Advanced Concepts in Machine Learning
Assignment 4

Eliott Simon, Kevin Müller

November 2021

1 First Assignment

1.1 Experiments with number of replicators

Since we can have the feature map which can be replicated multiple times(based on the number of replicator), we tested the model's accuracy according to different number of configurations.

		Feature Map			
Replicators		1	2	3	4
Amplitude	1	0.8	0.825	0.925	0.75

From this table, it seems that the best configuration involves the feature to use 3 replicators, since this yields to the best accuracy.

1.2 Data Encoding File

To start with the assignment, we went looking at the encoding data strategy (feature_map) as it is suggested in the Assignment. The given code uses **ZZFeatureMap** to encode the data into the quantum state space. It is initialized with the parameters n = 2 as feature_dimension and with reps = 1, which is the number of repeated circuits.

As it is explained in the DataEncoding file, there are currently 3 available feature maps that can be used to encode the data in Qiskit.

1. **ZZFeatureMap** - Second-order Pauli-Z evolution circuit.
2. **ZFeatureMap** - First-order Pauli-Z evolution circuit.
3. **PauliFeatureMap** - More general form of feature map. Own paulis (list of strings) can be defined. The default value equals ZZFeatureMap. First- and second-order Pauli-Z evolution circuits are subclasses of PauliFeatureMap.

To start we just wanted to find out what the difference between the first- and second-order Pauli-Z evolution circuit is. With the given code we can score over **80% accuracy** when no parameters are changed and we use **ZZFeatureMap**.

When we use **ZFeatureMap** the accuracy goes down towards **70%**. We think that this comes from the fact, that in the first-order only one state is possible. Therefore there can be no interaction between the features of the encoded data and we are not able to use entanglement. Because we use 2 features, **ZZFeatureMap** seems to be the right feature map for the given task.

To reach better classification with **ZZFeatureMap** we then explored the structure of the variational circuit (var_form). The given code uses *var_form = RealAmplitudes(n, reps=1).* We looked at all the different var forms available and let them run to see how the performance changes. Other than **RealAmplitudes**, there are:

1. **TwoLocal** - The two-local circuit. Using the same code and just changing to TwoLocal, resulted in a **35%** accuracy score on the classification.
2. **NLocal** - The n-local circuit class.
3. **EfficientSU2** - Hardware efficient version of 2-local circuit.

So just testing on the vanilla code without changing the parameters, **RealAmplitudes** gave the best results for the variational circuit for this task. That's why we decided to first also keep that and look at the different optimizers there are.

The given code uses **COBYLA (Contrained Optimization By Linear Approximation optimizer)**, which is a numerical opzimization method for constrained problems, where the derivative of the objective function is not known. Other ones we wanted to test on the given code are:

1. SPSA (Simultaneous Perturbation Stochastic Approximation optimizer) - In this experiment we got **82.5%** which is about the same as in the default setup.
2. SLSQP (Sequential Least Squares Programming optimize) - Here we got **80%** accuracy, which is also equal to the default COBYLA.

So both other optimizers give similar results as COBYLA, so we also decided to stay with it. Since none of the changes really improved the accuracy by itself, we looked more into the different parameters, that are able to be changed.

For that we looked into the documentation and tried various parameters with the feature map of ZZFeatureMap. One idea was to increase **reps** (number of repeated circuits) to see which influence a repeated circuit has on the computation. We chose for the final result as **reps = 2**, because with that we could achieve 90 - 100% accuracy in predicting. Another observation is that already for a value of reps = 3, the prediction gets worse again (75% accuracy). So reps = 2 seems to be the perfect choice here, because with reps = 1 we can only achieve in between 80 - 90%.

1.3 Adding more noise to the data

In this section, we will investigate how the model deals with data that contains much more noise (i.e the data is very hard to partition into two sets). To do that, we can play with the *gap* parameter in the *ad_hoc_data*. By using a value of 0.01, the data looks as follows:

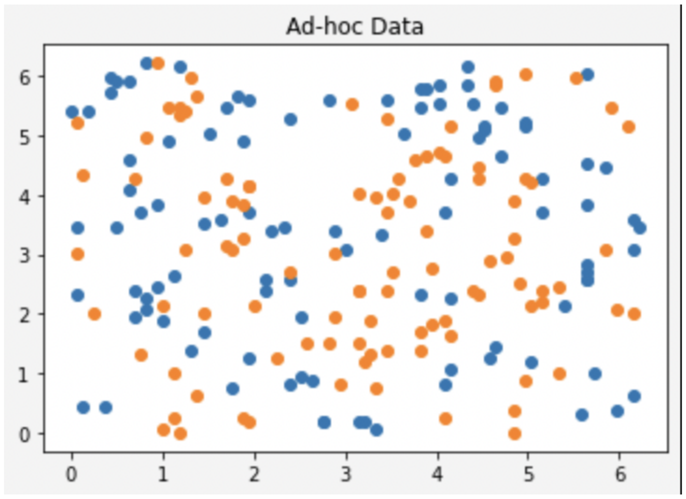

Figure 1: Noisy data

It is obvious that this data is not easy to handle since the instances are very noisy. When applying the model to this data, we get an accuracy = 67.5%.

2 Second Assignment

Find the complete final code attached in canvas. For the complete overview in the report, we added just the parts that we have done ourselves to solve the excercises.

2.1 Excercise 1

```
x = [-0.1,0.2]

encode_map = ZZFeatureMap(feature_dimension=2,
reps=4, entanglement='linear', insert_barriers=True)
encode_circuit = encode_map.bind_parameters(x)
encode_circuit.decompose().draw(output='mpl')
```

2.2 Excercise 2

```
x = [-0.1,0.2]
y = [0.4,-0.6]

zz_map = ZZFeatureMap(feature_dimension=2, reps=4,
entanglement='linear', insert_barriers=True)
zz_kernel = QuantumKernel(feature_map=zz_map, quantum_instance=Aer.
get_backend('statevector_simulator'))
zz_circuit = zz_kernel.construct_circuit(x, y)

backend = Aer.get_backend('qasm_simulator')
job = execute(zz_circuit, backend, shots=8192,
                seed_simulator=1024, seed_transpiler=1024)
counts = job.result().get_counts(zz_circuit)

counts['00']/sum(counts.values())
```