"From zero to hero" Qiskit Machine Learning tutorial on a real dataset

QAMP Spring 2022

Project #5

Final Checkpoint 9-Jun-2022

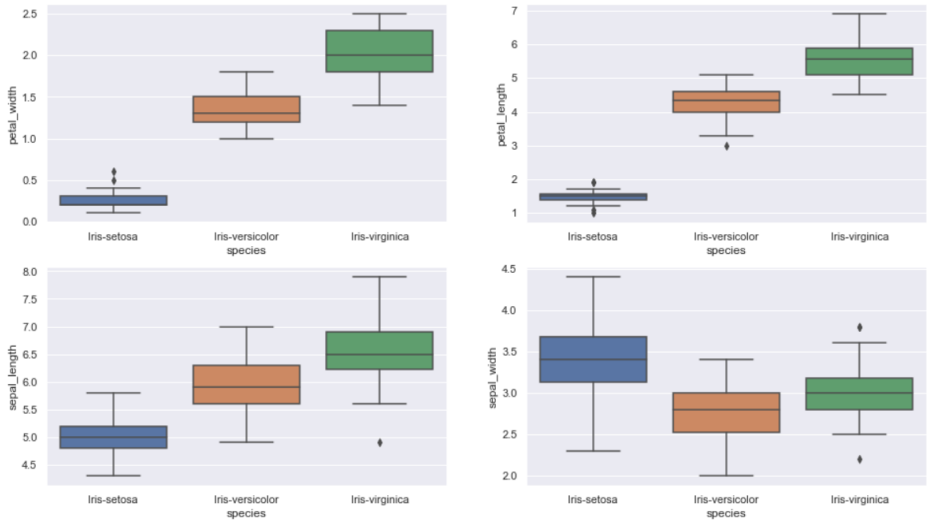
Mentor: Anton Dekusar
Mentee: Eric Michiels

Qiskit

Variational Quantum Classifier applied on the Iris dataset

- Analysis of Iris Dataset

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.datasets import load_iris
        import time

        iris_data = load_iris()
        iris_data_df = pd.DataFrame(data=iris_data.data, columns=iris_data.feature_names)
        iris_data_df["target"] = iris_data.target
```



Petals have bright colors

Sepals are green in color

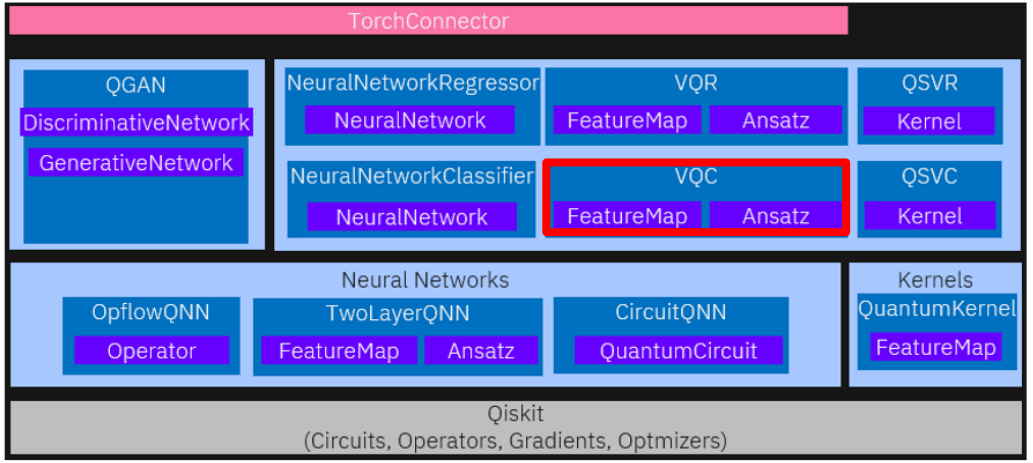
```
In [3]: # What are the names of the species and features?
        print(set(iris_data["target"]))
        print(iris_data["target_names"])
        print(iris_data["feature_names"])

        {0, 1, 2}
        ['setosa' 'versicolor' 'virginica']
        ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

In [4]: # What is the shape of the Iris Dataset?
        iris_data_df.shape

Out[4]: (150, 5)
```

Observations:

- Setosa has smaller characteristics and is less distributed
- Versicolor is distributed averagely and has average characteristics (features)
- Virginica has large features and is highly distributed.

Variational Quantum Classifier applied on the Iris dataset

- Which part of Qiskit is being explored?



For Training, an instance of an optimizer is used

- COBYLA
- SLSQP
- SPSA
- ...

```
optimizer = COBYLA(maxiter=120, tol=0.001)
```

A callback function is used by some optimizers, allow us to follow up how training is going on, in the process of optimizing an objective function

```
# Callback function that draws a live plot when the .fit() method is called
def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()
```

Variational Quantum Classifier applied on the Iris dataset

- Data Normalization

```
mms = MinMaxScaler()
X_features = iris_data['data']
X_features = mms.fit_transform(X_features)
```

- VQC expects One-Hot Encoding

```
labels = iris_data['target']
labels = OneHotEncoder(sparse=False).fit_transform(labels.reshape(-1, 1))
```

- Eliminate the impact of varying pseudo random numbers

```
algorithm_globals.random_seed = 123
```

Variational Quantum Classifier applied on the Iris dataset

- Feature Map

```
feature_dim = 4
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=1, entanglement='full')
ZZ_feature_map.draw('mpl')
```

- Ansatz

```
num_qubits = feature_dim
variational_circ = RealAmplitudes(num_qubits, entanglement='full', reps=4)
variational_circ.draw('mpl')
```

- Backend and Initial Point

```
backend = Aer.get_backend('statevector_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=algorithm_globals.random_seed,
        seed_transpiler=algorithm_globals.random_seed)
initial_point = algorithm_globals.random.random(variational_circ.num_parameters)
```

Variational Quantum Classifier applied on the Iris dataset

- Split Up the Data Set

```
X_train, X_test, y_train, y_test = train_test_split(X_features, labels, test_size = 0.2,
                                                    random_state=algorithm_globals.random_seed)
```

- Initialize VQC

```
vqc = VQC(feature_map=ZZ_feature_map,
          ansatz=variational_circ,
          optimizer=optimizer,
          quantum_instance=quantum_instance,
          initial_point=initial_point,
          callback=callback_graph)
```

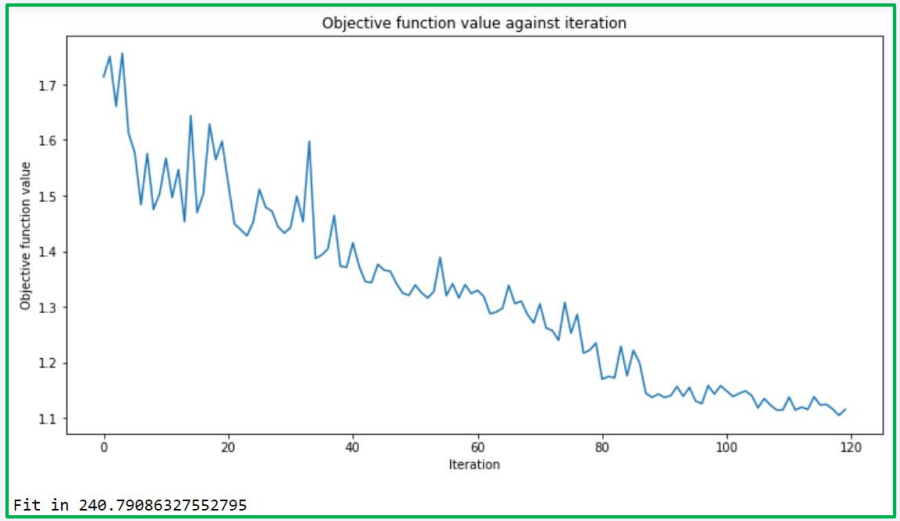
- Start a timer

```
start = time.time()
```

Variational Quantum Classifier applied on the Iris dataset

- Follow up the training

```
objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

# fit classifier to data
vqc.fit(X_train, y_train)

elapsed = time.time() - start
print(f"Fit in {elapsed}")

# return to default figsize
plt.rcParams["figure.figsize"] = (6, 4)

plt.show()
```



- Score the classifier

```
print(f"Q train score: {vqc.score(X_train, y_train)}")
print(f"Q test score : {vqc.score(X_test, y_test)}")
```

```
Q train score: 0.825
Q test score : 0.8
```

Variational Quantum Classifier applied on the Iris dataset

- Additional Experiments

```
optimizer = SPSA(maxiter=10)
```

```
Fit in 138.7473428249359
Q train score: 0.575
Q test score : 0.5333333333333333
```

```
ZZ_feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=2, entanglement='circular')
```

```
Fit in 210.19560956954956
Q train score: 0.8083333333333333
Q test score : 0.7333333333333333
```

```
variational_circ = RealAmplitudes(num_qubits, entanglement='circular', reps=3)
```

```
Fit in 216.05357551574707
Q train score: 0.7666666666666667
Q test score : 0.7
```

Variational Quantum Classifier applied on the Iris dataset

- Additional Experiments

```
variational_circ = EfficientSU2(num_qubits)
```

```
Fit in 260.9183466434479
Q train score: 0.8416666666666667
Q test score : 0.8333333333333334
```

```
backend = Aer.get_backend('aer_simulator')
```

```
Fit in 275.43174409866333
Q train score: 0.8416666666666667
Q test score : 0.6666666666666666
```

- Best Combination of Options so far

- COBYLA as optmizer
- ZZFeatureMap as Feature Map with 1 repetition and full entanglement
- EfficientSU2 as Ansatz
- Statevector_simulator as backend

- Many more experiments could be done …

Lessons Learned

- Know your data structure and content !
- Selecting the right parameter combination is not obvious
- VQC and QSVM are “different animals”

- Personal feedback:
  - Time issues ☹
  - The journey is not finished ☺