[data_b.feature_names], df['target'],
random_state=1)
print(X_train)
print(X_test)
print(Y_train)
print(y_test)

clf = DecisionTreeClassifier(max_depth=5, random_state=1, criterion='gini')  #
'gini'/'entropy'
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))

# tree.plot_tree(clf)
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('Dicision_tree.png')
```

OUTPUT:

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-
I_ML/Decision_tree.py

    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)
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
..	...	...	...	...
133	6.3	2.8	5.1	1.5
137	6.4	3.1	5.5	1.8
72	6.3	2.5	4.9	1.5
140	6.7	3.1	5.6	2.4
37	4.9	3.6	1.4	0.1

[112 rows x 4 columns]

	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
75	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
56	6.3	3.3	4.7	1.6
141	6.9	3.1	5.1	2.3
44	5.1	3.8	1.9	0.4
29	4.7	3.2	1.6	0.2
120	6.9	3.2	5.7	2.3
94	5.6	2.7	4.2	1.3
5	5.4	3.9	1.7	0.4
102	7.1	3.0	5.9	2.1
51	6.4	3.2	4.5	1.5
78	6.0	2.9	4.5	1.5
42	4.4	3.2	1.3	0.2
92	5.8	2.6	4.0	1.2

66	5.6	3.0	4.5	1.5
31	5.4	3.4	1.5	0.4
35	5.0	3.2	1.2	0.2
90	5.5	2.6	4.4	1.2
84	5.4	3.0	4.5	1.5
77	6.7	3.0	5.0	1.7
40	5.0	3.5	1.3	0.3
125	7.2	3.2	6.0	1.8
99	5.7	2.8	4.1	1.3
33	5.5	4.2	1.4	0.2
19	5.1	3.8	1.5	0.3
73	6.1	2.8	4.7	1.2
146	6.3	2.5	5.0	1.9
91	6.1	3.0	4.6	1.4
135	7.7	3.0	6.1	2.3
69	5.6	2.5	3.9	1.1
128	6.4	2.8	5.6	2.1
114	5.8	2.8	5.1	2.4
48	5.3	3.7	1.5	0.2
53	5.5	2.3	4.0	1.3
28	5.2	3.4	1.4	0.2

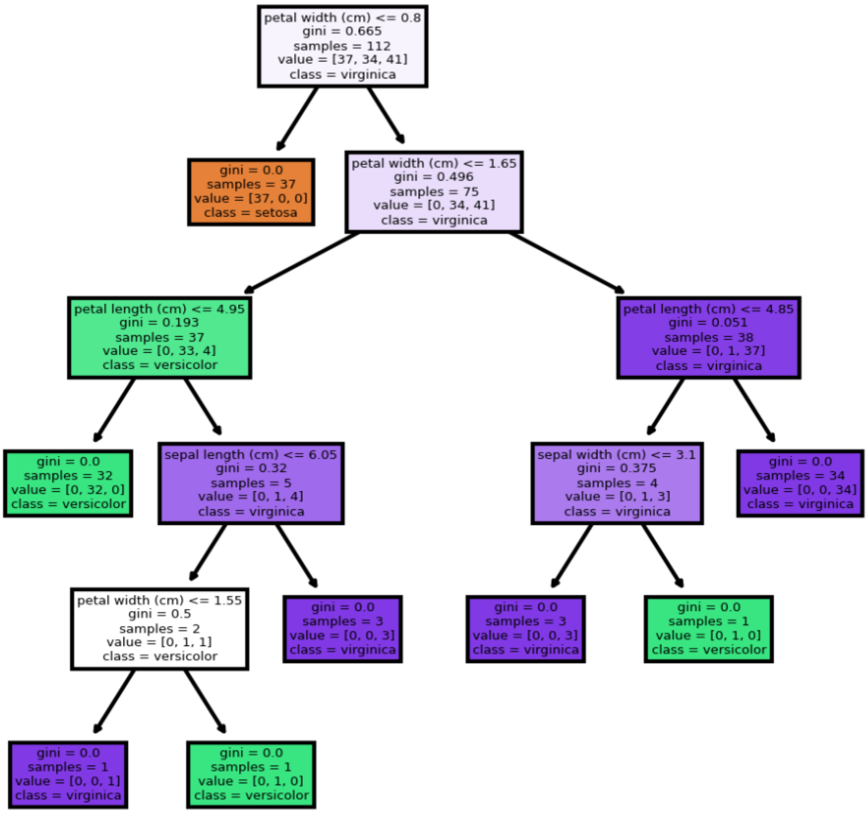
54	1
108	2
112	2
17	0
119	2
..	
133	2
137	2
72	1
140	2
37	0

Name: target, Length: 112, dtype: int32

14	0
98	1
75	1
16	0
131	2
56	1
141	2
44	0
29	0
120	2
94	1
5	0
102	2

```
51     1
78     1
42     0
92     1
66     1
31     0
35     0
90     1
84     1
77     1
40     0
125    2
99     1
33     0
19     0
73     1
146    2
91     1
135    2
69     1
128    2
114    2
48     0
53     1
28     0
Name: target, dtype: int32
14     0
98     1
75     1
16     0
131    2
56     1
141    2
44     0
29     0
120    2
94     1
5      0
102    2
51     1
78     1
42     0
92     1
66     1
31     0
35     0
```

```
90     1
84     1
77     1
40     0
125    2
99     1
33     0
19     0
73     1
146    2
91     1
135    2
69     1
128    2
114    2
48     0
53     1
28     0
Name: target, dtype: int32 [0 1 1 0 2 1 2 0 0 2 1 0 2 1 1 0 1 1 0 0 1 1 2 0 2 1 0 0 1 2 1 2 1 2 2
0 1
 0]
Accuracy:  0.9736842105263158
```


petal width (cm) <= 0.8
gini = 0.665
samples = 112
value = [37, 34, 41]
class = virginica
gini = 0.0
samples = 37
value = [37, 0, 0]
class = setosa
petal width (cm) <= 1.65
gini = 0.496
samples = 75
value = [0, 34, 41]
class = virginica
petal length (cm) <= 4.95
gini = 0.193
samples = 37
value = [0, 33, 4]
class = versicolor
petal length (cm) <= 4.85
gini = 0.051
samples = 38
value = [0, 1, 37]
class = virginica
gini = 0.0
samples = 32
value = [0, 32, 0]
class = versicolor
sepal length (cm) <= 6.05
gini = 0.32
samples = 5
value = [0, 1, 4]
class = virginica
sepal width (cm) <= 3.1
gini = 0.375
samples = 4
value = [0, 1, 3]
class = virginica
gini = 0.0
samples = 34
value = [0, 0, 34]
class = virginica
petal width (cm) <= 1.55
gini = 0.5
samples = 2
value = [0, 1, 1]
class = versicolor
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor

Practical – 5.2 : Write a program to calculate popular attribute selection measures (ASM) like Information Gain, Gain Ratio, and Gini Index etc. for decision tree.

Practical No: 6
Practical Name: Implement simple KNN using Euclidean distance in Python.
--

Code: KNN using Euclidean distance

```
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 :

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-
I_ML/KNN.py
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
75                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]]

Process finished with exit code 0
```
##

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
from sklearn.model_selection import train_test_split
#data_b = load_iris()
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)

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)
```

OUTPUT:

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-I_ML/KNN.py
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                  0.08216
546        10.32         16.35  ...          0.2681                  0.07399
293        11.85         17.46  ...          0.3101                  0.07007
350        11.66         17.07  ...          0.2731                  0.06825

[6 rows x 30 columns]
562    0
291    1
16     0
546    1
293    1
350    1
Name: target, dtype: int32
     mean radius  mean texture  ...  worst symmetry  worst fractal dimension
421        14.69         13.98  ...          0.2827                  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]]
Number of correct predictions= 134
Number of wrong predictions =  9

Process finished with exit code 0
```

Practical No: 7

Practical Name: Write a program to implement the 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 Iris 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
from sklearn.model_selection import train_test_split
data_b = load_iris()
#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)

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)
# corr = cm[0, 0] + cm[1, 1] + cm[2, 2]   # ----for iris
# corr = cm[0, 0] + cm[1, 1]  #----for breast cancer
corr = 0
for i in range(len(data_b.target_names)):
    corr = corr + cm[i, i]
wrg = len(y_test) - corr
print("Number of correct predictions=", corr)
print("Number of wrong predictions = ", wrg)
```

OUTPUT:

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-I_ML/KNN.py
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
75                 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]]
Number of correct predictions= 38
Number of wrong predictions =  0

Process finished with exit code 0
```

##

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
from sklearn.model_selection import train_test_split
#data_b = load_iris()
```

```
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)

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)
# corr = cm[0, 0] + cm[1, 1] + cm[2, 2]   # ----for iris
# corr = cm[0, 0] + cm[1, 1]  #----for breast cancer
corr = 0
for i in range(len(data_b.target_names)):
    corr = corr + cm[i, i]
wrg = len(y_test) - corr
print("Number of correct predictions=", corr)
print("Number of wrong predictions = ", wrg)
```

OUTPUT:

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-
I_ML/KNN.py
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                  0.08216
546        10.32         16.35  ...          0.2681                  0.07399
293        11.85         17.46  ...          0.3101                  0.07007
350        11.66         17.07  ...          0.2731                  0.06825

[6 rows x 30 columns]
562    0
```

```
291    1
16     0
546    1
293    1
350    1
Name: target, dtype: int32
     mean radius  mean texture  ...  worst symmetry  worst fractal dimension
421        14.69         13.98  ...          0.2827                  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]]
Number of correct predictions= 134
Number of wrong predictions =  9

Process finished with exit code 0
```

Practical No.: 8
Practical Name: Write a Program for Confusion Matrix and calculate Precision, Recall, F-Measure

---

```
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 d3ataset
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)
```

OUTPUT:

```
Confusion Matrix (Irish Dataset):
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Precision (Irish Dataset): 1.0
Recall (Irish Dataset): 1.0
F-measure (Irish Dataset): 1.0

Confusion Matrix (Breast Cancer Dataset):
[[38  5]
 [ 0 71]]
Precision (Breast Cancer Dataset): 0.9342105263157895
Recall (Irish Dataset): 1.0
F-measure (Irish Dataset): 0.9659863945578232
```

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
```

Practical – 10: Write a program to implement the naïve Bayesian classifier for a sample training dataset stored as a . CSV file. Compute the accuracy of the classifier, considering a few test data sets.

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix

from sklearn import datasets
iris = datasets.load_iris()  # loading dataset
x = iris.data[:, ]  # input
y = iris.target  # target
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:- ", cm)
```

OUTPUT:

```
C:\Users\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/PycharmProjects/MCA-I_ML/Naive_bays_short.py
Features :  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Confusion Matrix:-  [[13  0  0]
 [ 0 16  0]
 [ 0  0  9]]

Process finished with exit code 0
```

Practical – 11.1: Implementing Agglomerative Clustering in Python.

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate sample data
X, y = make_blobs(n_samples=200, centers=4, random_state=0)

# Create an instance of AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=4)

# Perform clustering
clustering.fit(X)

# Retrieve the cluster labels
labels = clustering.labels_

# Plot the data points with their corresponding cluster labels
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Agglomerative Clustering")
plt.show()
```

output:

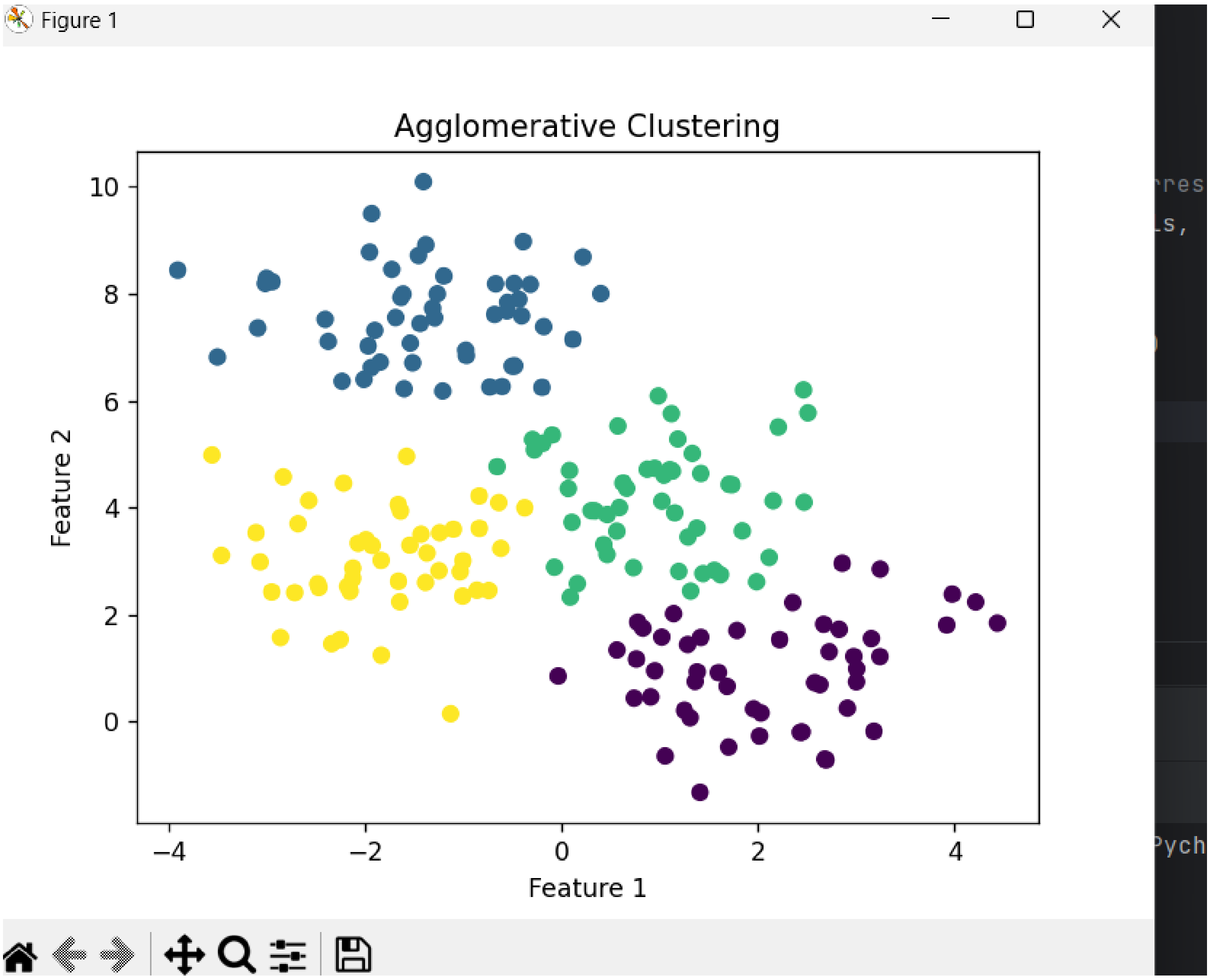


Figure 1
Agglomerative Clustering
Feature 2
10
8
6
4
2
0
−4
−2
0
2
4
Feature 1

Practical – 11.2: 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

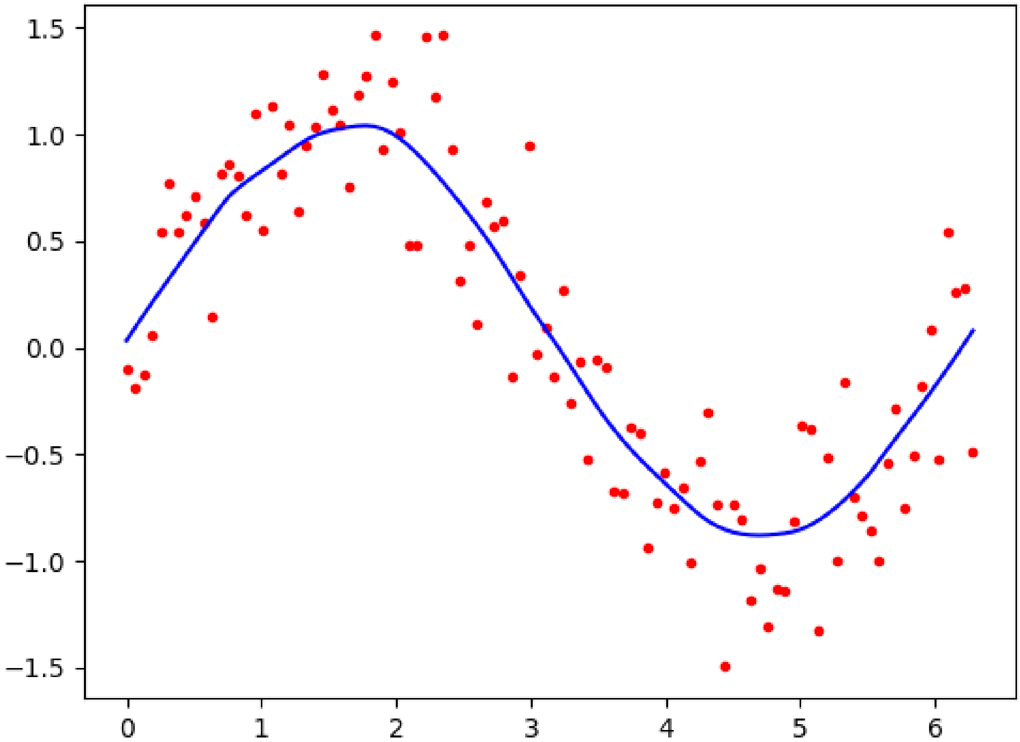
0 1 2 1 2 2 2 2 0 2 0

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]

Practical – 12: 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:



1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5
6

Practical - 13.1: 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  0.29428266]
 [ 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 -0.32163589]]
Input =  [[2]
 [3]]
Hidden =  [[0.37915865]
 [0.31171721]
 [0.36284346]
 [0.58104275]]
Output =  [[0.52124119]
 [0.46381691]]
Y =  [[0.52124119]
 [0.46381691]]
```

Practical No - 13.2: Classification Of Iris Dataset By Applying Artificial Neural Network With Back-Propagation Algorithm

```
# Classification of iris data set by applying artificial neural network using Back-propagation 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\sejal\MCA-I_ML\Scripts\python.exe C:/Users/sejal/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  0.35763949]
 [-0.77270015 -0.00419192]
 [ 0.31066799 -0.36004278]
 [ 0.13275579  0.05427426]]
weight 2 [[ 0.00214572 -0.08730011  0.21651309]
 [ 0.60151869 -0.48253284  0.51413704]]
```

--XXX--