MARIANO MOLLO

IMPLEMENTAZIONE DI UN ALGORITMO KNN MULTICLASSE SU HARDWARE QUANTISTICO

[10 settembre 2019 at 9:49 – classicthesis v4.6]

Università degli Studi di Napoli "Federico II"

Scuola Politecnica e delle Scienze di Base

Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Triennale in Fisica

IMPLEMENTAZIONE DI UN ALGORITMO KNN MULTICLASSE SU HARDWARE QUANTISTICO

Relatori:
Giovanni Acampora
Autilia Vitiello

Candidato:
Mariano Mollo
Matr. N85000880

Anno Accademico 2019/2020

[10 settembre 2019 at 9:49 – classicthesis v4.6]

SOMMARIO

Il campo del quantum machine learning è tra gli ambiti di ricerca più promettenti per quanto riguarda il miglioramento dell'analisi dei big data. I problemi predominanti con cui devono confrontarsi i computer classici sono i limiti di memoria e la capacità di elaborazione in tempi appropriati. Sfruttando le proprietà quantomeccaniche della materia, i computer quantistici possono veicolare nuove metodologie d'analisi dati. A tal fine si esplorano le possibilità offerte dall'implementazione di un algoritmo KNN, attraverso il kit di sviluppo software Qiskit e la piattaforma online IBM Q Experience.

ABSTRACT

Quantum Machine Learning is among the most anticipated fields when dealing with improvements in big data analysis. The main issues classical computers have to confront with are limits in memory and enormous computation times. Making use of quantum mechanical properties of matter, quantum computers can spur the development of new methods to improve big data analysis. With that in mind, this thesis explores the capabilities of an implementation of a KNN algorithm, using the Qiskit software development kit and the online platform IBM Q Experience.

RINGRAZIAMENTI

Ringrazio la mia famiglia per avermi offerto l'opportunità di dedicarmi allo studio per tanto tempo, senza dovermi preoccupare di tante contingenze della vita, garantendo amore incondizionato e supporto economico e psicologico.
Ringrazio i miei amici per avermi offerto occasioni di svago e sostegno durante questi lunghi periodi di studio.
Ringrazio il mio relatore prof. Giovanni Acampora, per avermi dato preziose risorse per affrontare questo lavoro di tesi.
Vorrei inoltre ringraziare Mark Fingerhuth per aver offerto i proprio consigli quando necessario, che hanno permesso di meglio comprendere i concetti e indirizzare gli sforzi.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

ELENCO DEGLI ALGORITMI

ACRONIMI

ML	machine learning
CQ	computer quantistico
QC	quantum computer
MLQ	machine learning quantistico
QML	quantum machine learning

QRAM quantum random access memory

KNN k-nearest neighbours

FF-QRAM flip-flop QRAM

QDB quantum database

Parte I

TEORIA

Introduzione al campo del quantum computing e del quantum machine learning, presentazione della teoria di base e degli strumenti usati.

1 INTRODUZIONE

Chiunque abbia un cellulare di nuova generazione è venuto in contatto con il campo del machine learning, la disciplina che si pone l'obiettivo di rendere i computer capaci di apprendere dall'esperienza. Gli esseri umani sono strettamente legati al processo di apprendimento, a partire dalla prima infanzia ed in maniera più o meno evidente durante tutto il corso vitale. Sebbene tale processo sembri automatico, in realtà è espressione di un complesso sistema di feedback, che in fin dei conti è codificato dal nostro codice genetico. Il machine learning (ML) (tradotto *apprendimento automatico*) prevede la scrittura del codice di programmazione che permetta ai computer di effettuare qualcosa di analogo all'apprendimento. Tali algoritmi sono il motore dei moderni assistenti vocali e dei sistemi che ci suggeriscono nuovi prodotti da acquistare o video da guardare, ma anche nel prevenire le frodi su carta di credito, per filtrare lo spam dalle nostre caselle email, nell'individuare e diagnosticare malattie, e tanto ancora [8].

Un esempio recente di uso del ML è la funzione di gestione adattiva della batteria, lanciata in Android 9, che usa il machine learning per predire quali applicazioni l'utente userà nelle prossime ore e quali no, in modo da dedicare la carica solo per le applicazioni più usate dal proprietario. [1]

Stando all'IBM [3], ogni giorno vengono creati circa $2,5 \times 10^{18}$ byte di dati: questo numero in costante crescita mette chi ha a che fare con i dati di fronte alla necessità di dotarsi di algoritmi avanzati che possando fare ordine in questo oceano di informazioni. Per trovare modelli e correlazioni su insiemi molto grandi è necessario un numero di operazioni che richiede molto tempo ed energia per essere portate a termine; trovare metodi efficienti per completare questi compiti diventa necessario sotto molti punti di vista.

Diversi lavori hanno evidenziato il ruolo nodale rispetto a questo problema che può avere l'uso della computazione quantistica, oggetto di intensa ricerca teorica da vari decenni e recentemente di sperimentazione da parte di studenti [2] e ricercatori [9].

Un computer quantistico (CQ) (o quantum computer (QC)) usa le proprietà peculiari dei sistemi quantomeccanici per manipolare ed elaborare l'informazione in modi inaccessibili ai normali computer classici. Così come un computer classico manipola bit, un computer quantistico fa uso dei cosiddetti bit quantistici (qubit). I bit e i qubit sono entrambi entità binarie, il che significa che possono assumere solo i valori 0 o 1. Ad ogni modo, un bit classico non probabilistico[1] può assumere solo uno di questi valori alla volta, laddove un qubit può trovarsi in una sovrapposizione lineare dei due stati. Il fatto di lavorare con una sovrapposizione di stati di un sistema, invece che con uno

1 Per informazioni sui bit classici probabilitici si faccia riferimento alla sezione 3.2 di [15].

stato definito alla volta, implica che si possono effettuare determinate operazioni che coinvolgono più stati contemporaneamente.

Nonostante ciò, uno degli ostacoli principali di questo campo è rappresentato proprio dalla caratteristica alla base della meccanica quantistica: quando si ha un qubit in una sovrapposizione di stati, non possiamo conoscerne i dettagli intrinseci in un determinato istante, a meno di effettuare una misura; se ci provassimo, otterremmo un risultato ben definito, facendo collassare la funzione d'onda che descrive il qubit, riducendolo in fin dei conti ad un bit con uno stato definito (esattamente quello che misuriamo), e perdendo tutte le informazioni quantistiche contenute nel qubit.

Per avere un'idea di come un CQ si rapporti con un computer ordinario, si possono considerare diversi algoritmi quantistici, che forniscono accelerazioni esponenziali se confrontati con le loro controparti classiche; l'algoritmo di fattorizzazione in numeri primi di Shor è uno tra i più famosi [12].

Alla luce di questi fatti, negli ultimi anni c'è stata particolare attenzione nei riguardi del nuovo campo del machine learning quantistico (MLQ) (o quantum machine learning (QML)), che mette insieme il ML con il quantum computing.

Più nello specifico, si parla di machine learning migliorato quantisticamente quando la computazione quantistica è usata per migliorare algoritmi di ML classico. Ciononostante, entrambi i campi sono benefici l'un l'altro, poiché il ML classico può anche essere usato per migliorare aspetti della computazione quantistica [5]. Questa tesi si occuperà solo del campo del machine learning aumentato quantisticamente.

1.1 MOTIVAZIONE

Il QML è un campo molto giovane e in fermento. Stiamo assistendo a qualcosa di analogo a quella che era la nascita dei computer classici, ovvero lo sviluppo di un numero sempre crescente di algoritmi, tecniche ed hardware di base in un ambiente in cui le cose consolidate sono ben poche: il QML è solo una delle branche che sono state reputate promettenti per quanto riguarda i vantaggi portati dai computer quantistici [7]. In ogni caso c'è necessità di inventare nuove strategie per risolvere anche i problemi più semplici: per avere risultati affidabili spesso si richiede un numero considerevole di qubit ed unità di memorizzazione che possano conservare informazioni quantistiche, come la quantum random access memory (QRAM). Al giorno d'oggi il numero massimo di qubit a superconduzione, da quanto viene riferito, è di 50 [4].

I campi che si prevede otterranno accelerazioni maggiori grazie al quantum computing sono il machine learning, la simulazione di sistemi quantomeccanici, i problemi di ottimizzazione e l'analisi finanziaria.

Negli ultimi anni ci sono state varie implementazioni innovative di algoritmi di MLQ [14]: seguendo i passi che hanno portato alla nascita del ML nel XX secolo, Tacchino et al. [13] hanno implementato un

percettrone su hardware quantistico; Schuld et al. [9] hanno invece implementato un algoritmo k-nearest neighbours (KNN). Questa tesi parte proprio da quest'ultimo articolo, analizzando le possibilità di miglioramento in conseguenza delle novità nell'hardware e nelle tecniche di codifica dei dati.

1.2 DOMANDA DI RICERCA

Prendendo come punto di partenza gli algoritmi proposti nell'ambito di ricerca del QML questa tesi proverà a rispondere alla seguente domanda:

> quali sono le prestazioni di un algoritmo k-nearest neighbours di quantum machine learning implementato su hadware quantistico di piccola e media dimensione attualmente disponibili e come scala all'aumentare delle risorse?

I capitoli seguenti introdurranno le basi teoriche necessarie e gli strumenti usati per rispondere a questa domanda.

2 TEORIA DI BASE

L'unità di base di un computer classico è il bit e l'unità di base di un computer quantistico è una generalizzazione del concetto di bit, chiamato qubit, che sarà discusso in sezione 2.1. Nella sezione 2.2 si presentano le porte logiche quantistiche, che manipolano i qubit.

2.1 BIT E QUBIT

Un bit è un'unità di informazione che descrive un sistema classico bidimensionale.

I due possibili stati sono generalmente scritti come 0 e 1, o meglio, nel nostro caso $|0\rangle$ e $|1\rangle$.

Usando una notazione matriciale, possiamo rappresentare questi due stati come

$$|0\rangle = \begin{matrix} 0 \\ 1 \end{matrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{matrix} 0 \\ 1 \end{matrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \tag{2.1}$$

dove i numeri a margine delle matrici indicano a quale stato si riferisce un determinato elemento di base. Poiché queste due rappresentazioni sono ortogonali, abbiamo il buon vecchio bit che può trovarsi in un determinato stato tra $|0\rangle$ e $|1\rangle$.

Un bit quantistico o qubit è un'unità di informazione che descrive un sistema quantistico bidimensionale.

Si rappresenterà il qubit come una matrice 2×1 a valori complessi

$$\begin{matrix} 0 \\ 1 \end{matrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}, \tag{2.2}$$

dove $|c_0|^2 + |c_1|^2 = 1$. Si noti che un bit classico è un caso particolare di qubit. $|c_0|^2$ va interpretata come la probabilità che dopo una misura del qubit, questo sarà trovato nello stato $|0\rangle$. $|c_1|^2$ va interpretata come la probabilità che dopo una misura del qubit, questo sarà trovato nello stato $|1\rangle$. Allorquando si misura lo stato di un qubit, questo diventa automaticamente un bit. Quindi non "vedremo" mai un qubit generico. Nonostante ciò, essi esistono e sono i protagonisti di questa avventura.

I bit $|0\rangle$ e $|1\rangle$ formano la base canonica di $\mathbb{C}^2$, quindi ogni qubit può essere scritto come

$$\begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = c_0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + c_1 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = c_0 |0\rangle + c_1 |1\rangle . \tag{2.3}$$

L'implementazione pratica di un qubit può essere fatta in vari modi:

- un elettrone in due differenti orbitali attorno al nucleo di un atomo;
- un fotone in uno tra due stati di polarizzazione;
- una particella subatomica che ha una tra le due direzioni di spin.

L'importante è che ci siano effetti abbastanza rilevanti di indeterminazione e di sovrapposizione per rappresentare un qubit.

Usando coordinate sferiche polari, un singolo qubit può essere visualizzato sulla cosiddetta sfera di Bloch (vedi figura 2.1), parametrizzando c_0 e c_1 dell'Eq. 2.3 come segue:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\,. \tag{2.4}$$

La sfera di Bloch ha raggio unitario. Lo stato $|0\rangle$ del qubit è definito in modo che si trovi lungo il semiasse z positivo e lo stato $|1\rangle$ è definito in modo che si trovi lungo il semiasse z negativo. È interessante notare che questi due stati sono mutuamente ortogonali in H_2, sebbene non lo siano sulla sfera di Bloch.

Gli stati del qubit sull'equatore della sfera, come gli assi coordinati x e y, rappresentano sovrapposizioni uniformi dove $|0\rangle$ e $|1\rangle$ hanno entrambi probabilità di misura pari a 0,5. L'asse x, ad esempio, rappresenta la sovrapposizione uniforme $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. Qualunque stato bidimensionale arbitrario $|\psi\rangle$ può essere decomposto nei suoi angoli polari θ e φ e visualizzato come un vettore sulla sfera di Bloch (dopo essere stato normalizzato, se necessario). Tale oggetto è chiamato vettore di Bloch dello stato $|\psi\rangle$ del qubit. La sfera di Bloch sarà lo strumento principale di visualizzazione per le manipolazioni di qubit in questa tesi.

I computer con un solo bit non sono molto interessanti, e tantomeno i CQ con un solo qubit. Se prendiamo un byte, ovvero otto bit, possiamo avere una generica configurazione 10011101. La rappresentazione associata sarebbe

$$|1\rangle\otimes|0\rangle\otimes|0\rangle\otimes|1\rangle\otimes|1\rangle\otimes|1\rangle\otimes|0\rangle\otimes|1\rangle\,. \tag{2.5}$$

Un qubit in questa configurazione appartiene a $\mathbb{C}^2\otimes\mathbb{C}^2\otimes\mathbb{C}^2\otimes\mathbb{C}^2\otimes\mathbb{C}^2\otimes\mathbb{C}^2\otimes\mathbb{C}^2\otimes\mathbb{C}^2 = (\mathbb{C}^2)^{\otimes 8}$. Questo spazio vettoriale ha dimensione $2^8 = 256$. La sua rappresentazione matriciale è allora

$$\begin{matrix} 00000000 \\ 00000001 \\ \vdots \\ 10011101 \\ \vdots \\ 11111110 \\ 11111111 \end{matrix}\begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix}. \tag{2.6}$$

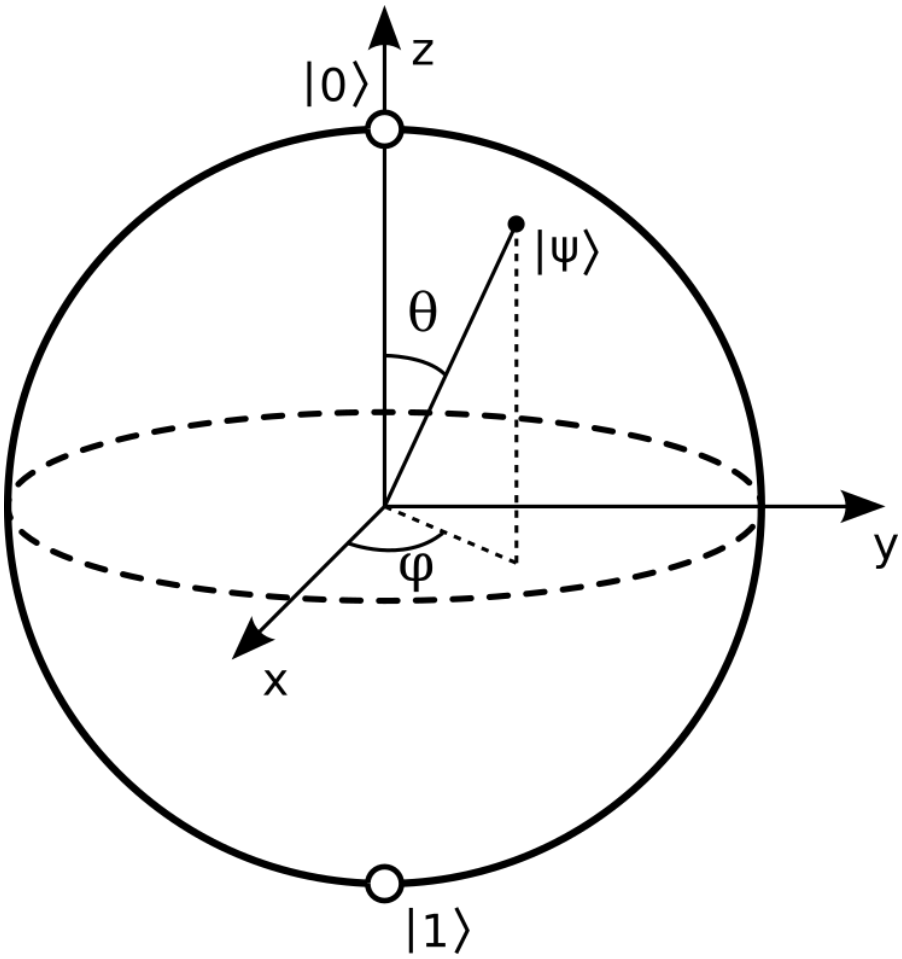


Figura 2.1: La sfera di Bloch

La generalizzazione al mondo quantistico è uno stato in sovrapposizione

$$\begin{matrix} 00000000 \\ 00000001 \\ \vdots \\ 11111110 \\ 11111111 \end{matrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{254} \\ c_{255} \end{pmatrix}, \tag{2.7}$$

dove $\sum_{i=0}^{255} |c_i|^2 = 1$. Nei computer classici è necessario indicare lo stato di ogni bit in un byte. Questo significa scrivere su otto bit. Nel caso quantistico, uno stato di otto qubit è dato scrivendo 256 numeri complessi. Questa crescita esponenziale è una delle ragioni per cui i ricercatori hanno manifestato interesse per la nozione di quantum computing.

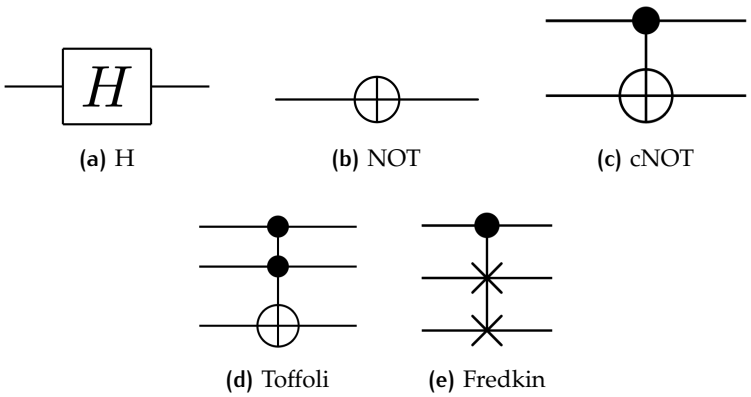
2.2 PORTE LOGICHE QUANTISTICHE

Una porta logica quantistica è un operatore che agisce sui qubit. Tali operatori saranno rappresentati da matrici unitarie.

Le porte logiche quantistiche che agiscono su un sigolo qubit possono essere rappresentate come matrici unitarie 2×2 le cui azioni su un qubit possono essere visualizzate come rotazioni della sfera di Bloch.

Le porte logiche quantistiche a più qubit agiscono su almeno due qubit allo stesso tempo. Similmente alle porte a singolo qubit, una porta quantistica a n qubit può essere rappresentata come una matrice unitaria $2^n \times 2^n$. Dato che sono coinvolti molteplici qubit, la sfera di Bloch non può più essere usata per visualizzarne l'azione.

Tra le porte quantistiche più usate si possono trovare le porte Hadamard, NOT, NOT controllata, Toffoli e Fredkin, visibili in figura 2.2.



(a) H **(b)** NOT **(c)** cNOT

(d) Toffoli **(e)** Fredkin

Figura 2.2: Alcune porte quantistiche

Molto importanti sono le tre matrici di Pauli:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -1 \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.8}$$

Si può notare che la porta X non è altro che la porta NOT.

Altre importanti matrici usate spesso sono

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad e \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}. \tag{2.9}$$

Ogni matrice unitaria 2×2, ovvero un'operazione su un qubit, può essere visualizzata come una rotazione o un'inversione della sfera di Bloch. Le matrici X, Y e Z sono modi di girare la sfera di Bloch di π attorno agli assi x, y e z rispettivamente. Si ricordi che X non è altro che la porta NOT, e porta $|0\rangle$ in $|1\rangle$ e viceversa. Ma per di più porta tutto quello che si trova sopra l'equatore sotto di esso. Le altre matrici di Pauli funzionano in maniera simile.

In alcuni casi non si vuole effettuare una rotazione completa di π ma si vuole ruotare la sfera di un angolo θ lungo una particolare direzione. In questo caso, dato un vettore tridimensionale $D = (D_x, D_y, D_z)$ di modulo 1, possiamo costruire una rotazione della sfera di Bloch attorno ad esso in questo modo:

$$R_D(\theta) = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}(D_x X + D_y Y + D_z Z). \tag{2.10}$$

La sfera di Bloch è utile solo per visualizzare stati e trasformazioni di un qubit. Quando abbiamo a che fare con più qubit, la dimensionalità degli stati non ci permette di avere una rappresentazione intuitivamente semplice.

Tra le porte a più qubit che verranno usate in questa tesi ci sono le porte controllate: queste hanno il loro effetto se tutti i qubit di controllo sono nello stato $|1\rangle$, altrimenti lasciano i bersagli della loro azione immutati. Si propone a titolo esemplificativo la porta Deutsch, ovvero una rotazione attorno ad un asse D di un qubit bersaglio, condizionata dallo stato di due qubit di controllo:

$$D_\theta : |a, b, c\rangle \mapsto \begin{cases} i\cos(\theta)\,|a, b, c\rangle + \sin(\theta)\,|a, b, 1-c\rangle & \text{per } a = b = 1, \\ |a, b, c\rangle & \text{altrimenti.} \end{cases} \tag{2.11}$$

Nei capitoli successivi verrà implementata la porta $C^n R_y(\theta)$ con n qubit di controllo.

Le due porte appena descritte non si trovano generalmente nel set universale di porte del computer quantistico che si va ad utilizzare, ma devono essere approssimate attraverso una successione di porte di base, che può essere più o meno lunga. La ricerca di nuovi modi per implementare porte complesse in maniera nativa [11] è uno degli sforzi per contrastare il fenomeno della decoerenza, uno degli ostacoli sostanziali più intriganti del calcolo quantistico.

La decoerenza è la perdita di purezza dello stato di un sistema quantistico come risultato di interazione con l'ambiente.

Esistono due paramentri per descrivere quanto è stabile un sistema nei confronti della decoerenza:

- il tempo di decoerenza longitudinale, ovvero il decadimento degli stati più energetici verso quelli meno energetici (si pensi ad un elettrone nello stato eccitato in un atomo);
- il tempo di decoerenza trasversale, tempo in cui statisticamente uno stato di sovrapposizione decade in una configurazione ben definita.

2.3 MACHINE LEARNING

Per comprendere la versione quantistica dell'algoritmo KNN pesato sulla distanza proposto da Schuld (2014) che verrà usato successivamente è richiesta una conoscenza della sua versione classica.

Il KNN non apprende come discriminare le classi dai vettori di apprendimento, ma memorizza il data set di apprendimento tutto intero. L'algoritmo in sé è piuttosto semplice e può essere riassunto con i seguenti passaggi:

- scegli il numero k e una metrica per la distanza;
- trova i k elementi più vicini al campione da classificare;
- assegna l'etichetta di classe con un voto a maggioranza.

In base alla metrica scelta, l'algoritmo KNN trova i k campioni nel data set di apprendimento che sono più vicini (simili) al punto da classificare. La classe del nuovo punto è allora determinata da un voto a maggioranza dei suoi k vicini. Il vantaggio principale di questo approccio basato sulla memoria è che il classificatore si adatta immediatamente man mano che aggiungiamo vettori di apprendimento. Dall'altro lato, il difetto è che la complessità computazionale per classificare nuovi campioni cresce al più linearmente con il numero di vettori di apprendimento. Per di più, non possiamo ignorare con leggerezza alcun vettore di training dato che non c'è un vero e proprio apprendimento. Così, lo spazio di archiviazione può diventare un problema nodale se si lavora con grandi data set. [8]

Si vede subito come le proprietà dei quantum computer di parallelizzazione del calcolo e di immagazzinamento dei dati possano essere decisive in problemi del genere di grandi dimensioni.

3 STRUMENTI

Il lavoro per questa tesi è stato svolto principalmente usando un computer con sistema operativo GNU/Linux Ubuntu Dell Inspiron 3552 con 8 GB di RAM. Per lo sviluppo dei circuiti quantistici si è usato gli strumenti di sviluppo e di simulazione messi a disposizione dall'IBM. Questi permettono di progettare, simulare ed eseguire su dei quantum computer reali gli algoritmi scritti. Il linguaggio di programmazione, sia per l'analisi dei dati che per la prototipazione del circuito, è Python; grazie al lavoro della comunità open source questo linguaggio si è imposto come strumento omnicomprensivo per una varietà sempre crescente di lavori di ricerca scientifica.

3.1 QISKIT

Qiskit è un'interfaccia di programmazione che permette di scrivere circuiti quantistici e simularne l'esecuzione sul proprio computer o inviare un ordine di esecuzione a un vero computer quantistico tramite l'interfaccia offerta dall'IBM Quantum Experience. È consigliata l'esecuzione tramite l'interfaccia Jupyter Notebook per la manipolazione dei risultati in tempo reale.

3.2 IBM Q EXPERIENCE

L'IBM Q Experience è un servizio offerto gratuitamente da IBM che permette a chiunque di avere a che fare con un computer quantistico. Sono presenti risorse didattiche per imparare a scrivere il primo circuito, strumenti di comunità come una piattaforma di domande e risposte e soprattutto un sistema per creare i propri algoritmi. Si può accedere agli ordini di esecuzione inviati tramite Qiskit e recuperarne i risultati in un secondo momento.

Parte II

PRATICA

Implementazione dell'algoritmo k-nearest neighbour e della tecnica di costruzione della QRAM; simulazione ed esecuzione su hardware quantistico dei circuiti progettati; analisi dei risultati ottenuti.

4 IMPLEMENTAZIONE

4.1 PREPARAZIONE DI UNO STATO QUANTISTICO

Per analizzare dei dati classici attraverso un computer quantistico abbiamo bisogno di codificare in qualche modo le informazioni contenute nei nostri insiemi. Nel caso specifico, si parla delle coordinate dei vettori nello spazio delle caratteristiche e la classe associata ad ognuno di essi. Per fare questo, costruiamo degli stati quantistici ad hoc che rappresentino i vettori dati in maniera coerente. La procedura usata in questa tesi segue la tecnica di costruzione flip-flop QRAM (FF-QRAM) proposta da Park, Petruccione e Rhee [6].

4.1.1 Flip-flop QRAM

La FF-QRAM è usata per generare un quantum database (QDB) inizializzato in maniera arbitraria. Nell'illustrare l'algoritmo di cotruzione verranno usati due registri quantistici: il primo, denotato genericamente $|j\rangle$, o con il pedice B, indica quale bus di memoria viene usato per il passaggio in corso, mentre il secondo, denotato $|b_l\rangle_R$, con il pedice R, sarà il registro che contiene i valori codificati del vettore dati. I vettori $|j\rangle_B$ vengono anche detti appartenere alla base computazionale. Lo stato finale dei qubit può essere arbitrario e l'ampiezza di probabilità ψ_j con cui è accessibile ciascuno stato $|j\rangle_B$ della base computazionale codifica i valori classici di partenza.

L'operazione QRAM sui qubit bus e registro sovrappone un insieme di dati classici D come

$$\mathrm{QRAM}(D) \sum_j \psi_j |j\rangle_B |0\rangle_R \equiv \sum_l \psi_l |\vec{d}^{(l)}\rangle_B |b_l\rangle_R , \tag{4.1}$$

La FF-QRAM è implementata sistematicamente con elementi comuni dei circuiti quantistici, che includono la porta di Pauli X controllata classicamente, $\bar{c}X$, e la porta di rotazione controllata da n qubit, $C^nR_p(\theta)$. La $\bar{c}X$ inverte il qubit bersaglio solo quando il bit classico di controllo è zero. La porta $C^nR_p(\theta)$ ruota il qubit bersaglio di θ attorno all'asse p della sfera di Bloch solo se tutti gli n qubit sono 1.

L'idea soggiacente al modello della FF-QRAM è rappresentata in figura 4.1, che descrive la procedura per sovrapporre due stringhe di bit indipendenti $\vec{d}^{(l)}$ e $\vec{d}^{(l+1)}$ con ampiezze di probabilità desiderate

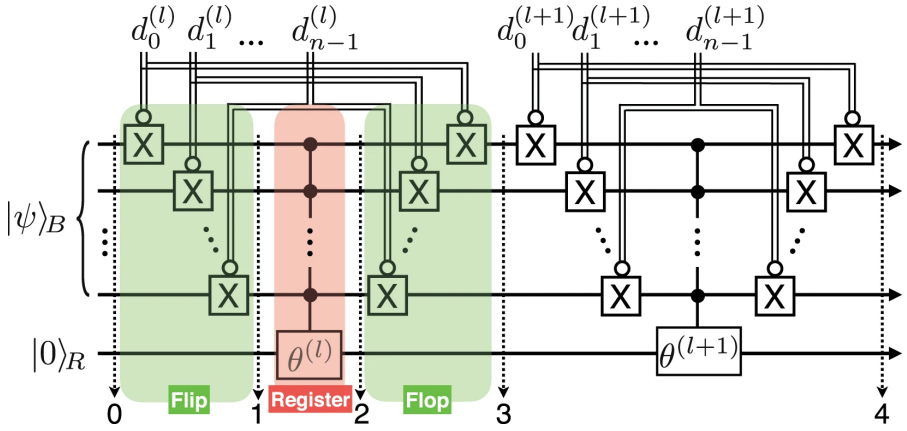


Figura 4.1: Procedimento di costruzione FF-QRAM
Circuito quantistico per FF-QRAM che scrive le stringhe di bit $\vec{d}^{(l)}$ e $\vec{d}^{(l+1)}$ come una sovrapposizione di stato quantistico con ampiezze di probabilità determinate da $\theta^{(l)}$ e $\theta^{(l+1)}$ rispettivamente, usando porte di rotazione controllate da più qubit. Le linee doppie indicano operazioni controllate classicamente, ed il cerchio vuoto (pieno) indica che la porta è attivata quando il bit (qubit) di controllo è 0 (1). Le frecce tratteggiate e numerate indicano i vari passaggi descritti nel testo principale.

nello stato del qubit bus $|\psi\rangle_B$. Lo stato iniziale può essere espresso focalizzandosi su $\vec{d}^{(l)}$ come

$$|\psi_0\rangle_l = \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle |0\rangle_R + \sum_{j \neq \vec{d}^{(l)}} \psi_j |j\rangle |0\rangle_R , \tag{4.2}$$

dove $|\psi_s\rangle_l$ denota lo stato degli (n) qubit nel processo di scrittura dell'l-esimo valore dati, osservato all's-esimo passo in figura 4.1.

Le porte $\bar{c}X$ controllate da $\vec{d}^{(l)}$ risistemano gli stati della base computazionale dei qubit bus cosicché $|\vec{d}^{(l)}\rangle$ diventa $|1\rangle^{\otimes n}$, ed il resto dei bit quantistici si invertono di conseguenza:

$$|\psi_1\rangle_l = \psi_{\vec{d}^{(l)}} |1\rangle^{\otimes n} |0\rangle_R + \sum_{|\overline{j \oplus \vec{d}^{(l)}}\rangle \neq |1\rangle^{\otimes n}} \psi_j |\overline{j \oplus \vec{d}^{(l)}}\rangle |0\rangle_R . \tag{4.3}$$

La sovralinea nell'ultimo termine indica che l'inversione del bit avviene se il bit di controllo è 0. Dopo il passo 1, la rotazione controllata dai qubit, $C^n R_y(\theta^{(l)})$, denotata come $\theta^{(l)}$ nella figura, è applicata al qubit registro. Lo stato quantistico al passo 2 diventa

$$|\psi_2\rangle_l = \psi_{\vec{d}^{(l)}} |1\rangle^{\otimes n} |\theta^{(l)}\rangle_R + \sum_{|\overline{j \oplus \vec{d}^{(l)}}\rangle \neq |1\rangle^{\otimes n}} \psi_j |\overline{j \oplus \vec{d}^{(l)}}\rangle |0\rangle_R , \tag{4.4}$$

dove $|\theta\rangle = \cos\theta |0\rangle + \sin\theta |1\rangle$. Le porte $\bar{c}X$ condizionate da $\vec{d}^{(l)}$ sono applicate di nuovo per far ritornare alla condizione precedente lo stato dei bus:

$$|\psi_3\rangle_l = \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle |\theta^{(l)}\rangle_R + \sum_{j \neq \vec{d}^{(l)}} \psi_l |j\rangle |0\rangle_R . \tag{4.5}$$

Il secondo giro registra i dati successivi di $\vec{d}^{(l+1)}$ e $\theta^{(l+1)}$:

$$|\psi_4\rangle_{l,l+1} = \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle |\theta^{(l)}\rangle_R + \\ + \psi_{\vec{d}^{(l+1)}} |\vec{d}^{(l+1)}\rangle |\theta^{(l+1)}\rangle_R + \sum_{j \neq \vec{d}^{(l)}, \vec{d}^{(l+1)}} \psi_j |j\rangle |0\rangle_R . \tag{4.6}$$

Questo processo può essere ripetuto tante volte quanti sono i dati da inserire. In questo modo, M valori possono essere registrati con pesi non uniformi per generare lo stato

$$\sum_{l=0}^{M-1} \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle \left[\cos\theta^{(l)} |0\rangle_R + \sin\theta^{(l)} |1\rangle_R\right] + \sum_{j \notin \{\vec{d}^{(l)}\}} \psi_j |j\rangle |0\rangle_R . \tag{4.7}$$

Infine, il QDB richiesto nell'eq. 4.1 può essere ottenuto selezionando un angolo $\theta^{(l)}$ appropriato che sia legato all'ampiezza di probabilità desiderata b_l, e postselezionando il risultato di misura $|1\rangle_R$. La probabilità di misurare $|1\rangle_R$ è

$$P(1) = \sum_{l=0}^{M-1} |\psi_{\vec{d}^{(l)}} \sin\theta^{(l)}|^2. \tag{4.8}$$

Il costo per preparare uno stato con questo procedimento è di $\mathcal{O}(n)$ qubit e $\mathcal{O}(Mn)$ operazioni quantistiche.

4.2 QKNN

L'algoritmo KNN quantistico pesato sulla distanza evidenziato in questa sezione è stato proposto da Schuld, Sinayskiy e Petruccione [10] e implementato da Schuld, Fingerhuth e Petruccione [9] sul computer quantistico a 5 qubit dell'IBM. L'algoritmo viene qui introdotto, per poi discutere nel capitolo successivo i risultati.

L'obiettivo è di classificare una registro di qubit in entrata basandosi su un numero più o meno grande di configurazioni di qubit di apprendimento. Ogni configurazione di apprendimento appartiene ad una certa classe, che è codificata in un registro quantistico di classe in entanglement con la rispettiva configurazione di apprendimento. L'idea principale è che l'algoritmo KNN calcola la distanza tra il motivo in entrata ed ogni configurazione di apprendimento attraverso una serie di porte quantistiche. Successivamente, queste distanze sono invertite in modo che i vettori di apprendimento vicini a quello di input hanno l'inverso della distanza maggiore rispetto ai vettori di apprendimento più remoti. Queste distanze inverse sono poi scritte nelle corrispondenti ampiezze dello stato di classe. Si noti che il modulo quadro di un'ampiezza di un particolare stato di classe determina la probabilità di misurare quella classe. Quindi, la probabilità di misurare una data classe dipende adesso dall'inverso delle distanze tra i

vettori di apprendimento della classe in esame ed il vettore d'input. L'inverso della distanza può essere visto come un peso dipendente dalla distanza, dato che aumenta la probabilità di misurare la classe con i vettori di apprendimento più vicini a quello d'input. I passaggi necessari per l'implementazione dell'algoritmo KNN quantistico sono evidenziati in dettaglio qui di seguito.

4.2.1 Struttura dell'algoritmo

Il primo passo dell'algoritmo consiste nel preparare una *sovrapposizione dell'insieme di apprendimento* che ne contenga i dati codificati. Usando un idoneo schema di preparazione dello stato, il circuito di classificazione porta un sistema di n qubit nello stato

$$|D\rangle = \frac{1}{\sqrt{2MC}} \sum_{m=1}^{M} |m\rangle \left(|0\rangle |\psi_x\rangle + |1\rangle |\psi_{t^m}\rangle\right) |c^m\rangle . \tag{4.9}$$

Qui $|m\rangle$ è un registro indice che prende valori $m = 1, \ldots, M$ e che contrassegna l'm-esimo vettore di apprendimento. Il secondo registro è un singolo qubit ancilla, il cui stato eccitato è in entanglement con il terzo registro che codifica l'm-esimo stato di apprendimento $|\psi_{t^m}\rangle = \sum_{i=0}^{N-1} t_i^m |i\rangle$, mentre il suo stato fondamentale è in entanglement con il terzo registro che codifica il vettore d'input $|\psi_x\rangle = \sum_{i=0}^{N-1} x_i |i\rangle$. Il quarto registro codifica la classe nell'ampiezza dei propri qubit. Effettivamente, si crea una funzione d'onda che contiene i vettori di apprendimento insieme ad M copie del nuovo input. La costante di normalizzazione C dipende dal processo di ottimizzazione dei dati; assumeremo in seguito che i vettori di apprendimento siano normalizzati e quindi $C = 1$.

Dopo aver preparato lo stato iniziale, il circuito quantistico consiste di sole tre operazioni. Prima, l'applicazione di una porta Hadamard sull'ancilla fa interferire le copie del vettore d'input con i vettori di apprendimento:

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^{M} |m\rangle \left(|0\rangle |\psi_{x+t^m}\rangle + |1\rangle |\psi_{x-t^m}\rangle\right) |c^m\rangle , \tag{4.10}$$

dove $|\psi_{x\pm t^m}\rangle = |\psi_x\rangle \pm |\psi_{t^m}\rangle$.

La seconda operazione è una misura condizionale che seleziona il ramo con l'ancilla nello stato $|0\rangle$. Questa postselezione ha successo con probabilità $p_{acc} = \frac{1}{4M} \sum_m |x + t^m|^2$. È più probabile che abbia successo se la distanza euclidea al quadrato complessiva dell'insieme dati di apprendimento rispetto al nuovo input è piccola. Se la misura condizionale ha successo, lo stato risultanta è dato da

$$\frac{1}{2\sqrt{M p_{acc}}} \sum_{m=1}^{M} \sum_{i=1}^{N} |m\rangle \left(x_i + t_i^m\right) |i\rangle |c^m\rangle . \tag{4.11}$$

Le ampiezze pesano i qubit classe $|c^m\rangle$ con la distanza dell'm-esimo vettore dati dal nuovo input. In questo stato, la probabilità di misurare il qubit classe nello stato $|s\rangle$

$$p(|c\rangle = |s\rangle) = \frac{1}{4Mp_{acc}} \sum_{m||c\rangle=|s\rangle} |x + t^m|^2, \tag{4.12}$$

riflette la probabilità di predire la classe s per il nuovo input.

La scelta di vettori caratteristica normalizzati assicura che $\frac{1}{4Mp_{acc}} \sum_m |x + t^m|^2 = 1 - \frac{1}{4Mp_{acc}} \sum_m |x - t^m|^2$, e la scelta della classe con maggior probabilità in questo modo implementa il classificatore. Questo significa che se il vettore d'input è molto vicino ai vettori di training di una data classe, quella classe uscirà come risultato più frequentemente delle altre.

5 RISULTATI E DISCUSSIONE

Per l'implementazione dell'algoritmo è stato scelto il ben noto data set Iris, consistente in una tabella di lunghezze e larghezze del petalo e del sepalo collegate a tre specie di fiore Iris. Come evidenziato in Schuld et al. [9], l'insieme dati si è dovuto standardizzare e normalizzare prima dell'utilizzo. Il passo successivo è stato ripetere l'esperienza dell'articolo per comprenderne appieno il funzionamento e poterlo estendere. Questa tesi parte dalla situazione base, in cui si usano due caratteristiche delle quattro disponibili, due vettori di apprendimento e si possono classificare solo due classi alla volta. Questa versione esemplificativa è stata fatta girare con successo sul processore quantistico a 5 qubit messo a disposizione dall'IBM. Di fronte alla disponibilità di un nuovo computer quantistico a 16 qubit[1], ci si chiede quali sono le prossibilità di miglioramento delle implementazioni pratiche. Si è lavorato sulle seguenti proprietà:

- aumentare il numero di classi riconosciute;
- aumentare il numero di caratteristiche considerate;
- aumentare il numero di vettori di training.

Per fare ciò, si è dovuto tenere in considerazione i compromessi in termini di numero di qubit ed efficienza di classificazione: infatti, per realizzare ognuno di questi punti c'è bisogno di impegnare più qubit, non solo per una questione di memoria, ma anche per la complessità delle operazioni da effettuare; un'esempio è dato dalla porta $C^nR_y(\theta)$, usata nel procedimento di costruzione della QRAM (vedi sezione 4.1.1), che necessita di un qubit ancilla in più per ogni qubit di controllo aggiuntivo.

5.1 OTTIMIZZAZIONE DEI DATI

Il data set Iris completo si presenta nella forma mostrata in figura 5.1, dove sono indicate le prime due caratteristiche sugli assi coordinati.

La prima operazione sui dati è la standardizzazione, che trasla e scala i punti in modo che abbiamo media nulla e deviazione standard unitaria. Si possono notare gli effetti nella figura 5.2, osservando i nuovi valori sugli assi.

1 https://developer.ibm.com/dwblog/2017/quantum-computing-16-qubit-processor/, Recuperato l'8 settembre 2019

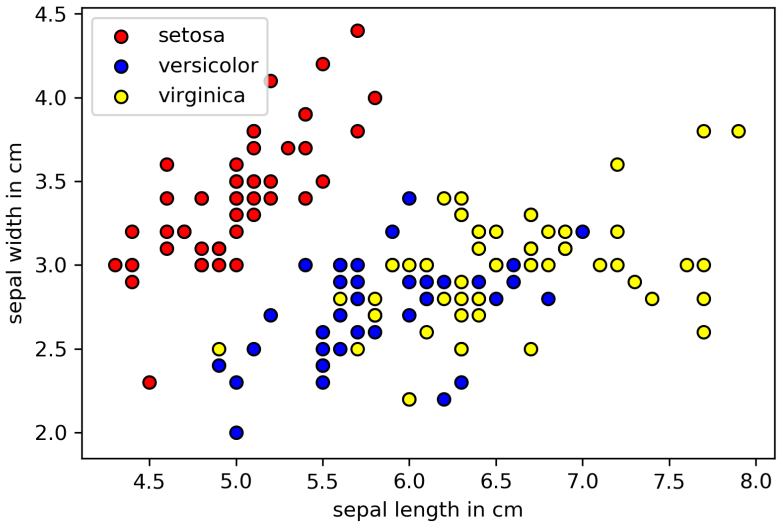


Figura 5.1: Data set Iris non elaborato

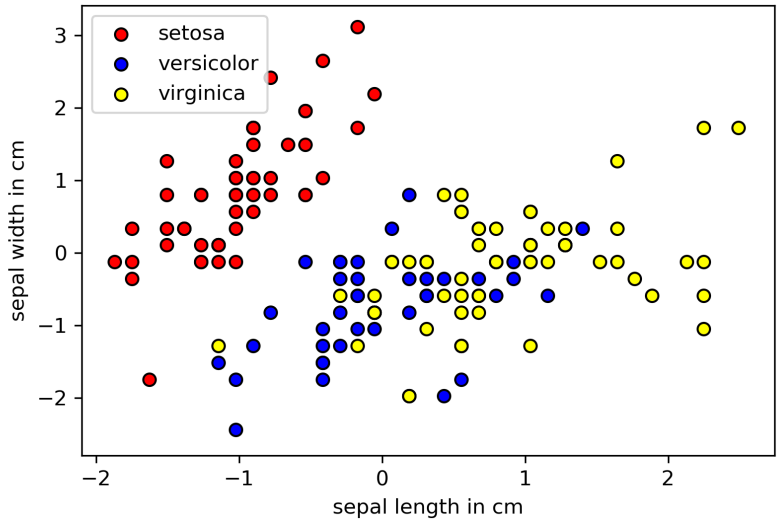


Figura 5.2: Data set Iris standardizzato

La normalizzazione termina il processo preliminare di ottimizzazione dei dati. Si può vedere l'effetto della normalizzazione su un data set con due caratteristiche in figura 5.3.

Avendo effettuato queste operazioni, si deve ora tradurre le coordinate di ciascun vettore nello spazio delle caratteristiche in un angolo di rotazione da applicare al qubit registro della QRAM.

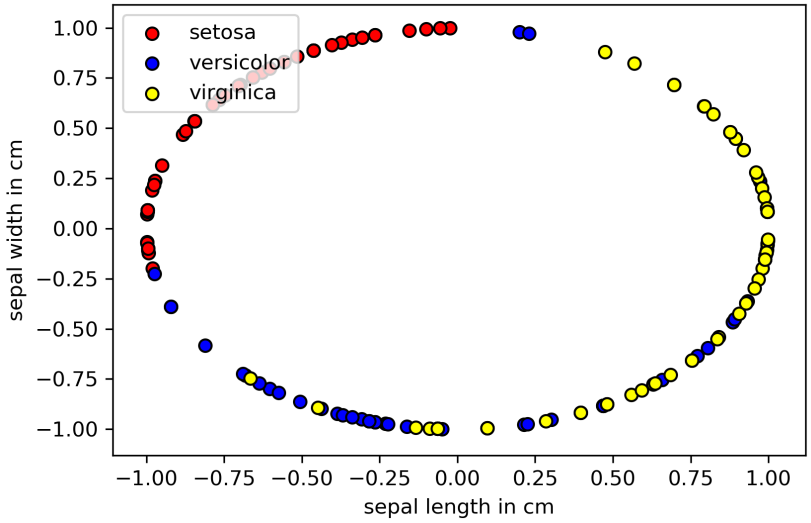


Figura 5.3: Data set Iris normalizzato

Facendo riferimento all'eq. 4.8, troviamo che lo stato costruito, dopo la misura condizionale, è

$$\sum_{l=0}^{M-1} \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle \sin\theta^{(l)} |1\rangle_R . \tag{5.1}$$

La relazione esistente tra i valori b_l, ovvero le caratteristiche standardizzate e normalizzate, e le ampiezze di probabilità nella QRAM è

$$\theta^{(l)} = \arcsin b_l. \tag{5.2}$$

5.2 SCRITTURA DELL'ALGORITMO

Avendo acquisito le nozioni teoriche e i parametri iniziali, si prova adesso a costruire il circuito quantistico attraverso le porte dell'insieme universale di base, disponibili grazie all'interfaccia IBM Q Experience ed al kit di sviluppo Qiskit.

Seguendo la figura A.1 in appendice, si esamina il circuito quantistico di base:

1. i qubit ancilla ed indice vengono posti in sovrapposizione uniforme;
2. il vettore d'input x è posto in entanglement con lo stato fondamentale dell'ancilla;
3. il vettore di training t^0 è posto in entanglement con lo stato eccitato dell'ancilla e con lo stato fondamentale del qubit indice;

4. il vettore di training t^1 è posto in entanglement con lo stato eccitato dell'ancilla e del qubit indice;
5. il qubit classe è invertito condizionato dal fatto che il qubit indice sia $|1\rangle$; questo completa la preparazione iniziale dello stato;
6. nell'ultimo passaggio la porta Hadamard fa interferire le copie di x con i vettori d'apprendimento e l'ancilla è misurata, seguita da una misura del qubit classe.

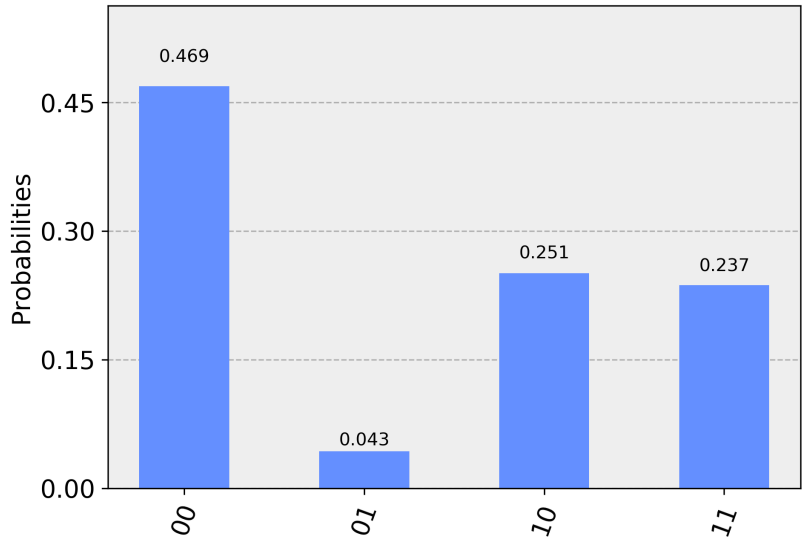
Considereremo validi per i nostri scopi solo le misure del qubit classe ottenute quando il qubit ancilla è trovato nello stato $|0\rangle$. Possiamo notare, tra le altre cose, che la porta $C^nR_y(\theta)$ è in realtà formata da una successione di più porte di base. Mentre nei primi tempi era necessario inserire manualmente i singoli elementi logici per eseguire una operazione complessa, nella realizzazione attuale si è sfruttato il lavoro della comunità open source di Qiskit, che ha messo a disposizione comandi veloci per aggiungere facilmente porte multi controllate.

5.2.1 Classificazione iniziale

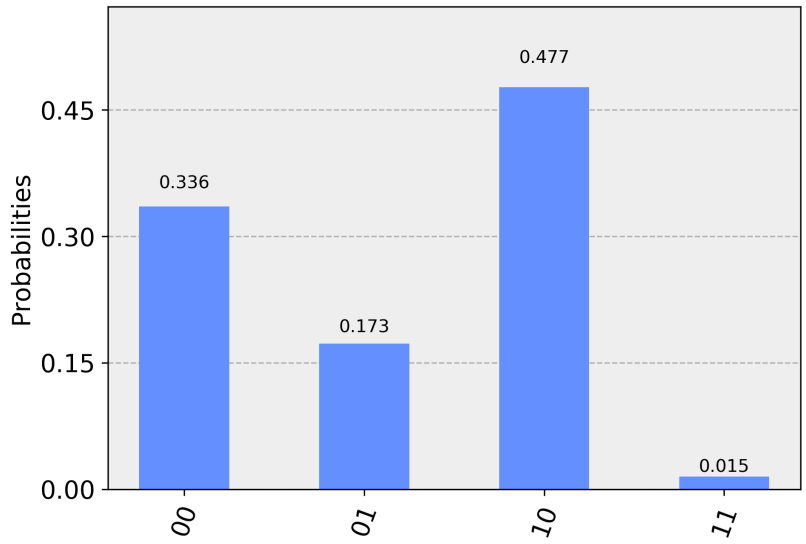
Si mostra un esempio di risultato di classificazione usando le prime due caratteristiche del data set e gli elementi appartenenti alle classi setosa e versicolor. Si scelgono come vettori di training un elemento per ciascuna classe, nel nostro caso il vettore 34 ed il vettore 86 dal data set, rispettivamente setosa e versicolor. Si assegna alla classe setosa lo stato $|0\rangle$ del qubit classe ed alla classe versicolor lo stato $|1\rangle$. Si sottopongono poi a classificazione, in due esperimenti separati, due vettori sconosciuti: il vettore 29 (setosa) ed il vettore 57 (versicolor).

Il primo passo è simulare l'esperimento sul computer in uso, dato che il problema in esame è facilmente eseguibile su un comune portatile. I risultati non filtrati per i due esperimenti sono visibili nel riquadro 5.4. Nella didascalia sono scritti i conteggi corrispondenti ad un determinato esito di misura sui due qubit ancilla e classe. La cifra sulla destra contiene la misura del qubit ancilla, quella sulla sinistra del qubit classe. Tali valori, normalizzati ad uno, sono rappresentati in un istogramma, che approssima la distribuzione di probabilità degli esiti di misura per grandi numeri di esecuzioni dell'algoritmo. Per questo motivo si userà preferibilmente un numero di esecuzioni pari al massimo permesso sui computer quantistici dell'IBM, ovvero 8192. Selezionando i risultati laddove il bit di destra è 0, abbiamo praticamente effettuato la misura condizionale necessaria al funzionamento dell'algoritmo. Nel riquadro 5.5 sono presentati i risultati filtrati in modo da considerare solo i risultati in cui il bit ancilla è 0.

Il passo successivo è eseguire questi stessi circuiti quantistici su un vero computer quantistico. È stata scelta la macchina a 16 qubit ibmq_16_melbourne per ottenere le misure per il vettore d'input setosa illustrate in figura 5.6. Gli inevitabili fenomeni di rumore rendono i



(a) Iris setosa

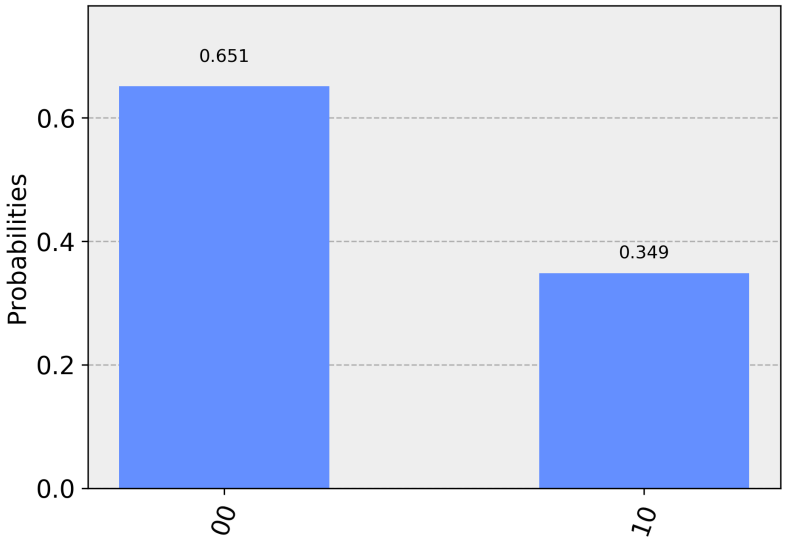


(b) Iris versicolor

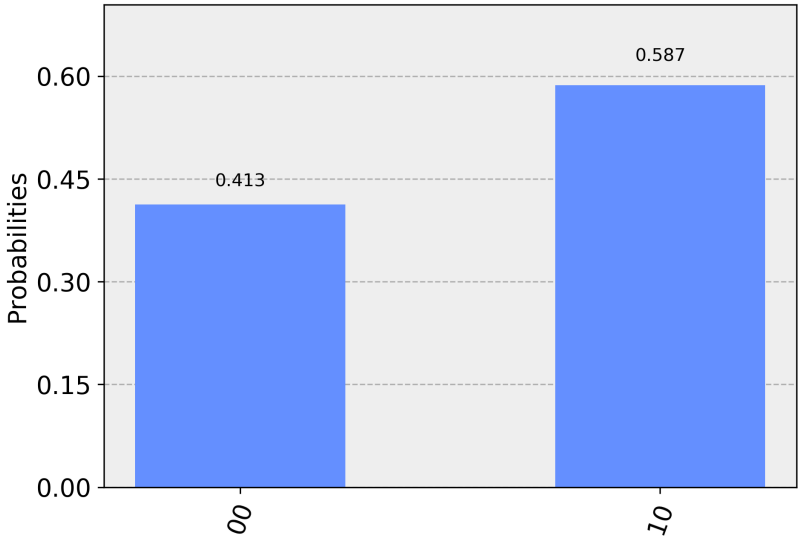
Figura 5.4: Simulazione del circuito.
I conteggi totali sono:
per setosa {'00': 3843, '10': 2056, '01': 352, '11': 1941};
per versicolor {'00': 2749, '10': 3908, '01': 1414, '11': 121}.

risultati meno piccati ma comunque distinguibili in questo caso.

Il discorso è diverso per il confronto tra le classi versicolor e virginica: visto che gli elementi di queste due classi non sono linearmente separabili, algoritmi come quello qui usato non sono efficaci nella



(a) Iris setosa



(b) Iris versicolor

Figura 5.5: Simulazione del circuito, risultati filtrati

loro distinzione. Uno dei metodi per aggirare questo problema è l'uso di feature map [9] nel processo di ottimizzazione. Infatti le misure effettuate danno risultati erronei o ambigui nella maggioranza dei casi (probabilità del 50% per entrambi i risultati).

Ad ogni modo, si procederà nel realizzare i punti definiti all'inizio di questo capitolo. Condizione necessaria per distinguere le tre classi con un solo esperimento è che si possano memorizzare almeno tre

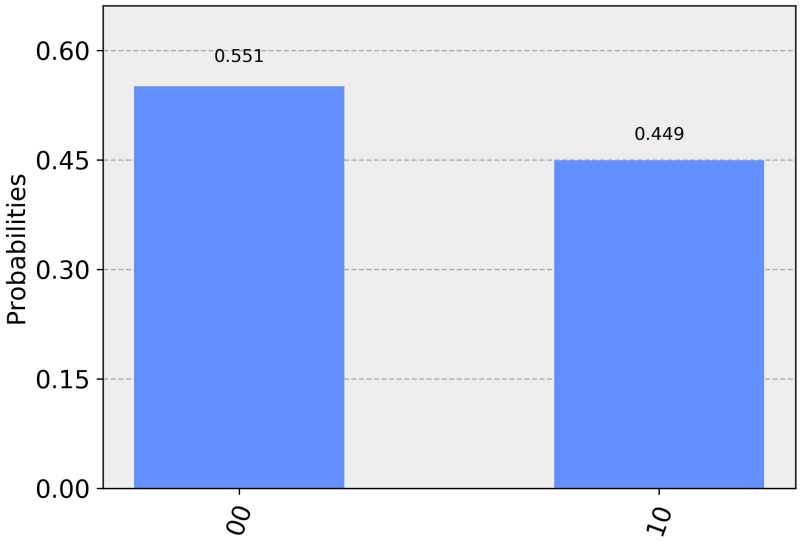


Figura 5.6: Esecuzione su hardware reale (setosa)

vettori di training, uno per ciascuna classe.

5.2.2 Aumento del numero di vettori d'apprendimento

Il numero di stati posseduti da n qubit, come abbiamo visto nella sezione sui fondamenti teorici, è 2^n. Nell'esperimento visto poco prima, si è dedicato un solo qubit per segnare l'indice dei vettori di apprendimento $|m\rangle$. Aggiungendo ulteriori qubit all'indice possiamo immagazzinare successivamente 4, 8, 16, 32, 64, ... vettori di training. Con 8 qubit per il registro $|m\rangle$ già siamo in grado di immagazzinare tutti i vettori del data set Iris. Nel caso di grandi insiemi, tuttavia, potrebbe essere appropriato ridurre il numero di vettori di training durante il processo di ottimizzazione, adottando un criterio di selezione che includa solo i vettori più significativi ai fini della classificazione.

La spesa totale in termini di risorse coinvolge due qubit per ogni incremento che facciamo: infatti, per mettere in entanglement un qubit in più del registro $|m\rangle$ con il qubit registro della QRAM, abbiamo bisogno di un qubit ancilla aggiuntivo per l'operazione $C^nR_y(\theta)$; questi qubit ancilla non sono rappresentati nei disegni dei circuiti per semplice convenienza visiva, ma se ne può trovare traccia nei frammenti di codice scritti per questa tesi[2].

2 Si veda https://github.com/visika/Tesi

5.2.3 Implementazione multiclasse

L'aggiunta della capacità di riconoscere tutte le tre classi in una sola esecuzione non è di natura differente dall'aggiungere qubit per avere maggiori vettori di apprendimento. Passiamo dall'avere un solo qubit classe, che ha associato il proprio stato $|0\rangle$ alla prima classe e lo stato $|1\rangle$ alla seconda, ad avere due qubit classe ed associare lo stato $|00\rangle$ alla prima classe, lo stato $|01\rangle$ alla seconda classe e lo stato $|10\rangle$ alla terza classe. Lo stato $|11\rangle$ non è associato ad alcuna classe. Nel processo di costruzione dello stato iniziale nella QRAM, oltre che ai qubit indice, i vettori di addestramento saranno in entanglement anche con i relativi stati dei due qubit classe. Osservando la figura 5.7 si vede che le porte $\bar{c}X$ sui qubit classe sono applicate solo all'inizio e alla fine del processo di codifica di tutte le componenti di un vettore di training.

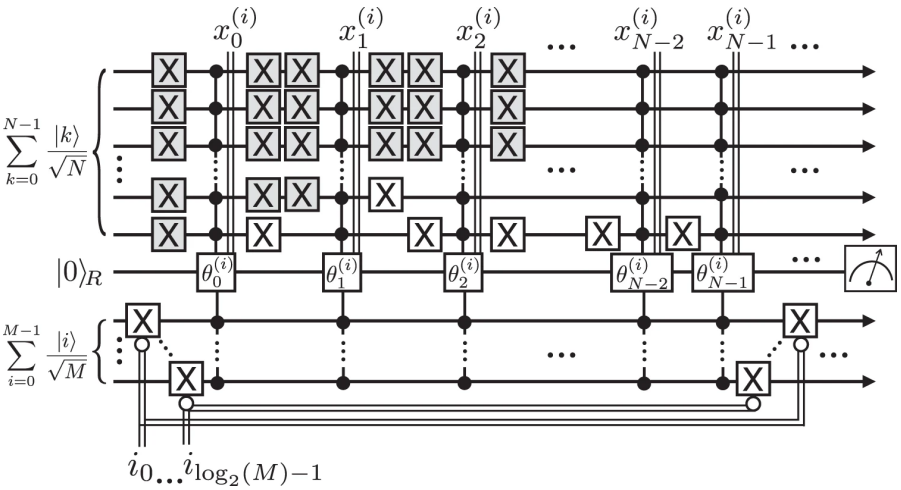


Figura 5.7: Circuito quantistico per preparare il QDB per un quantum support vector machine.
Sono mostrate le porte per scrivere solo l'i-esimo vettore di apprendimento. Le porte ombreggiate in grigio sono aggiunte esclusivamente per illustrare il processo flip-flop, e non sono implementate in pratica.

5.2.4 Aumento del numero di caratteristiche

L'Iris data set possiede 4 caratteristiche. È poco dispendioso includerle tutte dunque usando semplicemente 2 qubit $|i\rangle$ e 2 qubit come ancilla della QRAM. Per verificare il miglioramento apportato da un aumento del numero di caratteristiche considerate, si prende in esame la classificazione del vettore d'input 54 (versicolor), con i vettori di training numero 51 (versicolor) e 146 (virginica) del data set. Il vettore d'input viene classificato correttamente durante la simulazione con due caratteristiche, con probabilità vicina al 51.4%. Effettuando la stessa misura, ma tenendo conto di tutte le quattro feature, arriviamo

ad una probabilità di classificazione corretta del 58.3% nel migliore dei casi. Sembra esserci un margine di miglioramento in determinati casi, ma non può essere garantito in maniera generale.

5.3 ESECUZIONE COMPLETA

Per studiare l'efficienza dell'algoritmo a diversi stadi di miglioramento, si divide il data set in un insieme dedicato all'addestramento ed un insieme di vettori da classificare. Al fine di avere risultati imparziali, per ogni esecuzione i vettori di training ed il vettore d'input sono scelti casualmente a partire dal data set completo. Si contano le classificazioni di successo rispetto al totale dei tentativi, al variare dei parametri come il numero di features o il numero di vettori di training usati.

Provando ad effettuare una classificazione a tre classi con 32 vettori di training otteniamo i seguenti esiti:

- i vettori della classe setosa sono correttamente classificati 10 volte su 10;
- i vettori della classe versicolor sono correttamente classificati 5 volte su 10;
- i vettori della classe virginica sono correttamente clasificati 9 volte su 10.

I risultati vengono necessariamente da simulazioni, in quanto sono necessari 19 qubit sotto queste condizioni.

Volendo sfruttare appieno le potenzialità del simulatore presso l'IBM, possiamo costruire un circuito che accetti $2^7 = 128$ vettori di training presi a caso, e lasciare i restanti vettori per la classificazione. Con l'uso di 23 qubit si arriva alla classificazione corretta della classe versicolor 8 volte su 10.

6 CONCLUSIONE

Partendo da un semplice classificatore KNN classico si è riusciti ad implementarne la versione quantistica proposta da Schuld et al. [9]. Dopo di che si è tentato di estenderne le funzioni, con l'intento principale di rendere il classificatore capace di distinguere la classe corretta tra tutte quelle dell'insieme dati. È stata usata la tecnica di costruzione quantum database arbitrari FF-QRAM proposta da Petruccione et al. [6].L'algoritmo ha avuto un discreto successo e fornisce una distribuzione di previsioni che conferma le speranze in una applicazione anche a livello commerciale di tecniche di machine learning quantistico.

Per questioni di limitatezza di tempo, non è stato possibile sottoporre ad analisi anche altri data set o implementare tecniche più raffinate di preprocessing dei dati. Il lettore interessato ad approfondire la materia in maniera più sistematica si può affidare a lavori analoghi facilmente reperibili online,[1] o ai tanti articoli sull'argomento che escono ogni anno in libera consultazione.

Lavorare a questa tesi mi ha permesso di esplorare le terre, a me precedentemente sconosciute, del machine learning e del quantum computing. Nonostante il livello di approfondimento delle materie raggiunto in questi mesi di lavoro sia ridicolo, mi ritengo soddisfatto di essere riuscito a mettere a frutto le mie pregresse conoscenze di informatica per partecipare a qualcosa di estremamente innovativo. La critica che faccio a me stesso è di avere iniziato a scrivere ben troppo tardi il testo della tesi; se dovessi ripetere questa esperienza, cercherei di fissare bene fin dall'inizio gli obiettivi di ricerca e tenere traccia dei progressi in maniera più precisa, aggiornando in corso d'opera il documento finale.

1 Si veda https://github.com/carstenblank/dc-qiskit-qml.

Parte III

APPENDICE

A APPENDICE

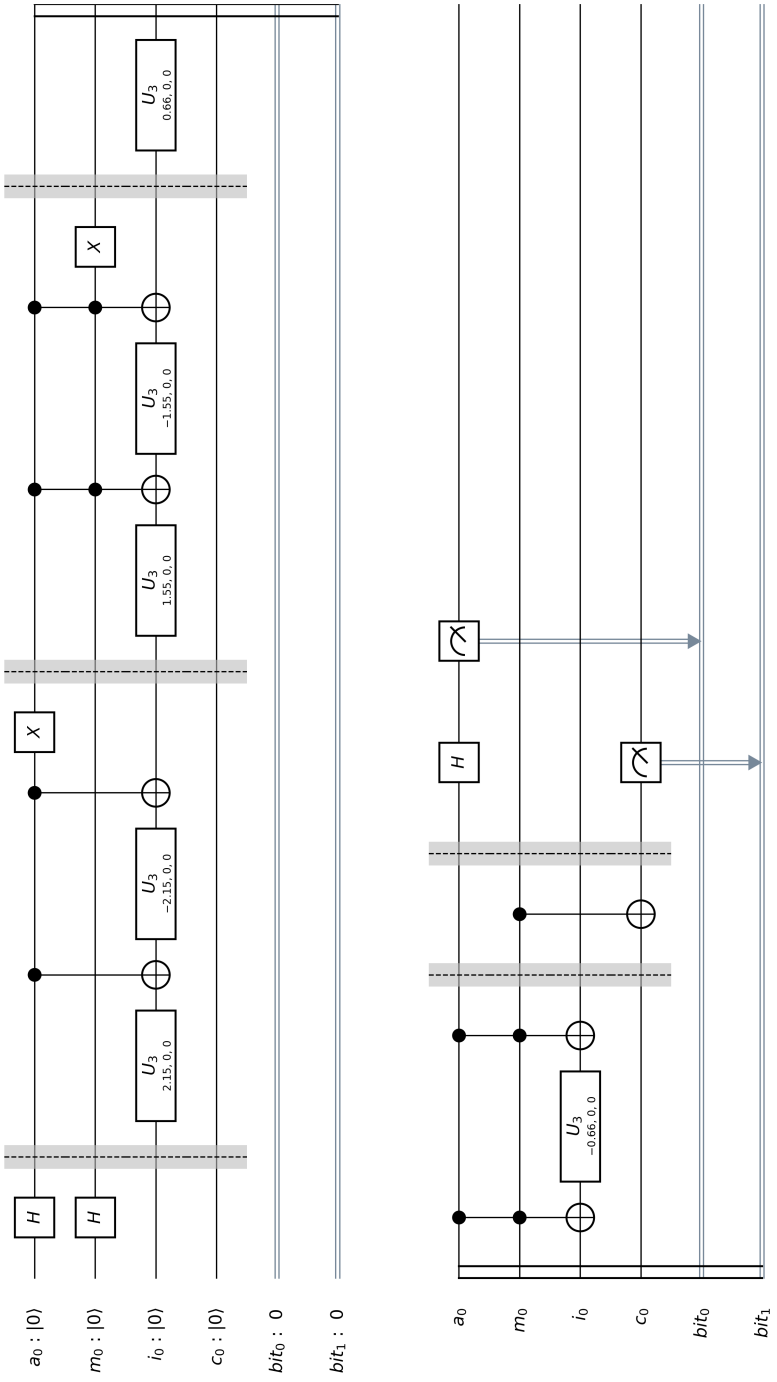


Figura A.1: Il circuito quantistico

BIBLIOGRAFIA

[1] *Android 10*. https://www.android.com/android-10/. [Online; recuperato il 7 settembre 2019]. 2019.

[2] Mark Fingerhuth. «Quantum-enhanced machine learning: Implementing a quantum k-nearest neighbour algorithm». In: *Bachelor Thesis, Maastricht University* (2017).

[3] Ralph Jacobson. «2.5 quintillion bytes of data created every day. How does CPG & Retail manage it?» In: *IBM Consumer Products Industry Blog* (2013). Recuperato il 6 settembre 2019. URL: https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/.

[4] Will Knight. *IBM Raises the Bar with a 50-Qubit Quantum Computer*. https://www.technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/. [Online; recuperato il 7 settembre 2019].

[5] U. Las Heras, U. Alvarez-Rodriguez, E. Solano e M. Sanz. «Genetic Algorithms for Digital Quantum Simulations». In: *Phys. Rev. Lett.* 116 (23 2016), p. 230504. DOI: 10.1103/PhysRevLett.116.230504. URL: https://link.aps.org/doi/10.1103/PhysRevLett.116.230504.

[6] Daniel K. Park, Francesco Petruccione e June-Koo Kevin Rhee. «Circuit-Based Quantum Random Access Memory for Classical Data». In: *Scientific Reports* 9.3949 (2019). DOI: 10.1038/s41598-019-40439-3.

[7] *Quantum Computing at IBM*. https://www.ibm.com/quantum-computing/learn/what-is-ibm-q. [Online; recuperato il 7 settembre 2019].

[8] Sebastian Raschka e Vahid Mirjalili. *Python Machine Learning*. 2ª ed. Packt, 2017. ISBN: 978-1-78712-593-3.

[9] M. Schuld, M. Fingerhuth e F. Petruccione. «Implementing a distance-based classifier with a quantum interference circuit». In: *EPL (Europhysics Letters)* 119.6 (2017). DOI: 10.1209/0295-5075/119/60002.

[10] Maria Schuld, Ilya Sinayskiy e Francesco Petruccione. «Quantum Computing for Pattern Classification». In: *PRICAI 2014: Trends in Artificial Intelligence*. A cura di Duc-Nghia Pham e Seong-Bae Park. Cham: Springer International Publishing, 2014, pp. 208–220. ISBN: 978-3-319-13560-1.

[11] Xiao-Feng Shi. «Deutsch, Toffoli, and cnot Gates via Rydberg Blockade of Neutral Atoms». In: *Phys. Rev. Applied* 9 (5 2018), p. 051001. DOI: 10.1103/PhysRevApplied.9.051001. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.9.051001.

[12] P. W. Shor. «Algorithms for quantum computation: Discrete logarithms and factoring». In: *Proc. 35nd Annual Symposium on Foundations of Computer Science (Shafi Goldwasser, ed.), IEEE Computer Society Press* (1994), pp. 124–134.

[13] Francesco Tacchino, Chiara Macchiavello, Dario Gerace e Daniele Bajoni. «An artificial neuron implemented on an actual quantum processor». In: *npj Quantum Information* (2019). URL: https://doi.org/10.1038/s41534-019-0140-4.

[14] Kristan Temme e Jay Gambetta. *Researchers Put Machine Learning on Path to Quantum Advantage*. https://www.ibm.com/blogs/research/2019/03/machine-learning-quantum-advantage/. [Online; recuperato il 7 settembre 2019]. 2019.

[15] Noson S. Yanofsky e Mirco A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2013.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Thank you very much for your feedback and contribution.