LAB-7

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from xgboost import XGBClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

# Load and preprocess

df = pd.read\_csv("lab3(dataset).csv")

df['price\_indicator'] = df['ram'] + df['battery\_power']

df['price\_range'] = pd.qcut(df['price\_indicator'], q=4, labels=[0, 1, 2, 3]).astype(int)

df = df.drop(columns=["id", "price\_indicator"])

X = df.drop(columns=["price\_range"])

y = df["price\_range"]

scaler = StandardScaler()

X = scaler.fit\_transform(X)

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

# Classifiers

def get\_classifiers():

    return {

        "SVM": SVC(probability=True),

        "Decision Tree": DecisionTreeClassifier(),

        "Random Forest": RandomForestClassifier(),

        "XGBoost": XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss'),

        "AdaBoost": AdaBoostClassifier(),

        "Naive Bayes": GaussianNB(),

        "MLP": MLPClassifier(max\_iter=500)

    }

results = []

for name, model in get\_classifiers().items():

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    results.append([name,

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

                    precision\_score(y\_test, y\_pred, average='weighted', zero\_division=0),

                    recall\_score(y\_test, y\_pred, average='weighted'),

                    f1\_score(y\_test, y\_pred, average='weighted')])

    print(f"\n{name}:\n{classification\_report(y\_test, y\_pred)}\n")

results\_df = pd.DataFrame(results, columns=["Classifier", "Accuracy", "Precision", "Recall", "F1 Score"])

print(results\_df)

from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor

from xgboost import XGBRegressor

from sklearn.neural\_network import MLPRegressor

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

# Same X, y as above

y\_reg = df["price\_range"]  # Regression target

def get\_regressors():

    return {

        "SVR": SVR(),

        "Decision Tree": DecisionTreeRegressor(),

        "Random Forest": RandomForestRegressor(),

        "XGBoost": XGBRegressor(),

        "AdaBoost": AdaBoostRegressor(),

        "Gradient Boosting": GradientBoostingRegressor(),

        "MLP": MLPRegressor(max\_iter=500)

    }

results = []

for name, model in get\_regressors().items():

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    results.append([name,

                    mean\_squared\_error(y\_test, y\_pred),

                    r2\_score(y\_test, y\_pred)])

    print(f"\n{name}:\nMSE: {mean\_squared\_error(y\_test, y\_pred)} | R2: {r2\_score(y\_test, y\_pred)}")

results\_df = pd.DataFrame(results, columns=["Regressor", "Test MSE", "Test R2"])

print(results\_df)

from sklearn.cluster import AgglomerativeClustering, DBSCAN

from scipy.cluster.hierarchy import dendrogram, linkage

import matplotlib.pyplot as plt

# Use only features for clustering

X\_clust = df.drop(columns=["price\_range"])

X\_clust\_scaled = scaler.fit\_transform(X\_clust)

# Hierarchical

hierarchical = AgglomerativeClustering(n\_clusters=4)

hierarchical\_labels = hierarchical.fit\_predict(X\_clust\_scaled)

print("Hierarchical Clustering Labels:", np.unique(hierarchical\_labels))

# DBSCAN

dbscan = DBSCAN(eps=1.5, min\_samples=5)

dbscan\_labels = dbscan.fit\_predict(X\_clust\_scaled)

print("DBSCAN Clustering Labels:", np.unique(dbscan\_labels))

# Dendrogram

plt.figure(figsize=(10, 5))

linkage\_matrix = linkage(X\_clust\_scaled, method='ward')

dendrogram(linkage\_matrix)

plt.title("Hierarchical Clustering Dendrogram")

plt.xlabel("Samples")

plt.ylabel("Distance")

plt.show()

from scipy.stats import randint

from sklearn.model\_selection import RandomizedSearchCV # Import RandomizedSearchCV

# Example for Random Forest

param\_dist = {

    "n\_estimators": randint(50, 200),

    "max\_depth": randint(3, 15),

    "min\_samples\_split": randint(2, 10)

}

rf = RandomForestClassifier()

search = RandomizedSearchCV(rf, param\_distributions=param\_dist, n\_iter=10, cv=5, scoring='accuracy', random\_state=42)

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

print("Best Params:", search.best\_params\_)

print("Best Score:", search.best\_score\_)

Results: