Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Applicazioni di Circuiti Quantistici Variabili nell'Imputazione di Dati Mancanti in Dataset Complessi

Relatore:
Chiar.ma Prof.ssa
Elisa Ercolessi

Correlatore:
Chiar.mo Prof.
Stefano Lodi

Presentata da:
Davide Donati

Sessione IV
Anno Accademico 2023/2024

Indice

Abstract

La ricostruzione di un dataset a partire da un insieme di dati incompleti è un tema di ricerca molto discusso, in quanto la presenza di dati mancanti o corrotti è un problema comune e di rilievo in molte applicazioni. In questo lavoro ho replicato un modello di Quantum Machine Learning per l'imputazione di dati mancanti, che sfrutta un circuito quantistico per apprendere la struttura dati e creare una stima ottimale per completare il dataset in esame. Il modello è stato testato inizialmente su delle distribuzioni di probabilità note e poi adattato per l'imputazione di dati mancanti in un dataset reale. I risultati ottenuti sono stati confrontati con metodi classici di imputazione, sullo stesso dataset, e hanno mostrato, nella maggior parte dei casi, una maggiore accuratezza delle ricostruzioni. Questo lavoro vuole mostrare come l'utilizzo di tecniche quantistiche sia tanto valido quanto promettente per la risoluzione di problemi di imputazione dati.

Introduzione

L'analisi dei dati riveste un ruolo fondamentale per la ricerca scientifica e per le applicazioni industriali, incrementando, quindi, il bisogno di metodi e strumenti per mantenere l'integrità e qualità dei dati. In questo contesto, la gestione dei dati mancanti è un problema critico che può, non solo, pregiudicare la correttezza delle analisi statistiche e dei modelli predittivi, ma anche compromettere la validità delle conclusioni tratte. Tradizionalmente, la comunità scientifica ha fronteggiato questo problema attraverso processi classici di imputazione di dati, come la sostituzione dei valori mancanti con grandezze statistiche, come la media o la mediana, o l'utilizzo di algoritmi di Machine Learning per la stima di essi come K-Nearest Neighbors (KNN) o Support Vector Machines (SVM).
Nei tempi più recenti, il panorama del calcolo e del trattamento dell'informazione ha visto emergere un nuovo paradigma: la computazione quantistica. Questa nuova tecnologia promette di rivoluzionare il mondo dell'informatica, attraverso la possibilità di lavorare con qubit e stati di sovrapposizione, la computazione quantistica offre prospettive inedite per risolvere problemi complessi e di alta complessità computazionale. La frontiera scientifica ha iniziato ad esplorare le potenzialità che può offrire la combinazione di Machine Learning e Computazione Quantistica, in particolare per la risoluzione di problemi di ottimizzazione e di classificazione, il Quantum Machine Learning.
L'obiettivo di questa tesi è esplorare la potenzialità di metodi ibridi di Quantum Machine Learning per la gestione dei dati mancanti. Partendo da distribuzioni di probabilità di dati, si propone un modello basato sulle Quantum Circuit Born Machines(QCBM) che sfrutta la computazione quantistica per apprendere e simulare queste distribuzioni, per la stima e la successiva imputazione dei valori assenti col fine di poter lavorare con dataset completi e di alta qualità.
Questo lavoro si propone di ricreare un modello di Quantum Machine Learning e di testarne l'efficacia su dataset sintetici, come delle semplici distribuzioni Gaussiane e di Maggioranza, e reali, in particolare è stato scelto per semplicità e quantità di studi a riguardo il dataset Iris comprendente di dettagli sulle lunghezze e larghezze dei petali e sepali di questi fiori. Inoltre, verrà messo a confronto con metodi di imputazione classica, utilizzando metriche di valutazione della qualità dei dati e di performance dei modelli.

Struttura della tesi

- Capitolo 1: Introduce il concetto dei dati mancanti, la loro classificazione e discute le metodologie classiche di imputazione più comuni.
- Capitolo 2: Fornisce una panoramica sulla meccanica quantistica, in particolare sulla computazione quantistica, necessaria per comprendere i concetti di qubit, stati di sovrapposizione e di creare un framework matematico per la comprensione e la realizzazione di algoritmi quantistici.
- Capitolo 3: Presenta il circuito quantistico ricreato, discute la sua struttura da un punto di vista teorico e ne spiega il funzionamento, testandolo su dataset sintetici.
- Capitolo 4: Applica il modello concepito su un dataset reale e lo confronta con metodi di imputazione classici.

Capitolo 1

Il problema: la gestione dei dati mancanti

1.1 Definizione del problema

I dati mancanti si riferiscono a informazioni non registrate per una determinata variabile nell'osservazione di interesse. Ad esempio, se in un sondaggio una persona non risponde alla domanda sul reddito, quel dato è considerato mancante. In statistica, una variabile è una caratteristica o proprietà misurabile che può assumere diversi valori. [1]

Un'altra definizione importante da dare è quella di dataset, ossia una raccolta strutturata di dati. Generalmente, in un dataset tabellare, ogni colonna rappresenta una specifica variabile, mentre ogni riga corrisponde a un'osservazione distinta. Ad esempio, in un sondaggio, le risposte di ciascun partecipante costituiscono le osservazioni, e le domande del sondaggio rappresentano le variabili. Un dataset completo e accurato è fondamentale per garantire analisi statistiche valide e affidabili. [2]

Durante una revisione periodica dello strumento QRISK, un algoritmo predittivo utilizzato da medici professionisti per la valutazione del rischio di malattie cardiovascolari, è emerso un problema significativo: l'assenza di dati in input relativi al colesterolo e all'HDL, comprometteva le analisi, risultando in correlazioni sbagliate tra i rischi calcolati e quelli effettivi. In particolare, si è scoperto che quasi il 70% dei valori di HDL risultava mancante. Questo è uno dei tanti esempi che evidenziano il problema della mancanza di dati e come esso influisce sulla qualità delle analisi scientifiche. [3]

Ricerche recenti evidenziano che circa l'80% dei dataset utilizzati nella ricerca scientifica presentano dati mancanti. Questo fenomeno è evidente anche nella psicologia e nell'educazione, dove un'analisi di diversi studi ha mostrato che il 50% degli articoli

esaminati conteneva dati incompleti. [4]

La mancanza di dati presenta vari problemi che possono significativamente influenzare la qualità e l'attendibilità dei dati. Di conseguenza, questa riduce il potere statistico, ossia la probabilità di un test statistico di rifiutare correttamente un'ipotesi nulla quando è falsa. Con ipotesi nulla si intende l'idea di base che non ci sia un effetto, una relazione o una differenza tra le variabili esaminate, un test statistico dovrebbe essere in grado di stabilire se le differenze o le associazioni osservate nei dati siano abbastanza forti da non essere attribuibili al caso, portando eventualmente al rifiuto dell'ipotesi nulla. La mancanza di singole osservazioni può, anche, introdurre bias nell'analisi e nella stima di parametri portando ad interpretazioni distorte dei risultati e a conclusioni ingannevoli. Un campione con una grande porzione di dati mancanti potrebbe smettere di riflettere le caratteristiche della popolazione che dovrebbe rappresentare, indebolendo di conseguenza la generalizzabilità dei risultati ottenuti, facendoli diventare inutilizzabili in altri campi. [5]

Di fronte a questa problematica, si è reso necessario stabilire una soglia di tolleranza per la presenza di dati mancanti, fissata comunemente al 5%. Oltre questa soglia, diventa essenziale adottare tecniche di imputazione per sostituire i valori mancanti con stime adeguate. [6]

In ambito statistico vengono sviluppate le prime, piuttosto semplici, tecniche di imputazione, che prevedevano l'assegnazione di valori fissi, come la media o la mediana della variabile interessata. In seguito sono stati esplorati altri approcci che includevano la rimozione delle righe o delle colonne con dati incompleti o la sostituzione con valori casuali. Tuttavia, queste metodologie presentavano diverse limitazioni come la perdita di informazioni, la riduzione della distribuzione dei dati o la possibilità di distorcere la distribuzione stessa.

Per superare questi problemi, negli ultimi anni si è assistito ad un crescente interesse verso lo sviluppo di tecniche di imputazione più sofisticate. Per esempio tecniche di machine learning permettono di stimare i valori mancanti sfruttando le relazioni intrinseche tra le variabili presenti nel dataset, garantendone una ricostruzione più accurata.

1.1.1 Classificazione dei Dati Mancanti

Dal punto di vista teorico, si possono classificare tra categorie principali di dati mancanti: MCAR (Missing Completely At Random), MAR (Missing At Random) e

MNAR (Missing Not At Random). [1]

Definiamo una variabile osservata come una caratteristica misurabile per la quale sono disponibili dati raccolti direttamente. Al contrario, una variabile non osservata, o latente è una caratteristica che non è stata misurata o per la quale i dati non sono disponibili.

Nel caso di MCAR, i dati mancanti sono distribuiti in modo completamente casuale, senza alcuna dipendenza dalle variabili osservate e non osservate. Questo è lo scenario ottimale, in quanto introduce un bias minimo nelle analisi statistiche. Un esempio può essere un questionario di ricerca in cui a una popolazione vengono sottoposte domande randomiche, se il campione risulta abbastanza ampio e ben selezionato, la probabilità che un individuo non risponda a una domanda sarà indipendente sia dalle risposte alle altre domande che dalle caratteristiche della popolazione stessa.

La situazione diventa più complessa con dati MAR, dove la probabilità di assenza dipende esclusivamente da variabili osservate, ma non dalla variabile mancante stessa. In questo caso, un esempio può essere rappresentato da un test a sorpresa in classe: la probabilità che uno studente non risponda a una domanda dipende dal suo livello di preparazione, ma non necessariamente dal contenuto specifico della domanda.

Il caso più problematico è quello dei dati MNAR, in cui la probabilità di assenza è legata direttamente alla variabile stessa. Questa tipologia non consente di fare precise assunzioni sui dati mancanti. Ad esempio, se si raccoglie informazioni sul reddito di una popolazione, potrebbe accadere che persone con guadagni molto alti o molto bassi scelgano di non rispondere, alterando così la distribuzione dei dati, quindi introducendo un bias significativo. [7]

Queste distinzioni possono essere formalizzate matematicamente. Nel caso MAR, la probabilità che una variabile Y sia mancante ($R_Y = 1$) dipende solo dalle variabili osservate X, quindi possiamo scrivere:

$$P(R_Y = 1 \mid Y, X) = P(R_Y = 1 \mid X)$$

Nel caso MNAR, invece, la probabilità di assenza dipende direttamente dalla variabile stessa:

$$P(R_Y = 1 \mid Y, X) = P(R_Y = 1 \mid Y)$$

Infine, per i dati MCAR, la probabilità di assenza è indipendente da tutte le altre variabili, sia osservate che non osservate:

$$P(R_Y = 1 \mid Y, X) = P(R_Y = 1)$$

1.2 Imputazione dei dati mancanti

La mancanza di dati in un dataset, come abbiamo visto, può risultare in diverse problematiche.

Uno dei primi approcci utilizzati per gestire i dati mancanti è l'omissione, essa consiste nella rimozione delle osservazioni che contengono valori mancanti. La modalità più comune che viene utilizzata è la listwise deletion, ovvero l'eliminazione di tutte le righe che contengono almeno un valore mancante. Questo metodo è molto semplice da implementare e garantisce un dataset pulito senza valori mancanti. Tuttavia, se i dati mancanti non sono distribuiti in modo casuale(MCAR), ma seguono un pattern specifico, questa strategia può introdurre bias. Il metodo gemello della listwise deletion è la pairwise deletion, che, invece, consiste nella rimozione delle colonne, quindi delle variabili, che contengono valori mancanti. Questo metodo viene utilizzato solo quando le variabili con dati mancanti sono poche e non influenzano significativamente il modello. Solitamente si predilige l'eliminazione delle singole osservazioni, rispetto alla perdita di variabili, in quanto queste ultime possono essere fondamentali per la costruzione del modello. [8]

Un approccio più sofisticato è l'interpolazione, che consiste nel riempire i valori assenti con stime basate sui valori esistenti. Questo metodo è particolarmente utile quando i dati mancanti sono sporadici e l'interpolazione non introduce distorsioni rilevanti. Essa viene utilizzata, per esempio, nelle serie temporali, dove è consuetudine interpolare un valore mancante basandosi sui valori precedenti e successivi, per mantenere la continuità della serie. [9]

Il maggior successo viene, però, riscontrato utilizzando l'imputazione, in quanto permette di preservare la completezza del dataset e, quindi, di applicare tecniche di analisi più sofisticate che richiedono campioni completi. L'imputazione prevede la sostituzione dei valori mancanti con valori stimati.

Un primo metodo di imputazione è l'imputazione singola, che consiste nella conversione di ogni singolo valore mancante con un unico valore calcolato. Le tecniche più semplici di imputazione singola sono Hot Deck e Cold Deck, che sostituiscono i valori mancanti con valori presi a caso da altre osservazioni della stessa variabile, nel primo caso, e da dataset esterni, nel secondo. Altri approcci che rientrano in questa categoria, ma richiedono maggiori calcoli e di conseguenza sono più accurati, sono Mean Substitution e Regression Imputation.

La Mean Substitution è il rimpiazzamento di ogni valore mancante con la media aritmetica dei valori noti della variabile corrispondente. Questo metodo non va a

distorcere la distribuzione della variabile, ma, proprio per questo motivo, riduce la varianza complessiva, ossia la variazione dei dati rispetto alla media. Questo porta ad una perdita di informazione, che può essere fondamentale per la costruzione del modello.

La Regression Imputation utilizza la regressione lineare per "riempire" i valori mancanti. In pratica, si costruisce un modello lineare basato sui dati disponibili, individuando la relazione tra la variabile di interesse e una o più variabili esplicite. Il modello, solitamente rappresentato dall'equazione $y = a + bx$, equazione di una retta generica, viene usato per prevedere i valori mancanti. Il vantaggio è che questo approccio può fornire stime molto accurate, minimizzando l'errore rispetto ai dati noti. Tuttavia, poiché i valori imputati sono esattamente quelli previsti dal modello, si perde la naturale variabilità dei dati, rendendo l'insieme imputed meno "rumoroso" rispetto a quanto accadrebbe realmente. Questo può portare a una sottostima dell'errore e a conclusioni troppo ottimistiche.

L'imputazione singola è un metodo molto semplice e veloce per risolvere il problema dei dati mancanti in maniera abbastanza efficace. Col tempo, però, si è capito che questo metodo non tiene conto dell'incertezza associata ai valori imputati, trattando i dati come se fossero stati osservati direttamente, quando in realtà sono solo approssimazioni. Portando, come abbiamo visto, a una sottovalutazione della varianza e a stime distorte.

Per superare queste limitazioni, sono state sviluppate tecniche di imputazione multiple, che prevedono un processo suddiviso in tre fasi:

- Imputazione: vengono creati m dataset diversi, in cui i valori mancanti vengono sostituiti con stime differenti.
- Analisi: ciascun dataset viene analizzato separatamente, producendo m risultati diversi.
- Pooling: i risultati ottenuti vengono combinati per calcolare na stima finale della media, della varianza e degli intervalli di confidenza.

Uno degli approcci più noti per l'imputazione multipla è il Multiple Imputation by Chained Equations(MICE). I limiti principali di questo metodo sono legati alle prestazioni, specialmente quando il numero di osservazioni è molto elevando o quando il dataset contiene variabili altamente non lineari o di grande dimensionalità.
Negli ultimi anni, sono state sviluppate tecniche più avanzate che sfruttano algoritmi di machine learning. Un esempio è il Multiple Imputation with Denoising Autoencoders(MIDA), che utilizza autoencoders per apprendere una rappresentazione latente dei dati e imputare i valori mancanti in maniera più accurata. Sebbene le tecniche di imputazione multipla offrano un alto grado di precisione, sono più complesse e

richiedono una maggiore potenza computazionale. [10]

1.2.1 Identificazione del tipo di dati mancanti

Per comprendere che tipo di pattern seguono i dati mancanti, si possono utilizzare modelli basati sulla teoria dei grafi. Ad esempio, supponiamo di avere tre variabili X, Y e Z, dove Z è completamente osservata, mentre X e Y contengono dati mancanti. In questo caso, la condizione per determinare se i dati sono MAR o MCAR è la seguente:

$$X \perp R_Y \mid (R_X, Z)$$

dove R_X e R_Y sono indicatori della mancanza dei dati. Questo significa che l'assenza dei dati in X è indipendente da Y se si conosce Z. Se questa condizione non è soddisfatta, è probabile che i dati siano MNAR.

Anche in presenza di dati MNAR, esistono strategie per gestire la mancanza di dati. Se, ad esempio, Y spiega la mancanza di dati in X e la mancanza di dati in Y è almeno MAR, è possibile stimare una distribuzione congiunta $P(X, Y)$ e utilizzare tale distribuzione per ricostruire i valori mancanti. [1]

1.3 Machine Learning

Il Machine Learning è una branca dell'intelligenza artificale che si occupa di sviluppare algoritmi e modelli capaci di apprendere pattern dai dati in input ed applicarli a nuovi dati per fare previsioni o prendere decisioni. In particolare, il Machine Learning si basa sull'idea che i sistemi possano migliorare automaticamente con l'esperienza, senza essere esplicitamente programmati. Queste capacità di apprendimento e generalizzazione sono alla base di molte applicazioni moderne, come il riconoscimento facciale, la traduzione automatica, la guida autonoma e il riconoscimento vocale. [11] [12]

Differisce dai metodi statistici tradizionali per la capacità di prendere decisioni autonome, l'utilizzo di funzioni di errore per la valutazione delle prestazioni e dell'accuratezza del modello e di un processo di ottimizzazione mirato a minimizzare l'errore. [13]

Possiamo evidenziare tre tipologie di apprendimento automatico: l'apprendimento supervisionato, l'apprendimento non supervisionato e l'apprendimento per rinforzo.

Nel Supervised learning, il modello viene addestrato su un dataset etichettato, che contiene, quindi, sia le variabili in input(feature) sia il relativo valore di output

atteso(target). L'obiettivo è individuare relazioni tra le feature e il target per poter fare previsioni accurate su nuovi dati. Un esempio classico di apprendimento supervisionato è il Support vector Machine(SVM), un modello che suddivide lo spazio delle feature in modo da massimizzare il margine tra classi differenti, migliorando così la capacità di classificazione.

L'Unsupervised Learning si basa su dati privi di etichette. In questo caso, il modello esplora autonomamente la struttura intrinseca dei dati per individuare pattern nascosti o raggruppare i dati in insiemi omogenei. A differenza dell'apprendimento supervisionato, che risponde a domande specifiche, l'apprendimento non supervisionato si concentra sulla scoperta di relazioni sottostanti. Un esempio emblematico è il K-Means Clustering, che raggruppa i dati in cluster in base alla loro similarità.

Il Reinforcement Learning è una tecnica di apprendimento basata su un sistema di ricompense e punizioni. Il modello apprende attraverso interazioni con un ambiente dinamico, prendendo decisioni sequenziali per massimizzare una ricompensa cumulativa. Questo approccio è ampiamente utilizzato in settori come la robotica, i giochi e la finanza, dove un agente deve ottimizzare le proprie azioni basandosi sui feedback ricevuti dall'ambiente. [14]

1.3.1 Machine Learning per l'imputazione dei dati mancanti

Partendo già da una breve spiegazione di cosa sia il Machine Learning, possiamo intuire come questa tecnica possa essere adottata anche per la gestione dei dati mancanti. Se pensiamo di prendere come input del nostro modello un dataset con valori mancanti, possiamo addestrarlo a riconoscere i pattern nei dati osservati e ad uscirne con una stima che possa sostituire i dati assenti. Questa è la base del funzionamento delle tecniche di imputazione basate su Machine Learning.

Tra gli approcci più utilizzati troviamo algoritmi come K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees, Random Forest, Clustering, Ensemble Methods, Multiple Imputation Chained Equations (MICE) e Denoising Autoencoders. Ciascuno di questi modelli sfrutta metodologie diverse per stimare i valori mancanti, offrendo vantaggi e limitazioni in base al contesto applicativo.

K-Nearest Neighbors

Uno dei metodi più semplici e intuitivi per l'imputazione dei dati mancanti è l'algoritmo K-Nearest Neighbors (KNN). Questo approccio si basa sull'assunzione che osservazioni

simili tendono ad avere valori simili. L'algoritmo, quindi, cerca i k punti più vicini, per imputare un valore mancante, in base ad una misura di distanza e calcola una stima ponderata dei valori osservati. [15]

Una delle formule più comuni per il calcolo della distanza tra due punti è la distanza euclidea, definita come:

$$Dist_{xy} = \sqrt{\sum_{k=1}^{m}(X_{ik} - X_{jk})^2}$$

dove X_{ik} è il valore dell'attributo j-esimo contenente dati mancanti e X_{jk} è il valore dell'attributo j-esimo contenente dati osservati.

Una volta identificati i vicini più prossimi, il valore mancante viene stimato come la media pesata dei valori osservati, secondo la formula:

$$X_k = \frac{\sum_{j=1}^{J} w_j v_j}{\sum_{j=1}^{J} w_j}$$

dove v_j rappresenta i valori osservati di attributi con dati mancanti e w_j è il peso associato a ciascun vicino, calcolato come:

$$w_j = \frac{1}{Dist_{xy}}$$

L'algoritmo KNN è versatile e adatto sia a variabili discrete che continue, ma presenza alcune limitazioni. La sua efficienza diminuisce all'aumentare della dimensione del dataset, poiché richiede il calcolo della distanza tra tutti i punti. Inoltre, in presenza di dati rumorosi o non lineari, può generare imputazioni meno affidabili, introducendo associazioni errate tra i dati.

Support Vector Machines

Un'altra tecnica efficace per l'imputazione è l'uso delle Support Vector Machines (SVM), un modello ampiamente utilizzato per la classificazione e la regressione. L'idea alla base dell'SVM è quella di trovare l'iperpiano ottimale che separi i dati in maniera da massimizzare il margine tra le classi.

Dato ill set di dati (x_i, y_i) con $y_i \pm 1$ che indica la classe a cui x_i appartiene, l'iperpiano è definito dall'equazione $w \cdot x + b = 1$, sicchè:

$$w \cdot x_i + b \geq 1$$

quando $y_i = 1$, mentre

$$w \cdot x_i + b \leq -1$$

quando $y_i = -1$, dove w è il vettore dei pesi, ortogonale all'iperpiano che ne definisce l'orientamento e l'inclinazione, b è il termine di bias, è uno scalare che determina la distanza dell'iperpiano dall'origine.

L'algoritmo cerca di trovare w e b tali da minimizzare la seguente funzione di costo:

$$\frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i$$

dove C è il parametro di regolarizzazione, che controlla il compromesso tra la complessità e l'accuratezza del modello sulla base di addestramento, un valore di C alto attribuisce una penalità maggiore per esempio agli errori di classificazione portando il modello a cercare di classificare correttamente tutti i punti del dataset, e ξ_i sono le variabili di slack che permettono la violazione dei vincoli di margine, ossia la distanza tra il punto e l'iperpiano, in presenza di dati non linearmente separabili.

Nel contesto dell'imputazione, SVM può essere utilizzato per predire i valori mancanti come un problema di classificazione o regressione, sfruttando i dati osservati per costruire un modello predittivo robusto. Questo metodo è particolarmente utile quando i dati presentano una chiara struttura separabile, ma può risultare meno efficace in contesti con elevata variabilità e distribuzioni complesse.

Decision Trees

Un altro approccio popolare si basa sugli alberi decisionali. Un Decision Tree costruisce una struttura gerarchica di decisioni che permette di stimare i valori mancanti in base ai dati disponibili. Ogni nodo dell'albero rappresenta una variabile predittiva, e le ramificazioni corrispondono alle regole di suddivisione dei dati.

Per imputare un valore mancante, si può costruire un albero decisionale specifico per la variabile interessata e utilizzare le decisioni apprese dai dati osservati per stimare il valore più probabile. Gli alberi decisionali sono efficienti e facilmente interpretabili, ma possono soffrire di overfitting, specialmente su dataset complessi.

Metodi Ensemble e Random Forest

Un approccio particolarmente efficace per migliorare la precisione delle previsioni e ridurre il rischio di overfitting è l'uso di modelli ensemble, tra cui il Random Forest è

uno dei più noti e utilizzati.

Il Random Forest è un modello di machine learning che combina più alberi decisionali per generare una previsione più stabile e accurata. A differenza dei singoli alberi decisionali, che possono tendere all'overfitting quando il dataset di addestramento è troppo complesso o contiene rumore, Random Forest introduce casualità nel processo di apprendimento, rendendo il modello più robusto. Ogni albero della foresta viene addestrato su un sottoinsieme casuale dei dati e utilizza una selezione randomizzata delle feature per la suddivisione dei nodi.

L'idea alla base di questo approccio è che, anziché affidarsi a un unico albero decisionale, il modello effettua una votazione a maggioranza (per problemi di classificazione) o calcola la media delle previsioni (per problemi di regressione). Questo meccanismo permette di ottenere un modello più generale e resistente alle fluttuazioni nei dati. Tuttavia, Random Forest può essere computazionalmente costoso, poiché l'addestramento di un gran numero di alberi richiede tempo e risorse. Inoltre, l'interpretabilità del modello è ridotta rispetto a un singolo albero decisionale, in quanto non esiste un unico percorso decisionale chiaro da seguire. [16]

Least Squares Support Vector Machines e CZ-LSSVM

Oltre ai classici metodi ensemble, esistono modelli avanzati che sono stati adattati per l'imputazione dei dati mancanti. Due esempi particolarmente interessanti sono il Least Squares Support Vector Machine (LSSVM) e il Clustered Z-score Least Squares Support Vector Machine (CZ-LSSVM).

L'LSSVM è un'estensione del Support Vector Machine (SVM), ma ottimizzato per problemi di regressione. Invece di risolvere il problema di ottimizzazione classico delle SVM, LSSVM utilizza un sistema di equazioni lineari, rendendo il calcolo più efficiente. Questo modello sfrutta una funzione di kernel per trasformare i dati in uno spazio a dimensione superiore, permettendo di trovare pattern anche in dataset con strutture complesse.

Il CZ-LSSVM, invece, è un'evoluzione dell'LSSVM pensata per la classificazione. Introduce una normalizzazione basata sul Clustered Z-score, che migliora la gestione dei dati mancanti, riducendo la distorsione introdotta dalle distribuzioni sbilanciate. Studi recenti hanno dimostrato che questi modelli superano le tecniche tradizionali di imputazione dei dati mancanti, soprattutto nei problemi di classificazione, offrendo prestazioni superiori in termini di accuratezza e generalizzazione. [17]

1.3.2 Processo di addestramento e valutazione

Indipendentemente dal modello utilizzato, il processo di imputazione dei dati mancanti segue una serie di passaggi ben definiti.

Il primo step consiste nella preparazione dei dati, che include la raccolta e la pulizie dei dati necessari da anomalie, errori e valori mancanti. I dati potrebbero aver bisogno di essere normalizzati o standardizzati per garantire una distribuzione uniforme e facilitare il processo di apprendimento, quindi si effettua una fase di preprocessing.

Il dataset viene poi in un insieme di dati mirato all'addestramento del modello e un insieme di dati di test per valutarne le prestazioni. Questo passaggio aiuta ad evitare situazioni di overfitting, in cui il modello si adatta eccessivamente ai dati di addestramento, perdendo la capacità di generalizzare su nuovi dati.

L'addestramento è mirato a trovare i parametri ottimali del modello che minimizzano la funzione di costo. La funzione di costo è una misura che quantifica la discrepanza tra i valori previsti dal modello e i valori reali.

1.3.3 Metriche di valutazione

Finito l'addestramento del modello si utilizzano i dati di test per verificare l'efficacia del modello ed eventualmente apportare modifiche. Sono state ideate diverse metriche per valutare le prestazioni del modello, tra cui l'accuratezza, la precisione, il recall, l'F1-score e l'area sotto la curva ROC. [18]

Metriche di classificazione

L'accuracy è una misura generale di correttezza del modello, calcolata come il rapporto tra il numero di previsioni corrette e il numero totale di previsioni. E' una metrica semplice e intuitiva, ma risulta penalizzante in presenza di classi sbilanciate. Se in un dataset di classificazione binaria il 90% dei dati appartiene alla classe A e solo il 10% alla classe B, un modello che preciso il 100% delle volte la classe A, risulterà avere un'accuracy del 90%, ma risulterebbe completamente inutile per identificare i casi della classe B.

La precision e la recall sono due metriche complementari che misurano la qualità delle previsioni. La precision, in particolare, indica la proporzione di previsioni positive corrette rispetto al numero totale di previsioni positive. La recall, invece, rappresenta

la proporzione di previsioni positive corrette rispetto al numero totale di veri positivi.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

dove TP, FP e FN indicano rispettivamente i True Positive, False Positive e False Negative. Una precision molto alta, va a significare che il modello è molto selettivo nell'etichettare i dati come positivi, riducendo il numero di falsi positivi. Al contrario, una recall molto alta indica che il modello è bravo a identificare i veri positivi, ma potrebbe avere un alto numero di falsi positivi.

Esiste un trade-off tra Precision e Recall: aumentando la Precision si rischia di ridurre la Recall e viceversa. Per esempio, in un sistema di rilevamento di frodi bancarie, un'alta Precision significa che il sistema segnala solo transazioni altamente sospette, mentre un'alta Recall implica che si preferisce segnalare molte più transazioni, aumentando però il numero di falsi positivi.

Per bilanciare questo trade-off, si utilizza l'F1-Score, che è la media armonica tra Precision e Recall:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

L'F1-Score varia tra 0 e 1, dove 1 indica una performance perfetta. È particolarmente utile quando Precision e Recall sono entrambe rilevanti per il problema in esame.

Un altro strumento essenziale per la valutazione di un modello di classificazione è la curva ROC (Receiver Operating Characteristic), che mostra il rapporto tra Veri Positivi (Recall) sull'asse delle ordinate e Falsi Positivi sull'asse delle ascisse, dove ogni punto sulla curva rappresenta un threshold diverso, mostrando come le performance del modello cambiano al variare del threshold.

L'Area Under the Curve(AUC) è un valore scalare che misura l'area sottesa alla curva ROC e rappresenta la capacità del modello di distinguere tra classi positive e negative per ogni threshold. Un AUC pari a 1 indica un modello perfetto, mentre un valore di 0.5 che non fa meglio del caso.

La Specificity, viene spesso affiancata alla Recall, in quanto misura la sensibilità del modello nel riconoscere i veri negativi:

$$Specificity = \frac{TN}{TN + FP}$$

Una Specificity elevata è utile in problemi in cui è importante evitare falsi positivi, come nel caso dei test medici, dove un risultato positivo errato potrebbe portare a trattamenti inutili.

Metriche di regressione

Nei problemi di regressione, l'obiettivo non è classificare dati in categorie, ma stimare un valore numerico il più vicino possibile alla realtà. Per questo motivo, si utilizzano metriche che quantificano l'errore tra i valori predetti e quelli reali.

Il Mean Squared Error (MSE) calcola la media dei quadrati delle differenze tra le predizioni e i valori effettivi:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

dove y_i è il valore reale e $\hat{y}_i$ è il valore predetto e m è il numero di osservazioni.
Il quadrato della differenza amplifica gli errori più grandi, rendendo l'MSE sensibile agli outlier, ossia ai valori estremi che possono influenzare significativamente il risultato

Per rendere l'MSE più interpretabile, si utilizza spesso la sua radice quadrata, chiamata Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{MSE}$$

L'RMSE è utile perché esprime l'errore nella stessa unità dei dati originali, facilitandone l'interpretazione. Più basso è l'RMSE, migliore è la capacità predittiva del modello.

Capitolo 2

Quantum Computing

2.1 Concetti di base

Partiamo con una panoramica della notazione e delle operazioni base che utilizzeremo in questo capitolo.

L'algebra lineare è lo studio di spazi di vettori ed operazioni lineari in questi spazi vettoriali. Lo spazio vettoriale usato nella computazione quantistica è lo spazio vettoriale complesso $\mathbb{C}^n$, di dimensione $n = 2^N$, dove N è il numero di qubit utilizzati.

Gli elementi di uno spazio vettoriali vengono chiamati vettori e vengono normalmente rappresentati utilizzando matrici colonna:

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

dove z_i sono numeri complessi $\in \mathbb{C}$. Un vettore di dimensione n è un elemento di $\mathbb{C}^n$.

Uno spazio vettoriale è definito tale se soddisfa le seguenti proprietà:
- **Addizione**: $\forall \mathbf{z}, \mathbf{w} \in \mathbb{C}^n$, $\mathbf{z} + \mathbf{w} \in \mathbb{C}^n$.

$$z + w = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} z_1 + w_1 \\ z_2 + w_2 \\ \vdots \\ z_n + w_n \end{bmatrix}$$

- **Moltiplicazione per scalare**: $\forall \mathbf{v} \in \mathbb{C}^n$, $\forall \alpha \in \mathbb{C}$, $\alpha \mathbf{v} \in \mathbb{C}^n$.

$$\alpha v = \alpha \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \alpha v_1 \\ \alpha v_2 \\ \vdots \\ \alpha v_n \end{bmatrix}$$

In meccanica quantistica viene utilizzata una notazione più compatta per rappresentare vettori e matrici. Questa notazione è chiamata notazione di Dirac o notazione bra-ket. Un vettore $\mathbf{v}$ viene rappresentato come un ket $|v\rangle$ e un vettore trasposto $\mathbf{v}^T$ viene rappresentato come un bra $\langle v|$.

Un prodotto scalare tra due vettori $|v\rangle$ e $|w\rangle$ viene rappresentato come $\langle v|w\rangle$. Questo prodotto è definito come:

$$\langle v|w\rangle = \sum_{i=1}^{n} v_i^* w_i$$

dove v_i^* è il complesso coniugato di v_i.

Vogliamo ora introdurre un concetto fondamentale dell'algebra lineare, che viene utilizzato in modo massiccio in meccanica quantistica: gli operatori lineari. Un operatore lineare tra due spazi vettoriali V e W è una funzione qualunque $A : V \to W$ che è lineare per tutti i suoi input:

$$A\left(\sum_i \alpha_i \mathbf{v}_i\right) = \sum_i \alpha_i A(\mathbf{v}_i)$$

dove α_i sono scalari e $\mathbf{v}_i$ sono vettori in V.

Una base per uno spazio vettoriale V è un insieme di vettori $|v_1\rangle, |v_2\rangle, \ldots, |v_n\rangle$ tali che ogni vettore $|v\rangle$ in V può essere scritto come una combinazione lineare $|v\rangle = \sum_i \alpha_i |v_i\rangle$ dei vettori dello spanning set.

Per esempio, una base per $\mathbb{C}^2$ è data dai vettori:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

e rappresenta un possibile stato di un singolo qubit.

Quindi un qualsiasi vettore $|v\rangle$ in $\mathbb{C}^2$ può essere scritto come:

$$|v\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

Scelta una base dello spazio vettoriale, un operatore lineare può essere rappresentato da una matrice. Questa matrice è definita come la matrice che agisce su un vettore $\mathbf{v}$ per produrre un nuovo vettore $\mathbf{w}$. Questa matrice è chiamata matrice dell'operatore e viene indicata con A. La matrice dell'operatore A è definita come:

$$A = [A_{ij}]_{i,j=1\cdots n}$$

dove $A_{ij} = \langle e_i|A|e_j\rangle$ e gli elementi $|e_i\rangle$ e $|e_j\rangle$ sono gli elementi della base.

Gli operatori in meccanica quantistica devono essere unitari, in quanto devono preservare la norma dei vettori, per garantire che queste operazioni siano reversibili e che la probabilità totale sia conservata. Un operatore U è definito unitario se soddisfa la seguente relazione:

$$U^\dagger U = UU^\dagger = I$$

dove $U^\dagger$ è l'aggiunto di U e I è l'identità.

L'operatore identità, insieme alle matrici di Pauli, sono gli operatori più comuni utilizzati per un singolo qubit. Essi sono definiti come:

$$I = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

$$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$$

$$Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}$$

$$Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

Possiamo facilmente dedurre i loro effetti su un vettore. Ad esempio, l'operatore X agisce su un vettore $|0\rangle$ come:

$$X|0\rangle = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}0\\ 1\end{bmatrix} = |1\rangle$$

Mentre su un vettore $|1\rangle$ come:

$$X|1\rangle = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \begin{bmatrix}1\\ 0\end{bmatrix} = |0\rangle$$

2.1.1 Meccanica Quantistica

La meccanica quantistica è un framework matematico per descrivere il comportamento di sistemi fisici a livello microscopico. I postulati della meccanica quantistica sono derivati da un lungo processo di sperimentazione e osservazione. Adesso che abbiamo dato le basi di algebra lineare per capire i postulati principali, possiamo procedere con la descrizione dei postulati più importanti al nostro lavoro.

Il primo postulato descrive il luogo in cui vivono i nostri sistemi fisici. Questo luogo è chiamato spazio di Hilbert e viene rappresentato come uno spazio vettoriale complesso. Il postulato dice:
- **Postulato 1**: Ad ogni sistema fisico isolato è associato uno spazio vettoriale complesso con prodotto interno definito, chiamato spazio di stato del sistema. Il sistema è completamente definito da un vettore chiamato vettore di stato. Il vettore di stato è un vettore unitario nello spazio di Hilbert.

La meccanica quantistica non ci dice, dato un sistema fisico, come trovare lo spazio degli stati, che dipende ovviamente dal sistema fisico che stiamo studiando.

Noi utilizzeremo il più semplice sistema quantistico, il qubit. Il qubit è rappresentato in uno spazio di Hilbert bidimensionale $\mathbb{C}^2$. Utilizzando una notazione passata dai bit classici, poniamo come base ortonormale del nostro sistema qubit i vettori $|0\rangle$ e $|1\rangle$. Possiamo quindi definire un generico stato di un qubit come:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

la condizione che il vettore di stato sia unitario, $\langle\psi|\psi\rangle = 1$, ci impone che α e β siano numeri complessi tali che $|\alpha|^2 + |\beta|^2 = 1$. Questa condizione è chiamata condizione di normalizzazione.

Il qubit differisce dal bit classico per la sua capacità di essere in una sovrapposizione degli stati fondamentali come mostrato sopra. Questa sovrapposizione è una delle caratteristiche più importanti della meccanica quantistica e permette ai computer quantistici di eseguire operazioni che sarebbero impossibili per i computer classici.

Diciamo che una combinazione lineare $\sum_i \alpha_i|\psi_i\rangle$ è una sovrapposizione degli stati $|\psi_i\rangle$ con ampiezze α_i. Quindi per esempio:

$$|\psi\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

è uno stato di qubit in una sovrapposizione di $|0\rangle$ e $|1\rangle$ con ampiezze $\frac{1}{\sqrt{2}}$ e $-\frac{1}{\sqrt{2}}$ rispettivamente.

Il secondo postulato della meccanica quantistica descrive come uno stato $|\psi\rangle$ di un sistema evolve nel tempo. Questo postulato è chiamato postulato di evoluzione temporale e dice:
- **Postulato 2**: L'evoluzione di un sistema quantistico chiuso è descritta da una trasformazione unitaria. Lo stato $|\psi\rangle$ a un tempo t_1 è legato allo stato $|\psi'\rangle$ a un tempo t_2 da un operatore unitario U che dipende solo da t_1 e t_2, tale che:

$$|\psi'\rangle = U|\psi\rangle$$

Come la meccanica quantistica non ci dice come trovare lo spazio degli stati, non ci dice neanche come trovare l'operatore unitario che descrive l'evoluzione temporale del sistema. Quando parliamo di qubit, risulta che qualsiasi operatore unitario può essere realizzato in un sistema reale.

Il secondo postulato richiede che i sistemi siano chiusi, che nella realtà l'unico sistema che può essere considerato chiuso è se prendiamo l'intero universo come sistema. Certi sistemi però possono essere approssimati come chiusi e quindi possiamo utilizzare l'evoluzione temporale unitaria.

Il secondo postulato descrive l'evoluzione temporale di un sistema chiuso in due tempi diversi, introduciamo quindi una versione più generale del postulato che descrive l'evoluzione temporale di un sistema in un intervallo di tempo infinitesimo, continuo. Questo postulato è chiamato postulato di Schrödinger e dice:
- **Postulato 2 (Schrödinger)**: L'evoluzione temporale di uno stato in un sistema chiuso è descritta dall'equazione di Schrödinger.

$$i\hbar\frac{d}{dt}|\psi\rangle = H|\psi\rangle$$

$\hbar$ è la costante di Planck ridotta, il quale valore viene determinato sperimentalmente. H è un operatore hermitiano chiamato Hamiltoniana del sistema. L'Hamiltoniana è una quantità che descrive l'energia totale del sistema e contiene tutte le informazioni necessarie per descrivere l'evoluzione temporale del sistema. In pratica però, risulta difficile costruire l'Hamiltoniana di un sistema reale.

Abbiamo discusso come si evolve un sistema chiuso in uno stato imperturbato dove non interagisce col mondo esterno. Si vuole però capire anche come descrivere un sistema nel momento in cui un agente esterno interagisce con esso e vuole osservare quello che succede.

Questo è il terzo postulato della meccanica quantistica, il postulato di misura. Questo postulato dice:
- **Postulato 3**: Le misure in meccanica quantistica sono descritte da una collezione

$\{M_m\}$ di operatori di misura. Questi operatori agiscono sullo spazio di stato del sistema che viene misurato. L'indice m si riferisce al possibile risultato della misura. Se lo stato del sistema è $|\psi\rangle$ prima della misura, la probabilità di ottenere il risultato m è data da:

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle$$

Dopo la misura, lo stato del sistema è:

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}$$

L'operatore di misura soddisfa la relazione di completezza:

$$\sum_m M_m^\dagger M_m = I$$

L'equazione di completezza esprime il fatto che la somma delle probabilità di ottenere un certo risultato è 1.

$$\sum_m p(m) = 1 = \sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle$$

Come esempio, consideriamo la misura di uno stato di qubit. Un generico stato di qubit è dato da:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

creiamo due operatori di misura M_0 e M_1 tali che:

$$M_0 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Il risultato della misura m può essere $|0\rangle$ con probabilità $|\alpha|^2$ e dopo la misura lo stato del sistema è:

$$\frac{M_0|\psi\rangle}{\sqrt{\langle\psi|M_0^\dagger M_0|\psi\rangle}} = \frac{\alpha|0\rangle}{|\alpha|} = |0\rangle$$

2.1.2 Quantum Bits

Quando si parla di qubit, non è facile visualizzare, come per altri sistemi del mondo reale, il suo comportamento. Possiamo però studiare indirettamente il comportamento di un qubit manipolandolo e trasformandolo in modo che i risultati delle misure siano più facili da interpretare.

Se possiamo interpretare un bit classico come una moneta che può trovarsi in un insieme di due stati separati, testa o croce, il qubit si può trovare invece in un continuo di questi stati. Questo è un concetto difficile da visualizzare, ma possiamo pensare al qubit come una sfera, la sfera di Bloch. La sfera di Bloch è una rappresentazione geometrica di uno stato di qubit.

Partendo dalla condizione di normalizzazione, $|\alpha|^2 + |\beta|^2 = 1$, possiamo riscrivere un generico stato di qubit come:

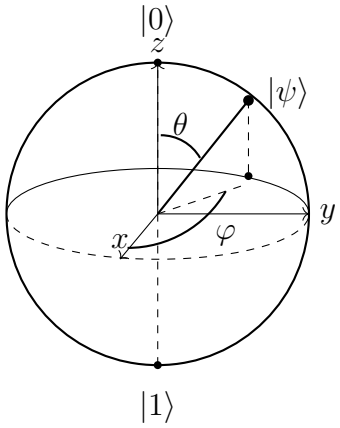
$$|\psi\rangle = e^{i\gamma} \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) |1\rangle$$

dove θ, ϕ e γ sono parametri reali. Il fattore $e^{i\gamma}$ è una fase globale che non ha effetti osservabili sul qubit. Questo ci porta a riscrivere l'equazione sopra come:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) |1\rangle$$

i parametri θ e ϕ definiscono la posizione dello stato di qubit sulla sfera di Bloch. θ è l'angolo tra lo stato $|0\rangle$, che corrisponde al polo nord della sfera nella direzione z, e lo stato $|\psi\rangle$. ϕ è l'angolo tra la proiezione di $|\psi\rangle$ sul piano xy e l'asse x.

La sfera di Bloch è una rappresentazione geometrica di uno stato di qubit e ci permette di visualizzare un qubit singolo. Viene utilizzata per studiare le trasformazioni di qubit e le porte quantistiche.



Verrebbe intuitivamente da pensare che essendo una sfera formata da infiniti punti, si possano memorizzare infinite informazioni in un qubit. Questo è vero, il problema nasce con l'estrazione dell'informazione in quanto come descritto precedentemente la

misurazione di un qubit è distruttiva. Quando misuriamo un qubit, otteniamo un risultato casuale, 0 o 1, con probabilità $|\alpha|^2$ e $|\beta|^2$ rispettivamente. Dopo la misura, lo stato del qubit è proiettato su $|0\rangle$ o $|1\rangle$.

2.1.3 Molteplici qubits

Se avessimo 2 bit classici, potremmo rappresentare 4 stati diversi: 00, 01, 10 e 11. Con due qubit possiamo utilizzare lo stesso ragionamento di prima, creando una base computazionale di 4 stati, $|00\rangle$, $|01\rangle$, $|10\rangle$ e $|11\rangle$. Dato che un qubit può essere in una sovrapposizione di questi stati, aggiungiamo un'ampiezza per ognuno di questi stati. Un generico stato di due qubit è:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

dove α_{00}, α_{01}, α_{10} e α_{11} sono numeri complessi. La condizione di normalizzazione ci dice che la somma dei quadrati delle ampiezze è 1, quindi:

$$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$$

Possiamo decidere di misurare solo uno dei due qubit. Per esempio, la probabilità di ottenere 0 come risultato per il primo qubit è $|\alpha_{00}|^2 + |\alpha_{01}|^2$, che ci lascia con lo stato:

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}$$

Questo concetto è generalizzabile a N qubit. Uno stato generico, in questo caso, è una sovrapposizione di tutti i possibili stati di N qubit e viene descritto come:

$$|\psi\rangle = \sum_{x=0}^{2^N-1} \alpha_x |x\rangle$$

dove x è una rappresentazione binaria di un numero intero tra 0 e $2^n - 1$.

2.2 Porte Quantistiche a singolo qubit

L'informazione, in un circuito classico, viene manipolata e propagata grazie all'utilizzo di cavi e porte logiche. L'unica porta logica non triviale in un circuito classico è la porta NOT, che inverte lo stato di un bit. Questa può essere ricreata in un circuito quantistico con una porta quantistica chiamata porta X. Questa porta quindi cambierà lo stato

di un qubit da $|0\rangle$ a $|1\rangle$ e viceversa. Abbiamo, però, parlato di stati di sovrapposizione, e quindi uno stato di sovrapposizione come $\alpha|0\rangle+\beta|1\rangle$ verrà trasformato in $\alpha|1\rangle+\beta|0\rangle$.

Possiamo quindi rappresentare in forma matriciale la porta X come:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Per visualizzare meglio l'effetto della porta, possiamo scrivere lo stato di prima come un vettore colonna e moltiplicarlo per la matrice X:

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Possiamo quindi evincere che le porte quantistiche sono rappresentate da matrici 2x2 unitarie. La condizione di normalizzazione deve essere mantenuta anche dopo che la porta ha agito sul qubit, quindi la matrice deve rispettare la condizione $U^\dagger U = I$.

Questa è l'unica condizione che deve essere rispettata da una porta quantistica, questo implica che l'esistenza di più porte quantistiche non triviali rispetto a quelle classiche. Una delle porte più importanti è la porta di Hadamard, che permette di creare stati di sovrapposizione. La porta di Hadamard è rappresentata dalla matrice:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

e agisce su uno stato di qubit come:

$$H \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{\alpha+\beta}{\sqrt{2}} \\ \frac{\alpha-\beta}{\sqrt{2}} \end{bmatrix}$$

Possiamo visualizzare meglio gli effetti della porta Hadamard sulla sfera di Bloch, possiamo scomporre il suo effetto in due rotazioni: una rotazione di $\frac{\pi}{2}$ attorno all'asse y e una rotazione di π attorno all'asse x.

Si possono avere infinite porte quantistiche che agiscono su un qubit, considerando che le matrici 2x2 sono infinite. Si può però descrivere in modo arbitrario una porta quantistica a qubit singolo utilizzando 3 matrici di rotazione:

$$U = e^{i\alpha} \begin{bmatrix} e^{\frac{-i\beta}{2}} & 0 \\ 0 & e^{\frac{i\beta}{2}} \end{bmatrix} \begin{bmatrix} \cos\left(\frac{\gamma}{2}\right) & -\sin\left(\frac{\gamma}{2}\right) \\ \sin\left(\frac{\gamma}{2}\right) & \cos\left(\frac{\gamma}{2}\right) \end{bmatrix} \begin{bmatrix} e^{\frac{-i\delta}{2}} & 0 \\ 0 & e^{\frac{i\delta}{2}} \end{bmatrix}$$

dove α, β, γ e δ sono parametri reali. La prima e la terza matrice sono matrici di rotazione attorno all'asse z, mentre la seconda è una matrice di rotazione attorno all'asse y.

2.2.1 Porte quantistiche a più qubit

In computazione classica è stato dimostrato che qualsiasi funzione può essere rappresentata attraverso una porta NAND, che viene quindi chiamata porta universale. In computazione quantistica, la porta universale è la porta CNOT, che è una porta a due qubit. La porta CNOT è formata da un qubit di controllo e un qubit target. La porta inverte lo stato del qubit target solo se lo stato del qubit di controllo è $|1\rangle$. La porta CNOT è rappresentata dalla matrice:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Può essere quindi vista come una generalizzazione della porta XOR classica in quanto possiamo vedere la sua azione come $|A, B\rangle \to |A, A \oplus B\rangle$.

Possiamo vedere come agisce la porta CNOT su uno stato di due qubit:

$$\text{CNOT} \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix} = \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{11} \\ \alpha_{10} \end{bmatrix}$$

Non possiamo direttamente riprodurre la NAND e tradurre tutti i calcoli che faremmo con un computer classico in quanto la NAND non è una porta reversibile. Mentre, come abbiamo ormai detto più volte, ogni operazione su un sistema di qubit può essere descritto come una matrice Unitaria, e la peculiarità di esse è che sono reversibili.

Ma vedremo nei prossimi paragrafi com'è possibile implementare qualunque circuito classico con un computer quantistico.

2.2.2 Circuiti quantistici

Iniziamo spiegando la notazione: i cavi che collegano i circuiti quantistici non rappresentano per forza cavi fisici, possono invece indicare il passaggio del tempo o lo scambio di particelle, come i fotoni. Solitamente, se non indicato, si suppone che i qubit siano tutti preparati nello stato $|0\rangle$ in laboratorio.

Il primo circuito che esamineremo è l'operazione di scambio tra due qubit. Questa operazione è importante in quanto ci permette di scambiare l'informazione tra due qubit. La porta SWAP può essere riprodotta con l'applicazione di 3 CNOT consecutive, la

matrice che rappresenta la porta SWAP può essere quindi ottenuta moltiplicando le matrici delle porte CNOT:

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Possiamo vedere come agisce la porta SWAP su uno stato di due qubit:

$$\text{SWAP} \begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix} = \begin{bmatrix} \alpha_{00} \\ \alpha_{10} \\ \alpha_{01} \\ \alpha_{11} \end{bmatrix}$$

Per continuare con la notazione, i circuiti quantistici spesso hanno bisogno di ritornare degli output classici, questo si fa tramite la misurazione dei qubit. La misurazione di un qubit è una misurazione distruttiva, quindi lo stato del qubit viene distrutto. La misurazione di un qubit è una misurazione proiettiva, quindi il risultato della misurazione è casuale. La probabilità di ottenere un certo risultato è data dalla norma quadra dell'ampiezza del risultato.

A differenza dei circuiti classici, ci sono principalmente 3 operazioni che non sono ammesse in un circuito quantistico. La prima è la possibilità di creare loop, quindi non possiamo avere cavi che ritornano su se stessi. La seconda è la possibilità di FANOUT, quindi non possiamo avere un cavo che si divide in più cavi. La terza è la possibilità di FANIN, quindi non possiamo avere un cavo che si unisce in più cavi.

2.2.3 Computazione classica su un computer quantistico

Prima abbiamo detto che i circuiti quantistici non possono direttamente simulare i circuiti classici in quanto molte porte logiche classiche sono irreversibili. Esiste però una porta logica classica che è reversibile e universale, che ci permette di simulare sia ogni porta logica classica che il FANOUT. Questa porta è la porta Toffoli.

La porta Toffoli è una porta a tre bit, con due bit di controllo e un bit target. La porta inverte lo stato del bit target solo se entrambi i bit di controllo sono 1. la porta Toffoli può essere anche implementata come gate quantistico, questo quindi ci permette di simulare ogni circuito classico su un computer quantistico.

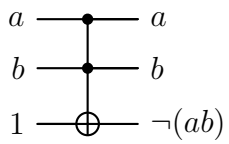
La porta Toffoli è rappresentata dalla matrice:

$$\text{Toffoli} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

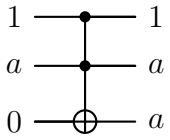
Questo ci permette di simulare ogni circuito classico deterministico su un computer quantistico. Nel caso arrivassimo a scoprire che la computazione classica è non deterministica, allora basterebbe produrre dei numeri casuali per simulare la computazione classica su un computer quantistico. Questo lo possiamo fare semplicemente applicando una porta Hadamard a un qubit inizializzato nello stato $|0\rangle$ e misurando questo avremo i risultati $|0\rangle$ e $|1\rangle$ con probabilità 0.5.[19]

Sapendo nella computazione classica esiste una porta universale, la porta NAND, ci basta dimostrare che possiamo simulare la porta NAND, utilizzando la porta Toffoli, e che possiamo riprodurre un FANOUT, sempre utilizzando la stessa porta, per ottenere che possiamo simulare ogni circuito classico su un computer quantistico.

In particolare, possiamo simulare la porta NAND utilizzando la porta Toffoli come segue:



E possiamo simulare il FANOUT utilizzando la porta Toffoli come segue:



Capitolo 3

Un circuito quantistico per l'imputazione dei dati mancanti

3.1 Introduzione

Adesso mettiamo in pratica i traguardi a cui siamo arrivati, utilizzando sia la computazione quantistica che il machine learning ai fini di tentar di risolvere il problema dei dati mancanti, utilizzando una tecnica innovativa che sfrutta le proprietà di entrambe le strategie.

In particolare, in questa sezione, cercherò di riprodurre i risultati ottenuti nell'articolo [20], implementando il circuito proposto e testandolo sulle stesse distribuzioni di dati.

In suddetto articolo, gli autori utilizzano, ai fini di dimostrare le funzionalità del circuito, due distribuzioni semplici per simulare l'andamento dei dati in un dataset: la distribuzione Gaussiana e la distribuzione di maggioranza.

Il passaggio di processare i dati del dataset in maniera che possano essere descritti utilizzando una funzione di distribuzione è molto comune, in quanto molti metodi di valutazione delle stime del modello si basano sulla distanza o le differenze tra funzioni. Come vedremo, anche in questo caso è stata utilizzata una funzione di distribuzione per valutare l'efficacia del circuito.

Proseguiamo descrivendo le distribuzioni utilizzate e, per essere sicuri di non incappare in errori ancora prima di cominciare a lavorare, verifichiamo che i dati siano stati generati correttamente attraverso un istogramma che ne rappresenti la distribuzione.

3.2 Tecnologie usate

Per l'allenamento e lo sviluppo di modelli di quantum machine learning è necessario l'utilizzo di strumenti computazionali in grado di gestire la complessità delle operazioni coinvolte. In questo progetto sono state utilizzate, quindi, tecnologie come la piattaforma CUDA di NVIDIA, per l'accelerazione delle operazioni di calcolo, e il framework Qiskit, in particolare il modulo Aer, per la simulazione efficiente di circuiti quantistici.

3.2.1 CUDA

CUDA (Compute Unified Device Architecture) è una piattaforma di calcolo parallelo e un API sviluppata da NVIDIA per l'utilizzo delle GPU (Graphics Processing Unit) come processori general-purpose. Questo permette di sfruttare la potenza di calcolo e parallelismo elevato delle GPU per operazioni di calcolo ad alta intensità, come quelle coinvolte nell'allenamento di modelli di machine learning.

CUDA è stato utilizzato per l'accelerazione delle operazioni di calcolo coinvolte nell'allenamento del modello, in modo da scaricare il carico computazionale delle operazioni più intensive sulla GPU, riducendo sostanzialmente i tempi di esecuzione. La natura parallela delle GPU è perfettamente adatta per i requisiti computazionali di modelli di machine learning, che richiedono l'esecuzione di molte e grandi operazioni su vettori e matrici. Utilizzando CUDA sono state, quindi, aumentate le prestazioni e la scalabilità del modello, permettendo di allenare modelli più complessi e di dimensioni maggiori. [21]

3.2.2 Qiskit e Aer

Qiskit è un framework open-source di quantum computing sviluppato da IBM, che permette di programmare e simulare circuiti quantistici. Il modulo Aer, in particolare, serve come simulatore di circuiti quantistici, permettendo di sperimentare e testare algoritmi quantistici in un ambiente controllato senza la necessità di hardware quantistico.

Aer fornisce diversi metodi di simulazione, tra cui il metodo 'statevector', che permette di simulare lo stato quantistico del circuito come un vettore di ampiezza, per riprodurre il comportamento del circuito in un ambiente ideale, privo di rumore e errori. Questo è particolarmente utile per testare e verificare la correttezza di algoritmi quantistici e per studiare le performance teoriche del modello.

Il simulatore Aer supporta l'accelerazione hardware tramite CUDA, permettendo di sfruttare la potenza di calcolo delle GPU per velocizzare le operazioni di simulazione.

Ha permesso, quindi, di ottenere risultati più rapidi e testare modelli più complessi, riducendo i tempi di sviluppo e di test. [22]

3.3 Definizione del problema

Supponiamo di avere una collezione di bitstrings $X \in \{0,1\}^{N+1}$, dove i primi N bit seguono una distribuzione di probabilità $p(X)$ e l'ultimo bit è mancante. Il nostro obiettivo è quello di riprodurre la distribuzione $p(X)$ anche per il bit mancante.

Il circuito che andremo a ricreare applica una serie di rotazioni su un qubit target, che rappresenta il bit mancante, inizializzandolo nello stato $|0\rangle$. Per ogni stato di input $|n\rangle|0\rangle$, dove $|n\rangle$ è una bitstring di lunghezza N, numero di qubit, l'output del circuito può essere descritto come:

$$|n\rangle(cos\theta_n|0\rangle + sin\theta_n|1\rangle)$$

dove θ_n è un parametro che dipende dalla distribuzione $p(X)$. Più precisamente abbiamo che le probabilità di ottenere 0 o 1 sono rispettivamente:

$$p(0|n) = cos^2\theta_n$$

$$p(1|n) = sin^2\theta_n$$

Per rappresentare ogni possibile distribuzione di probabilità servirebbe un numero esponenziale di parametri rispetto al numero di qubit, 2^N.
Come abbiamo visto dalla teoria, possiamo descrivere qualsiasi circuito quantistico utilizzando una matrice unitaria U, in particolare il nostro circuito utilizza i qubit di input come controlli ed applica delle rotazioni sul qubit target.
La matrice unitaria U è definita come segue:

$$U(\Theta) = \bigoplus_{n=0}^{2^N-1} R_y(\theta_n)$$

con $\Theta = (\theta_0, \theta_1, \ldots, \theta_{2^N-1})$ e dove $R_y(\theta_n)$ è una rotazione di angolo θ_n attorno all'asse y, meglio definita come:

$$R_y(\theta_n) = \begin{pmatrix} cos\frac{\theta_n}{2} & -sin\frac{\theta_n}{2} \\ sin\frac{\theta_n}{2} & cos\frac{\theta_n}{2} \end{pmatrix}$$

$U(\Theta)$ può essere riprodotto utilizzando una serie di C_mNOT gates, dove $m = 1, 2, \ldots, N$ qubit agiscono come controlli, ogni controllo è associato ad una rotazione $R_y(\theta_n)$.

Calcolando il numero di possibili gate utilizzabili con N qubit in input, otteniamo che il numero di parametri necessari per descrivere il circuito è $\sum_{m=1}^{N} \binom{N}{m} = 2^N - 1$. Per

recuperare il parametro 2^N possiamo creare una rotazione extra parametrica.

Creare un circuito del genere richiede un numero esponenziale di porte, che rende il circuito molto complesso e difficile da ottimizzare. Per questo sono state proposte due alternative che utilizzano un numero lineare e un numero quadratico di porte.
Per l'esattezza viene proposto un circuito lineare che utilizza $\sum_{m=0}^{1} \binom{N}{m} = N + 1$ porte, e un circuito quadratico che utilizza $\sum_{m=0}^{2} \binom{N}{m} = \frac{N^2+N+2}{2}$ porte.
In questo caso gli indici m iniziano da 0 in quanto contiamo già per la rotazione finale.

3.4 Stato di output

Dato che il circuito ha una struttura ben definita, si può recuperare un espressione analitica per gli angoli θ_n come una funzione di rotazioni $R_y(\alpha_i)$ con $i = 1, 2, ..., M$, dove M è il numero di porte del circuito.
Nel circuito lineare possiamo descrivere la forma della matrice unitaria U come:

$$U = \bigoplus_b U_b$$

dove $b = \{b_1, b_2, ..., b_N\}$ è l'insieme di tutte le possibili combinazioni di bit di lunghezza N.
Ogni blocco U_b è una matrice unitaria 2×2 che agisce sul qubit target, e può essere scritta come:

$$U_b = X^{S_N} R_y(\theta_b)$$

dove $S_N = \sum_{i=1}^{n} b_i \mod 2$ è una somma parziale delle bitstring b, e $\theta_b = \alpha_0 + \sum_{n=1}^{N} (-1)^{\sum_{j=1}^{n} b_j} \alpha_n$ è un angolo che dipende dalle rotazioni $R_y(\alpha_i)$.

Per il circuito quadratico si può scrivere la matrice unitaria come:

$$U_q = X^{S_N + Q_{N-1,N}} \times R_y \left[\sum_{n=1}^{N} \left[(-1)^{S_n} \theta_n + \sum_{m=n+1}^{N} (-1)^{Q_{n,m}+S_n} \alpha_{n,m} \right] \right]$$

dove $Q_{N-1,N} = \sum_{i=1}^{N} \sum_{j=i+1}^{N} b_i b_j \mod 2$ è una somma parziale delle coppie di bit $b_i b_j$, e $\alpha_{n,m}$ è un angolo che dipende dalle rotazioni $R_y(\alpha_i)$.

La matrice unitaria del circuito esponenziale si può scrivere come:

$$U_q = X^{E_1(N) + E_2(N-1,N) + \cdots + E_N(1,...,N)}$$

$$\times R_y \left[\sum_{n_1=1}^{N} (-1)^{E_1} \theta_{n_1} + \sum_{n_2=n_1+1}^{N} (-1)^{E_2} \alpha_{n_1,n_2} + [\dots \right.$$
$$\left. + \sum_{n_N=n_{N-1}+1}^{N} (-1)^{E_N} \alpha_{n_1,\dots,n_N} \right] \dots \Bigg]$$

dove $E_1, \dots, E_n$ sono delle fasi definite da:

$$E_j(n_1, \dots, n_j; q) = \sum_{a_1=1}^{n_1} \cdots \sum_{a_j=a_{j-1}+1}^{n_j} b_{a_1} \cdots b_{a_j} \mod 2$$

e gli angoli $\alpha_{n_1}, \dots, \alpha_{n_1,\dots,n_N}$ definiscono altrettante rotazioni.

3.5 Distribuzioni

Nell'articolo la distribuzione Gaussiana è definita come segue:

$$\rho(n, 0) = \frac{1}{\sqrt{2^N}} \frac{1}{\sqrt{2\pi}} e^{-(n-\frac{N-1}{2})^2}$$

$$\rho(n, 1) = (1 - \rho(n, 0))$$

n è la bitstring, che rappresenta le possibili combinazioni di N bit, dove N è il numero di qubit. In particolare, questa distribuzione suggerisce la probabilità che data una certa combinazione di bit il risultato sia 0 o 1. E' una distribuzione gaussiana, normalizzata da un fattore $\frac{1}{\sqrt{2^N}}$ in quanto dobbiamo mantenere la probabilità generale uguale a 1.

La distribuzione di maggioranza assegna al qubit target il valore più frequente tra quelli in input:

$$p(n, x) = \begin{cases} \frac{1}{\sqrt{2^N}} & \text{se } f_x(n) > f_{\bar{x}}(n), \\ \frac{1}{2\sqrt{2^N}} & \text{se } f_x(n) = f_{\bar{x}}(n), \\ 0 & \text{altrimenti,} \end{cases}$$

dove $\bar{x} = x \oplus 1$

3.6 Hellinger Distance

Utilizziamo una funzione che ci permette di valutare la differenza tra la distribuzione target in input, creata a partire dal dataset originale omettendo i dati mancanti, e la distribuzione che otteniamo come output del circuito, che deve essere il più simile possibile alla distribuzione target.

Nell'articolo è stata scelta la Hellinger Distance, in quanto è semplice, veloce da calcolare ed è una funzione monotona di altre distanze utilizzate comunemente come la distanza di Bhattacharyya e di Jensen-Shannon.

La Hellinger Distance prende, quindi, in input le due distribuzioni e ne calcola una semplificata distanza euclidea, definita come segue:

$$d_H(p,q) = \sqrt{1 - \sum_x \sqrt{p(x)q(x)}}$$

il termine $\sum_x \sqrt{p(x)q(x)}$ è chiamato coefficiente di Bhattacharyya, che misura la sovrapposizione tra le due distribuzioni in input, in questo caso p e q.

Nel nostro caso, quindi p sarà una distribuzione nella forma $p(n,a)$ definita come la distribuzione gaussiana, o di maggioranza, sopra esplicitate.
La distribuzione q, invece, è quella generata dallo stato di uscita del circuito quantistico, che dipende dagli angoli di rotazione θ_n .
La distanza di Hellinger tra queste due specifiche distribuzioni è data dalla formula:

$$d_H(\phi_T, \psi(\bar{\theta})) = \sqrt{1 - |\langle \phi_T | \psi(\bar{\theta}) \rangle|} = \sqrt{1 - \sum_{n=0}^{2^N-1} \frac{\cos(\theta_n - \bar{\theta}_n)}{2^N}}$$

dove ϕ_T è la distribuzione target, $\psi(\bar{\theta})$ è la distribuzione generata dal circuito quantistico. In questo caso θ_n è l'angolo target associato all'input n e $\bar{\theta}_n$ è l'angolo di rotazione effettivamente realizzato dal circuito.

L'obiettivo è quindi quello di minimizzare la distanza di Hellinger tra le due distribuzioni, ovvero minimizzare la funzione obiettivo:

$$\min_{\bar{\theta}} d_H(\phi_T, \psi(\bar{\theta}))$$

Possiamo calcolare un upper bound per la distanza di Hellinger, calcolata partendo da una distribuzione target approssimata a solo $M < 2^N$ parametri, come:

$$\max_{|\phi_T\rangle} min_{\bar{\theta}} d_H(\phi_T, \psi(\bar{\theta})) = \sqrt{1 - \frac{M}{2^N}}$$

Questo risultato indica che la distanza più difficile da approssimare ci lascierà al massimo una distanza di $\sqrt{1 - \frac{M}{2^N}}$.

3.7 Ottimizzazione

Per ottenere migliori risultati durante un processo di addestramento, solitamente vengono utilizzate delle tecniche di ottimizzazione che, in questo caso serviranno a trovare i parametri da inputare nel circuito in maniera tale da minimizzare la Hellinger Distance tra la distribuzione target e quella ottenuta come output del circuito.

In particolare, nella sezione classica del circuito si vanno ad imputare direttamente i parametri ottimali seguendo i calcoli analitici degli angoli θ_n riportati nelle sezioni precedenti, con successivi tentativi di ottimizzazione utilizzando l'ottimizzatore SLSQP.

SLSQP è un ottimizzatore che utilizza un metodo di programmazione quadratica sequenziale per risolvere problemi di ottimizzazione non lineare con vincoli di uguaglianza e disuguaglianza. [23]

Nella sezione quantistica del circuito, inizialmente avevo provato ad utilizzare un metodo gradient-free, che quindi non richiede la valutazione della derivata della funzione obiettiva, COBYLA, era stato scelto poichè resiliente al rumore del circuito. Tuttavia, dopo lunghe sessioni di test, ho visto che dava comunque risultati migliori l'ottimizzatore SLSQP, che è stato quindi utilizzato per l'ottimizzazione dei parametri del circuito quantistico, in accordo con l'articolo. [24]

3.8 Analisi dei risultati

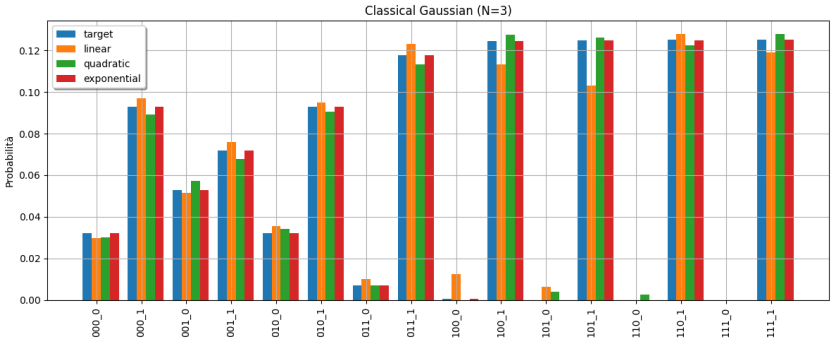
Per avere un confronto visibile, ho deciso di emulare classicamente il circuito quantistico, seguendo la descrizione analitica dello stato di output.
L'idea è quella di calcolare direttamente gli angoli θ_n partendo da un gruppo di parametri inizialmente randomici, di dimensionalità variabile in base al tipo di circuito che si vuole emulare.
Una volta calcolati gli angoli la probabilità di output del circuito classico sarà data da:

$$p_{n,0} = \cos^2(\theta_n)$$

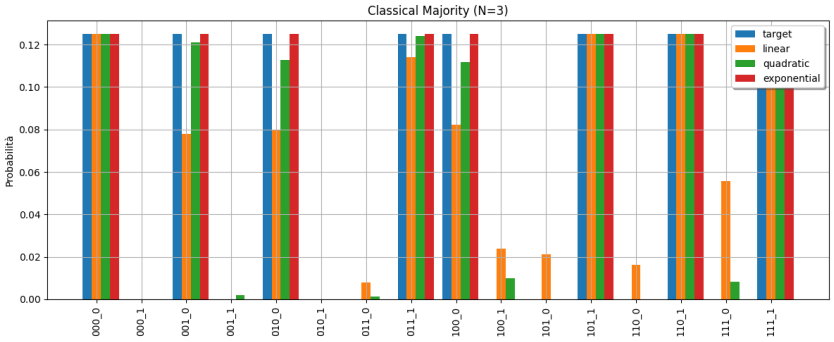
$$p_{n,1} = \sin^2(\theta_n)$$

che corrispondono alla probabilità di misurare lo stato $|0\rangle$ e $|1\rangle$ rispettivamente.
Successivamente verrà calcolata la distanza di Hellinger partendo dai risultati ottenuti e seguirà la parte di ottimizzazione.

Nel secondo metodo, invece, il circuito quantistico viene effettivamente simulato tramite l'utilizzo di AerSimulator.
Per approfondire, al circuito quantistico vengono inputate le singole bitstring n e dei parametri iniziali α randomici, che vanno a definire le rotazioni condizionate che avverranno sul qubit di output.
I primi N qubit di input vengono messi in uno stato di sovrapposizione utilizzando una porta di Hadamard, mentre l'ultimo qubit viene inizializzato nello stato $|0\rangle$.
I circuiti per il caso lineare e quadratico sono mostrati nell'appendice dell'articolo. [20]
Infine, viene misurato il qubit di output. Questo processo viene effettuato per ogni bitstring n per creare la distribuzione di probabilità $\psi(\bar{\theta})$.
Come descritto prima, anche per il circuito classico, in base alla distanza di Hellinger calcolata, vengono ottimizzati i parametri iniziali e si ripete il processo finchè l'ottimizzatore non converge.



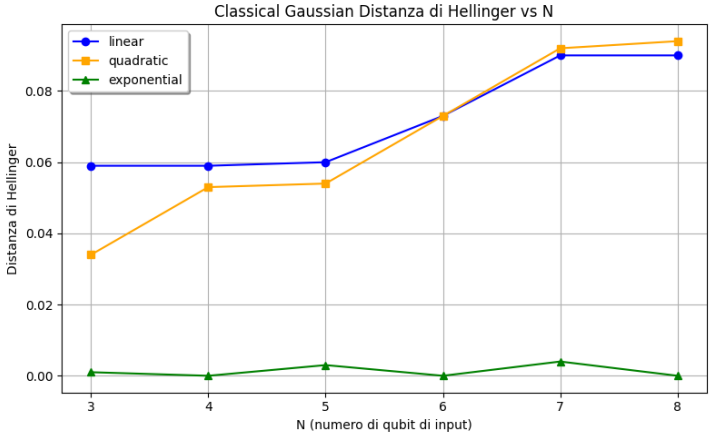
(a)



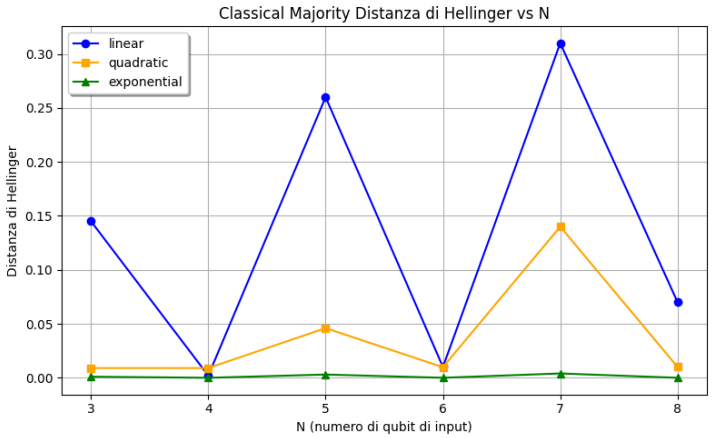
(b)

Figura 3.1: (a) Grafico rappresentativo della differenza tra la distribuzione target Gaussiana e le distribuzioni ottenute dalle simulazioni classiche degli stati di output dei circuiti lineare, quadratico ed esponenziale. (b) Grafico rappresentativo della differenza tra la distribuzione target di Maggioranza e le distribuzioni ottenute dalle simulazioni classiche degli stati di output dei circuiti lineare, quadratico ed esponenziale. Analizzati su $N = 3$, i circuiti esponenziale e quadratico simulano più precisamente le due distribuzioni, mentre quello lineare ha più difficoltà con la distribuzione di Maggioranza.

Come si può evincere dai grafici in figura 3.1, gli algoritmi che utilizzano un numero di parametri più elevato sono più efficaci, come ci aspettavamo dal momento che ci permettono di descrivere meglio la distribuzione target.
In particolare, l'algoritmo con un numero esponenziale di parametri riesce a riprodurre perfettamente entrambe le distribuzioni.
Ho inoltre provato a vedere all'aumentare del numero di qubit come cambia l'efficacia dell'algoritmo. Ciò è mostrato in figura 3.2.



(a)



(b)

Figura 3.2: (a) Variare della distanza di Hellinger, tra la distribuzione target Gaussiana e le simulazioni classiche dei circuiti lineare, quadratico ed esponenziale, all'aumentare di N da $N = 3$ a $N = 8$. (b) Variare della distanza di Hellinger, tra la distribuzione target di Maggioranza e le simulazioni classiche dei circuiti lineare, quadratico ed esponenziale, all'aumentare di N da $N = 3$ a $N = 8$.

In teoria l'efficacia dell'algoritmo dovrebbe aumentare all'aumentare del numero di qubit in quanto si ha una maggiore quantità di informazioni a disposizione. Questo, però, non è sempre vero, come si può notare dai grafici 3.2. Questo può essere dato da diversi fattori: la complessità dell'input può portare ad un paesaggio di ottimizzazione più complesso risultando in minimi locali, oppure semplicemente non aumentando troppo il numero di parametri nei casi lineare e quadratico si è ancora in un numero di parametri basso dove l'ottimizzazione può dare qualche errore di regolazione.
Una curiosità che ho notato è che nella distribuzione di maggioranza, con un numero

di qubit pari, entrambi gli algoritmi riproducono meglio la distribuzione. Penso che ciò sia dato dalla definizione stessa della distribuzione, in quanto si crea la possibilità di avere un numero di 0 e 1 uguale, causando una voluta ottimizzazione verso la media.

Successivamente ho implementato il circuito quantistico, per il quale ho cercato di ricreare grafici simili a quelli ottenuti con l'algoritmo classico, riportati in figura 3.3

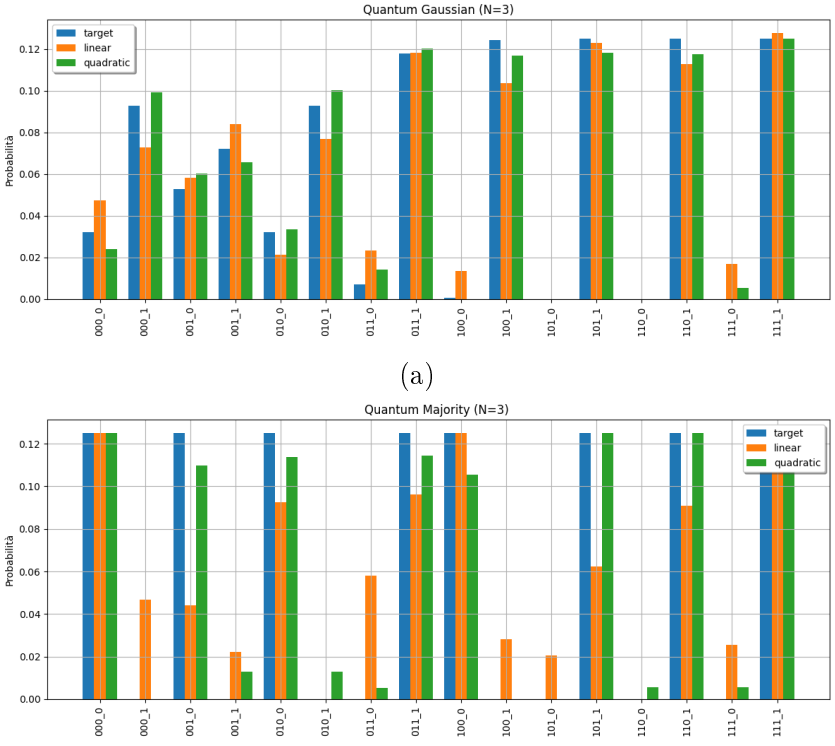
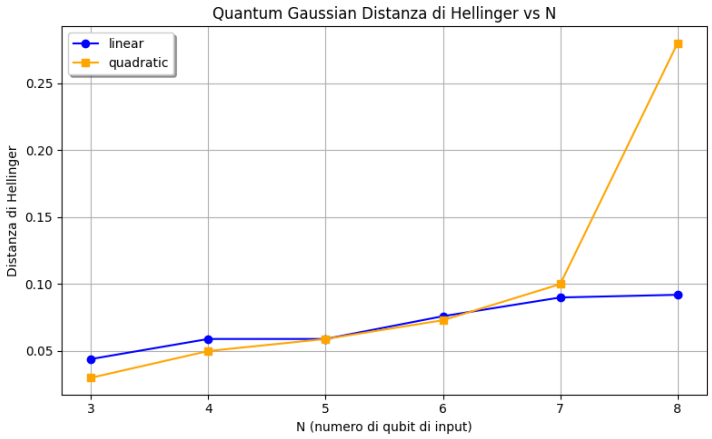


Figura 3.3: (a) Grafico rappresentativo della differenza tra la distribuzione target Gaussiana e le distribuzioni ottenute dalle simulazioni quantistiche dei circuiti lineare e quadratico. (b) Grafico rappresentativo della differenza tra la distribuzione target di Maggioranza e le distribuzioni ottenute dalle simulazioni quantistiche dei circuiti lineare e quadratico. Analizzati su $N = 3$, replicano abbastanza bene i risultati ottenuti con le simulazioni classiche.

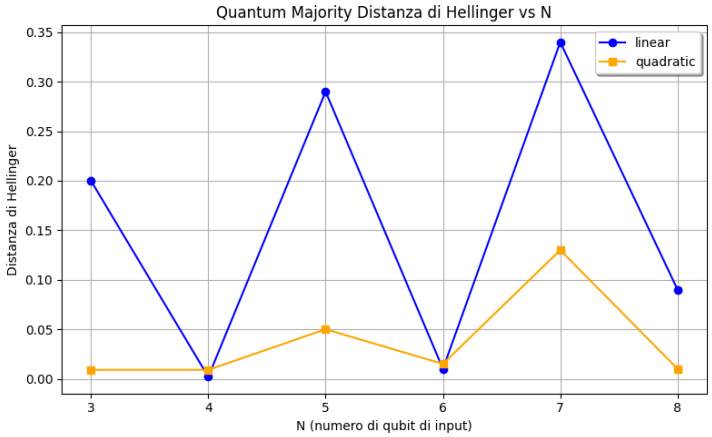
Il circuito simulato quantisticamente, risulta meno efficace del circuito simulato classicamente per struttura, in quanto quello classico agisce direttamente su delle probabilità, mentre quello quantistica applica rotazioni condizionate dagli input e misura un singolo qubit. Questo schema, realizza di fatto una funzione θ_n degli angoli di rotazione e combina $cos^2\theta_n$ e $sin^2\theta_n$ per ottenere la probabilità di ottenere 0 o 1. Per distribuzioni,

come la majority, ci potrebbe essere bisogno di più termini di controllo per ottenere una distribuzione netta.

Anche nel caso del simulatore quantistico, ho esaminato vari numeri di qubit, come mostrato in figura 3.4.



(a)



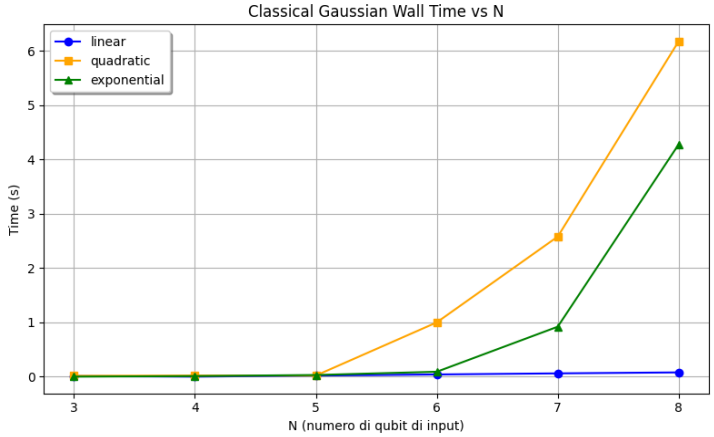
(b)

Figura 3.4: (a) Variare della distanza di Hellinger, tra la distribuzione target Gaussiana e le simulazioni quantistiche dei circuiti lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$. (b) Variare della distanza di Hellinger, tra la distribuzione target di Maggioranza e le simulazioni quantistiche dei circuiti lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$.

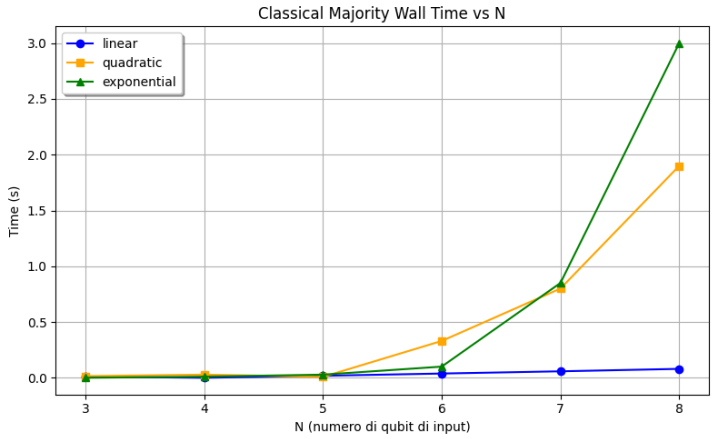
Come per la simulazione classica, all'aumentare del numero di qubit si nota un appiattimento della funzione che descrive il variare della distanza di Hellinger all'aumentare del numero di qubit. Questo fenomeno è tipico nei circuiti variazionali quantistici, noto come Barren Plateau. Più precisamente è una regione nello spazio dei parametri in cui il gradiente della funzione di costo tende a diventare esponenzialmente piccolo all'aumentare del numero di qubit. Questo implica che l'ottimizzazione diventa estremamente difficile poichè le variazioni della funzione di costo sono trascurabili, impedendo un apprendimento efficace.

Ho voluto, inoltre, mantenere traccia del wall time e del cpu time per vedere se ci fossero differenze significative tra gli algoritmi.
Il wall time è il tempo di orologio, ovvero il tempo che trascorre tra l'inizio e la fine di un processo, mentre il cpu time è il tempo di CPU, ovvero il tempo che la CPU impiega per eseguire un processo.

L'andamento di tali tempi è mostrato, per i vari casi, nelle figure 3.5 – 3.8.

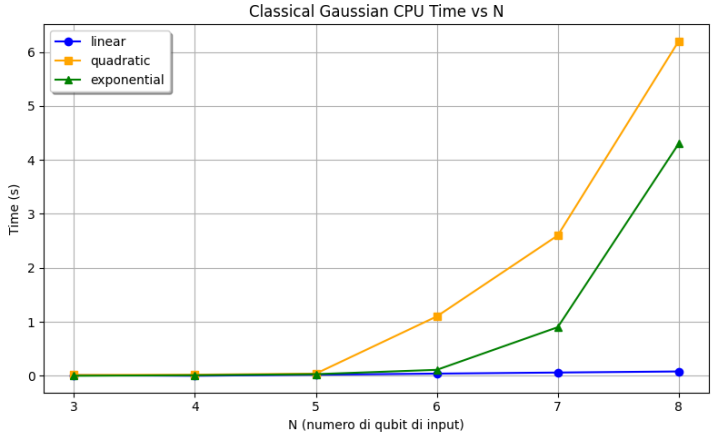


(a)

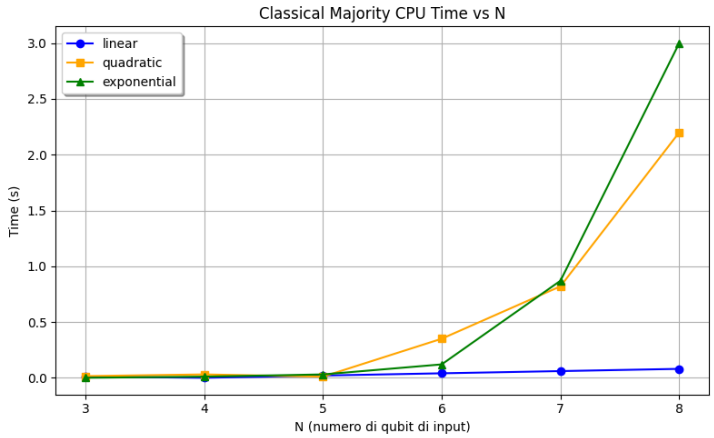


(b)

Figura 3.5: (a) Wall time per la simulazione classica del circuito lineare, quadratico ed esponenziale, all'aumentare di N da $N = 3$ a $N = 8$. (b) Wall time per la simulazione quantistica del circuito lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$.

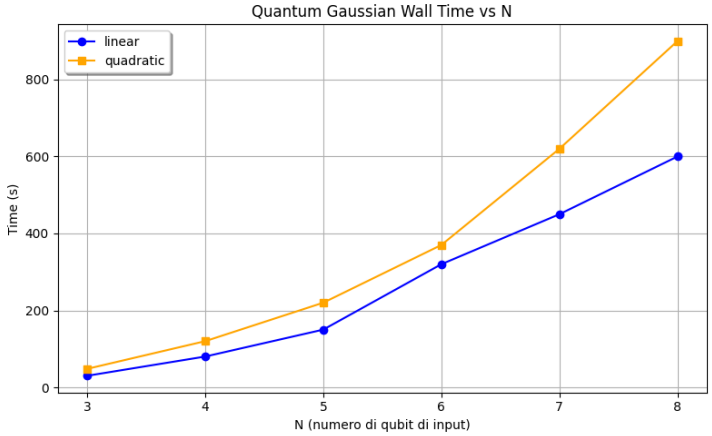


(a)

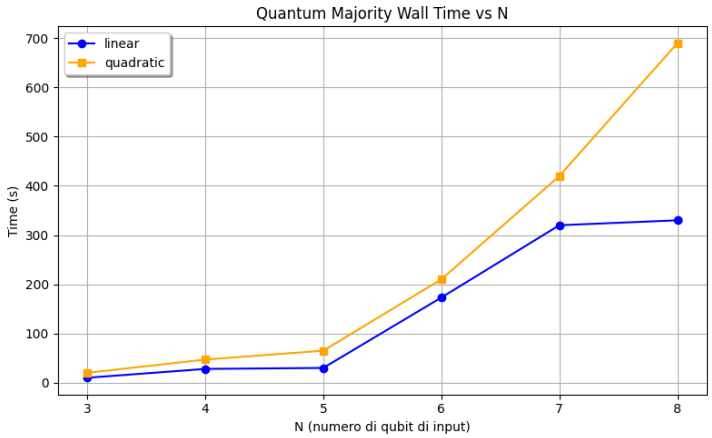


(b)

Figura 3.6: (a) Cpu time per la simulazione classica del circuito lineare, quadratico ed esponenziale, all'aumentare di N da $N = 3$ a $N = 8$. (b) Cpu time per la simulazione quantistica del circuito lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$.

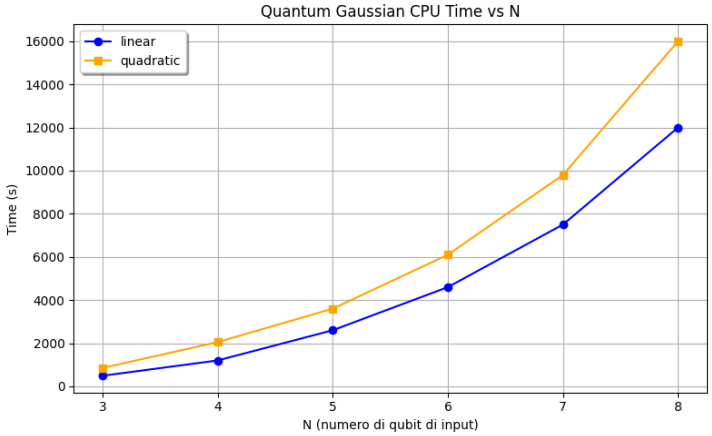


(a)

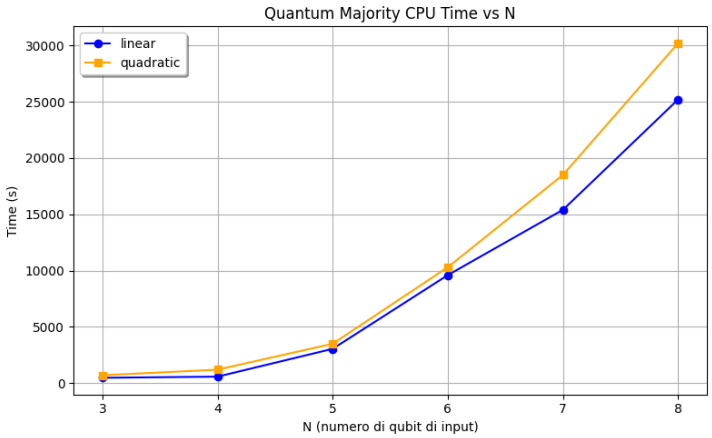


(b)

Figura 3.7: (a) Wall time per la simulazione quantistica del circuito lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$. (b) Wall time per la simulazione quantistica del circuito lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$.



(a)



(b)

Figura 3.8: (a) Cpu time per la simulazione quantistica del circuito lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$. (b) Cpu time per la simulazione quantistica del circuito lineare e quadratico, all'aumentare di N da $N = 3$ a $N = 8$.

E' interessante notare come il wall time e il cpu time per la simulazione classica misurano praticamente lo stesso tempo, invece, per la simulazione quantistica, il cpu time è molto più basso del wall time. Secondo la definizione data precedentemente, potrebbe risultare controintuitivo che il cpu time possa risultare minore del wall time, tuttavia, quello che vado a calcolare è una somma sul tempo per ogni core del processore. Inoltre, nella simulazione classica, soprattutto della distribuzione di maggioranza, il CPU time cresce proporzionalmente al numero di qubit, seguendo un andamento che riflette la struttura del circuito utilizzato: lineare per il circuito lineare, quadratico per quello quadratico ed esponenziale per quello esponenziale. Questo comportamento è dovuto al numero di parametri, che aumenta rispettivamente in modo lineare, quadratico o esponenziale rispetto al numero di qubit.

Capitolo 4

Applicazione del circuito quantistico a un dataset reale

4.1 Introduzione

In questa sezione, andrò a riutilizzare il circuito quantistico presentato nel capitolo 3 per testarlo su un dataset reale. Per poi confrontare i risultati ottenuti con quelli dei metodi classici più famosi.

4.1.1 Dataset

Il dataset utilizzato è un famoso dataset su cui vengono testati molti algoritmi di machine learning, il dataset Iris.
E' stato scelto in quanto il dataset contiene solo 150 istanze di piante di iris, divise in 3 classi di 50 istanze ciascuna: iris setosa, iris versicolor e iris virginica. Ogni istanza contiene 4 features: lunghezza e larghezza del sepalo e del petalo. Questo fa del dataset una maniera semplice di testare algoritmi di classificazione, per cui si trovano molti esempi con cui confrontare i risultati. [25]

4.2 Implementazione del dataset

Il dataset Iris è composto da valori reali in centimetri di lunghezza e larghezza del sepalo e del petalo, che non posso essere direttamente utilizzati come input del circuito quantistico descritto nello scorso capitolo. Per questo motivo, si è deciso di binarizzare i dati, ossia di trovare un modo per tradurre i valori reali in valori binari. Inizialmente, avevo deciso di utilizzare 4 bit per ogni feature, riuscendo a suddividere i valori in intervalli di

lunghezza 0.5 centimetri, tuttavia risultava troppo complesso e dispendioso in termini di risorse avendo $2^{16} = 65536$ possibili combinazioni. Per questo motivo, ho deciso di utilizzare solo 2 bit per ogni feature, suddividendo i valori in intervalli di lunghezza di circa 0.9 in quanto il valore massimo e minimo per ogni feature è 7.9 e 4.3 rispettivamente. In questo modo, il numero di possibili combinazioni è $2^8 = 256$, un numero più gestibile. [26]

I dati binarizzati dovranno andare a comporre una distribuzione di probabilità empirica. Questa distribuzione sarà utilizzata, come in precedenza, per ottimizzare i parametri del circuito attraverso, nuovamente, la minimizzazione della distanza di Hellinger.
Più precisamente prendo i dati in tuple [sepal length, sepal width, petal length, petal width] dal dataset e dopo aver calcolato un ratio tra il valore della feature e il valore minimo della feature, moltiplico il risultato per 4. In questo modo ottengo un numero tra 0 e 4, di cui prendo la parte intera che, convertita in binario, andrà a comporre la stringa binaria che rappresenta la feature.
Per esempio prendiamo la prima tupla del dataset Iris, che ha valori $[5.1, 3.5, 1.4, 0.2]$. Sappiamo che il range della feature 'sepal length' va da 4.3 a 7.9, ossia è di 3.6, quindi calcoliamo il ratio:

$$\frac{5.1 - 4.3}{7.9 - 4.3} = \frac{0.8}{3.6} \approx 0.222$$

Moltiplicando per 4 otteniamo $[0.888] = 0$, che convertito in binario diventa 00 che comporrà le prime due posizioni della stringa binaria.

Ottenute le stringhe binarie per ogni feature, calcolo banalmente il numero di volte che ogni stringa binaria compare nel dataset creando una distribuzione di probabilità.

4.3 Modello generativo

Il circuito quantistico viene simulato su tutte le bitstring possibili in maniera che riproduca le probabilità:

$$p(n, x) = \frac{1}{2^N} p(\text{output} = x | \text{input} = n)$$

dove $\frac{1}{2^N}$ è un parametro di normalizzazione su tutte le possibili bitstring, N numero di bit, e $p(\text{output} = x|\text{input} = n)$ è la probabilità condizionata di ottenere l'output x dato l'input n con n un bitstring specifico.
Successivamente, vengono sommate le probabilità di ogni bitstring su ogni output:

$$p(n) = p(n, 0) + p(n, 1)$$

In questo modo, si ottiene una mappa di probabilità che il circuito ha imparato a riprodurre, simile a quella empirica.

Per vedere quanto il circuito sia riuscito ad imparare la distribuzione di probabilità, mascheriamo un sottoinsieme di campioni, circa il 5% impostandoli a NaN, Not a Number.
Allora discretizziamo i dati, binarizzandoli come descritto in precedenza, creando un dizionario dei dati noti che mappa la posizione del bit al valore del bit. L'obiettivo è quindi quello di campionare i bit mancanti seguendo la distribuzione di probabilità appresa dal circuito.
Definiamo un insieme di tutti i bitstring compatibili:

$$\mathcal{C} = \{n \in \{0 \dots 2^N - 1\} : \forall (k, b) \in \text{known\_bits}, ((n >> k)\&1) = b\}$$

dove known_bits è il dizionario dei dati noti. In altre parole, $\mathcal{C}$ è il sottoinsieme di tutti i bitstring i cui bit in posizione k è uguale al valore b se (k, b) è nel dizionario dei dati noti.
La probabilità di un bitstring compatibile n è data da:

$$p*(n) = \frac{p(n)}{\sum_{n' \in \mathcal{C}} p(n')} \text{ per } n \in \mathcal{C}$$

Quindi campioniamo un bitstring compatibile n con probabilità $p*(n)$.
In questo caso, il circuito è stato utilizzato come un modello generativo: se i bit noti indicano "petalo lungo" e "sepalo corto", il circuito cercherà di generare un bitstring che rispetti queste condizioni.

Ottenuto il bitstring completo, lo traduciamo nuovamente in valori reali, in modo da ottenere un dataset che non presenta più valori mancanti.

4.4 Valutazione del modello

Avendo utilizzato il circuito quantistico come modello generativo, ho voluto, per ottenere una valutazione dell'accuratezza delle predizioni, allenare un classificatore, in questo caso un Support Vector Machine (SVM), sul dataset generato.[27] [28]
Questo ci serve per capire se le feature ricostruite, con dei valori fittizi, sono coerenti con la struttura di Iris.
Il classificatore è stato allenato sul dataset generato e testato sul dataset originale.
Di seguito riporto delle tabelle che mostrano i risultati ottenuti, dove le classi sono del dataset sono state etichettate come 0, 1 e 2, che corrispondono rispettivamente a

"setosa", "versicolor" e "virginica".

Tabella 4.1: Classification report con imputazione lineare

Classe	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.94	1.00	0.97	16
1	1.00	0.89	0.94	18
2	0.92	1.00	0.96	11
Accuracy		0.96 (45 campioni tot.)		
MSE		0.04		
RMSE		0.21		
Macro avg	0.95	0.96	0.96	45
Weighted avg	0.96	0.96	0.96	45

Dalla tabella si può notare che utilizzando il circuito di imputazione lineare, il classificatore ha ottenuto un'accuratezza del 96%.
In generale, i risultati come mostrati dalle medie dei risultati delle metriche pesate sul numero di campioni, support, per classe e anche non pesate, sono molto buoni.
Questo ci fa capire che il circuito è riuscito a generare delle feature che sono coerenti con la struttura di Iris.

Tabella 4.2: Classification report con imputazione quadratica

Classe	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.88	0.94	0.91	16
1	1.00	0.83	0.91	18
2	0.77	0.91	0.83	11
Accuracy		0.89 (45 campioni tot.)		
MSE		0.24		
RMSE		0.49		
Macro avg	0.88	0.89	0.88	45
Weighted avg	0.90	0.89	0.89	45

Come mostrato in tabella 4.2, il circuito di imputazione quadratica, invece, è meno accurato rispetto a quello lineare, con un'accuratezza del 89%.
La spiegazione di questo comportamento potrebbe essere dovuto proprio all'introduzione di parametri in più da parte del circuito quadratico, che potrebbero aver introdotto del rumore nei dati o in generale non aver avuto una quantità di dati sufficiente, trattandosi di un dataset piccolo, per poter creare dei parametri che siano coerenti con la struttura

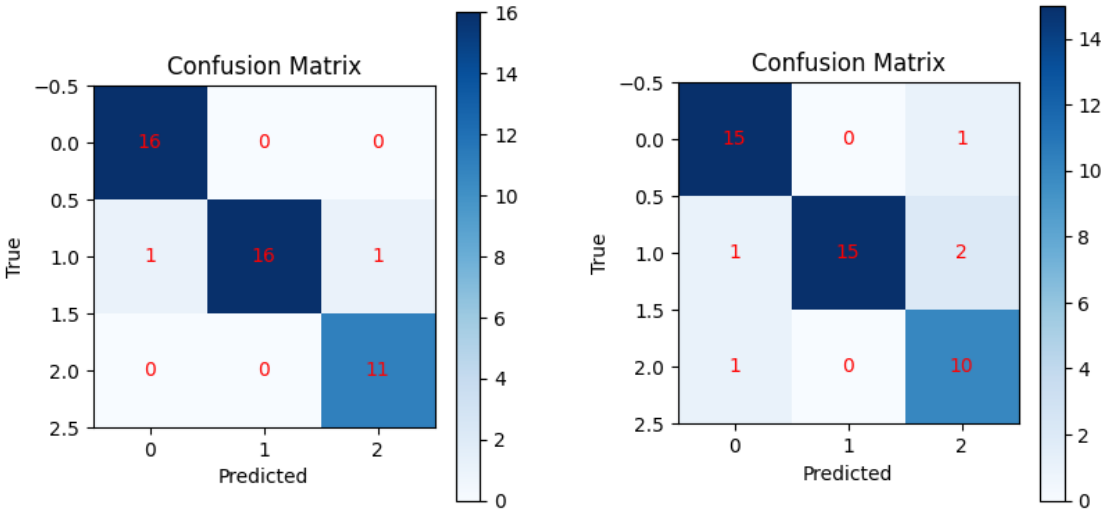
di Iris.
Entrambi i circuiti hanno utilizzato come ottimizzatore di parametri SLSQP, con lo stesso numero di iterazioni e di ripetizioni. In un dataset così piccolo, era quasi scontato che il circuito lineare avrebbe ottenuto risultati migliori, poichè con lo stesso potere di calcolo deve trovare meno parametri.

Per comprovare questa teoria, ho deciso di testare nuovamente il circuito quadratico aumentando, rispetto al circuito lineare, il numero di iterazioni e di ripetizioni, questo è effettivamente risultato in un sostanziale miglioramento dell'accuratezza, arrivando a 92%. Mostro questi risultati in tabella 4.3.

Tabella 4.3: Classification report con imputazione quadratica con più iterazioni e ripetizioni

Classe	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.94	0.94	0.94	16
1	1.00	0.83	0.91	18
2	0.79	1.00	0.88	11
Accuracy		0.92 (45 campioni tot.)		
MSE		0.15		
RMSE		0.39		
Macro avg	0.91	0.92	0.91	45
Weighted avg	0.93	0.92	0.92	45

Un ulteriore metodo per visualizzare i risultati ottenuti è quello di utilizzare la matrice di confusione.



(a) (b)

Figura 4.1: (a) Matrice di confusione per il classificatore allenato sul dataset ricostruito con il circuito lineare. (b) Matrice di confusione per il classificatore allenato sul dataset ricostruito con il circuito quadratico. $0, 1, 2$ corrispondono rispettivamente a "setosa, versicolor, virginica" su entrambi gli assi, mentre i numeri all'interno delle celle rappresentano il numero di campioni. Per esempio, nei riquadri $(0, 1)$ e $(2, 1)$ della figura (a) ci sono rispettivamente 1 e 1 campioni che dovevano essere classificati come "versicolor" ma sono stati classificati come "setosa" e "virginica".

Sulla diagonale principale della matrice di confusione, si possono vedere i campioni che sono stati classificati correttamente. Tutte le osservazioni fuori dalla diagonale principale sono state classificate in maniera errata.

4.5 Confronto con metodi classici

Ho deciso di ricercare degli articoli che trattassero lo stesso esatto problema, così da confrontare ed analizzare i risultati ottenuti con altre metodologie.
In particolare l'articolo [29] riportava l'accuracy ottenuta dall'allenamento di due differenti classificatori, un Support Vector Machine e un Decision Tree, su dei datasets nei quali i dati erano stati imputati utilizzando metodi di imputazione sia statistici che di machine learning.
Riporto di seguito i risultati mostrati dall'articolo.

Tabella 4.4: Confronto con metodi classici

Metodo	**SVM**	**Decision Tree**
Original Dataset	0.96	0.94
Missing Dataset	0.89	0.89
LD-based Imputation	0.91	0.91
Imputed-KNN Dataset	0.94	0.95
Imputed-SKNN Dataset	0.95	0.97

E' sorprendente vedere come il circuito quantistico, nella maggior parte dei casi abbia ottenuto risultati migliori di metodi classici riconosciuti per il loro utilizzo in compiti di imputazione di dati.

Un altro articolo [16] calcola l'RMSE per valutare l'accuratezza dei metodi di imputazione utilizzati, anche qua come metodi vengono scelti KNN e Random Forest. Tale accuratezza varia i risultati in base alla percentuale di dati mancanti, ma in generale il circuito quantistico ha ottenuto risultati migliori rispetto ai metodi classici.

Tabella 4.5: Confronto con metodi classici

Metodo	**KNN**	**Random Forest**
5% Missing Data	0.68	0.65
10% Missing Data	0.24	0.28
15% Missing Data	0.19	0.26

E' interessante vedere come pur aumentando le percentuali di dati mancanti, aumenti l'accuratezza di certi circuiti. Questo l'ho ritrovato anche in altri articoli, dove viene interpretato come la casualità, specifica al caso di imputazione con KNN, nella scelta dei seed per l'estrazione dei dati, i quali in certi casi si potevano trovare più lontani

dalla media e quindi causare un errore più grande. [30]

Altri studi portano percentuali di accuratezza utilizzando KNN e degli imputatori iterativi basati su modelli di regressione Random Forest, ottenendo risultati comunque inferiori rispetto al circuito quantistico, rispettivamente del 0.86 e del 0.89. [31]

Come possiamo vedere il metodo classico utilizzato più spesso per l'imputazione di dati mancanti è KNN, i risultati che esso ottiene variano molto in base alla percentuale di dati mancanti e in generale, presumo, alle diverse implementazioni di esso. Risulta comunque, in quasi la totalità dei casi, inferiore al circuito quantistico proposto.

Conclusioni

In questa tesi si è cercato di dimostrare come la computazione quantistica possa essere utilizzata per risolvere problemi di imputazione di dati mancanti, utilizzando un circuito che sfrutta le proprietà della computazione quantistica di lavorare su stati di sovrapposizione.
I risultati sperimentali, ottenuti sia su distribuzioni sintetiche sia sul dataset Iris, dimostrano che il modello proposto è in grado di approssimare le distribuzioni di probabilità associate ai dati mancanti e di fornire imputazioni competitive, o in altri casi più accurate, rispetto a metodi classici come KNN e Random Forest.
Il circuito proposto è una rivisitazione di un Quantum Circuit Born Machine(QCBM), che è un modello di machine learning quantistico generativo creato per apprendere e riprodurre distribuzioni di probabilità. Si basa sul concetto della legge di Born, che afferma che la probabilità di trovare un sistema quantistico in uno stato $|x\rangle$ è data dal modulo quadro dell'ampiezza di probabilità α_x:

$$p(x) = |\alpha_x|^2$$

Utilizzata per generare delle ampiezze di probabilità utilizzate per campionare nuovi dati.
Nell'ambito dell'imputazione dei dati mancanti il QCBM offre un approccio innovativo rispetto ai metodi tradizionali di machine learning, grazie alla sua capacità di rappresentare e campionare distribuzioni ad alta dimensionalità. Può potenzialmente fornire imputazioni più accurate in un dataset con strutture complesse o correlazioni non lineari.
L'evoluzione dei dispositivi quantistici di nuova generazione, unita allo sviluppo di algoritmi di ottimizzazione ibrida classico/quantistica sempre più sofisticati, potrebbe aprire nuovi scenari, consentendo di gestire in modo efficace dataset di dimensioni sempre maggiori e strutture di dipendenza più complesse. Con l'avanzamento delle tecnologie quantistiche e l'aumento della coerenza dei qubit, modelli come il QCBM potrebbero diventare strumenti fondamentali per affrontare problemi computazionalmente intensivi, tra cui l'imputazione dei dati mancanti, offrendo soluzioni più efficienti e precise rispetto agli approcci classici.

In prospettiva, l'integrazione di tecniche di quantum machine learning con metodologie consolidate per la gestione dei dati mancanti rappresenta un'interessante frontiera di ricerca, capace di offrire strumenti più potenti per l'analisi di big data in ambito scientifico, finanziario e industriale.

In conclusione, il lavoro qui descritto costituisce un passo verso l'utilizzo concreto dei circuiti quantistici per la ricostruzione dei dati mancanti e la modellazione di distribuzioni probabilistiche complesse. Sarà interessante, in futuro, verificare la scalabilità di tali algoritmi su hardware quantistici, senza dover ricorrere alla simulazione, e integrare strategie di riduzione del noise per migliorare la robustezza delle soluzioni proposte.

Bibliografia

[1] Wikipedia contributors. "Missing data." *Wikipedia, The Free Encyclopedia.* Disponibile su: https://en.wikipedia.org/wiki/Missing_data

[2] Wikipedia contributors. "Dataset." *Wikipedia, The Free Encyclopedia.* Disponibile su: https://it.wikipedia.org/wiki/Dataset

[3] Jonathan A. C. Sterne, Ian R. White, John B. Carlin, Michael Spratt, Patrick Royston, Michael G. Kenward, Angela M. Wood, James R. Carpenter. "Multiple Imputation for missing data in epidemiological and clinical research: potential and pitfalls." *PubMed Central.* Disponibile su: https://pmc.ncbi.nlm.nih.gov/articles/PMC2714692/

[4] Yiran Dong, Chao-Ying Joanne Peng. "Principled missing data methods for researchers." *PubMed Central.* Disponibile su: https://pmc.ncbi.nlm.nih.gov/articles/PMC3701793/

[5] Hyung Kang. "The prevention and handling of the missing data." *PubMed Central.* Disponibile su: https://pmc.ncbi.nlm.nih.gov/articles/PMC3668100/

[6] William J. Montelpare, Emily Read, Teri McComber, Alyson Mahar, Krista Ritchie. "Working with Missing Data." *Applied Statistics in Healthcare Research.* Disponibile su: https://pressbooks.library.upei.ca/montelpare/chapter/working-with-missing-data/

[7] Chanelle J. Howe, Lauren E. Cain, Joseph W. Hogan. "Are all biases missing data problems?" *PubMed Central.* Disponibile su: https://pmc.ncbi.nlm.nih.gov/articles/PMC4643276

[8] Wikipedia contributors. "Listwise deletion." *Wikipedia, The Free Encyclopedia.* Disponibile su: https://en.wikipedia.org/wiki/Listwise_deletion

[9] Wikipedia contributors. "Interpolation." *Wikipedia, The Free Encyclopedia.* Disponibile su: https://en.wikipedia.org/wiki/Interpolation

[10] Wikipedia contributors. "Imputation (statistics)." *Wikipedia, The Free Encyclopedia*. Disponibile su: https://en.wikipedia.org/wiki/Imputation_(statistics)

[11] Sara Brown. "Machine learning, explained." *MIT Sloan School*. Disponibile su: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[12] Wikipedia contributors. "Model selection." *Wikipedia, The Free Encyclopedia*. Disponibile su: https://en.wikipedia.org/wiki/Model_selection

[13] IBM. "What is Machine Learning?" *IBM*. Disponibile su: https://www.ibm.com/think/topics/machine-learning/

[14] Wikipedia contributors. "Machine learning." *Wikipedia, The Free Encyclopedia*. Disponibile su: https://en.wikipedia.org/wiki/Machine_learning

[15] Alireza Amirteimoori, Sohrab Kordrostami. "A Euclidean distance-based measure of efficiency in data envelopment analysis." *Taylor & Francis Online*. Disponibile su: https://www.tandfonline.com/doi/abs/10.1080/02331930902878333

[16] Tlamelo Emmanuel, Thabiso Maupong, Dimane Mpoeleng, Thabo Semong, Banyatsang Mphago, Oteng Tabona. "A survey on missing data in machine learning." *Taylor & Francis Online*. Disponibile su: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00516-9/

[17] Nicholas J. Horton, Stuart R. Lipsitz, Michael Parzen. "A Potential for Bias When Rounding in Multiple Imputation." *Taylor & Francis Online*. Disponibile su: https://www.tandfonline.com/doi/abs/10.1198/0003130032314

[18] Pure Storage. "What are machine learning performance metrics?" *Pure Storage*. Disponibile su: https://www.purestorage.com/knowledge/machine-learning-performance-metrics.html

[19] Michael A. Nielsen, Isaac L. Chuang. "Quantum Computation and Quantum Information." *Quantum Computation and Quantum Information*. Disponibile su: https://profmcruz.wordpress.com/wp-content/uploads/2017/08/quantum-computation-and-quantum-information-nielsen-chuang.pdf

[20] Claudio Sanavio, Simone Tibaldi, Edoardo Tignone, Elisa Ercolessi. "Quantum Circuit for Imputation of Missing Data." *arXiv*. Disponibile su: https://arxiv.org/abs/2405.04367

[21] Nvidia Corporation. "CUDA Education & Training." *Nvidia*. Disponibile su: https://developer.nvidia.com/cuda-education-training

[22] Qiskit Development Team. "Simulator." *Qiskit*. Disponibile su: https://qiskit.github.io/qiskit-aer/tutorials/1_aersimulator.html

[23] Andrew D. Wendorff, Emilio Botero, Juan J. Alonso. "Comparing Different Off-the-Shelf Optimizers' Performance in Conceptual Aircraft Design." *Stanford University*. Disponibile su: https://www.emiliobotero.com/publications/opt_compare.pdf

[24] Altair MotionSolve. "Optimization Search Methods." *Altair MotionSolve*. Disponibile su: https://2022.help.altair.com/2022/hwsolvers/ms/topics/solvers/ms/optimization_search_methods_r.htm

[25] Wikipedia contributors. "Dataset Iris." *Wikipedia, The Free Encyclopedia*. Disponibile su: https://it.wikipedia.org/wiki/Dataset_Iris

[26] ARCCA Training Material. "Scikit Learn - The Iris Dataset." *Scikit Learn*. Disponibile su: https://arcca.github.io/An-Introduction-to-Machine-Learning-Applications/03-scikit-learn-iris-dataset/index.html

[27] Scikit Learn. "Plot different SVM classifiers in the iris dataset." *Scikit Learn*. Disponibile su: https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html

[28] Scikit Learn. "SVC." *Scikit Learn*. Disponibile su: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[29] Syed Tahir Hussain Rizvi et al. "Analysis of Machine Learning Based Imputation of Missing Data." *Taylor & Francis Online*. Disponibile su: https://www.tandfonline.com/doi/full/10.1080/01969722.2023.2247257

[30] Analytics Vidhya. "Effectiveness of KNN Imputation, Part I: The Iris Dataset." *Medium*. Disponibile su: https://medium.com/analytics-vidhya/effectiveness-of-knn-imputation-part-i-the-iris-dataset-e784f157275a

[31] Muller A. "Missing Values" *GitHub*. Disponibile su: https://amueller.github.io/aml/01-ml-workflow/08-imputation.html