THE UNIVERSITY OF MELBOURNE

Qiskit-AI based Prediction for Financial Wellbeing

Speaker: Ming Chen
mingc4@student.unimelb.edu.au

Team*: Anthony Hsu (BMM Lab), Ming Chen (CS) , Nitin Yadav (BMM Lab) Coach: Desiree (IBM) and Charles (Physics)

Problem Introduction

Financial wellbeing involves measuring people's financial comfort in the context of their day to day expenses, future savings (e.g., retirement) and their resilience to financial shocks.

The plan is to **train quantum machine learning algorithm**(s) over **real world financial wellbeing data** with the aim to better model financial wellbeing measure given the parameters.

```
df_raw
```

✓ 0.6s

	fwb	sample	fs_score	fs_assess	fk_interest	fk_inflation	fk_vola	living
0	1	2	44	4	1	1	1	1
1	1	1	43	3	1	1	1	2
2	0	1	42	3	1	1	1	2
3	0	1	42	3	1	0	1	-1
4	0	1	42	3	0	0	1	3
...	...	...	...	...	...	...	...	...
6384	1	3	47	3	1	1	1	2
6385	1	3	59	4	1	0	0	2
6386	1	1	51	3	1	0	1	2
6387	0	1	54	4	1	0	1	2
6388	0	3	42	3	1	0	1	2

6389 rows × 8 columns

Problem Solving

Model used-

```
from qiskit_machine_learning.algorithms import PegasosQSVC
```

```
from qiskit_machine_learning.algorithms import PegasosQSVC

# should test with different (tau, C), to find the optimal scores.
# Doing grid search

print("PegasosQSVC classification grid search param set and mean accuracy: ")
count = 1

for C in qSVC_param_grid['C_grid']: # regulaztion param
    for tau in qSVC_param_grid['tau_grid']: # iterations
        pegasos_qsvc = PegasosQSVC(quantum_kernel=qkernel, C=C, num_steps=tau)
        # training
        pegasos_qsvc.fit(train_features, train_labels)

        # testing
        # Output the mean accuracy on the given test data and labels.
        pegasos_score = pegasos_qsvc.score(test_features, test_labels)

        print("ParmaSet@" + str(count))
        print(f"C : {C},  tau : {tau}, test_accuracy : {pegasos_score}")
        count+=1

        #print(f" test score: {pegasos_score}")
```

Pegasos: Primal Estimated sub-GrAdient SOlver for SVM

INPUT: S, λ, T
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
  Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.
  Set $\eta_t = \frac{1}{\lambda t}$
  If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:
    Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$
  Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):
    Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$
  [Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$]
OUTPUT: $\mathbf{w}_{T+1}$

Experimental Results

Optim Methods ---- Circuit Encoding + Grid Search :

being.ipynb | QSVM_example.ipynb | QSVM_FWB.ipynb | QSVM_FW

```
C : 2000,  tau : 20, test_accuracy : 0.6580594679186228
ParmaSet@26
C : 2000,  tau : 50, test_accuracy : 0.6807511737089202
ParmaSet@27
C : 2000,  tau : 100, test_accuracy : 0.6369327073552425
ParmaSet@28
C : 2000,  tau : 200, test_accuracy : 0.6917057902973396
ParmaSet@29
C : 2000,  tau : 400, test_accuracy : 0.701095461658842
ParmaSet@30
C : 2000,  tau : 1000, test_accuracy : 0.7034428794992176
ParmaSet@31
C : 5000,  tau : 20, test_accuracy : 0.45774647887323944
ParmaSet@32
C : 5000,  tau : 50, test_accuracy : 0.6697965571205008
ParmaSet@33
C : 5000,  tau : 100, test_accuracy : 0.6697965571205008
ParmaSet@34
C : 5000,  tau : 200, test_accuracy : 0.6697965571205008
ParmaSet@35
C : 5000,  tau : 400, test_accuracy : 0.7339593114241002
ParmaSet@36
C : 5000,  tau : 1000, test_accuracy : 0.6744913928012519
ParmaSet@37
C : 20000,  tau : 20, test_accuracy : 0.6025039123630673
ParmaSet@38
C : 20000,  tau : 50, test_accuracy : 0.6580594679186228
ParmaSet@39
C : 20000,  tau : 100, test_accuracy : 0.6964006259780907
ParmaSet@40
C : 20000,  tau : 200, test_accuracy : 0.7089201877934272
ParmaSet@41
C : 20000,  tau : 400, test_accuracy : 0.5477308294209703
ParmaSet@42
```

Data encoding circuits

These `BlueprintCircuit` encode classical data in quantum states and are used as feature maps for cl

`PauliFeatureMap` ([feature_dimension, reps, ...])	The Pauli Expansion circuit.
`ZFeatureMap` (feature_dimension[, reps, ...])	The first order Pauli Z-evolution circuit.
`ZZFeatureMap` (feature_dimension[, reps, ...])	Second-order Pauli-Z evolution circuit.
`StatePreparation` (params[, num_qubits, ...])	Complex amplitude state preparation.

Best Result till now----
featureMap: ZFeatureMap()
C : 5000,
tau : 400,
test_accuracy : 0.7339593114241002