1)import sys

inf=99999 g=[[0,4,3,inf,inf,inf,inf], [inf,0,inf,inf,12,5,inf], [inf,inf,0,7,10,inf,inf], [inf,inf,inf,0,2,inf,inf], [inf,inf,inf,inf,0,inf,5], [inf,inf,inf,inf,inf,0,16], [inf,inf,inf,inf,inf,inf,0],] h=[14,12,11,6,4,11,0] src=0 goal=6 class obj: def __init__(self,cost,path): self.cost=cost self.path=path arr=[] new_item=obj(h[src],[src]) arr.append(new_item)

while arr:

cur_item=arr[0] cur_node=cur_item.path[-1] cur_cost=cur_item.cost cur_path=cur_item.path for i in range(0,len(h)):

if g[cur_node][i]!=inf and g[cur_node][i]!=0: new_cost=cur_cost-h[cur_node]+h[i]+g[cur_node][i] new_path=cur_path.copy() new_path.append(i) if i==goal: print(new_cost) print(new_pathnew_item=obj(new_cost,new_path)

arr.append(new_item); arr.pop(0)

arr=sorted(arr,key=lambda item:item.cost

2) import os import time def get_node (mark_road,extended): temp=[0] i=0 while 1: current=temp[i] if current not in extended: return current else: for child in mark_road[current]: if child not in temp: temp.append(child) i+=1 def get_current(s,nodes_tree): if len(s)==1: return s[0] for node in s: flag=True for edge in nodes_tree(node): for child_nod in edge: if child_nod in s: flag=False if flag: return node def get_pre(current,pre,pre_list): if current==0: return for pre_node in pre[current]: if pre_node not in pre_list: pre_list.append(pre_node) return def ans_print(mark_rode,node_tree): print("The final connection is as follows") temp=[0] while temp: time.sleep(1) print(f"[{temp[0]}]-->{mark_rode[temp[0]]}") for child in mark_rode[temp[0]]: if node_tree[child]!=[[child]]: temp.append(child) temp.pop(0) time.sleep(5) os.system('cls'

return def AOstar(nodes_tree,h_val): futility=0xfff extended=[] choice=[] mark_rode={0:None} solved={} pre={0:[]} for i in range(1,9): pre[i]=[] for i in range(len(nodes_tree)): solved[i]=False os.system('cls') print("The connection process is as follows") time.sleep(1) while not solved[0] and h_val[0]

pre_list=[] if current!=0: get_pre(current,pre,pre_list) s.extend(pre_list) if not solved[0]: print("The query failed, the path could not be found") else: ans_print(mark_rode, nodes_tree) return if __name__=="__main__": nodes_tree={} nodes_tree[0]=[[1],[4,5]] nodes_tree[1]=[[2],[3]] nodes_tree[2]=[[3],[2,5]] nodes_tree[3]=[[5,6]] nodes_tree[4]=[[5],[8]] nodes_tree[5]=[[6],[7,8]] nodes_tree[6]=[[7,8]] nodes_tree[7]=[[7]] nodes_tree[8]=[[8]] h_val=[3,2,4,4,1,1,2,0,0] AOstar(nodes_tree, h_val)

3) import csv a = [] csvfile = open('pgm3.csv', 'r') reader = csv.reader(csvfile) print("Data present in csv file is: ") for row in reader: a.append(row) print(row) num_attributes = len(a[0]) - 1 print("\nInitial hypothesis is ") s = ['0'] * num_attributes g = ['?'] * num_attributes print("The most specific: ", s) print("The most general: ", g) for j in range(0, num_attributes): s[j] = a[0][j] print("\nThe candidate algorithm") temp = [] for i in range(0, len(a)): if (a[i][num_attributes] == 'yes'): for j in range(0, num_attributes): if (a[i][j] != s[j]): s[j] = '?' for j in range(0, num_attributes): for k in range(1, len(temp)): if temp[k][j] != '?' and temp[k][j] != s[j]: del temp[k] print("\nfor instance {0} the space hypothesis is s{0}\n".format(i + 1), s) if (len(temp) == 0): print("\nfor instance {0} the hypothesis is G{0}\n".format(i + 1), g) else: print("\nfor instance {0} the hypothesis is G{0}\n".format(i + 1), temp) if (a[i][num_attributes] == 'no'): for j in range(0, num_attributes): if (s[j] != a[i][j] and s[j] != '?'): g[j] = s[j] temp.append(g) g = ['?'] * num_attributes print("\nFor instance{0} the hypothesis is s{0}\n".format(i + 1), s) print("\nFor instance{0} the hypothesis is g{0}\n".format(i + 1), temp)

4) import pandas as pd from collections import Counter import math # Read the data tennis = pd.read_csv('pgm4.csv') print("\nGiven PlayTennis Data Set:\n\n", tennis) def entropy(alist): c = Counter(x for x in alist) instances = len(alist) prob = [x/instances for x in c.values()] return sum([-p*math.log(p, 2) for p in prob]) def information_gain(d, split, target): splitting = d.groupby(split) n = len(d.index) agent = splitting.agg({target: [entropy, lambda x: len(x)/n]}) agent.columns = ['Entropy', 'Observations'] newentropy = sum(agent['Entropy'] * agent['Observations']) oldentropy = entropy(d[target]) return oldentropy - newentropy def id3(sub, target, a): count = Counter(x for x in sub[target]) if len(count) == 1: return next(iter(count)) else: gain = [information_gain(sub, attr, target) for attr in a] print("\nGain =", gain) maximum = gain.index(max(gain)) best = a[maximum] print("\nBest Attribute:", best) tree = {best: {}} remaining = [i for i in a if i != best] for val, subset in sub.groupby(best): subtree = id3(subset, target, remaining) tree[best][val] = subtree return tree names = list(tennis.columns) print("\nList of Attributes:", names) names.remove('PlayTennis') print("\nPredicting Attributes:", names) # Convert the 'observations' column to a dictionary tree = id3(tennis, 'PlayTennis', names) print("\n\nThe Resultant Decision Tree is:\n") print(tree)

5) 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 derivatives_sigmoid(x): return x*(1-x) epoch=7000 lr=0.1 inputlayer_neuron=2 hiddenlayer_neuron=3 output_neuron=1 wh=np.random.uniform(size=(inputlayer_neuron,hiddenlayer_neuron)) bh=np.random.uniform(size=(1,hiddenlayer_neuron)) wout=np.random.uniform(size=(hiddenlayer_neuron,output_neuron)) bout=np.random.uniform(size=(1,output_neuron)) 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=derivatives_sigmoid(output) d_output=Eo*outgrad EH=d_output.dot(wout.T) hiddengrad=derivatives_sigmoid(hlayer_act) d_hiddenlayer=EH*hiddengrad wout+=hlayer_act.T.dot(d_output)*lr print("Input:\n"+str(X)) print("Actual Output:\n"+str(y)) print("Predicted Output:\n",output

6) import csv import math import random import statistics def cal_probability(x,mean,stdev): exponent=math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2)))) return (1/(math.sqrt(2*math.pi)*stdev))*exponent dataset=[] dataset_size=0 with open('pgm6.csv') as csvfile: lines=csv.reader(csvfile) for row in lines: dataset.append([float(attr)for attr in row]) dataset_size=len(dataset) print("Size of dataset is: ",dataset_size) train_size=int(0.7*dataset_size) print(train_size) x_train=[] x_test=dataset.copy() training_indexes=random.sample(range(dataset_size),train_size) for i in training_indexes: x_train.append(dataset[i]) x_test.remove(dataset[i]) classes={} for samples in x_train: last=int(samples[-1]) if last not in classes: classes[last]=[] classes[last].append(samples) print(classes) summaries={} for classValue,training_data in classes.items(): summary=[(statistics.mean(attribute),statistics.stdev(attribute)) for attribute in zip(*training_data)] del summary[-1] summaries[classValue]=summary print(summaries) x_prediction=[] for i in x_test: probabilities={} for classValue, classSummary in summaries.items(): probabilities[classValue]=1 for index, attr in enumerate(classSummary): probabilities[classValue]*=cal_probability(i[index],attr[0],attr[1]) best_label,best_prob=None,-1 for classValue,probability in probabilities.items(): if best_label is None or probability>

best_prob: best_prob=probability
best_label=classValue
x_prediction.append(best_label) correct=0 for index,key in enumerate(x_test): if x_test[index][-1]==x_prediction[index]: correct+=1
print("Accuracy:",correct/(float(len(x_test)))*100)

7) import numpy as np import pandas as pd from matplotlib import pyplot as plt from sklearn.mixture import GaussianMixture from sklearn.cluster import KMeans
data=pd.read_csv("pgm7.csv") print("Input data and shape") print(data.shape)
data.head() f1=data['v1'].values
f2=data['v2'].values x=np.array(list(zip(f1,f2)))
print("X",x) print("Graph for which dataset")
plt.scatter(f1,f2,c='black',s=7) plt.show()
Kmeans=KMeans(20,random_state=0)
labels=Kmeans.fit(x).predict(x)
print("Labels",labels)
centroids=Kmeans.cluster_centers_
print("centeroids",centroids)
plt.scatter(x[:,0],x[:,1],c=labels,s=40,cmap="viridis"); print("Grapg using KMeans Algorithm")
plt.scatter(centroids[:,0],centroids[:,1],marker='*',s=200,c='#050505') plt.show()
gmm=GaussianMixture(n_components=3).fit(x) labels=gmm.predict(x)
probs=gmm.predict_proba(x)
size=10*probs.max(1)**3 print("Graph using EM algorithm")
plt.scatter(x[:,0],x[:,1],c=labels,s=size,cmap='viridis'); plt.show()

8) from sklearn.datasets import load_iris
iris=load_iris() x=iris.data y=iris.target
print(x[:5],y[:5]) from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.4,random_state=1)
print(iris.data.shape) print(len(xtrain))
print(len(ytest)) from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=1)
knn.fit(xtrain,ytrain) pred=knn.predict(xtest)
from sklearn import metrics
print("Accuracy",metrics.accuracy_score(ytest,pred)) print(iris.target_names[2])
ytestn=[iris.target_names[i] for i in ytest]
predn=[iris.target_names[i] for i in pred]
print(" predicted actual") for i in range(len(pred)): print(i," ",predn[i]," ",ytestn[i])

9) import numpy as np import matplotlib.pyplot as plt import pandas as pd
tou=0.5 data=pd.read_csv("pgm9.csv")
X_train=np.array(data.total_bill) print(X_train)
X_train=X_train[:,np.newaxis]
print(len(X_train)) y_train=np.array(data.tip)
X_test=np.array([i/10 for i in range(500)])
X_test=X_test[:,np.newaxis] y_test=[] count=0
for r in range(len(X_test)): wts=np.exp(-np.sum((X_train-X_test[r])**2,axis=1)/(2*tou**2))
W=np.diag(wts)
factor1=np.linalg.inv(X_train.T.dot(W).dot(X_train))
parameters=factor1.dot(X_train.T).dot(W).dot(y_train) prediction=X_test[r].dot(parameters)
y_test.append(prediction) count+=1
print(len(y_test)) y_test=np.array(y_test)
plt.plot(X_train.squeeze(),y_train,'o')
plt.plot(X_test.squeeze(),y_test,'o') plt.show()