QUANTEN-COMPUTER AUF DER TITANIC

Katharina Simbeck | Shirin Riazy

Erstveröffentlichung in:
GRENZEN IN ZEITEN TECHNOLOGISCHER UND SOZIALER DISRUPTION
Beiträge und Positionen der HTW Berlin, Hg. Stefanie Molthagen-Schnöring,
BWV Berliner Wissenschafts-Verlag, **ISBN** 978-3-8305-3957-5.

ABSTRACT

Heute übliche elektronische Computer arbeiten digital, sie verwenden als Grundlage aller Operationen die binären Zustände 0 und 1. Quantencomputer versprechen einen technologischen Sprung mit völlig neuen Anwendungsbereichen. In diesem Artikel werden erste Schritte für den Einstieg in die Programmierung von Quantencomputern vorgestellt, bspw. für den Einsatz in der Lehre. Dafür verwenden wir den klassischen Titanic-Datensatz und erzeugen ein Quantencomputing-Modell zur Vorhersage der Überlebenswahrscheinlichkeit.

1. GRUNDLAGEN

1.1 Quantencomputing

Auch wenn die Möglichkeiten digitaler Computer heute sehr groß sind, gibt es hochkomplexe Berechnungen, die selbst mit leistungsfähigen Maschinen nicht in endlicher Zeit lösbar sind, wohl aber mit Quantencomputern, z. B. Primfaktorenzerlegung von sehr großen Zahlen oder die effiziente Berechnung diskreter Logarithmen. [1]

Die Prinzipien des Quantencomputing sind seit den 1980er Jahren [2, 3] bekannt, ihre technische Umsetzung ist jedoch mit großen Herausforderungen verbunden. Eine besondere Herausforderung ist die Störanfälligkeit von Quantencomputern, man versucht diese z. B. durch starke Kühlung und Abschirmung zu verbessern. [4] Mittlerweile sind erste Quantencomputer auch als Cloud Dienstleistung verfügbar, sie verfügen jedoch nur über wenige Qubits. [5]

[1] P. Shor, „Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer“, SIAM J. Comput., vol. 26, no. 5, S. 1484–1509, Oct. 1997.

[2] Y. Manin, „Computable and uncomputable“, Sov. Radio Mosc., vol. 128, 1980.

[3] R. P. Feynman, „Simulating physics with computers“, Int. J. Theor. Phys., vol. 21, no. 6, pp. 467–488, 1982.

[4] R. Das et al., „Cryogenic Qubit Integration for Quantum Computing“, in IEEE 68th Electronic Components and Technology Conference: ECTC 2018 : 29 May-1 June 2018, San Diego, California : proceedings, San Diego, CA, USA, S. 504–514, 2018.

[5] R. LaRose, „Overview and Comparison of Gate Level Quantum Software Platforms“, Quantum, vol. 3, S. 130, Mar. 2019.

VORHER		NACHHER	
Control-Qubit	Ziel-Qubit	Control-Qubit	Ziel-Qubit
|0⟩	|0⟩	|0⟩	|0⟩
|0⟩	|1⟩	|0⟩	|1⟩
|1⟩	|0⟩	|1⟩	|1⟩
|1⟩	|1⟩	|1⟩	|0⟩

Abbildung 1: Wahrheitstabelle des CNOT Gates.

Qubits werden in Quantencomputern an Stelle von herkömmlichen Bits verwendet. Anders als Bits, können sich Qubits nicht nur in den Zuständen 0 oder 1 befinden, sondern auch in Überlagerungen (Linearkombinationen) dieser Zustände, die auch als Superposition bezeichnet wird. Die zugrundeliegenden Zustände werden mit |0⟩ und |1⟩ bezeichnet. Bei Messung des Zustandes des Qubits ergibt sich dann das Ergebnis |0⟩ oder |1⟩ mit einer gewissen Wahrscheinlichkeit. Darüber hinaus haben Qubits eine weitere besondere Eigenschaft: Sie können miteinander verschränkt sein und nehmen dann immer den gleichen Zustand ein. **[6]**

Quantencomputer können über Gates gesteuert werden, die mit den in der digitalen Elektronik verwendeten AND, OR, XOR Gates vergleichbar sind. Bekannte Gates sind das Hadamard Gate (abgekürzt: H) und das CNOT Gate. Das Hadamard Gate erzeugt eine Quantenüberlagerung, sodass die Zustände |0⟩ und |1⟩ bei Messung gleich wahrscheinlich (equal superposition) sind. Das CNOT Gate operiert auf zwei Qubits (sog. Ziel- und Control-Qubits) und invertiert das Zielqubit, wenn das Control-Qubit 1 ist (Also (|1⟩, |0⟩) wird zu (|1⟩, |1⟩)). Das klassische Analogon zum CNOT Gate ist ein reversibles XOR Gate (**siehe Abbildung 1** für eine detaillierte Wahrheitstabelle des CNOT Gates).

Qubits können durch einen Einheitsvektor, der vom Mittelpunkt der sogenannten Bloch-Kugel zu ihrer Oberfläche zeigt, visualisiert werden. Die Bloch-Kugel hat die Besonderheit, dass die orthogonalen Vektoren |0⟩ und |1⟩ auf den gegenüberliegenden Polen angezeigt werden und damit auf einer Achse zu liegen scheinen.

1.2 Quanten Machine Learning

Support vector machines (SVM) zählen zu den verbreitetsten Methoden des maschinellen Lernens. Mittlerweile wurden vielfältige Varianten sogenannter „Quantum SVMs“ (QSVM) zur optimierten Berechnung von SVMs auf Quantencomputern publiziert **[7, 8, 9]**.

[6] J. Bub, Quantum Entanglement and Information. [Online] Available: https://plato.stanford.edu/archives/spr2019/entries/qt-entangle/.

[7] P. Rebentrost, M. Mohseni, and S. Lloyd, „Quantum support vector machine for big data classification“, Phys. Rev. Lett., vol. 113, no. 13, 130503, 2014.

[8] Z. Li, X. Liu, N. Xu, and J. Du, „Experimental realization of a quantum support vector machine“, Phys. Rev. Lett., vol. 114, no. 14, 140504, 2015.

[9] V. Havlicek et al., „Supervised learning with quantum enhanced feature spaces“, ArXiv180411326 Quant-Ph Stat, Apr. 2018.

[10] https://www.kaggle.com/c/titanic/data, abgerufen am 02.04.2019.

[11] G. Aleksandrowicz et al., „Qiskit: An Open-source Framework for Quantum Computing“, Zenodo, 23-Jan-2019

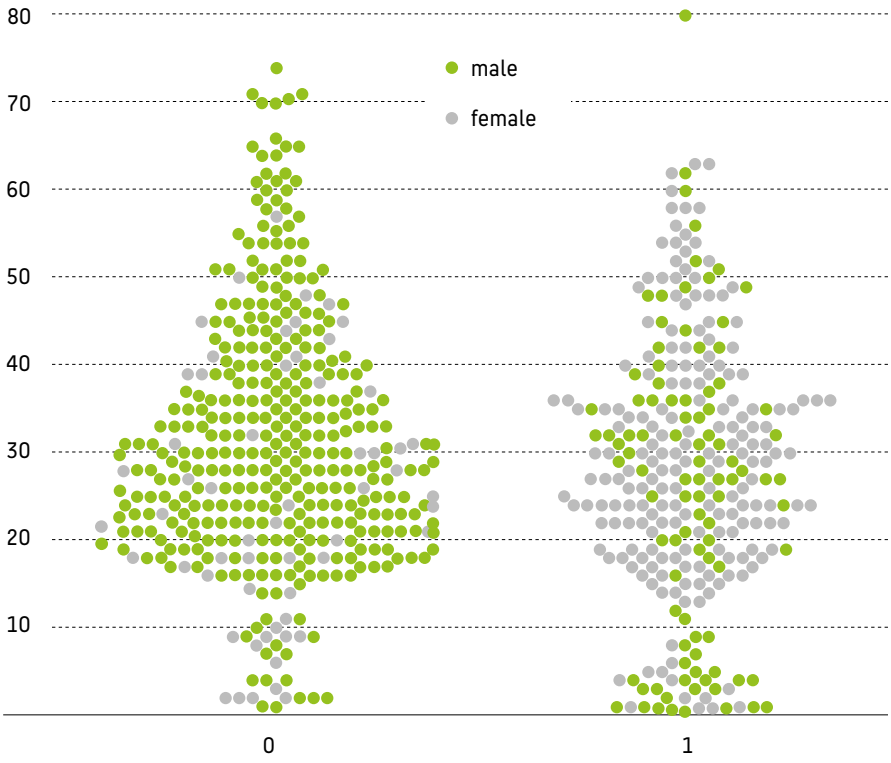

Abbildung 2: Darstellung der Altersverteilung der Passagiere der Titanic, aufgeschlüsselt nach Geschlecht und nach Überleben. Eigene Darstellung.

1.3 Titanic Datensatz

Der Titanic Datensatz [10] beschreibt echte Passagierdaten der RMS Titanic und wird weltweit in der Lehre von Statistik und Machine Learning eingesetzt. Der Datensatz enthält Daten zu 887 Passagieren der Titanic. Zu jedem Passagier sind unter anderem folgende Merkmale enthalten: Ob die Person das Unglück überlebte, das Geschlecht, Alter, Klasse und Preis des Tickets.

Im ersten Schritt der vorliegenden Auswertung wird jedes der Merkmale als Zahlenwert formuliert und standardisiert. Im Anschluss wurden 40 zufällig ausgewählte Datenpunkte zum Training und 20 Datenpunkte zum Testen des Algorithmus' genutzt. Wie man in **Abbildung 2** sieht, wurden „Frauen und Kinder zuerst" gerettet.

2. ANWENDUNG DES QUANTUM INFORMATION SCIENCE KIT

Qiskit [11], das Quantum Information Science Kit, ist ein open-source Framework zur Nutzung von IBMs Cloud Quantencomputing Service. Es beinhaltet mehrere Komponenten, darunter Aer, eine Komponente zur Simulation von Quantencomputing, die lokal auf Geräten des Nutzers arbeitet. Naturgemäß ist die Simulation von Quantenalgorithmen auf herkömmlichen Computern mit erheblichen Performanzeinbußen verbunden.

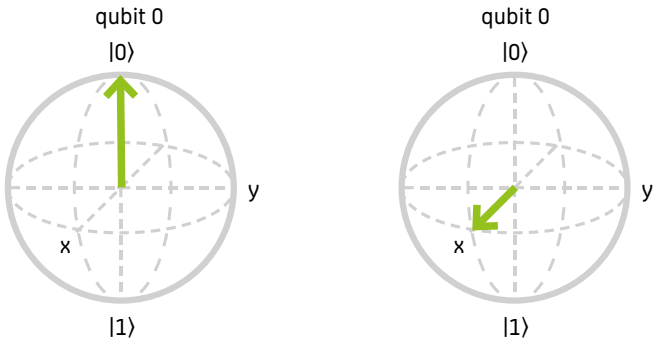

Abbildung 3: Bloch-Kugel des -Vektors (links) sowie die Superposition nach Anwendung des Hadamard Gates (rechts).

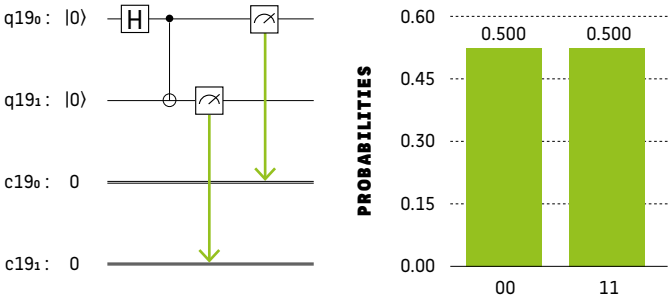

Abbildung 4: Quantenschaltung mit 2 Qubits und 2 Gates (Münzwurf) und Ergebnis.

Die Codebeispiele dieses Abschnittes orientieren sich am edX Onlinekurs „Quantum Machine Learning“ von Peter Wittek an der University of Toronto [12] und sind online verfügbar. [13] **Listing 1** zeigt eine exemplarische Nutzung von qiskit. Qiskit wurde in einer Anaconda Umgebung mit **pip install qiskit** installiert. Als Backend wird ein Simulator ausgewählt und ein Quantenregister mit einem einzigen Qubit angelegt. Per Definition werden Qubits in qiskit mit dem Zustand |0⟩ initialisiert. Wenn diese einfache Quantenschaltung (bestehend aus nur einem Qubit) 100-mal gemessen wird, ergibt sich 100-mal das Ergebnis |0⟩: **{‘0‘: 100}**.

Listing 1: Erstellung eines Quantenregisters mit nur einem Qubit
Das Qubit wird jetzt als Bloch-Kugel mit dem |0⟩-Vektor (Nordpol) dargestellt **[siehe Abbildung 3]**. Hierfür wird ein anderes Simulationsbackend benötigt (**statevector_simulator**).

Listing 2: Anzeigen der Bloch-Kugel für ein Qubit
Im nächsten Schritt soll eine einfache Schaltung mit 2 Qubits und 2 Gates implementiert werden **[siehe Listing 3]**. Zunächst sorgt ein Hadamard Gate am ersten Qubit dafür, dass dieses mit gleicher Wahrscheinlichkeit die Zustände |0⟩ und |1⟩ annimmt. Falls das erste Qubit |1⟩ ist, wird das CNOT Gate den Zustand des zweiten Qubits ebenfalls auf |1⟩ setzen, sonst bleibt es im Initialisierungszustand |0⟩. Die Schaltung wird 100 mal ausgeführt, das Ergebnis als Histogramm gespeichert **[siehe Abbildung 4]**.

Listing 3: Quantenschaltung mit 2 Qubits und 2 Gates (Münzwurf)
Da die Messung des Quantenzustandes ein zufälliges Ergebnis mit einer gewissen Wahrschein-

[12] https://courses.edx.org/courses/course-v1:University_of_TorontoX+UTQML101x+1T2019/, abgerufen am 7.4.2019, Login notwendig.

[13] https://iug.htw-berlin.de/?page_id=370

[14] R. LaRose, „Overview and Comparison of Gate Level Quantum Software Platforms“, Quantum, vol. 3, S. 130, Mar. 2019.

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute
from qiskit import BasicAer

backend = BasicAer.get_backend('qasm_simulator')

q = QuantumRegister(1)
c = ClassicalRegister(1)
circuit = QuantumCircuit(q, c)
circuit.measure(q, c)
job = execute(circuit, backend, shots=100)
result = job.result()
print(result.get_counts(circuit))
```

Listing 1: Erstellung eines Quantenregisters mit nur einem Qubit.

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute
from qiskit import BasicAer
from qiskit.tools.visualization import plot_bloch_multivector


backend_statevector = BasicAer.get_backend('statevector_simulator')
q = QuantumRegister(1)
c = ClassicalRegister(1)
circuit = QuantumCircuit(q, c)
circuit.measure(q, c)
job = execute(circuit, backend_statevector)
plot_bloch_multivector(job.result().get_statevector(circuit)).savefig("bloch1.pn
```

Listing 2: Anzeigen der Bloch-Kugel für ein Qubit.

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute
from qiskit import BasicAer
from qiskit.tools.visualization import circuit_drawer, plot_histogram

q = QuantumRegister(2)
c = ClassicalRegister(2)
circuit = QuantumCircuit(q, c)
circuit.h(q[0])
circuit.cx(q[0], q[1])
circuit_drawer(circuit).savefig("circuit.png")
circuit.measure(q, c)
circuit_drawer(circuit).savefig("circuit_measured.png")
backend = BasicAer.get_backend('qasm_simulator')
job = execute(circuit, backend, shots=100)
plot_histogram(job.result().get_counts(circuit)).savefig("hist.png")
```

Listing 3: Quantenschaltung mit 2 Qubits und 2 Gates (Münzwurf)

lichkeit ergibt, kann für die Simulation von Quantencomputern ein Zufallsgenerator mit Hilfe eines Startwertes (Seed) initialisiert werden. Wegen des Zufallsprozesses werden Quantencomputerprogramme in der Regel mehrfach durchlaufen, „shots“ bezeichnet die Anzahl der Durchläufe der simulierten Quantenschaltung. **[14]**

```
seed = 10598

feature_map = SecondOrderExpansion(num_qubits = feature_dim, depth = 2, entanglement = 'linear')
qsvm = QSVMKernel(feature_map, training_input, test_input, datapoints[0])

backend = Aer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots = 1024, seed = seed, seed_mapper = seed)

result = qsvm.rum(quantum_instance)
```

Listing 4: Ausgewählter Teil eines Quellcodes zur Implementierung von Quanten Machine Learning (in Anlehnung an ein Tutorial von IBM) **[16]**

METHODE (ANZAHL TRAININGS-/ TEST-DATENSÄTZE)	QSVM (40/20)	SVM (40/20)	SVM (700/100)
Dimension der Features = 6	0,725 (916,30 Sekunden)	0,75	0,85

Abbildung 5: Anteil korrekter Klassifikationen unterschiedlicher Algorithmen (in Klammern: Rechenzeit für lokal simulierten Quantencomputer).

Wenn der Code auf einem echten Quantencomputer von IBM in der Cloud laufen soll, wird ein API Token benötigt. Auf dem echten Quantencomputer zeigen sich dann auch die Unzulänglichkeiten der heutigen Technologie. Eine Gleichverteilung der Ergebnisse |00⟩ und |11⟩ wird nicht erreicht, die eigentlich unmöglichen Ergebnisse |01⟩ und |10⟩ treten ebenfalls auf.

3. QUANTEN MACHINE LEARNING

Glücklicherweise muss nicht jede Aufgabenstellung als abstrakte Schaltung mit Quantengattern implementiert werden. In der Bibliothek **qiskit_aqua** gibt es bereits Algorithmen für verschiedene Anwendungsbereiche, darunter auch eine Quantenimplementierung von Kernel-SVMs. **[15]** **Listing 4** zeigt, wie ein Modell der Überlebenswahrscheinlichkeit der Titanic-Passagiere mit einer Quanten-Kernel SVM erzeugt werden kann.

Listing 4: Ausgewählter Teil eines Quellcodes zur Implementierung von Quanten Machine Learning (in Anlehnung an ein Tutorial von IBM) **[16]**

Die Simulation des Codes mit 6 Features **[17]** und 40 Trainingsdatensätzen erreichte eine Genauigkeit von 72,5 % bei den 20 Testdatensätzen. **Abbildung 5** zeigt die Ergebnisse von QSVM und SVM im Vergleich. Insgesamt liefert die Implementierung des QSVM Algorithmus in der Simulation vergleichbare Ergebnisse zu einer klassischen Implementierung der SVM.

[15] https://qiskit.org/aqua

[16] https://github.com/Qiskit/qiskit-tutorials/blob/master/qiskit/aqua/artificial_intelligence/qsvm_kernel_classification.ipynb

[17] Die Anzahl der Features darf die Anzahl der Qubits nicht übersteigen.

4. RESULTATE UND AUSBLICK

Quantencomputer könnten in den nächsten Jahren in einzelnen Anwendungsgebieten Entwicklungssprünge erlauben. Bereits jetzt ist es möglich, die Grundlagen der Programmierung von Quantencomputern mit Hilfe von Python Bibliotheken in der Lehre einzuführen. Leider sind die Qiskit Bibliotheken noch unvollständig dokumentiert und häufigen Änderungen unterworfen. Auch ist heute noch nicht klar, welcher Technologieansatz und welche Anbieter sich durchsetzen werden.

Wir erwarten, dass erste Anwendungsgebiete in der Wirtschaftsinformatik Optimierungen von hochkomplexen Logistikprozessen sein könnten, die heute mit numerischen Algorithmen approximiert werden.