import numpy as np

from qiskit\_aqua.input import SVMInput

from qiskit\_qcgpu\_provider import QCGPUProvider

from qiskit\_aqua import run\_algorithm

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.decomposition import PCA

# Define objective function for NSGA-II

def objective\_function(hyperparameters):

# Extract hyperparameters

depth, entanglement, shots = hyperparameters

# Prepare dataset

cancer = datasets.load\_breast\_cancer()

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(cancer.data, cancer.target, test\_size=0.3, random\_state=109)

scaler = StandardScaler().fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

pca = PCA(n\_components=2).fit(X\_train)

X\_train = pca.transform(X\_train)

X\_test = pca.transform(X\_test)

minmax\_scale = MinMaxScaler((-1, 1)).fit(X\_train)

X\_train = minmax\_scale.transform(X\_train)

X\_test = minmax\_scale.transform(X\_test)

# Prepare QSVM input

training\_input = {'Benign': X\_train[Y\_train == 0], 'Malignant': X\_train[Y\_train == 1]}

test\_input = {'Benign': X\_test[Y\_test == 0], 'Malignant': X\_test[Y\_test == 1]}

algo\_input = SVMInput(training\_input, test\_input, test\_input)

# Define QSVM parameters

params = {

'problem': {'name': 'svm\_classification', 'random\_seed': 10598},

'algorithm': { 'name': 'QSVM.Kernel' },

'backend': {'name': 'qasm\_simulator', 'shots': shots},

'feature\_map': {'name': 'SecondOrderExpansion', 'depth': depth, 'entanglement': entanglement}

}

# Run QSVM

result = run\_algorithm(params, algo\_input)

# Evaluate performance

accuracy = result['testing\_accuracy']

# Additional performance metrics can be computed

return accuracy

# Implement NSGA-II

def nsga2(objective\_function, population\_size, generations):

# Define hyperparameter search space

depth\_range = [1, 2, 3]

entanglement\_options = ['linear', 'full']

shots\_range = [1024, 2048, 4096]

# Initialize population

population = []

for \_ in range(population\_size):

depth = np.random.choice(depth\_range)

entanglement = np.random.choice(entanglement\_options)

shots = np.random.choice(shots\_range)

population.append((depth, entanglement, shots))

# Main loop

for \_ in range(generations):

# Evaluate population

fitness\_values = [objective\_function(individual) for individual in population]

# Select parents using NSGA-II selection mechanism

# Perform crossover and mutation

# Generate next generation

# Update population

# Repeat until convergence or maximum generations

# Return Pareto-optimal solutions

pareto\_front = ... # Implement Pareto front extraction

return pareto\_front

# Example usage

pareto\_front = nsga2(objective\_function, population\_size=10, generations=5)

for solution in pareto\_front:

print("Hyperparameters:", solution)