In [11]:

```
import csv
import pandas as pd
import numpy as np
```

In [20]:

```
data = pd.read_csv('Pgm 1 Tennis Dataset.csv')

a = np.array(data.iloc[:,1:])
print(a)
```

```
[['Sunny' 'Hot' 'High' 'Weak' 'No']
 ['Sunny' 'Hot' 'High' 'Strong' 'No']
 ['Overcast' 'Hot' 'High' 'Weak' 'Yes']
 ['Rain' 'Mild' 'High' 'Weak' 'Yes']
 ['Rain' 'Cool' 'Normal' 'Weak' 'Yes']
 ['Rain' 'Cool' 'Normal' 'Strong' 'No']
 ['Overcast' 'Cool' 'Normal' 'Strong' 'Yes']
 ['Sunny' 'Mild' 'High' 'Weak' 'No']
 ['Sunny' 'Cool' 'Normal' 'Weak' 'Yes']
 ['Rain' 'Mild' 'Normal' 'Weak' 'Yes']
 ['Sunny' 'Mild' 'Normal' 'Strong' 'Yes']
 ['Overcast' 'Mild' 'High' 'Strong' 'Yes']
 ['Overcast' 'Hot' 'Normal' 'Weak' 'Yes']
 ['Rain' 'Mild' 'High' 'Strong' 'No']]
```

In [18]:

```
numAttr = len(a[0]) - 1
hypothesis = ['0'] * numAttr
print("Initial Hypo : ", hypothesis)
```

```
Initial Hypo :  ['0', '0', '0', '0', '0', '0']
```

In [19]:

```
for i in range(len(a)):
    if a[i][numAttr] == 'Yes':
        for j in range(0, numAttr):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'

print("Final Hypo : ", hypothesis)
```

```
Final Hypo :  ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

In [3]:

```
import csv
import pandas as pd
import numpy as np

data=pd.read_csv("Pgm 2 Tennis.csv")
concepts = np.array(data.iloc[:,:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
```

```
[[' Sunny' 'Warm' 'Normal' 'Strong' ' Warm' 'Same']
 [' Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 [' Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 [' Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

In [7]:

```
def learn(concepts, target):
    spec = concepts[0].copy()
    gen = [["?" for i in range(len(spec))] for i in range(len(spec))]

    for i, h in enumerate(concepts):
        if target[i]=='Yes':
            for x in range(len(h)):
                if h[x] != spec[x]:
                    spec[x] = '?'
                    gen[x][x] = '?'

        else:
            for x in range(len(h)):
                if h[x] != spec[x]:
                    gen[x][x] = spec[x]
                else:
                    gen[x][x] = '?'

    indices = [
        i for i, val in enumerate(gen) if val == ['?'] * len(spec)
    ]
    for i in indices:
        gen.remove(['?'] * len(spec))

    return spec, gen
```

In [8]:

```
spec, gen = learn(concepts, target)
print("most specific : ", spec)
print("most general : ", gen)
```

```
most specific :  [' Sunny' 'Warm' '?' 'Strong' '?' '?']
most general :  [[' Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?',
'?', '?', '?']]
```

In [ ]:

In [86]:

```
import numpy as np
import math
import pandas as pd
```

In [87]:

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

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

In [88]:

```
def subtables(data, col):
    dict = {}
    items = np.unique(data[:, col])

    count = np.zeros((items.shape[0], 1), dtype=int)

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

    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
        # remove column
        dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict
```

In [89]:

```
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)

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

    return sums
```

In [90]:

```
def gainRatio(data, col):
    items, dict = subtables(data, col)

    totalSize = data.shape[0]

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

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/totalSize
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])

    totalEntropy = entropy(data[:,-1])

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

    return totalEntropy
```

In [91]:

```
def createNode(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] = gainRatio(data, col)

    split = np.argmax(gains)
    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)
    items, dict = subtables(data, split)

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

    return node
```

In [92]:

```
def readData(filename):
    data = pd.read_csv(filename, header=None)
    metadata = np.array(data.iloc[0, :])
    traindata = np.array(data.iloc[1:, :])

    return (metadata, traindata)
```

In [93]:

```
def empty(size):
    s = ''
    for x in range(size):
        s += "    "
    return s
```

In [94]:

```
def printTree(node, level):
    if node.answer!= "":
        print(empty(level), node.answer)

    print(empty(level), node.attribute)

    for value, n in node.children:
        print(empty(level+1), value)
        printTree(n, level + 2)
```

In [95]:

```
metadata, traindata  = readData("Pgm 3 TennisDT.csv")

node = createNode(traindata, metadata)

printTree(node, 0)
```

```
Outlook
    Overcast
        b'Yes'

    Rain
        Wind
            b'Strong'
                b'No'

            b'Weak'
                b'Yes'

    Sunny
        Humidity
            b'High'
                b'No'

            b'Normal'
                b'Yes'
```

In [133]:

```
import csv
import random
import math
import operator
```

In [134]:

```
def safeDiv(x, y):
    if y==0:
        return 0
    return x/y
```

In [135]:

```
def loadCSV(filename):
    lines = csv.reader(open(filename))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset
```

In [136]:

```
def stdev(numbers):
    avg = mean(numbers)
    variance = safeDiv(
        sum([pow(x-avg, 2) for x in numbers]), float(len(numbers) - 1)
    )
    return math.sqrt(variance)
```

In [137]:

```
def splitDataset(dataset, ratio):
    trainSize = int(len(dataset) * ratio)
    trainSet = []
    copy = dataset
    while len(trainSet) < trainSize:
        trainSet.append(copy.pop(0))
    return trainSet, copy
```

In [138]:

```
def mean(numbers):
    return safeDiv(sum(numbers), float(len(numbers)))
```

In [139]:

```
def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries
```

In [140]:

```
def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if vector[-1] not in separated:
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated
```

In [141]:

```
def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries
```

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$ Mean

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu)^2\right]^{0.5}$$ Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$ Normal distribution

In [1]:

```
def calculateProbability(x, mean, stdev):
    exponent = math.exp(-safeDiv(math.pow(x-mean,2), (2 * math.pow(stdev, 2))))
    final = safeDiv(1, (math.sqrt(2 * math.pi) * stdev)) * exponent
    return final
```

In [143]:

```
def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
    return probabilities
```

In [144]:

```
def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel
```

In [145]:

```
def getPredictions(summaries, test):
    predictions = []
    for i in range(len(test)):
        result = predict(summaries, test[i])
        predictions.append(result)
    return predictions
```

In [146]:

```
def getAccuracy(test, predictions):
    correct = 0
    for i in range(len(test)):
        if test[i][-1] == predictions[i]:
            correct = 1
    accuracy = safeDiv(correct, float(len(test))) * 100.0
    return accuracy
```

In [148]:

```
def main():
    dataset = loadCSV('Pgm 5 ConceptLearning.csv')
    #print(data)
    splitRatio = 0.99
    train, test = splitDataset(dataset, splitRatio)

    summaries = summarizeByClass(train)
    predictions = getPredictions(summaries, test)
    actual = []

    for i in range(len(test)):
        vector = test[i]
        actual.append(vector[-1])

    print("Actual : ", actual)
    print('Predicted : ',predictions)
    accuracy = getAccuracy(test, predictions)
    print("Accuracy : ", accuracy)

main()
```

```
Actual :  [5.0]
Predicted :  [5.0]
Accuracy :  100.0
```

6.

Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

In [2]:

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [6]:

```
play_tennis = pd.read_csv("Pgm 6 PlayTennis.csv")
play_tennis.head(2)
```

Out[6]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No

In [9]:

```
number = LabelEncoder()
play_tennis['Outlook'] = number.fit_transform(play_tennis['Outlook'])
play_tennis['Temperature'] = number.fit_transform(play_tennis['Temperature'])
play_tennis['Humidity'] = number.fit_transform(play_tennis['Humidity'])
play_tennis['Wind'] = number.fit_transform(play_tennis['Wind'])
play_tennis['Play Tennis'] = number.fit_transform(play_tennis['Play Tennis'])

play_tennis.head()
```

Out[9]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1

In [11]:

```
features = ["Outlook", "Temperature" , "Humidity", "Wind"]
target = "Play Tennis"

XTrain, XTest, YTrain, YTest = train_test_split(play_tennis[features], play_tennis[target],
```

In [14]:

```
model = GaussianNB()
model.fit(XTrain, YTrain)

pred = model.predict(XTest)
accuracy = accuracy_score(YTest, pred)
print("Accuracy : ", accuracy)
```

```
Accuracy :  0.8
```

In [20]:

```
print(model.predict([[2,1,0,0]]))
```

```
[0]
```

In [ ]:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 27 22:26:49 2021

@author: Harsh
"""

import numpy as np
import csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('Pgm 7 heart.csv')

del heartDisease['oldpeak']
del heartDisease['slope']
del heartDisease['ca']
del heartDisease['thal']

heartDisease = heartDisease.replace('?',np.nan)

print(heartDisease.head())
model = BayesianModel([('age', 'heartdisease'), ('gender', 'heartdisease'),
                       ('exang', 'heartdisease'),('trestbps','heartdisease'),('fbs','heartd
                      ('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease'

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('gender', 'trestbps'), ('gende
                       ('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisea
                      ('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease'


model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)


print("Inferencing with Bayesian Network")

HeartDisease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q)

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 100})
print(q)
```

Program 8

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Harsh R

In [1]:

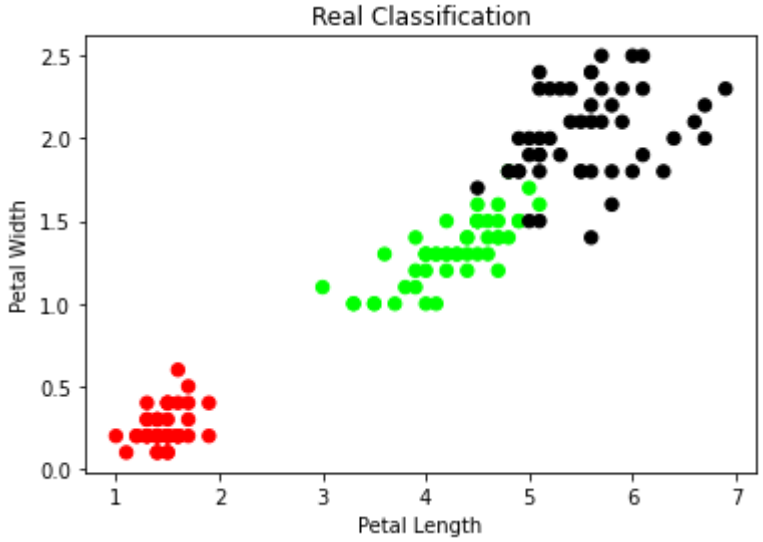
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
```

In [2]:

```
iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
#print(X.shape)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']
```

In [3]:

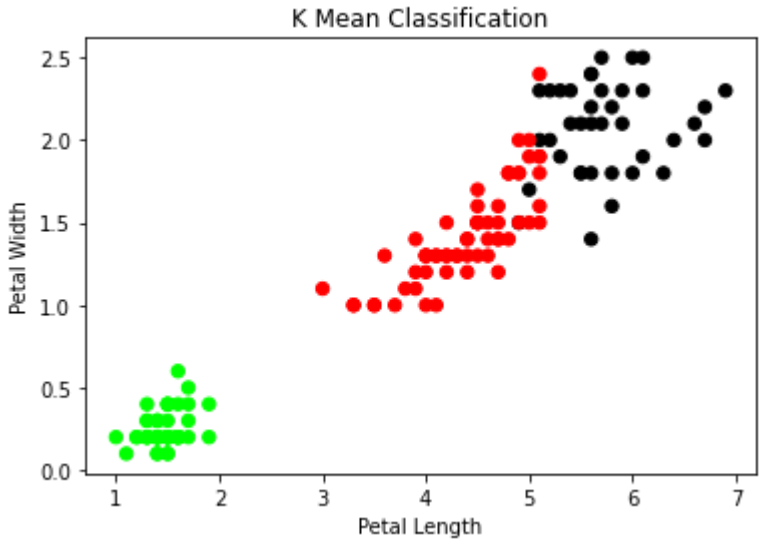
```
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.figure()
#plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```



In [4]:

```
model = KMeans(n_clusters=3)
model.fit(X)

plt.figure()
#plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))
plt.show()
```

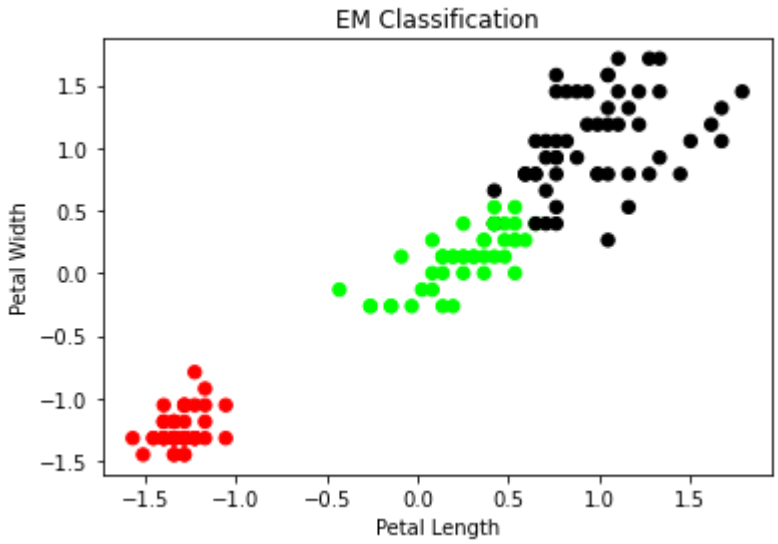
```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
```



In [5]:

```
from sklearn import preprocessing
scalar = preprocessing.StandardScaler()
scalar.fit(X)
xsa = scalar.transform(X)

xs=pd.DataFrame(xsa, columns=X.columns)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

ypred = gmm.predict(xs)

plt.figure()
plt.scatter(xs.Petal_Length, xs.Petal_Width, c=colormap[ypred], s=40)
plt.title('EM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of EM : ',sm.accuracy_score(y, ypred))
print('The Confusion matrixof EM : ',sm.confusion_matrix(y, ypred))
plt.show()
```

```
The accuracy score of EM :  0.9666666666666667
The Confusion matrixof EM :  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```



Program 9

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Harsh R

In [18]:

```
import sklearn.datasets as dt
import pandas as pd
from sklearn.metrics import accuracy_score
```

In [19]:

```
iris=dt.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width']
y = pd.DataFrame(iris.target)
y.columns = ['target']
```

In [ ]:

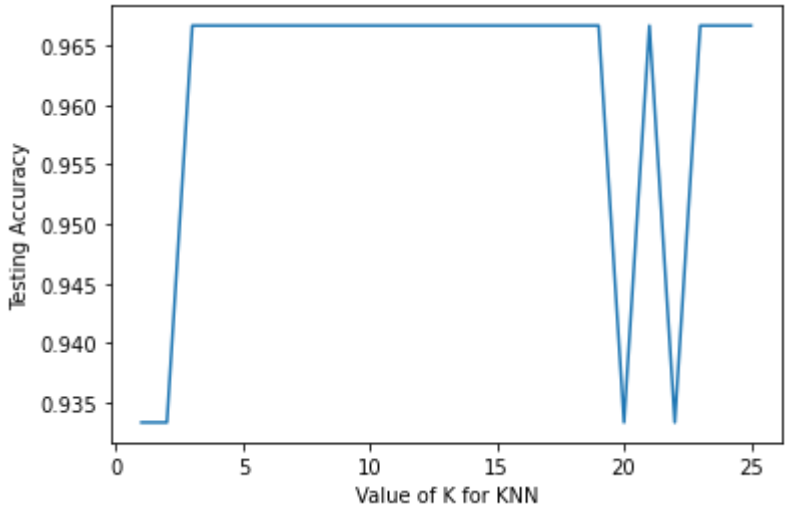
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

xtrain, xtest, ytrain, ytest = train_test_split(X,y,test_size=0.2,random_state=4)

KRange = range(1,26)
scores = {}

for k in KRange:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(xtrain, ytrain)
    ypred = model.predict(xtest)
    scores[k] = accuracy_score(ytest, ypred)
```

In [21]:

```
import matplotlib.pyplot as plt

#plot the relationship between K and the testing accuracy
plt.plot(KRange,[i for key,i in scores.items()])
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Out[21]:

```
Text(0, 0.5, 'Testing Accuracy')
```



In [24]:

```
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X,y)
```

```
C:\Users\Harsh\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
```

Out[24]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [26]:

```
xnew = [[1,2,3,4],[5,4,2,2]]

ypred = model.predict(xnew)

classes = {0:'setosa',1:'versicolor',2:'virginica'}
print(classes[ypred[0]])
print(classes[ypred[1]])
```

```
versicolor
setosa
```

Progarm 10

Implement the non-parametric Locally Weighted Regressionalgorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Harsh R

In [11]:

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

def lowess(x,y,f,iter):
    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

    ypred = np.zeros(n)
    delta = np.ones(n)

    for iteration in range(iter):
        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)
            ypred[i] = beta[0] + beta[1] * x[i]

        res = y - ypred
        s = np.median(np.abs(res))
        delta = np.clip(res / (6.0 * s), -1, 1)
        delta = (1-delta**2)**2

    return ypred
```

In [12]:

```
if __name__ == '__main__':
    n=100
    f=0.25
    x=np.linspace(0, 2*math.pi, n)
    y=np.sin(x) + 0.3*np.random.randn(n)

    ypred = lowess(x,y,f,3)

    import pylab as p1
    p1.clf()
    p1.plot(x,y, label="NOiSY")
    p1.plot(x,ypred, label="Predicted")
    p1.legend()
    p1.show()
```