import numpy as np

import pandas as pd

import seaborn as sns

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

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler

from sklearn.svm import SVC

""" 3. Load the dataset as dataframe using pandas """

df = pd.read\_csv('/content/glass source classification dataset.csv')

df

""" 4. Handle missing values if needed """

df.isnull().sum()

impute = SimpleImputer(missing\_values=np.nan, strategy='mean')

impute.fit(df[['Ca']])

df['Ca'] = impute.transform(df[['Ca']])

df.isnull().sum()

""" 5.Encode Categorical Features """

enc = LabelEncoder()

df['Ba'] = enc.fit\_transform(df['Ba'])

df['Fe'] = enc.fit\_transform(df['Fe'])

df['Type'] = enc.fit\_transform(df['Type'])

""" 6. Scale all the values between 0-1 with proper scaling technique """

scaler= MinMaxScaler()

scaler.fit(df)

MinMaxScaler()

df\_scaled = scaler.transform(df)

print("per-feature minimum after scaling:\n {}".format(df\_scaled.min(axis=0)))

print("per-feature maximum after scaling:\n {}".format(df\_scaled.max(axis=0)))

x = df.iloc[:,1:12]

y = df.iloc[:,-1]

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

svc = SVC(kernel="linear")

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

predictions = svc.predict(x\_test)

pre\_svc = accuracy\_score(y\_test, predictions)

nnc=MLPClassifier(hidden\_layer\_sizes=(7), activation="relu", max\_iter=10000)

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

predictions = svc.predict(x\_test)

pre\_mlp = accuracy\_score(y\_test, predictions)

rfc = RandomForestClassifier(n\_estimators=50)

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

predictions = svc.predict(x\_test)

pre\_rfc = accuracy\_score(y\_test, predictions)

pca = PCA(n\_components=5)

principal\_components= pca.fit\_transform(df)

principal\_df = pd.DataFrame(data=principal\_components, columns=["principle component 1", "principle component 2","principle component 3","principle component 4","principle component 5"])

main\_df=pd.concat([principal\_df, df[['Type']]], axis=1)

X= main\_df.drop("Type" , axis=1)

y= main\_df["Type"]

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

svc = SVC(kernel="linear")

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

predictions = svc.predict(x\_test)

post\_svc = accuracy\_score(y\_test, predictions)

nnc=MLPClassifier(hidden\_layer\_sizes=(7), activation="relu", max\_iter=10000)

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

predictions = svc.predict(x\_test)

post\_mlp = accuracy\_score(y\_test, predictions)

rfc = RandomForestClassifier(n\_estimators=50)

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

predictions = svc.predict(x\_test)

post\_rfc = accuracy\_score(y\_test, predictions)

barWidth = .5

plt.subplots(figsize =(15, 8))

pre\_pca = [pre\_svc, pre\_mlp, pre\_rfc]

post\_pca = [post\_svc, post\_mlp, post\_rfc]

algo = ["svc","random-forest", "mlp"]

width = 0.2

plt.bar(np.arange(len(algo)),pre\_pca, width=width, label ='pre-pca' , color='r' )

plt.bar(np.arange(len(algo)) + width, post\_pca, width=width ,label ='post-pca', color='g')

plt.xticks(ticks=indices, labels=algo)

plt.xlabel("algo")

plt.ylabel("accuracy")

plt.title("Pre-PCA vs Post-PCA")

plt.legend()

plt.show()