In [ ]:

```
!pip install heuristicsearch
```

In [ ]:

```
from heuristicsearch.a_star_search import AStar

aj_list = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
}

heuristics = {'A': 10, 'B': 8, 'C': 5, 'D': 7, 'E': 3, 'F': 6, 'G': 5, 'H': 3, 'I':

graph = AStar(aj_list, heuristics)

graph.apply_a_star(start='A', stop='J')
```

In [ ]:

```
from heuristicsearch.ao_star import AOStar

print("Graphs-1")

heuristic = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'G': 1, 'H': 7, 'J': 1, 'T':

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

graph = AOStar(aj_list, heuristic, 'A')

graph.applyAOStar()
```

In [ ]:

```
#    Regression Algorithm
import numpy as np

inputNeurons=2

hiddenlayerNeurons=2

outputNeurons=2



input = np.random.randint(1,100,inputNeurons)

output = np.array([5.0,10.0])

hidden_layer=np.random.rand(1,hiddenlayerNeurons)



hidden_biass=np.random.rand(1,hiddenlayerNeurons)
```

```
output_bias=np.random.rand(1,outputNeurons)

hidden_weights=np.random.rand(inputNeurons,hiddenlayerNeurons)

output_weights=np.random.rand(hiddenlayerNeurons,outputNeurons)



def sigmoid (layer):

        return 1/(1 + np.exp(-layer))



def gradient(layer):

       return layer*(1-layer)

for i in range(50):

       hidden_layer=np.dot(input,hidden_weights)

       hidden_layer=sigmoid(hidden_layer+hidden_biass)

       output_layer=np.dot(hidden_layer,output_weights)

       output_layer=sigmoid(output_layer+output_bias)

       error = (output-output_layer)
       gradient_outputLayer=gradient(output_layer)

       error_terms_output=gradient_outputLayer * error

       error_terms_hidden=gradient(hidden_layer)*np.dot(error_terms_output,output_w

       gradient_hidden_weights = np.dot(input.reshape(inputNeurons,1),error_terms_h

       gradient_ouput_weights = np.dot(hidden_layer.reshape(hiddenlayerNeurons,1),e

       hidden_weights = hidden_weights + 0.05*gradient_hidden_weights
       output_weights = output_weights + 0.05*gradient_ouput_weights
       print("**********************")
       print("iteration:",i,"::::",error)
       print("#####output######",output_layer)
```

In [ ]:

```
#id3
import math
import pandas as pd
from pprint import pprint
from collections import Counter

def entropy(probs):
      return sum([-prob * math.log(prob, 2) for prob in probs])

def entropy_list(a_list):
     cnt = Counter(x for x in a_list)
     num_instance = len(a_list) * 1.0
     probs = [x / num_instance for x in cnt.values()]
     return entropy(probs)

def info_gain(df, split, target, trace=0):
     df_split = df.groupby(split)
     nobs = len(df.index) * 1.0
     df_agg_ent = df_split.agg({target: [entropy_list, lambda x: len(x) / nobs]})
     df_agg_ent.columns = ["entropy", "propObserved"]
     new_entropy = sum(df_agg_ent["entropy"] * df_agg_ent["propObserved"])
     old_entropy = entropy_list(df[target])
     return old_entropy - new_entropy

def id3(df, target, attribute_name, default_class=None):
     cnt = Counter(x for x in df[target])
     if len(cnt) == 1:
          return next(iter(cnt))
     elif df.empty or (not attribute_name):
         return default_class
     else:
         default_class = max(cnt.keys())
         gains = [info_gain(df, attr, target) for attr in attribute_name]
         index_max = gains.index(max(gains))
         best_attr = attribute_name[index_max]
         tree = {best_attr: {}}
         remaining_attr = [x for x in attribute_name if x != best_attr]
         for attr_val, data_subset in df.groupby(best_attr):
               subtree = id3(data_subset, target, remaining_attr, default_class)
               tree[best_attr][attr_val] = subtree
         return tree

def classify(instance, tree, default=None):
       attribute = next(iter(tree))
       if instance[attribute] in tree[attribute].keys():
            result = tree[attribute][instance[attribute]]
            if isinstance(result, dict):
                  return classify(instance, result)
            else:
                   return result
       else:
            return default

df_tennis = pd.read_csv('playtennis.csv')
print(df_tennis)
attribute_names = list(df_tennis.columns)
attribute_names.remove('PlayTennis')
tree = id3(df_tennis, 'PlayTennis', attribute_names)
print('\n\n The resultant decision tree is: \n\n')
pprint(tree)
```

In [ ]:

```
#                Naïve Bayes Classifier
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('id3.csv')
print("The first 5 Values of data is :\n", data.head())
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())
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_Wind = LabelEncoder()
X.Wind = le_Wind.fit_transform(X.Wind)
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 = 0.25)
classifier = GaussianNB()
classifier.fit(X_train, y_train)
predicted = classifier.predict(X_test)
predictTestData = classifier.predict([[1, 0, 1, 0]])

from sklearn.metrics import accuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
print("Predicted Value for individual Test Data:", predictTestData)
```

In [ ]:

```
# K-Means & EM Algorithm
from sklearn import datasets

from sklearn import metrics

from sklearn.cluster import KMeans

from sklearn.model_selection import train_test_split



iris = datasets.load_iris()

print(iris)

X_train,X_test,y_train,y_test = train_test_split(iris.data,iris.target)

model =KMeans(n_clusters=3)

model.fit(X_train,y_train)

model.score

print('K-Mean: ',metrics.accuracy_score(y_test,model.predict(X_test)))
#-------Expectation and Maximization----------

from sklearn.mixture import GaussianMixture

model2 = GaussianMixture(n_components=3)

model2.fit(X_train,y_train)

model2.score

print('EM Algorithm:',metrics.accuracy_score(y_test,model2.predict(X_test)))
```

In [ ]:

```
#     KNN Algorithm
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

iris=datasets.load_iris()

print("Iris Data set loaded...")

x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size

#random_state=0

```

```
for i in range(len(iris.target_names)):

     print("Label", i , "-",str(iris.target_names[i]))

classifier = KNeighborsClassifier(n_neighbors=5)

classifier.fit(x_train, y_train)

y_pred=classifier.predict(x_test)

print("Results of Classification using K-nn with K=5 ")

for r in range(0,len(x_test)):

      print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r])," Predicte

      print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

In [ ]:

```
## Regression Algorithmimport numpy as np

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 1000)
y = np.log(np.abs((x ** 2) - 1) + 0.5)
x = x + np.random.normal(scale=0.05, size=1000)
plt.scatter(x, y, alpha=0.3)

def local_regression(x0, x, y, tau):
    x0 = np.r_[1, x0]
    x = np.c_[np.ones(len(x)), x]
    xw = x.T * radial_kernel(x0, x, tau)
    beta = np.linalg.pinv(xw @ x) @ xw @ y
    return x0 @ beta

def radial_kernel(x0, x, tau):
    return np.exp(np.sum((x - x0) ** 2, axis=1) / (-2 * tau ** 2))

def plot_lr(tau):
    domain = np.linspace(-5, 5, num=500)
    pred = [local_regression(x0, x, y, tau) for x0 in domain]
    plt.scatter(x, y, alpha=0.3)
    plt.plot(domain, pred, color="red")
    plt.show()

plot_lr(1)
```

In [ ]:

```
#Candidate Elimination Algorithm for EnjoySport
import numpy as np
import pandas as pd

data = pd.read_csv('playtennis.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 j in range(len(specific_
    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] = '?'
        elif 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(f'Steps of Candidate Elimination Algorithm {i + 1}')
        print('Specific_h:', specific_h)
        print('General_h:', general_h)

    indices = [i for i, val in enumerate(general_h) if val == ['?' for _ in range(l

    for i in indices:
        general_h.remove(['?' for _ in range(len(specific_h))])

    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print('----------------Final Answer----------------\n')
print('Final specific_h:\n', s_final)
print('Final general_h:\n', g_final)
```