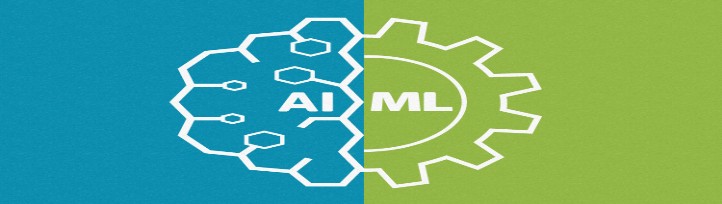
Department of Computer Science Engg.

**VIISemester**

**AI&MLLABPROGRAMS (18CSL76)**



**Syllabus**

1. Implement A\* Search algorithm.
2. Implement AO\* Search algorithm.
3. For agiven set oftraining dataexamples storedina .CSV file, implement anddemonstratethe

Candidate-Elimination algorithmtooutputadescriptionof the set ofall hypothesesconsistentwiththetraining examples.

1. Writeaprogramtodemonstratetheworking ofthedecision treebased ID3 algorithm. Useanappropriatedata set for buildingthedecisiontreeandapplythis knowledgetoclassify anewsample.
2. BuildanArtificial Neural Network byimplementing theBackpropagationalgorithmandtest thesameusing appropriate datasets.
3. Writeaprogramtoimplement thenaïveBayesianclassifier for asampletraining dataset storedas a .CSVfile. Computetheaccuracyoftheclassifier, considering fewtestdatasets.
4. Apply EMalgorithmtocluster aset ofdata storedina .CSVfile. Usethesamedataset for clustering using

k-Meansalgorithm. Comparetheresults ofthesetwoalgorithms andcomment on thequality of clustering. You canadd Java/Python ML libraryclasses/APIin theprogram.

1. Writeaprogramtoimplement k-Nearest Neighbour algorithmtoclassify theirisdataset. Print bothcorrect and wrongpredictions.Java/Python MLlibraryclasses canbeusedfor thisproblem.
2. Implement thenon-parametricLocally Weighted Regression algorithminordertofitdatapoints. Select appropriate datasetfor your experiment anddrawgraphs.
   1. Implement A\* Search algorithm.

**SourceCode:**

import sys inf=99999 g=[

[0,1,inf,inf,inf,10],

[1,0,2,1,inf,inf],

[inf,2,0,inf,5,inf],

[inf,1,inf,0,3,4],

[inf,inf,3,0,2],

[10,inf,inf,4,2,0],

]

h= [5,3,4,2,6,0]

src=0

goal = 5 class obj:

def init (self,cost,path):

self.cost= cost self.path =path

arr= []

new\_item= obj(h[src],[src]) arr.append(new\_item) #a\* algorithm

whilearr: cur\_item= arr[0]

cur\_node= cur\_item.path[-1] cur\_cost=cur\_item.cost cur\_path=cur\_item.path foriinrange(0,len(h)):

ifg[cur\_node][i]!=infandg[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) ifi==goal:

print(new\_cost) print(new\_path)

# sys.exit()

new\_item= obj(new\_cost,new\_path) arr.append(new\_item)

arr.pop(0)

arr=sorted(arr,key=lambdaitem:item.cost)

**OUTPUT**

17

[0, 2, 3, 4, 6]

18

[0, 2, 4, 6]

21

[0,1,4,6]

25

[0,1,5,6]

* 1. **ImplementAO\*Searchalgorithm importtime**

**importos defget\_node(mark\_road,extended):**

**temp=[0]**

**i=0 while1:**

**current=temp[i] ifcurrentnotinextended:**

**returncurrent else:**

**for childinmark\_road[current]: ifchildnotintemp:**

**temp.append(child) i+=1**

**defget\_current(s,nodes\_tree):**

**iflen(s)==1: returns[0] fornodeins: flag=True**

**foredgeinnodes\_tree(node):**

**for child\_nodeinedge:**

**ifchild\_nodeins: flag=False**

**ifflag: returnnode**

**defget\_pre(current,pre,pre\_list):**

**ifcurrent==0: return**

**forpre\_nodeinpre[current]: ifpre\_nodenotinpre\_list:**

**pre\_list.append(pre\_node)**

**get\_pre(pre\_node,pre,pre\_list) return**

**defans\_print(mark\_rode,node\_tree): print("Thefinalconnectionisasfollow:") temp=[0]**

**whiletemp: time.sleep(1)**

**print(f"[{temp[0]}] >{mark\_rode[temp[0]]}")**

**forchildinmark\_rode[temp[0]]: ifnode\_tree[child]!=[[child]]:**

**temp.append(child)**

**temp.pop(0) time.sleep(5) os.system('cls') return**

**defAOstar(node\_tree,h\_val):**

**futility=0xfff extended=[] choice=[] mark\_rode={0:None} solved={}**

**pre={0:[]}**

**foriinrange(1,9): pre[i]=[]**

**foriinrange(len(nodes\_tree)): solved[i]=False**

**os.system('cls')**

**print("Theconnectionprocessisasfollows") time.sleep(1)**

**whilenotsolved[0]andh\_val[0]<futility: node=get\_node(mark\_rode,extended) extended.append(node) ifnodes\_tree[node]isNone:**

**h\_val[node]=futility**

**continue forsuc\_edgeinnodes\_tree[node]:**

**forsuc\_nodeinsuc\_edge: ifnodes\_tree[suc\_node]==[[suc\_node]]:**

**solved[suc\_node]=True**

**s=[node] whiles:**

**current=get\_current(s,nodes\_tree) s.remove(current) origen\_h=h\_val[current] origen\_s=solved[current] min\_h=0xfff**

**for edgeinnodes\_tree[current]: edge\_h=0**

**fornodeinedge: edge\_h+=h\_val[node]+1**

**ifedge\_h<min\_h: min\_h=edge\_h h\_val[current]=min\_h mark\_rode[current]=edge**

**ifmark\_rode[current] notinchoice:**

**choice.append(mark\_rode[current]) print(f"[{current}] {mark\_rode[current]}")**

**time.sleep(1)**

**for child\_nodeinmark\_rode[current]: pre[child\_node].append(current)**

**solved[current]=True**

**for nodeinmark\_rode[current]: solved[current]=solved[current]andsolved[node]**

**iforigen\_s!=solved[current]or origen\_h!=h\_val[current]: pre\_list=[]**

**ifcurrent!=0: get\_pre(current,pre,pre\_list)**

**s.extend(pre\_list)**

**ifnotsolved[0]:**

**print("Thequeryfailed,thepathcouldnotbefound!") 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)**

**OUTPUT:**

**Theconnectionprocessisasfollows [0] [1]**

**[1]-----[2]**

**[0]-----[4, 5]**

**[4]-----[8]**

**[5]-----[7, 8]**

**Thefinal connectionisasfollow:**

**[0]----->[4, 5]**

**[4]----->[8]**

**[5]----->[7, 8]**

**Program:3.CANDIDATEELIMINATIONALGORITHM**

importcsv

a=[]

csvfile=open('1.csv','r')

reader=csv.reader(csvfile)for rowin reader:

a.append(row) print(row)

num\_attributes=len(a[0])-1 print("Initial hypothesisis") S=['0']\*num\_attributes G=['?']\*num\_attributes print("Themost specific: ",S) print("Themost general : ",G)

for j in range(0,num\_attributes): S[j]=a[0][j]

print("Thecandidatealgorithm\n") 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]!='?' andtemp[k][j]!=S[j]:del temp[k]

print("Forinstance{0} thehypothesis is S{0}".format(i+1),S) if(len(temp)==0):

print("Forinstance{0}thehypothesisisG{0}".format(i+1),G)else:

print("Forinstance{0} thehypothesisisS{0}".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("Forinstance{0} thehypothesis is S{0}".format(i+1),S) print("For instance {0}the hypothesisis G{0}".format(i+1),temp)

**output:**

['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']Initial hypothesisis

Themostspecific: ['0', '0', '0', '0', '0', '0']

Themostgeneral : ['?', '?', '?', '?', '?', '?']The candidate algorithm

Forinstance 1 the hypothesisis S1 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm','Same'] Forinstance 1 thehypothesisis G1 ['?', '?', '?', '?', '?', '?']

Forinstance 2 the hypothesisis S2 ['Sunny', 'Warm', '?', 'Strong', 'Warm','Same']For instance 2 thehypothesisis G2 ['?', '?', '?', '?', '?', '?']

Forinstance 3 the hypothesisis S3 ['Sunny', 'Warm','?','Strong', 'Warm','Same']

Forinstance3 thehypothesisisG3 [['Sunny','?', '?', '?', '?', '?'], ['?', 'Warm','?','?','?', '?'], ['?', '?', '?','?', '?', 'Same']]

Forinstance 4 thehypothesisis S4 ['Sunny','Warm', '?', 'Strong', '?', '?']

Forinstance 4 thehypothesisis S4 [['Sunny','?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

**Program: 4.ID3 ALGORITHM**

importpandasaspd

fromcollectionsimport Counter importmath

tennis=pd.read\_csv('playtennis.csv')

print("\nGiven Play Tennis Data Set:\n\n", tennis)

defentropy(alist):

c=Counter(x forxinalist) instances=len(alist)

prob =[x / instancesforx inc.values()] return sum( [-p\*math.log(p, 2) forp inprob])

definformation\_gain(d, split, target): splitting=d.groupby(split)

n=len(d.index)

agent =splitting.agg({target:[entropy, lambdax: len(x)/n] })[target]#aggregatingagent.columns = ['Entropy', 'observations']

newentropy =sum( agent['Entropy'] \* agent['observations']) oldentropy=entropy(d[target])

returnoldentropy - newentropy

defid3(sub, target, a):

count =Counter(x forxinsub[target])# classof YES/NOif len(count) == 1:

return next(iter(count)) # nextinput dataset, or raises StopIterationwhen EOFishit

else:

gain=[information\_gain(sub, attr, target) forattr ina] print("Gain=",gain)

maximum=gain.index(max(gain)) best=a[maximum]

print("Best Attribute:",best)

tree= {best:{}}

remaining =[i for i inaifi !=best]

forval, subset insub.groupby(best): subtree =id3(subset,target,remaining) tree[best][val] =subtree

return tree

names=list(tennis.columns) print("List of Attributes:", names) names.remove('PlayTennis') print("Predicting Attributes:", names)

tree=id3(tennis,'PlayTennis',names) print("\n\nThe Resultant Decision Tree is:\n")

print(tree)

**Dataset:playtennis.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PlayTennis | Outlook | Temperature | Humidity | Wind |
| No | Sunny | Hot | High | Weak |
| No | Sunny | Hot | High | Strong |
| Yes | Overcast | Hot | High | Weak |
| Yes | Rain | Mild | High | Weak |
| Yes | Rain | Cool | Normal | Weak |
| No | Rain | Cool | Normal | Strong |
| Yes | Overcast | Cool | Normal | Strong |
| No | Sunny | Mild | High | Weak |
| Yes | Sunny | Cool | Normal | Weak |
| Yes | Rain | Mild | Normal | Weak |
| Yes | Sunny | Mild | Normal | Strong |
| Yes | Overcast | Mild | High | Strong |
| Yes | Overcast | Hot | Normal | Weak |
| No | Rain | Mild | High | Strong |

**output:**

GivenPlay Tennis DataSet:

PlayTennis Outlook Temperature Humidity Wind

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | No | Sunny | Hot | High | Weak |
| 1 | No | Sunny | Hot | High | Strong |
| 2 | Yes | Overcast | Hot | High | Weak |
| 3 | Yes | Rain | Mild | High | Weak |
| 4 | Yes | Rain | Cool | Normal | Weak |
| 5 | No | Rain | Cool | Normal | Strong |
| 6 | Yes | Overcast | Cool | Normal | Strong |
| 7 | No | Sunny | Mild | High | Weak |
| 8 | Yes | Sunny | Cool | Normal | Weak |
| 9 | Yes | Rain | Mild | Normal | Weak |
| 10 | Yes | Sunny | Mild | Normal | Strong |
| 11 | Yes | Overcast | Mild | High | Strong |
| 12 | Yes | Overcast | Hot | Normal | Weak |
| 13 | No | Rain | Mild | High | Strong |

List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity','Wind'] Predicting Attributes: ['Outlook','Temperature', 'Humidity', 'Wind']

Gain=[0.2467498197744391, 0.029222565658954647, 0.15183550136234136,

0.04812703040826927]

Best Attribute: Outlook

Gain=[0.01997309402197489, 0.01997309402197489, 0.9709505944546686]

Best Attribute: Wind

Gain=[0.5709505944546686, 0.9709505944546686, 0.01997309402197489]

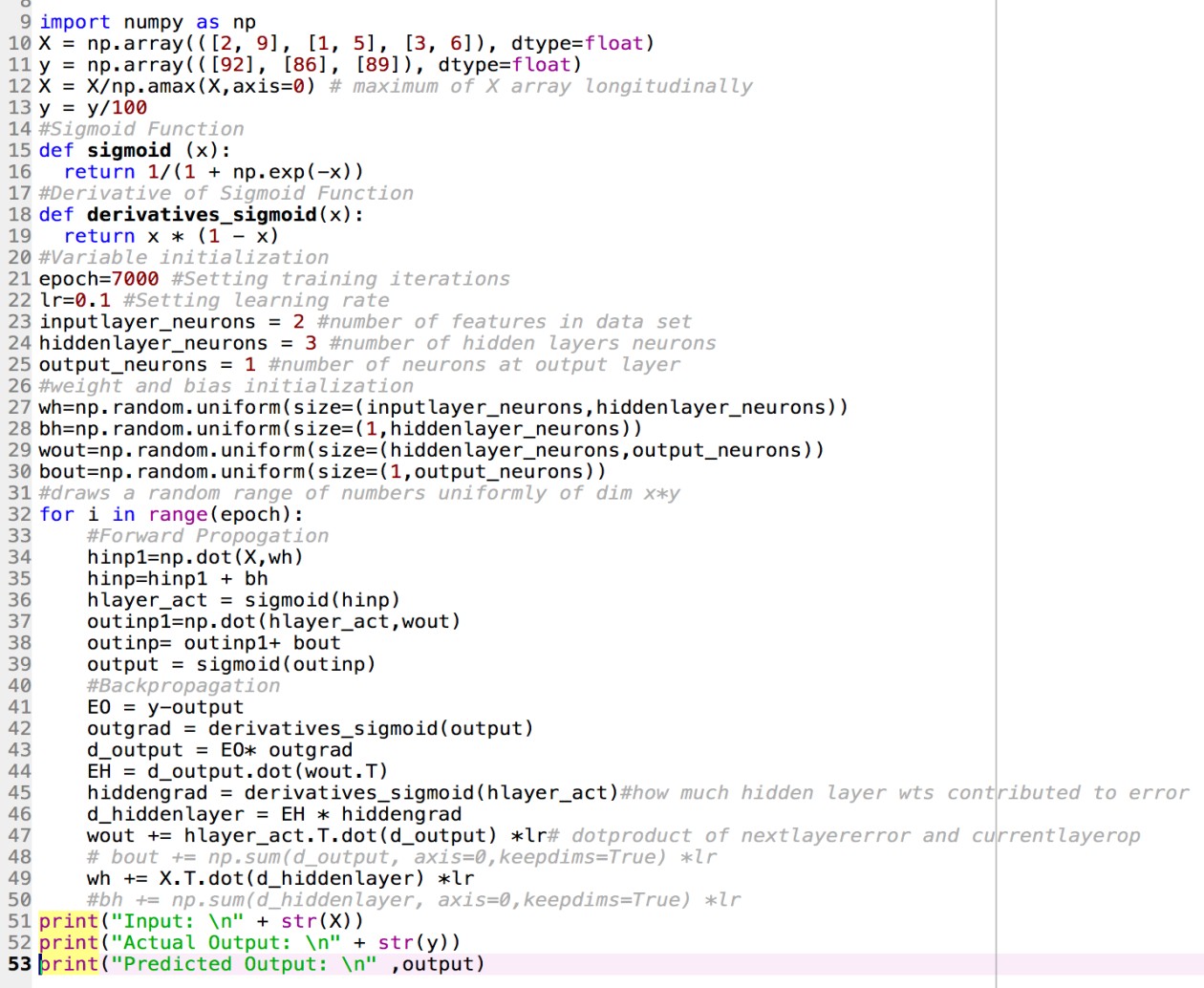
Best Attribute: Humidity

The Resultant DecisionTree is :

{'Outlook': {'Overcast':'Yes', 'Rain': {'Wind': {'Strong': 'No', 'Weak':'Yes'}}, 'Sunny': {'Humidity':

{'High': 'No','Normal':'Yes'}}}}

**Program:5.BACKPROPOGATION**

**SourceCode**

importnumpyasnp 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

defsigmoid(x):

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

defderivatives\_sigmoid(x): returnx\*(1-x)

epoch=7000 lr=0.1

inputlayer\_neurons=2 hiddenlayer\_neurons=3 output\_neurons=1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) bout=np.random.uniform(size=(1,output\_neurons))

for i inrange(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)

E0=y-output outgrad=derivatives\_sigmoid(output) d\_output=E0\*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)

output

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output: [[0.92]

[0.86]

[0.89]]

Predicted Output: [[0.89282584]

[0.87763012]

[0.89905218]]

**Program: 6.NAÏVEBAYESIANCLASSIFIER**

importcsv importmath importrandom importstatistics

defcal\_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

withopen('lab5.csv') ascsvfile: lines=csv.reader(csvfile)

for rowin lines:

dataset.append([float(attr) forattrinrow]) dataset\_size=len(dataset)

print("Sizeofdataset 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 lastnotinclasses:

classes[last]=[]

classes[last].append(samples)

print(classes) summaries={}

forclassValue,training\_data inclasses.items(): summary=[(statistics.mean(attribute),statistics.stdev(attribute)) forattribute in

zip(\*training\_data)] del summary[-1]

summaries[classValue]=summary

print(summaries) X\_prediction=[]

for i in X\_test: probabilities={}

forclassValue,classSummary insummaries.items(): probabilities[classValue]=1

for index,attrinenumerate(classSummary): probabilities[classValue]\*=cal\_probability(i[index],attr[0],attr[1])

best\_label,best\_prob=None,-1

forclassValue,probability inprobabilities.items():if best\_labelisNoneorprobability>best\_prob:

best\_prob=probability best\_label=classValue

X\_prediction.append(best\_label) correct=0

for index,keyinenumerate(X\_test):

if X\_test[index][-1]==X\_prediction[index]: correct+=1

print("Accuracy:",correct/(float(len(X\_test)))\*100)

**6Dataset: 6.csv**

6,148,72,35,0,33.6,0.627,50,1

1,85,66,29,0,26.6,0.351,31,0

8,183,64,0,0,23.3,0.627,32,1

1,89,66,23,94,28.1,0.167,21,0

0,137,40,35,168,43.1,2.288,33,1

5,116,74,0,0,25.6,0.201,30,0

3,78,50,32,88,31,0.284,26,1

10,115,0,0,0,35.3,0.134,29,0

2,197,70,45,543,30.5,0.158,53,1

8,125,96,0,0,0,0.232,54,1

4,110,92,0,0,37.6,0.191,30,0

10,168,74,0,0,38,0.537,34,1

10,139,80,0,0,27.1,1.441,57,0

1,189,60,23,846,30.1,0.398,59,1

5,166,72,19,175,25.8,0.587,51,1

7,100,0,0,0,30,0.484,32,1

**6output:**

Sizeofdatasetis: 768 537

{0: [[1.0, 107.0, 68.0, 19.0, 0.0, 26.5, 0.165, 24.0, 0.0],[1.0, 144.0, 82.0, 40.0, 0.0, 41.3, 0.607, 28.0,

0.0],[1.0, 105.0, 58.0, 0.0, 0.0, 24.3, 0.187, 21.0, 0.0]

{0: [(3.454022988505747, 3.1284989024698904),(110.01724137931035, 26.938498454745453),

(67.92528735632185, 18.368785190361336),(19.612068965517242, 15.312369913377424),

(68.95689655172414, 105.42637942980888),(30.54080459770115, 7.710567727617014),

(0.4458764367816092,0.31886309966940785),(31.74712643678161, 12.079437732209673)], 1:

[(4.64021164021164, 3.7823318201241096),(143.07407407407408, 32.13758346670748),

(72.03174603174604, 19.92883742963596),(22.49206349206349, 18.234179691371473),

(99.04232804232804, 127.80927573836007),(35.351851851851855, 7.308750166698269),

(0.5427301587301587,0.3832947121639522),(36.43386243386244, 10.813315097901606)]}

Accuracy: 78.78787878787878

**Program: 7.EMALGORITHM**

importnumpyasnp importpandasaspd

frommatplotlibimport pyplotasplt

fromsklearn.mixture import GaussianMixture fromsklearn.clusterimport KMeans

data = pd.read\_csv('lab8.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 whole 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("centroids ",centroids)

plt.scatter(X[:, 0], X[:, 1], c=labels,s=40, cmap='viridis');print('Graph using KmeansAlgorithm')

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 EMAlgorithm')

plt.scatter(X[:, 0], X[:, 1], c=labels, s=size, cmap='viridis'); plt.show()

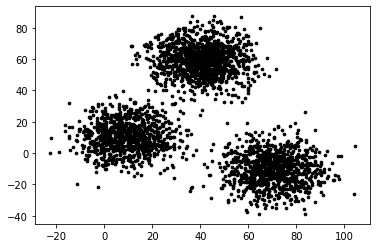
OUTPUT:

**Input Data and Shape (3000, 3)**

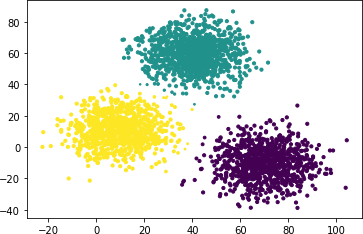
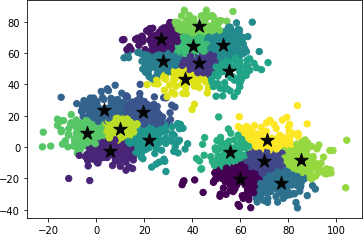
**X [[ 2.072345 -3.241693][ 17.93671 15.78481 ][ 1.083576 7.319176]...**

**[ 64.46532 -10.50136 ][ 90.72282 -12.25584 ][ 64.87976 -24.87731 ]]**

**Graph for whole dataset**



**labels [ 2 5 14 ... 4 16 0]**



**centroids [[ 59.83204156 -20.27127019]**

|  |  |  |  |
| --- | --- | --- | --- |
| **[** | **26.93926814** | **68.72877415]** | |
| **[** | **5.74728456** | **-2.4354335 ]** | |
| **[** | **42.74508801** | **53.78669448]** | |
| **[** | **69.93697849** | **-8.99255106]** | |
| **[** | **19.32058349** | **22.32585954]** | |
| **[** | **3.32731778** | **23.630905** | **]** |
| **[** | **76.820093** | **-23.03153657]** | |
| **[** | **27.80251033** | **54.98355311]** | |
| **[** | **52.85959994** | **65.33275606]** | |
| **[** | **22.0826464** | **4.72511417]** | |
| **[** | **55.18393576** | **48.32773467]** | |
| **[** | **55.89985798** | **-3.10396622]** | |
| **[** | **40.09743894** | **64.23009528]** | |
| **[** | **-4.04689718** | **8.812598** | **]** |
| **[** | **42.75426718** | **77.03129218]** | |
| **[** | **85.39067866** | **-8.33454658]** | |
| **[** | **9.89401653** | **11.85203706]** | |
| **[** | **37.08384976** | **43.23678776]** | |
| **[** | **71.10416952** | **4.2786267 ]]** | |

**Graph using Kmeans Algorithm**

**Graph using EM Algorithm**

**Program:8.K-NEARESTNEIGHBOUR**

importnumpyasnp

fromsklearn.datasets importload\_iris iris=load\_iris()

x=iris.data y=iris.target print(x[:5],y[:5])

fromsklearn.model\_selection import train\_test\_splitxtrain,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))

fromsklearn.neighbors import KNeighborsClassifier knn=KNeighborsClassifier(n\_neighbors=1) knn.fit(xtrain,ytrain)

pred=knn.predict(xtest)

fromsklearn import metrics print("Accuracy",metrics.accuracy\_score(ytest,pred)) print(iris.target\_names[2]) ytestn=[iris.target\_names[i] for i inytest] predn=[iris.target\_names[i] fori inpred]

print(" predicted Actual") for i in range(len(pred)):

print(i," ",predn[i]," ",ytestn[i])

OUTPUT:

[[5.13.51.40.2]

[4.93. 1.40.2]

[4.73.2 1.3 0.2]

[4.63.1 1.5 0.2]

[5. 3.61.40.2]][0000 0]

(150, 4)

9060

Accuracy 0.9666666666666667

virginicapredicted Actual 0 setosa setosa

1 versicolor versicolor

1. versicolorversicolor
2. setosasetosa
3. virginicavirginica
4. virginicaversicolor
5. virginicavirginica
6. setosasetosa
7. setosasetosa
8. virginicavirginica
9. versicolorversicolor

**Program: 9.LOCALLYWEIGHTEDREGRESSIONALGORITHM**

importnumpyas np

importmatplotlib.pyplot asplt importpandasaspd

tou= 0.5 data=pd.read\_csv("lab10.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 inrange(500)])X\_test

=X\_test[:, np.newaxis] y\_test =[]

count= 0

forrinrange(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)

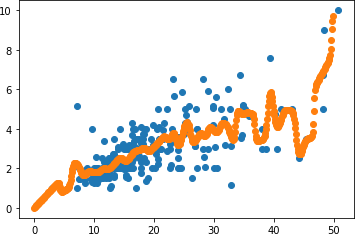
count+=1 print(len(y\_test))

y\_test =np.array(y\_test) plt.plot(X\_train.squeeze(),y\_train,'o')

plt.plot(X\_test.squeeze(), y\_test,'o')plt. show()

**DATASET:[245rows]**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| total\_bill | tip | sex | smoker | day | time | size |
| 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| 21.01 | 3.5 | Male | No | Sun | Dinner | 3 |
| 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |
| 25.29 | 4.71 | Male | No | Sun | Dinner | 4 |
| 8.77 | 2 | Male | No | Sun | Dinner | 2 |
| 26.88 | 3.12 | Male | No | Sun | Dinner | 4 |
| 15.04 | 1.96 | Male | No | Sun | Dinner | 2 |
| **Output** |  |  |  |  |  |  |



**DesignBasedPrograms**

**Program: 1. FINDS**

**Dataset: 1.csv**

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

num\_attributes=6 a=[]

print("\n The giventrainingdataset \n")

csvfile=open('1.csv','r') reader=csv.reader(csvfile) for rowin reader:a.

append(row)

print(row)

print("Theinitial valuesofhypothesis") hypothesis=['0']\*num\_attributes print(hypothesis)

for j in range(0,num\_attributes): hypothesis[j]=a[0][j]

for i in range(0,len(a)): if(a[i][num\_attributes]=='Yes'):

for j in range(0,num\_attributes):

if(a[i][j]!=hypothesis[j]): hypothesis[j]='?'

else:

hypothesis[j]=a[i][j]

print("Fortraining instanceno:",i," thehypothesisis",hypothesis) print("Themaximallyspecifichypothesisis",hypothesis)

**output:**

Thegiventraining dataset

['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']

Theinitialvaluesof hypothesis ['0', '0', '0', '0', '0', '0']

Fortraininginstanceno:0 thehypothesisis['Sunny', 'Warm', 'Normal', 'Strong', 'Warm','Same']Fortraining instanceno:1 thehypothesisis['Sunny','Warm','?','Strong', 'Warm', 'Same']

Fortraininginstanceno: 2 thehypothesisis['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']Fortraining instanceno: 3 thehypothesisis['Sunny','Warm', '?', 'Strong', '?', '?']

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