# Qiskit for Quantum Machine Learning: Gates, Functions, and Tools

Grok

August 4, 2025

**Abstract**

This document provides a comprehensive overview of Qiskits quantum gates, functions, and tools, with a focus on their application in quantum machine learning (QML), particularly for a Quantum Support Vector Machine (QSVM) project classifying a binary Iris dataset. It includes a detailed table of all standard Qiskit gates, along with key functions and tools used in QML.

## 1 Quantum Gates in Qiskit

Quantum gates are the building blocks of quantum circuits in Qiskit, manipulating qubits for computations. The following table lists all standard gates available in Qiskits `qiskit.circuit` module, with brief explanations and their relevance to QML, such as in the `ZZFeatureMap` used for QSVM.

Table 1: Qiskit Quantum Gates

| Gate | Purpose | QML Use | Code Example |
|---|---|---|---|
| **Hadamard (H)** | Creates superposition: $\|0\rangle \rightarrow (\|0\rangle + \|1\rangle)/\sqrt{2}$, $\|1\rangle \rightarrow (\|0\rangle - \|1\rangle)/\sqrt{2}$. | Initializes qubits in feature maps (e.g., `ZZFeatureMap`). | `circuit.h(qr[0])` |
| **Pauli X** | Bit flip: $\|0\rangle \leftrightarrow \|1\rangle$. | Manipulates qubit states in feature maps or variational circuits. | `circuit.x(qr[0])` |
| **Pauli Y** | Bit and phase flip: $\|0\rangle \rightarrow i\|1\rangle$, $\|1\rangle \rightarrow -i\|0\rangle$. | Used in variational circuits for QML. | `circuit.y(qr[0])` |
| **Pauli Z** | Phase flip: $\|1\rangle \rightarrow -\|1\rangle$. | Common in phase-based feature maps. | `circuit.z(qr[0])` |
| **RX**($\theta$) | Rotates around X-axis by angle $\theta$. | Parameterized rotations in feature maps or VQC. | `circuit.rx(theta, qr[0])` |
| **RY**($\theta$) | Rotates around Y-axis by angle $\theta$. | Used in variational circuits or feature maps. | `circuit.ry(theta, qr[0])` |

Table 1: Qiskit Quantum Gates (continued)

| Gate | Purpose | QML Use | Code Example |
|---|---|---|---|
| **RZ($\theta$)** | Rotates around Z-axis by angle $\theta$. | Encodes data in `ZZFeatureMap` for QSVM. | `circuit.rz(theta, qr[0])` |
| **Phase (P)** | Applies phase $e^{i\theta}$ to $\|1\rangle$. | Introduces phase shifts in feature maps. | `circuit.p(theta, qr[0])` |
| **S** | Applies $\pi/2$ phase: $\|1\rangle \to i\|1\rangle$ (square root of Z). | Precise phase manipulation in complex circuits. | `circuit.s(qr[0])` |
| **S$^\dagger$** | Adjoint of S: $\|1\rangle \to -i\|1\rangle$. | Rare, used for phase adjustments. | `circuit.sdg(qr[0])` |
| **T** | Applies $\pi/4$ phase: $\|1\rangle \to e^{i\pi/4}\|1\rangle$. | Fine-grained phase control in circuits. | `circuit.t(qr[0])` |
| **T$^\dagger$** | Adjoint of T: $\|1\rangle \to e^{-i\pi/4}\|1\rangle$. | Rare, used in precise phase adjustments. | `circuit.tdg(qr[0])` |
| **U1($\theta$)** | Equivalent to P gate: phase rotation by $\theta$. | Legacy gate, used in older QML circuits. | `circuit.u1(theta, qr[0])` |
| **U2($\phi$, $\lambda$)** | General single-qubit gate with two parameters. | Rarely used in QML due to complexity. | `circuit.u2(phi, lambda, qr[0])` |
| **U3($\theta$, $\phi$, $\lambda$)** | General single-qubit gate with three parameters. | Flexible but rarely used in QML due to simpler alternatives. | `circuit.u3(theta, phi, lambda, qr[0])` |
| **CNOT (CX)** | Flips target qubit if control is $\|1\rangle$. | Creates entanglement in `ZZFeatureMap` for QSVM. | `circuit.cx(qr[0], qr[1])` |
| **CZ** | Applies Z to target if control is $\|1\rangle$. | Entangling gate in quantum kernels. | `circuit.cz(qr[0], qr[1])` |
| **CY** | Applies Y to target if control is $\|1\rangle$. | Rare, used in specific variational circuits. | `circuit.cy(qr[0], qr[1])` |
| **CPhase($\theta$)** | Controlled phase rotation by $\theta$. | Phase manipulation in complex QML circuits. | `circuit.cp(theta, qr[0], qr[1])` |
| **CRX($\theta$)** | Controlled X-rotation by $\theta$. | Parameterized control in variational circuits. | `circuit.crx(theta, qr[0], qr[1])` |
| **CRY($\theta$)** | Controlled Y-rotation by $\theta$. | Used in variational circuits or custom feature maps. | `circuit.cry(theta, qr[0], qr[1])` |
| **CRZ($\theta$)** | Controlled Z-rotation by $\theta$. | Common in variational circuits for QML. | `circuit.crz(theta, qr[0], qr[1])` |
| **Swap** | Swaps states of two qubits. | Rearranges qubits in circuit design. | `circuit.swap(qr[0], qr[1])` |
| **CSwap (Fredkin)** | Swaps two target qubits if control is $\|1\rangle$. | Rare, used in complex circuit designs. | `circuit.cswap(qr[0], qr[1], qr[2])` |

Table 1: Qiskit Quantum Gates (continued)

| Gate | Purpose | QML Use | Code Example |
|---|---|---|---|
| **Toffoli (CCX)** | Flips target if both controls are $\|1\rangle$. | Used in complex multi-qubit circuits. | `circuit.ccx(qr[0], qr[1], qr[2])` |
| **Multi-Controlled X (MCX)** | Generalizes CNOT to multiple controls. | Rare in QML, used in advanced algorithms. | `circuit.append(MCXGate(3), [qr[0], qr[1], qr[2], qr[3]])` |
| **Unitary** | Applies custom unitary matrix to qubits. | Specialized operations in advanced QML circuits. | `circuit.unitary(matrix, [qr[0]])` |

## 2 Key Qiskit Functions

Functions in Qiskit facilitate circuit construction, execution, and manipulation. Below are the most relevant functions for QML, particularly for the Iris QSVM project, implemented using `QuantumKernel` and `QSVC`.

- **`QuantumCircuit(qr, cr)`**: Creates a circuit with quantum (`qr`) and classical (`cr`) registers. Used to build `ZZFeatureMap` for QSVM. *Example*: `qc = QuantumCircuit(2, 2)`.
- **`circuit.append(gate, qubits)`**: Adds a gate to the circuit. Used to construct custom feature maps. *Example*: `qc.append(HGate(), [0])`.
- **`circuit.measure(qubit, cbit)`**: Measures a qubit into a classical bit. Less common in QSVM (kernel-based) but used in VQC. *Example*: `qc.measure(0, 0)`.
- **`circuit.bind_parameters(params)`**: Assigns values to parameterized gates. Critical for encoding Iris data in `ZZFeatureMap`. *Example*: `qc.bind_parameters(data)`.
- **`execute(circuit, backend, shots)`**: Runs a circuit on a backend. Used implicitly in `QuantumKernel` for QSVM. *Example*: `job = execute(qc, backend, shots=1024)`.
- **`backend.run(circuit)`**: Modern execution method for circuits. Used with IBM Quantum hardware. *Example*: `job = backend.run(qc)`.
- **`circuit.compose(other)`**: Combines circuits. Useful for stacking feature maps. *Example*: `qc.compose(feature_map)`.
- **`circuit.to_gate()`**: Converts a circuit to a reusable gate. Simplifies complex feature maps. *Example*: `gate = qc.to_gate()`.
- **`transpile(circuit, backend)`**: Optimizes circuit for a backend. Ensures efficient QSVM execution. *Example*: `tc = transpile(qc, backend)`.

## 3 Qiskit Tools and Modules

Qiskits ecosystem includes modules and tools for quantum computing and QML. Below are the key components, with emphasis on their use in the Iris QSVM project.

- **Qiskit Terra**: Core library for circuit construction and visualization. Used to build `ZZFeatureMap`. *Key Features*: `QuantumCircuit`, `circuit.draw('mpl')`.
- **Qiskit Aer**: High-performance simulators. `AerSimulator` computes kernel matrices in QSVM. *Example*: `backend = AerSimulator()`.
- **Qiskit Machine Learning**: Implements QML algorithms. `QuantumKernel` and `QSVC` are core to the Iris project. *Example*: `kernel = QuantumKernel(feature_map, quantum_instance=backend)`.
- **Qiskit Algorithms**: Provides utilities like `ZZFeatureMap` and variational forms for VQC. *Example*: `feature_map = ZZFeatureMap(2, reps=1)`.
- **Qiskit IBM Runtime**: Interfaces with IBM Quantum hardware. Optional for running QSVM on real devices. *Example*: `service = QiskitRuntimeService(token="YOUR_TOK`
- **Qiskit Visualization**: Visualizes circuits and results. Used to plot decision boundaries in QSVM. *Example*: `circuit.draw('mpl')`.
- **Qiskit Circuit Library**: Pre-built circuits like `ZZFeatureMap`, `ZFeatureMap`. Essential for QSVM kernels. *Example*: `feature_map = ZZFeatureMap(2)`.

## 4 Conclusion

This document summarizes Qiskits gates, functions, and tools, with a focus on their application in QML, particularly the QSVM for the Iris dataset. The gate table includes all standard Qiskit gates, while the listed functions and tools are the most relevant for QML tasks. For further details, refer to the Qiskit Documentation or explore community resources on X.