

# IMPLEMENTAZIONE DI UN ALGORITMO KNN MULTICLASSE SU HARDWARE QUANTISTICO

TESI DI LAUREA SPERIMENTALE IN FISICA

MARIANO MOLLO N85000880

RELATORI:

GIOVANNI ACAMPORA

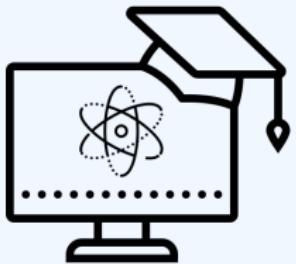
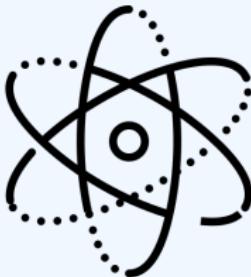
AUTILIA VITIELLO

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II  
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

25 OTTOBRE 2019



# QUANTUM MACHINE LEARNING



## Machine Learning

Permette ai computer di apprendere dai dati: riconoscimento facciale, guida autonoma, etc.

## Quantum Computing

Impiega la meccanica quantistica per eseguire calcoli più velocemente ed in maniera inaccessibile ai computer classici

## Quantum Machine Learning

Risolvere problemi difficili e sconosciuti impiegando la meccanica quantistica

# OBIETTIVO DI RICERCA

È possibile implementare su un computer quantistico un algoritmo k-nearest neighbours multiclasse, in modo da migliorare le prestazioni ed il numero di problemi risolvibili?

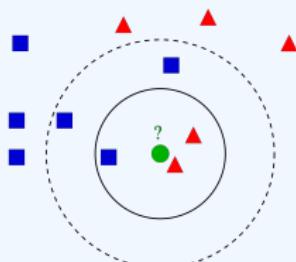
Passaggi:

- Riprodurre l'algoritmo di classificazione KNN quantistico a due classi proposto da Schuld et al. [4] e implementarne una versione multiclasse su processore quantistico dell'IBM
- Analizzare le capacità dell'algoritmo attraverso esecuzioni su hardware quantistico e tramite simulazione

# K-NEAREST NEIGHBOURS CLASSICO PESATO

L'algoritmo di classificazione KNN è uno tra i più semplici del ML ed è un lazy learner

$k$  è un numero naturale



Dato un insieme di vettori di training:

$$D = v_0, \dots, v_{M-1},$$

$$v_i \in \{\text{classe}_0, \text{classe}_1, \dots\}$$

Dato un nuovo vettore  $x$ :

- considera i  $k$  vettori di training più vicini ad  $x$
- classifica  $x$  con un voto a maggioranza

Si assegnano pesi dipendenti da  $\frac{1}{\text{distanza}}$  per aumentare l'influenza dei vettori più vicini

# ALGORITMO KNN QUANTISTICO

Stato quantistico iniziale:  $|a\rangle \otimes |i\rangle \otimes |c\rangle \otimes |m\rangle$

$$|\psi_0\rangle = \frac{1}{\sqrt{2M}} \sum_{m=1}^M (|0\rangle |\psi_x\rangle + |1\rangle |\psi_{t^m}\rangle) |c^m\rangle |m\rangle$$

Calcolo della distanza:

si fanno interferire gli stati applicando  $H|a\rangle$

$$|\psi_1\rangle = \frac{1}{2\sqrt{M}} \sum_{m=1}^M \left[ |0\rangle (|\psi_x\rangle + |\psi_{t^m}\rangle) + |1\rangle (|\psi_x\rangle - |\psi_{t^m}\rangle) \right] |c^m\rangle |m\rangle$$

Misura condizionale:  $|a\rangle = |0\rangle$

$$|\psi_2\rangle = \frac{1}{2\sqrt{M}} \sum_{m=1}^M \sum_{i=1}^N (x_i + t_i^m) |0\rangle |i\rangle |c^m\rangle |m\rangle$$

# ALGORITMO KNN QUANTISTICO

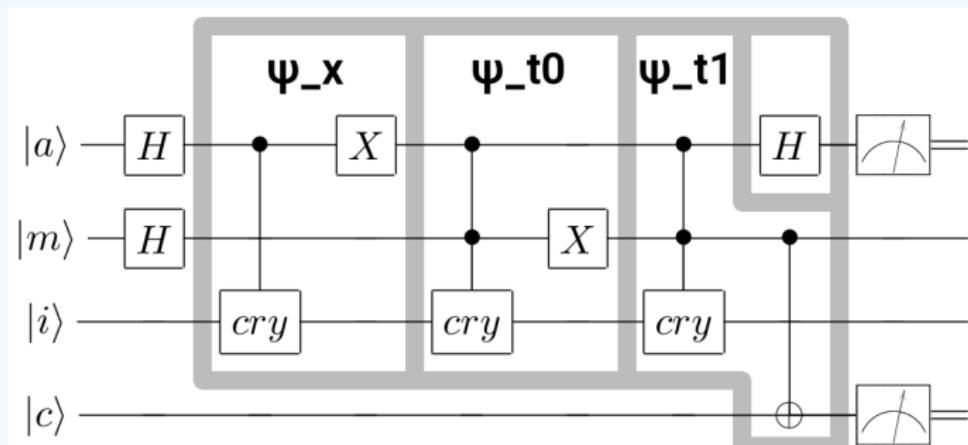
Se i vettori sono appropriatamente normalizzati la probabilità di misurare una data classe è:

$$P(|c^m\rangle = |s\rangle) = 1 - \sum_{m|c^m=s} \frac{1}{4M} |x - t^m|^2$$

Classificazione:

$$c = \begin{cases} 0 & \text{se } P(|c^0\rangle) \text{ maggiore} \\ 1 & \text{se } P(|c^1\rangle) \text{ maggiore} \\ \text{etc...} & \end{cases}$$

# ALGORITMO KNN QUANTISTICO

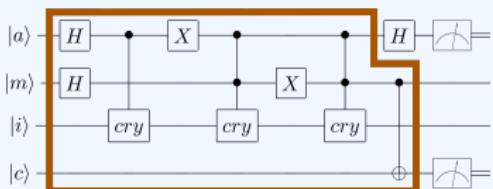


La complessità algoritmica del procedimento, ignorando la preparazione dello stato, è  $\mathcal{O}\left(\frac{1}{P(MC)}\right)$ , dove  $P(MC)$  è la probabilità di successo della misura condizionale (misura dell'ancilla nello stato  $|o\rangle$ ). Dipendendo questa solo dalla configurazione iniziale del sistema, si dice che l'algoritmo ha un tempo di esecuzione costante. [2]

# ESTENDERE L'IMPLEMENTAZIONE

L'implementazione appena vista lavora con:

- 2 vettori di training
- 2 dimensioni
- 2 classi



L'obiettivo è generalizzarne il funzionamento a:

- M vettori di training
- N dimensioni
- più di due classi

Per realizzare ciò si deve intervenire sulla procedura di preparazione dello stato iniziale.

## FF-QRAM

Per codificare dati classici nelle ampiezze di probabilità è stata usata la tecnica di costruzione di stati flip-flop QRAM proposta da Petruccione et al. [3] La FF-QRAM è usata per inizializzare efficientemente un QDB in maniera arbitraria.

L'operazione QRAM sui qubit sovrappone un insieme di dati classici  $D = \left\{ \left( \vec{d}^{(l)}, b_l \right) \mid 0 \leq l < M \right\}$  come

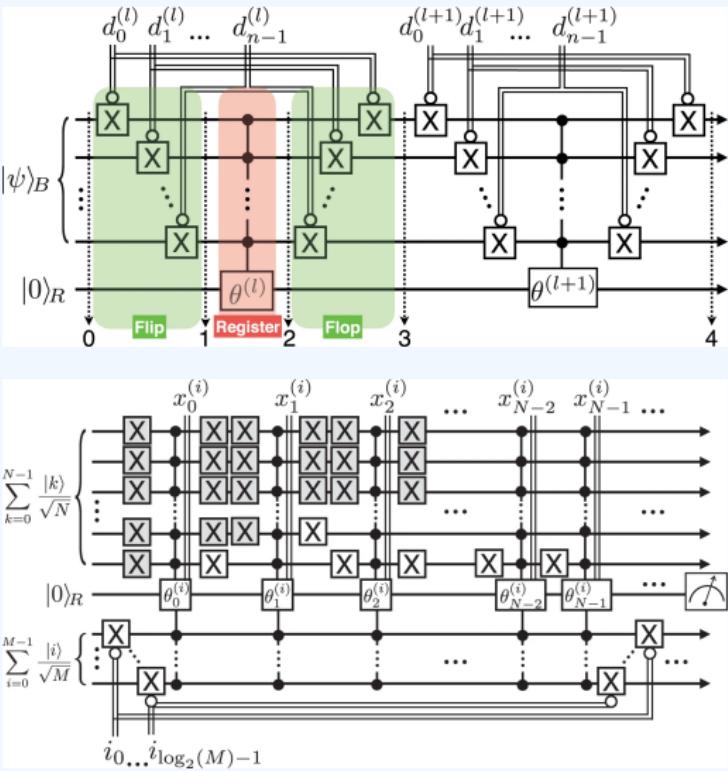
$$\text{QRAM}(D) \sum_j \psi_j |j\rangle_B |o\rangle_R \equiv \sum_l \psi_l |\vec{d}^{(l)}\rangle_B |b_l\rangle_R ,$$

dove  $\vec{d}^{(l)}$  rappresenta un indirizzo di memoria con  $n$  bit di informazione e  $b_l$  è l'attributo ad esso associato (per es. numero reale).

# CODIFICARE DATI CLASSICI NELLE AMPIEZZE

La complessità algoritmica di questa routine è  $\mathcal{O}(MN)$ , dove  $M$  è il numero di vettori e  $N$  la loro lunghezza. La quantità di risorse hardware necessarie va come  $\mathcal{O}(\log_2(MN))$ . [3]

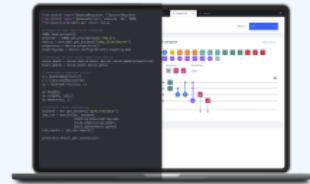
Per codificare più di due classi una procedura di tipo QRAM risulta conveniente.



## Qiskit

Struttura open source di sviluppo software [1] per

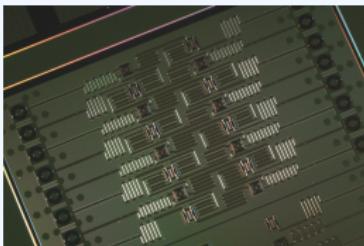
- progettare circuiti quantistici
- simularli sul proprio computer personale
- inviare ordini di esecuzione su hardware quantistico reale
- visualizzare i risultati



## IBM Q Experience

- accessibile al pubblico
- permette simulazioni ideali e con rumore
- fino a 14 qubit superconduttori
- fino a 32 qubit simulati

# STATO DELL'ARTE



L'ibmq\_16\_melbourne è il computer quantistico accessibile al pubblico a partire dal 2018 tramite interfaccia web. I tempi di decoerenza sono dell'ordine di  $T1 \approx 25 \div 88\mu s$ ,  $T2 \approx 15 \div 105\mu s$ .

L'IBM Q System One è il primo computer quantistico a circuiti commerciali al mondo, introdotto dall'IBM nel gennaio 2019. L'IBM Q System One possiede 20 qubit.

Il numero limitato di qubit ha posto vincoli sulla grandezza dei data set analizzabili.

# RISULTATI

## CLASSIFICAZIONE DI CLUSTER

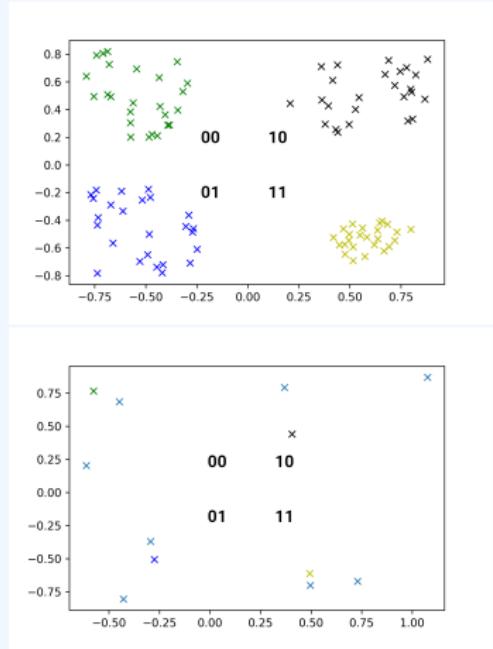
# CLASSIFICAZIONE DI CLUSTER

Per verificare il funzionamento base dell'algoritmo, è stato creato un insieme dati ad hoc.

(14 qubit)

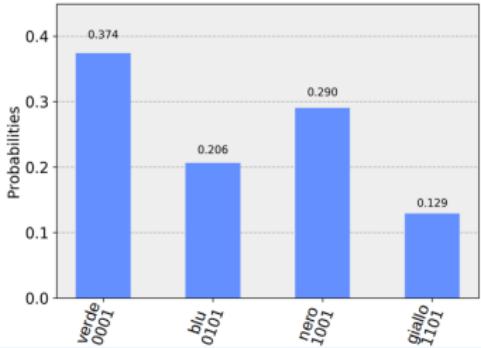


Sono stati selezionati otto vettori di training e si è classificato un vettore d'input per ciascuna classe.

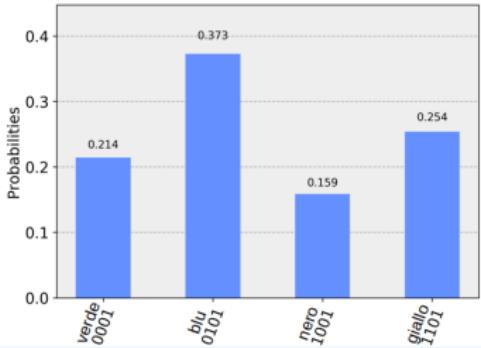


# CLASSIFICAZIONE DI CLUSTER (SIMULAZIONE)

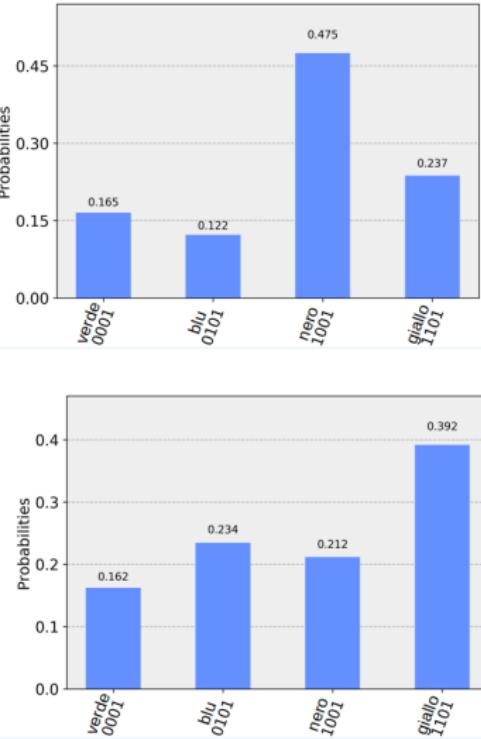
Classe verde



Classe blu



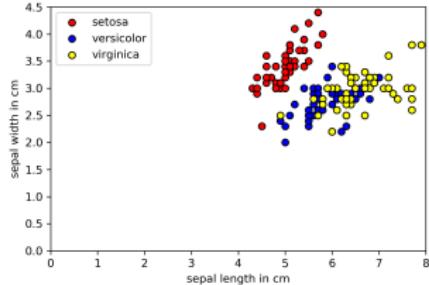
Classe giallo



# RISULTATI

**DATA SET IRIS**

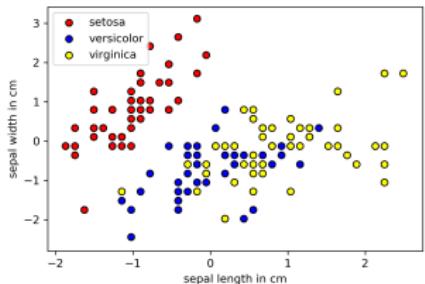
# DATA SET IRIS



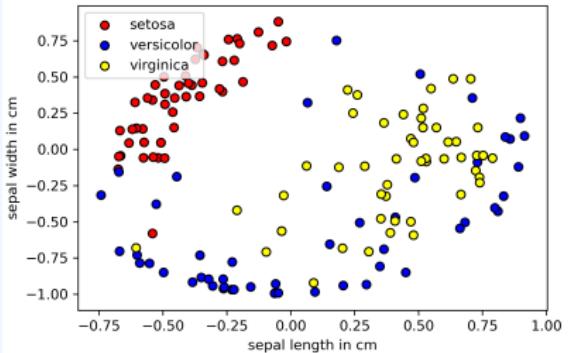
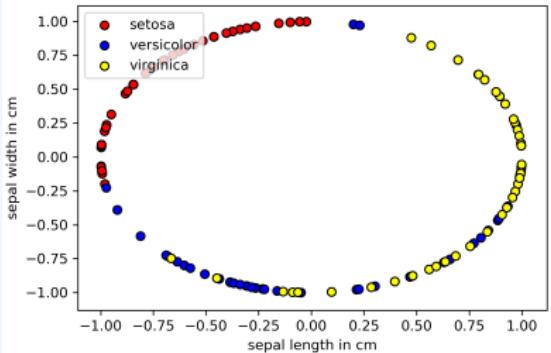
Per avere un confronto con un insieme conosciuto si sono analizzate le tre classi del noto data set Iris.



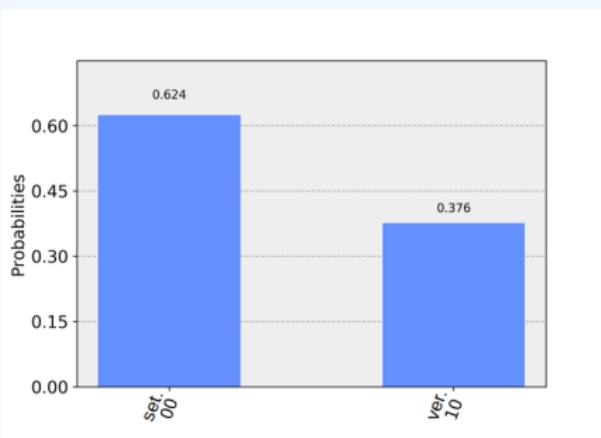
# PREPARAZIONE DEI DATI



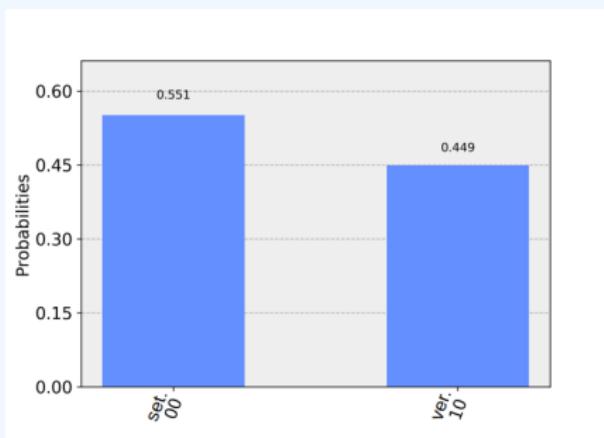
Si è centrato nell'origine e scalato i dati in modo che avessero varianza unitaria. Successivamente il data set è stato normalizzato per ragioni di compatibilità con l'algoritmo.



# CLASSIFICAZIONE A DUE CLASSI (SETOSA VS. VERSICOLOR)

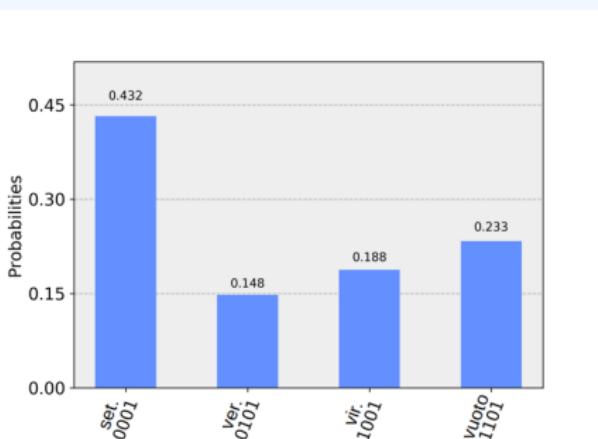


Simulazione su setosa

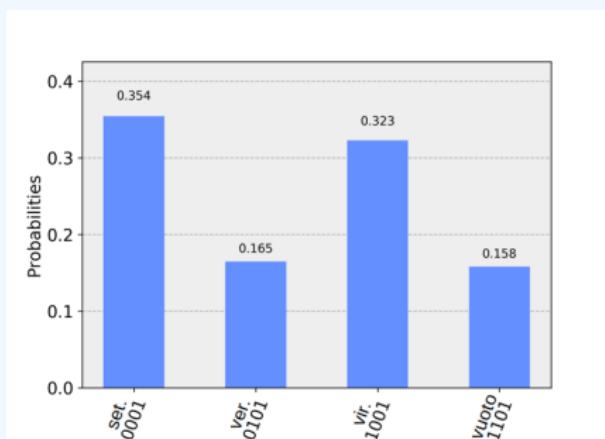


Esecuzione reale su setosa

# CLASSIFICAZIONE MULTICLASSE (SETOSA VS. VERSI- COLOR VS. VIRGINICA)



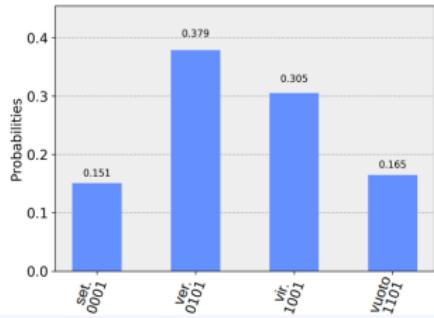
Simulazione su setosa



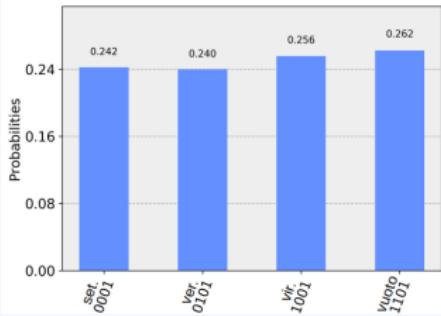
Esecuzione reale su setosa

# CLASSIFICAZIONE MULTICLASSE

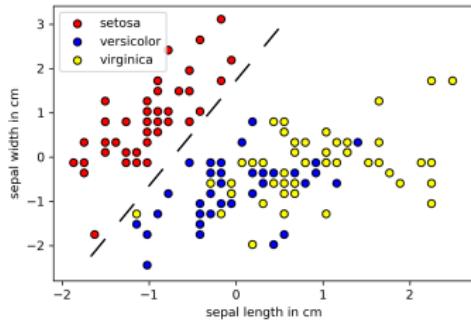
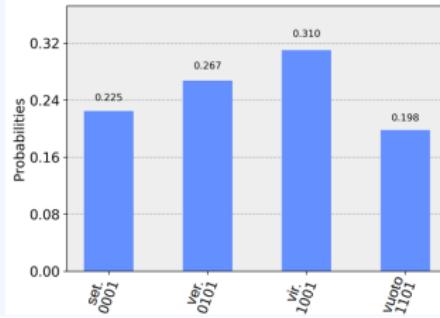
Sim. versicolor



Reale versicolor



Sim. virginica



# EFFICIENZA DI CLASSIFICAZIONE

classe	esiti positivi
verde	100%
blu	88,9%
nero	100%
giallo	100%

Simulazione su cluster  
con otto vettori di  
training

classe	$m = 3$	$m = 5$	$m = 7$
setosa	100%	100%	100%
versicolor	30%	50%	80%
virginica	60%	90%	90%

Risultati positivi per simulazione su  
Iris con  $2^m$  vettori di training

classe	esiti positivi
setosa	81,8%
versicolor	62,2%
virginica	59,4%

Risultati classici con pesi

# **CONCLUSIONE**

# RIASSUNTO

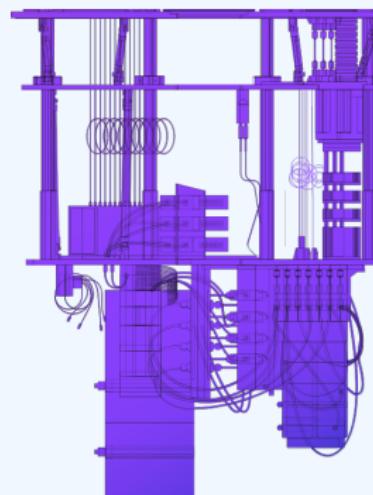
- L'elaborazione quantistica è nella frontiera dei supercomputer e ha il potenziale di accelerare gli algoritmi di machine learning classico
- È stata riprodotta un'implementazione di algoritmo KNN quantistico di classificazione binaria su hardware di piccola e media scala e se ne è esteso il funzionamento grazie alla procedura di costruzione di stati arbitrari FF-QRAM in modo da renderlo multiclass.
- La complessità algoritmica è stimata come  $\mathcal{O}(MNr)$ , usando  $\mathcal{O}(\log_2(MN))$  risorse hardware ( $r$  è il numero di run). L'algoritmo kNN classico non ottimizzato impiega  $\mathcal{O}(MNk)$  operazioni<sup>1</sup>, impiegando  $\mathcal{O}(MN)$  risorse di memoria.

---

<sup>1</sup>[http://www.cs.haifa.ac.il/~rita/ml\\_course/lectures/KNN.pdf](http://www.cs.haifa.ac.il/~rita/ml_course/lectures/KNN.pdf)

# PROSPETTIVE

- Far girare gli algoritmi su computer con maggiori risorse, sia in termini di numero di qubit che di tempi di decoerenza
- A tal proposito, sarebbe interessante l'esecuzione sul computer a 53 qubit annunciato quest'anno  
 $(m = 22 \implies 2^m \approx 4 \times 10^6)$
- Mitigare gli errori di esecuzione reale con i filtri di qiskit-ignis
- Si attende lo sviluppo di corrispettivi quantistici per algoritmi di IA più complessi



DOMANDE?

# FONTI

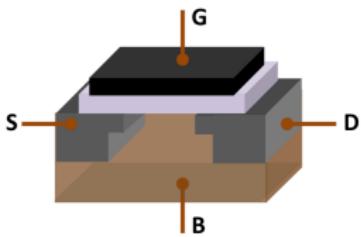
-  HÉCTOR ABRAHAM ET AL.  
**QISKIT: AN OPEN-SOURCE FRAMEWORK FOR QUANTUM COMPUTING**, 2019.
-  MARK FINGERHUTH.  
**QUANTUM-ENHANCED MACHINE LEARNING: IMPLEMENTING A QUANTUM K-NEAREST NEIGHBOUR ALGORITHM.**  
*Bachelor Thesis, Maastricht University*, 2017.
-  DANIEL K. PARK, FRANCESCO PETRUCCIONE, AND JUNE-KOO KEVIN RHEE.  
**CIRCUIT-BASED QUANTUM RANDOM ACCESS MEMORY FOR CLASSICAL DATA.**  
*Scientific Reports*, 9(3949), 2019.
-  M. SCHULD, M. FINGERHUTH, AND F. PETRUCCIONE.  
**IMPLEMENTING A DISTANCE-BASED CLASSIFIER WITH A QUANTUM INTERFERENCE CIRCUIT.**  
*EPL (Europhysics Letters)*, 119(6), 2017.

Riferimenti completi su: <https://github.com/visika/Tesi>

## IMMAGINI

- Visage Technologies Face Tracking and Analysis, by Abyssus
- Tesla Model 3 Headlights in Dever, Photo by Vlad Tchompalov on Unsplash
- An example of a diseased cassava leaf.  
<https://www.blog.google/technology/ai/ai-takes-root-helping-farmers-identify-diseased-plants/>

# **QUANTUM COMPUTING**

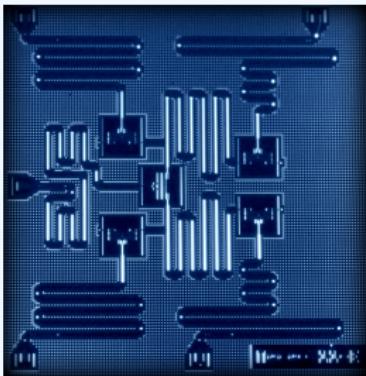


- Solitamente implementati attraverso MOSFET<sup>a</sup>
- 2 stati definiti: 0 e 1
- Può trovarsi in uno tra gli stati 0 o 1

---

<sup>a</sup>MOSFET: Metal Oxide Semiconductor Field Effect Transistor

# QUBIT



- Implementato in diversi modi: giunzioni superconduttrive, ioni intrappolati, fotoni polarizzati...
- Due stati definiti:  $|0\rangle$  e  $|1\rangle$
- Può trovarsi in una sovrapposizione degli stati  $|0\rangle$  e  $|1\rangle$

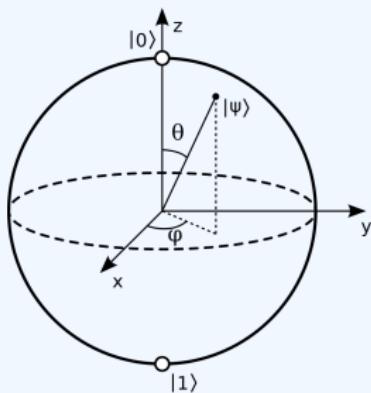
# QUBIT

Matematicamente, la sovrapposizione di un qubit è espressa come

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \quad \alpha, \beta \in \mathbb{C},$$

dove  $\alpha$  e  $\beta$  sono chimate ampiezze di probabilità.  
L'ultimo termine è chiamato vettore di probabilità.

# SFERA DI BLOCH



Un qubit si può visualizzare su una 2-sfera parametrizzando  $\alpha$  e  $\beta$  in coordinate polari

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle ,$$

dove  $0 \leq \theta \leq \pi$  e  $0 \leq \varphi < 2\pi$

## REGISTRO DI 2 QUBIT

Un computer quantistico con  $n$  qubit ha  $2^n$  ampiezze di probabilità.

Lo stato di un registro a più qubit è rappresentato dal prodotto tensoriale dello stato dei singoli qubit.

$$|00\rangle = |0\rangle \otimes |0\rangle$$

$$|\psi\rangle = c_0 |00\rangle + c_1 |01\rangle + c_2 |10\rangle + c_3 |11\rangle = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

## REGISTRO DI $n$ QUBIT

Un computer quantistico con  $n$  qubit ha  $2^n$  ampiezze di probabilità.

Lo stato di un registro a più qubit è rappresentato dal prodotto tensoriale dello stato dei singoli qubit.

$$|00\dots00\rangle = |0\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle \otimes |0\rangle$$

$$|\psi\rangle = c_0 |00\dots00\rangle + c_1 |00\dots01\rangle + \dots + c_{n-2} |11\dots10\rangle +$$

$$+ c_{n-1} |11\dots11\rangle = \begin{matrix} 00\dots00 \\ 00\dots01 \\ \vdots \\ 11\dots10 \\ 11\dots11 \end{matrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix}$$

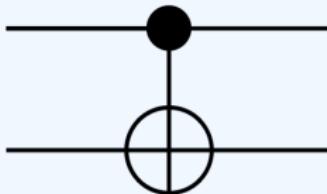
# PORTE LOGICHE QUANTISTICHE

Per manipolare  $n$  qubit esistono apposite porte logiche quantistiche, che sono operatori unitari rappresentabili come matrici  $2^n \times 2^n$ .

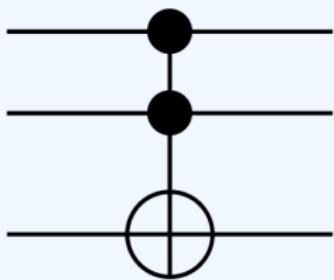
Hadamard



CNOT = CX



Toffoli



$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} \mathbb{I}_6 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

# PUNTI DI FORZA DEI QUANTUM COMPUTER

# di qubit	RAM classica richiesta
5	256 byte
25	2 gigabyte
50	8000 terabyte
275	numero di atomi nell'universo osservabile

Le  $n$  ampiezze di probabilità possono essere usate per memorizzare quantità enormi di informazioni. Possiamo inoltre creare e lavorare su più copie in parallelo degli stessi dati.

# ALGORITMO PER CLASSIFICATORE BINARIO

## Inizializzazione dei registri quantistici

```
a = QuantumRegister(1, 'a')
m = QuantumRegister(1, 'm')
i = QuantumRegister(1, 'i')
c = QuantumRegister(1, 'c')
b = ClassicalRegister(2, 'bit')
circuit = QuantumCircuit(a, m, i, c, b)
```

## Sovrapposizione degli stati

```
circuit.h(a)
circuit.h(m)
```

## Codifica del vettore d'input

```
circuit.cry(xo, a[o], i[o])
circuit.x(a) # entanglement dell'ancilla con o
```

# ALGORITMO PER CLASSIFICATORE BINARIO

## Codifica dei vettori di training

```
circuit.mcry(to, a[:] + m[:], i[o], None)
circuit.x(m) # entanglement di m con o
circuit.mcry(t1, a[:] + m[:], i[o], None)
circuit.cx(m, c) # entanglement della classe 1 con m 1
```

## Interferenza degli stati

```
circuit.h(a)
```

## Operazione di misura

```
circuit.measure(a, b[o])
circuit.measure(c, b[1])
```

```
# circuit.draw(output='mpl')
```

# ALGORITMO PER CLASSIFICATORE MULTICLASSE

## Inizializzazione dei registri quantistici

```
a = QuantumRegister(1, 'a') # knn ancilla  
m = QuantumRegister(2, 'm') # training vector index  
i = QuantumRegister(2, 'i') # feature index  
r = QuantumRegister(1, 'r') # rotation qubit  
q = QuantumRegister(5, 'q') # qram ancilla  
c = QuantumRegister(2, 'c') # class  
b = ClassicalRegister(4, 'bit')  
circuit = QuantumCircuit(a, m, i, r, q, c, b)
```

## Sovrapposizione degli stati

```
circuit.h(a)  
circuit.h(m)  
circuit.h(i)  
circuit.h(c)
```

# ALGORITMO PER CLASSIFICATORE MULTICLASSE

## Codifica dei vettori

```
# circuit.cry(theta, control, target)
# circuit.mcry(theta, controls, target, ancillae)
# » Encode the input vector »
encodeVector(circuit, inputVirginica, i, a[:] + i[:, r[o], q])
circuit.x(a) # entanglement dell'ancilla con o
# » Encode the training vectors »
buildTrainingState(trainingArray)
```

## Interferenza e misura

```
circuit.measure(r, b[o])
circuit.h(a)
circuit.measure(a, b[1])
circuit.measure(c[o], b[2])
circuit.measure(c[1], b[3])
```

# ALGORITMO PER CLASSIFICATORE MULTICLASSE

## Definizione dei costruttori

```
def encodeTraining(circuit, data, i, controls, rotationQ, ancillaQ,  
c, m):  
    # Header  
    encodeClass(circuit, c)  
    encodeIndex(circuit, m)  
    # Encoder  
    encodeVector(circuit, data, i, controls, rotationQ, ancillaQ)  
    # Footer  
    encodeClass(circuit, c)  
    encodeIndex(circuit, m)
```

# ALGORITMO PER CLASSIFICATORE MULTICLASSE

## Definizione del codificatore

```
def encodeVector(circ, data, i, controls, rotationQ, ancillaQ):
# |00>
    circuit.x(i)
    circuit.mcry(data[0], controls, rotationQ, ancillaQ)
    circuit.x(i)
# |01>
    circuit.x(i[1])
    circuit.mcry(data[1], controls, rotationQ, ancillaQ)
    circuit.x(i[1])
# |10>
    circuit.x(i[0])
    circuit.mcry(data[2], controls, rotationQ, ancillaQ)
    circuit.x(i[0])
# |11>
    circuit.mcry(data[3], controls, rotationQ, ancillaQ)
```

# ROUTINE FF-QRAM

$$|\psi_0\rangle_l = \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle |\text{o}\rangle_R + \sum_{j \neq \vec{d}^{(l)}} \psi_j |j\rangle |\text{o}\rangle_R$$

$$|\psi_1\rangle_l = \psi_{\vec{d}^{(l)}} |\mathbf{1}\rangle^{\otimes n} |\text{o}\rangle_R + \sum_{\overline{|j \oplus \vec{d}^{(l)}}} \psi_j \overline{|j \oplus \vec{d}^{(l)}} |\text{o}\rangle_R$$

$$|\psi_2\rangle_l = \psi_{\vec{d}^{(l)}} |\mathbf{1}\rangle^{\otimes n} |\theta^{(l)}\rangle_R + \sum_{\overline{|j \oplus \vec{d}^{(l)}}} \psi_j \overline{|j \oplus \vec{d}^{(l)}} |\text{o}\rangle_R$$

$$|\psi_3\rangle_l = \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle |\theta\rangle_R + \sum_{j \neq \vec{d}^{(l)}} \psi_j |j\rangle |\text{o}\rangle_R$$

$$|\psi_4\rangle_{l,l+1} = \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle |\theta^{(l)}\rangle_R + \psi_{\vec{d}^{(l+1)}} |\vec{d}^{(l+1)}\rangle |\theta^{(l+1)}\rangle_R + \sum_{j \neq \vec{d}^{(l)}, \vec{d}^{(l+1)}} \psi_j |j\rangle |\text{o}\rangle_R$$

# ROUTINE FF-QRAM

$$\sum_{l=0}^{M-1} \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle \left[ \cos \theta^{(l)} |\text{o}\rangle_R + \sin \theta^{(l)} |\text{1}\rangle_R \right] + \sum_{j \notin \{\vec{d}^{(l)}\}} \psi_j |j\rangle |\text{o}\rangle_R$$

$$P(1) = \sum_{l=0}^{M-1} |\psi_{\vec{d}^{(l)}} \sin \theta^{(l)}|^2$$

# QRAM PER QSVM

