LAB - 1 Implement A* Search algorithm

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print("Path does not exist!")
            return None
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print("Path found: {}".format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print("Path does not exist!")
    return None
```

```
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        "A": 11,
        "B": 6,
        "C": 99,
        "D": 1,
        "E": 7,
        "G": 0,
    }

    return H_dist[n]

Graph_nodes = {
    "A": [("B", 2), ("E", 3)],
    "B": [("C", 1), ("G", 9)],
    "C": None,
    "E": [("D", 6)],
    "D": [("G", 1)],
}
aStarAlgo("A", "G")
```

Output

```
Path found: ['A', 'E', 'D', 'G']
['A', 'E', 'D', 'G']
```

LAB - 3 For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import csv

file = open("lab3ds.csv")
data = list(csv.reader(file))[1:]
concepts = []
target = []
```

```
for i in data:
    concepts.append(i[:-1])
    target.append(i[-1])

specific_h = ["0"] * len(concepts[0])
general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]

for i, instance in enumerate(concepts):
    if target[i] == "Yes":
        for x in range(len(specific_h)):
            if specific_h[x] == "0":
                specific_h[x] = instance[x]
            elif instance[x] != specific_h[x]:
                specific_h[x] = "?"
                general_h[x][x] = "?"
    if target[i] == "No":
        for x in range(len(specific_h)):
            if instance[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(["?", "?", "?", "?", "?", "?"])

print("Final Specific:", specific_h, sep="\n")
print("Final General:", general_h, sep="\n")
```

Output

```
Final Specific:
['Sunny', 'Warm', '?', 'Strong', '?', '?']
Final General:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Dataset

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Cloudy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

LAB - 4 Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value] / len(df[Class])
        entropy += -fraction * np.log2(fraction)
    return entropy


def find_entropy_attribute(df, attribute):
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = len(
                df[attribute][df[attribute] == variable][df[Class] ==
target_variable]
            )
            den = len(df[attribute][df[attribute] == variable])
            fraction = num / (den + eps)
            entropy += -fraction * log(fraction + eps)
        fraction2 = den / len(df)
        entropy2 += -fraction2 * entropy
    return abs(entropy2)


def find_winner(df):
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df) - find_entropy_attribute(df, key))
    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node, value):
    return df[df[node] == value].reset_index(drop=True)

def buildTree(df, tree=None):
    node = find_winner(df)
    attValue = np.unique(df[node])
    if tree is None:
        tree = {}
        tree[node] = {}
    for value in attValue:
        subtable = get_subtable(df, node, value)
        clValue, counts = np.unique(subtable["play"], return_counts=True)
```

```
        if len(counts) == 1:
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)
    return tree

import pandas as pd
import numpy as np

eps = np.finfo(float).eps
from numpy import log2 as log

df = pd.read_csv("tennis.csv")
print("\n Given Play Tennis Data Set:\n\n", df)
tree = buildTree(df)
import pprint

pprint.pprint(tree)

test = {"Outlook": "Sunny", "Temperature": "Hot", "Humidity": "High",
"Wind": "Weak"}

def func(test, tree, default=None):
    attribute = next(iter(tree))
    print(attribute)
    if test[attribute] in tree[attribute].keys():
        print(tree[attribute].keys())
        print(test[attribute])
        result = tree[attribute][test[attribute]]
        if isinstance(result, dict):
            return func(test, result)
        else:
            return result
    else:
        return default

ans = func(test, tree)
print(ans)
```

Dataset

Outlook	Temperature	Humidity	Wind	play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Output

Given Play Tennis Data Set:

```
      Outlook Temperature Humidity    Wind play
0       Sunny         Hot     High    Weak   No
1       Sunny         Hot     High  Strong   No
2    Overcast         Hot     High    Weak  Yes
3        Rain        Mild     High    Weak  Yes
4        Rain        Cool   Normal    Weak  Yes
5        Rain        Cool   Normal  Strong   No
6    Overcast        Cool   Normal  Strong  Yes
7       Sunny        Mild     High    Weak   No
8       Sunny        Cool   Normal    Weak  Yes
9        Rain        Mild   Normal    Weak  Yes
10      Sunny        Mild   Normal  Strong  Yes
11   Overcast        Mild     High  Strong  Yes
12   Overcast         Hot   Normal    Weak  Yes
13       Rain        Mild     High  Strong   No
```

```
{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

```
Outlook
dict_keys(['Overcast', 'Rain', 'Sunny'])
Sunny
Humidity
dict_keys(['High', 'Normal'])
High
No
```

LAB - 5 Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
import numpy as np

x = np.array(([2,9],[1,5],[3,6]),dtype=float)
y = np.array(([92],[86],[89]),dtype=float)

x = x/np.amax(x, axis=0)
y = y/100

def sigmoid(x):
   return 1/(1+np.exp(-x))

def derivatives_sigmoid(x):
```

```
    return x*(1-x)

epoch = 5
lr = 0.1

inputlayer_neurons = 2
hiddenlayer_neurons = 3
outputlayer_neurons = 1

wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons, outputlayer_neurons))
bout = np.random.uniform(size=(1, outputlayer_neurons))

for i in range(epoch):
    hinp1 = np.dot(x, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)

    EO = y - output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) * lr
    wh += x.T.dot(d_hiddenlayer) * lr

    print("--Epoch-",i+1,"--Starts--")
    print("Input :\n"+str(x))
    print("Actual Output : \n"+str(y))
    print("Predicted Output : \n", output)
    print("--Epoch-",i+1,"--Ends--")

print("Input :\n"+str(x))
print("Actual Output : \n"+str(y))
print("Predicted Output : \n", output)
```

Output

```
--Epoch- 1 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81504223]
 [0.8014937 ]
 [0.81597075]]
--Epoch- 1 --Ends--
--Epoch- 2 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81604173]
 [0.80245966]
 [0.8169656 ]]
--Epoch- 2 --Ends--
--Epoch- 3 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
```

```
 [[0.81702096]
 [0.80340646]
 [0.81794026]]
--Epoch- 3 --Ends--
--Epoch- 4 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81798054]
 [0.80433467]
 [0.81889534]]
--Epoch- 4 --Ends--
--Epoch- 5 --Starts--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
Predicted Output :
 [[0.81892105]
 [0.80524483]
 [0.81983142]]
--Epoch- 5 --Ends--
Input :
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output :
[[0.92]
 [0.86]
 [0.89]]
```

```
Predicted Output :
 [[0.81892105]
 [0.80524483]
 [0.81983142]]
```

LAB - 6 Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

data = pd.read_csv('tennis.csv')
print("The first 5 Values of data is :\n", data.head())

X = data.iloc[:, :-1]
print("\nThe First 5 values of the train attributes is\n", X.head())

Y = data.iloc[:, -1]
print("\nThe First 5 values of target values is\n", Y.head())

obj1= LabelEncoder()
X.Outlook = obj1.fit_transform(X.Outlook)
print("\n The Encoded and Transformed Data in Outlook \n",X.Outlook)

obj2 = LabelEncoder()
X.Temperature = obj2.fit_transform(X.Temperature)

obj3 = LabelEncoder()
X.Humidity = obj3.fit_transform(X.Humidity)

obj4 = LabelEncoder()
X.Wind = obj4.fit_transform(X.Wind)
print("\n The Encoded and Transformed Training Examples \n", X.head())

obj5 = LabelEncoder()
```

```
Y = obj5.fit_transform(Y)
print("The class Label encoded in numerical form is",Y)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20)

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is: ", accuracy_score(classifier.predict(X_test), Y_test))
```

Output

```
The first 5 Values of data is :
     Outlook Temperature Humidity    Wind Play
0      Sunny         Hot     High    Weak   No
1      Sunny         Hot     High  Strong   No
2   Overcast         Hot     High    Weak  Yes
3       Rain        Mild     High    Weak  Yes
4       Rain        Cool   Normal    Weak  Yes

The First 5 values of the train attributes is
     Outlook Temperature Humidity    Wind
0      Sunny         Hot     High    Weak
1      Sunny         Hot     High  Strong
2   Overcast         Hot     High    Weak
3       Rain        Mild     High    Weak
4       Rain        Cool   Normal    Weak

The First 5 values of target values is
0     No
1     No
2    Yes
3    Yes
4    Yes
Name: Play, dtype: object

The Encoded and Transformed Data in Outlook
0     2
1     2
```

```
2     0
3     1
4     1
5     1
6     0
7     2
8     2
9     1
10    2
11    0
12    0
13    1
Name: Outlook, dtype: int64

The Encoded and Transformed Training Examples
    Outlook  Temperature  Humidity  Wind
0         2            1         0     1
1         2            1         0     0
2         0            1         0     1
3         1            2         0     1
4         1            0         1     1

The class Label encoded in numerical form is [0 0 1 1 1 0 1 0 1 1 1 1 1 0]

Accuracy is: 1.0
```

Dataset

Outlook	Temperature	Humidity	Wind	play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

LAB - 7 Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = load_iris()
print("\n IRIS Dataset:\n", dataset.data)
print("\n IRIS Features:\n", dataset.feature_names)
print("\n IRIS Target:\n", dataset.target)
print("\n IRIS Target:\n", dataset.target_names)
```

```
X = pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y=pd.DataFrame(dataset.target)
y.columns=['Targets']

print(y)

plt.figure(figsize=(8,5))
colormap=np.array(['red','lime','blue'])

plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=20)
plt.title('Before Clustering')

plt.subplot(1,3,2)
model = KMeans(n_clusters=3)
model.fit(X)
predY = np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=20)
plt.title('KMeans')

scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)

y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=20)
plt.title('GMM Clustering')
```

Output

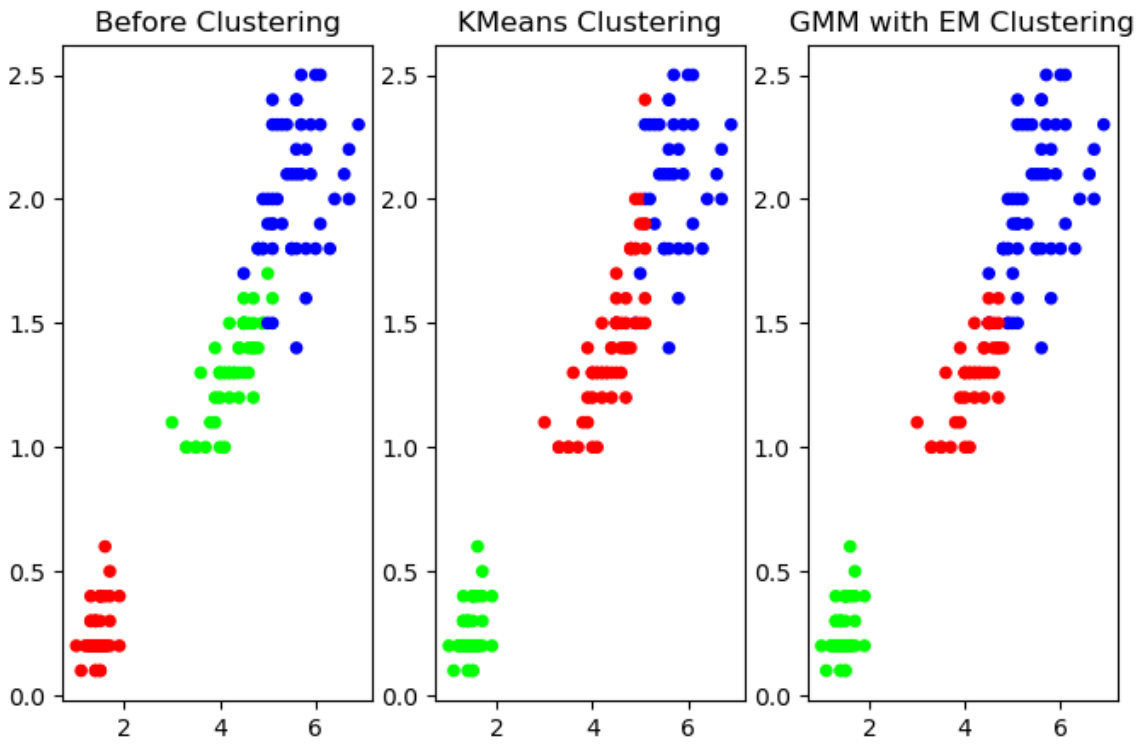
```
IRIS Dataset:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
```

```
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5.  3.4 1.5 0.2]
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
```

```
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
```

```
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]

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

 IRIS Target:
 [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]
```

```
 IRIS Target:
 ['setosa' 'versicolor' 'virginica']
     Targets
0           0
1           0
2           0
3           0
4           0
..        ...
145         2
146         2
147         2
148         2
149         2
```



LAB - 8 Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

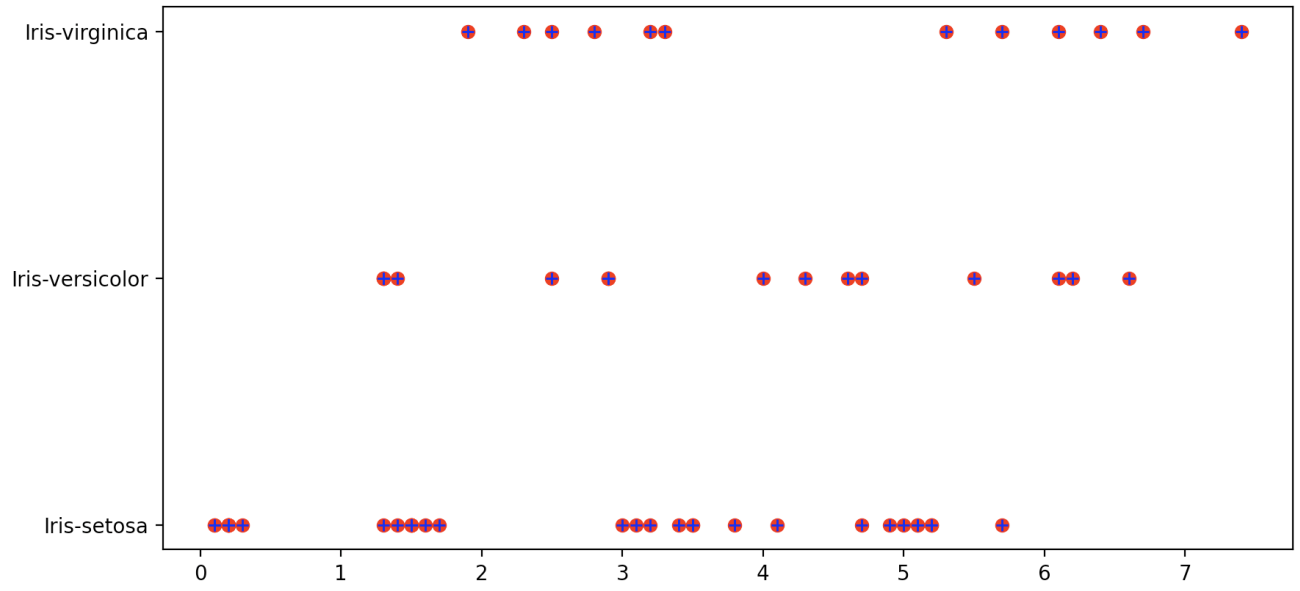
```
from sklearn import metrics
import matplotlib.pyplot as plt

assigned_names = ['sepal-length', 'sepal-width', 'petal-length',
'petal-width', 'Class']

dataset = pd.read_csv("iris2.csv", names=assigned_names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print
("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label',
'Correct/Wrong'))
print
("-------------------------------------------------------------------------")
for label in ytest:
   print ('%-25s %-25s' % (label, ypred[i]), end="")
   if (label == ypred[i]):
       print (' %-25s' % ('Correct'))
   else:
       print (' %-25s' % ('Wrong'))
   i = i + 1
print
("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print
("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest,
ypred))
print
("-------------------------------------------------------------------------")
```

```
print('Accuracy of the classifer is %0.2f' %
metrics.accuracy_score(ytest,ypred))
print
("-----------------------------------------------------------------------")
plt.plot(Xtest,ytest,'ro')
plt.plot(Xtest,ytest,'b+')
```

Output

```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2

-----------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
-----------------------------------------------------------------------
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-virginica            Iris-virginica            Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-virginica            Iris-virginica            Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-virginica            Iris-virginica            Correct
-----------------------------------------------------------------------

Confusion Matrix:
 [[5 0 0]
 [0 4 0]
 [0 0 6]]
-----------------------------------------------------------------------

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         5
Iris-versicolor       1.00      1.00      1.00         4
 Iris-virginica       1.00      1.00      1.00         6
```

```
       accuracy                           1.00        15
      macro avg       1.00      1.00      1.00        15
   weighted avg       1.00      1.00      1.00        15

---------------------------------------------------------------------
Accuracy of the classifer is 1.00
---------------------------------------------------------------------
```



LAB 9 - Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point, xmat, ymat, k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat, ymat, k):
```

```
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='yellow')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'black', linewidth=2)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output