TEAM 1- CARDIO CARE MACHINE LEARNING FOR HEALTH CARE

MODEL CODE

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

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

from sklearn.feature\_selection import SelectKBest, chi2

from imblearn.over\_sampling import SMOTE

data = pd.read\_csv("heart.csv")

data.fillna(data.mean(), inplace=True)

X = data.drop(columns=["target"])

y = data["target"]

chi2\_selector = SelectKBest(chi2, k=5)

X\_selected = chi2\_selector.fit\_transform(X, y)

selected\_features = X.columns[chi2\_selector.get\_support()].tolist()

scaler = StandardScaler()

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

smote = SMOTE(sampling\_strategy='auto', random\_state=42)

X\_res, y\_res = smote.fit\_resample(X\_scaled, y)

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

def objective\_function(params):

    C, gamma = params

    model = SVC(C=C, gamma=gamma, kernel='rbf')

    scores = cross\_val\_score(model, X\_train, y\_train, cv=5, scoring='accuracy')

    return np.mean(scores)

class JellyfishOptimization:

    def \_\_init\_\_(self, objective\_function, bounds, population\_size, iterations, alpha=3, beta=0.1):

        self.objective\_function = objective\_function

        self.bounds = np.array(bounds)

        self.population\_size = population\_size

        self.iterations = iterations

        self.alpha = alpha

        self.beta = beta

        self.population = None

        self.best\_solution = None

        self.best\_score = float("-inf")

    def initialize\_population(self):

        self.population = np.random.uniform(self.bounds[:, 0], self.bounds[:, 1], (self.population\_size, len(self.bounds)))

    def evaluate\_population(self):

        fitness = np.array([self.objective\_function(ind) for ind in self.population])

        best\_index = np.argmax(fitness)

        if fitness[best\_index] > self.best\_score:

            self.best\_score = fitness[best\_index]

            self.best\_solution = self.population[best\_index]

        return fitness

    def move\_jellyfish(self, fitness):

        new\_population = np.copy(self.population)

        for i in range(self.population\_size):

            if np.random.rand() < 0.5:

                r1, r2 = np.random.choice(self.population\_size, 2, replace=False)

                direction = self.population[r1] - self.population[r2]

                new\_position = self.population[i] + self.alpha \* np.random.rand() \* direction

            else:

                jellyfish\_mean = np.mean(self.population, axis=0)

                direction = jellyfish\_mean - self.population[i]

                new\_position = self.population[i] + self.beta \* np.random.rand() \* direction

            new\_population[i] = np.clip(new\_position, self.bounds[:, 0], self.bounds[:, 1])

        return new\_population

    def optimize(self):

        self.initialize\_population()

        for \_ in range(self.iterations):

            fitness = self.evaluate\_population()

            self.population = self.move\_jellyfish(fitness)

        return self.best\_solution

bounds = [(0.1, 1000), (0.001, 1)]

jellyfish = JellyfishOptimization(objective\_function, bounds, population\_size=10, iterations=50)

best\_params = jellyfish.optimize()

svm\_model = SVC(C=best\_params[0], gamma=best\_params[1], kernel='rbf')

svm\_model.fit(X\_train, y\_train)

y\_pred = svm\_model.predict(X\_test)

cv\_scores = cross\_val\_score(svm\_model, X\_train, y\_train, cv=5)

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

jfo\_precision = precision\_score(y\_test, y\_pred)

jfo\_recall = recall\_score(y\_test, y\_pred)

jfo\_f1 = f1\_score(y\_test, y\_pred)

print("Selected Features:", selected\_features)

print("Accuracy:", jfo\_accuracy)

print("Precision:", jfo\_precision)

print("Recall:", jfo\_recall)

print("F1 Score:", jfo\_f1)

print("Cross-Validation Scores:", cv\_scores)

print("Mean Cross-Validation Accuracy:", np.mean(cv\_scores))

import numpy as np

import matplotlib.pyplot as plt

models = ['SVM', 'KNN', 'Gradient Boosting', 'SVM with JFO']

metrics = {

    'SVM': [svm\_accuracy, svm\_precision, svm\_recall, svm\_f1],

    'KNN': [knn\_accuracy, knn\_precision, knn\_recall, knn\_f1],

    'Gradient Boosting': [gb\_accuracy, gb\_precision, gb\_recall, gb\_f1],

    'SVM with JFO': [jfo\_accuracy, jfo\_precision, jfo\_recall, jfo\_f1]

}

for model, values in metrics.items():

    print(f"{model}:")

    print(f"  Accuracy: {values[0]}")

    print(f"  Precision: {values[1]}")

    print(f"  Recall: {values[2]}")

    print(f"  F1 Score: {values[3]}\n")

x = np.arange(len(models))

width = 0.2

fig, ax = plt.subplots()

rects1 = ax.bar(x - 1.5\*width, [metrics[m][0] for m in models], width, label='Accuracy')

rects2 = ax.bar(x - 0.5\*width, [metrics[m][1] for m in models], width, label='Precision')

rects3 = ax.bar(x + 0.5\*width, [metrics[m][2] for m in models], width, label='Recall')

rects4 = ax.bar(x + 1.5\*width, [metrics[m][3] for m in models], width, label='F1 Score')

ax.set\_xlabel("Models")

ax.set\_ylabel("Score")

ax.set\_title("Performance Comparison of Models")

ax.set\_xticks(x)

ax.set\_xticklabels(models)

ax.legend()

plt.show()