Pattern Recognition

**Name: Enrolment No: Semester: 7**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Definition** | **Date** | **Signature** | **Remarks** |
| **1** | To demonstrate use of preprocessing Machine Learning using Python Programming. |  |  |  |
| **2** | To implement Linear Regression on a sample dataset using Python Programming |  |  |  |
| **3** | To implement Polynomial Regression on a sample dataset using Python Programming. |  |  |  |
| **4** | To implement Decision Tree Regression on a sample dataset using Python Programming. |  |  |  |
| **5** | To implement Decision Tree Classification on a sample dataset using Python Programming. |  |  |  |
| **6** | To implement K-Nearest Neighbours Classification on a sample dataset using Python Programming. |  |  |  |
| **7** | To implement K-Means Clustering on a sample dataset using Python Programming. |  |  |  |
| **8** | To implement Hierarchical Clustering on a sample dataset using Python Programming. |  |  |  |
| **9** | To implement Handwritten Character Recognition using Artificial Neural Networks using Python Programming (using tensorflow/keras).To implement Handwritten Character Recognition using Artificial Neural Networks using Python Programming  (using tensorflow/keras). |  |  |  |
| **10** | To implement ANN using Python Programming without using any library. |  |  |  |
| **11** | To implement Customer Churn Prediction using ANN. |  |  |  |

`

PRACTICAL:1

Aim : To demonstrate use of preprocessing Machine Learning using Python Programming.

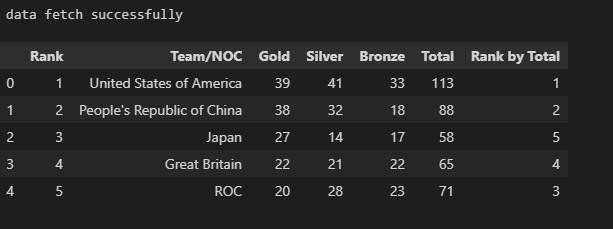
#### import pandas as pd import seaborn as sn

import matplotlib.pyplot as plt import numpy as np

data= pd.read\_csv('C:/Users/DELL/Downloads/archive (3)/Medals.csv',low\_memory= False)

print('data fetch successfully') data.head()

Output:-



x = data.iloc[:, :-1].values y = data.iloc[:, -1].values

print(x)

Output:-

[[1 'United States of America' 39 41 33 113]

[2 "People's Republic of China" 38 32 18 88]

[3 'Japan' 27 14 17 58]

[4 'Great Britain' 22 21 22 65]

[5 'ROC' 20 28 23 71]

[6 'Australia' 17 7 22 46]

[7 'Netherlands' 10 12 14 36]

[8 'France' 10 12 11 33]

[9 'Germany' 10 11 16 37]

`

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

print(y)

[10 'Italy' 10 10 20 40]

[11 'Canada' 7 6 11 24]

[12 'Brazil' 7 6 8 21]

[13 'New Zealand' 7 6 7 20]

[14 'Cuba' 7 3 5 15]

[15 'Hungary' 6 7 7 20]

[16 'Republic of Korea' 6 4 10 20]

[17 'Poland' 4 5 5 14]

[18 'Czech Republic' 4 4 3 11]

[19 'Kenya' 4 4 2 10]

[20 'Norway' 4 2 2 8]

[21 'Jamaica' 4 1 4 9]

[22 'Spain' 3 8 6 17]

[23 'Sweden' 3 6 0 9]

[24 'Switzerland' 3 4 6 13]

[25 'Denmark' 3 4 4 11]

Output:-

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [ 0 1 | | 4 3 2 | | 5 8 9 | | 7 6 10 11 12 15 12 12 16 19 20 22 21 14 | | | | | |
| 21 | 17 | 19 | 22 | 23 | 21 | 23 |  |  |  |  |  |
| 24 | 25 | 25 | 22 | 18 | 17 | 26 | 26 | 27 | 26 | 26 | 27 28 28 13 23 26 26 23 24 |
| 26 | 26 | 27 | 23 | 24 | 25 |  |  |  |  |  |  |
| 26 | 26 | 28 | 28 | 28 | 28 | 28 | 29 | 29 | 29 | 25 | 23 25 26 27 26 27 27 28 28 |
| 28 | 29 | 29 | 29 | 29 | 29 |  |  |  |  |  |  |
| 29 | 22 | 26 | 28 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29] |

from sklearn.impute import SimpleImputer

imputa = SimpleImputer(missing\_values='NAN', strategy='constant', fill\_value=" work-from-home")

imputa.fit(x[:, 1:3])

x[:, 1:3] = imputa.transform(x[:, 1:3]) print(x)

**Output:-**

`

[[1 'United States of America' 39 41 33 113]

[2 "People's Republic of China" 38 32 18 88]

[3 'Japan' 27 14 17 58]

[4 'Great Britain' 22 21 22 65]

[5 'ROC' 20 28 23 71]

[6 'Australia' 17 7 22 46]

[7 'Netherlands' 10 12 14 36]

[8 'France' 10 12 11 33]

[9 'Germany' 10 11 16 37]

[10 'Italy' 10 10 20 40]

[11 'Canada' 7 6 11 24]

[12 'Brazil' 7 6 8 21]

[13 'New Zealand' 7 6 7 20]

[14 'Cuba' 7 3 5 15]

[15 'Hungary' 6 7 7 20]

[16 'Republic of Korea' 6 4 10 20]

[17 'Poland' 4 5 5 14]

[18 'Czech Republic' 4 4 3 11]

[19 'Kenya' 4 4 2 10]

[20 'Norway' 4 2 2 8]

`

from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder

ENENNNO:O:--119900331133111166000055 SUSUB:B:--PRPR NNAME:AME:--DDhhartiarti SaSarrvvaiyaaiya

ct = ColumnTransformer(transformers = [('encoder', OneHotEncoder(),[1])], rema inder ='passthrough')

x = np.array(ct.fit\_transform(x))

print(x)

**Output:-**

**(0, 90) 1.0**

**(0, 93) 1.0**

**(0, 94) 39.0**

**(0, 95) 41.0**

**(0, 96) 33.0**

**(0, 97) 113.0**

**(1, 64) 1.0**

**(1, 93) 2.0**

**(1, 94) 38.0**

**(1, 95) 32.0**

**(1, 96) 18.0**

**(1, 97) 88.0**

**(2, 45) 1.0**

**(2, 93) 3.0**

**(2, 94) 27.0**

**(2, 95) 14.0**

**(2, 96) 17.0**

**(2, 97) 58.0**

**(3, 33) 1.0**

**(3, 93) 4.0**

**(3, 94) 22.0**

**(3, 95) 21.0**

**(3, 96) 22.0**

**(3, 97) 65.0**

**(4, 70) 1.0**

from sklearn.preprocessing import LabelEncoder le=LabelEncoder()

y = le.fit\_transform(y) print(y)

**output:-**

### [ 0 1 4 3 2 5 8 9 7 6 10 11 12 15 12 12 16 19 20 22 21 14 21 17 19 22 23 21 23

24 25 25 22 18 17 26 26 27 26 26 27 28 28 13 23 26 26 23 24 26 26 27 23 24 25

26 26 28 28 28 28 28 29 29 29 25 23 25 26 27 26 27 27 28 28 28 29 29 29 29 29

29 22 26 28 29 29 29 29 29 29 29 29]

`

from sklearn.model\_selection import train\_test\_split

#x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, rand om\_state=1)

x\_train=x x\_test=y

y\_trin=test\_size=0.2 y\_test=random\_state=1

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

x\_train[:, 3:] = sc.fit\_transform(x\_train[:, 3:])

x\_test[:, 3:] = sc.fit\_transform(x\_test[:, 3:])

`

PRACTICAL:2

Aim : To implement Linear Regression on a sample dataset using Python Programming

import matplotlib.pyplot as plt import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

diabetes\_X = diabetes\_X[:, np.newaxis, 2]

regr = linear\_model.LinearRegression() diabetes\_y\_train = diabetes\_y[:-20] diabetes\_y\_test = diabetes\_y[-20:]

regr.fit(diabetes\_X\_train, diabetes\_y\_train) diabetes\_y\_pred = regr.predict(diabetes\_X\_test) print('Coefficients: ', regr.coef\_

print('Mean squared error: %.2f'

% mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred))

Output:-

**Mean squared error: 2548.07**

print('Coefficient of determination: %.2f'

% r2\_score(diabetes\_y\_test, diabetes\_y\_pred))

Output:-

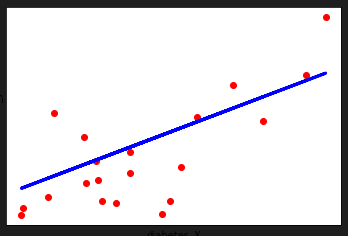
Coefficient of determination is 0.47

#### plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color='red') plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color='blue', linewidth=3) plt.xlabel("diabetes\_X")

plt.ylabel("diabetes\_y") plt.xticks(())

plt.yticks(()) plt.show()

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya



PRACTICAL:3

Aim: To implement Polynomial Regression on a sample dataset using Python Programming.

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:, 1:-1].values y = dataset.iloc[:, -1].values

from sklearn.linear\_model import LinearRegression lin\_reg = LinearRegression()

lin\_reg.fit(X, y)

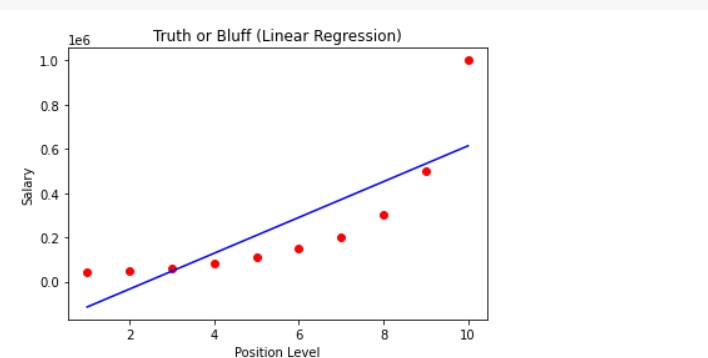
from sklearn.preprocessing import PolynomialFeatures poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X) lin\_reg\_2 = LinearRegression() lin\_reg\_2.fit(X\_poly, y) plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg.predict(X), color = 'blue') plt.title('Truth or Bluff (Linear Regression)') plt.xlabel('Position Level') plt.ylabel('Salary')

plt.show()

**OUTPUT**



plt.scatter(X, y, color = 'red')

plt.plot(X, lin\_reg\_2.predict(poly\_reg.fit\_transform(X)), color = 'blue') plt.title('Truth or Bluff (Polynomial Regression)')

plt.xlabel('Position level') plt.ylabel('Salary') plt.show()

X\_grid = np.arange(min(X), max(X), 0.1) X\_grid = X\_grid.reshape((len(X\_grid), 1)) plt.scatter(X, y, color = 'red')

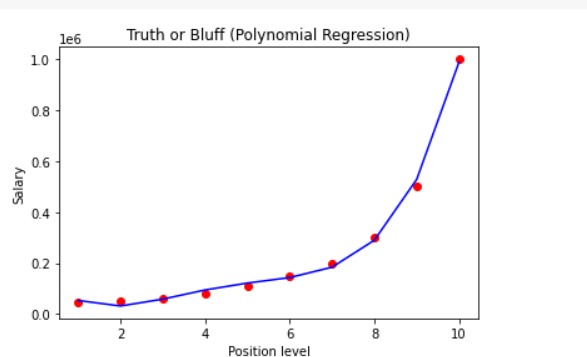
plt.plot(X\_grid, lin\_reg\_2.predict(poly\_reg.fit\_transform(X\_grid)), color = 'blue')

plt.title('Truth or Bluff (Polynomial Regression)') plt.xlabel('Position level')

plt.ylabel('Salary') plt.show()

lin\_reg\_2.predict(poly\_reg.fit\_transform([[6.5]]))

**OUTPUT**



DATA SET



PRACTICAL:4

Aim : To implement Decision Tree Regression on a sample dataset using Python Programming.

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

from sklearn.tree import DecisionTreeRegressor

dataset = np.array( [['Asset Flip', 100, 1000],

['Text Based', 500, 3000],

['Visual Novel', 1500, 5000],

['2D Pixel Art', 3500, 8000],

['2D Vector Art', 5000, 6500],

['Strategy', 6000, 7000],

['First Person Shooter', 8000, 15000],

['Simulator', 9500, 20000],

['Racing', 12000, 21000],

['RPG', 14000, 25000],

['Sandbox', 15500, 27000],

['Open-World', 16500, 30000],

['MMOFPS', 25000, 52000],

['MMORPG', 30000, 80000]

])

X = dataset[:, 1:2].astype(int) y = dataset[:, 2].astype(int)

regressor = DecisionTreeRegressor(random\_state=0) regressor.fit(X, y)

y\_pred = regressor.predict([[3750]])

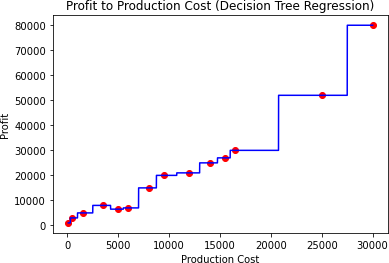
X\_grid = np.arange(min(X), max(X), 0.01) X\_grid = X\_grid.reshape((len(X\_grid), 1)) plt.scatter(X, y, color='red')

plt.plot(X\_grid, regressor.predict(X\_grid), color='blue') plt.title('Profit to Production Cost (Decision Tree Regression)') plt.xlabel('Production Cost')

plt.ylabel('Profit') plt.show()

#### output:-

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya



PRACTICAL:5

Aim : To implement Decision Tree classification on a sample dataset using Python Programming

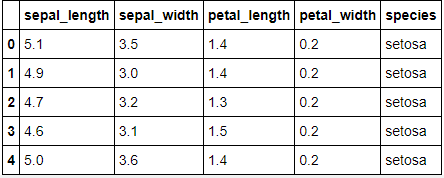
import numpy as np import pandas as pd

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

df = sns.load\_dataset('iris') df.head()



|  |
| --- |
| df.info() |
| Output:- |
| <class 'pandas.core.frame.DataFrame'> RangeIndex: 150 entries, 0 to 149 Data columns (total 5 columns):  # Column Non-Null Count Dtype |
| 1. sepal\_length 150 non-null float64 2. sepal\_width 150 non-null float64 3. petal\_length 150 non-null float64 4. petal\_width 150 non-null float64 5. species 150 non-null object dtypes: float64(4), object(1)   memory usage: 6.0+ KB |
| df.isnull().any() |
| Output:- |
| sepal\_length False  sepal\_width False |

petal\_length False petal\_width False

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

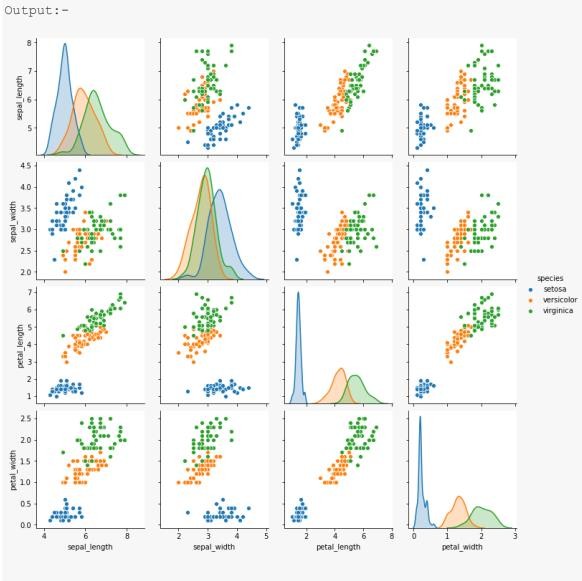
ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

species False dtype: bool

No Null values observed

sns.pairplot(data=df, hue = 'species') plt.savefig("pne.png")

Output:-



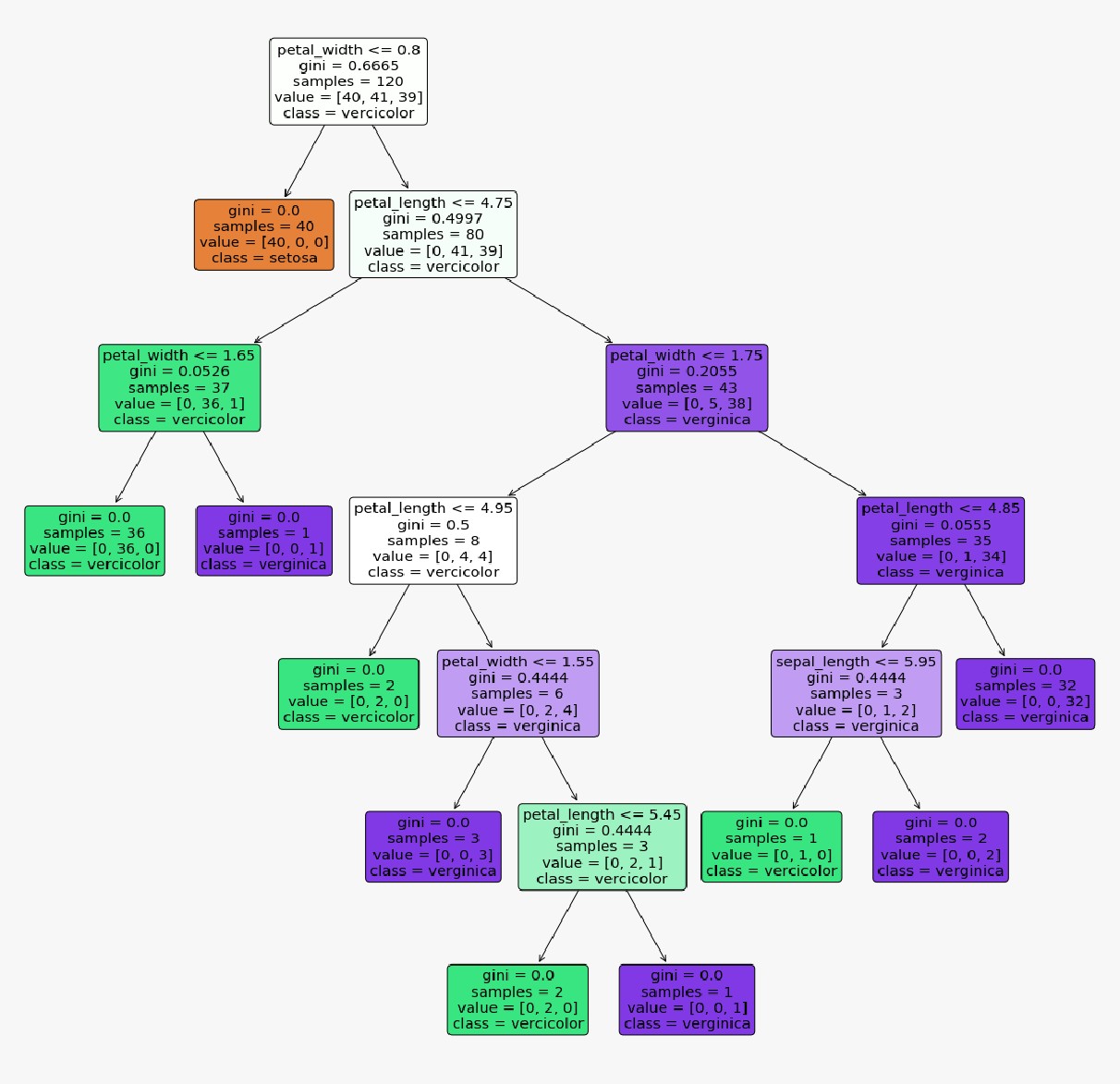
sns.heatmap(df.corr())plt.savefig("one.png")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
| dtree=DecisionTreeClassifier() dtree.fit(X\_train,y\_train) print('Decision Tree Classifer Created') | | | | |
| y\_pred = dtree.predict(X\_test)  print("Classification report - \n", classification\_report(y\_test,y\_pred)) | | | | |
| Output:- |  |  |  |  |
| Classification | report - precision |  |  |  |
|  | recall | f1-score | support |
| 0 | 1.00 | 1.00 | 1.00 | 10 |
| 1 | 1.00 | 1.00 | 1.00 | 9 |
| 2 | 1.00 | 1.00 | 1.00 | 11 |
| accuracy |  |  | 1.00 | 30 |
| macro avg | 1.00 | 1.00 | 1.00 | 30 |
| weighted avg | 1.00 | 1.00 | 1.00 | 30 |
| cm = confusion\_matrix(y\_test, y\_pred) plt.figure(figsize=(5,5))  sns.heatmap(data=cm,linewidths=.5, annot=True,square = True, cmap = 'Blues') plt.ylabel('Actual label')  plt.xlabel('Predicted label')  all\_sample\_title = 'Accuracy Score: {0}'.format(dtree.score(X\_test, y\_test)) plt.title(all\_sample\_title, size = 15) | | | | |

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

|  |
| --- |
| plt.savefig("one.png") |
| Output:-  p4(3.png |
| plt.figure(figsize = (20,20))  dec\_tree = plot\_tree(decision\_tree=dtree, feature\_names = df1.columns,  class\_names =["setosa", "vercicolor", "verginica"] , fill ed = True , precision = 4, rounded = True)  plt.savefig("one.png") |
| **Output:-** |

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya



PRACTICAL:6

Aim : To implement K-NN (K-Nearest Neighbor) Classification algorithm on a sample dataset using Python Programming

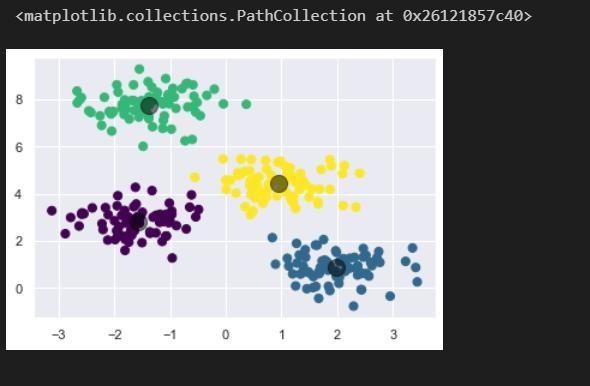
|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier  from sklearn.model\_selection import train\_test\_split from sklearn.datasets import load\_iris  import numpy as np  import matplotlib.pyplot as plt |
| irisData = load\_iris() |
| X = irisData.data y = irisData.target |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |
| neighbors = np.arange(1, 9) train\_accuracy = np.empty(len(neighbors)) test\_accuracy = np.empty(len(neighbors)) |
| for i, k in enumerate(neighbors):  knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train)  train\_accuracy[i] = knn.score(X\_train, y\_train) test\_accuracy[i] = knn.score(X\_test, y\_test)  plt.plot(neighbors, test\_accuracy, label='Testing dataset Accuracy') plt.plot(neighbors, train\_accuracy, label='Training dataset Accuracy') plt.legend()  plt.xlabel('n\_neighbors')  plt.ylabel('Accuracy') plt.show() |
| Output:-  p5.PNG |

PRACTICAL:7

Aim : To implement K-Means Clustering on a sample dataset using Python Programming.

|  |
| --- |
| import seaborn as sns  import numpy as np  import matplotlib.pyplot as plt  %matplotlib inline sns.set() |
| from sklearn.datasets.\_samples\_generator import make\_blobs  X, y\_true = make\_blobs(n\_samples=300, centers=4,cluster\_std=0.60, random\_state=0) plt.scatter(X[:, 0], X[:, 1], s=50, color=["blue"]) |
| 6.1.PNG |
|  |
| from sklearn.cluster import KMeans kmeans = KMeans(n\_clusters=4) kmeans.fit(X)  y\_kmeans = kmeans.predict(X) |
| plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis') centers = kmeans.cluster\_centers\_  plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5) |

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya



from sklearn.metrics import pairwise\_distances\_argmin def find\_clusters(X, n\_clusters, rseed=2):

# 1. Randomly choose clusters

rng = np.random.RandomState(rseed)

i = rng.permutation(X.shape[0])[:n\_clusters] centers = X[i]

while True:

# 2a. Assign labels based on closest center

labels = pairwise\_distances\_argmin(X, centers) # 2b. Find new centers from means of points new\_centers = np.array([X[labels == i].mean(0) for i in range(n\_clusters)])

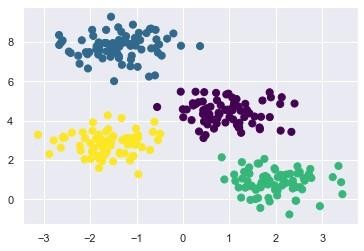
# 2c. Check for convergence

if np.all(centers == new\_centers): break

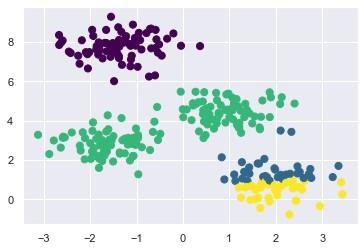
centers = new\_centers return centers, labels

centers, labels = find\_clusters(X, 4) plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya



centers, labels = find\_clusters(X, 4, rseed=0) plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis')

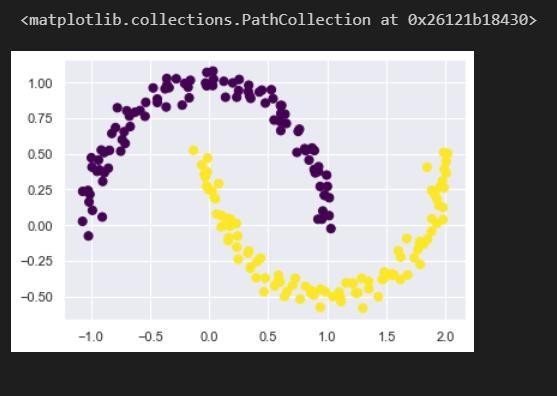


labels = KMeans(6, random\_state=0).fit\_predict(X) plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')

from sklearn.cluster import SpectralClustering model = SpectralClustering(n\_clusters=2,

affinity='nearest\_neighbors',assign\_labels='kmeans')labels = model.fit\_predict(X) plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis')

ENENNNO:O:--119900331133111166000055 SUSUB:B:--PRPR NNAME:AME:--DDhhartiarti SaSarrvvaiyaaiya



# PRACTICAL:8

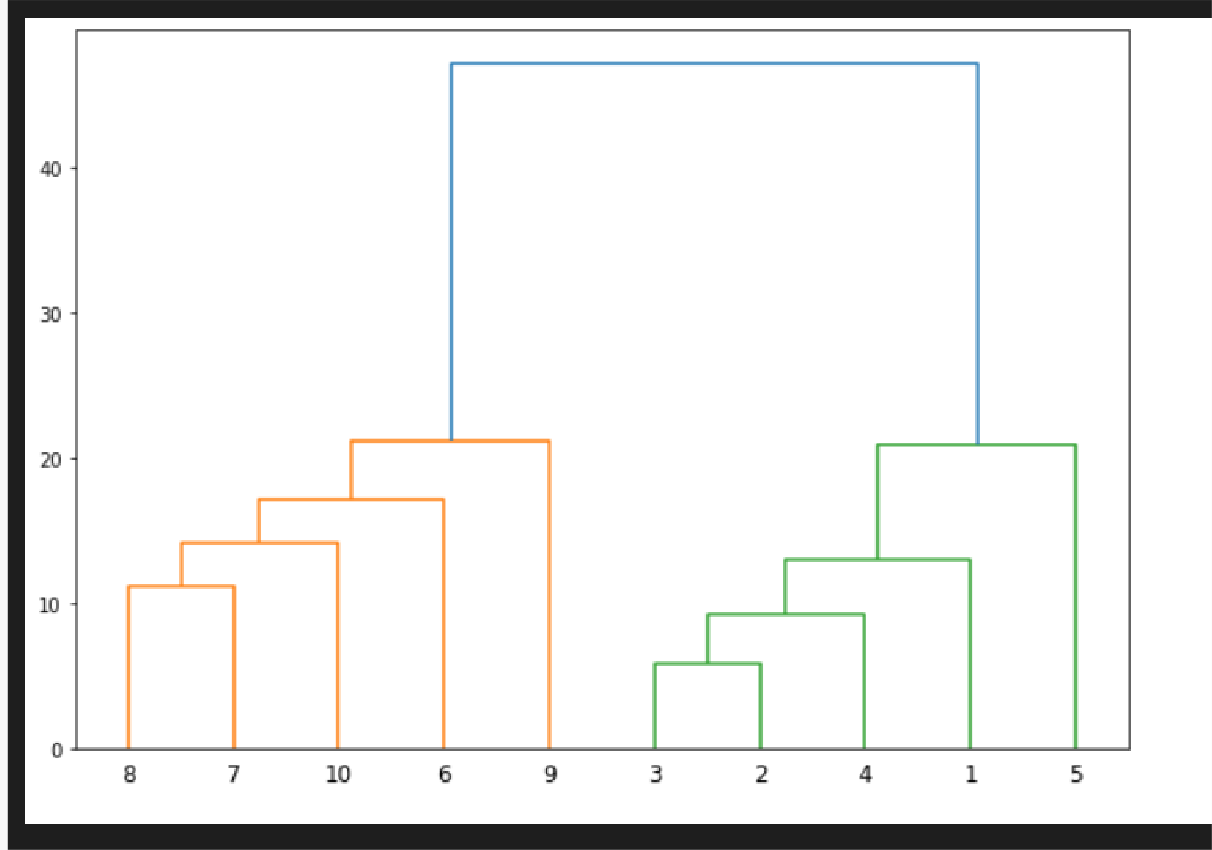
#### Aim : To implement Hierarchical Clustering on a sample dataset using Python Programming.

|  |
| --- |
| import numpy as np  X = np.array([[5,3], [10,15],  [15,12],  [24,10],  [30,30],  [85,70],  [71,80],  [60,78],  [70,55],  [80,91],]) |
| import matplotlib.pyplot as plt labels = range(1, 11) plt.figure(figsize=(10, 7)) plt.subplots\_adjust(bottom=0.1)  plt.scatter(X[:, 0], X[:, 1], label='True Position')  for label, x, y in zip(labels, X[:, 0], X[:, 1]):  plt.annotate( label,  xy=(x, y), xytext=(-3, 3),  textcoords='offset points', ha='right', va='bottom')  plt.show() |
| 7.1.PNG |

from scipy.cluster.hierarchy import dendrogram, linkage from matplotlib import pyplot as plt

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

linked = linkage(X, 'single') labelList = range(1, 11) plt.figure(figsize=(10, 7)) dendrogram(linked, orientation='top', labels=labelList, distance\_sort='descending', show\_leaf\_counts=True) plt.show()



# PRACTICAL:9

#### Aim : To To implement Handwritten Character Recognition using Artificial Neural Networks using Python Programming (using tensorflow/keras).

from google.colab import drive drive.mount('/content/drive')

from future import print\_function import matplotlib.pyplot as plt import matplotlib.pyplot as pyplot import numpy as np

import pandas as pd

from keras import backend as K

from keras.layers import Conv2D, MaxPooling2D from keras.layers import Dense, Dropout, Flatten from keras.models import Sequential

from keras.datasets import mnist import keras

%tensorflow\_version 1.x

batch\_size = 100

num\_classes = 446

epochs = 40

img\_rows, img\_cols = 28, 28 read = pd.read\_csv(

"/content/drive/My Drive/DT/CSV File/train\_28\_split.csv").values arr = np.array

arr = read.reshape(29890, 28, 28, 1) x\_train = arr

readt = pd.read\_csv(

"/content/drive/My Drive/DT/CSV File/test\_28\_split.csv").values arrt = np.array

arrt = readt.reshape(17, 28, 28, 1) x\_test = arrt

y\_train = pd.read\_csv(

"/content/drive/My Drive/DT/CSV File/train\_28\_split\_label.csv").va lues

y\_test = pd.read\_csv(

"/content/drive/My Drive/DT/CSV File/test\_28\_split\_label.csv").val ues

if K.image\_data\_format() == 'channels\_first':

x\_train = x\_train.reshape(x\_train.shape[0], 1, img\_rows, img\_cols) x\_test = x\_test.reshape(x\_test.shape[0], 1, img\_rows, img\_cols) input\_shape = (1, img\_rows, img\_cols)

else:

x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1) x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1) input\_shape = (img\_rows, img\_cols, 1)

x\_train = x\_train.astype('float32') x\_test = x\_test.astype('float32') x\_train /= 255

x\_test /= 255

print('x\_train shape:', x\_train.shape)

print(x\_train.shape[0], 'train samples') print(x\_test.shape[0], 'test samples') y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes) model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape))

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Dropout(0.25)) model.add(Flatten()) model.add(Dense(128, activation='relu')) model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax')) model.compile(loss=keras.losses.categorical\_crossentropy, optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])

model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=epochs,

verbose=1, validation\_data=(x\_test, y\_test))

score = model.evaluate(x\_test, y\_test, verbose=0) print('Test loss:', score[0])

print('Test accuracy:', score[1], ' (i.e) ', score[1]\*100)

#### Output:-

x\_train shape: (29890, 28, 28, 1)

29890 train samples

17 test samples

Train on 29890 samples, validate on 17 samples Epoch 1/40

29890/29890 [==============================] - 3s 96us/step -

loss: 5.6972 - acc: 0.0161 - val\_loss: 5.4265 - val\_acc: 0.0000e+00 Epoch 2/40

29890/29890 [==============================] - 2s 81us/step -

loss: 5.0635 - acc: 0.0711 - val\_loss: 4.4552 - val\_acc: 0.1176 Epoch 3/40

29890/29890 [==============================] - 2s 81us/step -

loss: 4.6288 - acc: 0.1120 - val\_loss: 3.6506 - val\_acc: 0.2941 Epoch 4/40

29890/29890 [==============================] - 2s 79us/step -

loss: 4.3015 - acc: 0.1497 - val\_loss: 3.1845 - val\_acc: 0.3529 Epoch 5/40

29890/29890 [==============================] - 2s 79us/step -

loss: 4.0127 - acc: 0.1802 - val\_loss: 2.7139 - val\_acc: 0.4118 Epoch 6/40

29890/29890 [==============================] - 2s 79us/step -

loss: 3.7666 - acc: 0.2090 - val\_loss: 2.3264 - val\_acc: 0.5294 Epoch 7/40

29890/29890 [==============================] - 2s 79us/step -

loss: 3.5683 - acc: 0.2359 - val\_loss: 2.1884 - val\_acc: 0.3529 Epoch 8/40

29890/29890 [==============================] - 2s 79us/step -

loss: 3.4099 - acc: 0.2535 - val\_loss: 1.7577 - val\_acc: 0.7059 Epoch 9/40

29890/29890 [==============================] - 2s 79us/step -

loss: 3.2674 - acc: 0.2753 - val\_loss: 1.4171 - val\_acc: 0.8235 Epoch 10/40

29890/29890 [==============================] - 2s 79us/step -

loss: 3.1389 - acc: 0.2943 - val\_loss: 1.2124 - val\_acc: 0.7647 Epoch 11/40

29890/29890 [==============================] - 2s 80us/step -

loss: 3.0336 - acc: 0.3073 - val\_loss: 1.1363 - val\_acc: 0.7647

a

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiy

Epoch 12/40

29890/29890 [==============================] - 2s 78us/step -

loss: 2.9201 - acc: 0.3271 - val\_loss: 0.9172 - val\_acc: 0.8235 Epoch 13/40

29890/29890 [==============================] - 2s 77us/step -

loss: 2.8270 - acc: 0.3426 - val\_loss: 0.8426 - val\_acc: 0.8235 Epoch 14/40

29890/29890 [==============================] - 2s 78us/step -

loss: 2.7424 - acc: 0.3580 - val\_loss: 0.8784 - val\_acc: 0.8235 Epoch 15/40

29890/29890 [==============================] - 2s 79us/step -

loss: 2.6378 - acc: 0.3743 - val\_loss: 0.6120 - val\_acc: 0.8235 Epoch 16/40

29890/29890 [==============================] - 2s 78us/step -

loss: 2.5572 - acc: 0.3875 - val\_loss: 0.6141 - val\_acc: 0.8235 Epoch 17/40

29890/29890 [==============================] - 2s 78us/step -

loss: 2.4638 - acc: 0.4020 - val\_loss: 0.4350 - val\_acc: 0.9412 Epoch 18/40

29890/29890 [==============================] - 2s 79us/step -

loss: 2.3913 - acc: 0.4173 - val\_loss: 0.4522 - val\_acc: 0.9412 Epoch 19/40

29890/29890 [==============================] - 2s 78us/step -

loss: 2.3112 - acc: 0.4326 - val\_loss: 0.4028 - val\_acc: 1.0000 Epoch 20/40

29890/29890 [==============================] - 2s 77us/step -

loss: 2.2427 - acc: 0.4453 - val\_loss: 0.3799 - val\_acc: 0.9412 Epoch 21/40

29890/29890 [==============================] - 2s 79us/step -

loss: 2.1764 - acc: 0.4594 - val\_loss: 0.3314 - val\_acc: 1.0000 Epoch 22/40

29890/29890 [==============================] - 2s 79us/step -

loss: 2.1120 - acc: 0.4675 - val\_loss: 0.3676 - val\_acc: 1.0000 Epoch 23/40

29890/29890 [==============================] - 2s 78us/step -

loss: 2.0545 - acc: 0.4795 - val\_loss: 0.2578 - val\_acc: 1.0000 Epoch 24/40

29890/29890 [==============================] - 2s 77us/step -

loss: 1.9934 - acc: 0.4906 - val\_loss: 0.2596 - val\_acc: 1.0000 Epoch 25/40

29890/29890 [==============================] - 2s 79us/step -

loss: 1.9445 - acc: 0.4988 - val\_loss: 0.3541 - val\_acc: 1.0000 Epoch 26/40

29890/29890 [==============================] - 2s 77us/step -

loss: 1.8953 - acc: 0.5147 - val\_loss: 0.3137 - val\_acc: 1.0000 Epoch 27/40

29890/29890 [==============================] - 2s 78us/step -

loss: 1.8497 - acc: 0.5207 - val\_loss: 0.2085 - val\_acc: 1.0000 Epoch 28/40

29890/29890 [==============================] - 2s 77us/step -

loss: 1.8007 - acc: 0.5282 - val\_loss: 0.2381 - val\_acc: 1.0000 Epoch 29/40

29890/29890 [==============================] - 2s 77us/step -

loss: 1.7628 - acc: 0.5359 - val\_loss: 0.2275 - val\_acc: 1.0000 Epoch 30/40

29890/29890 [==============================] - 2s 79us/step -

loss: 1.7202 - acc: 0.5483 - val\_loss: 0.2172 - val\_acc: 0.9412 Epoch 31/40

29890/29890 [==============================] - 2s 78us/step -

loss: 1.6807 - acc: 0.5517 - val\_loss: 0.1507 - val\_acc: 1.0000 Epoch 32/40

29890/29890 [==============================] - 2s 78us/step -

loss: 1.6270 - acc: 0.5684 - val\_loss: 0.1784 - val\_acc: 1.0000 Epoch 33/40

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

29890/29890 [==============================] - 2s 76us/step -

loss: 1.5990 - acc: 0.5727 - val\_loss: 0.2037 - val\_acc: 0.9412 Epoch 34/40

29890/29890 [==============================] - 2s 78us/step -

loss: 1.5810 - acc: 0.5754 - val\_loss: 0.1858 - val\_acc: 1.0000 Epoch 35/40

29890/29890 [==============================] - 2s 78us/step -

loss: 1.5561 - acc: 0.5791 - val\_loss: 0.1615 - val\_acc: 1.0000 Epoch 36/40

29890/29890 [==============================] - 2s 80us/step -

loss: 1.5228 - acc: 0.5904 - val\_loss: 0.1523 - val\_acc: 1.0000 Epoch 37/40

29890/29890 [==============================] - 2s 79us/step -

loss: 1.5039 - acc: 0.5956 - val\_loss: 0.2009 - val\_acc: 0.9412 Epoch 38/40

29890/29890 [==============================] - 2s 79us/step -

loss: 1.4786 - acc: 0.5965 - val\_loss: 0.2195 - val\_acc: 0.9412 Epoch 39/40

29890/29890 [==============================] - 2s 79us/step -

loss: 1.4577 - acc: 0.6041 - val\_loss: 0.1707 - val\_acc: 1.0000 Epoch 40/40

29890/29890 [==============================] - 2s 79us/step -

loss: 1.4364 - acc: 0.6063 - val\_loss: 0.1980 - val\_acc: 0.9412

Test loss: 0.19803081452846527

Test accuracy: 0.9411764740943909 (i.e) 94.11764740943909

dictionary = {0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9', 10

: 'અ', 11: 'અ˙', 12: 'અ:', 13: 'આ', 14: 'ઇ' , 15: 'ઈ', 16: 'ઉ', 17: 'ઊ', 18: 'ઋ', 19: 'એ',

20: 'ઐ', 21: 'ઓ', 2 2: 'ઔ', 23: 'ક', 24: 'ક˙', 25: 'ક:', 26: 'કl', 27: 'કક', 28: 'કL', 29: ' ક¸'

, 30: 'ક'˛, 31: 'ક˛', 32: 'ક`', 33: 'ક`', 34: 'ક ', 35: 'ક”', 36: ' ', 37

: ' ˙', 38: ' :', 39: ' l', 40: ' ', 41: ' L', 42: ' ' ¸, 43: ' ˛', 44: ' `', 45: ' `', 46: ' ', 47: ' ”', 48: 'ખ', 49: 'ખ ˙', 50: 'ખ:', 51: 'ખl

', 52: ' ખ', 53: 'ખL', 54: 'ખ' ¸, 55: 'ખ ˛', 56: 'ખ`', 57: 'ખ`', 58: 'ખ ',

59: 'ખ”', 60: 'ગ', 61: 'ગ ˙', 62: 'ગ:', 63: 'ગl', 64: ' ગ', 65: 'ગL', 66

: 'ગ' ¸, 67: 'ગ ˛', 68: 'ગ`', 69: 'ગ`', 70: 'ગ ', 71: 'ગ”', 72: 'ઘ', 73: '

ઘ ˙', 74: 'ઘ:', 75: 'ઘl', 76: 'ઘઘ', 77: 'ઘL', 78: 'ઘ' ¸, 79: 'ઘ ˛', 80: 'ઘ`',

81: 'ઘ`', 82: 'ઘ ', 83: 'ઘ”', 84: 'ચ', 85: 'ચ ˙', 86: 'ચ:', 87: 'ચl', 88

: ' ચ', 89: 'ચL', 90: 'ચ' ¸, 91: 'ચ ˛', 92: 'ચ`', 93: 'ચ`', 94: 'ચ ', 95: '

ચ”', 96: 'છ', 97: 'છ˙', 98: 'છ:', 99: 'છl', 100: 'ઘછ', 101: 'છL', 102: 'છ¸', 103: 'છ˛', 104: 'છ`'

, 105: 'છ`', 106: 'છ ', 107: 'છ”', 108: 'જ', 10

9: 'જ:', 110: 'å', 111: 'જજ', 112: '$', 113: 'જ¸', 114: 'જ˛', 115: 'જ'` , 116: 'જ'` , 117: 'જ ',

118: 'જ”', 119: ' ', 120: ' ˙', 121: ' :', 122: ' l', 123: ' ', 124: ' L', 125: ' ' ¸, 126: '

˛', 127: ' `', 128: ' `', 1

29: ' ', 130: ' ”', 131: 'ઝ', 132: 'ઝ ˙', 133: 'ઝ:', 134: 'ઝl', 135: '

ઝ', 136: 'ઝL', 137: 'ઝ¸', 138: 'ઝ˛', 139: 'ઝ`', 140: 'ઝ`', 141: 'ઝ ', 14

2: 'ઝ”', 143: 'ઞ', 144: 'ટ', 145: 'ટ˙', 146: 'ટ:', 147: 'ટl', 148: 'કટ',

149: 'ટL', 150: 'ટ¸', 151: 'ટ'˛, 152: 'ટ`', 153: 'ટ`', 154: 'ટ ', 155: 'ટ”' , 156: 'ઠ', 157: 'ઠ˙', 1

58: 'ઠ:', 159: 'ઠl', 160: 'કઠ', 161: 'ઠL', 162: ' ઠ¸', 163: 'ઠ˛', 164: 'ઠ`', 165: 'ઠ`', 166: 'ઠ ', 16

7: 'ઠ”', 168: 'ડ', 169: 'ડ˙', 170: 'ડ:', 171: 'ડl', 172: 'કડ', 173: 'ડL', 174: 'ડ¸', 175: 'ડ'˛, 176

: 'ડ' ` , 177: 'ડ`', 178: 'ડ ', 179: 'ડ”', 180: 'ઢ', 181: 'ઢ˙', 182: 'ઢ:', 1

83: 'ઢl', 184: 'કઢ', 185: 'ઢL', 186: 'ઢ¸', 187: 'ઢ'˛, 188: 'ઢ`', 189: 'ઢ`',

190: 'ઢ ', 191: 'ઢ”', 192: 'ણ', 193: 'ણ ˙', 194: 'ણ:', 195: 'ણl', 196: ' ણ', 197: 'ણL', 198: '

ણ' ¸, 199: 'ણ ˛', 200: 'ણ`', 201: 'ણ`', 202: 'ણ ', 2

03: 'ણ”', 204: 'ત', 205: 'ત ˙', 206: 'ત:', 207: 'તl', 208: 'ઘત', 209: '

તL', 210: 'ત' ¸, 211: 'ત ˛', 212: 'ત`', 213: 'ત`', 214: 'ત ', 215: 'ત”', 216

: ' ', 217: ' ˙', 218: ' l', 219: 'ઘ ', 220: ' L', 221: ' ' ¸, 222: ' ˛',

223: ' `', 224: ' `', 225: ' ', 226: ' ”', 227: 'થ', 228: 'થ ˙', 229: '

થ:', 230: 'થl', 231: 'ઘથ', 232: 'થL', 233: 'થ' ¸, 234: 'થ ˛', 235: 'થ`', 23

6: 'થ`', 237: 'થ ', 238: 'થ”', 239: 'દ', 240: 'દ˙', 241: 'દ:', 242: 'દl',

243: 'કદ', 244: 'દL', 245: 'દ¸', 246: 'દ˛', 247: 'દ`', 248: 'દ`', 249: 'દ

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

', 250: 'દ”', 251: 'Ç', 252: 'ધ', 253: 'ધ ˙', 254: 'ધ:', 255: 'ધl', 256: 'ઘધ', 257: 'ધL', 258: '

ધ' ¸, 259: 'ધ ˛', 260: 'ધ`', 261: 'ધ`', 262: 'ધ ', 2

63: 'ધ”', 264: 'ન', 265: 'ન ˙', 266: 'ન:', 267: 'નl', 268: 'ઘન', 269: 'નL

', 270: 'ન' ¸, 271: 'ન ˛', 272: 'ન`', 273: 'ન`', 274: 'ન ', 275: 'ન”', 276: 'પ', 277: 'પ ˙', 278:

'પ:', 279: 'પl', 280: 'ઘપ', 281: 'પL', 282: 'પ' ¸,

283: 'પ ˛', 284: 'પ`', 285: 'પ`', 286: 'પ ', 287: 'પ”', 288: 'ફ', 289: 'ફ'˙ , 290: 'ફ:', 291: 'ફl'

, 292: 'કફ', 293: 'ફL', 294: 'ફ¸', 295: 'ફ˛', 296: ' ફ`', 297: 'ફ`', 298: 'ફ ', 299: 'ફ”', 300: 'બ',

301: 'બ ˙', 302: 'બ:', 303: 'બl', 304: ' બ', 305: 'બL', 306: 'બ' ¸, 307: 'બ ˛', 308: 'બ`', 30

9: 'બ`',

310: 'બ ', 311: 'બ”', 312: 'ભ', 313: 'ભ ˙', 314: 'ભ:', 315: 'ભl', 316: '

ભ', 317: 'ભL', 318: 'ભ' ¸, 319: 'ભ ˛', 320: 'ભ`', 321: 'ભ`', 322: 'ભ ', 32

3: 'ભ”', 324: 'મ', 325: 'મ ˙', 326: 'મ:', 327: 'મl', 328: 'ઘમ', 329: 'મL

', 330: 'મ' ¸, 331: 'મ ˛', 332: 'મ`', 333: 'મ`', 334: 'મ ', 335: 'મ”', 336: 'ય', 337: 'ય ˙', 338:

'ય:', 339: 'યl', 340: 'ઘય', 341: 'યL', 342: 'ય' ¸,

343: 'ય ˛', 344: 'ય`', 345: 'ય`', 346: 'ય ', 347: 'ય”', 348: 'ર', 349: 'ર˙' , 350: 'ર:', 351: 'રl',

352: 'કર', 353: 'રL', 354: ' ', 355: '5', 356: ' ર'` , 357: 'ર'` , 358: 'ર ', 359: 'ર”', 360: 'લ', 36

1: 'લ ˙', 362: 'લ:', 363: 'લl', 364: ' લ', 365: 'લL', 366: 'લ' ¸, 367: 'લ ˛', 368: 'લ`', 369: 'લ`'

,

370: 'લ ', 371: 'લ”', 372: 'ળ', 373: 'ળ˙', 374: 'ળ:', 375: 'ળl', 376: '

ળ', 377: 'ળL', 378: 'ળ¸', 379: 'ળ˛', 380: 'ળ`', 381: 'ળ`', 382: 'ળ ', 38

3: 'ળ”', 384: 'વ', 385: 'વ ˙', 386: 'વ:', 387: 'વl', 388: 'ઘવ', 389: 'વL

', 390: 'વ' ¸, 391: 'વ ˛', 392: 'વ`', 393: 'વ`', 394: 'વ ', 395: 'વ”', 396:

'શ', 397: 'શ ˙', 398: 'શ:', 399: 'શl', 400: 'ઘશ', 401: 'શL', 402: 'શ' ¸,

403: 'શ ˛', 404: 'શ`', 405: 'શ`', 406: 'શ ', 407: 'શ”', 408: 'R', 409: '

Rl', 410: 'ષ', 411: 'ષ ˙', 412: 'ષ:', 413: 'ષl', 414: 'ઘષ', 415: 'ષL', 41

6: 'ષ' ¸, 417: 'ષ ˛', 418: 'ષ`', 419: 'ષ`', 420: 'ષ ', 421: 'ષ”', 422: 'સ',

423: 'સ ˙', 424: 'સ:', 425: 'સl', 426: 'ઘસ', 427: 'સL', 428: 'સ' ¸, 429: 'સ ˛', 430: 'સ`', 431:

'સ`', 432: 'સ ', 433: 'સ”', 434: 'હ', 435: 'હ˙', 436

: 'હ:', 437: 'હl', 438: 'કહ', 439: 'હL', 440: 'હ¸', 441: 'હ'˛, 442: 'હ' ` , 4

43: 'હ`', 444: 'હ ', 445: 'હ”'}

example = x\_test[2]

prediction = model.predict(example.reshape(1, 28, 28, 1)) hard\_maxed\_prediction = np.zeros(prediction.shape) hard\_maxed\_prediction[0][np.argmax(prediction)] = 1

plt.imshow(example.reshape(28, 28), cmap="gray") plt.show()

print('Predicted character is:', dictionary[np.argmax(prediction)])

#### 8.1.pngOutput:-

PRACTICAL:10

Aim : To implement ANN using Python Programming without using any library.

import numpy as np class NeuralNetwork():

def init (self): np.random.seed(1)

self.synaptic\_weights = 2 \* np.random.random((3, 1)) - 1 def sigmoid(self, x):

return 1 / (1 + np.exp(-x)) def sigmoid\_derivative(self, x):

return x \* (1 - x)

def train(self, training\_inputs, training\_outputs, training\_iterations): for iteration in range(training\_iterations):

output = self.think(training\_inputs) error = training\_outputs - output

adjustments = np.dot(training\_inputs.T, error \* self.sigmoid\_derivative(output)) self.synaptic\_weights += adjustments

def think(self, inputs):

inputs = inputs.astype(float)

output = self.sigmoid(np.dot(inputs, self.synaptic\_weights)) return output

if name == " main ":

neural\_network = NeuralNetwork() print("Beginning Randomly Generated Weights: ") print(neural\_network.synaptic\_weights) training\_inputs = np.array([[0, 0, 1],

[1, 1, 1],

[1, 0, 1],

[0, 1, 1]])

training\_outputs = np.array([[0, 1, 1, 0]]).T

neural\_network.train(training\_inputs, training\_outputs, 15000) print("Ending Weights After Training: ") print(neural\_network.synaptic\_weights)

user\_input\_one = str(input("User Input One: ")) print("user\_input\_one :- ", user\_input\_one) user\_input\_two = str(input("User Input Two: ")) print("user\_input\_two :- ",user\_input\_two) user\_input\_three = str(input("User Input Three: ")) print("user\_input\_three :- ", user\_input\_three) print("Considering New Situation: ", user\_input\_one, user\_input\_two, user\_input\_three)

print("New Output data: ") print(neural\_network.think(

np.array([user\_input\_one, user\_input\_two, user\_input\_three])))

#### Output:-

Beginning Randomly Generated Weights: [[-0.16595599]

[ 0.44064899]

[-0.99977125]]

Ending Weights After Training: [[10.08740896]

[-0.20695366]

[-4.83757835]]

user\_input\_one :- 1

user\_input\_two :- 0

user\_input\_three :- 0

Considering New Situation: 1 0 0 New Output data: [0.9999584]

# PRACTICAL:11

#### Aim : To implement Customer Churn Prediction using ANN.

import numpy as np import tensorflow as tf import pandas as pd

dataset = pd.read\_csv(r'C:\Users\admin\OneDrive\Desktop\CLG Work\SEM 7 \PR\Practical\Practical 1 0/dataset.csv')

dataset.shape

Output :- (10000, 14)

x = dataset.iloc[:, 3:-1].values y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

x[:, 2] = le.fit\_transform(x[:, 2]) print(x)

#### output:-

[[619 'France' 0 ... 1 1 101348.88] [608 'Spain' 0 ... 0 1

112542.58] [502 'France' 0 ... 1 0 113931.57] ... [709 'France' 0 ...

0 1 42085.58] [772 'Germany' 1 ... 1 0 92888.52] [792 'France' 0 ... 1

0 38190.78]]

from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])

], remainder='passthrough') x = np.array(ct.fit\_transform(x))

from sklearn.model\_selection import train\_test\_split X\_train, X\_test, y\_train, y\_test = train\_test\_split( x, y, test\_size=0.2, random\_state=1)

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

ann = tf.keras.models.Sequential() ann.add(tf.keras.layers.Dense(units=6, activation='relu')) ann.add(tf.keras.layers.Dense(units=6, activation='relu')) ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

ann.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

ann.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['ac curacy'])

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

ann.fit(X\_train, y\_train, batch\_size=32, epochs=100)

#### output:-

Epoch 1/100

250/250 [==============================] - 2s 931us/step - loss: 0.6624 - accuracy:

0.6181

Epoch 2/100

250/250 [==============================] - 0s 932us/step - loss: 0.4831 - accuracy:

0.8014

Epoch 3/100

250/250 [==============================] - 0s 803us/step - loss: 0.4499 - accuracy:

0.8081

Epoch 4/100

250/250 [==============================] - 0s 932us/step - loss: 0.4374 - accuracy:

0.8105

Epoch 5/100

250/250 [==============================] - 0s 893us/step - loss: 0.4305 - accuracy:

0.8111

Epoch 6/100

250/250 [==============================] - 0s 1ms/step - loss: 0.4251 - accuracy:

0.8145

Epoch 7/100

250/250 [==============================] - 0s 676us/step - loss: 0.4210 - accuracy:

0.8179

Epoch 8/100

250/250 [==============================] - 0s 668us/step - loss: 0.4166 - accuracy:

0.8207

Epoch 9/100

250/250 [==============================] - 0s 709us/step - loss: 0.4122 - accuracy:

0.8246

Epoch 10/100

250/250 [==============================] - 0s 700us/step - loss: 0.4084 - accuracy:

0.8256

Epoch 11/100

250/250 [==============================] - 0s 565us/step - loss: 0.4047 - accuracy:

0.8291

Epoch 12/100

250/250 [==============================] - 0s 634us/step - loss: 0.4013 - accuracy:

0.8267

Epoch 13/100

show more (open the raw output data in a text editor) ... 250/250 [==============================] - 0s 583us/step - loss:

0.3340 - accuracy: 0.8639 Epoch 99/100 250/250 [==============================] - 0s 597us/step - loss: 0.3342 -

accuracy: 0.8614 Epoch 100/100 250/250 [==============================] - 0s 605us/step - loss: 0.3340 -

accuracy: 0.8627

**Predicting the result of a single observation**

ENNO:-190313116005 SUB:-PR NAME:-Dharti Sarvaiya

## Using this ANN model to predict if the customer with the following informations will leave the bank:

**Geography: France Credit Score: 600 Gender: Male**

**Age: 40 years old Tenure: 3 years**

**Balance: \$ 60000 Number of Products: 2**

**Does this customer have a credit card? Yes Is this customer an Active Member: Yes Estimated Salary: \$ 50000**

**So, should we say goodbye to that customer**?

print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])))

**output:-**

[[0.]]

y\_predicted = ann.predict(X\_test) y\_predicted = (y\_predicted > 0.5) print(np.concatenate((y\_predicted.reshape(

len(y\_predicted), 1), y\_test.reshape(len(y\_test), 1)), 1))

#### Output:-

[[0 0] [0 0] [0 0] ... [0 0] [0 0] [1 0]]

from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_predicted)

print(cm)

accuracy\_score(y\_test, y\_predicted)

#### output:-

[[1541 44] [ 234 181]]

## Therefore, Accuracy = 86%

**Total correct predictions that customers stay in bank = 1541 Total correct predictions that customers leave the bank = 185 Total incorrect predictions that customers stay in the bank = 230 Total incorrect predictions that customers leave in the bank = 44**