PROGRAM 1:

NO DATASET USED

PIP command: pip install heuristicsearch

PROGRAM:

```
from heuristicsearch. a_star_search import AStar


adjacency_list = {
    'A': [('B',1), ('C',3), ('D',7)],
    'B': [('D',5)],
    'C': [('D',12)]
}


heuristics = {'A':1, 'B':1, 'C':1, 'D':1}


graph=AStar (adjacency_list, heuristics)
graph. apply_a_star (start='A', stop='D')
```

OUTPUT:

```
Path
A -> B -> D
Cost
0 -> 1 -> 6
```

PROGRAM 2:

NO DATASET USED

PIP command: pip install heuristicsearch

PROGRAM:

```
from heuristicsearch.ao_star import AOStar

print ("Graph - 1")
heuristic = {'A':1, 'B':6, 'C':12, 'D':10, 'E':4, 'F':4, 'G':5, 'H':7}

adjacency_list = {
    'A': [ [('B',1), ('C',1)], [('D',1)] ],
    'B': [ [('G',1)], [('H',1)] ],
    'D': [ [('E',1), ('F',1)]],
}

graph = AOStar (adjacency_list, heuristic, 'A')
graph. applyAOStar ()
```

OUTPUT

```
    Graph - 1
PROCESSING NODE : A
-----------------------------------------------------------------------------------
11 ['D']

PROCESSING NODE : D
-----------------------------------------------------------------------------------
10 ['E', 'F']

PROCESSING NODE : A
-----------------------------------------------------------------------------------
11 ['D']

PROCESSING NODE : E
-----------------------------------------------------------------------------------
0 []
```

```
PROCESSING NODE : D
-------------------------------------------------------------------------------------------
6 ['E', 'F']

PROCESSING NODE : A
-------------------------------------------------------------------------------------------
7 ['D']

PROCESSING NODE : F
-------------------------------------------------------------------------------------------
0 []

PROCESSING NODE : D
-------------------------------------------------------------------------------------------
2 ['E', 'F']

PROCESSING NODE : A
-------------------------------------------------------------------------------------------
3 ['D']

FOR THE SOLUTION, TRAVERSE THE GRAPH FROM THE START NOD
E: A
------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
------------------------------------------------------------
```

PROGRAM 3:

DATASET USED: **trainingexamples.csv**

NO USING PIP COMMAND

PROGRAM:

```
import csv
with open("trainingexamples.csv") as f:
    csv_file = csv. reader(f)
    data = list(csv_file)

    specific = data [0][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j]!=specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j]!=specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"
```

```
        print ("\nStep " + str (data.index(i)+1) + " of Candidate Elimination Algorithm")
        print(specific)
        print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
            if j!= '?':
                gh.append(i)
                break
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

OUTPUT

```
Step 1 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'
, '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'
, '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?
', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 4 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?
', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Final Specific hypothesis:
['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

PROGRAM 4:

DATASET USED: **P4_train.csv, P4_test.csv**

PIP COMMAND: pip install decision-tree-ID3-Algorithm

PROGRAM:

```
from decisiontree.ID3Algorithm import ID3
import csv

def load_csv(filename):
    lines=csv.reader(open(filename,"r"))
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

dataset_train, headers_train = load_csv("P4_train.csv")
dataset_test, headers_test = load_csv("P4_test.csv")

id3 = ID3(dataset_train,headers_train,dataset_test,headers_test)
id3.build_tree()
id3.classify()
```

OUTPUT

```
The decision tree for the dataset using ID3 algorithm is
Outlook
├─── overcast
│    └─── yes
├─── rain
│    └─── Wind
│         ├─── strong
│         │    └─── no
│         └─── weak
│              └─── yes
└─── sunny
     └─── Humidity
          ├─── high
          │    └─── no
          └─── normal
               └─── yes

The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:
no
The tree traversal for the test instance is:
Outlook
└─── rain
     └─── Wind
          └─── strong
               └─── no

The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:
yes
The tree traversal for the test instance is:
Outlook
└─── sunny
     └─── Humidity
          └─── normal
               └─── yes
```

PROGRAM 5:

NO DATASET USED

PIP command: pip install backpropagation

PROGRAM:

```
import numpy as np
from backpropagation.NeuralNetwork import NeuralNetwork
from backpropagation.Backpropagation import Backpropagation
from backpropagation.Sigmoid import Sigmoid

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

nn = NeuralNetwork(2,3,1)
nn.initalize_weights(True)

acivation_function = Sigmoid()
bp = Backpropagation(nn,5000,0.1,acivation_function)
bp.train(X,y)
bp.predict(X,y)
```

OUTPUT

For input [0.66666667 1.   ] the predicted output is 0.893111833473951 and the actual output is 0.92
For input [0.33333333 0.55555556] the predicted output is 0.8803416957986782 and the actual output is 0.86
For input [1.   0.66666667] the predicted output is 0.896434492394645 and the actual output is 0.89

PROGRAM 6:

DATASET USED: p-tennis.csv

NO USING PIP COMMAND

PROGRAM:

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

# Load Data from CSV
data = pd.read_csv('p-tennis.csv')
print("The first 5 Values of data is :\n", data.head())

# obtain train data and train output
X = data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())

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

# convert them in numbers
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 output is\n", X.head())

le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
```

```
print("\nNow the Train output is\n",y)

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))
```

OUTPUT

```
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

The First 5 values of the 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

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

Now the Train output 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

Now the Train output is
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 0.6666666666666666

PROGRAM 7:

DATASET USED: **kmeansdata.csv**

NO USING PIP COMMAND

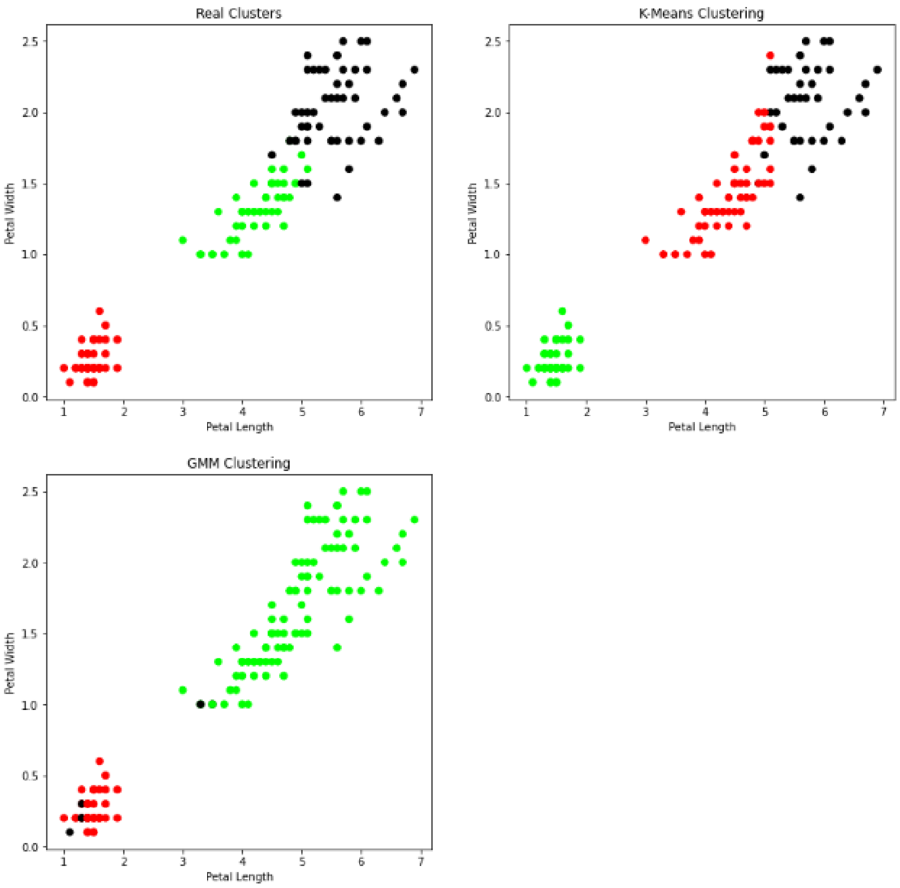
PROGRAM:

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

# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples
belongs to

# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
```

```
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_],
s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering
matched the true labels more closely than the Kmeans.')
```

OUTPUT

```
Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.
```



PROGRAM 8:

NO DATASET USED

NO USING PIP COMMAND

<u>PROGRAM:</u>

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
iris=datasets.load_iris()
```

```
x = iris.data
y = iris.target
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

OUTPUT

```
sepal-length sepal-width petal-length petal-width
[[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]
 ……………………………………………..
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[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]
```

```
Confusion Matrix
[[13  0  0]
 [ 0 17  1]
 [ 0  0 14]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        18
           2       0.93      1.00      0.97        14

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

PROGRAM 9:

NO DATASET USED

NO USING PIP COMMAND

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta
def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()
X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)
draw(10)
```

```
draw(0.1)
draw(0.01)
draw(0.001)
```

OUTPUT