Lab Work:

Lab#1: Data normalization

#1.1 Implementation of Standard scaler

import numpy as np

import pandas as pd

class StandardNorm:

  def scale(self,d):

    for i in d.columns:

      mean=d[i].mean()

      sd=d[i].std()

      d[i]=(d[i]-mean)/sd

    return d

data=pd.DataFrame([[45000,42],[32000,26],[58000,48],[37000,32]],columns=['Salary','Age'])

print("Original Data")

print(data)

s=StandardNorm()

df=s.scale(data)

print("\nScaled Data")

print(df)

#1.2 Implementation of min-max scaler

import pandas as pd

class MinMaxNorm:

  def scale(self,d):

    for i in d.columns:

      min=d[i].min()

      max=d[i].max()

      d[i]=(d[i]-min)/(max-min)

    return d

data=pd.DataFrame([[45000,42],[32000,26],[58000,48],[37000,32]],columns=['Salary','Age'])

print("Original Data")

print(data)

s=MinMaxNorm()

print("\nScaled Data")

df=s.scale(data)

print(df)

1.3 Generate the similarity and dissimilarity matrix (proximity matrix) for binary, nominal and ordinal data.

Lab #2 Data cleaning and transformation

#Lab 2.1 : Data Cleaning (handling missing value and handling inconsistent value

import pandas as pd

import numpy as np

data = pd.read\_csv('/content/drive/My Drive/Data/employees.csv')

print("Original Data")

print(data[0:25])

# Removing missing values

data=data.dropna(axis=0)

# Removing duplicate rows

data.drop\_duplicates(keep='first',inplace=True)

# Removing column Boonus %

del data['Bonus %']

# Correcting Inconsitencies among values

data['Team']=data['Team'].str.replace('Fin','Finance')

data['Team']=data['Team'].str.replace('Mkt','Marketing')

data['Team']=data['Team'].str.replace('Financeance','Finance')

print("Cleaned Data")

print(data[0:25])

#Lab 2.2 : Data Cleaning (Handling missing numerical value with mean and interpolation

import pandas as pd

import numpy as np

data = pd.read\_csv('/content/drive/My Drive/Data/employees.csv')

print("Original Data")

print(data[0:25])

# Filling missing values with mean

data['Salary']=data['Salary'].fillna(data['Salary'].mean())

print("Cleaned Data")

print(data[0:25])

data = pd.read\_csv('/content/drive/My Drive/Data/employees.csv')

print("Original Data")

print(data[0:25])

# Filling missing values with interpolate

data['Salary']=data['Salary'].interpolate(method="linear")

print("Cleaned Data")

print(data[0:25])

#Lab 2.3 : Data Transformation ( derive age attribute from DOB)

import pandas as pd

data=pd.read\_csv('F:/DWDM/lab/python/student.csv')

print("Original Data")

print(data[0:5])

data["DOB"] = pd.to\_datetime(data["DOB"])

today = pd.to\_datetime("today").date()

data["AGE"] = (today - data["DOB"].dt.date).dt.days / 365.25

data["AGE"] = data["AGE"].astype(int)

#data.to\_csv("F:/DWDM/lab/python/data\_with\_age.csv", index=False)

#data.to\_csv("F:/DWDM/lab/python/student.csv", index=True, mode="w")

print("Transformed Data")

#this will print with index

#print(data[0:5])

#print without index

print(data.to\_string(index=False))

LAB#3: Clustering

#K-means Clustering

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data=100\*np.random.rand(1000,2)

print(\*data)

km=KMeans(n\_clusters=3,init='random')

km.fit(data)

centers = km.cluster\_centers\_

labels = km.labels\_

print("Cluster Centers:",\*centers)

print("Cluster Labels:",\*labels)

colors = ["r","g","b"]

markers=["+","x","\*"]

for i in range(len(data)):

    plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])

plt.scatter(centers[:, 0],centers[:, 1], marker = "o", s=50, linewidths = 5)

plt.show()

#Mini-Batch K-means Clustering

import time

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import MiniBatchKMeans

data=100\*np.random.rand(10000,2)

print(\*data)

mbk=MiniBatchKMeans(n\_clusters=5,init='random', batch\_size=3000)

t0= time.time()

mbk.fit(data)

t1= time.time()

tt=t1-t0

print("Total Time:",tt)

cents = mbk.cluster\_centers\_

labels = mbk.labels\_

print("Cluster Centers:",\*cents)

print("Labels:",\*labels)

colors = ["g","r","b",'y','m']

markers=["+","x","\*",'.','d']

for i in range(len(data)):

    plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])

plt.scatter(cents[:, 0],cents[:, 1], marker = "o", s=50, linewidths = 5)

plt.show()

#5.1 Clustering Iris data using KMedoids

from sklearn.datasets import load\_iris

import numpy as np

import matplotlib.pyplot as plt

from sklearn\_extra.cluster import KMedoids

from sklearn.preprocessing import StandardScaler

from sklearn import metrics

iris\_data=load\_iris()

x=iris\_data.data

y=iris\_data.target

print(\*x)

print("Actual Group:",\*y)

sc=StandardScaler()

sc.fit(x)

sx=sc.transform(x)

km=KMedoids(n\_clusters=3)

km.fit(sx)

py=km.fit\_predict(sx)

print("Predicted Group",\*py)

fig = plt.figure(figsize = (12, 8))

ax = fig.add\_subplot(111, projection='3d')

colors = ["g","r","b"]

markers=["+","x","\*"]

for i in range(len(sx)):

  ax.scatter(sx[i][0], sx[i][1], sx[i][2],color=colors[py[i]], marker=markers[py[i]])

plt.show()

hs=metrics.homogeneity\_score(y, py)  #Homoginity Score

print("Homogeniety Score:",hs)

sc=metrics.silhouette\_score(sx, py, metric='euclidean') #Silhouette Coefficient

print("Silhouette Coefficient:",sc)

#5.2 Clustering Iris data using Agglomerative

from sklearn.datasets import load\_iris

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from sklearn.preprocessing import StandardScaler

from sklearn import metrics

iris\_data=load\_iris()

x=iris\_data.data

y=iris\_data.target

sc=StandardScaler()

sc.fit(x)

sx=sc.transform(x)

ac=AgglomerativeClustering(n\_clusters=3)

ac.fit(sx)

py=ac.fit\_predict(sx)

fig = plt.figure(figsize = (12, 8))

ax = fig.add\_subplot(111, projection='3d')

colors = ["g","r","b"]

markers=["+","x","\*"]

for i in range(len(x)):

  ax.scatter(sx[i][0], sx[i][1], sx[i][2],color=colors[py[i]], marker=markers[py[i]])

plt.show()

hs=metrics.homogeneity\_score(y, py)  #Homoginity Score

print("Homogeniety Score:",hs)

sc=metrics.silhouette\_score(sx, py, metric='euclidean') #Silhouette Coefficient

print("Silhouette Coefficient:",sc)

Lab#4: Classification and Prediction

#Diabetes Prediction Using Naive Bayes Classifier

import pandas as pd

from sklearn import metrics

from sklearn.naive\_bayes import GaussianNB

dataset = pd.read\_csv('/content/drive/My Drive/Data/Diabetes.csv')

split = int(len(dataset)\*0.7)

train, test = dataset[:split], dataset[split:]

p = train['Pragnency'].values

g = train['Glucose'].values

bp= train['Blod Pressure'].values

st= train['Skin Thikness'].values

ins= train['Insulin'].values

bmi= train['BMI'].values

dfp= train['DFP'].values

a= train['Age'].values

d= train['Diabetes'].values

trainfeatures=zip(p,g,bp,st,ins,bmi,dfp,a)

traininput=list(trainfeatures)

#print(traininput)

model = GaussianNB()

model.fit(traininput,d)

p = test['Pragnency'].values

g = test['Glucose'].values

bp= test['Blod Pressure'].values

st= test['Skin Thikness'].values

ins= test['Insulin'].values

bmi= test['BMI'].values

dpf= test['DFP'].values

a= test['Age'].values

d= test['Diabetes'].values

testfeatures=zip(p,g,bp,st,ins,bmi,dpf,a)

testinput=list(testfeatures)

predicted= model.predict(testinput)

print("Actual Class:   ", \*d)

print("Predicted Class:", \*predicted)

print("Confusion Matrix")

print(metrics.confusion\_matrix(d, predicted))

print("\*\*\*\*\*\*\*\*Classifiaction Measures\*\*\*\*\*\*\*\*\*")

print("Accuracy:",metrics.accuracy\_score(d,predicted))

print("Recall:",metrics.recall\_score(d,predicted))

print("Precision:",metrics.precision\_score(d,predicted))

print("F1-Score:",metrics.f1\_score(d,predicted))

import pandas as pd

from sklearn import metrics

from sklearn.tree import DecisionTreeClassifier

dataset = pd.read\_csv('/content/drive/My Drive/Data/Diabetes.csv')

split = int(len(dataset)\*0.7)

train, test = dataset[:split], dataset[split:]

p = train['Pragnency'].values

g = train['Glucose'].values

bp= train['Blod Pressure'].values

st= train['Skin Thikness'].values

ins= train['Insulin'].values

bmi= train['BMI'].values

dpf= train['DFP'].values

a= train['Age'].values

d= train['Diabetes'].values

trainfeatures=zip(p,g,bp,st,ins,bmi,dpf,a)

traininput=list(trainfeatures)

model = DecisionTreeClassifier(criterion = "entropy", max\_depth=8)

model.fit(traininput,d)

p = test['Pragnency'].values

g = test['Glucose'].values

bp= test['Blod Pressure'].values

st= test['Skin Thikness'].values

ins= test['Insulin'].values

bmi= test['BMI'].values

dpf= test['DFP'].values

a= test['Age'].values

d= test['Diabetes'].values

testfeatures=zip(p,g,bp,st,ins,bmi,dpf,a)

testinput=list(testfeatures)

predicted= model.predict(testinput)

print("Actual Class:   ",\*d)

print("Predicted Class:",\*predicted)

print("Confusion Matrix")

print(metrics.confusion\_matrix(d, predicted))

print("\*\*\*\*\*\*\*\*Classifiaction Measures\*\*\*\*\*\*\*\*\*")

acc=metrics.accuracy\_score(d,predicted)

f1=metrics.f1\_score(d, predicted)

rec=metrics.recall\_score(d, predicted)

pre= metrics.precision\_score(d, predicted)

print("Accuracy Score:",acc)

print("Recall Score:",rec)

print("Precision-Score:",pre)

print("F1-Score:",f1)