import pandas as pd

import numpy as np

from sklearn.model\_selection import StratifiedKFold, cross\_val\_predict

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, roc\_curve, auc

import matplotlib.pyplot as plt

from tabulate import tabulate

# ------------------------------

# Load dataset

# ------------------------------

df = pd.read\_csv("heart\_failure\_clinical\_records\_dataset.csv")

# Balance the dataset (undersampling)

dead = df[df["DEATH\_EVENT"] == 1]

alive = df[df["DEATH\_EVENT"] == 0].sample(n=len(dead), random\_state=42)

df\_balanced = pd.concat([dead, alive]).sample(frac=1, random\_state=42)

X = df\_balanced.drop(columns=["DEATH\_EVENT"])

y = df\_balanced["DEATH\_EVENT"]

# Standardize features

scaler = StandardScaler()

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

# Stratified K-Fold

cv = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=42)

# ------------------------------

# Classifiers with regularization to reduce overfitting

# ------------------------------

models = {

"LogisticRegression": LogisticRegression(max\_iter=1000, C=1.0, penalty='l2', solver='liblinear'),

"SVM": SVC(probability=True, C=1.0, kernel='rbf'),

"RandomForest": RandomForestClassifier(n\_estimators=100, max\_depth=5, random\_state=42),

"NaiveBayes": GaussianNB(),

"XGBoost": XGBClassifier(eval\_metric='logloss', use\_label\_encoder=False, n\_estimators=100, max\_depth=3, learning\_rate=0.1, random\_state=42),

"AdaBoost": AdaBoostClassifier(n\_estimators=100, learning\_rate=0.5, random\_state=42)

}

# ------------------------------

# Evaluation function

# ------------------------------

def evaluate(y\_true, y\_pred, y\_prob):

acc = accuracy\_score(y\_true, y\_pred)

prec = precision\_score(y\_true, y\_pred)

rec = recall\_score(y\_true, y\_pred)

f1 = f1\_score(y\_true, y\_pred)

auc\_val = roc\_auc\_score(y\_true, y\_prob)

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

spec = cm[0, 0] / (cm[0, 0] + cm[0, 1]) # Specificity

return acc, prec, rec, f1, spec, auc\_val

# ------------------------------

# Run models and store results

# ------------------------------

results = {}

print("\nEvaluating models...\n")

for name, model in models.items():

print(f"🔍 {name}...")

y\_pred = cross\_val\_predict(model, X\_scaled, y, cv=cv)

y\_prob = cross\_val\_predict(model, X\_scaled, y, cv=cv, method='predict\_proba')[:, 1]

results[name] = evaluate(y, y\_pred, y\_prob)

# Create DataFrame of results

results\_df = pd.DataFrame(results, index=["Accuracy", "Precision", "Recall", "F1-Score", "Specificity", "AUC"]).T

results\_df = results\_df.round(4)

# ------------------------------

# 1️⃣ Mean Accuracy + AUC Plots

# ------------------------------

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

plt.bar(results\_df.index, results\_df["Accuracy"], color='skyblue')

plt.title("Mean Accuracy of Classifiers (10-Fold CV)")

plt.ylabel("Accuracy")

plt.ylim(0, 1)

plt.grid(axis='y')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

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

plt.bar(results\_df.index, results\_df["AUC"], color='salmon')

plt.title("Mean AUC of Classifiers (10-Fold CV)")

plt.ylabel("AUC")

plt.ylim(0, 1)

plt.grid(axis='y')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

# ------------------------------

# 2️⃣ ROC Curves (Indicative Run)

# ------------------------------

plt.figure(figsize=(10, 8))

for name, model in models.items():

model.fit(X\_scaled, y)

y\_prob = model.predict\_proba(X\_scaled)[:, 1]

fpr, tpr, \_ = roc\_curve(y, y\_prob)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'{name} (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Chance')

plt.title("ROC Curves (Indicative Run)")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend(loc="lower right")

plt.grid(True)

plt.tight\_layout()

plt.show()

# ------------------------------

# 3️⃣ Print Performance Table

# ------------------------------

print("\n📋 Summary of Classifier Performance Metrics:\n")

print(tabulate(results\_df, headers="keys", tablefmt="fancy\_grid"))

# Optional: save results

results\_df.to\_csv("model\_comparison\_results.csv")