Tutorial

- Official Tutorial: pyQPanda
- Code path: /cefs/higgs/shaqy/Quantum/QC_HEP/For_tutorial

Set up:

- pip install pyqpanda

Requirements:

Linux

software	version
GCC	>= 5.4.0
Python	>= 3.7.0 && <= 3.9.0

Everything is similar as IBM:

Different: Wuyuan don't have enough built-in functions like QSVM. We need to do by ourself.

- Only three steps: Use IBM Qiskit to generate QSVM kernel (gen_qasm.py shown in page 2.)

↓

QASM(Quantum Assembly Language):

↓

Convert QASM to program in wuyuan (page 3)
Get the kernel matrix.

↓

The same way as IBM:
Use classical way to get results.

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[6];
creg c[6];
u2(0,pi) q[0];
rz(0.522150994) q[0];
u3(0.261075497,0.0,0.0) q[0];
u3(0.658964753,0.0,0.0) q[0];
rz(1.317929506) q[0];
u2(0,pi) q[0];
u2(0,pi) q[1];
rz(0.845885038) q[1];
u3(0.422942519,0.0,0.0) q[1];
u3(0.0224528909,0.0,0.0) q[1];
rz(0.0449057818) q[1];
u2(0,pi) q[1];
```

Qiskit part

Same as IBM tutorial:

- Create a feature map and kernel using Qiskit

```
rng = np.random.RandomState(0)
def gen_qasm():
  seed=2022
  backend = BasicAer.get_backend("statevector_simulator")
  feature_map_cus = customised_feature_maps.FeatureMap(num_qubits=6, depth=1, degree=1, entanglement='full', inverse=False)


  for i in range(1):
    seed = 2022
    algorithm_globals.random_seed = seed

    data = pd.read_csv("sample_%d.csv" % i)
    train = data[0:100]
    test = data[100:200]
    train_label = train.pop('tag')
    test_label = test.pop('tag')

    train_data = train.to_numpy()
    test_data = test.to_numpy()
    X_train = train_data
    X_test = test_data

    q_backend = QuantumInstance(backend, shots=10, seed_simulator=None, seed_transpiler=None)
    q_kernel = QuantumKernel(feature_map=feature_map_cus, quantum_instance=q_backend)

    qsvm_kernel_matrix_train = q_kernel.evaluate(x_vec=X_train)
    qsvm_kernel_matrix_test = q_kernel.evaluate(x_vec=X_test,y_vec=X_train)
    kernel_train_IBM = np.asmatrix(qsvm_kernel_matrix_train)
    kernel_test_IBM = np.asmatrix(qsvm_kernel_matrix_test)

    x = ParameterVector('x', length=6)
    y = ParameterVector('y', length=6)
    circuit = q_kernel.construct_circuit(x,y)
    circuit = q_backend.transpile(circuit)[0]

    for i in range(100):
      for j in range(100):
        f = open("all_qasm_train/qasm_%d_%d.txt"%(i,j),'w')
        cirtemp = circuit.assign_parameters({x:X_train[i], y:X_train[j]}, inplace=False)
        f.write(cirtemp.qasm())
        f.close()
```

- Official Tutorial: pyQPanda
- Code path: /cefs/higgs/shaqy/Quantum/QC_HEP/For_tutorial

Different part: (In red frame.)

- Save the dataset into a file(txt or .csv)
- For different (i,j) in train/test dataset:
  - Generate a QASM file.
  - One file represent a feature map for one point in kernel matrix.

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[6];
creg c[6];
u2(0,pi) q[0];
rz(0.522150994) q[0];
u3(0.261075497,0.0,0.0) q[0];
u3(0.658964753,0.0,0.0) q[0];
rz(1.317929506) q[0];
u2(0,pi) q[0];
u2(0,pi) q[1];
rz(0.845885038) q[1];
u3(0.422942519,0.0,0.0) q[1];
u3(0.0224528909,0.0,0.0) q[1];
rz(0.0449057818) q[1];
u2(0,pi) q[1];
```

OriginQ wuyuan part

- Official Tutorial: pyQPanda
- Code path: /cefs/higgs/shaqy/Quantum/QC_HEP/For_tutorial

- Apply a key in the OriginQ website.(One key can run 1000 jobs in one day.)
- Apply X qubits and classical bits which used to save the result.
- Convert QASM to program.
- Use real_chip_type:origin_wuyuan_d5 to run. (Only d4 and d5 can use, d5 is better)
- The value with [‘00000’] is the point of kernel matrix. Save it.

```
from pyqpanda import *

def run(i,j):
    QCM = QCloud()
    QCM.init_qvm("58DCD70A14814A4DB73ACF5C8F854FA2"

    qlist = QCM.qAlloc_many(5)
    clist = QCM.cAlloc_many(5)

    qvm = init_quantum_machine(QMachineType.CPU)
    qvm.init_qvm()

    prog_trans, qv, cv = convert_qasm_to_qprog("all_qasm_train/qasm_%d_%d.txt"%(i,j), qvm)
    try:
        result = QCM.real_chip_measure(prog_trans, 10000, real_chip_type.origin_wuyuan_d5)
    except:
        print("job %d %d failed" % (i,j))
        return
    value = result['00000']
    f = open("results_train_try/result_%d_%d.txt" %(i,j),'w')
    f.write(str(value))
    f.close()

if __name__=="__main__":

  for i in range(19,20):
    for j in range(i+1,100):
      run(i,j)
```

Classical part

- Official Tutorial: pyQPanda
- Code path: /cefs/higgs/shaqy/Quantum/QC_HEP/For_tutorial

➢ Import the kernel matrix.

```
score = []
for i in range(1):

  data = pd.read_csv("sample_%d.csv" % i)
  train = data[0:100]
  test = data[100:200]
  train_label = train.pop('tag')
  test_label = test.pop('tag')
  train_label_oh = label_binarize(train_label, classes=[1,-1])
  test_label_oh = label_binarize(test_label, classes=[1,-1])

  test_kernel = []
  for i in range(100):
    test_kernel_line = []
    for j in range(100):
      value = get_kernel(i,j,'test')
      test_kernel_line.append(value)
    test_kernel.append(test_kernel_line)
  test_kernel_array = np.array(test_kernel, np.float32)

  train_kernel = []
  for i in range(100):
    train_kernel_line = []
    for j in range(100):
      if i == j:
        train_kernel_line.append(1.0)
      else:
        if i>j:
          # the matrix is symmetric
          # switch i and j as j is always greater than i in data
          value = get_kernel(j,i,'train')
          train_kernel_line.append(value)
        else:
          value = get_kernel(i,j,'train')
          train_kernel_line.append(value)
    train_kernel.append(train_kernel_line)
  train_kernel_array = np.array(train_kernel, np.float32)
```

➢ Then use the classical svc to get the final results, generate AUC value and draw plots.

```
#QSVM
    svc = SVC(C=30, probability=True, kernel="precomputed")
    #svc = QSVC(C=5, probability=True, quantum_kernel="precomputed")
    csvc = svc.fit(train_kernel_array, train_label)
    predictions = csvc.predict_proba(test_kernel)
    fpr, tpr, _ = sklearn.metrics.roc_curve(test_label_oh, predictions[:, 0])
```