

USN: 1BM20CS119

MACHINE LEARNING LAB OBSERVATION

IRIS DATASET:

- Features in the Iris dataset:
 1. sepal length in cm
 2. sepal width in cm
 3. petal length in cm
 4. petal width in cm
- Target classes to predict:
 1. Iris Setosa
 2. Iris Versicolour
 3. Iris Virginica

```
In [8]: from sklearn.datasets import load_iris  
iris=load_iris()
```

```
In [9]: print(iris)
```

```
{'data': array([[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],  
               [5.4, 3.8, 1.5, 0.3],  
               [5.2, 3.5, 1.5, 0.2],  
               [5. , 3.4, 1.4, 0.3],  
               [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]]),
```

```
In [5]: type(iris)
```

```
Out[5]: function
```

```
In [12]: iris.keys()
```

```
Out[12]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
In [13]: iris
```

```
array([[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]])
```

```
In [17]: print(iris['target_names'])

['setosa' 'versicolor' 'virginica']
```

```
In [20]: n_samples,n_features=iris.data.shape
print("no.of samples:",n_samples)
print("no.of features:",n_features)

no.of samples: 150
no.of features: 4
```

```
In [28]: iris.data[[12,26,89,114]]
```

```
Out[28]: array([[4.8, 3. , 1.4, 0.1],
               [5. , 3.4, 1.6, 0.4],
               [5.5, 2.5, 4. , 1.3],
               [5.8, 2.8, 5.1, 2.4]])
```

```
In [29]: print(iris.data.shape)

(150, 4)
```

```
In [31]: print(iris.target.shape)

(150,)
```

```
In [32]: import numpy as np
np.bincount(iris.target)
```

...

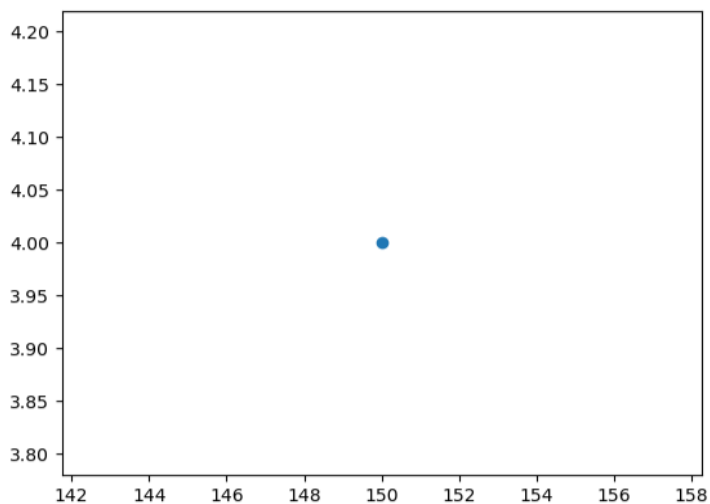
Scattered graph for samples vs features.

```
In [32]: import numpy as np
np.bincount(iris.target)
```

```
Out[32]: array([50, 50, 50], dtype=int64)
```

```
In [42]: import matplotlib.pyplot as plt
plt.scatter(n_samples,n_features)
```

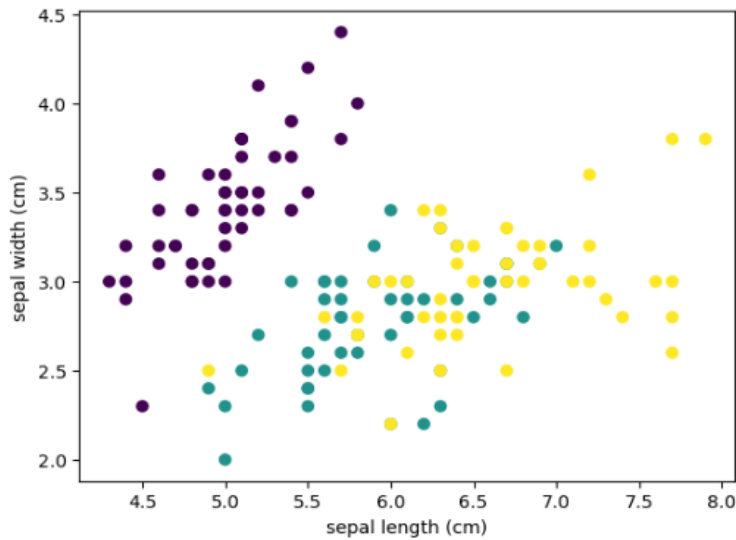
```
Out[42]: <matplotlib.collections.PathCollection at 0x1d1c8c45550>
```



Scattered graph: with first two features(sepal width vs sepal length)
The three colors represents three different classes respectively.

In [47]:

```
features = iris.data.T
plt.scatter(features[0], features[1],
            c=iris.target)
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1]);
```



In [49]: iris.data[[1,2,3,4,5]]

```
Out[49]: array([[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]])
```

WINE DATASET:

In [51]: `from sklearn.datasets import load_wine`
`wine=load_wine()`

In [52]: `print(wine)`

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
                1.065e+03],
                [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
                1.050e+03],
                [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
```

In [57]: `wine.data`

```
Out[57]: array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
                1.065e+03],
                [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
                1.050e+03],
                [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
                1.185e+03],
                ...,
                [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
                8.350e+02],
                [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
                8.400e+02],
                [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
                5.600e+02]])
```

In [58]: `wine.keys()`

```
Out[58]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

In [60]: `print(wine['target_names'])`

```
['class_0' 'class_1' 'class_2']
```

```
In [9]: print(wine['feature_names'])
```

```
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
```

```
In [11]: import numpy as np  
np.bincount(wine.target)
```

```
Out[11]: array([59, 71, 48], dtype=int64)
```

Date: 05/04/2023

Lab 2: FIND-S ALGORITHM FOR ENJOY SPORT:

Program 2 – Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file Data set:Enjoysport

a. Enjoysport

Example	Sky	AirTemp	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

Algorithm:

initialize h to the most specific hypothesis in H $h = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

1. First training example $X_1 = \langle \text{Sunny, Warm, Normal, Strong Warm Same} \rangle$. EnjoySport=+ve Observing. The first training example, it is clear that hypothesis h is too specific. None of the " \emptyset " constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example $h_1 = \langle \text{Sunny, Warm, Normal, Strong Warm, Same} \rangle$.

2. Consider the second training example $x_2 = \langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle$. EnjoySport+ve. The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example. Now $h_2 = \langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$

3. Consider the third training example $x_3 = \langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle$ EnjoySport ve. The FIND-S algorithm simply ignores every negative example. So the hypothesis remain as before, so $h_3 = \langle \text{Sunny, Warm, ?, Strong, Warm, Same} \rangle$

4. Consider the fourth training example $x_4 = \langle \text{Sunny, Warm, High, Strong, Cool, Change, EnjoySport +ve} \rangle$. The fourth example leads to a further generalization of h as $h_4 = \langle \text{Sunny, Warm, ?, Strong, ?, ?} \rangle$

5. So the final hypothesis is $\langle \text{Sunny, Warm, ?, Strong, ?, ?} \rangle$

```
import csv
```

```
def update_hypothesis(x, h):  
    if x[h] == [ ]:  
        return x
```

```
    for i in range(0, len(h))
```

```
        if (x[i].upper != h[i].upper):  
            h[i] = 'i'
```

```
    return h
```

```
if __name__ == "__main__":
```

```
    data = [ ]
```

```
    h = [ ]
```

```
    with open('NS.csv', 'a') as file:
```

```
        reader = csv.reader(file)
```

```
        print("DATA :")
```

```
        for row in reader:
```

```
            data.append(row)
```

```
            print(row)
```


if data:

for x in data:

if x[-1].upper() == "YES"

x.pop()

h = updateHypothesis(x, h)

print ("\n Hypothesis :", h)

Output:

DATA:

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

Hypothesis:

['Sunny', 'Warm', '?', 'Strong', '?', '?']

Sunny
05/04/2023



enjoysport.csv



File Edit View Insert Format Data Tools Extensions Help



100%



123

Default...



10



B

I



A



A1 Sky

	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							
7							

```
import csv
def updateHypothesis(x,h):
    if h == []:
        return x
    for i in range(0,len(h)):
        if x[i].upper() != h[i].upper():
            h[i] = '?'
    return h

if __name__=="__main__":
    data=[]
    h=[]
    with open('/kaggle/input/findsdata/ws.csv','r') as file:
        reader = csv.reader(file)
        print("DATA:")
        for row in reader:
            data.append(row)
            print(row)
    if data:
        for x in data:
            if x[-1].upper() == "YES":
                x.pop()
            # print(x)
            h=updateHypothesis(x,h)
    print("\n HYPOTHESIS :",h)
```

DATA:

```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change']
```

HYPOTHESIS : ['Sunny', 'Warm', '?', 'Strong', '?', '?']

DATE: 12/04/2023

LAB 3: CANDIDATE- ELIMINATION- ENJOY SPORT

Program 3: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
Data set: Enjoysport

Example	Sky	AirTemp	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

12/04/23

For a given set of training data examples stored in a .CSV file, implement and demonstrate the candidate elimination algorithm.

Candidate elimination algorithm

Initialize G to the set of maximally general hypothesis H
Initialize S to the set of maximally specific hypothesis H

For each training example $d \in D$

- if d is positive example
 - remove from G any hypothesis inconsistent with d
 - for each hypothesis s in S that is not consistent with d
 - remove s from S
 - Add to S all minimal generalization h of s such that
 - h is consistent with d and some member of G is more general than h
 - Remove from S any hypothesis that is more general than another hypothesis in S
- if d is negative example
 - remove from S any hypothesis inconsistent with d
 - for each hypothesis g in G that is not consistent with d
 - remove g from G

add to G all minimal specializations h of g such that

- h is consistent with d some member of S is more specific than h
- remove from G any hypothesis that is less general than another hypothesis in G

Output:-

Final Specific h :

['Sunny', 'Warm', '?', '?', 'Strong', '?', '? ']

Final General h :

['Sunny', '?', '?', '?', '?', '?', '? '] ['?', 'Warm', '?', '?', '?', '?', '? ']

~~Final hypothesis~~

CREATING CSV FILE:

enjoysport.csv ☆ 📁 ☁

File Edit View Insert Format Data Tools Extensions Help

100% | \$ % .0 .00 123 | Default... | - 10 + | B I A

A1 | fx Sky

	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							
7							

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('/kaggle/input/candidate-algo/enjoysports.csv'))
concepts = np.array(data.iloc[:,0:-1])
# print(concepts)
target = np.array(data.iloc[:,-1])
# print(target)
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
```

```

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")

```

Final Specific_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:

['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']

Program 4: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ALGORITHM:

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples
- Otherwise Begin
 - $A \leftarrow$ the attribute from Attributes that best* classifies Examples
 - The decision attribute for Root $\leftarrow A$
 - For each possible value, v_i , of A,
 - Add a new tree branch below Root, corresponding to the test $A = v_i$
 - Let $Examples_{v_i}$ be the subset of Examples that have value v_i for A
 - If $Examples_{v_i}$ is empty
 - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
 - Else below this new branch add the subtree $ID3(Examples_{v_i}, Target_attribute, Attributes - \{A\})$
 - End
 - Return Root

03/02/2020

Date _____
Page _____

ID3 Algorithm

Problem: To demonstrate the working of the decision tree based on ID3 algorithm

Step 1: Create a root node for the tree

Step 2: If all examples are positive then return tree root with label = +

Step 3: If all examples are negative then return single node tree root with label = -

Step 4: If the attribute is empty, return the single node tree root = most common value of target attribute in examples

Step 5: Otherwise begin

- A ← attribute from the attributes that best classifies examples
- The decision attribute for root ← A
- For each possible value v_i of A
 - * Add a new tree branch below root, corresponding to the test $A = v_i$
 - * Let $subexamples_i$ be the subset of examples that have value v_i for A
 - * If $subexamples_i$ is empty
 - ↳ then below this branch add a leaf node with label = most common value of target attribute in examples

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

data = pd.read_csv("/kaggle/input/id3hhh/id3.csv")
features = [feat for feat in data]
features.remove("Answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isleaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["Answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
```

```
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print("\n", uniq)
    gain = entropy(examples)
    #print("\n", gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print("\n", subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    #print("\n", gain)
    return gain

def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print("\n", examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print("\nMax feature attr", max_feat)
    uniq = np.unique(examples[max_feat])
    #print("\n", uniq)
    for u in uniq:
        #print("\n", u)
        subdata = examples[examples[max_feat] == u]
        #print("\n", subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isleaf = True
```

Decision Tree is:

```
Outlook
  overcast -> ['yes']
  rain
    Wind
      strong -> ['no']
      weak -> ['yes']
  sunny
    Humidity
      high -> ['no']
      normal -> ['yes']
```

Predicted Label for new example {'Outlook': 'sunny', 'Temperature': 'hot', 'Humidity': 'normal', 'Wind': 'strong'} is: ['yes']

PROGRAM 5: Simple linear regression program

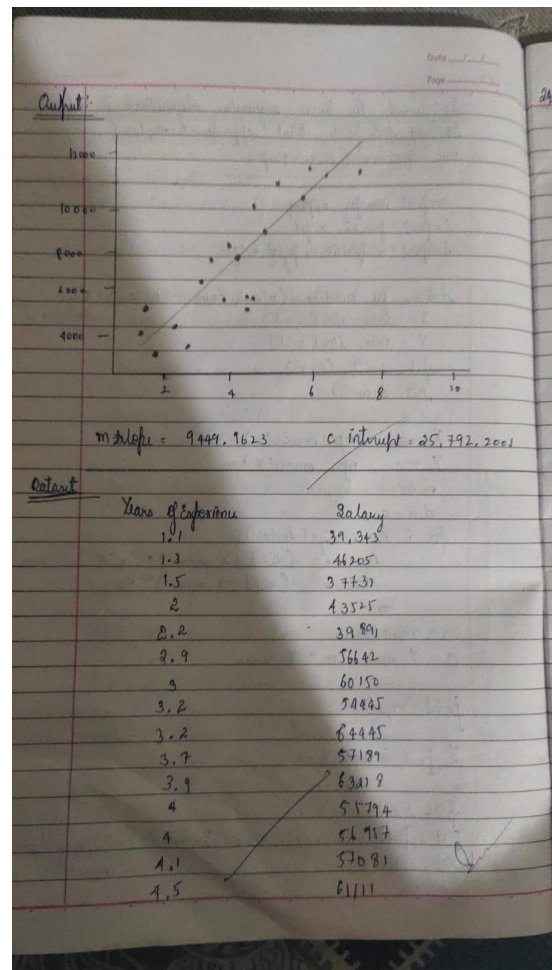
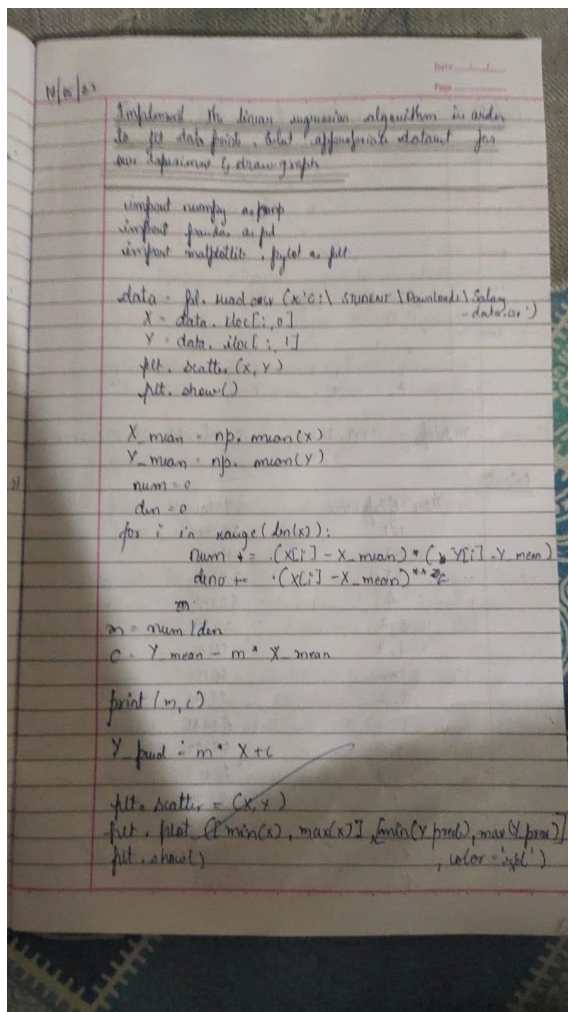
dt: 17/05/2023

Dataset used:

YearsExperience	Salary
1.1	39343.00
1.3	46205.00
1.5	37731.00
2.0	43525.00
2.2	39891.00
2.9	56642.00
3.0	60150.00
3.2	54445.00
3.2	64445.00
3.7	57189.00

ALGORITHM:

- The main function to calculate values of coefficients
- Initialize the parameters.
- Predict the value of a dependent variable by giving an independent variable.
- Calculate the error in prediction for all data points.
- Calculate partial derivatives w.r.t a_0 and a_1 .
- Calculate the cost for each number and add them.
- Update the values of a_0 and a_1 .



```
# Making imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)
```

```
data = pd.read_csv(r'C:\Users\STUDENT\Downloads\Salary_Data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
```

```
# Building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)

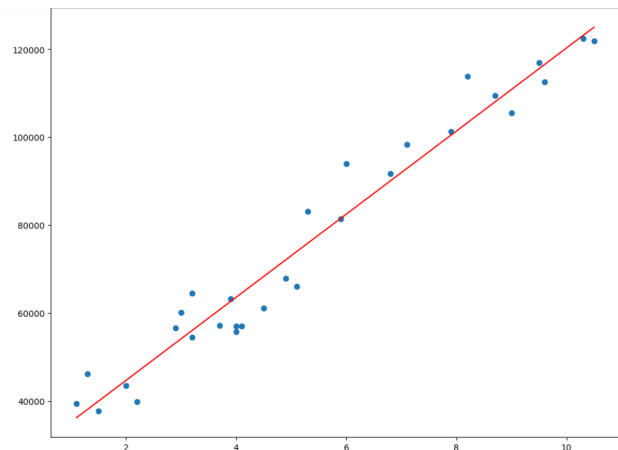
num = 0
den = 0
for i in range(len(X)):
    num += (X[i] - X_mean) * (Y[i] - Y_mean)
    den += (X[i] - X_mean)**2
m = num / den
c = Y_mean - m * X_mean

print(m, c)
```

449.962321455077 25792.20019866869

```
# Making predictions
Y_pred = m * X + c

plt.scatter(X, Y) # actual
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red') # predicted
plt.show()
```



Dt:17-05-4043

Program 6:Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

Data set used:

Color	Type	Origin	Stolen
Red	Sports	Domestic	Yes
Red	Sports	Domestic	No
Red	Sports	Domestic	Yes
Yellow	Sports	Domestic	No
Yellow	Sports	Imported	Yes
Yellow	SUV	Imported	No
Yellow	SUV	Imported	Yes
Yellow	SUV	Domestic	No

Algorithm:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Formula for naive bayes classifier is as follows →

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.
4. Test accuracy of the result and visualizing the test set result.

```

import numpy as np
import math
import csv
import pdb
def read_data(filename):

    with open(filename,'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    testset = list(dataset)
    i=0
    while len(trainSet) < trainSize:
        trainSet.append(testset.pop(i))
    return [trainSet, testset]

def classify(data,test):

    total_size = data.shape[0]
    print("\n")
    print("training data size=",total_size)
    print("test data size=",test.shape[0])

```

```

countYes = 0
countNo = 0
probYes = 0
probNo = 0
print("\n")
print("target    count    probability")

for x in range(data.shape[0]):
    if data[x,data.shape[1]-1] == 'Yes':
        countYes +=1
    if data[x,data.shape[1]-1] == 'No':
        countNo +=1

probYes=countYes/total_size
probNo= countNo / total_size

print('Yes',"\t",countYes,"\t",probYes)
print('No',"\t",countNo,"\t",probNo)

prob0 =np.zeros((test.shape[1]-1))
prob1 =np.zeros((test.shape[1]-1))
accuracy=0
print("\n")
print("instance prediction  target")

for t in range(test.shape[0]):
    for k in range (test.shape[1]-1):
        count1=count0=0
        for j in range (data.shape[0]):
            #how many times appeared with no
            if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='No':
                count0+=1
            #how many times appeared with yes
            if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='Yes':

```

```

    probno=probNo
    probyes=probYes
    for i in range(test.shape[1]-1):
        probno=probno*prob0[i]
        probyes=probyes*prob1[i]
    if probno>probyes:
        predict='No'
    else:
        predict='Yes'

    print(t+1,"\t",predict,"\t\t",test[t,test.shape[1]-1])
    if predict == test[t,test.shape[1]-1]:
        accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return

metadata,traindata= read_data("naive.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
print("\n The Training data set are:")
for x in trainingset:
    print(x)

testing=np.array(testset)
print("\n The Test data set are:")
for x in testing:
    print(x)
classify(training,testing)

```

The Training data set are:

```

['Red', 'Sports', 'Domestic', 'Yes']
['Red', 'Sports', 'Domestic', 'No']
['Red', 'Sports', 'Domestic', 'Yes']
['Yellow', 'Sports', 'Domestic', 'No']
['Yellow', 'Sports', 'Imported', 'Yes']
['Yellow', 'SUV', 'Imported', 'No']

```

The Test data set are:

```

['Yellow' 'SUV' 'Imported' 'Yes']
['Yellow' 'SUV' 'Domestic' 'No']
['Red' 'SUV' 'Imported' 'No']
['Red' 'Sports' 'Imported' 'Yes']

```

```

training data size= 6
test data size= 4

```

target	count	probability
Yes	3	0.5
No	3	0.5

instance	prediction	target
1	No	Yes
2	No	No
3	No	No
4	Yes	Yes

Program 7:K- means clustering

Algorithm:

Initialize k means with random values

For a given number of iterations:

Iterate through items:

Find the mean closest to the item by calculating the euclidean distance of the item with each of the means

Assign item to mean

Update mean by shifting it to the average of the items in that cluster

Dataset:

X	Y
0.4967141530112327	-0.13826430117118466
0.6476885381006925	1.5230298564080254
-0.23415337472333597	-0.23413695694918055
1.5792128155073915	0.7674347291529088
-0.4694743859349521	0.5425600435859647
-0.46341769281246226	-0.46572975357025687
0.24196227156603412	-1.913280244657798
-1.7249178325130328	-0.5622875292409727
-1.0128311203344238	0.3142473325952739

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def kmeans(X, K, max_iters=100):
    # Randomly initialize centroids
    centroids = X[np.random.choice(range(len(X)), size=K, replace=False)]

    for _ in range(max_iters):
        # Assign each data point to the nearest centroid
        clusters = [[] for _ in range(K)]
        for x in X:
            distances = [np.linalg.norm(x - centroid) for centroid in centroids]
            cluster_index = np.argmin(distances)
            clusters[cluster_index].append(x)

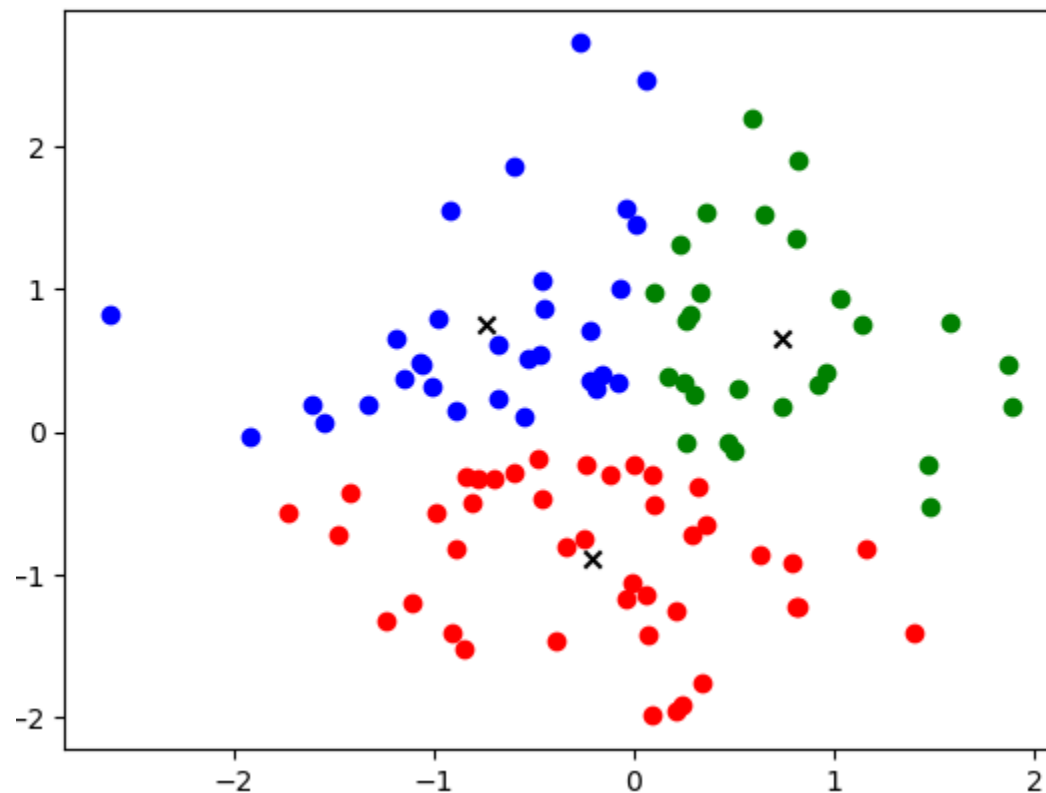
        # Update centroids
        new_centroids = []
        for cluster in clusters:
            if cluster:
                new_centroids.append(np.mean(cluster, axis=0))
            else:
                # If a centroid has no assigned points, keep the previous centroid value
                new_centroids.append(centroids[clusters.index(cluster)])

        # Check for convergence
        if np.allclose(centroids, new_centroids):
            break

        centroids = new_centroids

    return centroids, clusters

```



7/06/23

Aim: Apply K-Means to cluster the data stored in CSV file

Code:

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

def kmeans(X, K, max_iterations=100):
    centroids = X[np.random.choice(X.shape[0], size=K, replace=False)]
    for _ in range(max_iterations):
        clusters = [[] for _ in range(K)]
        for x in X:
            distances = [np.linalg.norm(x - centroid) for centroid in centroids]
            cluster_index = np.argmin(distances)
            clusters[cluster_index].append(x)
        new_centroids = []
        for cluster in clusters:
            if cluster:
                new_centroids.append(np.mean(cluster, axis=0))
            else:
                new_centroids.append(centroids[cluster_index])
        centroids = new_centroids
    return clusters
```

If np.isnan(centroids, new_centroids):
break

centroids = new_centroids

plt.scatter(X, clusters)

data = pd.read_csv('C:\\Users\\STUDENT\\Downloads\\data.csv')

X = data.values

K = 3

centroids, clusters = kmeans(X, K)

centroids = np.array(centroids)

colors = ['r', 'g', 'b']

for i, cluster in enumerate(clusters):
for point in cluster:
plt.scatter(point[0], point[1], c=colors[i])

plt.scatter(centroids[i, 0], centroids[i, 1], c='k', marker='x')
plt.show()

Dt: 07/06/23

Program 8: KNN ALGORITHM

Dataset used: Iris dataset

Algorithm:

- Select the number K of the neighbor
- Calculate the Euclidean distance of K number of neighbors
- Take the K nearest neighbors as per the calculated Euclidean distance.
- Among these k neighbors, count the number of the data points in each category.
- Assign the new data points to that category for which the number of the neighbor is maximum.

```
import csv
import math

# Function to load the dataset from a CSV file
def load_dataset(filename):
    dataset = []
    with open(filename, 'r') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Function to convert string columns to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Function to calculate the Euclidean distance between two data points
def euclidean_distance(instance1, instance2, length):
    distance = 0
    for i in range(length):
        distance += pow((instance1[i] - instance2[i]), 2)
    return math.sqrt(distance)

# Function to find k nearest neighbors
def get_neighbors(train_set, test_instance, k):
    distances = []
    length = len(test_instance) - 1
    for train_instance in train_set:
        dist = euclidean_distance(test_instance, train_instance, length)
        distances.append((train_instance, dist))
```

```
# Function to predict the class label for a test instance
def predict_class(neighbors):
    class_votes = {}
    for neighbor in neighbors:
        class_label = neighbor[-1]
        if class_label in class_votes:
            class_votes[class_label] += 1
        else:
            class_votes[class_label] = 1
    sorted_votes = sorted(class_votes.items(), key=lambda x: x[1], reverse=True)
    return sorted_votes[0][0]

# Function to evaluate the accuracy of predictions
def get_accuracy(test_set, predictions):
    correct = 0
    for i in range(len(test_set)):
        if test_set[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test_set))) * 100.0

# Load the dataset
dataset = load_dataset('iris.csv')

# Convert string columns to float
for i in range(4):
    str_column_to_float(dataset, i)

# Split the dataset into training and testing sets
split_ratio = 0.7
train_size = int(len(dataset) * split_ratio)
train_set = dataset[:train_size]
test_set = dataset[train_size:]

# Make predictions
k = 3
```

```
Expected: setosa Predicted: setosa
Expected: setosa Predicted: setosa
Expected: virginica Predicted: virginica
Expected: virginica Predicted: virginica
Expected: virginica Predicted: virginica
Expected: setosa Predicted: setosa
Expected: versicolor Predicted: versicolor
Expected: versicolor Predicted: versicolor
Expected: setosa Predicted: setosa
Expected: setosa Predicted: setosa
Expected: versicolor Predicted: versicolor
Expected: versicolor Predicted: versicolor
Expected: versicolor Predicted: versicolor
Expected: versicolor Predicted: versicolor
Expected: versicolor Predicted: versicolor
Expected: virginica Predicted: virginica
Expected: virginica Predicted: virginica
Expected: virginica Predicted: virginica
Expected: setosa Predicted: setosa
Expected: virginica Predicted: virginica
Expected: setosa Predicted: setosa
Expected: versicolor Predicted: versicolor
Expected: versicolor Predicted: versicolor
Expected: virginica Predicted: virginica
Expected: virginica Predicted: virginica
Expected: setosa Predicted: setosa
Expected: setosa Predicted: setosa
Expected: versicolor Predicted: versicolor
Expected: versicolor Predicted: versicolor
Expected: setosa Predicted: setosa
Expected: virginica Predicted: virginica
```

Dt: 7/06/23

Program 9: Program: Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

Algorithm for k means clustering:

- Initialize k means with random values
- For a given number of iterations:
- Iterate through items:
- Find the mean closest to the item by calculating the euclidean distance of the item with each of the means
- Assign item to mean
- Update mean by shifting it to the average of the items in that clusters

Algorithm for EM algorithm:

- The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.
- This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.
- This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.
- The last step is to check if the values of latent variables are converging or not.

Dataset: Iris dataset

Program 10: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points.
Select the appropriate data set for your experiment and draw graphs.

Algorithm:

1. F is approximated near X_q using a linear function:

$$\hat{f}(x) = w_0 + \sum_{u=1}^k w_u K_u(d(x_u, x))$$

2. Minimize the squared error:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) (f(x) - \hat{f}(x)) a_j(x)$$

3. It is weighted because the contribution of each training example is weighted by its distance from the query point.

Dataset: tip.csv

```

def localWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

```

```

[ ] def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0)
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();

```

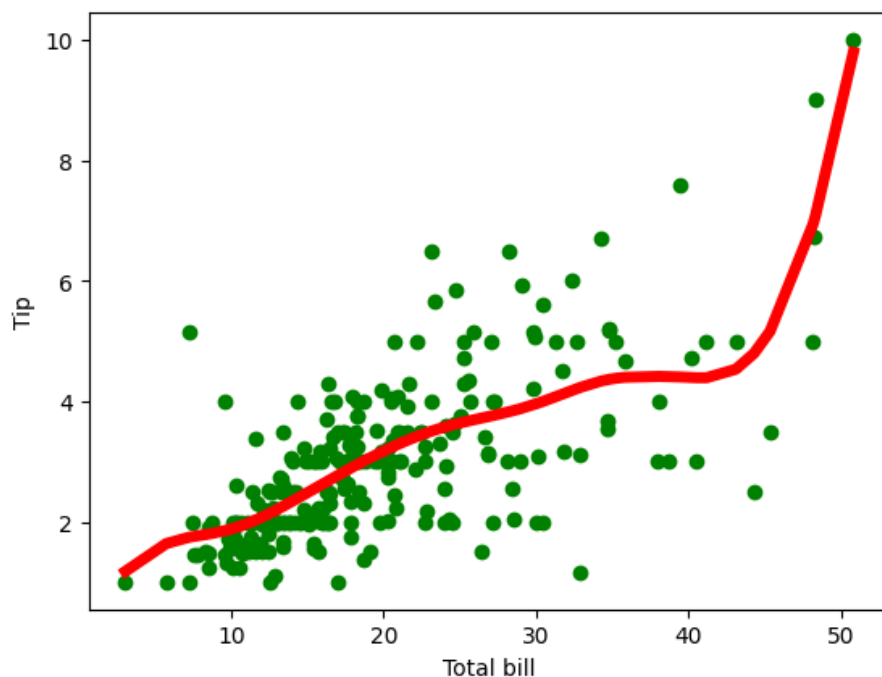
```

data = pd.read_csv('/content/tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

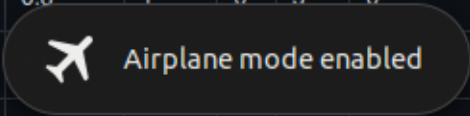
# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)

```



Lab 11: Write a program to construct a bayesian network considering training data .Use this model to make predictions

Dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0					
57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
63	1	4	130	254	0	2	147	0	1.4	2	1	7	2

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

heartDisease = pd.read_csv('heart_disease.csv')
heartDisease = heartDisease.replace('?', np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

print('\n Attributes and datatypes')
print(heartDisease.dtypes)

model= BayesianModel([('age', 'Heartdisease'), ('sex', 'Heartdisease'), ('exang', 'Heartdisease'), ('cp', 'Heartdisease')])
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['Heartdisease'], evidence={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['Heartdisease'], evidence={'cp':2})
print(q2)
```

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg

Heartdisease	phi(Heartdisease)
Heartdisease(0)	0.1012
Heartdisease(1)	0.0000
Heartdisease(2)	0.2392
Heartdisease(3)	0.2015
Heartdisease(4)	0.4581

2. Probability of HeartDisease given evidence= cp

Heartdisease	phi(Heartdisease)
Heartdisease(0)	0.3610
Heartdisease(1)	0.2159
Heartdisease(2)	0.1373
Heartdisease(3)	0.1537
Heartdisease(4)	0.1321
