In [6]:

```python
G={
    "A":[("B",6),("F",3)],
    "B":[("C",3),("D",2)],
    "C":[("D",1),("E",5)],
    "D":[("C",1),("E",8)],
    "E":[("I",5),("J",5)],
    "F":[("G",1),("H",7)],
    "G":[("I",3)],
    "H":[("I",2)],
    "I":[("E",5),("J",3)]
}
H={'A':10,'B':5,'C':5,'D':7,'E':3,'F':6,'G':5,'H':3,'I':1,'J':0}
def astar(s,e):
    opens=set(s)
    close=set()
    g={}
    p={}
    g[s]=0
    p[s]=s
    while len(opens)>0:
        n=None
        for v in opens:
            if n==None or g[v]+H[v]<g[n]+H[n]:
                n=v
        if n==e or G[n]==None:
            pa=[]
            while p[n]!=n:
                pa.append(n)
                n=p[n]
            pa.append(s)
            pa.reverse()
            print(pa)
            return
        for m,w in G.get(n,None):
            if m not in opens and m not in close:
                opens.add(m)
                g[m]=g[n]+w
                p[m]=n
            elif g[m]>g[n]+w:
                g[m]=g[n]+w
                p[m]=n
                if m in close:
                    close.remove(m)
                    opens.add(m)
        opens.remove(n)
        close.add(n)
    print("Path not found")
astar('A','J')
```

```
['A', 'F', 'G', 'I', 'J']
```

In [19]:

```
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.solved={}
        self.status={}
        self.parent={}
    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 [34]:

```
import csv
d=list(csv.reader(open('trainingexamples.csv')))
s=d[1][:-1]
g=[['?' for j in range(len(s))] for i in range(len(s))]
k=0
print("S[{}] : {} \nG[{}] : {} \n".format(k,['?']*len(s),k,g))
for i in d:
    if i[-1]=='Y':
        for j in range(len(s)):
            if i[j]!=s[j]:
                s[j]='?'
                g[j][j]='?'
    elif i[-1]=='N':
        for j in range(len(s)):
            if i[j]!=s[j]:
                g[j][j]=s[j]
            else:
                g[j][j]='?'
    gh=[]
    k+=1
    for a in g:
        for j in a:
            if j!='?':
                gh.append(a)
    print("S[{}] : {} \nG[{}] : {} \n".format(k,s,k,gh))
```

```
S[0] : ['?', '?', '?', '?', '?', '?']
G[0] : [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

S[1] : ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
G[1] : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?',
'?'], ['?', '?', 'Normal', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Sam
e']]

S[2] : ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
G[2] : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?',
'?'], ['?', '?', 'Normal', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Sam
e']]

S[3] : ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
G[3] : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', 'Same']]

S[4] : ['Sunny', 'Warm', '?', 'Strong', '?', '?']
G[4] : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?',
'?']]
```

In [56]:

```
from collections import Counter
from pprint import pprint
import pandas as pd
import math
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)*1.0
    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)*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
def id3(df,attr,target,default=None):
    c=Counter(x for x 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 [68]:

```
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.max(x,axis=0)
y=y/100
def sigmoid(x):
    return 1/(1+np.exp(-x))
def derivative(x):
    return x*(1-x)
ep=5000
lr=0.1
inputl=2
hiddenl=3
outputl=1
weight_hidden=np.random.uniform(size=(inputl,hiddenl))
bias_hidden=np.random.uniform(size=(1,hiddenl))
weight_output=np.random.uniform(size=(hiddenl,outputl))
bias_output=np.random.uniform(size=(1,outputl))
for i in range(ep):
    hidden_layer=x.dot(weight_hidden)+bias_hidden
    hidden_layer=sigmoid(hidden_layer)

    output=hidden_layer.dot(weight_output)+bias_output
    output=sigmoid(output)

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

    error_hidden=error_output.dot(weight_output.T)
    d_hidden=derivative(hidden_layer)
    error_hidden=error_hidden*d_hidden

    weight_output+=hidden_layer.T.dot(error_output)*lr
    weight_hidden+=x.T.dot(error_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]
[0.89]
Predicted:
[0.89298053]
[0.88219026]
[0.89499811]
```

In [97]:

```
import csv,math
def div(x,y):
    return 0 if y==0 else x/y
def mean(x):
    return div(sum(x),float(len(x)))
def std(x):
    a=mean(x)
    return math.sqrt(div(sum((i-a)**2 for i in x),float(len(x)-1)))
def calprob(x,m,s):
    return div(1,s*math.sqrt(2*math.pi))*(math.exp(-div((x-m)**2,2*s**2)))
test=data=[list(map(float,j)) for j in [ i for i in list(csv.reader(open('diabetes2.csv')))
print('data: ',*data,sep='\n')
train=[test.pop(0) for i in range(int(len(data)*0.9))]
print('Train: ',*train,'Test: ',*test,sep='\n')
separated={}
for i in train:
    if i[-1] not in separated:
        separated[i[-1]]=[]
    separated[i[-1]].append(i[:-1])
print("Separated:",*separated.items(),sep="\n")
summaries={}
for cv,ci in separated.items():
    summaries[cv]=[[mean(x),std(x)] for x in zip(*ci)]
print("Summaries:",*summaries.items(),sep='\n')
prediction=[]
for i in range(len(test)):
    prob={}
    for cv,ci in summaries.items():
        prob[cv]=1
    for j in range(len(ci)):
        x=test[i][j]
        m,s=ci[j]
        prob[cv]*=calprob(x,m,s)
    ba,bp=None,0
    for clv,cp in prob.items():
        if ba==None or cp>bp:
            ba=clv
            bp=cp
    prediction.append(ba)
print("Prediction:",prediction)
act=[i[-1] for i in test]
print("Actual    :",act)
c=sum([act[i]==prediction[i] for i in range(len(act))])
print("Accuracy  : {:.2f}%".format(div(c,float(len(act)))*100))
```

```
data:
[1.0, 1.0, 1.0, 1.0, 5.0]
[1.0, 1.0, 1.0, 2.0, 5.0]
[2.0, 1.0, 1.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 2.0, 5.0]
[2.0, 3.0, 2.0, 2.0, 10.0]
[1.0, 2.0, 1.0, 1.0, 5.0]
[1.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 2.0, 1.0, 10.0]
[1.0, 2.0, 2.0, 2.0, 10.0]
[2.0, 2.0, 1.0, 2.0, 10.0]
```

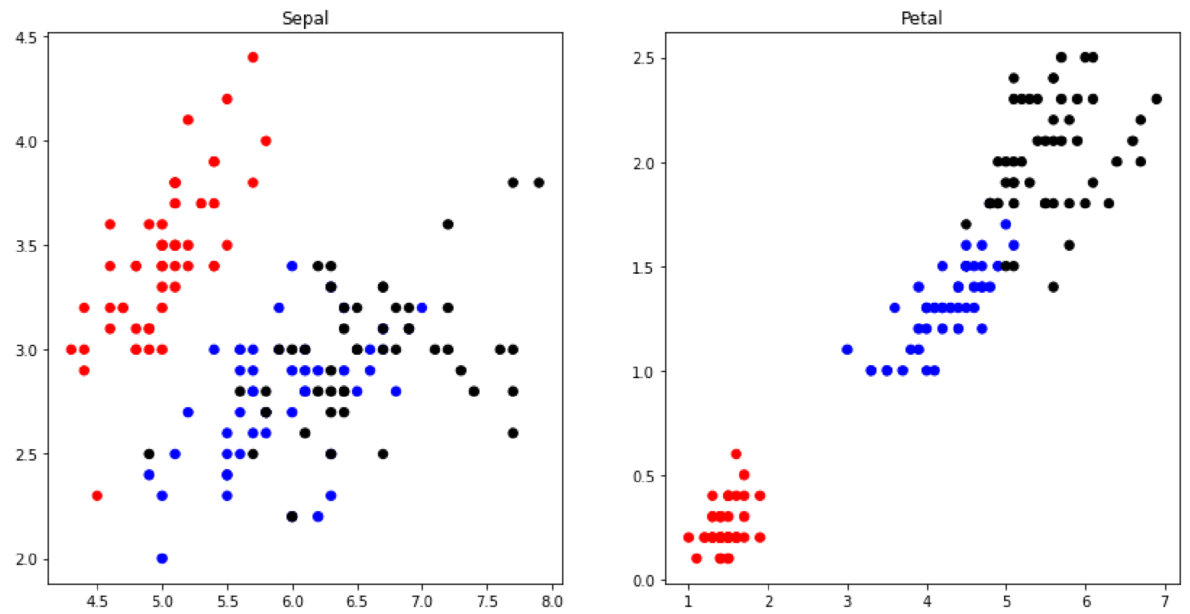
```
[2.0, 1.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 2.0, 5.0]
Train:
[1.0, 1.0, 1.0, 1.0, 5.0]
[1.0, 1.0, 1.0, 2.0, 5.0]
[2.0, 1.0, 1.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 2.0, 5.0]
[2.0, 3.0, 2.0, 2.0, 10.0]
[1.0, 2.0, 1.0, 1.0, 5.0]
[1.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 2.0, 1.0, 10.0]
[1.0, 2.0, 2.0, 2.0, 10.0]
[2.0, 2.0, 1.0, 2.0, 10.0]
Test:
[2.0, 1.0, 2.0, 1.0, 10.0]
[3.0, 2.0, 1.0, 2.0, 5.0]
Separated:
(5.0, [[1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 2.0], [3.0, 3.0, 2.0, 2.0], [1.
0, 2.0, 1.0, 1.0]])
(10.0, [[2.0, 1.0, 1.0, 1.0], [3.0, 2.0, 1.0, 1.0], [3.0, 3.0, 2.0, 1.0],
[2.0, 3.0, 2.0, 2.0], [1.0, 3.0, 2.0, 1.0], [3.0, 2.0, 2.0, 1.0], [1.0, 2.0,
2.0, 2.0], [2.0, 2.0, 1.0, 2.0]])
Summaries:
(5.0, [[1.5, 1.0], [1.75, 0.9574271077563381], [1.25, 0.5], [1.5, 0.57735026
91896257]])
(10.0, [[2.125, 0.8345229603962802], [2.25, 0.7071067811865476], [1.625, 0.5
175491695067657], [1.375, 0.5175491695067657]])
Prediction: [5.0, 5.0]
Actual    : [10.0, 5.0]
Accuracy  : 50.00%
```

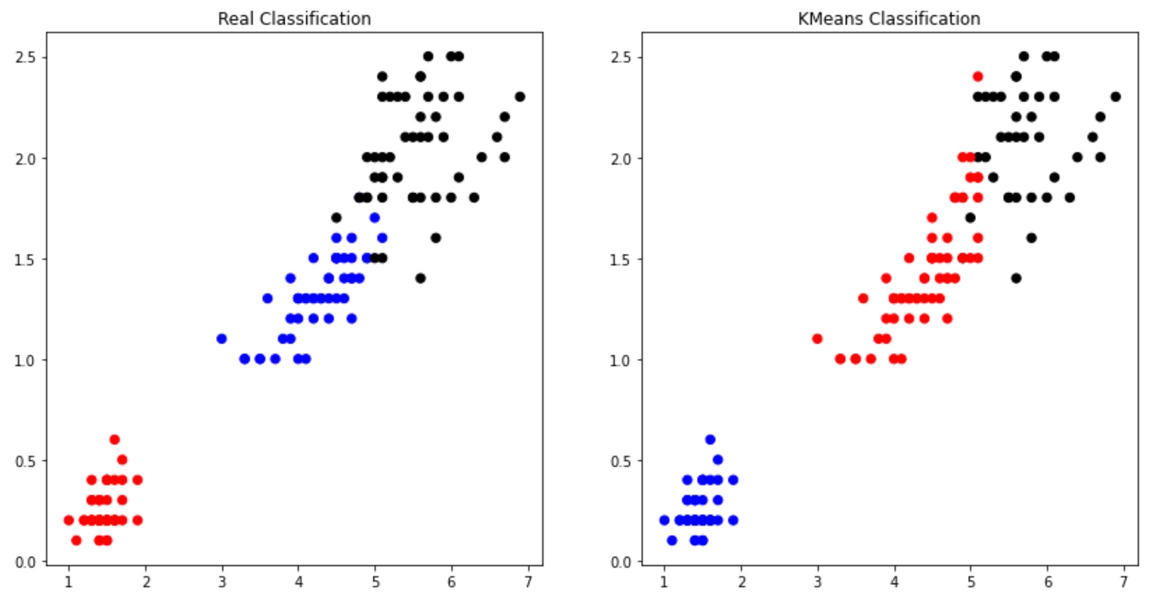
In [148]:

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn import preprocessing
import sklearn.metrics as sm
import matplotlib.pyplot as pt
import pandas as pd
import numpy as np
df=load_iris()
x=pd.DataFrame(df.data)
x.columns=['sl','sw','pl','pw']
print(x.sample(5))
y=pd.DataFrame(df.target)
y.columns=['t']
print(y.sample(5))
pt.figure(figsize=(14,7))
c=np.array(['red','blue','black'])
pt.subplot(1,2,1)
pt.scatter(x.sl,x.sw,color=c[y.t],s=40)
pt.title('Sepal')
pt.subplot(1,2,2)
pt.scatter(x.pl,x.pw,color=c[y.t],s=40)
pt.title('Petal')
pt.show()
model=KMeans(n_clusters=3)
model.fit(x)
pt.figure(figsize=(14,7))
c=np.array(['red','blue','black'])
pt.subplot(1,2,1)
pt.scatter(x.pl,x.pw,color=c[y.t],s=40)
pt.title('Real Classification')
pt.subplot(1,2,2)
pt.scatter(x.pl,x.pw,color=c[model.labels_],s=40)
pt.title('KMeans Classification')
pt.show()
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)):
        p=l2.index(s[i])
        s[i]=l1[p]
    return s
m=rename(model.labels_)
print("What Kmeans Thought: ",m)
print("Accuracy : {:.2f}%".format(sm.accuracy_score(y,m)*100))
print("Confusion Matix: ",*sm.confusion_matrix(y,m),sep='\n')

km=preprocessing.StandardScaler()
km.fit(x)
xsa=km.transform(x)
xs=pd.DataFrame(xsa,columns=x.columns)
print(xs.sample(5))
p=GaussianMixture(n_components=3)
p.fit(xs)
ps=p.predict(xs)
pt.figure(figsize=(14,7))
```

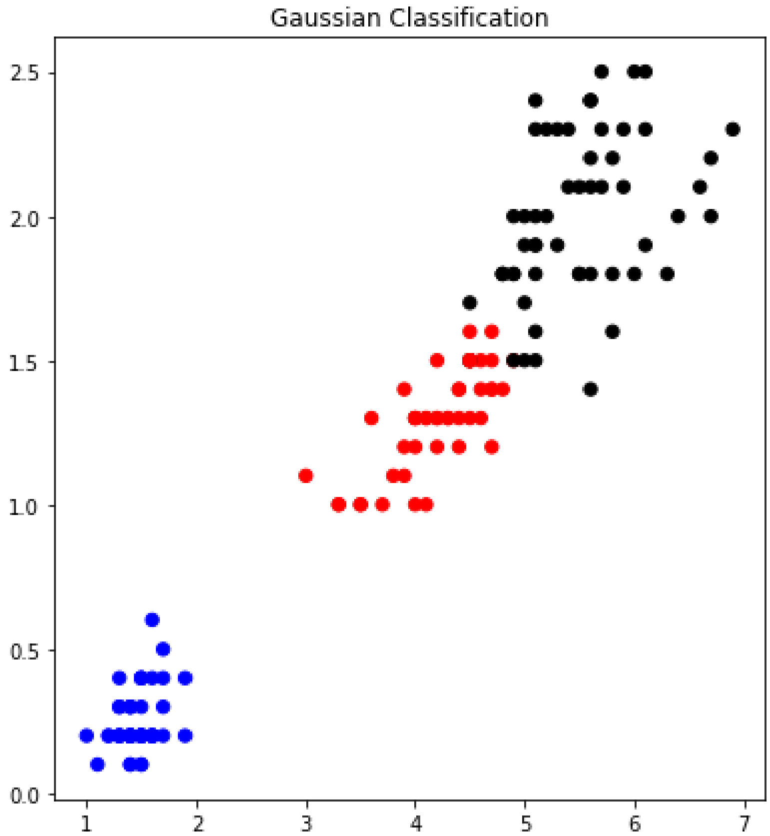
```python
pt.subplot(1,2,1)
pt.scatter(x.pl,x.pw,color=c[ps],s=40)
pt.title('Gaussian Classification')
pt.show()

m=rename(ps)
print("What Gaussian Thought: ",m)
print("Accuracy : {:.2f}%".format(sm.accuracy_score(y,m)*100))
print("Confusion Matix: ",*sm.confusion_matrix(y,m),sep='\n')
```

```
      sl   sw   pl   pw
126  6.2  2.8  4.8  1.8
25   5.0  3.0  1.6  0.2
40   5.0  3.5  1.3  0.3
53   5.5  2.3  4.0  1.3
96   5.7  2.9  4.2  1.3
     t
95   1
140  2
68   1
82   1
99   1
```





```
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 : 89.33%
Confusion Matix:
[50  0  0]
[ 0 48  2]
[ 0 14 36]
          sl        sw        pl        pw
86  1.038005  0.098217  0.535409  0.395774
69 -0.294842 -1.282963  0.080709 -0.130755
18 -0.173674  1.709595 -1.169714 -1.183812
99 -0.173674 -0.592373  0.194384  0.132510
9  -1.143017  0.098217 -1.283389 -1.447076
```




```
What Gaussian 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 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 : 96.67%
Confusion Matix:
[50  0  0]
[ 0 45  5]
[ 0  0 50]
```

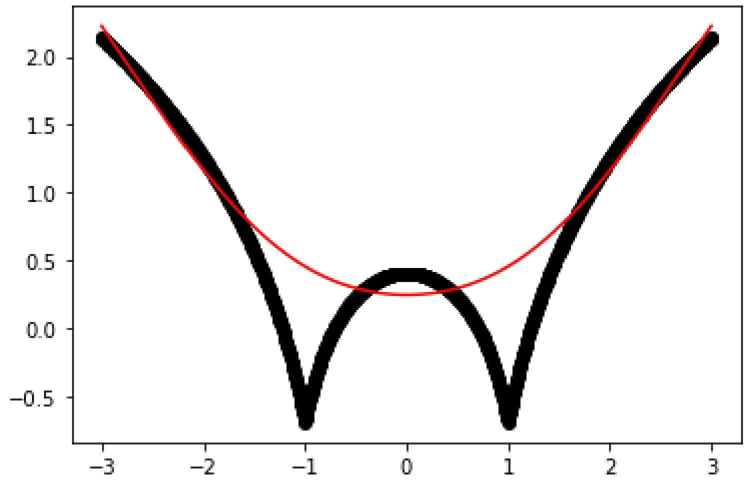
In [149]:

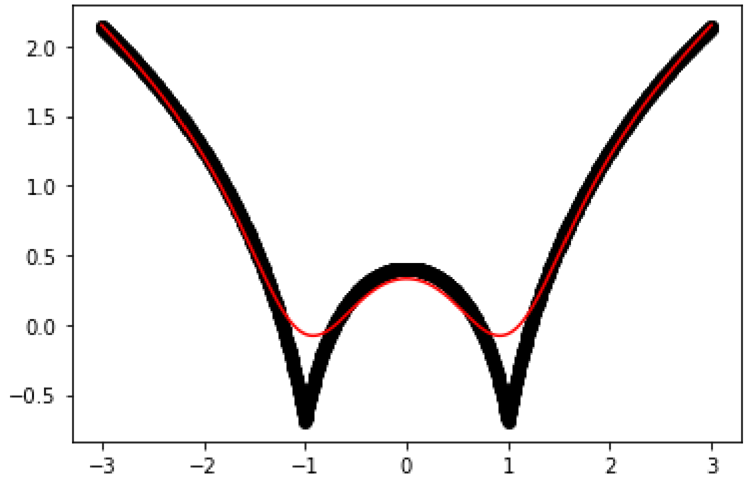
```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
d=load_iris()
xtrain,xtest,ytrain,ytest=train_test_split(d.data,d.target,random_state=0)
print("XTRAIN: ",*xtrain,"XTEST: ",*xtest,"YTRAIN: ",ytrain,"YTEST: ",ytest,sep="\n")
kn=KNeighborsClassifier(n_neighbors=5)
kn.fit(xtrain,ytrain)
xnew=np.asarray([[5,2.9,1,0.2]])
p=kn.predict(xnew)
print(d.target_names[p[0]])
print("{:<10} : {:<10}".format("Actual","Predict"))
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("Accuarcy   : {:.2f}%".format(kn.score(xtest,ytest)*100))
```

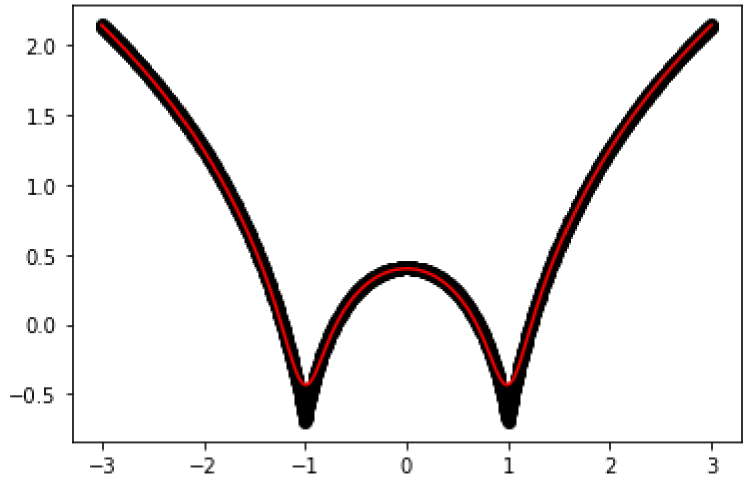
```
XTRAIN:
[5.9 3.  4.2 1.5]
[5.8 2.6 4.  1.2]
[6.8 3.  5.5 2.1]
[4.7 3.2 1.3 0.2]
[6.9 3.1 5.1 2.3]
[5.  3.5 1.6 0.6]
[5.4 3.7 1.5 0.2]
[5.  2.  3.5 1. ]
[6.5 3.  5.5 1.8]
[6.7 3.3 5.7 2.5]
[6.  2.2 5.  1.5]
[6.7 2.5 5.8 1.8]
[5.6 2.5 3.9 1.1]
[7.7 3.  6.1 2.3]
[6.3 3.3 4.7 1.6]
[5.5 2.4 3.8 1.1]
[6.3 2.7 4.9 1.8]
[6.3 2.8 5.1 1.5]
```

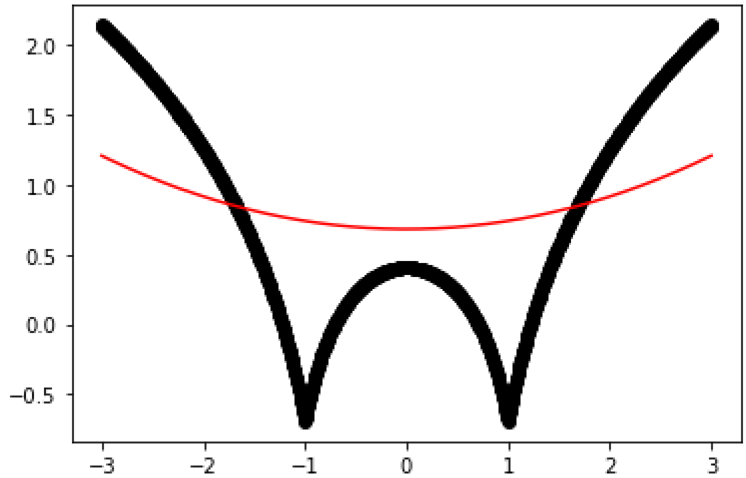
In [150]:

```
import numpy as np
import matplotlib.pyplot as pt
def linear_regression(x0,x,y,t):
    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*t))
    return np.linalg.pinv(xw @ x) @ xw @ y @ x0
def draw(t):
    p=[linear_regression(x0,x,y,t) for x0 in d]
    pt.plot(x,y,'o',color="black")
    pt.plot(d,p,color="red")
    pt.show()
x=np.linspace(3,-3,num=1000)
d=x
y=np.log(np.abs(x**2-1)+0.5)
draw(1)
draw(0.1)
draw(0.01)
draw(10)
```




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 [ ]:

```

```