In [1]:

```
def astar(startnode,stopnode):
    open_set=set(startnode)
    closed_set=set()
    g={}
    parents={}
    parents[startnode]= startnode
    g[startnode]=0
    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==stopnode 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==stopnode:
            path=[]
            while(parents[n]!=n):
                path.append(n)
                n= parents[n]
            path.append(startnode)
            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(n):
    if n in Graph_nodes:
        return Graph_nodes[n]
    else:
        return None

def heuristic(n):
    H_dist = {
        'S': 5,
        'A': 3,
        'B': 4,
        'C': 2,
        'D': 6,
        'G': 0,
    }

    return H_dist[n]
```

```python
Graph_nodes = {
    'S': [('A', 1), ('G', 10)],
    'A': [('B', 2), ('C', 1)],
    'B': [('D', 5)],
    'C': [('D', 3), ('G', 4)],
    'D': [('G', 2)],
}

astar('S', 'G')
```

```
Path found ['S', 'A', 'C', 'G']
```

Out[1]:

```
['S', 'A', 'C', 'G']
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [1]:

```
class Graph:
    def __init__(self, g, heuristic, startnode):
        self.graph = g
        self.H=heuristic
        self.start = startnode
        self.solution={}
        self.parents={}
        self.status={}

    def applyAO(self):
        self.aostar(self.start, False)

    def getHeuristic(self,n):
        return self.H.get(n,0)
    def getStatus(self,n):
        return self.status.get(n,0)
    def getNeighbor(self,n):
        return self.graph.get(n,'')
    def setHeuristic(self,n,v):
        self.H[n]=v
    def setStatus(self,n,v):
        self.status[n]=v

    def mincost(self,v):
        minimumcost=0
        costtochildnodelist={}
        costtochildnodelist[minimumcost]=[]
        flag=True
        for nodeval in self.getNeighbor(v):
            cost=0
            nodelist=[]
            for c, weight in nodeval:
                cost=cost + self.getHeuristic(c) + weight
                nodelist.append(c)
            if flag==True:
                minimumcost=cost
                costtochildnodelist[minimumcost]=nodelist
                flag=False
            else:
                if minimumcost>cost:
                    minimumcost=cost
                    costtochildnodelist[minimumcost]=nodelist
        return minimumcost, costtochildnodelist[minimumcost]




    def printsol(self):
        print("The start node is" , self.start)
        print("------------------------------------------")
        print("The solution graph is")
        print(self.solution)

    def aostar(self, v, backtracking):
        print("HEURISTIC VALUES" ,self.H)
        print("Solution Graph", self.solution)
        print("Processing Node" ,v)
        print("------------------------------------------------------------------------

        if self.getStatus(v)>=0:
            minimumCost, childnodelist = self.mincost(v)
            self.setHeuristic(v,minimumCost)
            self.setStatus(v, len(childnodelist))
```

```python
            solved=True
            for childnode in childnodelist:
                self.parents[childnode]=v
                if self.getStatus(childnode)!=-1:
                    solved =solved & False
            if solved==True:
                self.setStatus(v,-1)
                self.solution[v]=childnodelist




            if v!=self.start:

                self.aostar(self.parents[v], True)


            if backtracking==False:
                for childnode in childnodelist:
                    self.setStatus(childnode,0)
                    self.aostar(childnode, False)


h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, '

graph = {
    'A' : [[('B',1), ('C',1)], [('D',1)]],
    'B' : [[('G',1)], [('H',1)]],
    'C' : [[('J',1)]],
    'D' : [[('E',1)],[('F',1)]],
    'G' : [[('I',1)]]
}


g1= Graph(graph,h1,'A')
g1.applyAO()
g1.printsol()
```

```
HEURISTIC VALUES {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': '3'}
Solution Graph {}
Processing Node A
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': '3'}
Solution Graph {}
Processing Node B
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': '3'}
Solution Graph {}
Processing Node A
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': '3'}
Solution Graph {}
Processing Node G
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': '3'}
Solution Graph {}
Processing Node B
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': '3'}
Solution Graph {}
Processing Node A
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': '3'}
Solution Graph {}
Processing Node I
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1, 'T': '3'}
Solution Graph {'I': []}
Processing Node G
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': '3'}
Solution Graph {'I': [], 'G': ['I']}
Processing Node B
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': '3'}
Solution Graph {'I': [], 'G': ['I'], 'B': ['G']}
Processing Node A
-----------------------------------------------------------------------------------------
-
HEURISTIC VALUES {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': '3'}
Solution Graph {'I': [], 'G': ['I'], 'B': ['G']}
Processing Node C
-----------------------------------------------------------------------------------------
-
```

```
HEURISTIC VALUES {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': '3'}
Solution Graph {'I': [], 'G': ['I'], 'B': ['G']}
Processing Node A
-------------------------------------------------------------------------------------------
HEURISTIC VALUES {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': '3'}
Solution Graph {'I': [], 'G': ['I'], 'B': ['G']}
Processing Node J
-------------------------------------------------------------------------------------------
HEURISTIC VALUES {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': '3'}
Solution Graph {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
Processing Node C
-------------------------------------------------------------------------------------------
HEURISTIC VALUES {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': '3'}
Solution Graph {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
Processing Node A
-------------------------------------------------------------------------------------------
The start node is A
-----------------------------------------
The solution graph is
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [23]:

```
import csv
a=[]
with open('../labprog/c2.csv') as csvfile:
    fdata= csv.reader(csvfile)
    for row in fdata:
        a.append(row)
        print(row)
```

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'Y']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'Y']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'N']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'Y']
```

In [24]:

```
num_att = len(a[0])-1
S=['0']*num_att
G=['?'] *num_att
print(S)
print(G)
temp=[]
```

```
['0', '0', '0', '0', '0', '0']
['?', '?', '?', '?', '?', '?']
```

In [25]:

```
for i in range(0,num_att):
    S[i]=a[0][i]
print("---------------------------------")
```

```
---------------------------------
```

In [27]:

```
for i in range(0, len(a)):
    if a[i][num_att]=='Y':
        for j in range(0,num_att):
            if S[j]!=a[i][j]:
                S[j]='?'
        for j in range(0,num_att):
            for k in range(0, len(temp)):
                if temp[k][j]!=S[j] and temp[k][j]!='?':
                    del temp[k]
    if a[i][num_att]=='N':
        for j in range(0,num_att):
            if S[j]!=a[i][j] and S[j]!='?':
                G[j]= S[j]
                temp.append(G)
                G =['?']*num_att
```

In [29]:

```
print(S)
if len(temp)==0:
    print(G)
else:
    print(temp)
```

```
['sunny', 'warm', '?', 'strong', '?', '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['sunny', '?',
'?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

In [ ]:

In [ ]:

```
```

In [28]:

```
from collections import Counter
from pprint import pprint
import pandas as pd
import math
```

In [36]:

```
data = pd.read_csv('tennis.csv')
attr = list(data.columns[1:])
attr.remove('PlayTennis')
```

In [41]:

```
def entropy(x):
    c = Counter(i for i in x)
    n = len(x) * 1.0
    prob= [i/n for i in c.values()]
    return sum(-i*math.log(i,2) for i in prob)
```

In [45]:

```
def information_gain(df,split,target):
    dfs= df.groupby(split)
    n= len(df)*1.0
    dfa = dfs.agg({target: [entropy, lambda x: len(x)/n]})[target]
    dfa.columns = ['entropy', 'prob']
    new = sum(dfa['entropy'] * dfa['prob'])
    old = entropy(df[target])
    return old-new
```

In [50]:

```
def id3(df,attr,target, default=None):
    c = Counter(i for i in df[target])
    if len(c)==1:
        return next(iter(c))
    elif df.empty or (not attr):
        return default
    else:
        default = max(c.keys())
        gain = [information_gain(df,att,target) for att in attr]
        best = attr[gain.index(max(gain))]
        tree= {best:{}}
        remain = [ i for i in attr if i!=best]
        for at, dfs in df.groupby(best):
            subtree= id3(dfs,remain,target,default)
            tree[best][at]= subtree
    return tree
```

In [51]:

```
tree = id3(data,attr,'PlayTennis')
pprint(tree)
```

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

In [ ]:

In [ ]:

In [3]:

```
import numpy as np
X=np.array(([1,2],[3,6], [2,5]),dtype=float)
Y =np.array(([92],[86], [89]),dtype=float)
X= X/np.amax(X,axis=0)
Y= Y/100
```

In [8]:

```
epooch = 500
lr =0.1
inputlay=2
hidden=3
output=1


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

In [9]:

```
wh = np.random.uniform(size=(inputlay,hidden))
bh = np.random.uniform(size=(1,hidden))
wout = np.random.uniform(size=(hidden, output))
bout = np.random.uniform(size=(1, output))
```

In [13]:

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

In [15]:

```
EO = Y - output
outgrad = derivative_sigmoid(output)
d_output = EO * outgrad
EH = d_output.dot(wout.T)
hgrad=derivative_sigmoid(hlayer_act)
h_output = EH * hgrad
```

In [17]:

```
wout+= hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(h_output) *lr
```

In [18]:

```
print("Expected outcome", str(Y))
print("Output", output)
print("Input", str(X))
```

```
Expected outcome [[0.92]
 [0.86]
 [0.89]]
Output [[1.05127091]
 [1.07260753]
 [1.06511096]]
Input [[0.33333333 0.33333333]
 [1.         1.        ]
 [0.66666667 0.83333333]]
```

In [ ]:

In [5]:

```
import math
import csv
def safe_div(x,y):
    if y==0:
        return 0
    return x/y
def mean(numbers):
    return(safe_div(sum(numbers) , float(len(numbers))))
def stdev(numbers):
    avg=mean(numbers)
    variance = safe_div(sum([pow(x-avg, 2) for x in numbers  ]), float(len(numbers)-1)
    return math.sqrt(variance)
```

In [4]:

```
def calculateprob(x, mean,stdev):
    expo= math.exp(  -safe_div( math.pow(x-mean,2), (2 * math.pow(stdev,2))    ))
    final = safe_div(expo, math.sqrt(2*math.pi)* stdev)
    return final
```

In [5]:

```
testset=dataset =[list(map(float,i)) for i in [x for x in list(csv.reader(open("ConceptI
trainset = [testset.pop(0) for x in range(int(len(dataset)*0.9))]
print(*testset ,sep="\n")
print(*trainset ,sep="\n")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [5], in <module>
----> 1 testset=dataset =[list(map(float,i)) for i in [x for x in list(csv.reader(open("
ConceptLearning.csv")))]]
      2 trainset = [testset.pop(0) for x in range(int(len(dataset)*0.9))]
      3 print(*testset ,sep="\n")

NameError: name 'csv' is not defined
```

In [6]:

```
separated= {}
for i in trainset:
    if i[-1] not in separated:
        separated[i[-1]]=[]
    separated[i[-1]].append(i)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [6], in <module>
      1 separated= {}
----> 2 for i in trainset:
      3     if i[-1] not in separated:
      4         separated[i[-1]]=[]

NameError: name 'trainset' is not defined
```

In [7]:

```
summaries = {}
for classvalue, instances in separated.items():
    summaries[classvalue] = [(mean(att), stdev(att)) for att in zip(*instances)]
    [i.pop() for i in summaries.values()]
    print("Summaries attributes by class")
    for i, j in summaries.items():
        print(i, ":", j)
```

In [8]:

```
prediction = []
for i in range(len(testset)):
    probab = {}
    for clv, classum in summaries.items():
        probab[clv]=1
    for j in range(len(classum)):
        mean, stdev = classum[j]
        x = testset[i][j]
        probab[clv]*= calculateprob(x, mean,stdev)
    bestLabel, bestProb = None, -1
    for cv, prob in probab.items():
        if bestLabel is None or prob>bestProb:
            bestProb = prob
            bestLabel = cv
    prediction.append(bestLabel)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [8], in <module>
      1 prediction = []
----> 2 for i in range(len(testset)):
      3     probab = {}
      4     for clv, classum in summaries.items():

NameError: name 'testset' is not defined
```

In [9]:

```
actual = [i[-1] for i in testset]
count = 0
for (i,j) in zip(actual,prediction):
    if i==j:
        count+=1
accuracy =  safe_div(count, float(len(actual))) *100
print("ACTUAL", actual)
print("PREDICTION", prediction)
print("Accuracy", accuracy)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [9], in <module>
----> 1 actual = [i[-1] for i in testset]
      2 count = 0
      3 for (i,j) in zip(actual,prediction):

NameError: name 'testset' is not defined
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [59]:

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

In [60]:

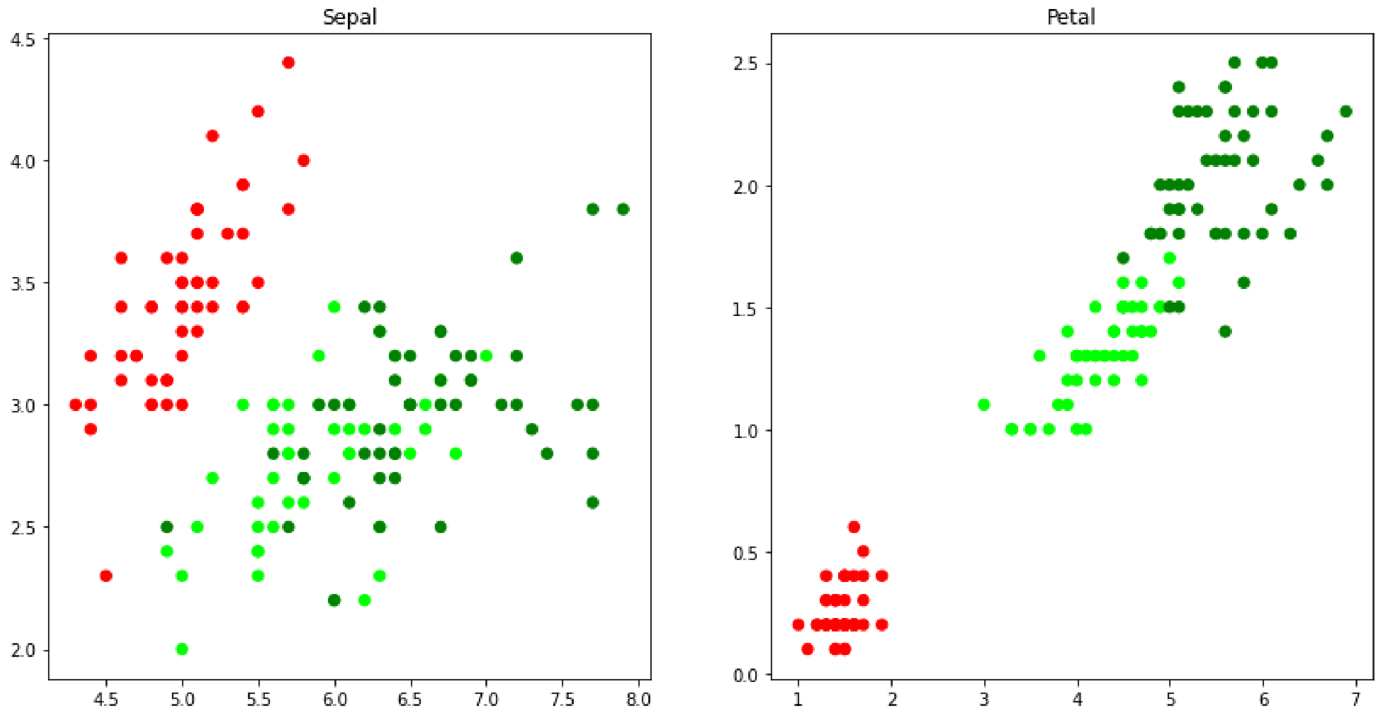
```
iris= load_iris()
X= pd.DataFrame(iris.data)
Y= pd.DataFrame(iris.target)

l1 = [0,1,2]
def rename(s):
    l2=[]
    for i in s:
        if i not in l2:
            l2.append(i)
    for i in range(len(s)):
        pos= l2.index(s[i])
        s[i]=l1[pos]
    return s
```

In [61]:

```
X.columns= ['Sep_L', 'Sep_W', 'Pet_L', 'Pet_W']
Y.columns = ['Targets']
```

In [62]:

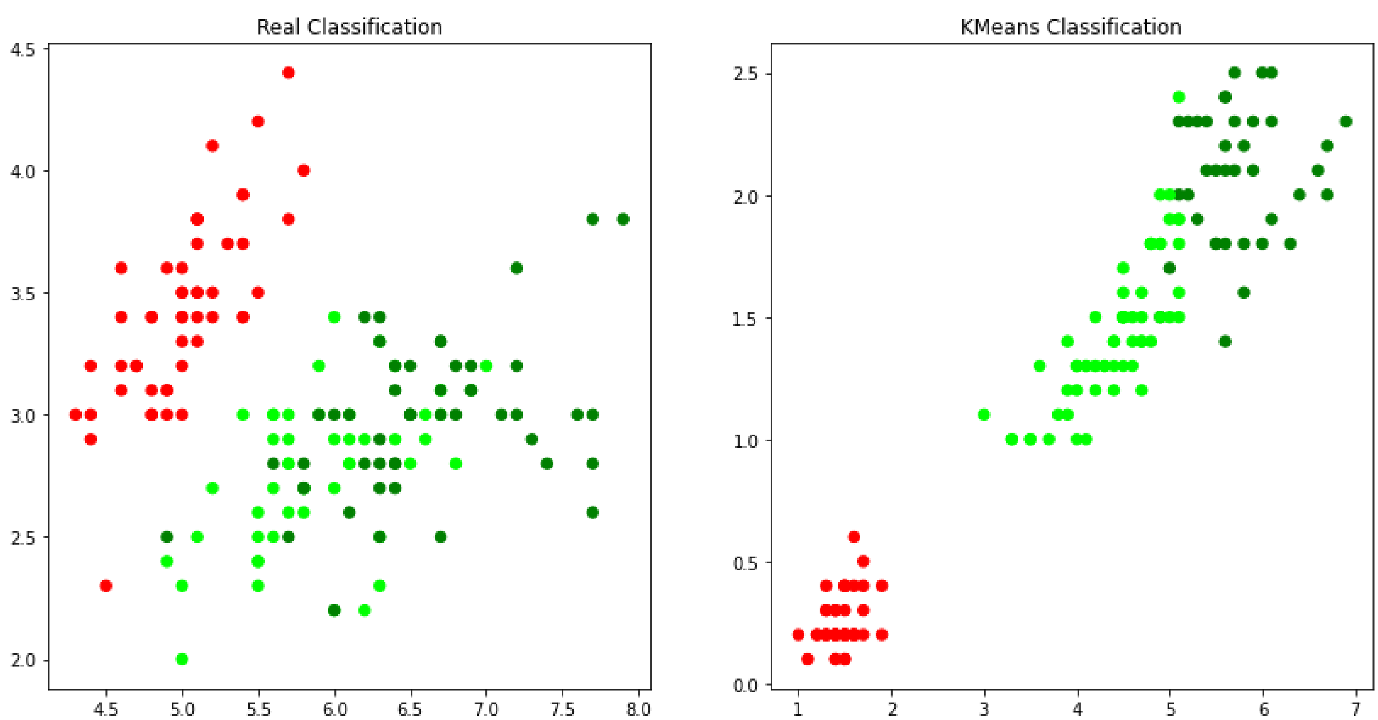
```
plt.figure(figsize=(14,7))
color = np.array(['red', 'lime', 'green'])

plt.subplot(1,2,1)
plt.scatter(X.Sep_L, X.Sep_W, c=color[Y.Targets], s=40)
plt.title("Sepal")

plt.subplot(1,2,2)
plt.scatter(X.Pet_L, X.Pet_W, c=color[Y.Targets], s=40)
plt.title("Petal")
plt.show()
```



In [63]:
```
model = KMeans(n_clusters= 3)
model.fit(X)
```

Out[63]:
```
KMeans(n_clusters=3)
```

In [64]:
```
plt.figure(figsize=(14,7))
color = np.array(['red', 'lime', 'green'])

plt.subplot(1,2,1)
plt.scatter(X.Sep_L, X.Sep_W, c=color[Y.Targets], s=40)
plt.title("Real Classification")

plt.subplot(1,2,2)
plt.scatter(X.Pet_L, X.Pet_W, c=color[model.labels_], s=40)
plt.title("KMeans Classification")
plt.show()
```



In [65]:

```
km = rename(model.labels_)
print("What KMEANS THOUGHT", km)
print("What is accuracy score", sm.accuracy_score(Y,km))
print("What is confusion matrix \n", sm.confusion_matrix(Y,km))
```

```
What KMEANS THOUGHT [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 0 0 0 1 1 2 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
What is accuracy score 0.8933333333333333
What is confusion matrix
 [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
```

In [66]:

```
scaler = preprocessing.StandardScaler()
```

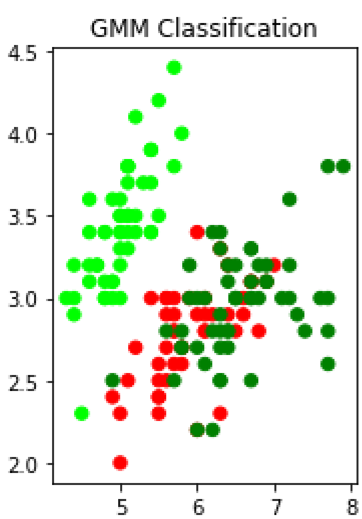
In [67]:

```
scaler.fit(X)
xsa= scaler.transform(X)
xs= pd.DataFrame(xsa, columns= X.columns)
print("xs is", xs.sample(5))
```

```
xs is          Sep_L     Sep_W     Pet_L     Pet_W
142 -0.052506 -0.822570  0.762758  0.922303
46  -0.900681  1.709595 -1.226552 -1.315444
130  1.886180 -0.592373  1.331133  0.922303
122  2.249683 -0.592373  1.672157  1.053935
20  -0.537178  0.788808 -1.169714 -1.315444
```

In [74]:

```
gmm = GaussianMixture(n_components=3)


gmm.fit(xs)
y_cluster_gmm =  gmm.predict(xs)

plt.subplot(1,2,1)
plt.scatter(X.Sep_L, X.Sep_W, c=color[y_cluster_gmm], s=40)
plt.title("GMM Classification")
plt.show()
```



In [75]:

```
em = rename(y_cluster_gmm)
```

```python
print("What GMM THOUGHT", km)
print("What is accuracy score", sm.accuracy_score(Y,em))
print("What is confusion matrix \n", sm.confusion_matrix(Y,em))
```

```
What GMM THOUGHT [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 0 1 1 2 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
What is accuracy score 0.9666666666666667
What is confusion matrix
 [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```

In [ ]:

In [ ]:

In [6]:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
```

In [9]:

```
iris_data = load_iris()
```

In [10]:

```
X_train, X_test, y_train, y_test= train_test_split(iris_data['data'], iris_data['target
```

In [11]:

```
print("IRIS DATASET", iris_data['data'])
print("X Test", X_test)
print("X_train", X_train)
```

```
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]]
X Test [[5.8 2.8 5.1 2.4]
 [6.  2.2 4.  1. ]
 [5.5 4.2 1.4 0.2]
 [7.3 2.9 6.3 1.8]
 [5.  3.4 1.5 0.2]
 [6.3 3.3 6.  2.5]
 [5.  3.5 1.3 0.3]
 [6.7 3.1 4.7 1.5]
 [6.8 2.8 4.8 1.4]
 [6.1 2.8 4.  1.3]
 [6.1 2.6 5.6 1.4]
 [6.4 3.2 4.5 1.5]
 [6.1 2.8 4.7 1.2]
 [6.5 2.8 4.6 1.5]
 [6.1 2.9 4.7 1.4]
 [4.9 3.6 1.4 0.1]
 [6.  2.9 4.5 1.5]
 [5.5 2.6 4.4 1.2]
 [4.8 3.  1.4 0.3]
 [5.4 3.9 1.3 0.4]
 [5.6 2.8 4.9 2. ]
 [5.6 3.  4.5 1.5]
 [4.8 3.4 1.9 0.2]
 [4.4 2.9 1.4 0.2]
 [6.2 2.8 4.8 1.8]
 [4.6 3.6 1.  0.2]
 [5.1 3.8 1.9 0.4]
 [6.2 2.9 4.3 1.3]
 [5.  2.3 3.3 1. ]
 [5.  3.4 1.6 0.4]
 [6.4 3.1 5.5 1.8]
 [5.4 3.  4.5 1.5]
 [5.2 3.5 1.5 0.2]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.2]
 [5.2 2.7 3.9 1.4]
 [5.7 3.8 1.7 0.3]
 [6.  2.7 5.1 1.6]]
X_train [[5.9 3.  4.2 1.5]
 [5.8 2.6 4.  1.2]
 [6.8 3.  5.5 2.1]
 [4.7 3.2 1.3 0.2]
 [6.9 3.1 5.1 2.3]
 [5.  3.5 1.6 0.6]
 [5.4 3.7 1.5 0.2]
 [5.  2.  3.5 1. ]
 [6.5 3.  5.5 1.8]
 [6.7 3.3 5.7 2.5]
```

```
 [6.  2.2 5.  1.5]
 [6.7 2.5 5.8 1.8]
 [5.6 2.5 3.9 1.1]
 [7.7 3.  6.1 2.3]
 [6.3 3.3 4.7 1.6]
 [5.5 2.4 3.8 1.1]
 [6.3 2.7 4.9 1.8]
 [6.3 2.8 5.1 1.5]
 [4.9 2.5 4.5 1.7]
 [6.3 2.5 5.  1.9]
 [7.  3.2 4.7 1.4]
 [6.5 3.  5.2 2. ]
 [6.  3.4 4.5 1.6]
 [4.8 3.1 1.6 0.2]
 [5.8 2.7 5.1 1.9]
 [5.6 2.7 4.2 1.3]
 [5.6 2.9 3.6 1.3]
 [5.5 2.5 4.  1.3]
 [6.1 3.  4.6 1.4]
 [7.2 3.2 6.  1.8]
 [5.3 3.7 1.5 0.2]
 [4.3 3.  1.1 0.1]
 [6.4 2.7 5.3 1.9]
 [5.7 3.  4.2 1.2]
 [5.4 3.4 1.7 0.2]
 [5.7 4.4 1.5 0.4]
 [6.9 3.1 4.9 1.5]
 [4.6 3.1 1.5 0.2]
 [5.9 3.  5.1 1.8]
 [5.1 2.5 3.  1.1]
 [4.6 3.4 1.4 0.3]
 [6.2 2.2 4.5 1.5]
 [7.2 3.6 6.1 2.5]
 [5.7 2.9 4.2 1.3]
 [4.8 3.  1.4 0.1]
 [7.1 3.  5.9 2.1]
 [6.9 3.2 5.7 2.3]
 [6.5 3.  5.8 2.2]
 [6.4 2.8 5.6 2.1]
 [5.1 3.8 1.6 0.2]
 [4.8 3.4 1.6 0.2]
 [6.5 3.2 5.1 2. ]
 [6.7 3.3 5.7 2.1]
 [4.5 2.3 1.3 0.3]
 [6.2 3.4 5.4 2.3]
 [4.9 3.  1.4 0.2]
 [5.7 2.5 5.  2. ]
 [6.9 3.1 5.4 2.1]
 [4.4 3.2 1.3 0.2]
 [5.  3.6 1.4 0.2]
 [7.2 3.  5.8 1.6]
 [5.1 3.5 1.4 0.3]
 [4.4 3.  1.3 0.2]
 [5.4 3.9 1.7 0.4]
 [5.5 2.3 4.  1.3]
 [6.8 3.2 5.9 2.3]
 [7.6 3.  6.6 2.1]
 [5.1 3.5 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [5.7 2.8 4.5 1.3]
 [6.6 3.  4.4 1.4]
 [5.  3.2 1.2 0.2]
 [5.1 3.3 1.7 0.5]
 [6.4 2.9 4.3 1.3]
 [5.4 3.4 1.5 0.4]
```

```
 [7.7 2.6 6.9 2.3]
 [4.9 2.4 3.3 1. ]
 [7.9 3.8 6.4 2. ]
 [6.7 3.1 4.4 1.4]
 [5.2 4.1 1.5 0.1]
 [6.  3.  4.8 1.8]
 [5.8 4.  1.2 0.2]
 [7.7 2.8 6.7 2. ]
 [5.1 3.8 1.5 0.3]
 [4.7 3.2 1.6 0.2]
 [7.4 2.8 6.1 1.9]
 [5.  3.3 1.4 0.2]
 [6.3 3.4 5.6 2.4]
 [5.7 2.8 4.1 1.3]
 [5.8 2.7 3.9 1.2]
 [5.7 2.6 3.5 1. ]
 [6.4 3.2 5.3 2.3]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 4.9 1.5]
 [6.7 3.  5.  1.7]
 [5.  3.  1.6 0.2]
 [5.5 2.4 3.7 1. ]
 [6.7 3.1 5.6 2.4]
 [5.8 2.7 5.1 1.9]
 [5.1 3.4 1.5 0.2]
 [6.6 2.9 4.6 1.3]
 [5.6 3.  4.1 1.3]
 [5.9 3.2 4.8 1.8]
 [6.3 2.3 4.4 1.3]
 [5.5 3.5 1.3 0.2]
 [5.1 3.7 1.5 0.4]
 [4.9 3.1 1.5 0.1]
 [6.3 2.9 5.6 1.8]
 [5.8 2.7 4.1 1. ]
 [7.7 3.8 6.7 2.2]
 [4.6 3.2 1.4 0.2]]
```

In [13]:

```
kn = KNeighborsClassifier(n_neighbors=5)
kn.fit(X_train, y_train)
```

Out[13]:

```
KNeighborsClassifier()
```

In [30]:

```
x_new = np.array([[5, 2.9, 1, 0.2]])
prediction = kn.predict(x_new)
```

In [31]:

```
print("Prediction value is: {} " .format(prediction))
print("predicted feature name is " ,iris_data["target_names"][prediction])
```

```
Prediction value is: [0]
predicted feature name is  ['setosa']
```

In [41]:

```
i = 1
x = X_test[i]
x_new = np.array([x])
for i in range(len(X_test)):
    x = X_test[i]
    x_new = np.array([x])
    prediction = kn.predict(x_new)
    print("\nACTUAL : {0} {1}, PREDICTED: {2}{3}" .format(y_test[i], iris_data["target_r
print("\nAccuracy score [ACCURACY] ", kn.score(X_test, y_test)  )
```

```
ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']
```

```
ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 2 virginica, PREDICTED: [2]['virginica']

ACTUAL : 1 versicolor, PREDICTED: [1]['versicolor']

ACTUAL : 0 setosa, PREDICTED: [0]['setosa']

ACTUAL : 1 versicolor, PREDICTED: [2]['virginica']

Accuracy score [ACCURACY]  0.9736842105263158
```

In [ ]:

In [ ]:

In [2]:

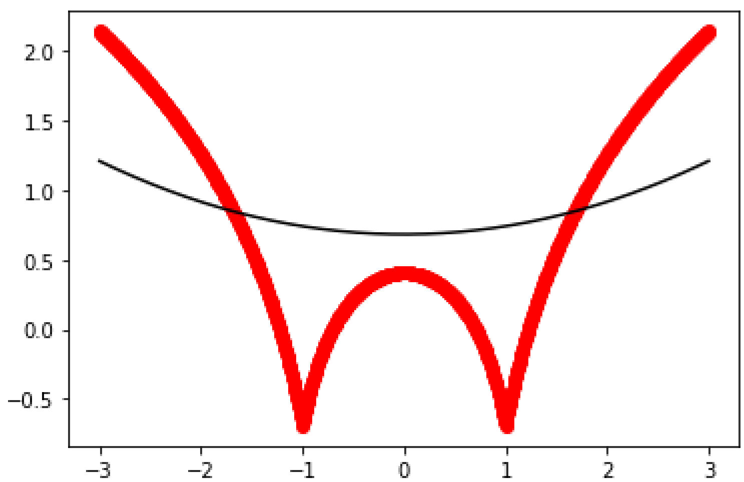
```
import numpy as np
import matplotlib.pyplot as plt

def local_reg(x0,X,Y,tau):
    x0 =[1,x0]
    X=[[1,i] for i in X]
    X= np.asarray(X)
    xw = X.T * np.exp(np.sum((X-x0)**2 ,axis=1)/ (-2*tau))
    beta= np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta
```

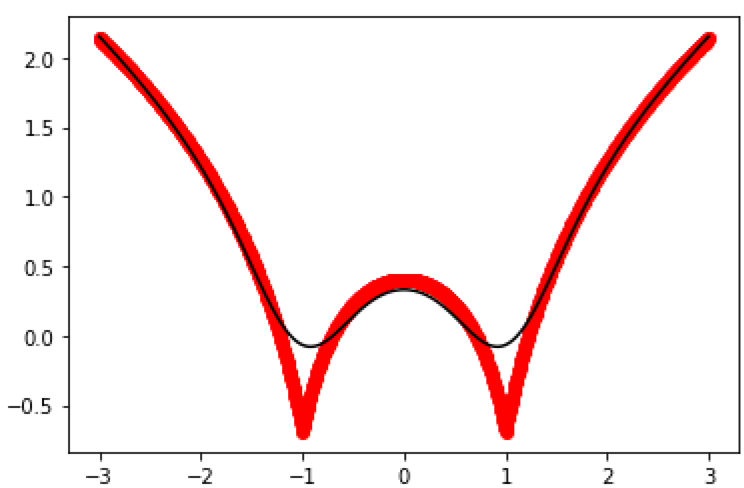
In [5]:

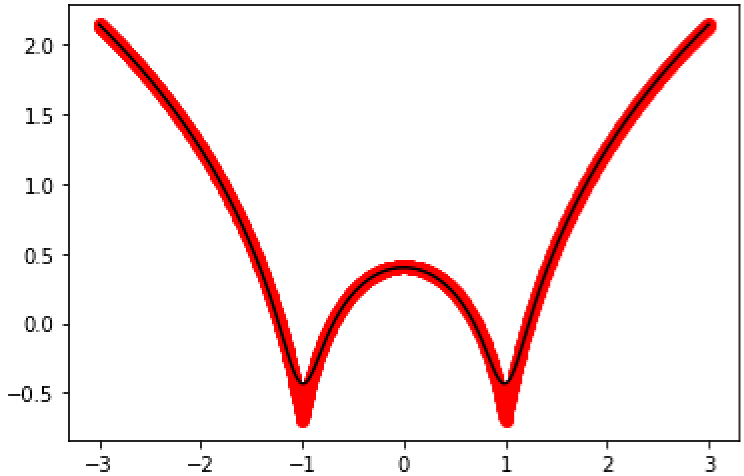
```
def draw(tau):
    prediction = [local_reg(x0,X,Y,tau) for x0 in domain]
    plt.plot(X,Y, 'o', color='red')
    plt.plot(domain, prediction, color='black')
    plt.show()
```

In [6]:

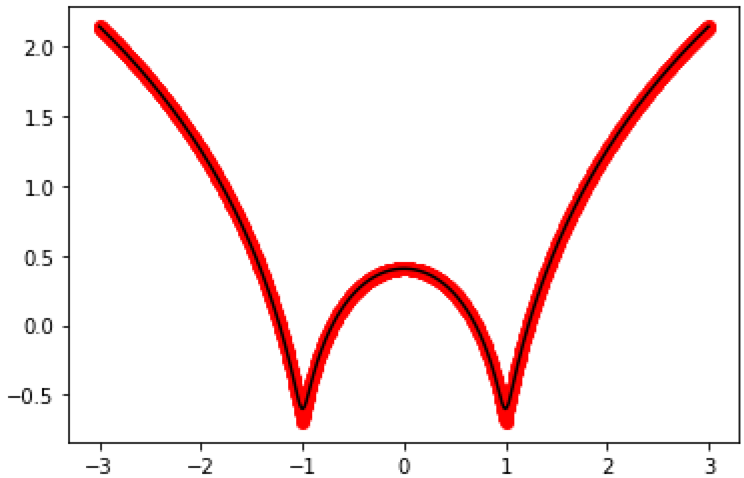
```
X= np.linspace(-3,3,num=1000)
domain = X
Y= np.log(np.abs(X**2 -1) +0.5)
draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
draw(0.0001)
```



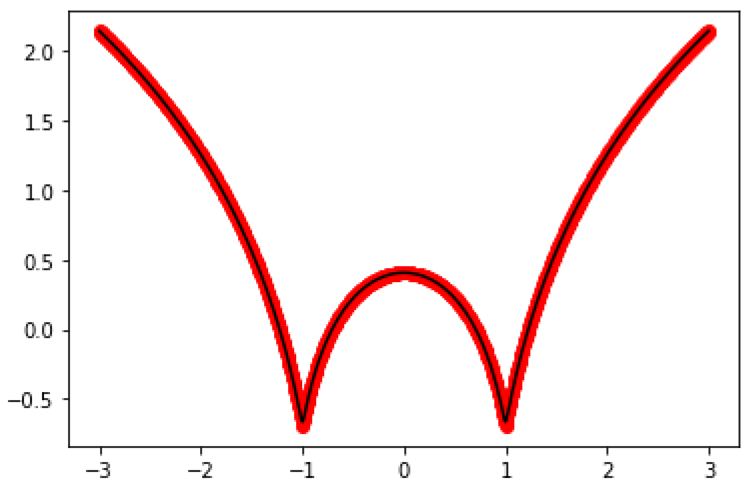




2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3



2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3



2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3

In [ ]:

```
7
```