IBM **Quantum**

Quantum Kernel Estimation

Hakan Doga, PhD
IBM Quantum

What are kernel methods in general?

- Kernel methods are a powerful class of machine learning algorithm that allow us to perform complex, non-linear transformations of data without explicitly computing the transformed feature space.

- A **kernel function** computes the dot product (inner product) of two vectors in a transformed feature space without explicitly performing the transformation. This is also known as the **kernel trick**.

- Various flavors of these methods can be used for classification tasks such as image recognition, bioinformatics or text categorization.

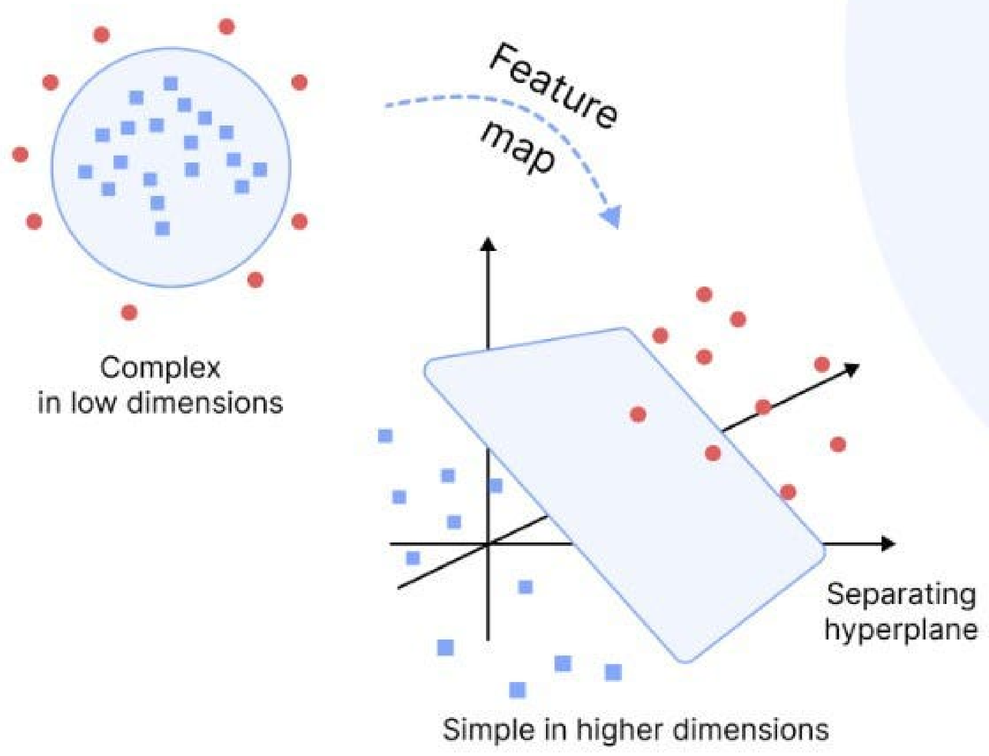
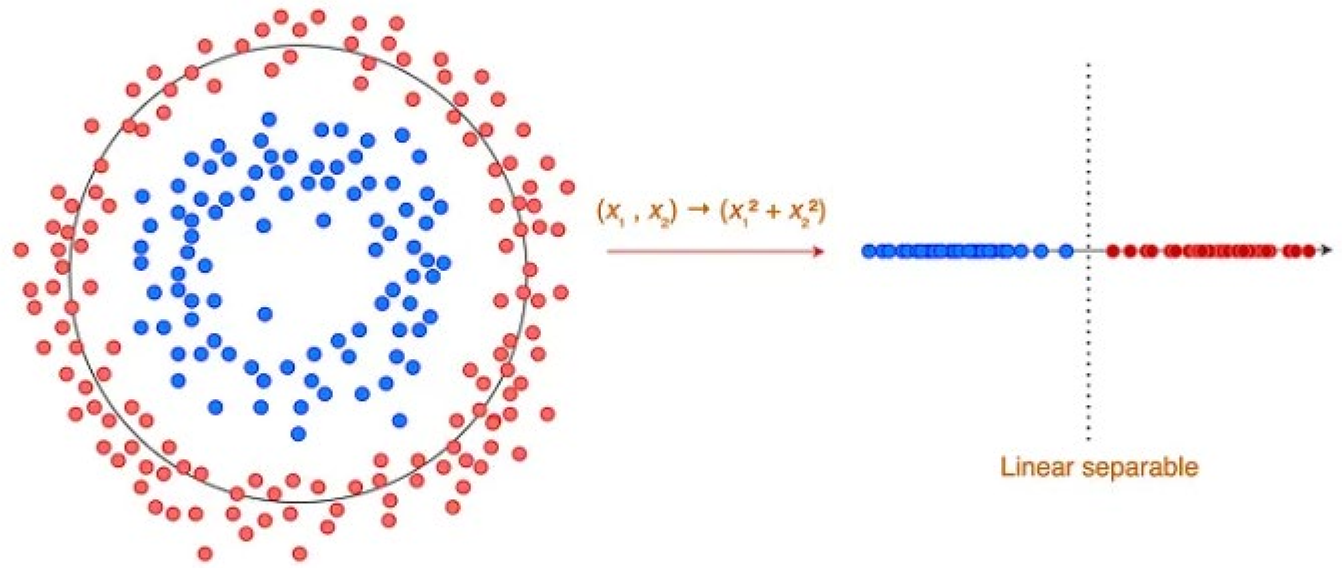


Image source: https://medium.com/@qjbqvwzmg/kernel-methods-in-machine-learning-theory-and-practice-b030bbe0eacc

Let’s take a closer look into kernels

- Recall that a feature map $\Phi\colon\ \mathbb{R}^d \to\ \mathbb{R}^{d'}$ mapping a feature vector $\vec{x}$ to usually a higher dimensional space, i.e. $d' > d$ so that the data is linearly separable.

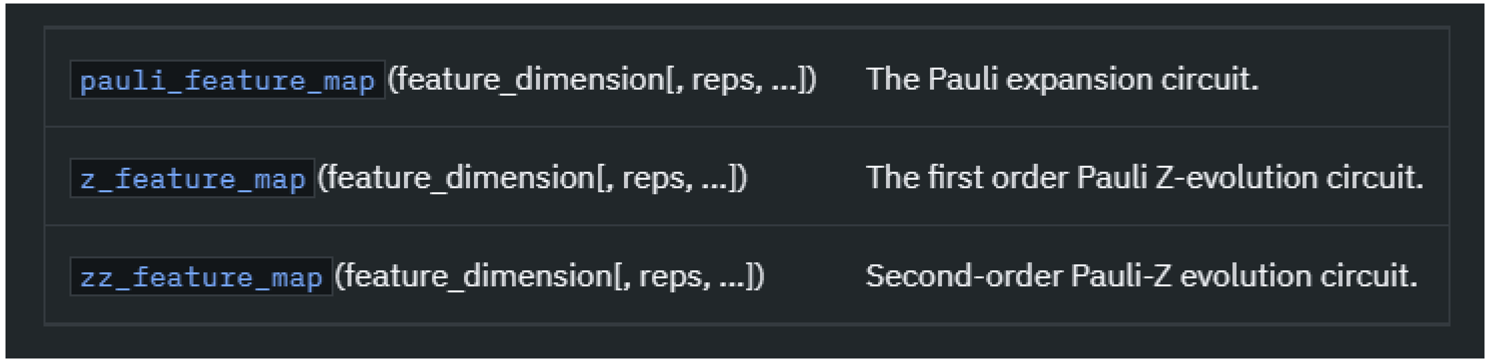
- A kernel function $K\colon\ \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that takes a pair of feature-mapped vectors and returns their inner product, i.e. $K(x, y) = \langle \Phi(x) | \Phi(y) \rangle$.



- A standard example where the data is not linearly separable in $\mathbb{R}^2$, in this simple case we can get away with the feature map $\Phi(x_1, x_2) = x_1^2 + x_2^2$ which linearly separates it, making it possible to classify the data more efficiently.

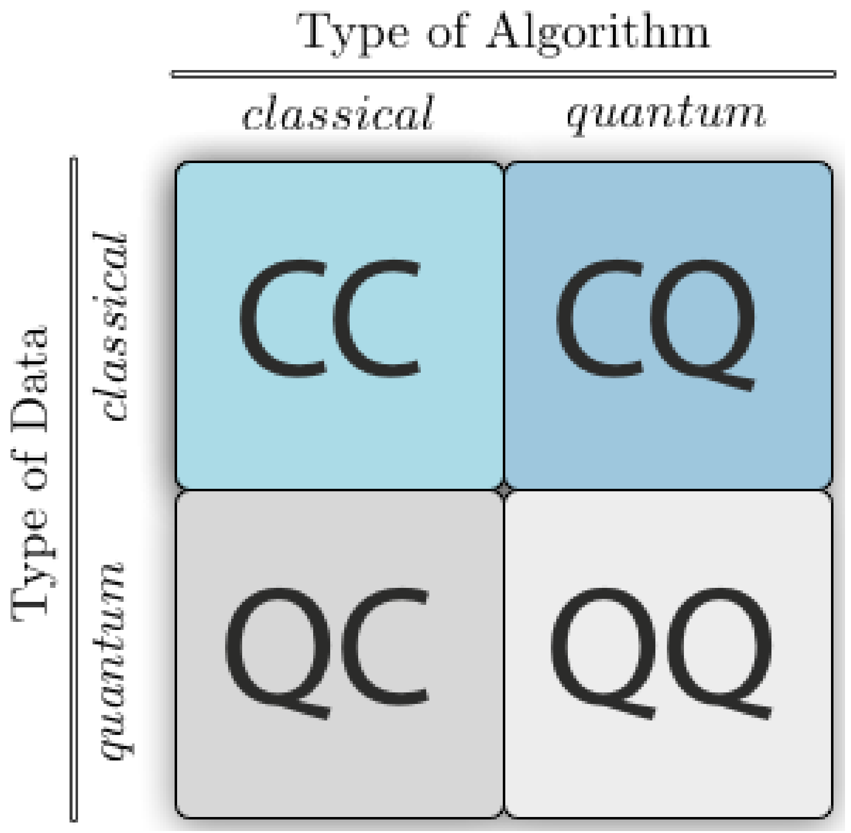
How does it work in quantum?

- Similar to classical ML, it all starts with data.
- An immediate challenge in QML in general is how do we encode classical information into a quantum computer?
- The data encoding is usually achieved through a specific parametrized quantum circuit that “loads” the data as quantum states.
- Qiskit circuit library contains several such functions, however one can also build a custom data encoding circuit that represents the data more accurately!



`pauli_feature_map`(feature_dimension[, reps, ...])	The Pauli expansion circuit.
`z_feature_map`(feature_dimension[, reps, ...])	The first order Pauli Z-evolution circuit.
`zz_feature_map`(feature_dimension[, reps, ...])	Second-order Pauli-Z evolution circuit.

Link to the Qiskit circuit library: https://docs.quantum.ibm.com/api/qiskit/circuit_library#data-encoding-circuits



Source:https://en.wikipedia.org/wiki/Quantum_machine_learning#/media/File:Qml_approaches.tif

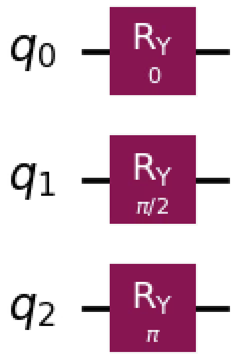
Building the kernel in quantum (some unsolicited math)

- Say for given two data vectors $\vec{x_i}$ and $\vec{x_j}$, let Φ be the quantum circuit that encodes these data vectors. Since all N-qubits are initialized in the $|0\rangle$ state, after applying the feature mapping circuit we have

$$\left.\begin{aligned}|\psi(\vec{x_i})\rangle &= \Phi(\vec{x_i})|0\rangle^{\otimes N}\\ |\psi(\vec{x_j})\rangle &= \Phi(\vec{x_j})|0\rangle^{\otimes N}\end{aligned}\right\}$$

$|.\rangle$ is always a column vector such as $\begin{pmatrix}1\\0\end{pmatrix}$

Φ is also a unitary matrix such as $\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$



$\longrightarrow$

A simple example of an angle encoding feature mapping circuit that encodes the data vector $\vec{x_j} = (0, \frac{\pi}{4}, \frac{\pi}{2})$

Building the kernel in quantum (some unsolicited math)

- Recall that we want to be able to compute the inner products of these feature-mapped vectors (now in quantum) as part of the kernel trick (or function).

- One way to perform this in the form of a quantum circuit is through UnitaryOverlap circuit which applies $\langle . | . \rangle$ operation. So, our kernel entry will be:

$$\langle \psi(\overrightarrow{x_i}) | \psi(\overrightarrow{x_j}) \rangle = \langle 0|^{\otimes N} \Phi^{\dagger}(\overrightarrow{x_j}) \Phi(\overrightarrow{x_i}) |0\rangle^{\otimes N}$$

This inner product is similar to $\langle 0|1\rangle = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0$, but with more complicated vectors!

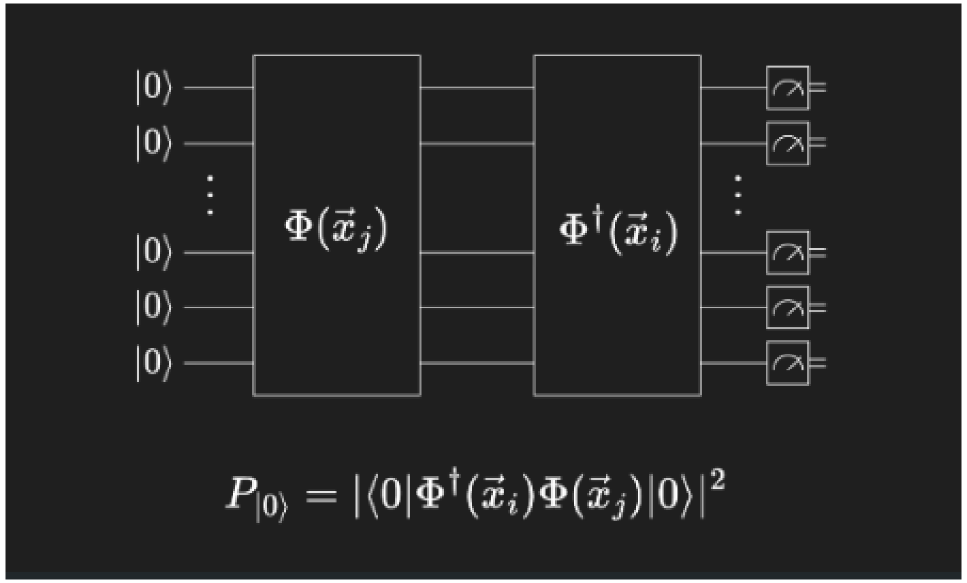
Building the kernel in quantum (some unsolicited math)

- Now, since we started the qubits with the all-zero state ($|0\rangle^{\otimes N}$), the probability of measuring the all-zero is:

$$P_0 = \left| \langle 0|^{\otimes N} \Phi^\dagger(\vec{x_j}) \Phi(\vec{x_i}) |0\rangle^{\otimes N} \right|^2$$

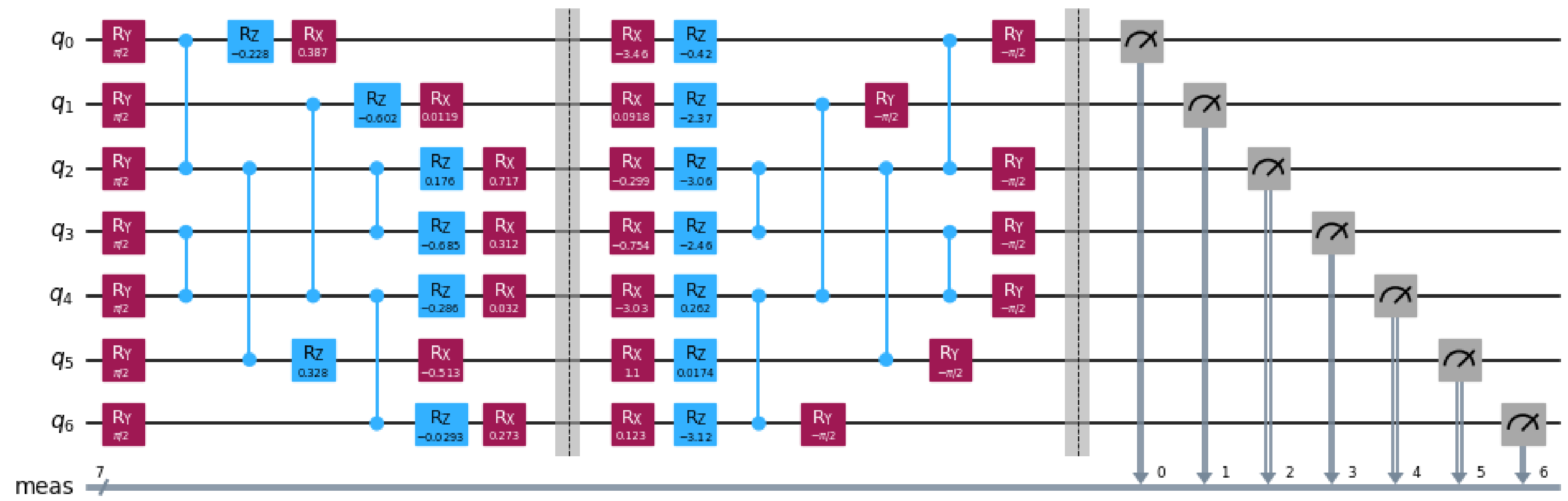
- This is exactly what we want to compute as the kernel entry in the kernel matrix (up to the norm square).

- Hence, our circuit will return measurement probabilities, and we want to get the probability of measuring the all-zero state!

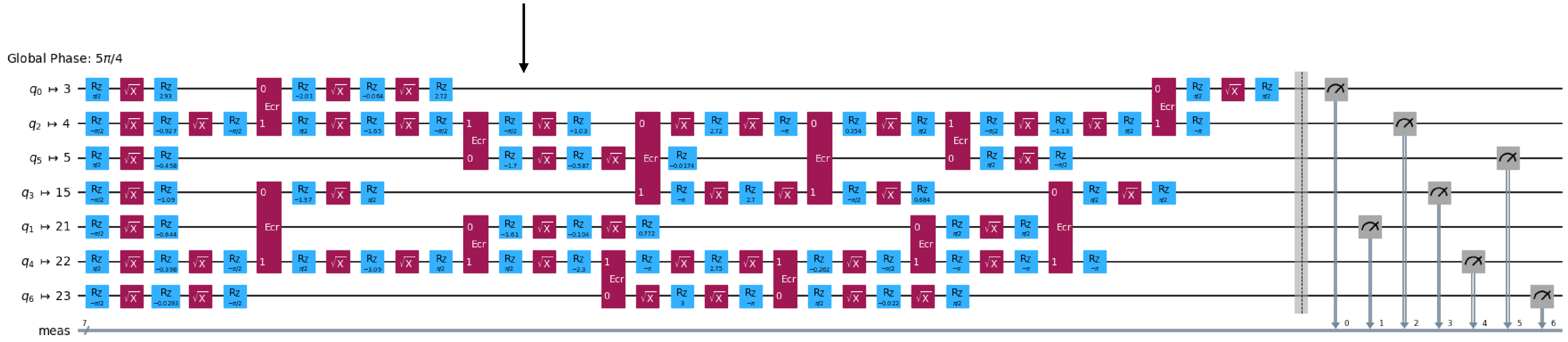

$|0\rangle$
$\Phi(\vec{x}_j)$
$\Phi^\dagger(\vec{x}_i)$
$P_{|0\rangle} = |\langle 0|\Phi^\dagger(\vec{x}_i)\Phi(\vec{x}_j)|0\rangle|^2$

After we build the kernel matrix, we can feed it into a classical classifier such as SVC!

An example circuit for the quantum kernel



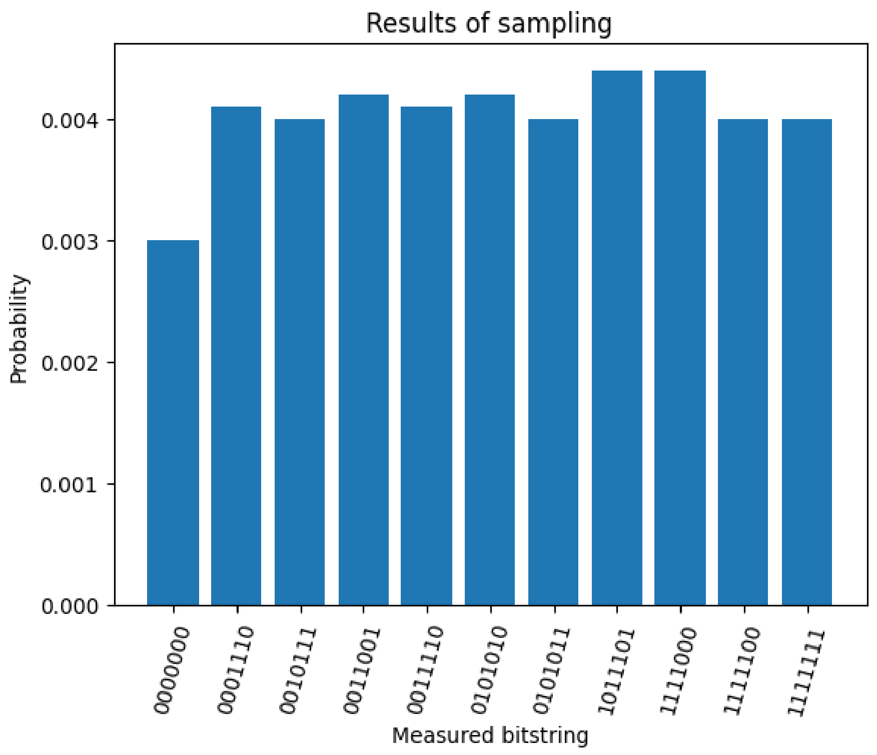
- The resulting UnitaryOverlap circuit looks like this. Now we are ready to execute this on a quantum system or a simulator for testing.
- We can choose a backend and transpile to circuit to map to the hardware!

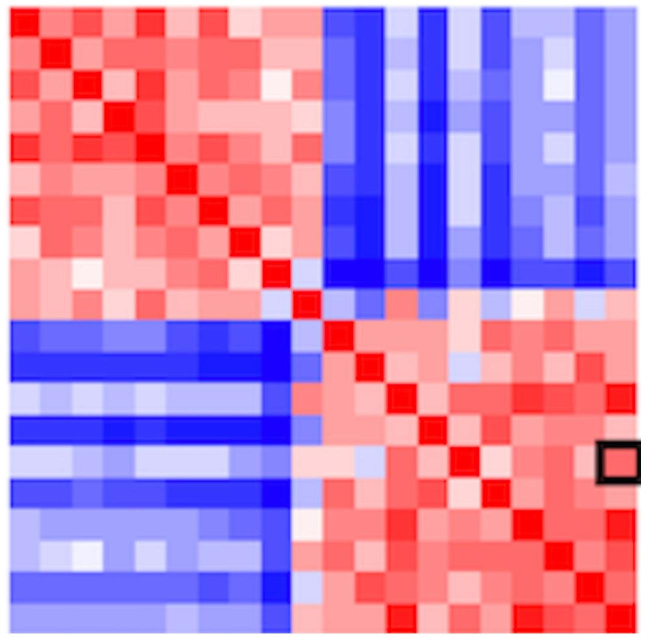


Notice that after the transpilation, the circuit changed. This is what is actually run on the quantum computer!

Working through an example

- Now using the Qiskit primitive, Sampler, we can run the circuit and obtain the probability distribution.
- Recall that we are interested in the probability of measuring the all-zero state.
- This value gives us the corresponding kernel matrix entry!



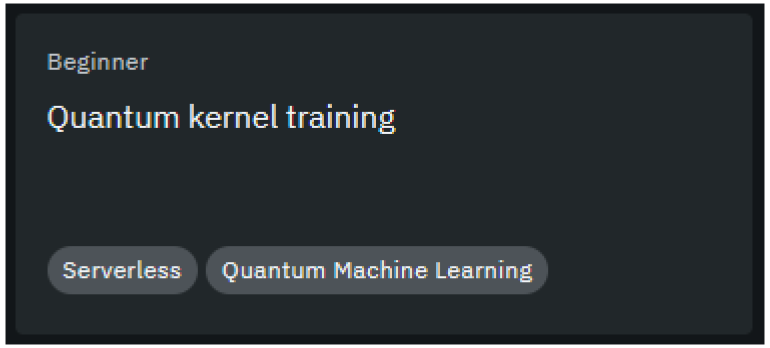


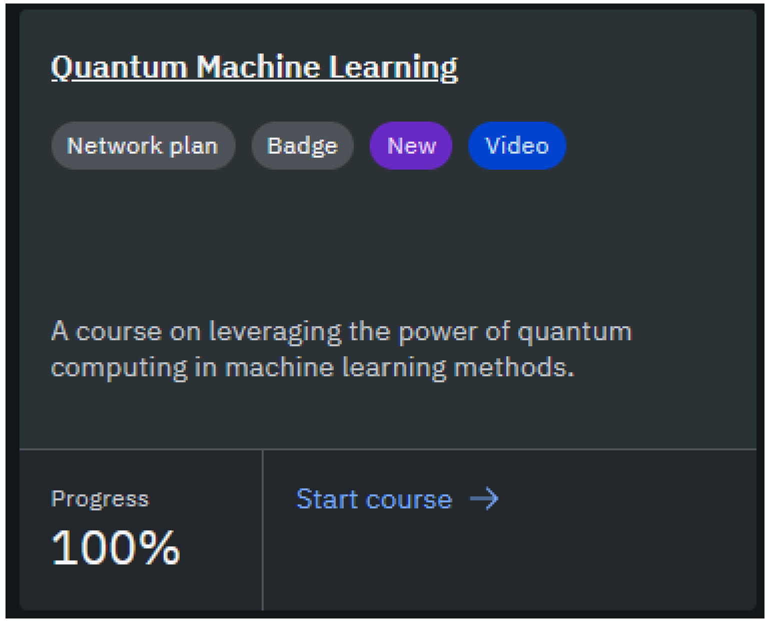
Kernel matrix, the entry for the samples used highlighted in the black box.

A summary

- We have seen the full workflow of Quantum Kernel Estimation method, which is a hybrid QML algorithm for binary classification tasks.
- A benefit of this approach is that it does not require any parameter training of a quantum circuit, so it is slightly different from the quantum variational approaches.
- A problem tailored feature map design is crucial as the problem gets bigger. It is good practice to design the feature map circuit, transpile it and see the overall complexity (circuit depth) before running the algorithm on a quantum computer (we usually want depth<100 even before transpilation).
- Moreover, since we need to compute every entry in the matrix for the full kernel matrix, the number of circuit executions increase with the number of samples. For most ML problem, you need to compute both training and test kernel matrices.

Resources from IBM Quantum Learning Platform





A course on leveraging the power of quantum computing in machine learning methods.

Link: https://learning.quantum.ibm.com/catalog/courses