MARIANO MOLLO

IMPLEMENTAZIONE DI UN ALGORITMO KNN MULTICLASSE SU HARDWARE QUANTISTICO

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 Sperimentale 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

[23 ottobre 2019 at 12:31 – classicthesis v4.6]

SOMMARIO

Sempre più tecnologie contemporanee si affidano a tecniche di machine learning per migliorare la propria efficienza, dall'industria produttiva all'assistente vocale sul cellulare, ai sistemi di guida autonoma. Al crescere della complessità e della quantià di dati, l'implementazione di queste tecniche diventa più ostica ed è necessario sviluppare strategie che permettano di eseguire gli algoritmi di machine learning in maniera efficiente. Il quantum machine learning impiega le proprietà di parallelizzazione del calcolo e di compressione delle informazioni dei computer quantistici per risolvere in maniera più efficiente problemi classici difficili e per affrontare classi di problemi completamente nuove. In questa tesi si implementa un riconoscitore KNN quantistico a due classi attraverso il kit di sviluppo software Qiskit e la piattaforma online IBM Q Experience, e se ne generalizza il funzionamento per renderlo multiclasse.

ABSTRACT

More and more present technologies rely heavily on Machine Learning techniques to improve their efficiency, ranging from production industry to voice assistants on smartphones, to online recommendation algorithms. As we discover new complexity classes and deal with bigger quantities of data, implementing these techniques becomes harder and it's mandatory to develop strategies to run Machine Learning algorithms more efficiently. Quantum Machine Learning employs computation parallelization and information compression properties of Quantum Computers to solve more efficiently hard classical problems or completely new classes of problems. In this thesis a two classes quantum KNN classifier is implemented, thanks to the Qiskit software development kit and the online platform IBM Q Experience, and its functioning is generalized to make it multiclass.

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 propri 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

IA intelligenza artificiale

MLQ machine learning quantistico

QML quantum machine learning

QRAM quantum random access memory

KNN k-nearest neighbours

QKNN quantum k-nearest neighbours

FF-QRAM flip-flop QRAM

QDB quantum database

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 continua durante tutta la loro vita. Sebbene tale processo sembri automatico, in realtà è espressione di un complesso sistema di feedback, che 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 il prodotto da acquistare o il video da guardare più affine ai nostri gusti, imparando dall'esperienza; ma sono impiegati 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 [11].

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. [3]

Secondo l'IBM [6], 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 possano 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 fondamentale rispetto a questo problema che può avere l'uso della computazione quantistica, oggetto da vari decenni di intensa ricerca teorica, e più recentemente anche di ricerca sperimentale [15] e tesi di laurea [5].

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. 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 stato defini-

1 Per informazioni sui bit classici probabilitici si faccia riferimento alla sezione 3.2 di [21].

to 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 miglioramenti esponenziali nel numero di operazioni se confrontati con le loro controparti classiche; l'algoritmo di fattorizzazione in numeri primi di Shor è uno tra i più famosi [18].

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)) [4] , 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. Cionondimeno, entrambi i campi sono benefici l'un l'altro, poiché il ML classico può anche essere usato per migliorare aspetti della computazione quantistica [8]. Questa tesi si occuperà solo del campo del machine learning migliorato 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. 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 superconduttivi, da quanto viene riferito, è di 50 [7].

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. [10]

Negli ultimi anni ci sono state varie implementazioni innovative di algoritmi di MLQ [20]: seguendo i passi che hanno portato alla nascita del ML nel XX secolo, Tacchino et al. [19] hanno implementato un percettrone su hardware quantistico; Schuld et al. [15] hanno invece implementato l'algoritmo lazy learner KNN. Questa tesi parte

proprio da quest'ultimo articolo, provando a riprodurne il lavoro e ad implementarne una versione multiclasse.

1.2 OBIETTIVO DI RICERCA

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

> 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?

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

2 QUANTUM COMPUTING E MACHINE LEARNING

Nella sezione 2.1 si discuteranno i concetti fondamentali su cui si base il funzionamento di un quantum computer. 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 discussa in sezione 2.1.1. Nella sezione 2.1.2 si presentano le porte logiche quantistiche, che permettono di manipolare i qubit. Infine, in sezione 2.1.3 si discuterà di uno dei nuovi problemi con cui si deve avere a che fare quando si lavora con oggetti quantomeccanici, ovvero la decoerenza.

Successivamente, in sezione 2.2, verrà introdotto brevemente il campo del machine learning, spiegando il funzionamento dell'algoritmo k-nearest neighbours, per poi discuterne la versione quantistica in sezione 2.3.

2.1 QUANTUM COMPUTING

2.1.1 Bit e qubit

Definizione 1. Un bit è un'unità di informazione che descrive un sistema classico bidimensionale.

I due possibili stati di un bit 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 che il bit che può trovarsi in uno stato tra $|0\rangle$ e $|1\rangle$.

Definizione 2. 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 questo lavoro.

Gli stati $|0\rangle$ 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.

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

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

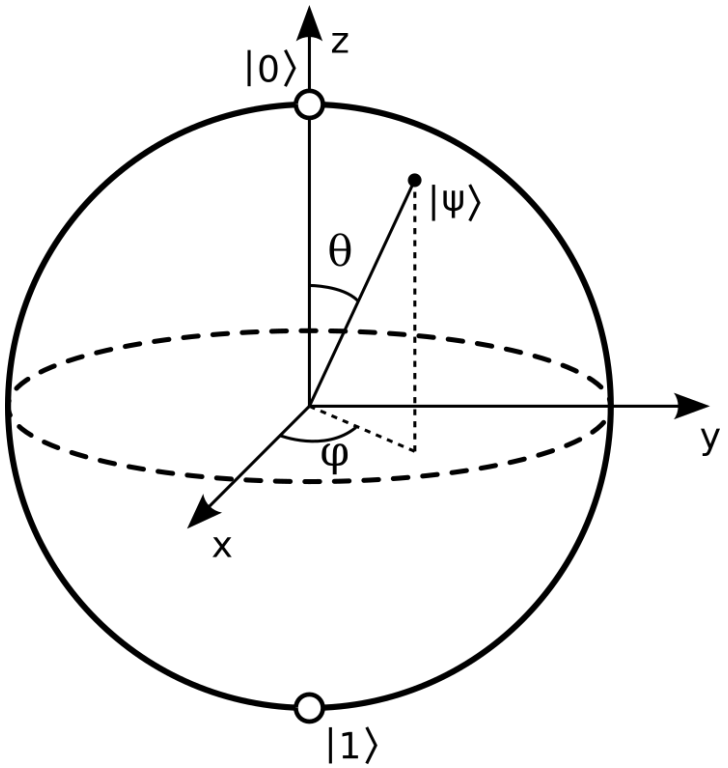


Figura 2.1: La sfera di Bloch

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

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

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

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

caso quantistico, uno stato di otto qubit è dato scrivendo 256 numeri complessi. Questa crescita esponenziale della capacità di immagazzinamento dei dati è una delle ragioni per cui i ricercatori hanno manifestato interesse per il concetto di quantum computing.

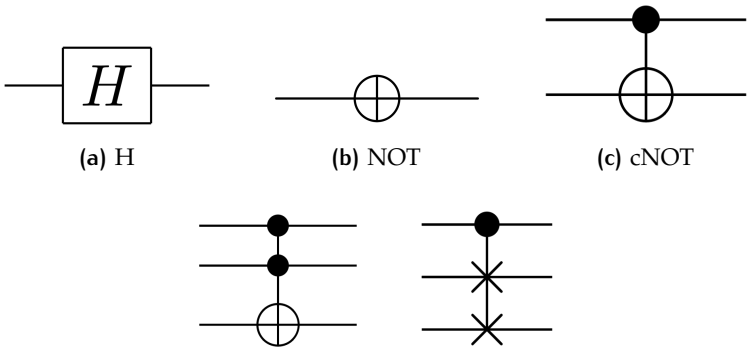
2.1.2 Porte logiche quantistiche

Definizione 3. 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 o inversioni 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 $2^n \times 2^n$.

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:

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

Altre importanti matrici usate spesso sono

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

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 noti che X non è altro che la porta NOT, e trasforma $|0\rangle$ in $|1\rangle$ e viceversa. Ma per di più porta tutto quello che si trova sopra l'equatore nella sfera di Bloch 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:

$$R_D(\theta) = \cos\frac{\theta}{2}\mathbb{1} - i\sin\frac{\theta}{2}(D_x X + D_y Y + D_z Z). \tag{2.10}$$

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 di fase applicata a 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 $C^n R_y(\theta)$, che effettua una rotazione di angolo θ attorno all'asse y, con n qubit di controllo.

Le due porte appena descritte non si trovano generalmente nel set universale di porte del computer quantistico, 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 [17] è uno degli sforzi per contrastare il fenomeno della decoerenza, uno degli ostacoli sostanziali nel calcolo quantistico.

2.1.3 Decoerenza

Definizione 4. 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 [13]:

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

2.2 MACHINE LEARNING CLASSICO

Definizione 5. Il machine learning, una branca dell'intelligenza artificiale (IA), è l'applicazione e lo studio di algoritmi che danno un significato ai dati.

La materia è divisa in tre sottocategorie principali: l'apprendimento supervisionato, l'apprendimento non supervisionato e l'apprendimento per rinforzo. Qui si parlerà solo di apprendimento supervisionato.

L'obiettivo dell'apprendimento supervisionato è di imparare un modello da dati etichettati (detti anche di training) che ci permetta di fare previsioni su dati futuri sconosciuti. Con il termine supervisionato si intende l'esistenza di un insieme di campioni dove i risultati aspettati (l'appartenenza ad una determinata classe, il risultato di una funzione) sono già conosciuti.

Un esempio di classificatore supervisionato è il filtro contro la posta indesiderata, che si può addestrare con un insieme di email già classificate come spam o non spam, in modo che esso riconosca in quale categoria vanno le nuove email in arrivo.

La classificazione è una sottocategoria del machine learning supervisionato il cui scopo è di predire le etichette di classe che categorizzano delle nuove istanze, basandosi su osservazioni passate. Tali classi sono valori discreti non ordinati che possono rappresentare l'appartenenza ad un gruppo delle istanze.

2.2.1 KNN

Il k-nearest neighbours (KNN) è un algoritmo di machine learning supervisionato per la classificazione. È chiamato lazy learner, in quanto non apprende una regola su come discriminare le classi dei vettori, ma memorizza il data set di apprendimento tutto intero. L'algoritmo in sé è piuttosto semplice e può essere riassunto nei 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.

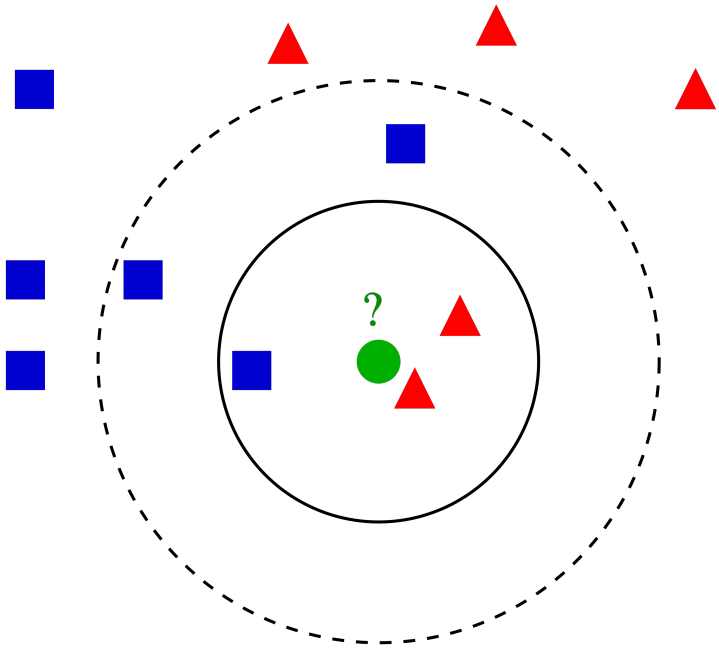


Figura 2.3: Rappresentazione dell'algoritmo KNN

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 basato sulla classe a cui appartengono i suoi k vicini (vedi figura 2.3).

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 a priori alcun vettore di training dato che non c'è un vero e proprio apprendimento. Così, lo spazio di archiviazione e il numero di distanze da calcolare possono diventare un problema nodale quando si lavora con grandi data set. [11]

2.3 MACHINE LEARNING QUANTISTICO

Il machine learning quantistico è la materia che unisce machine learning e meccanica quantistica. Bimonte et al. [4] scrivono:

> la meccanica quantistica segue notoriamente modelli atipici. I metodi di machine learning classico come le reti neurali profonde (deep neural network) frequentemente hanno la caratteristica di poter sia riconoscere modelli statistici sia produrre dati che ne possiedano gli stessi andamenti: riconoscono i motivi che producono. Questa osservazione

porta alla seguente speranza. Se piccoli processori di informazioni quantistiche possono produrre motivi statistici che sono computazionalmente difficili da produrre per un computer classico, allora forse essi possono anche riconoscere motivi che sono ugualmente difficili da riconoscere classicamente. La realizzazione di questa speranza dipende dal fatto che si possano trovare algoritmi quantistici efficienti per il machine learning. Un algoritmo quantistico è un insieme di istruzioni che risolvono un problema, come determinare se due grafici sono isomorfi, che possono essere eseguite su un computer quantistico. Il software del quantum machine learning fa uso degli algoritmi quantistici come parte di un'implementazione maggiore. Analizzando i passi prescritti dagli algoritmi quantistici, diventa chiaro che hanno il potenziale di superare in prestazioni gli algoritmi classici per problemi specifici (cioè, ridurre il numero di passi necessari). Questo potenziale è conosciuto come accelerazione quantistica [14].

Per esempio, i computer quantistici possono cercare in una banca dati non ordinata con N voci in un tempo proporzionale a $\sqrt{N}$, ovvero $O(\sqrt{N})$, mentre un computer classico con accesso a scatola nera alla stessa banca dati impiega un tempo proporzionale ad N: dunque il computer quantistico esibisce un'accelerazione che va come la radice quadrata rispetto al computer classico.

Nel quantum machine learning il computer quantistico analizza dati classici, che sono codificati come stati quantistici, oppure dati quantistici. Sempre afferente allo stesso campo è l'uso di tecniche di machine learning classico per trovare modelli in dinamica quantistica. Ci si limiterà al caso dell'uso di computer quantistici per analizzare dati classici.

Nella sezione seguente si descriverà la versione quantistica dell'algoritmo KNN visto nella sezione 2.2.1, proposta ed implementata sul processore quantistico a 5 qubit dell'IBM da Schuld et al. [16], [15].

2.3.1 QKNN

L'idea alla base dell'algoritmo quantum k-nearest neighbours (QKNN) è di usare il fenomeno di interferenza quantistica per ottenere una misura della distanza con un calcolo in parallelo. Dotandosi di una procedura efficiente per la preparazione dello stato, l'algoritmo ottiene la riduzione logaritmica nelle dimensioni e nel numero di dati in input che ci si aspetta usando una codifica quantistica dei dati classici [12]. Si considera l'attività di classificazione supervisionata di un motivo binario: dato un data set $D = \{(t^1, c^1), \ldots, (t^M, c^M)\}$ di input $t^m \in \mathbb{R}^N$ con le rispettive etichette di classe $c^m \in \{-1, 1\}$ per $m = 1, \ldots, M$ e un nuovo input $x \in \mathbb{R}^N$, si trovi l'etichetta $c \in \{-1, 1\}$ che corrisponde al nuovo input.

Il classificatore, implementato tramite un circuito di interferenza quantistica ed una funzione di soglia, è dato da

$$c = \operatorname{sgn}\left(\sum_{m=1}^{M} c^m \left[1 - \frac{1}{4M}|x - t^m|^2\right]\right). \tag{2.12}$$

L'algoritmo che implementa il classificatore codifica le caratteristiche dell'input nelle ampiezze di probabilità del sistema quantistico e le manipola attraverso porte quantistiche; questa strategia è uno dei fattori ritenuti responsabili dell'accelerazione quantistica. Dato un vettore $x \in \mathbb{R}^N$ normalizzato con modulo unitario e $N = 2^n$, la codifica nelle ampiezze permette di descrivere lo stato con un registro di n qubit $|\psi_x\rangle = \sum_{i=0}^{N-1} x_i |i\rangle$, dove $|i\rangle$ è un registro indice che segnala l'i-esima componente del vettore classico attraverso l'i-esimo elemento della base computazionale.

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 (nella sezione 4.1.1 se ne vedrà un esempio pratico), il circuito di classificazione porta un sistema di n qubit nello stato

$$|D\rangle = \frac{1}{\sqrt{2MC}} \sum_{m=1}^{M} |m\rangle \left(|0\rangle_a |\psi_x\rangle_i + |1\rangle_a |\psi_{t^m}\rangle_i\right) |c^m\rangle \,. \tag{2.13}$$

Qui il primo registro $|m\rangle$ è un registro indice che prende valori $m = 1, \ldots, M$ e che contrassegna l'm-esimo vettore di apprendimento. Il secondo registro $|a\rangle$ è formato da un singolo qubit ancilla, il cui stato eccitato è in entanglement con il terzo registro $|i\rangle$ 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:

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^{M} |m\rangle \left(|0\rangle |\psi_{x+t^m}\rangle + |1\rangle |\psi_{x-t^m}\rangle\right) |c^m\rangle \,, \tag{2.14}$$

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

$$\frac{1}{2\sqrt{M p_{acc}}} \sum_{m=1}^{M} \sum_{i=1}^{N} |m\rangle \left(x_i + t_i^m\right) |i\rangle |c^m\rangle \,. \tag{2.15}$$

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 $|s\rangle$

$$p(|c\rangle = |s\rangle) = \frac{1}{4Mp_{acc}} \sum_{m||c\rangle=|s\rangle} |x + t^m|^2, \tag{2.16}$$

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.

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 evoluto come strumento omnicomprensivo per una varietà sempre crescente di lavori di ricerca scientifica.

3.1 QISKIT

Qiskit [1] 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.

3.3 ESEMPIO DI ALGORITMO

Come rapido esempio dell'uso di Qiskit si propone il codice che riproduce l'algoritmo QKNN presentato nella sezione 2.3.1.

I valori x0, t0, t1 corrispondono agli angoli di rotazione per il vettore d'input e per i due vettori di training rispettivamente. La

Listing 3.1: Algoritmo per il QKNN

```
import qiskit.aqua.circuits.gates.controlled_ry_gates

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)

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

circuit.cry(x0,a[0],i[0])
circuit.x(a) # swap entanglement with ancilla to 0

circuit.mcry(t0,a[:]+m[:],i[0],None)
circuit.x(m) # swap entanglement with m index to 0

circuit.mcry(t1,a[:]+m[:],i[0],None)

circuit.cx(m,c) # entangle class 1 with m index 1

circuit.h(a)
circuit.measure(a,b[0])
circuit.measure(c,b[1])

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

A partire dalla versione 0.12 di Qiskit la porta mcry è inclusa nei pacchetti base e non c'è bisogno di importarla separatamente. (Effettivamente tentarci comporta un errore nel programma.)

porta mcry non si trovava nelle funzioni di base di Qiskit ma è stata importata separatamente attraverso il comando in cima. Il disegno del circuito è presente in appendice nella figura A.1: si può notare come le porte di rotazione controllata siano in realtà realizzate con una sequenza di porte di base più o meno complicata. Tale processo è gestito automaticamente da Qiskit.

4 IMPLEMENTAZIONE MULTICLASSE

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 [9].

4.1.1 Flip-flop QRAM

La FF-QRAM è usata per memorizzare 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 $D = \left\{ \left(\vec{d}^{(l)}, b_l \right) | 0 \leqslant l < M \right\}$, dove $\vec{d}^{(l)}$ rappresenta un indirizzo di memoria con n bit di informazione e b_l è l'attributo ad esso associato, come

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

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, $C^n R_p(\theta)$. La $\bar{c}X$ inverte il qubit bersaglio solo quando il bit classico di controllo è zero. La porta $C^n R_p(\theta)$ 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

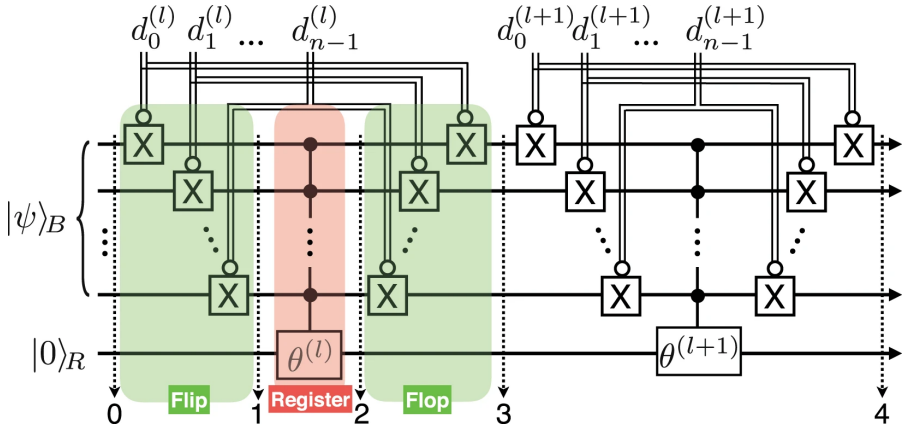


Figura 4.1: Procedimento di costruzione FF-QRAM [9]
Circuito quantistico per FF-QRAM che scrive le stringhe di bit $\vec{d}^{(l)}$ e $\vec{d}^{(l+1)}$ come una sovrapposizione di stato quantistico con ampiezze di probabilità determinate da $\theta^{(l)}$ e $\theta^{(l+1)}$ rispettivamente, usando porte 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 APPROCCIO MULTICLASSE

L'algoritmo QKNN presentato in sezione 2.3.1 è stato implementato su un computer a 5 qubit, di cui solo 4 sono stati usati. La codifica usata prevedeva che un vettore bidimensionale $x = (x_0, x_1)$ normalizzato fosse codificato nello stato del qubit $|i\rangle$ come $|\psi_x\rangle = x_0 |0\rangle + x_1 |1\rangle$ attraverso una rotazione controllata dai qubit $|a\rangle$ e $|m\rangle$. Si faccia riferimento alla figura 4.2. Parametrizzando l'angolo di Bloch attraverso la relazione

$$x_0 = \cos(\theta), \quad x_1 = \sin(\theta), \tag{4.9}$$

otteniamo la relazione tra i vettori d'input e l'angolo θ

$$\theta = \arctan\frac{x_1}{x_0}. \tag{4.10}$$

La relazione appena mostrata è una delle prime che verranno modificate per permettere l'uso della QRAM, quando si vorranno inserire vettori di dimensione maggiore di 2.

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

$$\sum_{l=0}^{M-1} \psi_{\vec{d}^{(l)}} |\vec{d}^{(l)}\rangle \sin\theta^{(l)} |1\rangle_R . \tag{4.11}$$

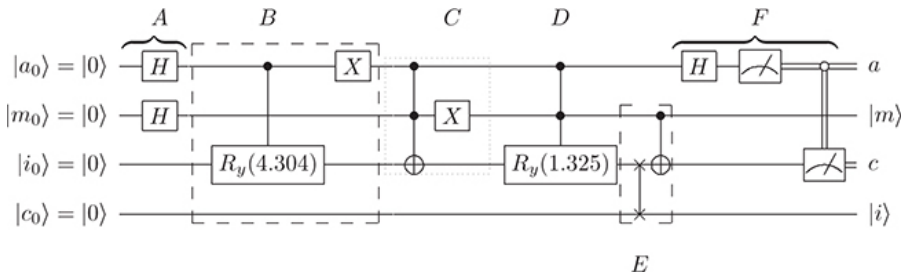


Figura 4.2: Il circuito quantistico base per il QKNN [15]

La relazione esistente tra i valori b_l, ovvero le componenti x_i, e le ampiezze di probabilità nella QRAM è allora

$$\theta = \arcsin x_i. \tag{4.12}$$

L'espansione delle capacità del circuito prevede dunque l'aumento dei qubit dedicati a ciascun registro quantistico tra i seguenti:

- al registro $|m\rangle$ per aumentare i vettori di apprendimento;
- al registro $|i\rangle$ per aumentare la dimensionalità dei vettori;
- al registro $|c\rangle$ per aumentare il numero di classi riconosciute.

Lo schema circuitale passa da quello mostrato in figura 4.2 a quello più generico mostrato in figura 4.3.

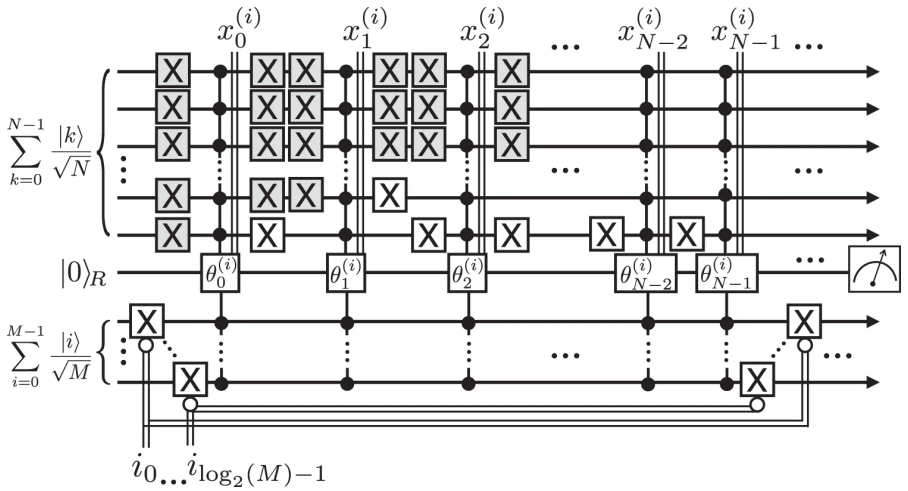


Figura 4.3: Circuito quantistico per preparare il QDB per un quantum support vector machine [9]
Sono mostrate le porte per scrivere solo l'i-esimo vettore di apprendimento. Le porte ombreggiate in grigio sono aggiunte esclusivamente per illustrare il processo flip-flop, e non sono implementate in pratica.

La codifica del vettore d'input prevede che i qubit di controllo siano solo quelli mostrati nella parte superiore della figura. Al contrario,

quando si codificano i vettori di apprendimento, li si pone in entanglement anche con i qubit indice $|m\rangle$ e classe $|c\rangle$, applicando le opportune porte $\overline{c}X$ all'inizio ed alla fine della loro codifica.

Adottando una configurazione di esempio con 2 qubit indice, 2 qubit caratteristica e 2 qubit classe si ottiene un classificatore capace di confrontare un vettore di input con 4 vettori di apprendimento aventi 4 caratteristiche ed appartenenti a 4 classi. Si riporta il codice relativo a tale realizzazione nel riquadro 4.1.

Listing 4.1: Algoritmo per il QKNN multiclasse

```
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)

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

# circuit.cry(theta, control, target)
# circuit.mcry(theta, controls, target, ancillae)

# >>> Encode the input vector >>>

encodeVector(circuit,inputVirginica,i,a[:]+i[:],r[0],q)

circuit.x(a) # swap entanglement with ancilla to 0

# <<< Encode the input vector <<<

# >>> Encode the training vectors >>>

buildTrainingState(trainingArray)

# <<< Encode the training vectors <<<

circuit.measure(r,b[0])

circuit.h(a)

circuit.measure(a,b[1])
circuit.measure(c[0],b[2])
circuit.measure(c[1],b[3])
```

La funzione `buildTrainingState` prende i vettori appartenenti a `trainingArray` e ne codifica i valori nelle ampiezze dei qubit attraverso la funzione mostrata nel riquadro 4.2.

Listing 4.2: Funzione per codificare i vettori

```
def encodeTraining(circuit,data,i,controls,rotationQubit,
    ancillaQubits,c,m):
# Header
encodeClass(circuit,c)
encodeIndex(circuit,m)

# Encoder
encodeVector(circuit,data,i,controls,rotationQubit,
    ancillaQubits)

# Footer
encodeClass(circuit,c)
encodeIndex(circuit,m)
```

Le funzioni nell'header e nel footer altro non fanno che applicare le giuste porte X, come descritto in sezione 4.1.1 e in figura 4.3. Più nello specifico, la funzione `encodeVector` applica le porte della parte superiore dell'immagine ed effettua le rotazioni controllate, mentre `encodeClass` e `encodeIndex` applicano quelle della parte inferiore[1].

Nel capitolo successivo si discutono i risultati di classificazione ottenuti in caso di simulazione ed esecuzione su hardware quantistico.

1 Per accedere al codice completo, fare riferimento alla cartella Script nella repository su GitHub https://github.com/visika/Tesi

5 RISULTATI E DISCUSSIONE

Per l'implementazione dell'algoritmo è stato scelto il ben noto data set Iris, una tabella compilata con lunghezze e larghezze del petalo e del sepalo di tre specie di fiore Iris: setosa, versicolor e virginica. Come evidenziato in Schuld et al. [15], l'insieme dati si è dovuto standardizzare e normalizzare prima dell'utilizzo.

5.1 PREPARAZIONE DEI DATI

Il data set Iris completo si presenta nella forma mostrata in figura 5.1, dove sono graficate due delle quattro caratteristiche del data set sugli assi coordinati.

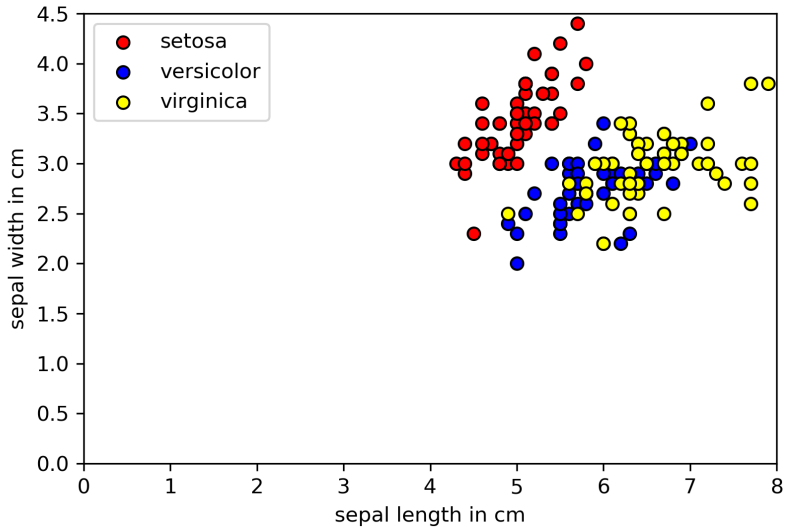


Figura 5.1: Data set Iris non elaborato

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.

La normalizzazione termina il processo di preparazione dei dati. Si può vedere l'effetto della normalizzazione su un data set con due o quattro caratteristiche nel riquadro 5.3.

Avendo effettuato queste operazioni, si traducono le coordinate di ciascun vettore nello spazio delle caratteristiche in un angolo di

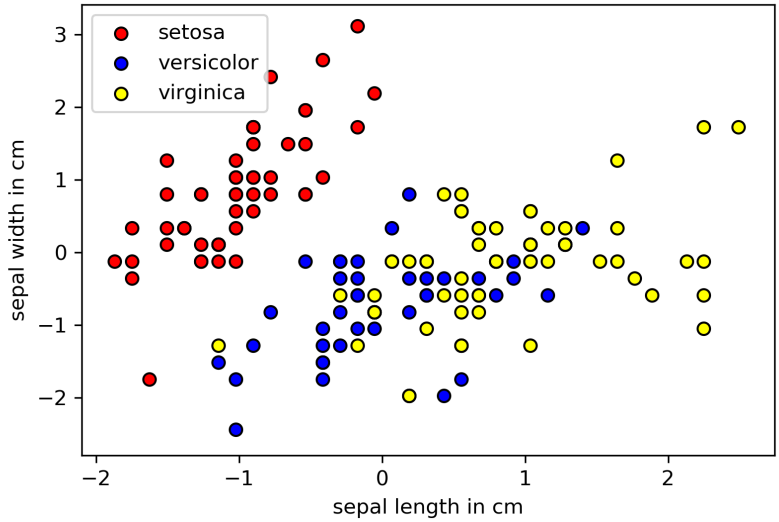
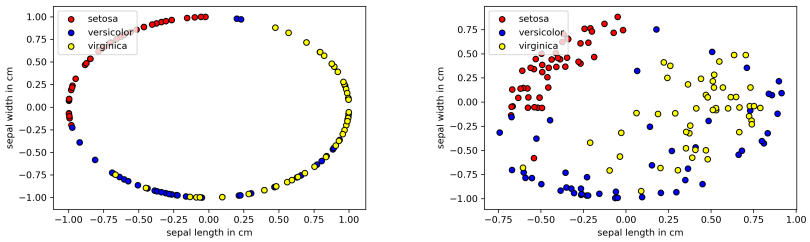


Figura 5.2: Data set Iris standardizzato



(a) 2 caratteristiche **(b)** 4 caratteristiche

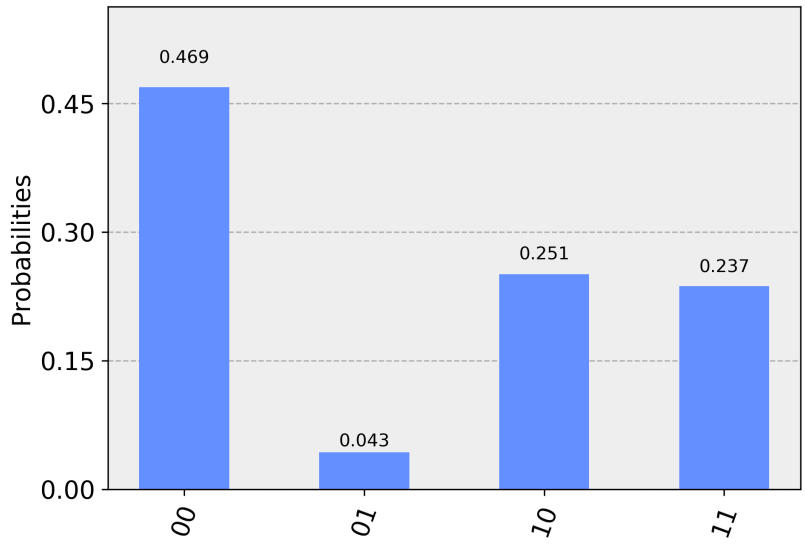
Figura 5.3: Data set Iris normalizzato

rotazione da applicare al qubit registro della QRAM, come descritto nell'eq. 4.12.

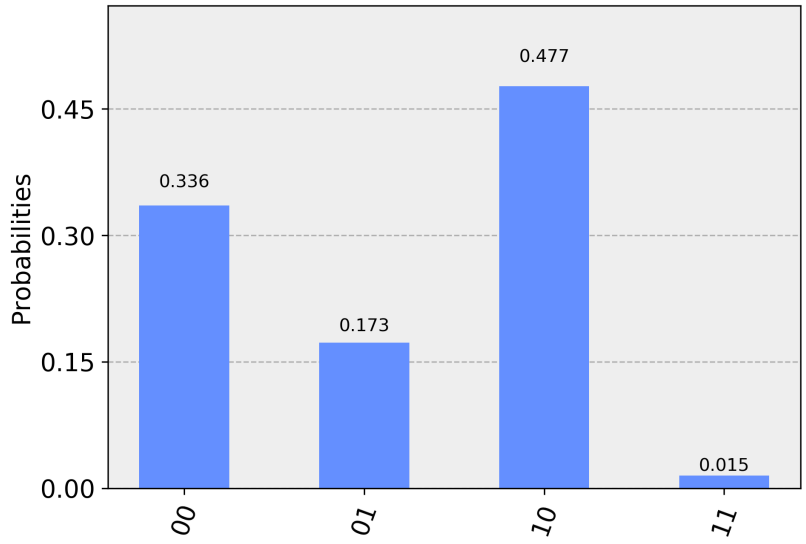
5.2 CLASSIFICAZIONE BASE

Per dare un esempio di classificazione binaria tra le classi setosa e versicolor si sono usate le prime due caratteristiche e si sono posti a confronto il vettore 29 (setosa) e 57 (versicolor) in input con i due vettori di training 34 (setosa) e 86 (versicolor). La classe setosa è stata collegata allo stato $|c = 0\rangle$, la classe versicolor allo stato $|c = 1\rangle$. I risultati ottenuti da una simulazione su computer portatile sono visibili in figura 5.4. Nella didascalia sono scritti i conteggi totali 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. L'istogramma riporta la normalizzazione

unitaria dei risultati, che approssima la distribuzione di probabilità per numeri grandi di esecuzioni dell'algoritmo. Si userà preferibilmente un numero di esecuzioni (shot) pari al massimo permesso dai computer quantistici dell'IBM, ovvero 8192.



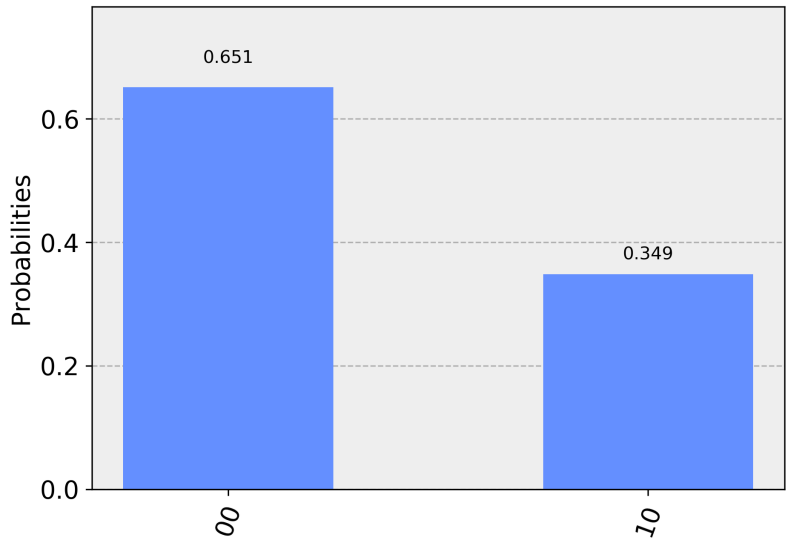
(a) Iris setosa



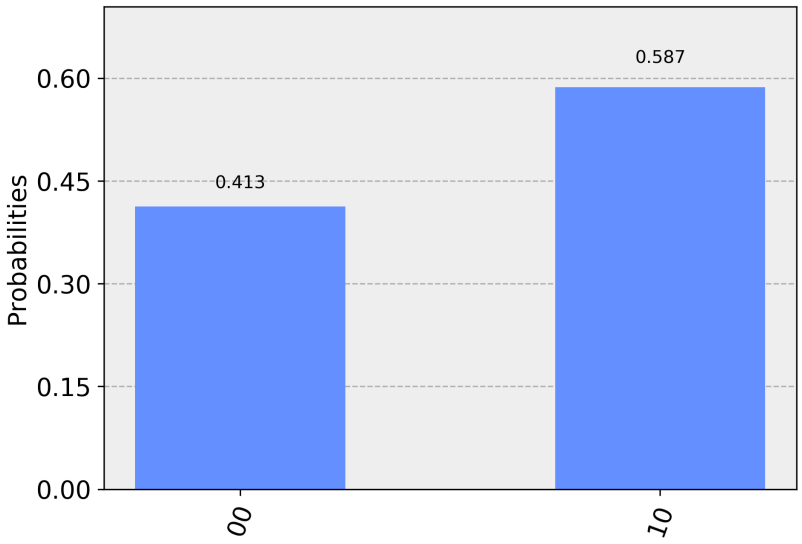
(b) Iris versicolor

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}.

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 dove il bit ancilla è 0.



(a) Iris setosa



(b) Iris versicolor

Figura 5.5: Simulazione del circuito, risultati filtrati

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 risultati meno piccati ma comunque distinguibili.

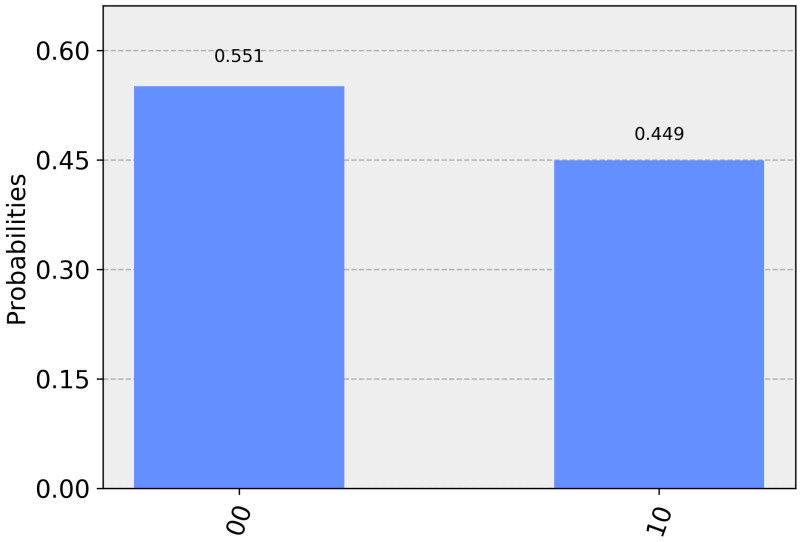


Figura 5.6: Esecuzione su hardware reale (setosa)

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 il KNN non sono efficaci nella loro distinzione. Infatti le misure effettuate danno risultati erronei o ambigui nella maggioranza dei casi (probabilità del 50% per entrambi i risultati). Uno dei metodi per aggirare questo problema è l'uso di feature map [15] nel processo di ottimizzazione.

5.3 LIMITI DI ESECUZIONE

Nell'esecuzione dell'algoritmo su hardware quantistico si è dovuto tenere conto dei limiti in termini di numero di qubit. Il computer quantistico disponibile tramite IBM Q Experience permette di usare al massimo 14 qubit superconduttivi. Considerando che, per ogni qubit di controllo aggiuntivo, la porta $C^nR_y(\theta)$ richiede un ulteriore qubit ancilla, la spesa in termini di risorse è di due qubit per ogni incremento tra numero di vettori di training, di classi o di caratteristiche. Alcune configurazioni ottimali per l'esecuzione con 14 qubit sono descritte in tabella 5.1. Si ricorda che dato un registro di n qubit, questo può conservare $N = 2^n$ valori nelle sue ampiezze di probabilità.

Un'esecuzione che sfrutti due qubit classe, tre qubit indice e un qubit caratteristica necessita di 5 qubit ancilla ed un qubit registro per la costruzione della QRAM, un qubit ancilla per il QKNN, per un totale

i	m	c
2	2	2
3	2	1
2	3	1
1	3	2

Tabella 5.1: Alcune configurazioni per i qubit

di 13 qubit. Una classificazione di questo tipo è stata eseguita sul processore quantistico ibm_16_melbourne: si sono confrontati vettori d'input casuali presi dalle tre classi con insiemi di apprendimento allo stesso modo costruiti casualmente, avendo cura di inserire in maniera uniforme elementi delle diverse classi. Con 3 qubit $|m\rangle$ abbiamo a disposizione 8 vettori di apprendimento; l'insieme di training possiede 2 vettori dalla classe setosa, 3 da versicolor e 3 da virginica. Si riportano in figura 5.7 i risultati (filtrati, come fatto precedentemente) di classificazione di un elemento della classe setosa; il circuito è stato eseguito 8192 volte e l'istogramma riporta la probabilità che il vettore d'input appartenga ad una determinata classe, come evidenziato nell'eq. 2.16. L'esito di classificazione è rappresentato dai due bit a sinistra, mentre i due bit a destra assicurano l'esecuzione delle misure condizionali $|a\rangle = |0\rangle$ e $|j\rangle_R = |1\rangle_R$. La classe setosa è legata al risultato di misura $|c\rangle = |00\rangle$, versicolor a $|c\rangle = |01\rangle$ e virginica a $|c\rangle = |10\rangle$; lo stato $|c\rangle = |11\rangle$ resta inutilizzato.

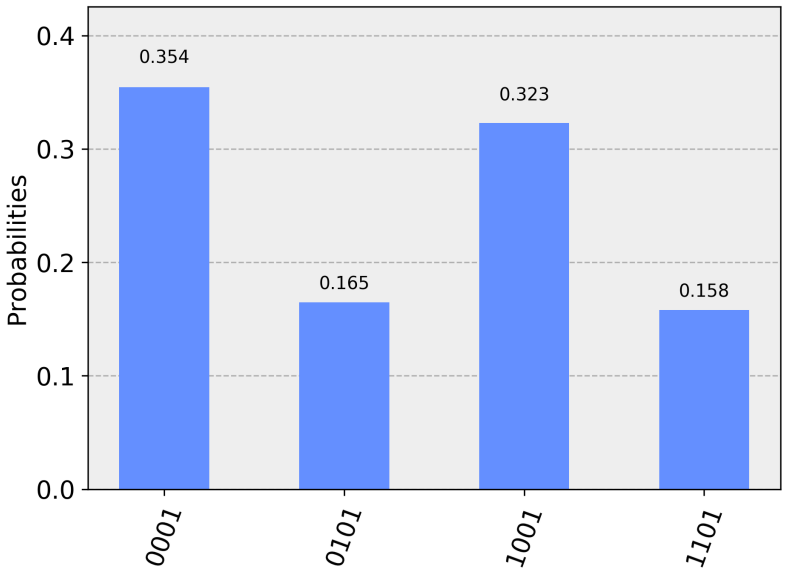


Figura 5.7: Risultati sperimentali multiclasse (setosa)

Si può notare come nell'esecuzione su hardware reale si presentino disturbi legati agli inevitabili fenomeni di decoerenza, dati dal grande

numero di porte usate; tra i risultati di esecuzione sperimentale è capitato infatti che i vettori setosa fossero erroneamente classificati come virginica, in un numero piccolo ma non insignificante di volte. Basti confrontare i risultati sperimentali con quelli di esecuzione simulata usando gli stessi vettori d'input e di training in figura 5.8 per rendersi conto dell'effetto non trascurabile delle interazioni ambientali sui qubit. Non sono disponibili dati in merito al tempo di esecuzione effettivo sul processore, in quanto l'informazione non viene più fornita nella versione del software in uso mentre si scrive questa tesi. Tenendo conto del tempo di attesa in coda, il tempo totale può andare da mezz'ora alle due ore in media.

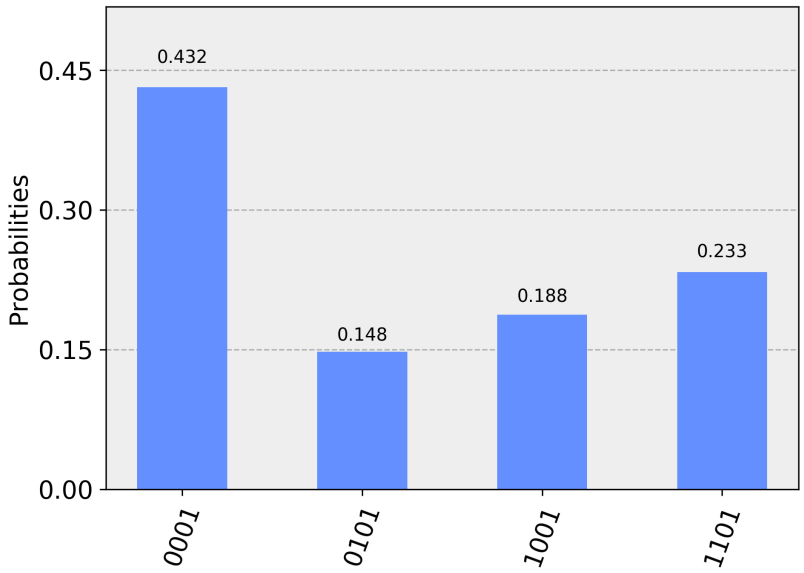


Figura 5.8: Risultati simulati multiclasse (setosa)

Per verificare l'efficienza dell'algoritmo con maggiori risorse, si sono eseguiti vari tentativi di classificazione con circuiti simulati, al variare del numero di vettori di training. In tabella 5.2 si possono trovare i relativi risultati, dove si intende che per ogni esperimento si è costruito un circuito che accettasse insiemi casuali di vettori dal data set ed è stato eseguito 8192 volte, dopo di che si sono rimescolati gli insiemi di training e si è rieseguito l'esperimento per 10 volte in totale.

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

Tabella 5.2: Risultati positivi di classificazione con 2^m vettori di training

I risultati vengono necessariamente da simulazioni, in quanto sono

necessari 23 qubit per il caso con $m = 7$ qubit. Un esempio di simulazione di classificazione positivo, in quanto con $2^7 = 128$ vettori di training è usato quasi tutto il data set, per la classe virginica è visibile in figura 5.9.

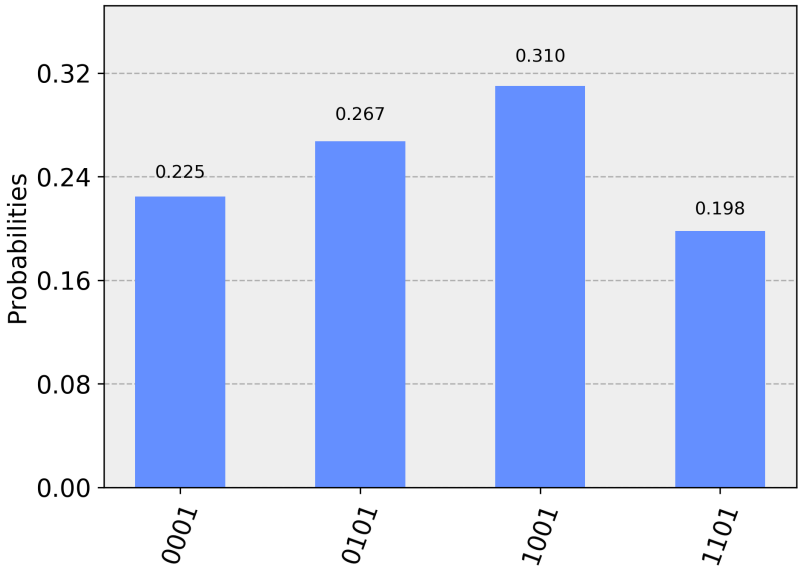


Figura 5.9: Classificazione di virginica multiclasse

Si fa presente che l'uso di tecniche di preprocessing avrebbero permesso di distinguere le classi versicolor e virginica con ben meno vettori di training, tuttavia uno tra gli scopi di questa tesi era verificare fino a che punto si possono immagazzinare vettori in quantità sempre maggiori ai fini della classificazione ed ottenere i risultati aspettati.

5.4 UN ESEMPIO PIÙ SEMPLICE

Per ottenere dei risultati più chiari, si è effettuata la classificazione anche su un data set semplificato, formato da vettori bidimensionali appartenenti a quattro classi, disposti in cluster ben separati tra loro.

Usando lo stesso procedimento visto nella sezione precedente, si sono selezionati due elementi da ciascun cluster e li si è etichettati con un colore; si è poi confrontato in quattro esecuzioni separate un terzo vettore casuale da ciascun gruppo.[1]

I risultati di simulazione vengono presentati nel riquadro 5.12, quelli di esecuzione su hardware reale non sono disponibili per mancanza di tempo dovuta alle scadenze di consegna della tesi.

1 Si veda [2] per una versione one vs. all in cui vengono implementati n classificatori quantistici binari di questo tipo, dove n è il numero di classi.

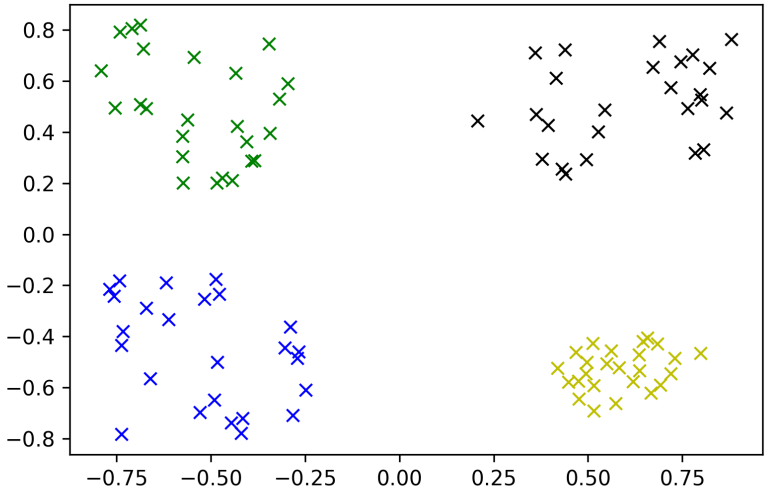


Figura 5.10: Data set a cluster

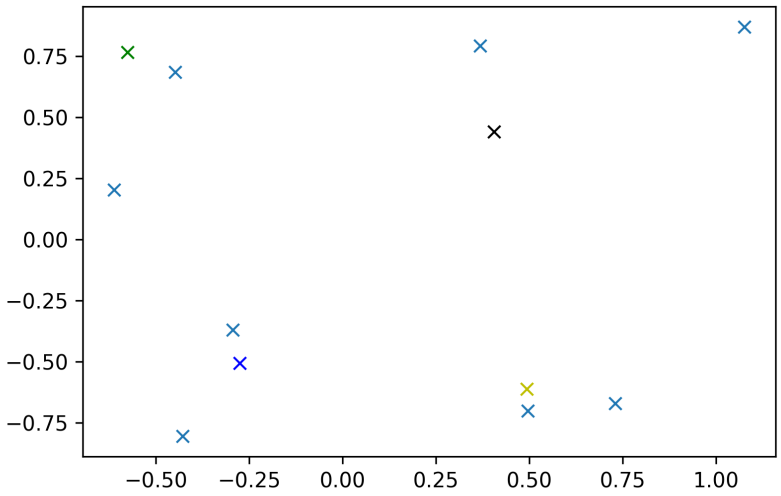
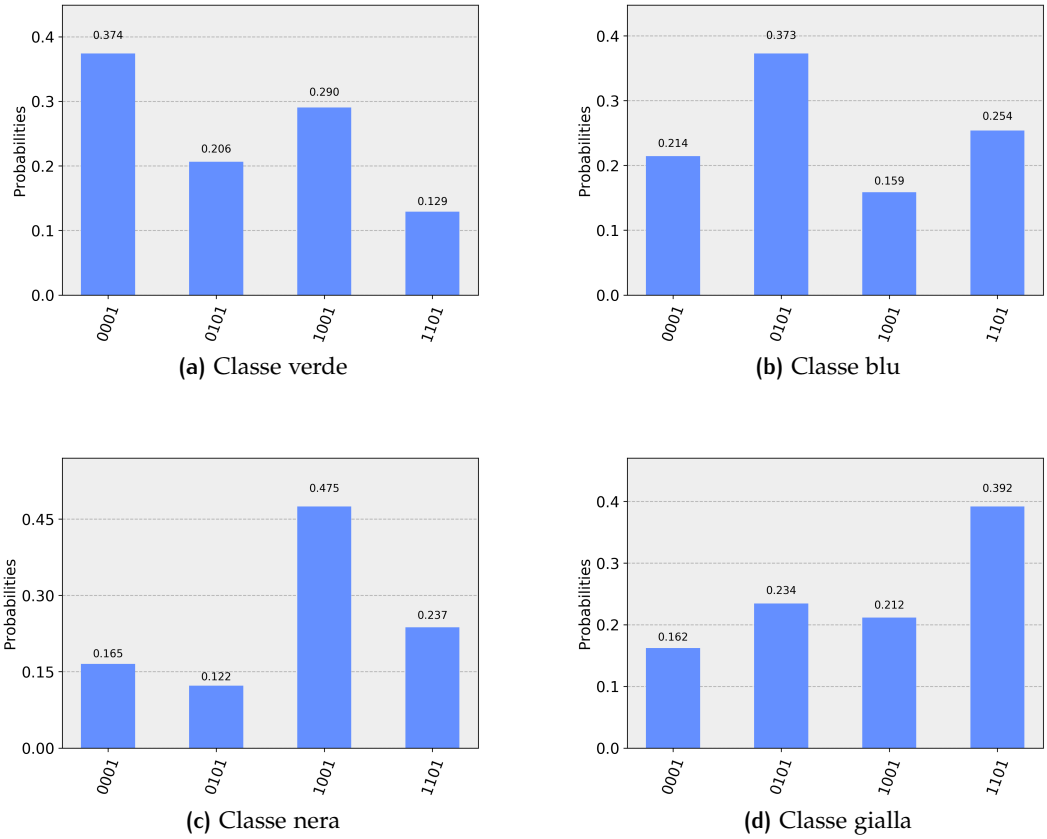


Figura 5.11: Vettori casuali dopo l'applicazione di arcoseno



(a) Classe verde

(b) Classe blu

(c) Classe nera

(d) Classe gialla

Figura 5.12: Risultati di simulazione su cluster semplici

6 CONCLUSIONE

Partendo da un semplice classificatore KNN classico si è riusciti ad implementarne la versione quantistica proposta da Schuld et al. [15]. 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 e non solo per due alla volta. È stata usata la tecnica di costruzione di quantum database arbitrari FF-QRAM proposta da Petruccione et al. [9]. L'algoritmo ha avuto un discreto successo e fornisce una distribuzione di previsioni in linea con le aspettative, sebbene le esecuzioni su hardware reale siano ancora significativamente in balia dei fenomeni di rumore. L'avvento di computer quantistici con un numero maggiore di qubit può aprire la strada ad approcci concreti di analisi dati con data set di dimensioni e complessità considerevoli, lasciando intravedere un'opportunità di applicazione commerciale nei confronti dei big data.

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], alla comunità di Qiskit su `https://quantumcomputing.stackexchange.com/` o ai tanti articoli sull'argomento che escono ogni anno in libera consultazione.

Un'interessante proseguimento di questo progetto potrebbe essere la messa a disposizione degli algoritmi qui studiati per chi volesse usarli per analizzare data set personalizzati. Questo potrebbe essere realizzato attraverso un'ulteriore ottimizzazione e documentazione del codice pubblicato su GitHub, oppure con la creazione di un programma completo apposito, o ancora con la costruzione di un'interfaccia web in cui caricare i propri file (CSV per esempio) e lasciare che il server ottimizzi i dati, crei il circuito quantistico e mandi l'ordine di esecuzione correlato.

6.1 COMMENTO

Lavorare a questa tesi mi ha permesso di esplorare le terre, del machine learning e del quantum computing, due materie fortemente moderne ed in continua evoluzione. Nonostante l'approfondimento delle materie compiuto in questi mesi di lavoro sia tutt'altro che com-

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

pleto, 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 assidua, aggiornando in corso d'opera il documento finale.

A APPENDICE

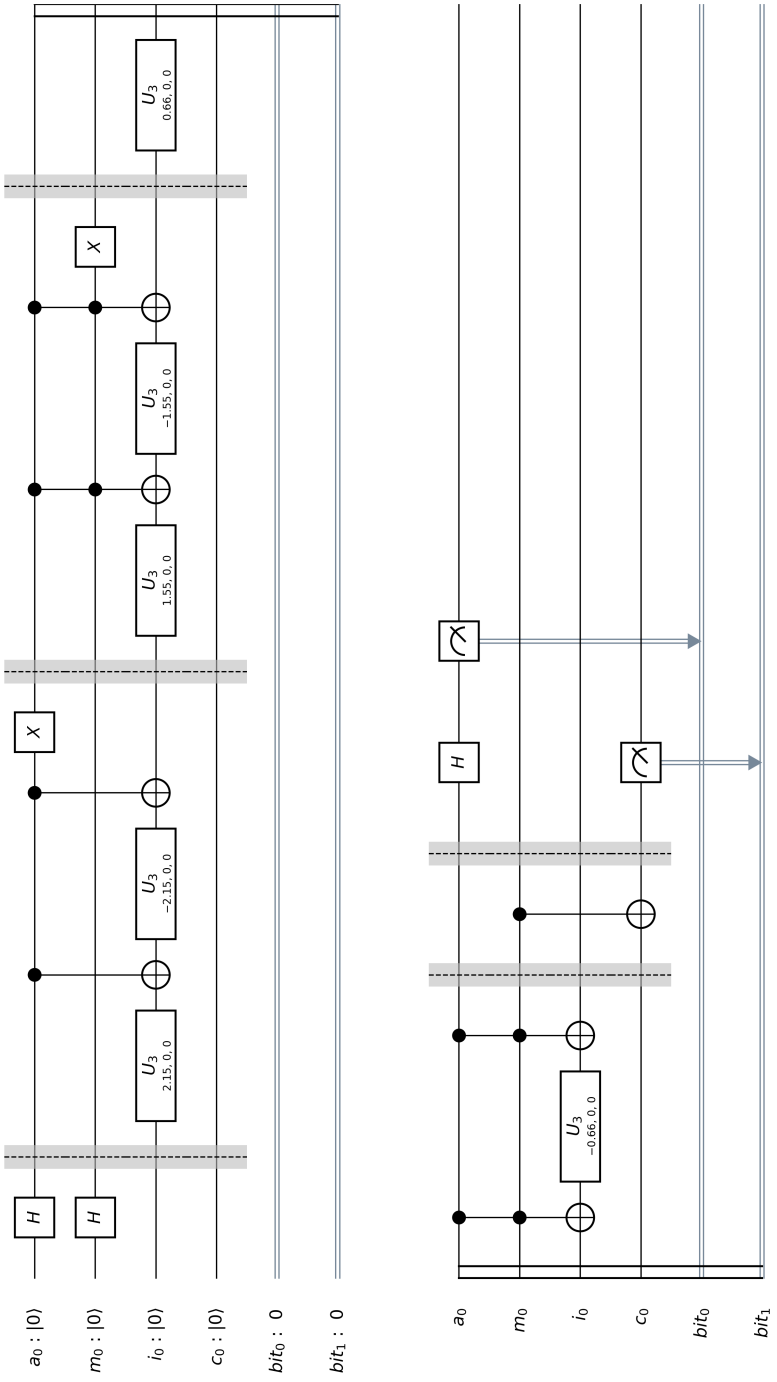


Figura A.1: Il circuito quantistico

BIBLIOGRAFIA

[1] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110.

[2] Sashwat Anagolum. *Quantum machine learning: distance estimation for k-means clustering*. https://towardsdatascience.com/quantum-machine-learning-distance-estimation-for-k-means-clustering-26bccfbfcc76. [Online; recuperato il 29 settembre 2019]. 2019.

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

[4] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe e Seth Lloyd. «Quantum machine learning». In: *Nature* 549 (2017), 195 EP –. URL: https://doi.org/10.1038/nature23474.

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

[6] 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/.

[7] 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].

[8] 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.

[9] 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.

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

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

[12] Patrick Rebentrost, Masoud Mohseni e Seth Lloyd. «Quantum Support Vector Machine for Big Data Classification». In: *Phys. Rev. Lett.* 113.130503 (2014).

[13] IBM Research e the IBM QX team. *Decoherence*. https://qiskit.github.io/ibmqx-user-guides/full-user-guide/002-The_Weird_and_Wonderful_World_of_the_Qubit/006-Decoherence.html. [Online; recuperato l'11 settembre 2019]. 2017.

[14] Troels F. Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V. Isakov, David Wecker, John M. Martinis, Daniel A. Lidar e Matthias Troyer. «Defining and detecting quantum speedup». In: *Science* 345.6195 (2014), pp. 420–424. ISSN: 0036-8075. DOI: 10.1126/science.1252319. eprint: https://science.sciencemag.org/content/345/6195/420.full.pdf. URL: https://science.sciencemag.org/content/345/6195/420.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

COLOPHON

Questo documento è stato composto usando il tema tipografico `classicthesis` sviluppato da André Miede e Ivo Pletikosić. Lo stile trae ispirazione dal libro fondamentale sulla tipografia *"The Elements of Typographic Style"* di Robert Bringhurst. `classicthesis` è disponibile per LaTeX e LyX:

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

Gli utenti soddisfatti di `classicthesis` di solito inviano una cartolina all'autore; una collezione di cartoline ricevute fin ora è esposta qui:

http://postcards.miede.de/

Grazie mille per il tuo riscontro e contributo.