import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

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

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.ensemble import VotingClassifier

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

from mrmr import mrmr\_classif # MRMR library

from salpSwarmOptimizer import SSA # SSA algorithm from library

# Step 1: Min-Max Normalization

def min\_max\_normalize(X):

scaler = MinMaxScaler()

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

return X\_scaled

# Step 2: Apply MRMR feature selection

def mrmr\_feature\_selection(X, y, n\_features):

selected\_features = mrmr\_classif(X, y, K=n\_features)

X\_selected = X[:, selected\_features]

return X\_selected

# Step 3: Apply SSA Optimization

def ssa\_optimization(X, y, n\_features):

# SSA configuration

optimizer = SSA(population\_size=50, iterations=100, n\_dim=n\_features, minimize=True)

# Objective function: could be a custom function based on your need, such as classification error

def objective\_function(features):

X\_subset = X[:, features]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, y, test\_size=0.3, random\_state=42)

clf = SVC() # Example: Use an SVM as objective

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

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

error = 1 - accuracy\_score(y\_test, y\_pred)

return error

best\_features = optimizer.optimize(objective\_function)

X\_optimized = X[:, best\_features]

return X\_optimized

# Step 4: Base Learners

def train\_base\_learners(X\_train, y\_train):

svm = SVC(probability=True)

nb = GaussianNB()

knn = KNeighborsClassifier()

mlp = MLPClassifier()

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

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

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

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

return svm, nb, knn, mlp

# Step 5: Ensemble Techniques (Voting Classifier)

def ensemble\_predictions(base\_learners, X\_test):

svm, nb, knn, mlp = base\_learners

ensemble = VotingClassifier(estimators=[('svm', svm), ('nb', nb), ('knn', knn), ('mlp', mlp)], voting='soft')

ensemble.fit(X\_test, y\_test)

return ensemble

# Step 6: Performance Calculation

def evaluate\_model(model, X\_test, y\_test):

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

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

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

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

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

return accuracy, precision, recall, f1

# Main Function

def main():

# Load the Braincancer dataset

data = pd.read\_csv('Braincancer.csv')

# Assuming the dataset has a 'target' column for labels

X = data.drop('target', axis=1).values # Features

y = data['target'].values # Labels

# Step 1: Normalize data

X\_normalized = min\_max\_normalize(X)

# Step 2: Feature Selection

X\_selected = mrmr\_feature\_selection(X\_normalized, y, n\_features=10)

# Step 3: SSA Optimization

X\_optimized = ssa\_optimization(X\_selected, y, n\_features=10)

# Split dataset for training and testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_optimized, y, test\_size=0.3, random\_state=42)

# Step 4: Base Learners

base\_learners = train\_base\_learners(X\_train, y\_train)

# Step 5: Ensemble Prediction

ensemble\_model = ensemble\_predictions(base\_learners, X\_test)

# Step 6: Evaluate the model

accuracy, precision, recall, f1 = evaluate\_model(ensemble\_model, X\_test, y\_test)

# Display performance

print(f"Accuracy: {accuracy:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1 Score: {f1:.2f}")

if \_\_name\_\_ == "\_\_main\_\_":

main()