Experiment- 1 FIND-S ALGORITHM

Dataset (Data.csv)

Example,Sky,Temp,Humidity,Wind,Water,Forecast,Enjoysport
1,Sunny,Warm,Normal,Strong,Warm,Same,Yes
2,Sunny,Warm,High,Strong,Warm,Same,Yes
3,Rainy,Cold,High,Strong,Warm,Change,No
4,Sunny,Warm,High,Strong,Cool,Change,Yes

Program:

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

In [2]:
```
d = pd.read_csv("Data.csv")
```

In [3]:
```
print(d)
```

```
   Example    Sky  Temp Humidity    Wind Water Forecast Enjoysport
0        1  Sunny  Warm   Normal  Strong  Warm     Same        Yes
1        2  Sunny  Warm     High  Strong  Warm     Same        Yes
2        3  Rainy  Cold     High  Strong  Warm   Change         No
3        4  Sunny  Warm     High  Strong  Cool   Change        Yes
```

In [4]:
```
a = np.array(d)[:,:-1]
print(" The attributes are: ",a)
```

```
 The attributes are:  [[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 [2 'Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 [3 'Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 [4 'Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

In [5]:
```
t = np.array(d)[:,-1]
print("The target is: ",t)
```

```
The target is:  ['Yes' 'Yes' 'No' 'Yes']
```

In [7]:
```
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
             specific_hypothesis = c[i].copy()
        break

    for i, val in enumerate(c):
         if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                 if val[x] != specific_hypothesis[x]:
                         specific_hypothesis[x] = '?'
    else:
         pass
    return specific_hypothesis
    print(" The final hypothesis is:",train(a,t))
```

In [8]:
```
print(" The final hypothesis is:",train(a,t))
```

```
 The final hypothesis is: ['?' 'Sunny' 'Warm' '?' 'Strong' '?' '?']
```

In [ ]:

Experiment-2 CANDIDATE ELIMINATION ALGORITHM

DATASET: (Data.cvs)

Example,Sky,Temp,Humidity,Wind,Water,Forecast,Enjoysport
1,Sunny,Warm,Normal,Strong,Warm,Same,Yes
2,Sunny,Warm,High,Strong,Warm,Same,Yes
3,Rainy,Cold,High,Strong,Warm,Change,No
4,Sunny,Warm,High,Strong,Cool,Change,Yes

Program:

In [2]:
```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('Data.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
```

In [5]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))]
                 for i in range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    specific_h[x]='?'
                    general_h[x][x]='?'
                print(specific_h)
        print(specific_h)
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x]=specific_h[x]
                else:
                    general_h[x][x]='?'
        print("Steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    indices = [i for i, val in enumerate(general_h)
               if val==['?','?','?','?','?','?']]
    for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:",s_final, sep="\n")
print("Final General_h:",g_final, sep="\n")
```

Experiment-3 ID3

Program:

In [1]:
```
import numpy as np
import math
import csv
```

In [2]:
```
def read_data(tennisdata):
    with open(tennisdata, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)
```

In [3]:
```
class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute
```

In [4]:
```
def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict
```

In [5]:
```
def entropy(S):
    items = np.unique(S)

    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums
```

In [6]:
```
def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv
```

In [7]:
```
def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)

    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node
```

In [8]:
```
def empty(size):
    s = ""
    for x in range(size):
        s += "   "
    return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)
```

In [9]:
```
metadata, traindata = read_data("tennisdata.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

```
Outlook
   Overcast
      b'Yes'
   Rainy
      Windy
         b'False'
            b'Yes'
         b'True'
            b'No'
   Sunny
      Humidity
         b'High'
            b'No'
         b'Normal'
            b'Yes'
```

Dataset: (tennisdata.csv)

Outlook,Temperature,Humidity,Windy,PlayTennis
Sunny,Hot,High,False,No
Sunny,Hot,High,True,No
Overcast,Hot,High,False,Yes
Rainy,Mild,High,False,Yes
Rainy,Cool,Normal,False,Yes
Rainy,Cool,Normal,True,No
Overcast,Cool,Normal,True,Yes
Sunny,Mild,High,False,No
Sunny,Cool,Normal,False,Yes
Rainy,Mild,Normal,False,Yes
Sunny,Mild,Normal,True,Yes
Overcast,Mild,High,True,Yes
Overcast,Hot,Normal,False,Yes
Rainy,Mild,High,True,No

Experiment- 4 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
def sigmoid (x):
     return 1/(1 + np.exp(-x))
def sigmoid (x):
    return 1/(1 + np.exp(-x))
epoch=7000
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))

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 =  sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
hiddengrad =  sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr

wh += X.T.dot(d_hiddenlayer) *lr

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

OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.75495298]
 [0.74126699]
 [0.75392083]]
```

Experiment-5 NAÏVE BAYESIAN CLASSIFIER

Program:

Dataset(same as Experiment 3 tennisdata.csv)

In [ ]:
```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
```

In [2]:
```
data = pd.read_csv('tennisdata.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  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3     Rainy        Mild     High  False        Yes
4     Rainy        Cool   Normal  False        Yes
```

In [3]:
```
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  False
1     Sunny         Hot     High   True
2  Overcast         Hot     High  False
3     Rainy        Mild     High  False
4     Rainy        Cool   Normal  False
```

In [4]:
```
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 [6]:
```
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      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0
```

In [7]:
```
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 1 0]
```

In [8]:
```
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: 0.6666666666666666
```

Experiment- 6 DOCUMENT CLASSIFICATION USING NAÏVE BAYESIAN CLASSIFIER

DATASET (document.csv)

I love this sandwich,pos
This is an amazing place,pos
I feel very good about these beers,pos
This is my best work,pos
What an awesome view,pos
I do not like this restaurant,neg
I am tired of this stuff,neg
I can't deal with this,neg
He is my sworn enemy,neg
My boss is horrible,neg
This is an awesome place,pos
I do not like the taste of this juice,neg
I love to dance,pos
I am sick and tired of this place,neg
What a great holiday,pos
That is a bad locality to stay,neg
We will have good fun tomorrow,pos
I went to my enemy's house today,neg

Program:

In [7]:
```
import pandas as pd
msg = pd.read_csv('document.csv', names=['message', 'label'])
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
```

```
Total Instances of Dataset:  18
```

In [8]:
```
X = msg.message
y = msg.labelnum
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
from sklearn.feature_extraction.text import CountVectorizer

count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
```

In [13]:
```
df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names_out())
print(df[0:5])
```

```
   about  am  amazing  an  awesome  beers  boss  dance  do  enemy  ...  tired  \
0      0   0        1   1        0      0     0      0   0      0  ...      0
1      0   0        0   0        0      0     0      0   0      0  ...      0
2      0   0        0   0        0      0     0      0   0      1  ...      0
3      1   0        0   0        0      1     0      0   0      0  ...      0
4      0   0        0   1        1      0     0      0   0      0  ...      0

   to  today  tomorrow  very  view  we  went  what  will
0   0      0         0     0     0   0     0     0     0
1   0      0         0     0     0   0     0     1     0
2   0      0         0     0     0   0     0     0     0
3   0      0         0     1     0   0     0     0     0
4   0      0         0     0     0   0     0     0     0

[5 rows x 44 columns]
```

In [14]:
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)
```

In [15]:
```
for doc, p in zip(Xtrain, pred):
    p = 'pos' if p == 1 else 'neg'
    print("%s -> %s" % (doc, p))
```

```
This is an amazing place -> neg
What a great holiday -> neg
He is my sworn enemy -> pos
I feel very good about these beers -> pos
This is an awesome place -> neg
```

In [16]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(ytest, pred))
print('Recall: ', recall_score(ytest, pred))
print('Precision: ', precision_score(ytest, pred))
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

```
Accuracy Metrics:

Accuracy:  0.6
Recall:  0.5
Precision:  0.5
Confusion Matrix:
 [[2 1]
 [1 1]]
```

Experiment- 7 BAYESIAN NETWORK

Dataset (heart_disease.csv)

age,Gender,Family,diet,Lifestyle,cholestrol,heartdisease
0,0,1,1,3,0,1
0,1,1,1,3,0,1
1,0,0,0,2,1,1
4,0,1,1,3,2,0
3,1,1,0,0,2,0
2,0,1,1,1,0,1
4,0,1,0,2,0,1
0,0,1,1,3,0,1
3,1,1,0,0,2,0
1,1,0,0,0,2,1
4,1,0,1,2,0,1
4,0,1,1,3,2,0
2,1,0,0,0,0,0
2,0,1,1,1,0,1
3,1,1,0,0,1,0
0,0,1,0,0,2,1
1,1,0,1,2,1,1
3,1,1,1,0,1,0
4,0,1,1,3,2,0

Program:

In [6]:

```
import pandas as pd
data=pd.read_csv("heart_disease.csv")
heart_disease=pd.DataFrame(data)
print(heart_disease)

from pgmpy.models import BayesianNetwork
model=BayesianModel([
        ('age','Lifestyle'),
        ('Gender','Lifestyle'),
        ('Family','heartdisease'),
        ('diet','cholestrol'),
        ('Lifestyle','diet'),
        ('cholestrol','heartdisease'),
        ('diet','cholestrol')
    ])

from pgmpy.estimators import MaximumLikelihoodEstimator
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')
print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age':int(input('Enter age :')),
    'Gender':int(input('Enter Gender :')),
    'Family':int(input('Enter Family history :')),
    'diet':int(input('Enter diet :')),
    'Lifestyle':int(input('Enter Lifestyle :')),
    'cholestrol':int(input('Enter cholestrol :'))
    })

print(q)
```

OUTPUT:

```
    age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
0     0       0       1     1          3           0             1
1     0       1       1     1          3           0             1
2     1       0       0     0          2           1             1
3     4       0       1     1          3           2             0
4     3       1       1     0          0           2             0
5     2       0       1     1          1           0             1
6     4       0       1     0          2           0             1
7     0       0       1     1          3           0             1
8     3       1       1     0          0           2             0
9     1       1       0     0          0           2             1
10    4       1       0     1          2           0             1
11    4       0       1     1          3           2             0
12    2       1       0     0          0           0             0
13    2       0       1     1          1           0             1
14    3       1       1     0          0           1             0
15    0       0       1     0          0           2             1
16    1       1       0     1          2           1             1
17    3       1       1     1          0           1             0
18    4       0       1     1          3           2             0
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }
For Gender Enter { Male:0, Female:1 }
For Family History Enter { yes:1, No:0 }
For diet Enter { High:0, Medium:1 }
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
For cholesterol Enter { High:0, BorderLine:1, Normal:2 }
Enter age :1
Enter Gender :1
Enter Family history :0
Enter diet :1
Enter Lifestyle :0
Enter cholestrol :1
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.0000 |
+-----------------+---------------------+
| heartdisease(1) |              1.0000 |
+-----------------+---------------------+
```

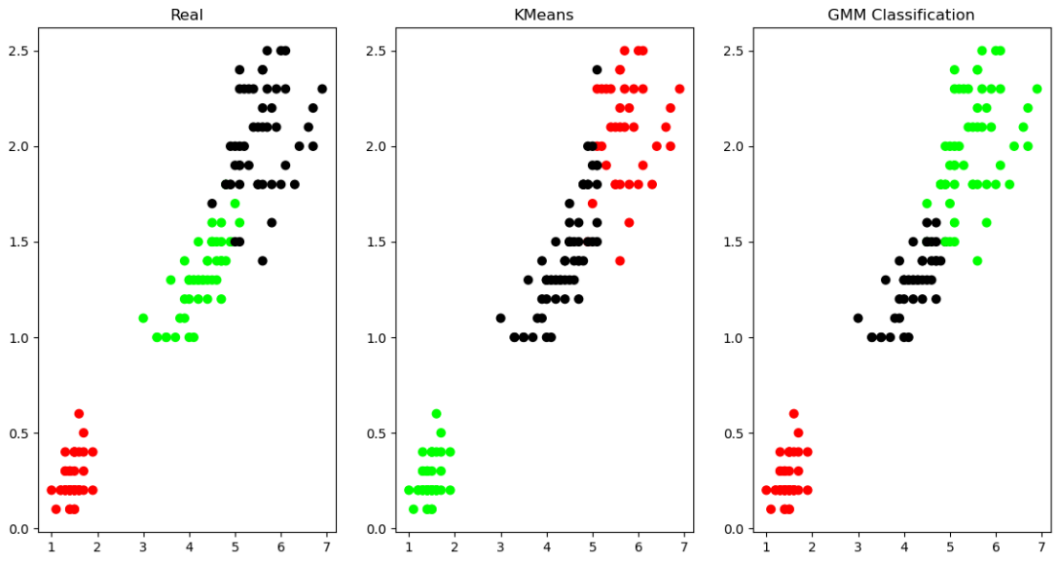
Experiment- 8 CLUSTERING BASED ON EXPECTATION MAXIMUM AND K-MEANS ALGORITHM

Program:

In [1]:
```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset=load_iris()
# print(dataset)
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
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=40)
plt.title('KMeans')

# GMM PLOT
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=40)
plt.title('GMM Classification')
```

OUTPUT:

Out[1]: Text(0.5, 1.0, 'GMM Classification')



Experiment-9 KNN ALGORITHM

Program:

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
dataset=load_iris()
#print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)
kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=1, p=2,
           weights='uniform')
for i in range(len(X_test)):
    x=X_test[i]
    x_new=np.array([x])
    prediction=kn.predict(x_new)
    print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["target_names"][prediction])
print(kn.score(X_test,y_test))
```

Output:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

Experiment 10 LOCALLY WEIGHTED REGRESSION ALGORITHM

Program:

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)

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

Output:

Out[3]: [<matplotlib.lines.Line2D at 0x1e7a9d2e580>]