**K means 1D**

#include <iostream>

#include <cstdlib>

#include <vector>

#include <cmath>

using namespace std;

vector <int> cls1,cls2;

int mean(vector <int> v){

int sum = 0;

for(int i=0;i<v.size();i++){

sum+=v[i];

}

float mean = (float)sum/(float)v.size();

return round(mean);

}

int main(){

vector <int> arr{2,3,4,10,11,12,20,25,30};

// int arr[] = {2,3,4,10,11,12,20,25,30};

int m1 = 4, m2 = 12,nm1=0,nm2=0,c=0;

vector <int> cls1,cls2;

for(int j=0;j<10;j++){

for(int i=0;i<9;i++){

if( abs(m1-arr[i]) < abs(m2-arr[i]) ){

cls1.push\_back(arr[i]);

}

else{

cls2.push\_back(arr[i]);

}

}

nm1 = m1;

nm2 = m2;

m1 = mean(cls1);

m2 = mean(cls2);

if(nm1==m1 && nm2==m2){

cout<<"Iterations: "<<c<<endl;

break;

}

cls1.clear();

cls2.clear();

c++;

}

for(int i=0;i<cls1.size();i++){

cout<<cls1[i]<<" ";

}

cout<<"\n";

for(int i=0;i<cls2.size();i++){

cout<<cls2[i]<<" ";

}

cout<<"\n";

return 0;

}

**K-Means 2D (2 clusters)**

from math import sqrt

def dist(l, m1, m2):

dm1=[]

dm2=[]

for i in range(len(l)):

dis = sqrt((l[i][0]-m1[0])\*\*2 + (l[i][1]-m1[1])\*\*2)

dm1.append(dis)

for i in range(len(l)):

dis = sqrt((l[i][0]-m2[0])\*\*2 + (l[i][1]-m2[1])\*\*2)

dm2.append(dis)

return dm1,dm2

d = {

0: {

'p': 'A',

'c': [1,1]

},

1: {

'p': 'B',

'c': [1,0]

},

2: {

'p': 'C',

'c': [0,2]

},

3: {

'p': 'D',

'c': [2,4]

},

4: {

'p': 'E',

'c': [3,5]

},

}

l = [[1,1],[1,0],[0,2],[2,4],[3,5]]

m1 = [1,1]

m2 = [1,0]

c1=[]

c2=[]

p1=[]

p2=[]

c=0

for j in range(10):

dm1, dm2 = dist(l, m1, m2)

for i in range(5):

if dm1[i] < dm2[i]:

c1.append(l[i])

else:

c2.append(l[i])

nm1 = m1

nm2 = m2

m1 = [sum(x)/len(x) for x in zip(\*c1)]

m2 = [sum(x)/len(x) for x in zip(\*c2)]

if m1==nm1 and m2==nm2:

print("Iterations: ",c)

break

c1.clear()

c2.clear()

c+=1

for k, v in d.items():

if v['c'] in c1:

p1.append(d[k]['p'])

else:

p2.append(d[k]['p'])

print(p1,c1)

print(p2,c2

**K-Means 2D (N clusters)**

| **Height** | **Weight** |
| --- | --- |
| **185** | **70** |
| **170** | **56** |
| **168** | **60** |
| **179** | **68** |
| **182** | **72** |
| **188** | **77** |
| **180** | **71** |
| **183** | **84** |
| **180** | **88** |
| **180** | **67** |
| **172** | **70** |

import math

#np.linalg.norm

def distn(l,k,m):

dl=[[] for x in range(k)]

for i in range(k):

for j in range(len(l)):

dis = math.dist(m[i],l[j])

dl[i].append(dis)

return dl

# k = int(input("Enter number of clusters: "))

k=5

nm = 0

d = [[185,70],[170,56],[168,60],[179,68],[182,72],[188,77],[180,71],[183,84],[180,88],[180,67],[172,70]]

m = [d[i] for i in range(k)]

c = [[] for x in range(k)]

while nm != m:

dis = distn(d, k, m)

c = [[] for x in range(k)]

for i in range(11):

minl=min(dis,key=lambda x:x[i])

ind = dis.index(minl)

c[ind].append(d[i])

nm = m

for j in range(k):

m1 = [sum(x) / len(x) for x in zip(\*c[j])]

m.append(m1)

if nm == m:

break

for i in range(len(c)):

print("Cluster",i+1," -",c[i])

**Naive Bayesian Classification**

| **Fruit** | **Yellow** | **Sweet** | **Long** | **Total** |
| --- | --- | --- | --- | --- |
| **Orange** | **350** | **450** | **0** | **650** |
| **Banana** | **400** | **300** | **350** | **400** |
| **Other** | **50** | **100** | **50** | **150** |
| **Total** | **800** | **850** | **400** | **1200** |

import pandas as pd

import numpy as np

data = pd.read\_csv('Fruit.csv')

l = dict()

val = [1]\*(len(data.values) - 1)

ran = len(data.values[0]) - 1

for i in range(len(val)):

for j in range(1,ran):

val[i] \*= (data.values[i][j])/(data.values[i][4])

l[f"{data.values[i][0]}"] = val[i]

# print(max(val))

# print(data.values[val.index(max(val))][0])

print(l)

**Naive Bayesian Classification(Car Dataset)**

**Colour 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**

**Red SUV Imported No**

**Red Sports Imported Yes**

import pandas as pd

data = pd.read\_csv('NB1.csv')

# inp = list(map(str, input().split()))

inp = ["Red","Sports","Domestic"]

l = [x for x in data.keys()]

total = len(data.values)

totaly = data[data[f'{l[-1]}'] == "Yes"].count()[0]

proby = totaly/total

totaln = total - totaly

probn = 1 - proby

py,pn = 1,1

for i in range(len(l) - 1):

daty = data[data[f'{l[i]}'] == f"{inp[i]}"][data[f'{l[-1]}'] == "Yes"].count()[0]

datn = data[data[f'{l[i]}'] == f"{inp[i]}"][data[f'{l[-1]}'] == "No"].count()[0]

py \*= daty/totaly

pn \*= datn/totaln

print(py\*proby)

print(pn\*probn)

if py\*proby > pn\*probn:

print("Yes")

else:

print("No")

—--------------------------------------

import pandas as pd

data = pd.read\_csv('Comps.csv')

# inp = list(map(str, input().split()))

inp = ["<=30","High","No","Fair"]

l = [x for x in data.keys()]

total = len(data.values)

cl = list(set(data[f'{l[-1]}']))

totlst = [data[data[f'{l[-1]}'] == f"{cl[i]}"].count()[0] for i in range(len(cl))]

problst = [totlst[i]/total for i in range(len(cl))]

dat = []

prob = [1]\*len(cl)

for i in range(len(l)-1):

for j in range(len(cl)):

dat.append(data[data[f'{l[i]}'] == f"{inp[i]}"][data[f'{l[-1]}'] == f"{cl[j]}"].count()[0])

for k in range(len(cl)):

prob[k] \*= dat[k]/totlst[k]

dat.clear()

prob = [prob[i]\*problst[i] for i in range(len(cl))]

print("Prediction:",cl[prob.index(max(prob))])

**Linear Regression**

| **x** | **y** |
| --- | --- |
| **0** | **1** |
| **1** | **3** |
| **2** | **2** |
| **3** | **5** |
| **4** | **7** |
| **5** | **8** |
| **6** | **8** |
| **7** | **9** |
| **8** | **10** |
| **9** | **12** |

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data = pd.read\_csv('LR.csv')

n = len(data.values)

x,y,xy,xsq,p,e = [],[],[],[],[],[]

sxy = [sum(x) for x in zip(\*data.values)]

for i in data.values:

x.append(i[0])

y.append(i[1])

xy.append(i[0]\*i[1])

xsq.append(i[0]\*\*2)

a = ((sxy[1]\*sum(xsq)) - (sxy[0]\*sum(xy)))/((n\*sum(xsq)) - sxy[0]\*\*2)

b = ((n\*sum(xy)) - (sxy[0]\*sxy[1]))/((n\*sum(xsq)) - sxy[0]\*\*2)

for i in range(n):

p.append(b\*data.values[i][0] + a)

# e.append(abs(p[i]-data.values[i][1]))

newx = int(input("Enter a value of x: "))

predp = newx \* b + a

print(f"Predicted y of value {newx} is {predp}")

xp = np.array(x)

yp = np.array(p)

nyp = np.array(y)

plt.plot(xp,yp)

plt.scatter(xp,nyp)

plt.show()

**KNN**

| **x0** | **x1** | **y** |
| --- | --- | --- |
| **5.3** | **3.7** | **Setosa** |
| **5.1** | **3.8** | **Setosa** |
| **7.2** | **3** | **Virginica** |
| **5.4** | **3.4** | **Setosa** |
| **5.1** | **3.3** | **Setosa** |
| **5.4** | **3.9** | **Setosa** |
| **7.4** | **2.8** | **Virginica** |
| **6.1** | **2.8** | **Verscicolor** |
| **7.3** | **2.9** | **Virginica** |
| **6** | **2.7** | **Verscicolor** |
| **5.8** | **2.8** | **Virginica** |
| **6.3** | **2.3** | **Verscicolor** |
| **5.1** | **2.5** | **Verscicolor** |
| **6.3** | **2.5** | **Verscicolor** |
| **5.5** | **2.4** | **Verscicolor** |

import math

import pandas as pd

c=[]

d=[]

k=5

p = [6,3]

data = pd.read\_csv('knn.csv')

len=len(data.values)

cl = [x for x in data['y']]

for i in range(2):

a = [x for x in data[f'x{i}']]

c.append(a)

c = [x for x in zip(\*c)]

for i in c:

d.append(math.dist(p,i))

cld = [x for x in zip(d,cl)]

cld.sort()

for i in range(len-k):

cld.pop()

print(max(cld, key=cld.count))

**Agglomerative Clustering(Complete Link)**

import math

dataset = [[0.40, 0.53], [0.22, 0.38], [0.35, 0.32], [0.26, 0.19], [0.08, 0.41], [0.45, 0.30]]

# diff = []

l = []

diff = [[0,9,3,6,11],[9,0,7,5,10],[3,7,0,9,2],[6,5,9,0,8],[11,10,2,8,0]]

# for i in range(len(dataset)):

# diff.append([])

# for j in range(len(dataset)):

# diff[i].append(math.sqrt((dataset[i][0] - dataset[j][0]) \*\* 2 + (dataset[i][1] - dataset[j][1]) \*\* 2))

for i in range(len(diff)):

for j in range(len(diff[i])):

if diff[i][j] == 0:

diff[i][j] = 999

ITERATION = 0

while (len(diff) > 1):

print("ITERATION:", ITERATION)

MIN = min(min(diff,key=min))

for i in range(len(diff)):

for j in range(len(diff[i])):

if diff[i][j] == MIN:

m = i

n = j

break

mergeRow1 = [k for k in diff[m]]

mergeRow2 = [k for k in diff[n]]

for i in range(len(mergeRow1)):

l.append(max(mergeRow1[i], mergeRow2[i]))

for i in diff:

data1 = i[n]

data2 = i[m]

i.remove(data1)

i.remove(data2)

if n > m:

del diff[n]

del diff[m]

del l[n]

del l[m]

else:

del diff[m]

del diff[n]

del l[m]

del l[n]

for i in range(len(diff)):

diff[i].append(l[i])

l.append(999)

diff.append([i for i in l])

print("Merged rows are: ", m + 1, ' ', n + 1)

print("Distance Table after merge: ")

for i in diff:

print(i)

print()

l.clear()

mergeRow1.clear()

mergeRow2.clear()

ITERATION += 1

**Density based Spatial Clustering**

from math import dist

# p = [1,2,3,4,5,12,15]

# l = [[0,1,2,3,4,11,14],

# [1,0,1,2,3,10,13],

# [2,1,0,1,2,9,12],

# [3,2,1,0,1,8,11],

# [4,3,2,1,0,7,10],

# [11,10,9,8,7,0,3],

# [14,13,12,11,10,3,0]]

p = [[2,10],[2,5],[8,4],[5,8],[7,5],[6,4],[1,2],[4,9]]

leng = len(p)

l = [[] for i in range(leng)]

for i in range(leng):

for j in range(leng):

l[i].append(dist(p[i],p[j]))

min=4

e=3

c=[0]\*leng

for x in range(leng):

for y in range(leng):

if l[x][y]<=e:

c[y]+=1

for i in range(len(c)):

if c[i]>=min:

print(f"{p[i]}: Core")

elif 1 < c[i] < min:

print(f"{p[i]}: Border")

else:

print(f"{p[i]}: Noise")

**ID3**

from numpy import log2 as log

import pandas as pd

import numpy as np

import pprint

def find\_entropy(df):

Class = df.keys()[-1]

entropy = 0

values = df[Class].unique()

for value in values:

fraction = df[Class].value\_counts()[value]/len(df[Class])

entropy += -fraction\*np.log2(fraction)

return entropy

def find\_entropy\_attribute(df,attribute):

Class = df.keys()[-1]

target\_variables = df[Class].unique()

variables = df[attribute].unique()

entropy2 = 0

for variable in variables:

entropy = 0

for target\_variable in target\_variables:

num = len(df[attribute][df[attribute]==variable][df[Class] ==target\_variable])

den = len(df[attribute][df[attribute]==variable])

fraction = num/(den+eps)

entropy += -fraction\*log(fraction+eps)

fraction2 = den/len(df)

entropy2 += -fraction2\*entropy

return abs(entropy2)

def find\_winner(df):

gain = []

for key in df.keys()[:-1]:

gain.append(find\_entropy(df)-find\_entropy\_attribute(df,key))

return df.keys()[:-1][np.argmax(gain)]

def get\_subtable(df, node,value):

return df[df[node] == value].reset\_index(drop=True)

def buildTree(df,tree=None):

Class = df.keys()[-1]

node = find\_winner(df)

attValue = np.unique(df[node])

if tree is None:

tree={}

tree[node] = {}

for value in attValue:

subtable = get\_subtable(df,node,value)

clValue,counts = np.unique(subtable[Class],return\_counts=True)

if len(counts)==1:

tree[node][value] = clValue[0]

else:

tree[node][value] = buildTree(subtable)

return tree

eps = np.finfo(float).eps

df = pd.read\_csv('Comps.csv')

print(df)

tree= buildTree(df)

pprint.pprint(tree)

**Apriori Itemsets**

import numpy as np

import pandas as pd

from itertools import combinations

df = pd.read\_csv('Apriori Dataset - Sheet1.csv')

dff = pd.DataFrame(df)

di = dff.Items.str.split(',')

suppCount = int(input("Enter the minimum support count : "))

Confidence = int(input("Enter the minimum confidence : "))

print("DATASET:")

print(df)

print()

# First set of Frequent Items L1

def validate(data, sc):

data1 = pd.DataFrame()

for i in range(len(data)):

if data.iloc[i, 1] >= sc:

temp = {'Itemsets': data.iloc[i, 0], 'supp\_count': data.iloc[i, 1]}

data1 = data1.append(temp, ignore\_index=True)

return data1

# To create combinations

def combinations1(list\_of\_items):

itemsets = []

i = 1

for entry in list\_of\_items:

proceding\_items = list\_of\_items[i:]

for item in proceding\_items:

if (type(item) is str):

if entry != item:

tuples = (entry, item)

itemsets.append(tuples)

else:

if entry[0:-1] == item[0:-1]:

tuples = entry + item[1:]

itemsets.append(tuples)

i = i + 1

if (len(itemsets) == 0):

return None

return itemsets

# To calculate freq of Itemset

# Validate is for single item,this one for multiple

def count\_itemsets(di, itemsets):

count\_itemset = {}

for row in di:

a = set(row)

for itemset in itemsets:

b = set(itemset)

if (b.intersection(a)) == b:

if itemset in count\_itemset:

count\_itemset[itemset] += 1

else:

count\_itemset[itemset] = 1

data = pd.DataFrame()

data['ItemSets'] = count\_itemset.keys()

data['supp\_count'] = count\_itemset.values()

return data

# Obviously to print rules

def rules(Itemsets, Confidence):

va = [] # if A->BC va is A

vb = [] # if A->BC va is BC

vc = [] # Confidence of above thingy

# Above 3 only contain strong rules

print("Rules : ")

for l in Itemsets:

l = frozenset(list(l))

c = [frozenset(q) for q in combinations(l, len(l) - 1)]

for a in c:

# to split a set ABC into A->BC & BC->A

b = l - a # b is The second part a is first part of arrow

ab = l

sab = 0 # support of ab

sa = 0 # support of a

sb = 0 # support of b

for q in di:

temp = set(q)

if (a.issubset(temp)):

sa += 1

if b.issubset(temp):

sb += 1

if (ab.issubset(temp)):

sab += 1

temp = sab / sa \* 100

# temp BC->A

print(list(a), "-->>", list(b), "Confidence : ", temp)

temp1 = sab / sb \* 100 # Formula for Confidence

# temp1 A->BC

print(list(b), "-->>", list(a), "Confidence : ", temp1)

if temp >= Confidence:

va.append(list(a))

vb.append(list(b))

vc.append(temp)

if (temp1 >= Confidence):

va.append(list(b))

vb.append(list(a))

vc.append(temp1)

# Obviously to print strong rules

print()

print("Valid Rules : ")

for i in range(len(va)):

print(va[i], "-->>", vb[i], "Confidence : ", vc[i])

count\_item = {}

for row in di:

for i in range(len(row)):

if (row[i] in count\_item):

count\_item[row[i]] += 1

else:

count\_item[row[i]] = 1

data = pd.DataFrame()

data['ItemSets'] = count\_item.keys()

data['supp\_count'] = count\_item.values()

freq = pd.DataFrame()

while (len(data) != 0):

data = validate(data, suppCount)

if (len(data) > 1) or (len(data) == 1 and int(data.supp\_count >= suppCount)):

freq = data

if (len(data) != 0):

Itemsets = combinations1(data.Itemsets)

data = count\_itemsets(di, Itemsets)

print()

print("ItemSet ")

print(freq)

print()

rules(freq.Itemsets, Confidence)