**12.ANN**

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder

from sklearn.compose import ColumnTransformer

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

from sklearn.metrics import confusion\_matrix, accuracy\_score

dataset = pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\12Churn\_Modelling12.csv")

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

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

X[:, 2] = LabelEncoder().fit\_transform(X[:, 2])

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

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

sc = StandardScaler()

X = sc.fit\_transform(X)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y)

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(loss='binary\_crossentropy', metrics=['accuracy'])

ann.fit(x\_train, y\_train, batch\_size=15, epochs=5)

y\_pred = ann.predict(x\_test)

y\_pred = (y\_pred > 0.5)

con = np.concatenate((y\_pred.reshape(len(y\_pred), 1), y\_test.reshape(len(y\_test), 1)), axis=1)

print("Predicted Output:\n", y\_pred.flatten())

print("Actual Output:\n", y\_test)

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

asr = accuracy\_score(y\_test, y\_pred)

print("Accuracy score of the model:", asr)

**11.NAVIE BAYESIAN CLASIFIER**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB

from sklearn.preprocessing import StandardScaler

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

from sklearn.metrics import confusion\_matrix, accuracy\_score

data = pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\11Social\_Network\_Ads11.csv")

x = data.iloc[:, [0, 1]].values

y = data.iloc[:, 2].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25, random\_state=0)

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.transform(x\_test)

classifier = GaussianNB()

classifier.fit(x\_train, y\_train)

y\_predict = classifier.predict(x\_test)

cm = confusion\_matrix(y\_test, y\_predict)

asr = accuracy\_score(y\_test, y\_predict)

print("Confusion Matrix:",cm)

print("Accuracy Score:", asr)

x1\_min, x1\_max = x\_train[:, 0].min() - 1, x\_train[:, 0].max() + 1

x2\_min, x2\_max = x\_train[:, 1].min() - 1, x\_train[:, 1].max() + 1

x1, x2 = np.meshgrid(np.arange(x1\_min, x1\_max, 0.01), np.arange(x2\_min, x2\_max, 0.01))

plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape))

plt.scatter(x\_train[:, 0], x\_train[:, 1], c=y\_train)

plt.title('Naive Bayes (Training Set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.show()

x1\_min, x1\_max = x\_test[:, 0].min() - 1, x\_test[:, 0].max() + 1

x2\_min, x2\_max = x\_test[:, 1].min() - 1, x\_test[:, 1].max() + 1

x1, x2 = np.meshgrid(np.arange(x1\_min, x1\_max, 0.01), np.arange(x2\_min, x2\_max, 0.01))

plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape))

plt.scatter(x\_test[:, 0], x\_test[:, 1], c=y\_test)

plt.title('Naive Bayes (Testing Set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.show()

**10: RANDOM FOREST**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import confusion\_matrix,accuracy\_score ,classification\_report

data=pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\10diabetes10.csv")

print("Dataset : ",data)

x=data.drop('Outcome',axis=1)

y=data['Outcome']

print("x : ",x)

print("y : ",y)

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=42)

model=RandomForestClassifier()

model.fit(x\_train,y\_train)

y\_predict=model.predict(x\_test)

print("Predicted value : ", y\_predict)

cm=confusion\_matrix(y\_test,y\_predict)

print("confusion matrix",cm)

cr=classification\_report(y\_test,y\_predict)

print("---classification report---", cr)

asc=accuracy\_score(y\_test,y\_predict)

print("Accuracy Score of the model : ",asc)

index=np.arange(0,len(y\_test))

fig,ax=plt.subplots(1,1)

plt.scatter(index,y\_test,c='red',label='Actual Value')

plt.scatter(index,y\_predict,c='blue',label='Predicted Value')

plt.legend()

plt.show()

**9: HIERARCHICAL CLUSTERING**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

import scipy.cluster.hierarchy as sch

data=pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\9Mall\_Customers9.csv")

x=data.iloc[:,[3,4]].values

dendrogram=sch.dendrogram(sch.linkage(x,method='ward'))

plt.title('Dendrogram')

plt.xlabel('Customers')

plt.ylabel('Euclidean Distances')

plt.show()

hc=AgglomerativeClustering(n\_clusters=7)

y\_hc=hc.fit\_predict(x)

print("Predicted value : ",y\_hc)

plt.scatter(x[y\_hc == 0,0],x[y\_hc == 0,1],s=100,c='red',label='Cluster1')

plt.scatter(x[y\_hc == 1,0],x[y\_hc ==1,1],s=100,c='blue',label='Cluster 2')

plt.scatter(x[y\_hc == 2,0],x[y\_hc ==2,1],s=100,c='green',label='Cluster 3')

plt.scatter(x[y\_hc == 3,0],x[y\_hc ==3,1],s=100,c='cyan',label='Cluster 4')

plt.scatter(x[y\_hc == 4,0],x[y\_hc ==4,1],s=100,c='magenta',label='Cluster 5')

plt.title('Clusters of Customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

**8: KMEANS**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data=pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\8Mall\_Customers8.csv")

print("Dataset : ",data)

x=data.iloc[:,[3,4]].values

wcss=[]

for i in range(1,11):

    kmeans=KMeans(n\_clusters=i)

    kmeans.fit(x)

    wcss.append(kmeans.inertia\_)

plt.plot(range(1,11),wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

kmeans=KMeans(n\_clusters=5)

y\_kmeans=kmeans.fit\_predict(x)

print("Predicted values : ",y\_kmeans)

plt.scatter(x[y\_kmeans == 0,0],x[y\_kmeans ==0,1],s=100,c='red',label='Cluster 1')

plt.scatter(x[y\_kmeans == 1,0],x[y\_kmeans ==1,1],s=100,c='blue',label='Cluster 2')

plt.scatter(x[y\_kmeans == 2,0],x[y\_kmeans ==2,1],s=100,c='green',label='Cluster 3')

plt.scatter(x[y\_kmeans == 3,0],x[y\_kmeans ==3,1],s=100,c='cyan',label='Cluster 4')

plt.scatter(x[y\_kmeans == 4,0],x[y\_kmeans ==4,1],s=100,c='magenta',label='Cluster 5')

plt.title('Clusters of Customers')

plt.xlabel('Anuual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

**7: KNN**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

report = classification\_report(y\_test, y\_pred)

print("Classification Report:",report)

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:",cm)

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

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')

plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_pred, cmap='viridis', marker='x', s=100)

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.title('k-NN Classification Results')

plt.show()

**6: LINEAR REGRESSION**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

data=pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\5Lab5\_Google\_Stock\_Price\_Train5.csv")

print("Dataset : ",data.head())

x=data.iloc[:,0].str.replace('/','').str.replace('-','').astype('int').values

y=data.iloc[:,-1].str.replace(',','').astype('int').values

x=x.reshape(len(x),1)

y=y.reshape(len(y),1)

print("x : ",x)

print("y : ",y)

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

model=LinearRegression()

model.fit(x\_train,y\_train)

y\_predict=model.predict(x\_test)

print("Predicted value : ",y\_predict)

plt.scatter(x\_test,y\_test,color='red')

plt.plot(x\_test,y\_predict,color='blue')

plt.title('Date Vs Volume')

plt.xlabel('Date')

plt.ylabel('Volume (K)')

plt.show()

**4: DECISION TREE**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

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

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.metrics import confusion\_matrix, accuracy\_score

data = pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\4Social\_Network\_Ads4.csv")

print("Dataset:",data)

all\_cols = data.columns

features = all\_cols[1:5]

for col in data.columns:

    if data[col].dtype == 'object':

        data[col] = LabelEncoder().fit\_transform(data[col])

inputs = data.iloc[:, :-1].values

target = data.iloc[:, -1].values

print("Inputs:",inputs)

print("Target:",target)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(inputs, target, test\_size=0.2, random\_state=0)

id3 = DecisionTreeClassifier()

id3.fit(x\_train, y\_train)

y\_predict = id3.predict(x\_test)

print("Predicted value:",y\_predict)

cm = confusion\_matrix(y\_test, y\_predict)

asr= accuracy\_score(y\_test, y\_predict)

print("Confusion Matrix:",cm)

print("Accuracy score of the model:",asr)

tree.plot\_tree(id3, feature\_names=features)

**3: CANDIDATE ELIMINATION**

import numpy as np

import pandas as pd

data = pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\3enjoysport3.csv")

concepts = np.array(data.iloc[:, 0:-1])

target = np.array(data.iloc[:, -1])

print("\nInstances are:\n", concepts)

print("\nTarget Values are: ", target)

def learn(concepts, target):

    specific\_h = concepts[0].copy()

    print("\nInitialization of specific\_h and general\_h")

    print("\nSpecific Boundary: ", specific\_h)

    general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

    print("\nGeneric Boundary: ", general\_h)

    for i, h in enumerate(concepts):

        print("\nInstance", i + 1, "is ", h)

        if target[i] == "yes":

            print("Instance is Positive ")

            for x in range(len(specific\_h)):

                if h[x] != specific\_h[x]:

                    specific\_h[x] = '?'

                    general\_h[x][x] = '?'

        if target[i] == "no":

            print("Instance is Negative ")

            for x in range(len(specific\_h)):

                if h[x] != specific\_h[x]:

                    general\_h[x][x] = specific\_h[x]

                else:

                    general\_h[x][x] = '?'

        print("Specific Boundary after ", i + 1, "Instance is ", specific\_h)

        print("Generic Boundary after ", i + 1, "Instance is ", general\_h)

        print("\n")

    indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

    for i in indices:

        general\_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h: ", s\_final, sep="\n")

print("Final General\_h: ", g\_final, sep="\n")

**2: FIND S**

import numpy as np

import pandas as pd

data = pd.read\_csv("C:\\Users\\udupa\\Downloads\\ML Lab Datasets\\2lab2.csv")

print("Dataset: ", data)

concepts = data.iloc[:, :-1].values

target = data.iloc[:, -1].values

print("Concepts: ", concepts)

print("Target: ", target)

def train(conc, tar):

    specific\_h = None

    for i, val in enumerate(tar):

        if val == "yes":

            specific\_h = conc[i].copy()

            break

    if specific\_h is not None:

        for i, val in enumerate(conc):

            if tar[i] == "yes":

                for x in range(len(specific\_h)):

                    if val[x] != specific\_h[x]:

                        specific\_h[x] = "?"

    return specific\_h

print("Final specific\_h:")

print(train(concepts, target))