from qiskit import BasicAer

from qiskit.utils import QuantumInstance, algorithm\_globals

from qiskit.algorithms.optimizers import COBYLA

from qiskit.circuit.library import TwoLocal, ZZFeatureMap

from qiskit\_machine\_learning.algorithms import VQC

from qiskit\_machine\_learning.datasets import ad\_hoc\_data

seed = 1376

algorithm\_globals.random\_seed = seed

# Use ad hoc data set for training and test data

feature\_dim = 2 # dimension of each data point

training\_size = 20

test\_size = 10

# training features, training labels, test features, test labels as np.array,

# one hot encoding for labels

training\_features, training\_labels, test\_features, test\_labels = \

ad\_hoc\_data(

training\_size=training\_size, test\_size=test\_size, n=feature\_dim, gap=0.3)

feature\_map = ZZFeatureMap(feature\_dimension=feature\_dim, reps=2, entanglement="linear")

ansatz = TwoLocal(feature\_map.num\_qubits, ['ry', 'rz'], 'cz', reps=3)

vqc = VQC(feature\_map=feature\_map,

ansatz=ansatz,

optimizer=COBYLA(maxiter=100),

quantum\_instance=QuantumInstance(BasicAer.get\_backend('statevector\_simulator'),

shots=1024,

seed\_simulator=seed,

seed\_transpiler=seed)

)

vqc.fit(training\_features, training\_labels)

score = vqc.score(test\_features, test\_labels)

print(f"Testing accuracy: {score:0.2f}")

# define data (input array X, target labels y)

from qiskit\_machine\_learning.datasets import ad\_hoc\_data

X\_train, y\_train, X\_test, y\_test = ad\_hoc\_data(20, 10, 2, 0.1)

# import the variational quantum classifier

from qiskit\_machine\_learning.algorithms import VQC

# import the feature map and ansatz circuits

from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes

# import the optimizer for the training

from qiskit.algorithms.optimizers import L\_BFGS\_B

# import backend

from qiskit.providers.aer import QasmSimulator

# construct classifier

num\_qubits = 2

vqc = VQC(feature\_map=ZZFeatureMap(num\_qubits),

ansatz=RealAmplitudes(num\_qubits, reps=1),

loss='cross\_entropy',

optimizer=L\_BFGS\_B(),

quantum\_instance=QasmSimulator())

# train classifier

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

# score result

vqc.score(X\_test, y\_test)

https://qiskit-quantum-knn.readthedocs.io/en/latest/

import qiskit\_quantum\_knn

from qiskit\_quantum\_knn.qknn import QKNeighborsClassifier

from qiskit\_quantum\_knn.encoding import analog

from qiskit import aqua

from sklearn import datasets

import qiskit as qk

# initialising the quantum instance

backend = qk.BasicAer.get\_backend('qasm\_simulator')

instance = aqua.QuantumInstance(backend, shots=10000)

# initialising the qknn model

qknn = QKNeighborsClassifier(

n\_neighbors=3,

quantum\_instance=instance

)

n\_variables = 2 # should be positive power of 2

n\_train\_points = 4 # can be any positive integer

n\_test\_points = 2 # can be any positive integer

# use iris dataset

iris = datasets.load\_iris()

labels = iris.target

data\_raw = iris.data

# encode data

encoded\_data = analog.encode(data\_raw[:, :n\_variables])

# now pick these indices from the data

train\_data = encoded\_data[:n\_train\_points]

train\_labels = labels[:n\_train\_points]

test\_data = encoded\_data[n\_train\_points:(n\_train\_points+n\_test\_points), :n\_variables]

test\_labels = labels[n\_train\_points:(n\_train\_points+n\_test\_points)]

qknn.fit(train\_data, train\_labels)

qknn\_prediction = qknn.predict(test\_data)

print(qknn\_prediction)

print(test\_labels)