CANDIDATE-ELIMINATION ALGORITHM

In [3]:

```
import pandas as pd
```

In [4]:

```
df=pd.read_csv("program1.csv")
spe_df=df.loc[df["enjoysport"].str.upper()=="YES"]
gene_df=df.loc[df["enjoysport"].str.upper()=="NO"]
spe_df=spe_df.iloc[:,:-1]
gene_df=gene_df.iloc[:,:-1]
base=spe_df.iloc[0]
for x in range(1,len(spe_df)):
    base=base.where(spe_df.iloc[x]==base,other="???")
print("Specific:-\n",base.values)
for x in range(len(gene_df)):
    base=base.where(base!=gene_df.iloc[x],other="???")
print("General")
for i,x in enumerate(base):
    if x!="???":
        l=["???"]*len(base)
        l[i]=x
        print(l)
```

```
Specific:-
 ['sunny' 'warm' '???' 'strong' '???' '???']
General
['sunny', '???', '???', '???', '???', '???']
['???', 'warm', '???', '???', '???', '???']
```

ID3 algorithm

In [1]:
```
import pandas as pd
import numpy as np
```

In [6]:
```
dataset=pd.read_csv('program2.csv',names=['outlook','temperature','humidity','wind','class',])
attributes=('Outlook','Temperature','Humidity','Wind','PlayTennis')
```

In [7]:
```
def entropy(target_col):
    elements,counts=np.unique(target_col,return_counts=True)
    entropy=np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts))
        for i in range(len(elements))])
    return entropy
```

In [9]:
```
def InfoGain(data,split_attribute_name,target_name="class"):
    total_entropy=entropy(data[target_name])
    vals,counts=np.unique(data[split_attribute_name],return_counts=True)
    Weighted_entropy=np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[target_name]) for i in
    range(len(vals))])
    Information_Gain=total_entropy-Weighted_entropy
    return Information_Gain
```

In [14]:
```
def ID3(data,originaldata,features,target_attribute_name="class",parent_node_class=None):
    if len(np.unique(data[target_attribute_name]))<=1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data)==0:
        return
        np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attibute_name],return_counts=True)[1])]
    elif len(features)==0:
        return parent_node_class
    else:
        parent_node_class=np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name],return_counts=True)[1])]
        item_values=[InfoGain(data,feature,target_attribute_name) for feature
        in features]
        best_feature_index=np.argmax(item_values)
        best_feature=features[best_feature_index]
        tree={best_feature:{}}
        features=[i for i in features if i!=best_feature]
        for value in np.unique(data[best_feature]):
            value=value
            sub_data=data.where(data[best_feature]==value).dropna()
            subtree=ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
            tree[best_feature][value]=subtree
    return(tree)
```

In [15]:
```
def predict(query,tree,default=1):
    for key in list(query.keys()):
        if key in list(tree.keys()):
            try:
                result=tree[key][query[key]]
            except:
                return default
            result=tree[key][query[key]]
            if isinstance(result,dict):
                return predict(query,result)
            else:
                return result
```

In [16]:
```
def train_test_split(dataset):
    training_data=dataset.iloc[:14].reset_index(drop=True)
    return training_data
```

In [17]:
```
def test(data,tree):
    queries=data.iloc[:,:-1].to_dict(orient="records")
    predicted=pd.DataFrame(columns=["predicted"])
    for i in range(len(data)):
        predicted.loc[i,"predicted"]=predict(queries[i],tree,1.0)
    print('The predicted accuracy is:',(np.sum(predicted["predicted"]==data["class"])/len(data))*100,'%')

XX=train_test_split(dataset)
training_data=XX
tree=ID3(training_data,training_data,training_data.columns[:-1])
print('\nDisplay Tree\n',tree)
print('len=',len(training_data))
test(training_data,tree)
```

```
Display Tree
 {'outlook': {'Outlook': 'PlayTennis', 'Overcast': 'Yes', 'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}
len= 14
The predicted accuracy is: 100.0 %
```

Backpropagation algorithm

In [1]:
```
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
```

In [2]:
```
def sigmoid (x):
    return 1/(1 + np.exp(-x))
def derivatives_sigmoid(x):
    return x * (1 - x)
```

In [3]:
```
epoch=5000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_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,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

In [4]:
```
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)
```

In [5]:
```
E0 = y-output
outgrad =derivatives_sigmoid(output)
d_output = E0* 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
```

In [6]:
```
print("Input:\n" + str(x))
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
```

In [7]:
```
print("Actual Output: \n" + str(y))
```

```
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
```

In [8]:

```
print("PredictedOutput: \n",output)
```

```
PredictedOutput: 
 [[0.8528972 ]
 [0.8434814 ]
 [0.85410258]]
```

naïve Bayesian classifier

In [1]:

```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
```

In [4]:

```
data = pd.read_csv('program4.csv')
print("The first 5 values of data is :\n",data.head())
```

```
The first 5 values of data is :
    Outlook Temperature Humidity   Windy PlayTennis
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
```

In [5]:

```
x = data.iloc[:,:-1]
print("\nThe first 5 values of train data is\n",x.head())
```

```
The first 5 values of train data is
    Outlook Temperature Humidity   Windy
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
```

In [6]:

```
y = data.iloc[:,-1]
print("\nThe first 5 values of train output is\n",y.head())
```

```
The first 5 values of train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object
```

In [8]:

```
le_Outlook = LabelEncoder()
x.Outlook = le_Outlook.fit_transform(x.Outlook)

le_Temperature=LabelEncoder()
x.Temperature=le_Temperature.fit_transform(x.Temperature)

le_Humidity=LabelEncoder()
x.Humidity=le_Humidity.fit_transform(x.Humidity)

le_Windy=LabelEncoder()
x.Windy=le_Windy.fit_transform(x.Windy)

print("\nNow the train data is :\n",x.head())
```

```
Now the train data is :
   Outlook  Temperature  Humidity  Windy
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
```

In [9]:

```python
le_PlayTennis=LabelEncoder()
y=le_PlayTennis.fit_transform(y)
print("\nNow the train output is \n",y)
```

```
Now the train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 2]
```

In [10]:

```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20)

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))
```

```
Accuracy is: 0.3333333333333333
```

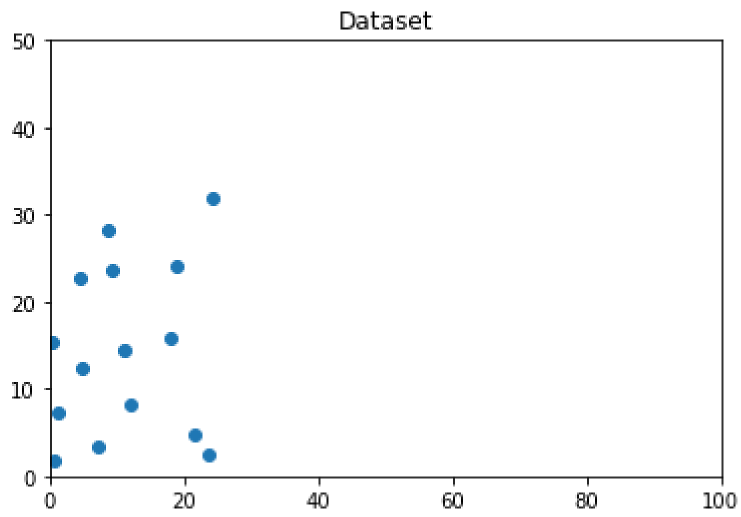
k-Means algorithm

In [1]:

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

In [3]:

```
X=pd.read_csv("program5.csv")
x1 = X['V1'].values
x2 = X['V2'].values
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()
```



In [5]:

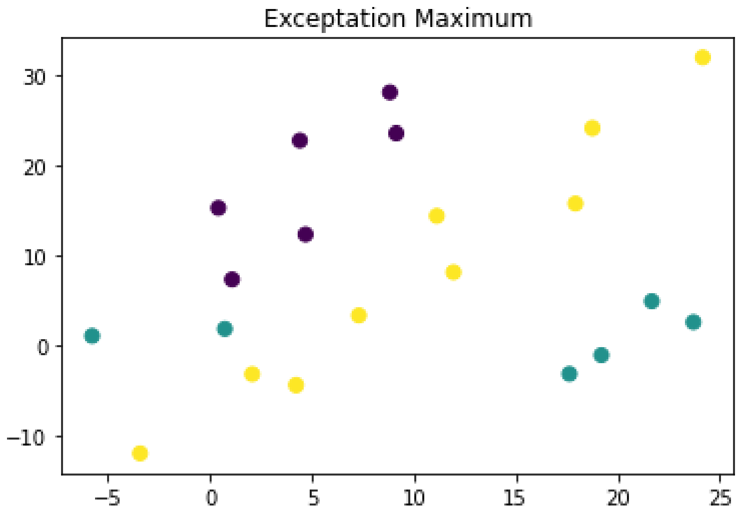
```
gmm = GaussianMixture(n_components=3)
gmm.fit(X)
em_predictions = gmm.predict(X)
print("\nEM predictions")
print(em_predictions)
print("mean:\n",gmm.means_)
print('\n')
print("Covariances\n",gmm.covariances_)
print(X)
```

```
EM predictions
[2 2 0 2 1 2 1 0 1 2 0 0 2 2 1 0 2 1 0 1 2]
mean:
 [[ 3.87004698 16.19467857]
 [14.03598519  0.7258207 ]
 [10.93962489  9.54606718]]


Covariances
 [[[ 17.14601232  30.4406603 ]
  [ 30.4406603   76.55265727]]

 [[ 99.34162937   5.04919157]
```

```
 [  5.04919157   9.32933094]]

[[ 72.59138887 114.03114933]
 [114.03114933 188.03678761]]]
[[  2.072345  -3.24169 ]
 [ 17.93671   15.78481 ]
 [  1.083576   7.319176]
 [ 11.12067   14.40678 ]
 [ 23.71155    2.557729]
 [ 24.16993   32.02478 ]
 [ 21.66578    4.892855]
 [  4.693684  12.34217 ]
 [ 19.21191   -1.12137 ]
 [  4.230391  -4.44154 ]
 [  9.12713   23.60572 ]
 [  0.407503  15.29705 ]
 [  7.314846   3.309312]
 [ -3.4384   -12.0253  ]
 [ 17.63935   -3.21235 ]
 [  4.415292  22.81555 ]
 [ 11.94122    8.122487]
 [  0.725853   1.806819]
 [  8.815273  28.1326  ]
 [ -5.77359    1.0248  ]
 [ 18.76943   24.16946 ]]
```

In [6]:

```
plt.title('Exceptation Maximum')
plt.scatter(X[:,0], X[:,1],c=em_predictions,s=50)
plt.show()
```



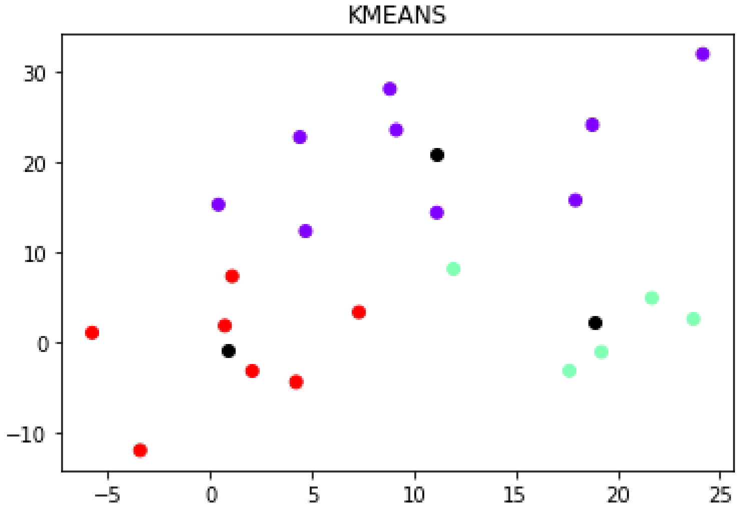
In [7]:

```
import matplotlib.pyplot as plt1
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
plt.title('KMEANS')
plt1.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt1.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1],
color='black')
```

```
[[11.05062467 20.95321333]
 [18.833962    2.2478702 ]
 [ 0.88786014 -0.89263186]]
[2 0 2 0 1 0 1 0 1 2 0 0 2 2 1 0 1 2 0 2 0]
```

Out[7]:

```
<matplotlib.collections.PathCollection at 0x1a2e020f9a0>
```


KMEANS
30
20
10
0
−10
−5
0
5
10
15
20
25

k-Nearest Neighbour algorithm

In [1]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,confusion_matrix
```

In [2]:

```
from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_label=iris.target
print(iris_data)
print(iris_label)
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_label)
```

```
[[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]]
[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]
```

In [3]:

```python
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
print('Confusion matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
Confusion matrix is as follows
[[13  0  0]
 [ 0 13  0]
 [ 0  0 12]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

In [ ]:

non-parametric Locally Weighted Regressionalgorithm

In [1]:

```
from math import ceil
import numpy as np
from scipy import linalg
```

In [2]:

```
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
```

In [3]:

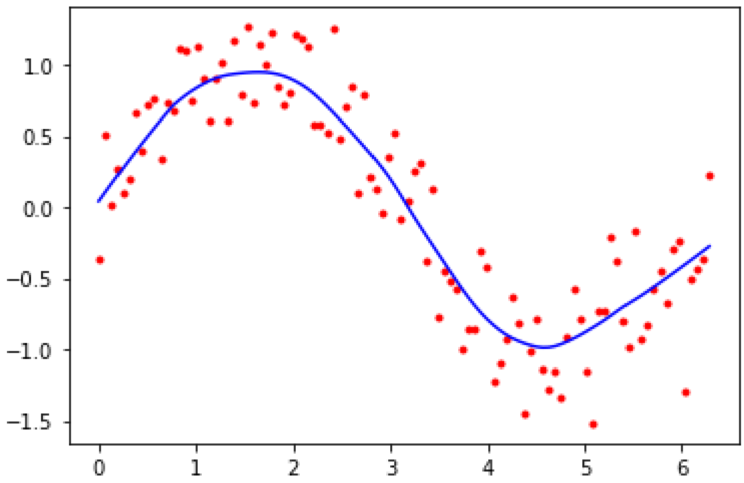
```
import math
n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f =0.25
iterations=3
yest = lowess(x, y, f, iterations)
```

In [4]:

```
import matplotlib.pyplot as plt
plt.plot(x,y,"r.")
plt.plot(x,yest,"b-")
```

Out[4]:

```
[<matplotlib.lines.Line2D at 0x298f822a910>]
```



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

A* Algorithm

In [18]:

```
from collections import deque
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]

    # This is heuristic function which is having equal values for all nodes
    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }
        return H[n]

    def a_star_algorithm(self, start, stop):
        open_lst = set([start])
        closed_lst = set([])

        poo = {}
        poo[start] = 0

        par = {}
        par[start] = start

        while len(open_lst) > 0:
            n = None

            for v in open_lst:
                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                    n = v;

            if n == None:
                print('Path does not exist!')
                return None

            if n == stop:
                reconst_path = []

                while par[n] != n:
                    reconst_path.append(n)
                    n = par[n]

                reconst_path.append(start)
                reconst_path.reverse()

                print('Path found: {}'.format(reconst_path))
                return reconst_path

            for (m, weight) in self.get_neighbors(n):
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    par[m] = n
                    poo[m] = poo[n] + weight

                else:
```

```
                    if poo[m] > poo[n] + weight:
                        poo[m] = poo[n] + weight
                        par[m] = n

                        if m in closed_lst:
                            closed_lst.remove(m)
                            open_lst.add(m)

            open_lst.remove(n)
            closed_lst.add(n)

        print('Path does not exist!')
        return None

adjac_lis = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}
graph1 = Graph(adjac_lis)
graph1.a_star_algorithm('A', 'D')
```

```
Path found: ['A', 'B', 'D']
```

Out[18]:

```
['A', 'B', 'D']
```

AO* Search algorithm

In [2]:

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}


    def applyAOStar(self):
        self.aoStar(self.start, False)
    def getNeighbors(self, v):
        return self.graph.get(v,'')
    def getStatus(self,v):
        return self.status.get(v,0)
    def setStatus(self,v, val):
        self.status[v]=val
    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0)
    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value

    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:",self.start)
        print("------------------------------------------------------------")
        print(self.solutionGraph)
        print("------------------------------------------------------------")


    def computeMinimumCostChildNodes(self, v):
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True

        for nodeInfoTupleList in self.getNeighbors(v):
            cost=0
            nodeList=[]

            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)

            if flag==True:
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList
                flag=False

            else:
                if minimumCost>cost:
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList
        return minimumCost, costToChildNodeListDict[minimumCost]

    def aoStar(self, v, backTracking):
        print("HEURISTIC VALUES :", self.H)
        print("SOLUTION GRAPH :", self.solutionGraph)
        print("PROCESSING NODE :", v)
        print("-----------------------------------------------------------------------------------------")

        if self.getStatus(v) >= 0:
            minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
            print(minimumCost, childNodeList)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True

            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False

            if solved==True:
                self.setStatus(v,-1)
                self.solutionGraph[v]=childNodeList

            if v!=self.start:
                self.aoStar(self.parent[v], True)

            if backTracking==False:
                for childNode in childNodeList:
                    self.setStatus(childNode,0)
                    self.aoStar(childNode, False)

print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
```

```python
graph1 = {
    '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(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

```
Graph - 1
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
6 ['G']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : G
-----------------------------------------------------------------------------------------
8 ['I']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
8 ['H']
HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
12 ['B', 'C']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : I
-----------------------------------------------------------------------------------------
0 []
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': []}
PROCESSING NODE : G
-----------------------------------------------------------------------------------------
1 ['I']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I']}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
2 ['G']
HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : C
-----------------------------------------------------------------------------------------
2 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : J
-----------------------------------------------------------------------------------------
0 []
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE : C
-----------------------------------------------------------------------------------------
1 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
5 ['B', 'C']
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
------------------------------------------------------------
```