In [10]:

```python
G={
 'S': [('A', 1), ('G', 10)],
 'A': [('B', 2), ('C', 1)],
 'B': [('D', 5)],
 'C': [('D', 3), ('G', 4)],
 'D': [('G', 2)],
}
H={'S': 5,
 'A': 3,
 'B': 4,
 'C': 2,
 'D': 6,
 'G': 0,
}
def astar(start,stop):
    opens=set(start)
    closed=set()
    g={}
    parent={}
    g[start]=0
    parent[start]=start

    while len(opens)>0:
        n=None
        for v in opens:
            if n==None or H[v]+g[v]<H[n]+g[n]:
                n=v
        if n==stop or G[n]==None:
            pass
        else:
            for (m,w) in get_neigh(n):
                if m not in opens and m not in closed:
                    opens.add(m)
                    parent[m]=n
                    g[m]=g[n]+w
                elif g[m]>g[n]+w:
                    g[m]=g[n]+w
                    parent[m]=n

                    if m in closed:
                        closed.remove(m)
                        opens.add(m)
        if n==None:
            print("Path not found")
            return None
        if n==stop:
            path=[]
            while parent[n]!=n:
                path.append(n)
                n=parent[n]
            path.append(start)
            path.reverse()
            print("Path found:{}".format(path))
            return path
        opens.remove(n)
        closed.add(n)
    print("Path does not exist\n. ")
    return None
def get_neigh(n):
    if n in G:
        return G[n]
    else:
```

```python
            return None
    astar('S','G')
```

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

Out[10]:

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

In [ ]:

```python

```

In [17]:

```
G={'A':[[('B',1),('C',1)],[('D',1)]],
   'B':[[('G',1)],[('H',1)]],
   'C':[[('J',1)]],
   'D':[[('E',1),('F',1)]],
   'G':[[('I',1)]],
  }
H={'A':1,'B':6,'C':2,'D':12,'E':2,'F':1,'G':5,'H':7,'I':7,'J':1}

class Graph:
    def __init__(self,S,G,H):
        self.s=S
        self.g=G
        self.h=H
        self.status={}
        self.parent={}
        self.solved={}

    def mincost(self,v):
        mcost=0
        mlist={}
        mlist[mcost]=[]
        flag=True
        for nt in self.g.get(v,""):
            c=0
            l=[]
            for n,w in nt:
                c+=self.h.get(n,0)+w
                l.append(n)
            if flag:
                mcost=c
                mlist[mcost]=l
                flag=False
            elif mcost>c:
                mcost=c
                mlist[mcost]=l
        return mcost,mlist[mcost]

    def p(self):
        print(self.solved)

    def aostar(self,v,back):
        print(v,self.solved)
        if self.status.get(v,0)>=0:
            mcost,mlist=self.mincost(v)
            self.h[v]=mcost
            self.status[v]=len(mlist)
            sol=True
            for n in mlist:
                self.parent[n]=v
                if self.status.get(n,0)!=-1:
                    sol=False
            if sol:
                self.status[v]=-1
                self.solved[v]=mlist
            if v!=self.s:
                self.aostar(self.parent[v],True)
            if not back:
                for n in mlist:
                    self.status[n]=0
                    self.aostar(n,False)
a=Graph('A',G,H)
a.aostar('A',False)
a.p()
```

```
A {}
B {}
A {}
G {}
B {}
A {}
I {}
G {'I': []}
B {'I': [], 'G': ['I']}
A {'I': [], 'G': ['I'], 'B': ['G']}
C {'I': [], 'G': ['I'], 'B': ['G']}
A {'I': [], 'G': ['I'], 'B': ['G']}
J {'I': [], 'G': ['I'], 'B': ['G']}
C {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
A {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
```

In [ ]:

```

```

In [8]:

```
import csv
a=[]
with(open('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 [13]:

```
num_attr=len(a[0])-1
temp=[]
S=['0']*num_attr
G=['?']*num_attr
for i in range(0, num_attr):
    S[i]=a[0][i]
for i in range(0,len(a)):
    if a[i][num_attr]=='Y':
        for j in range(0,num_attr):
            if a[i][j]!=S[j]:
                S[j]='?'
        for j in range(0,num_attr):
            for k in range(0,len(temp)):
                if temp[k][j]!=S[j] and temp[k][j]!='?':
                    del temp[k]
    if a[i][num_attr]=='N':
        for j in range(0,num_attr):
            if a[i][j]!=S[j] and S[j]!="?":
                G[j]=S[j]
                temp.append(G)
                G=['?']*num_attr
print(S)
if len(temp)==0:
    print(G)
else:
    print(temp)
```

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

In [ ]:

In [21]:

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

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

def entropy(x):
    c=Counter(i for i in x)
    n=len(x)*0.1
    prob=[i/n for i in c.values()]
    return sum(-i*math.log(i,2) for i in prob)
def information_gain(df,split,target):
    dfs=df.groupby(split)
    n=len(df)*0.1
    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
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
tree=id3(data,attr,'PlayTennis')
pprint(tree)
```

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

In [ ]:

In [18]:

```
import numpy as np
x=np.array([[2,9],[1,5],[3,6]],dtype=float)
y=np.array([[92],[86],[83]])
x=x/np.max(x,axis=0)
y=y/100
ep=5000
lr=0.1
def sigmoid(x):
    return(1/(1+np.exp(-x)))
def derivative(x):
    return(x*(1-x))
input_l=2
hidden_l=3
output_l=1

w_hidden=np.random.uniform(size=(input_l,hidden_l))
b_hidden=np.random.uniform(size=(1,hidden_l))
w_output=np.random.uniform(size=(hidden_l,output_l))
b_output=np.random.uniform(size=(1,output_l))

for i in range(ep):
    hidden=x.dot(w_hidden)+b_hidden
    hidden=sigmoid(hidden)
    output=hidden.dot(w_output)+b_output
    output=sigmoid(output)

    err_output=y-output
    d_output=derivative(output)
    err_output=err_output*d_output

    err_hidden=err_output.dot(w_output.T)
    d_hidden=derivative(err_hidden)
    err_hidden=err_hidden*d_hidden

    w_output+=hidden.T.dot(err_output)*lr
    w_hidden+=x.T.dot(err_hidden)*lr
print("Actual:",*x, "Output:",*y,"Predicted:",*output,sep="\n")
```

```
Actual:
[0.66666667 1.        ]
[0.33333333 0.55555556]
[1.         0.66666667]
Output:
[0.92]
[0.86]
[0.83]
Predicted:
[0.87456296]
[0.8612028]
[0.87419039]
```

In [ ]:

In [9]:
```
import math,csv
```

In [40]:
```
def safe_div(x,y):
    if y==0:
        return 0
    else:
        return x/y

def mean(n):
    return safe_div(sum(n),float(len(n)))

def stdev(n):
    a=mean(n)
    v= safe_div(sum([pow(x-a,2) for x in n]),float(len(n)-1))
    return math.sqrt(v)

def calcProb(x,mean,stdev):
    exp=math.exp(-safe_div((x-mean)**2,2*stdev**2))
    return safe_div(exp,(2*math.pi)**2*stdev)

testset=data=[list(map(float,i))for i in[x for x in list(csv.reader(open("ConceptLearning.csv")))]]
trainset=[testset.pop(0) for x in range(int(len(data)*0.9))]

seperated={}
for i in trainset:
    if i[-1] not in seperated:
        seperated[i[-1]]=[]
    seperated[i[-1]].append(i[:-1])
# print(seperated.items())
summaries={}
for cv, instances in seperated.items():
    summaries[cv]=[(mean(attr),stdev(attr))for attr in zip(*instances)]
# print(summaries.items())
pred=[]
for i in range(len(testset)):
    prob={}
    for cv, cs in summaries.items():
        prob[cv]=1
    for j in range(len(cs)):
        m,s=cs[j]
        x=testset[i][j]
        prob[cv]*=calcProb(x,m,s)
    bl,bp=None,-1
    for cv,p in prob.items():
        if bl==None or bp<p:
            bl=cv
            bp=p
    pred.append(bl)
actual=[i[-1]for i in testset]
count=0
for i,j in zip(actual,pred):
    if i==j:
        count+=1
print("Accuracy={}".format(safe_div(count,float(len(actual)))*100))
print(pred)
print(actual)
```

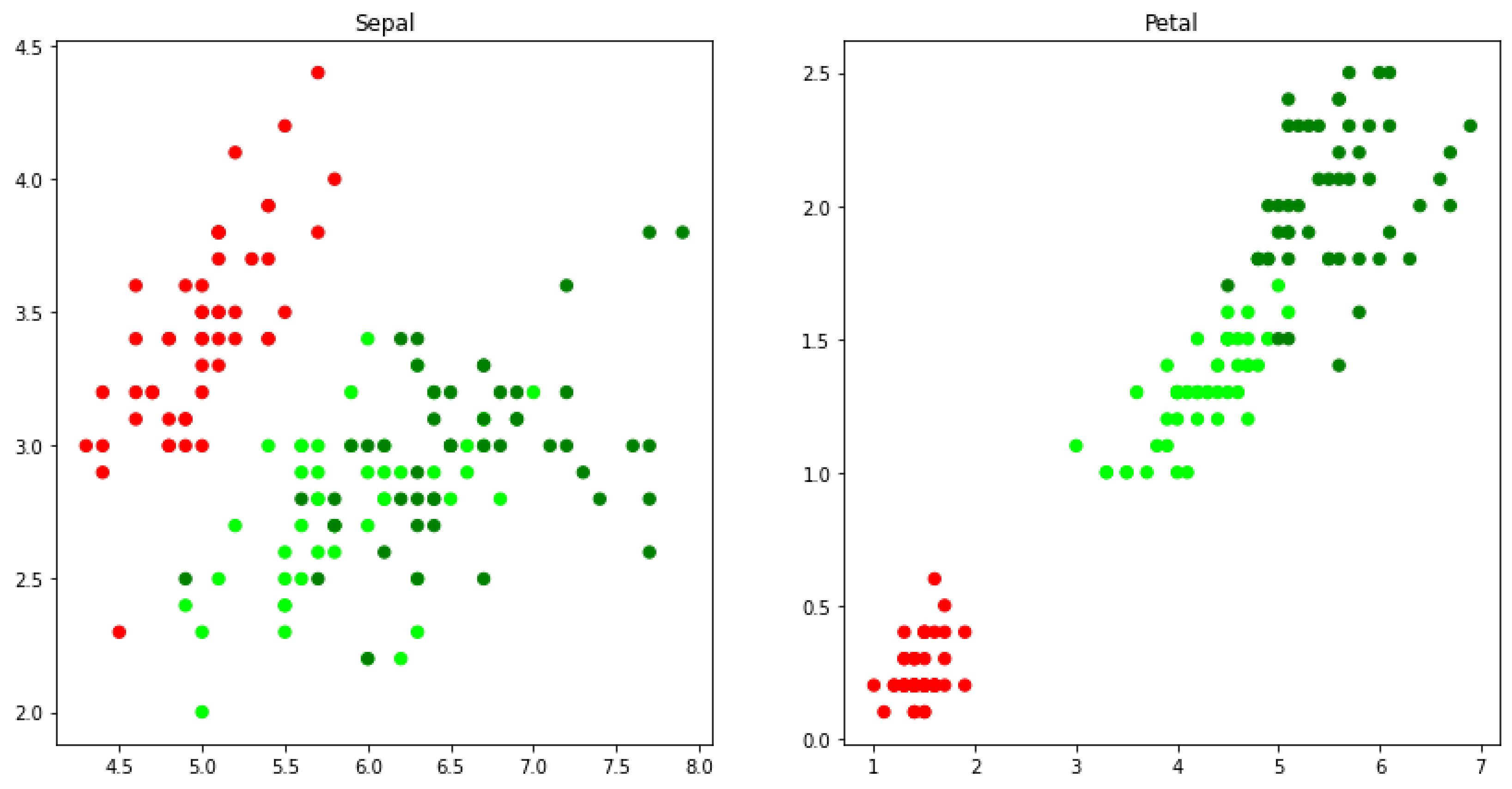
```
Accuracy=50.0
[5.0, 5.0]
[10.0, 5.0]
```

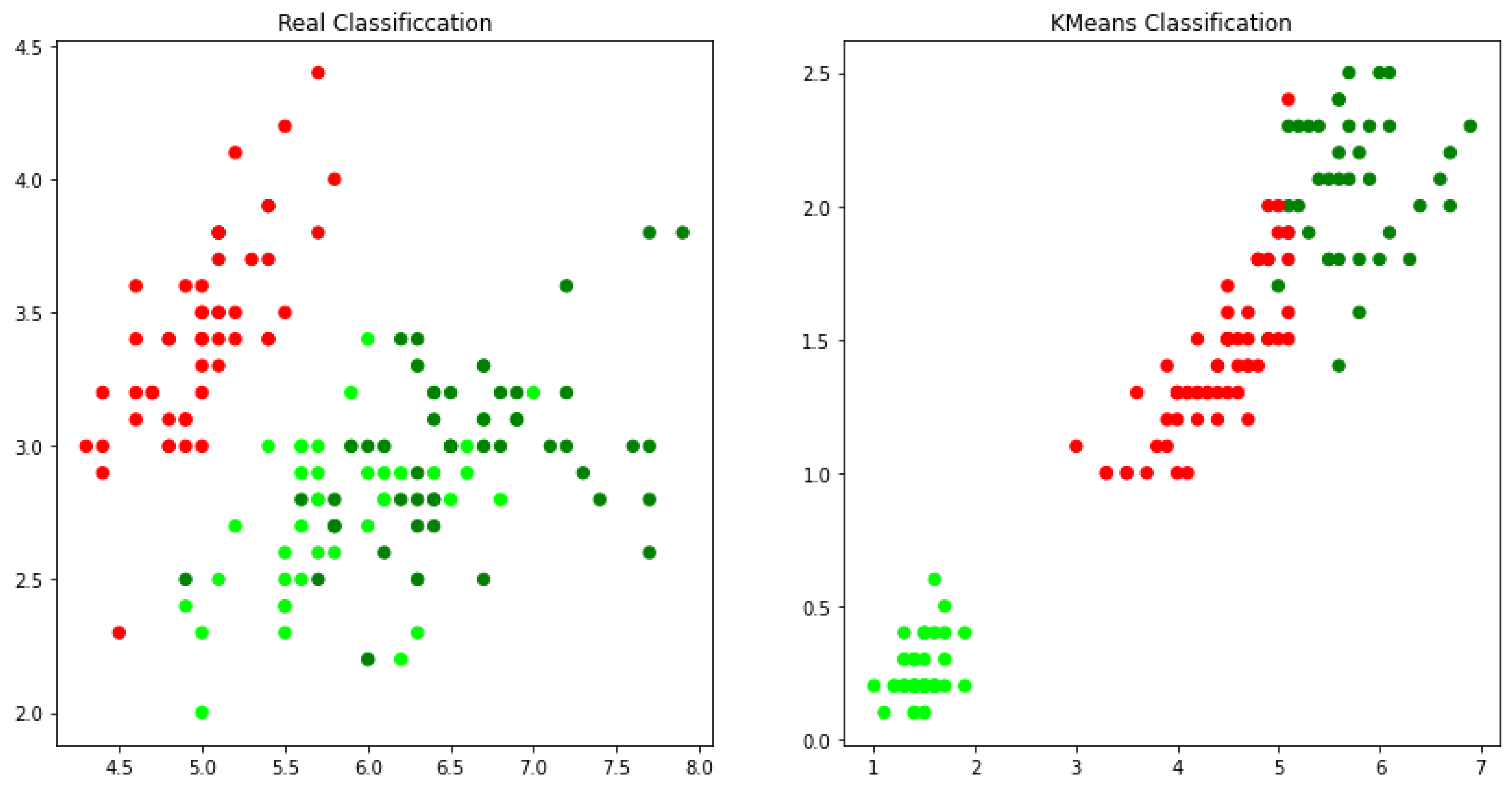
In [ ]:

In [16]:

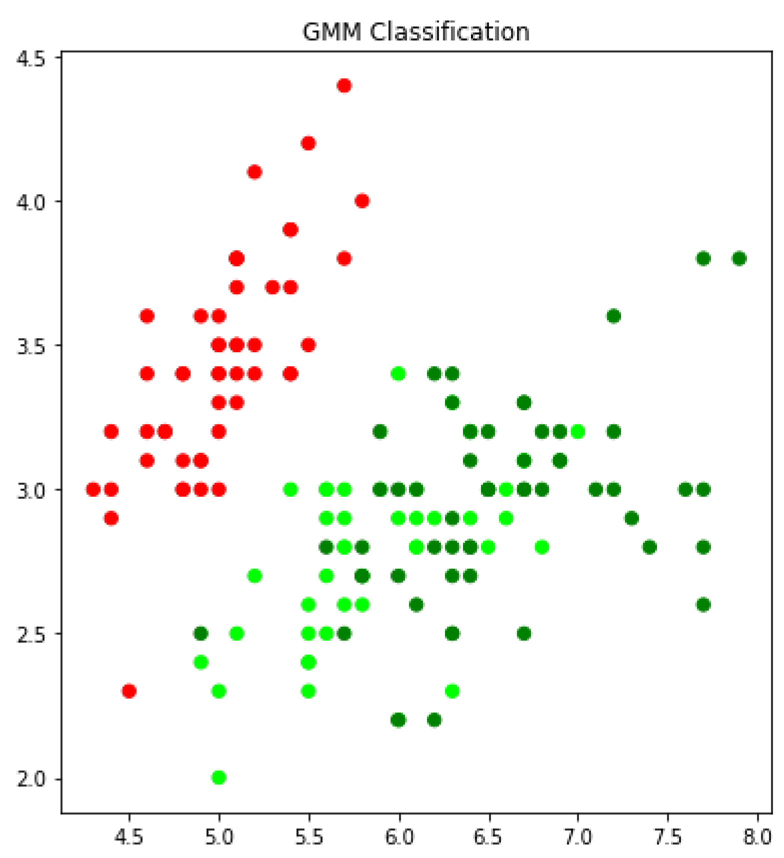
```python
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

D=load_iris()
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
X=pd.DataFrame(D.data)
Y=pd.DataFrame(D.target)

X.columns=["Sl","Sw","Pl","Pw"]
Y.columns=["target"]

plt.figure(figsize=(14,7))
color=np.array(['red','lime','green'])
print(Y)
plt.subplot(1,2,1)
plt.scatter(X.Sl,X.Sw,c=color[Y.target],s=40)
plt.title("Sepal")

plt.subplot(1,2,2)
plt.scatter(X.Pl,X.Pw,c=color[Y.target],s=40)
plt.title("Petal")
plt.show()

model=KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14,7))
color=np.array(['red','lime','green'])

plt.subplot(1,2,1)
plt.scatter(X.Sl,X.Sw,c=color[Y.target],s=40)
plt.title("Real Classificcation")

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

m=rename(model.labels_)
print("What KMeans thought:{}\n,".format(m))
print("Accuracy:{}\n".format(sm.accuracy_score(Y,m)))
print("Confusion Matrix:\n{}\n".format(sm.confusion_matrix(Y,m)))

scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
print("Xs is :{}\n".format(xs.sample(5)))

plt.figure(figsize=(14,7))
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
Y_clusture_gmm=gmm.predict(xs)

plt.subplot(1,2,1)
plt.scatter(X.Sl,X.Sw,c=color[Y_clusture_gmm],s=40)
plt.title("GMM Classification")
plt.show()

m=rename(Y_clusture_gmm)
print("What GMM Thought:{}\n,".format(m))
print("Accuracy:{}\n".format(sm.accuracy_score(Y,m)))
print("Confusion Matrix:\n{}\n".format(sm.confusion_matrix(Y,m)))
```

```
     target
0         0
1         0
2         0
3         0
4         0
..      ...
145       2
146       2
147       2
148       2
149       2

[150 rows x 1 columns]
```




```
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 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]
,
Accuracy:0.8933333333333333

Confusion Matrix:
[[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]

Xs is :           Sl        Sw        Pl        Pw
76   1.159173 -0.592373  0.592246  0.264142
121 -0.294842 -0.592373  0.649083  1.053935
15  -0.173674  3.090775 -1.283389 -1.052180
111  0.674501 -0.822570  0.876433  0.922303
49  -1.021849  0.558611 -1.340227 -1.315444
```




```
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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1
 1 1 1 2 1 1 1 1 1 2 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]
,
Accuracy:0.9666666666666667

Confusion Matrix:
[[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```

In [ ]:

In [12]:

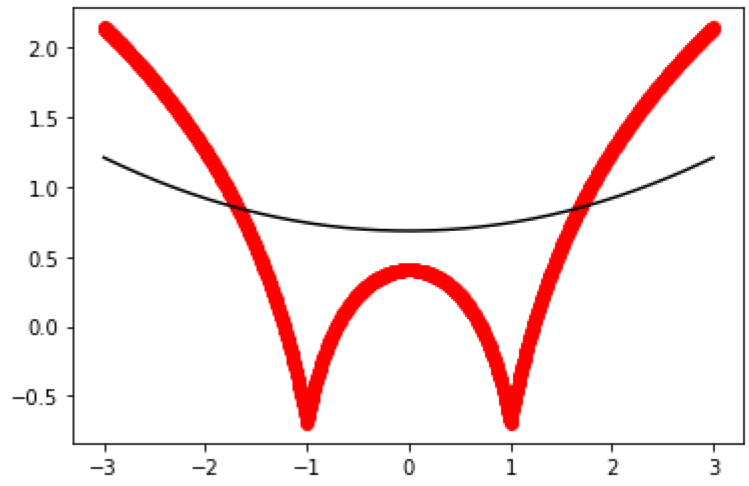
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np

d=load_iris()
xtrain,xtest,ytrain,ytest=train_test_split(d.data,d.target,random_state=0)
kn=KNeighborsClassifier(n_neighbors=5)
kn.fit(xtrain,ytrain)
xnew=np.asarray([[5,1.9,1,0.2]])
p=kn.predict(xnew)
print("Predicted value is: {}\n".format(d.target_names[p[0]]))
print("{:<10}:{:<10}".format("Actual","Predicted"))
for i in range (len(xtest)):
    xnew=np.asarray([xtest[i]])
    p=kn.predict(xnew)
    print("{:<10}:{:<10}".format(d.target_names[ytest[i]],d.target_names[p[0]]))
print("Accuracy:{:.2f}%".format(kn.score(xtest,ytest)))
```

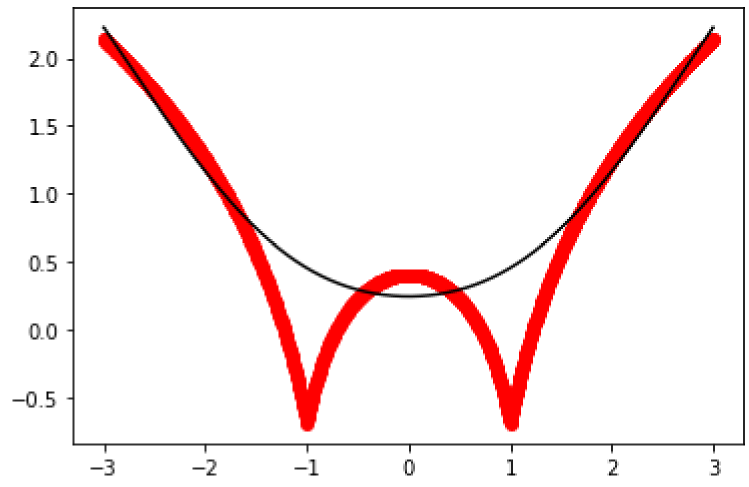
```
Predicted value is: setosa

Actual    :Predicted
virginica :virginica
versicolor:versicolor
setosa    :setosa
virginica :virginica
setosa    :setosa
virginica :virginica
setosa    :setosa
versicolor:versicolor
versicolor:versicolor
versicolor:versicolor
virginica :virginica
versicolor:versicolor
versicolor:versicolor
versicolor:versicolor
versicolor:versicolor
setosa    :setosa
versicolor:versicolor
versicolor:versicolor
setosa    :setosa
setosa    :setosa
virginica :virginica
versicolor:versicolor
setosa    :setosa
setosa    :setosa
virginica :virginica
setosa    :setosa
setosa    :setosa
versicolor:versicolor
versicolor:versicolor
setosa    :setosa
virginica :virginica
versicolor:versicolor
setosa    :setosa
virginica :virginica
virginica :virginica
versicolor:versicolor
setosa    :setosa
versicolor:virginica
Accuracy:0.97%
```

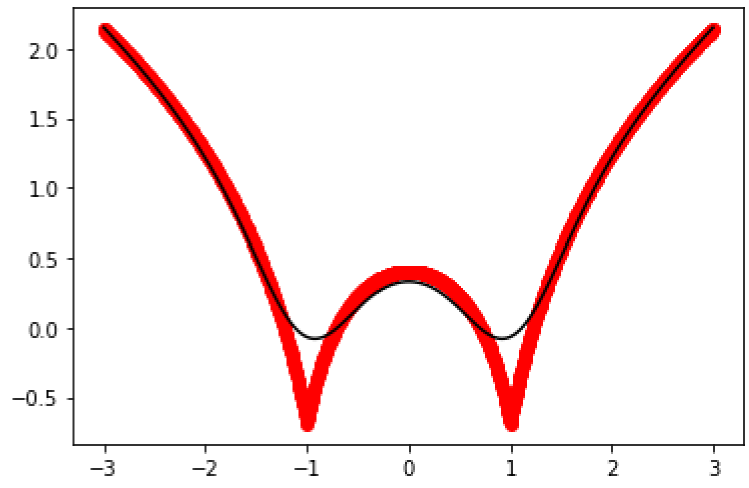
In [ ]:

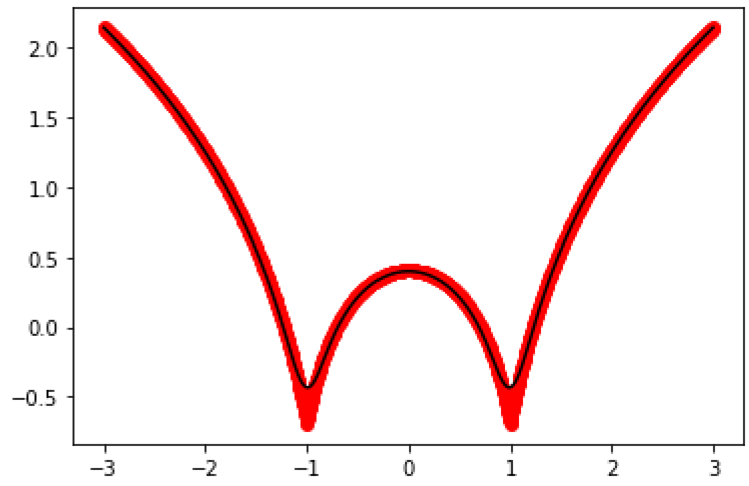
In [18]:

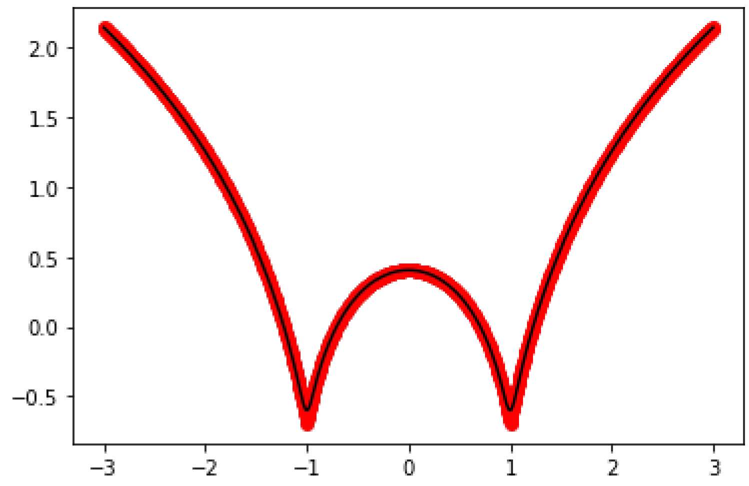
```
import matplotlib.pyplot as plt
import numpy as np

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

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











In [ ]: