

**DATA MINING (IE 6318) HW3**

A drawing of a cartoon character

Description automatically generated

**Submittied by**

**ANUJ ANANDAKUMAR**

**1001746608**

**(10/16/2020)**

**Libraries used for the assignment**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.pyplot import figure

from mpl\_toolkits.mplot3d import Axes3D

import seaborn as sns

%matplotlib inline

import yellowbrick as yb

from yellowbrick.features import ParallelCoordinates

from yellowbrick.datasets import load\_occupancy

import plotly.graph\_objects as go

from sklearn import preprocessing

import plotly.express as px

from math import \*

from decimal import Decimal

from math import \*

from decimal import Decimal

import numpy as np

import pandas as pd

from math import \*

from decimal import Decimal

import scipy as stats

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

data=pd.read\_csv('hw2\_iris.csv')

data.head()

Table

Description automatically generated

**Explore the Iris dataset and report the following:**

**1) 2D scatter plots of the four features (creates a matrix of scatter plots for each pair of the features)**

fig = px.scatter\_matrix(data,

dimensions=["sepal length","sepal width","petal width"],

color="class")

fig.show()

Chart, scatter chart

Description automatically generated**2) 3D scatter plot of three features: sepal length, sepal width, petal width.**

fig = plt.figure(num=None, figsize=(8, 6), dpi=80)

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

x = data['sepal length']

y = data['sepal width']

z = data['petallength']

ax.scatter(x,y,z,c='r',marker = 'o')

ax.set\_xlabel('sepal length')

ax.set\_ylabel('sepal width')

ax.set\_zlabel('petallength')

plt.show()

Chart, scatter chart

Description automatically generated

**3) Visualization of the feature matrix (column 1-4) as an image.**

irisdataset= pd.read\_csv("hw2\_iris.csv", index\_col='class')

plt.pcolor(irisdataset)

sns.heatmap(irisdataset, annot=True)

A picture containing door

Description automatically generated

**4) For each class, generate histograms for the four features.**

plt.figure(num=None, figsize=(8, 6), dpi=80)

data.hist()

plt.show()

Chart, histogram

Description automatically generated

**5) For each class, generate boxplots of the four features.**

fig, axs = plt.subplots(2,2,figsize=(10,7))

x = data['class']

y = data['sepal length']

sns.boxplot(x = data['class'], y = data['sepal length'], ax = axs[0,0]);

sns.boxplot(x = data['class'], y = data['sepal width'], ax = axs[0,1]);

sns.boxplot(x = data['class'], y = data['petallength'], ax = axs[1,0]);

sns.boxplot(x = data['class'], y = data['petal width'], ax = axs[1,1]);

fig.tight\_layout(pad=1.0);

Chart, box and whisker chart

Description automatically generated

**6) Calculate the correlation matrix of the four features**

df = pd.DataFrame(data,columns=['sepal length','sepal width','petallength','petal width'])

corrMatrix = df.corr()

print (corrMatrix)

Text

Description automatically generated

7) Visualize the correlation matrix as an image.

corr = data.corr()

sns.heatmap(corr,

xticklabels=corr.columns.values,

yticklabels=corr.columns.values)

plt.show()

Shape, square

Description automatically generated

**8) Create a parallel coordinate plot of the four features**

data2=preprocessing.LabelEncoder()

speciesId=data2.fit\_transform(list(data["class"]))

speciesId

data["speciesId"]=speciesId

data

Table

Description automatically generated

fig = px.parallel\_coordinates(data,color="speciesId", labels={"class": "Species","sepal width": "Sepal Width", "sepal length": "Sepal Length",

"petal width": "Petal Width", "petallength": "Petal Length", },

color\_continuous\_scale=px.colors.diverging.Tealrose,

color\_continuous\_midpoint=2)

fig.show()

Chart, radar chart

Description automatically generated

2. Practice Data Distance Measures

**1) Make a function for Minkowski Distance. (3 function inputs: vector A, vector B, and order r)**

def minkowski(A,B,r):

vari1=np.array(A)

vari2=np.array(B)

vari3=np.array(vari1) - np.array(vari2)

sum=0

for i in range(4):

sum+=pow(abs(vari3[i]),r)

dist = pow(sum,1/r)

return round(dist,3)

**2) Make a function for T-statistics Distance. (3 function inputs: time series A, time series B)**

def tstatistics(A,B):

mean=abs(np.mean(tstat1)-np.mean(tstat2))

tstat3=tstat1-tstat2

variance=np.var(tstat3)

tstatdist=mean/variance

return tstatdist

**3) Make a function for Mahalanobis Distance. (3 function inputs: vector A, vector B, and covariance matrix M)**

def mahalanobis(X,Y,M):

var1=np.array(X)

var2=np.array(Y)

var3=np.array(var1) - np.array(var2)

var4=np.transpose(var3)

M\_inv=np.linalg.inv(M)

a=np.matmul(var3,M\_inv)

b=np.matmul(a,var4)

r = np.sqrt(b)

return r

**3. For a new iris data sample S with a feature vector of [5.0000, 3.5000, 1.4600, 0.2540], calculate the distances between the new sample and the 150 samples in the iris dataset using the distance functions you made:**

**1) Calculate Minkowski distances with r = 1, 2, 10, respectively, and plot the obtained distances.**

bluba = data.to\_numpy()

duluba = bluba[:,:4]

x = [5.0000, 3.5000, 1.4600, 0.2540]

distance = np.zeros(shape=(150,))

for j in range(150):

y=duluba[j]

z=1

distance[j]=minkowski(x,y,z)

print(distance)

plt.figure(figsize = (8, 6))

plt.plot(distance)

plt.xlabel('Sample Number')

plt.ylabel('Distance')

plt.show()

**2) Calculate Mahalanobis distances and plot the obtained distances.**

bluba = data.to\_numpy()

duluba = bluba[:,:4]

x = [5.0000, 3.5000, 1.4600, 0.2540]

distance = np.zeros(shape=(150,))

for j in range(150):

y=duluba[j]

z=1/2/10 # can use any one.

distance[j]=minkowski(x,y,z)

print(distance)

plt.figure(figsize = (8, 6))

plt.plot(distance)

plt.xlabel('lot Number')

plt.ylabel('distance from the lot')

plt.show()

r=1

Chart

Description automatically generated

r=2

Chart

Description automatically generated

r=10

Chart

Description automatically generated

**2) Calculate Mahalanobis distances and plot the obtained distances.**

dataMahal=data2.copy()

def mahalanobis(x, datanew, cov=None):

x\_mu = x - np.mean(datanew)

if not cov:

cov = np.cov(datanew.values.T)

inv\_covmat = np.linalg.inv(cov)

left = np.dot(x\_mu, inv\_covmat)

mahal = np.dot(left, x\_mu.T)

return mahal.diagonal()

#create new column in dataframe that contains Mahalanobis distance for each row

x= [5.0000, 3.5000, 1.4600, 0.2540]

dataMahal['mahalanobis'] = mahalanobis(x=data2, datanew=data2[['sepal length', 'sepal width', 'petallength', 'petal width']])

#display first five rows of dataframe

dataMahal

Table

Description automatically generated

A=dataMahal['mahalanobis'].to\_numpy()

plt.figure(figsize = (8, 6))

plt.plot(A)

plt.xlabel('lot no')

plt.ylabel('Distance')

plt.show()

Chart

Description automatically generated

**4. Feature matrix normalization**

**1) Normalize the feature matrix of the IRIS dataset such that each feature has a mean of 0 and a standard deviation of 1 after normalization.**

datanorm=data.to\_numpy()

features=datanorm[:,:4]

norm=preprocessing.scale(features)

print(norm.round(2))

A picture containing shape

Description automatically generated

**2) Calculate the correlation matrix of the four features after normalization**.

df = pd.DataFrame(norm)

corrMatrix = df.corr()

print (corrMatrix)

Text

Description automatically generated

**3) Compare the correlation matrix before and after normalization. If they are the same?**

Text

Description automatically generatedText

Description automatically generated

Yes, they are same.

**5. Principle Component Analysis (PCA) on the IRIS dataset**

**1) Create 2D scatter plots of each pair of the four components**

features = ["sepal length","sepal width", "petallength","petal width"]

pca = PCA()

components = pca.fit\_transform(data[features])

labels = {

str(i): f"PC {i+1} ({var:.1f}%)"

for i, var in enumerate(pca.explained\_variance\_ratio\_ \* 100)

}

fig = px.scatter\_matrix(

components,

labels=labels,

dimensions=range(4),

color=data["class"]

)

fig.update\_traces(diagonal\_visible=False)

fig.show()

A picture containing qr code

Description automatically generated

**2) 3D scatter plot of the first three components**

df = px.data.iris()

X = df[['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']]

pca = PCA(n\_components=3)

components = pca.fit\_transform(X)

total\_var = pca.explained\_variance\_ratio\_.sum() \* 100

fig = px.scatter\_3d(

components, x=0, y=1, z=2, color=df['species'],

title=f'Total Explained Variance: {total\_var:.2f}%',

labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3'}

)

fig.show()

Chart, radar chart, scatter chart

Description automatically generated

**3) Obtain the variance of each component and visualize in a figure plot.**

iris\_dataset= pd.read\_csv("hw2\_iris.csv")

iris\_dataset.head()

classtypes = ["sepal length","sepal width", "petallength","petal width"]

a = iris\_dataset.loc[:, classtypes].values

b = iris\_dataset.loc[:,['class']].values

a = StandardScaler().fit\_transform(x)

pd.DataFrame(data = a, columns = features).head()

pca = PCA(n\_components=4)

principalComponents = pca.fit\_transform(x)

irisnewvar = pd.DataFrame(data = principalComponents

, columns = ['first component', 'second component', 'third component','fourth component'])

v1=np.array(irisnewvar['first component'])

v2=np.array(irisnewvar['second component'])

v3=np.array(irisnewvar['third component'])

v4=np.array(irisnewvar['fourth component'])

varianceone=np.var(v1)

variancetwo=np.var(v2)

variancethree=np.var(v3)

variancefour=np.var(v4)

print("Variance of first component:", varianceone, "\nVariance of second component :", variancetwo,

"\nVariance of third component :", variancethree, "\nVariance of fourth component :", variancefour)

Text

Description automatically generated

**4) Calculate the correlation matrix of the four components**

corrmatrix=irisnewvar.corr()

print(corrmatrix.round())

sns.heatmap(corrmatrix.round(), annot=True)

plt.show()

Graphical user interface, text

Description automatically generatedChart, waterfall chart

Description automatically generated

**6. For the dataset with two time series in the “hw2\_ts.txt” file, perform the following analysis:**

**1) Visualize the two time series in one figure plot.**

df = pd.read\_csv("hw2ts.csv")

plt.figure(figsize=(12,5))

plt.xlabel(' total time ')

ax1 = df.t1.plot(color='blue', grid=True, label='time series 1')

ax2 = df.t2.plot(color='red', grid=True, secondary\_y=True, label='timeseries 2 ')

h1, l1 = ax1.get\_legend\_handles\_labels()

h2, l2 = ax2.get\_legend\_handles\_labels()

plt.legend(h1+h2, l1+l2, loc=2)

plt.show()

Chart, line chart, histogram

Description automatically generated

**2) Calculate the T-statistics distance between the two time series using the function you made in 2.**

ts\_data= pd.read\_csv("hw2ts.csv")

ts1=np.array(ts\_data["t1"])

ts2=np.array(ts\_data["t2"])

def tstatistics(A,B):

mean=abs(np.mean(ts1)-np.mean(ts2))

ts3=ts1-ts2

var=np.var(ts3)

tdist=mean/var

return tdist

tstatistics(ts1,ts2)

0.10731262850806823

**3) Calculate the correlation of the two time series**

corrmatrix=ts\_data.corr()

print(corrmatrix)

sns.heatmap(corrmatrix, annot=True)

plt.show()

Graphical user interface, application

Description automatically generated