Università degli Studi di Milano Bicocca
Dipartimento di Fisica
Corso di Laurea Triennale

Projected Quantum Support Vector Machine per classificazione di pattern di rumore

Relatore:
Prof. Andrea Giachero

Correlatore:
Dr. Roberto Moretti

Candidata:
Erika Giacomantonio
Matricola: 859667

ANNO ACCADEMICO 2022/2023

Indice

Sommario

La meccanica quantistica è un teoria sviluppata tra gli anni ’20 e ’30 che ha rivoluzionato la Fisica grazie alla sua capacità di descrivere fenomeni su scala atomica e subatomica con un’accuratezza senza precedenti. Negli anni ’80 il fisico Richard Feynman intuì le potenzialità di sfruttare sistemi quantistici, manipolandone l’evoluzione temporale, per risolvere efficacemente problemi complessi, come per esempio la simulazione di interazioni a molti corpi, unendo la meccanica quantistica alla scienza dell’informazione, fornendo così le basi del quantum computing. Il quantum computing costituisce oggi un settore di ricerca in rapida espansione, quest’ultima alimentata da continui progressi tecnologici, teorici e dallo sviluppo di nuove strategie di calcolo. L’unità fondamentale su cui si basano i computer quantistici moderni sono i qubit, i quali sono sistemi a due livelli energetici differenti definiti $|0\rangle$ e $|1\rangle$. Il presente lavoro di tesi si inserisce in una promettente linea di ricerca del quantum computing nota come Quantum Machine Learning (QML), controparte quantistica del Machine Learning (ML). Il ML è un ramo dell’informatica e dell’intelligenza artificiale che si concentra sullo sviluppo di modelli addestrati per risolvere un’ampia varietà di problemi, tra cui classificazione, regressione, e generazione di dati. Lo studio si focalizza sullo sviluppo di un approccio quantistico al Support Vector Machine (SVM) un algoritmo di apprendimento supervisionato largamente utilizzato per risolvere problemi di classificazione, ossia il Quantum Support Vector Machine (QSVM). Il SVM si fonda sul concetto di kernel, una funzione che quantifica la similitudine tra coppie di elementi da classificare all’interno di un dataset. Determinare la funzione di kernel ottimale per un dato problema di classificazione è un aspetto cruciale che determina l’accuratezza di classificazione del modello finale. La differenza tra SVM e QSVM consiste nella funzione di kernel, la quale nel secondo caso viene calcolata con tecniche quantistiche e prende il nome di quantum kernel. Lo studio si propone di sviluppare e confrontare due tipologie di quantum kernel, quantificandone l’impatto sull’accuratezza del modello. La prima tipologia, la più popolare, è considerata lo standard nello sviluppo di QSVM e prende il nome di Fidelity Quantum Kernel. Quest’ultima è particolarmente esposta ad un problema noto come exponential concentration, per il quale all’aumentare dei qubit utilizzati, il valore della funzione di kernel si avvicna progressivamente ad una costante, ostacolando l’apprendimento del QSVM. La seconda tipologia, nota come Projected Quantum Kernel, si propone di

mitigare questo problema tramite operazioni di riduzione dello spazio di Hilbert a cui lo stato finale del sistema a multi-qubit appartiene. In questa tesi, il QSVM si applica alla classificazione di pattern di rumore di due diversi computer quantistici di tipo superconduttivo. Il dataset contiene un insieme di distribuzioni di stati finali calcolati da due distinti processori, soggetti a diverse tipologie di errore. L'interesse verso questo dataset è motivato dal fatto che l'identificazione ed in ultima istanza la correzione dell'errore costituisce un'ulteriore branca del quantum computing, chiamata error-mitigation. Dopo aver valutato e accuratamente ottimizzato i QSVM, confrontando le prestazioni del modello con SVM classici, i risultati ottenuti evidenziano come SVM con diversi kernel classici esibiscono un'accuratezza di classificazione compresa tra il 75% e il 90%. Sullo stesso dataset, i QSVM testati raggiungono accuratezze confrontabili coi migliori SVM ottenuti (90%). La mitigazione dell'exponential concentration è stata valutata confrontando la varianza dei valori della funzione di kernel valutata sugli elementi del dataset. Rsipetto al Fidelity quantum kernel, la varianza ottenuta con il Projected quantum kernel migliora significativamente per un numero di qubit superiore ad otto. La varianza aumenta circa del 26% per quantum kernel a otto qubit, del 36% a dieci qubit, 34% a dodici qubit, evidenziando il successo del metodo projected. Questo studio mostra come il Projected quantum kernel sia una valida soluzione al problema dell'exponential concentration. Lo studio suggerisce inoltre, su un dataset ridotto per motivi di efficienza di calcolo, che l'accuratezza di QSVM con projected quantum kernel sia paragonabile sia a QSVM con Fidelity quantum kernel che ai migliori SVM con kernel classico. In un prossimo futuro, il QML potrebbe fornire nuovi ed efficienti algoritmi di analisi dati, affiancando o sostituendo i modelli di intelligenza artificiale classica già esistenti.

Capitolo 1

Machine Learning

In questo capitolo viene fornita un'introduzione al mondo del machine learning classico, introducendo i concetti fondamentali di un algoritmo chiamato Support Vector Machine e della sua implementazione tradizionale. Questo capitolo pone le basi del lavoro di tesi, dedicato all'implementazione della controparte quantistica del SVM conosciuta come Quantum Support Vector Machine (QSVM).

1.1 Introduzione al machine learning

Il machine learning (ML) è un ramo dell'informatica e dell'intelligenza artificiale che si concentra sull'utilizzo di algoritmi, in grado di apprendere informazioni dai dati stessi, i quali vengono addestrati per fare classificazioni o previsioni. [17]

Il Machine Learning deriva dai progressi nello studio dell'intelligenza artificiale intrapresi dalla seconda metà del secolo scorso. Uno dei primi a menzionare la possibilità della sua nascita fu Alan Turing nell'articolo del 1950 riguardante il Test di Turing [26]. Ai giorni nostri il machine learning ha numerose applicazioni in varie aree scientifiche tra cui per esempio le scienze biomediche, le scienze ambientali e la finanza. Come già asserito nel 2001 in [27], esso gioca un ruolo importante nel supporto della ricerca scientifica e nello sviluppo tecnologico. Nella fisica il Machine Learning ha numerose applicazioni poiché permette di elaborare grandi quantità di dati, per esempio ha giocato un ruolo importante nell'analisi dei dati nella fisica delle alte energie [9], inoltre un settore in rapida espansione riguarda l'applicazione dei computer quantistici al machine learning con lo scopo di potenziarne modelli già esistenti o crearne di nuovi più efficienti sfruttando l'informazione quantistica [24].

Le operazioni che si possono svolgere col machine learning sono di due tipi:

- regressione: L'obiettivo è predire o conoscere l'output che in questo caso è continuo.

- classificazione: L'obiettivo è dividere in diverse classi, l'output in questo caso è discreto.

La base di lavoro è il dataset. Il dataset è un insieme ordinato di dati che l'algoritmo di ML deve elaborare. Gli elementi distintivi di un dataset generalmente sono i seguenti:

- un pattern, ossia un campione di dati.
- le feature, che sono caratteristiche associate ad ogni pattern.
- le label, che sono delle variabili binarie o categoriche che assegnano una classe ad ogni pattern. Non tutti i dataset le possiedono.

Il dataset viene suddiviso in training set, validation set e test set. Il training set è generalmente composto dal 70% del dataset ed è utilizzato per allenare l'algoritmo; il validation set viene utilizzato per ottimizzare gli iperparametri del modello, che sono per definizione i parametri di un modello ML che si ottimizzano durante il training; il test set viene utilizzato per quantificare l'attendibilità del modello ed è composto da dati che non vengono utilizzati né nella fase di training né nella fase di validation.

1.2 Tipologie di machine learning

Le tipologie di machine learning si suddividono in 3 categorie principali:

- Supervised machine learning: è il tipo di tecnica trattata in questo elaborato in quanto il dataset a disposizione (descritto nella sezione 3.1) presenta già delle label. Il fatto che presenti delle label indica che il dataset è provvisto del target di output, quindi, in seguito alla fase di training dove i pattern presentano le label, al modello durante il test vengono dati in input campioni sprovvisti di label ed esso li confronta con gli esempi forniti dal set di training in modo tale da prevedere quella corretta da assegnare [23]. Uno degli algoritmi utilizzati per questo tipo di tecnica è il Support Vector Machine (SVM). Siccome è il tipo di tecnica che è stata utilizzata per svolgere l'analisi dati di questo elaborato verrà descritta più nel dettaglio successivamente nella sezione 1.2.1.
- Unsupervised machine learning: in questo caso si utilizzano algoritmi di machine learning per analizzare set di dati privi di label. Questa tipologia permette di raggruppare dati simili fra di loro o di scoprire correlazioni nascoste nelle informazioni senza la necessità di un intervento umano. Un approccio comune per questo tipo di tecnica è la K-means clustering, cui scopo è quello di raggruppare in classi dati sprovvisti di label. Inizialmente l'algoritmo prende in input il numero k di raggruppamenti, ai quali viene assegnato in maniera casuale un

centro, successivamente si calcola la distanza quadratica media di ogni punto da esso e in seguito il centro del gruppo si sposta verso i campioni più vicini a quest'ultimo, questo processo si itera fino a che il centro non si può più spostare verso un raggruppamento di dati meno distante [3].

- Semi-supervised machine learning: in questa tecnica durante il training si utilizza un set di dati provvisto di labels di dimensioni ridotte per guidare la classificazione e l'estrazione di caratteristiche da un set di dati privo di labels di dimensione maggiore.

Esiste anche una tecnica simile al supervised machine learning chiamata reinforcement machine learning, la differenza dal supervised sta nel fatto che in questo caso l'algoritmo non viene addestrato utilizzando dei dati di training ma il modello apprende le informazioni tramite prove ed errori [17].

1.2.1 Supervised Machine Learning

Come già introdotto nella sezione 1.2 il Supervised machine learning è una tecnica di ML in cui il dataset che si ha a disposizione è provvisto delle label nella fase di addestramento, quindi si hanno a disposizione sia gli input $\vec{x} \in \mathbb{R}^N$ che i corrispettivi target di output $y \in \mathbb{Z}$. In questo approccio lo scopo è quello di trovare una funzione che metta in relazione i pattern con le rispettive label, quindi:

$$f(\vec{x}) = y \tag{1.1}$$

Nella pratica in realtà si cerca di approssimare il più possibile la funzione (1.1) agendo su dei parametri $\vec{P}$ in modo tale da ottenere una funzione del tipo:

$$f(\vec{x}, \vec{P}) = y \tag{1.2}$$

Per ogni elemento del training set è possibile definire una funzione detta Loss Function $L(y, f(\vec{x}, \vec{P}))$ che fornisce una misura della differenza tra l'output fornito dall'algoritmo e quello atteso. Successivamente si può ottenere una funzione detta funzione di rischio facendo una media su tutti gli elementi del training set, supponendo di averne N si ottiene:

$$R(\vec{P}) = \frac{1}{N} \sum_{k=1}^{N} L(y, f(\vec{x}, \vec{P})) \tag{1.3}$$

Quindi la Loss function si riferisce all'errore di classificazione dell'algoritmo sul singolo evento mentre la funzione di rischio fa una media di questo errore sull'intero dataset.

Gli algoritmi di Supervised machine learning si dividono in due classi principali:

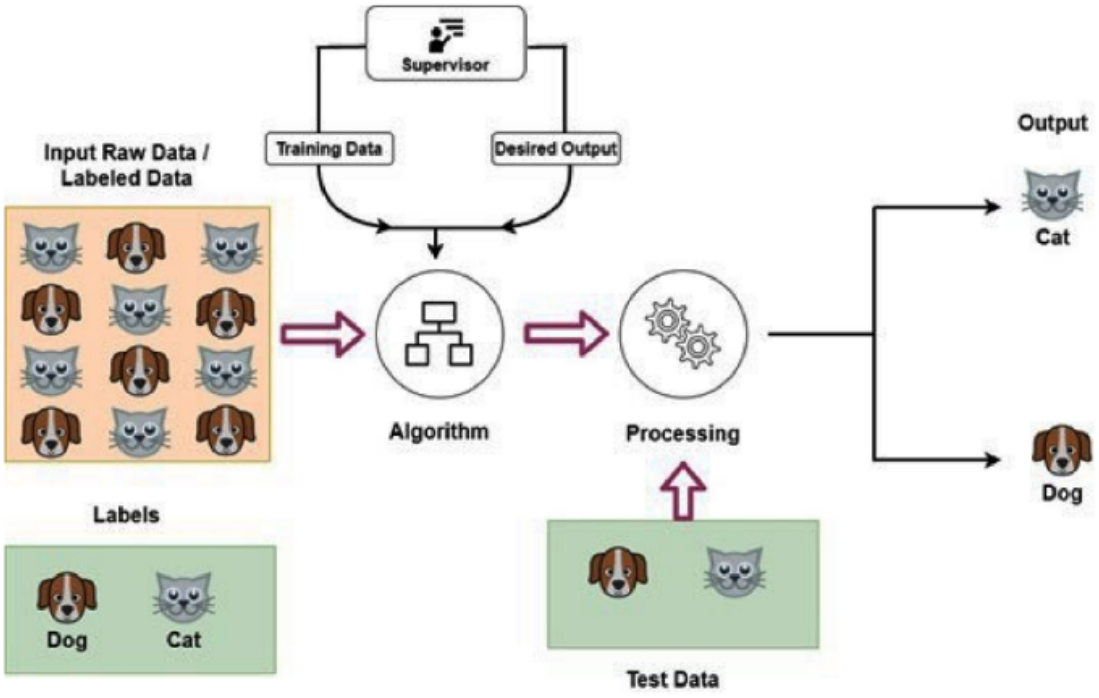

Figura 1.1: Schema logico del funzionamento del supervised machine learning.

- algoritmi di classificazione: alcuni esempi sono il Support Vector Machine, gli alberi decisionali e il K-nearest neighbour.
- algoritmi di regressione: alcuni esempi sono la regressione lineare e la regressione polinomiale [13].

Quando si addestra un modello si cerca sempre di evitare il problema dell'overfitting. L'overfitting è ciò che avviene quando il modello ha un eccessivo adattamento al set di training e ciò comporta la costruzione di una funzione (1.2) che classifica con un'elevata accuratezza il set di training ma non fa altrettanto col test set [9]. Per ovviare questo problema si possono attuare diverse strategie come per esempio aumentare i campioni del dataset e ridurre i parametri addestrabili del modello.

1.3 Support Vector Machine

I Support vector machines (SVMs) sono un set di algoritmi per tecniche di supervised machine learning usati per problemi di classificazione, regressione e per l'identificazione di outlier [12], furono sviluppati dalla comunità informatica negli anni '90 e la loro popolarità non ha ancora smesso di crescere. I SVMs si sono mostrati molto efficienti e vengono utilizzati principalmente nel caso di classificazioni binarie, quindi

quando sono presenti due leabels, (come nel caso di questo elaborato) ma possono essere utilizzati anche in presenza di più labels.

Per dataset in cui le feature sono numeriche ad ogni pattern è assegnato un punto nello spazio delle feature e il SVM permette la classificazione di un dato a seconda della regione in cui trova rispetto ad un iperpiano. L'iperpiano è scelto in maniera tale da separare il più possibile il set di training nelle due classi, quindi si devono massimizzare i "margini" che sono la distanza tra l'iperpiano e i dati più vicini ad esso. Nel caso in cui i dati siano separabili linearmente trovare l'iperpiano è abbastanza banale, nel caso in cui invece non siano separabili linearmente si definisce una funzione detta kernel, la quale verrà discussa più nel dettaglio nella sezione 1.3.1, che permette di passare dallo spazio delle features corrente ad uno di dimensione maggiore. [25]

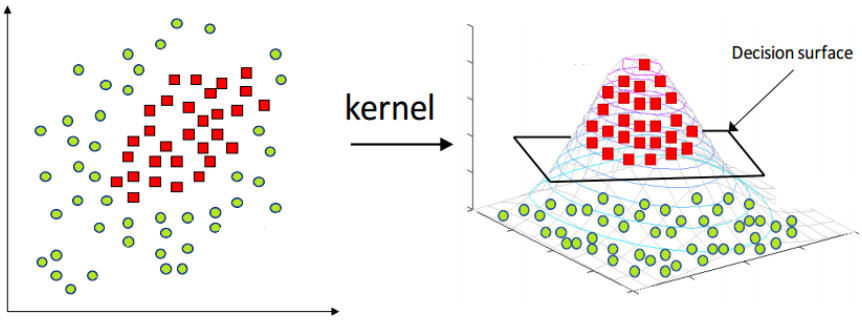

Figura 1.2: a sinistra le due classi di dati diverse non sono separabili linearmente ma passando ad uno spazio di dimensione maggiore tramite il kernel le classi diventano separabili linearmente come a destra.

Nel caso di classificazione binaria il problema viene formalizzato nel seguente modo: Per classificare si deve stimare una funzione $f : \mathbb{R}^N \longrightarrow \{\pm 1\}$ dato un training set N-dimensionale definito nel seguente modo:

$$T = \{(\vec{x_1}, y_1), \ldots, (\vec{x_N}, y_N)\} \in \mathbb{R}^N \times \{\pm 1\} \tag{1.4}$$

dove $\vec{x_i} \in \mathbb{R}^N$ è il vettore delle features dell'i-esimo punto e $y_i \in \{\pm 1\}$ è la sua label. La funzione classificherà correttamente dei nuovi campioni $(\vec{x_i}, y_i)$ generati dalla stessa distribuzione di probabilità $P(\vec{x_i}, y)$ del set di training [16].
Sia $\phi : \mathbb{R}^N \longrightarrow F$ la mappa che permette il passaggio dallo spazio delle features iniziale ad uno spazio delle features F di dimensione maggiore.
Un iperpiano può essere definito dall'equazione:

$$\vec{w}^T \cdot \phi(\vec{x_i}) - b = 0, w \in \mathbb{R}^N, b \in \mathbb{R} \tag{1.5}$$

Lo scopo è quello di trovare l'iperpiano che massimizzi i margini definiti:

$$M = \min_{1 \leq i \leq n} y_i(\vec{w}^T \cdot \phi(\vec{x}_i) - b) \tag{1.6}$$

L'iperpiano coi margini massimizzati si ottiene risolvendo il seguente problema di ottimizzazione:

$$\max_{\vec{w},b} M \tag{1.7}$$

$$\sum_{j=1}^{N} w_j^2 = 1 \tag{1.8}$$

$$y_i(\vec{w}^T \cdot \phi(\vec{x}_i) - b) \geq M, \ \forall i = 1...N \tag{1.9}$$

così la funzione f di classificazione è definita come:

$$f(\vec{x}) = sign(\vec{w}^T \cdot \phi(\vec{x}_i) - b) \tag{1.10}$$

Se il segno di $y_i f(\vec{x}_i)$ è positivo allora il dato è stato correttamente classificato altrimenti la classificazione è errata.
Questo metodo è chiamato "maximal margin classifier" e si utilizza quando esiste un iperpiano che separi completamente le classi. Tuttavia in molti casi non esistono tali iperpiani, quindi se ne ricerca uno che separi quasi completamente le classi e questo metodo si chiama "soft margin classifier".

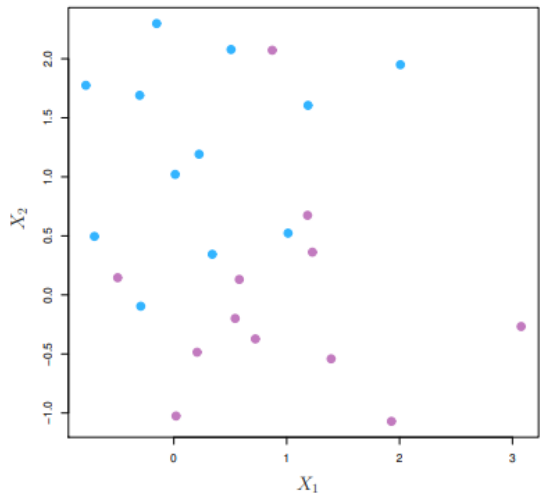

Figura 1.3: in figura vengono mostrate due classi di dati in blu e in viola. In questo caso il metodo del maximal margin classifier non può essere utilizzato perché le classi non sono completamente separabili.

Nel metodo del soft margin l'iperpiano è scelto in maniera tale da separare la maggior

parte del training set nelle due cassi nonostante alcuni dati possano essere dalla parte sbagliata. Ora il problema di ottimizzazione è il seguente:

$$\max_{\vec{w},\varepsilon,b} M \tag{1.11}$$

$$\sum_{j=1}^{N} w_j^2 = 1 \tag{1.12}$$

$$y_i(\vec{w}^T \cdot \phi(\vec{x}_i) - b) \geq M(1-\varepsilon_i) \tag{1.13}$$

$$\varepsilon_i \geq 0, \ \sum_{i=1}^{n} \varepsilon_i \leq C \tag{1.14}$$

dove C è un parametro positivo regolabile che si decide a priori. $\varepsilon_1 ... \varepsilon_n$ sono delle variabili di scarto che permettono al singolo dato di essere dalla parte sbagliata dell'iperpiano. Anche in questo caso per classificare i campioni si valuta tramite la funzione (1.10) da che parte dell'iperpiano si trovino.

Il ruolo dell'iperparametro C è quello di attribuire una penalità all'errore di classificazione, quindi il fatto che un dato appartenente ad una certa classe si trovi nella regione dell'iperpiano a cui viene associata una classe differente. Se $C = 0$ siamo nel caso in cui $\varepsilon_1 = ... = \varepsilon_n = 0$ e ci troviamo nel caso iniziale del "maximal margin classifier" perché violazioni non sono concesse. Invece $C > 0$ implica un limite alle osservazioni che possono essere classificate in modo sbagliato perché si deve rispettare la condizione (1.14). Più C è grande più il margine sarà largo e sono permesse più violazioni, più C è piccolo più il margine sarà stretto e le violazioni permesse saranno di meno. [25]

1.3.1 Kernel trick

Certe volte può essere complicato risolvere il problema di ottimizzazione della SVM col metodo del soft margin a livello computazionale per via del fatto che la dimensione dello spazio in cui la funzione ϕ mappa le feature può essere più grande di quello originale. Un modo più conveniente per semplificare il problema è quello di risolvere un problema di massimi e minimi vincolati tramite moltiplicatori di Lagrange. La risoluzione di questo problema è equivalente a risolvere il problema (1.11):

$$\max \mathscr{L}(\vec{\alpha}) = \sum_{i=1}^{l} \alpha_i - \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \phi(\vec{x}_i) \cdot \phi(\vec{x}_j) \tag{1.15}$$

con $0 \leq \alpha_i \leq C$, $\sum_{i=1}^{l} \alpha_i y_i = 0$ dove gli α_i sono moltiplicatori di Lagrange che determinano l'equazione dell'iperpiano $\vec{w} = \sum_{i=1}^{l} \alpha_i y_i \vec{x}_i$

La feature map $\phi(\vec{x}_i)$ in (1.15) appare solo all'interno di un prodotto scalare, questo permette di evitare il suo calcolo esplicito essendo necessario solo il calcolo di prodotti interni di ϕ valutata su coppie di vettori $\vec{x}_i$ e $\vec{x}_j$. Si definisce una funzione detta "funzione di Kernel" tale che:

$$K : \mathbb{R}^N \times \mathbb{R}^N \longrightarrow \mathbb{R} \tag{1.16}$$

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i) \cdot \phi(\vec{x}_j) \tag{1.17}$$

ora (1.15) diventa

$$\max \mathscr{L}(\vec{\alpha}) = \sum_{i=1}^{l} \alpha_i - \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j) \tag{1.18}$$

Questo è detto "kernel trick". La funzione di Kernel deve presentare le seguenti caratteristiche: deve essere simmetrica quindi $K(\vec{x}_i, \vec{x}_j) = K(\vec{x}_j, \vec{x}_i)$ e deve essere semidefinita positiva in modo tale da garantire che (1.18) presenti un punto di massimo locale perciò

$$\sum_{i,j=1}^{l} c_i c_j K(\vec{x}_i, \vec{x}_j) \geq 0 \tag{1.19}$$

$\forall c_i \; c_j \in \mathbb{R}$

Ora utilizzando il kernel la funzione di classificazione (1.10) diventa [16]:

$$f(\vec{x}) = sign(\sum_{i=1}^{N} \alpha_i K(\vec{x}_i, \vec{x}_j) - b) \tag{1.20}$$

Le SVMs differiscono dal Kernel che si sceglie prima di addestrare la macchina [21].

I kernel più comuni sono il kernel lineare, il kernel gaussiano, il kernel polinomiale ed il kernel sigmoidale [12].

Kernel lineare

$$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j \tag{1.21}$$

Il kernel lineare è il tipo di kernel più semplice e valuta la correlazione tra due variabili valutando il valore del prodotto scalare tra di loro, più il prodotto scalare è vicino a 1 più le variabili sono correlate viceversa se si avvicina a 0.

Kernel polinomiale

$$K(\vec{x}_i, \vec{x}_j) = (\gamma \vec{x}_i \cdot \vec{x}_j + r)^d \tag{1.22}$$

Questo tipo di kernel è un kernel non lineare. d è un intero positivo ed indica il grado del polinomio, con $d = 1$ ci riconduciamo al caso (1.21) mentre con $d > 1$ possiamo passare ad uno spazio delle feature di dimensione maggiore utilizzando polinomi di grado d.

Kernel gaussiano

$$K(\vec{x}_i, \vec{x}_j) = \exp(-\gamma ||\vec{x}_i - \vec{x}_j||^2), \quad \gamma > 0, \gamma \in \mathbb{R} \tag{1.23}$$

Questo kernel è detto kernel gaussiano (o kernel radiale) ed è uno dei kernel più utilizzati. Il funzionamento di questo kernel è il seguente: se un campione del test set $\vec{x}^*$ si trova lontano da un campione del training test $\vec{x}_i$ allora la distanza euclidea tra i due risulterà grande, di conseguenza (1.23) risulterà piccolo. Siccome si utilizza la funzione (1.20) per classificare, i dati del set di training che sono lontani da $\vec{x}^*$ non producono alcun effetto nell'assegnazione della label a $\vec{x}^*$. Il kernel gaussiano ha un comportamento molto locale e solo i dati del set di training vicino a quello del test set che sto considerando contribuiscono alla sua classificazione [25].

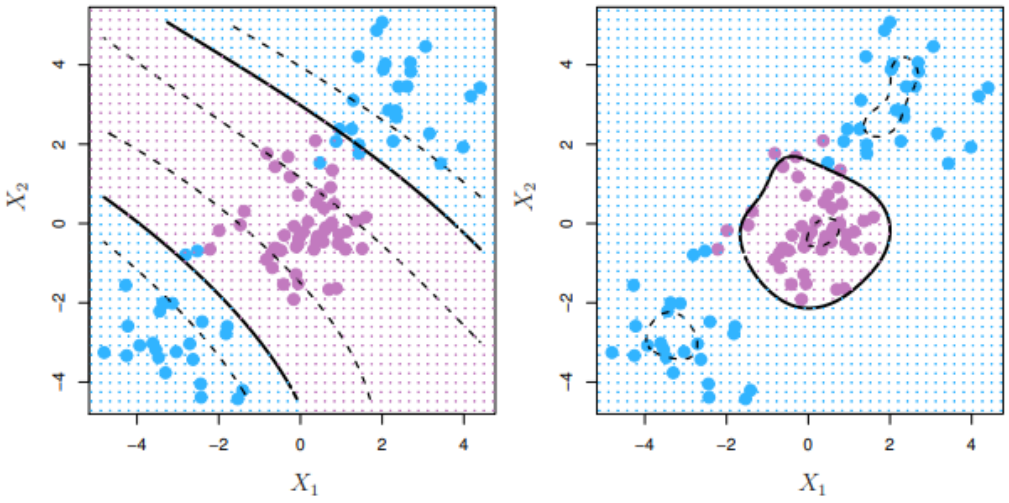

Figura 1.4: nel caso a sinistra è stato utilizzato un kernel polinomiale, nel caso a destra invece è stato utilizzato il kernel gaussiano, tutti e due sono stati in grado di separare le due classi ma in questo caso il kernel gaussiano risulta essere una scelta più appropriata.

Un altro vantaggio che porta allenare una SVM utilizzando il kernel trick al posto di definire una feature map è quello di poter implementare implicitamente degli spazi delle feature infinito dimensionali. Per esempio nel caso del kernel gaussiano per

$n = 1$ il kernel può essere fattorizzato come un vettore con un numero infinito di entrate scritto in espansione di Taylor [21]:

$$K(x_i, x_j) = e^{-||x_i - x_j||^2} = e^{-||x_i||^2} e^{-||x_i||^2} \sum_{k=0}^{+\infty} \frac{2^k x_i^k x_j^k}{k!} \tag{1.24}$$

Kernel sigmoidale

$$K(\vec{x_i}, \vec{x_j}) = \tanh(\gamma \vec{x_i} \cdot \vec{x_j} + r) \tag{1.25}$$

Questo tipo di kernel a seconda della scelta di γ e r si comporta in maniera simile al kernel gaussiano ma generalmente non è una scelta migliore rispetto a quest'ultimo [19].

Capitolo 2

Quantum computing e Quantum Machine Learning

2.1 Introduzione al quantum computing

Il Quantum computing è una tecnologia emergente che sfrutta le leggi della meccanica quantistica per risolvere problemi troppo complessi per i computer classici [18].

Il dispositivo che permette di sfruttare le proprietà della meccanica quantistica come la sovrapposizione di stati e l'entanglement per elaborare dati in maniera più efficiente di un computer classico è il computer quantistico. Ogni computer quantistico può essere simulato da un computer classico ma per specifici problemi la simulazione delle operazioni che esso compie richiede, in generale, una quantità di tempo e di risorse eccessive per qualsiasi applicazione pratica. Ad oggi, uno dei più importanti problemi aperti del quantum computing riguarda la ricerca del cosiddetto "quantum advantage", ossia l'individuazione di applicazioni per le quali gli algoritmi quantistici portano un significativo aumento della velocità computazionale rispetto a qualsiasi alternativa classica.

Il Quantum computing ha fatto grandi progressi negli ultimi anni ma non è un'idea recente. Agli inizi degli anni '80 il fisico Richard Feynman proprose agli informatici di sviluppare nuovi modelli computazionali basati sulla meccanica quantistica. Negli anni '90 il quantum computing ha iniziato a fare progressi in seguito allo sviluppo dei primi algoritmi capaci di perfomare meglio su computer quantistici rispetto a quelli classici [14]. Attualmente i dispositivi quantistici si trovano nella cosiddetta Noisy Intermediate-Scale Quantum era (NISQ-era), questo significa che per avere un tasso di errore piccolo devono eseguire circuiti con un numero limitato di qubit ed entanglement e inoltre hanno pochi qubit, il computer quantistico con il maggior numero di qubit attualmente ne ha 1121 [11] . Nell'Ottobre del 2019 un computer quantistico da 53 qubit in circa 200 secondi ha eseguito un milione di volte un circuito

quantistico (descritti in sezione 2.3.3), per il completamento di questa operazione ci sarebbero voluti circa 10.000 anni utilizzando un supercomputer classico [4], questo fu uno studio rivoluzionario che per primo ha sfidato la computazione classica.

2.2 Qubits

Un Qubit è l'oggetto matematico che rappresenta l'unità fondamentale del quantum computing. La differenza tra bit classico e qubit sta nel fatto che mentre un bit classico può assumere solo due valori, 0 e 1, il qubit è un oggetto a cui si associa una funzione d'onda, in generale è la sovrapposizione di due autostati $|0\rangle$ e $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

α e β sono due numeri complessi, quando misuriamo un qubit potremmo ottenere come risultato 0 con una probabilità $|\alpha|^2$ oppure uno con una probabilità $|\beta|^2$, quindi:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

poiché la somma delle probabilità deve essere 1.

$|0\rangle$ e $|1\rangle$ formano una base ortonormale nello spazio di Hilbert di dimensione 2:

$$B = \left\{ |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix};\ |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} \tag{2.3}$$

e prende il nome di base computazionale [15].

Possiamo visualizzare i qubit geometricamente nel seguente modo: considero gli stati $|0\rangle$ e $|1\rangle$ come il polo nord e il polo sud di una sfera di raggio 1 chiamata sfera di Bloch. Data la relazione (2.2) possiamo riscrivere l'equazione (2.1) come:

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle\right) \tag{2.4}$$

dove θ, ϕ e γ sono numeri reali. θ e ϕ definiscono due punti nello spazio tridimensionale. Il fattore di fase $e^{i\gamma}$ viene fissato in modo tale da rendere l'ampiezza dello stato $|0\rangle$ reale quindi (2.4) diventa:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \tag{2.5}$$

Ciò che è stato descritto fin'ora è il caso di un sistema ad un solo qubit. La potenzialità del quantum computing risiede nel lavorare con stati a più qubit, come descritto nella sezione successiva.

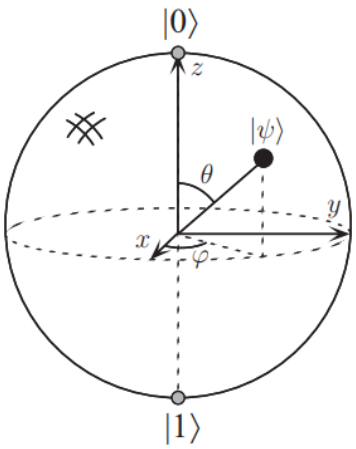

Figura 2.1: rappresentazione di un qubit nella sfera di Bloch.

2.2.1 Sistema a multi-qubit

Quando si va a studiare un sistema a più di un qubit la logica da seguire è la stessa per il singolo qubit. Quando se ne considera uno solo lo spazio di Hilbert in cui è contenuto il qubit ha dimensione 2 poiché gli unici due stati possibili sono $|0\rangle$ e $|1\rangle$. Quando si aumenta il numero di qubit cresce anche il numero di combinazioni che si possono fare tra di essi e di conseguenza lo spazio vettoriale che vanno a formare si estende poiché si combinano gli spazi di Hilbert di ogni qubit tramite prodotto tensoriale. Per n qubit la dimensione dello spazio vettoriale sarà di dimensione 2^n perché sono 2 gli stati possibili che ogni singolo qubit può assumere. Si supponga per semplicità di avere un sistema a due qubit, con H_1 e H_2 i relativi spazi di Hilbert, lo spazio complessivo si ottiene tramite il prodotto tensoriale tra i due:

$$H = H_1 \otimes H_2 = \{|0\rangle \otimes |0\rangle, |0\rangle \otimes |1\rangle, |1\rangle \otimes |0\rangle, |1\rangle \otimes |1\rangle\} \tag{2.6}$$

per semplificare la notazione si può scrivere:

$$H = \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\} \tag{2.7}$$

quindi dati i quattro possibili stati $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, una coppia di qubit può anche esistere come sovrapposizione di questi 4 stati nel seguente modo:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.8}$$

i coefficienti α_{ij} sono dei numeri complessi e la misura dello stato $|ij\rangle$ avrà una porbabilità di $|\alpha_{ij}|^2$, anche in questo caso la seguente condizione di normalizzazione deve essere verificata:

$$\sum_{i,j\in\{0,1\}} |\alpha_{ij}|^2 = 1 \tag{2.9}$$

Misurando il primo qubit esso si può trovare nello stato 0 con una probabilità di $|\alpha_{00}|^2 + |\alpha_{01}|^2$ e lo stato a seguito di questa prima misura risulterà essere:

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} \tag{2.10}$$

dove il fattore $\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}$ è un fattore di ri-normalizzazione.

Uno stato a due qubit molto importante è conosciuto come stato di Bell o coppia EPR:

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.11}$$

la particolarità di questo stato è che misurando il primo qubit posso avere 2 possibili risultati: posso ottenere 0 o 1 con una probabilità di $\frac{1}{2}$, se ottengo 0 lo stato post-misurazione sarà $|\psi'\rangle = |00\rangle$, invece nell'altro caso sarà $|\psi'\rangle = |11\rangle$. Questo implica una forte correlazione tra le due misure, il risultato della prima misura influenza il risultato della seconda. Questa proprietà è detta entanglement, quindi due sistemi S_1 e S_2 sono in stato di entanglement se il loro stato complessivo $|\psi\rangle_{12}$ non può essere scritto in forma fattorizzata come prodotto tensoriale di due stati per i singoli sottoinsiemi $|\psi\rangle_1 \otimes |\psi\rangle_2$ [10]. Questa carattersitica peculiare del quantum computing (ed in generale della meccanica quantistica) può essere sfrutata come risorsa per elaborare informazioni in un modo che sarebbe impossibile o molto più difficile nella computazione classica.

2.2.2 Implementazione hardware di un qubit

La realizzazione fisica di un qubit più comunque è quella che utilizza dei materiali superconduttori raffreddati a temperature criogeniche. Quando un materiale superconduttore si trova al di sotto della sua temperatura critica si verifica il fenomeno della superconduttività. La superconduttività è un fenomeno nel quale le cariche si muovono all'interno del materiale senza incontrare resistenza. A queste temperature, dell'ordine di decine di mK, gli elettroni formano le cosiddette Coppie di Cooper, ossia quasiparticelle di natura bosonica formate da due elettroni in uno stato legato. Siccome il principio di esclusione di Pauli non si applica alle coppie di cooper, e seguono la distribuzione di Bose-Einstein, a temperature sufficientemente basse tendono ad occupare, per la maggior parte, lo stato fondamentale e questo fenomeno prende il nome di condensazione di Bose-Einstein. Raggiunta questa condizione il materiale manifesterà le proprietà della superconduttività, l'assenza di resistenza è fondamentale per ottenere degli stati quantistici coerenti.

Esistono diversi design di qubit superconduttivi e uno tra quelli più promettenti è

il transmon qubit. Quantisticamente il sistema può essere descritto come un oscillatore anarmonico che viene realizzato tramite un circuito LC dove in questo caso L è un'induttanza non lineare ottenuta aggiungendo in parallelo una giunzione Josephson. Grazie agli effetti anarmonici prodotti dalla giunzione Josephson viene resa distinguibile la frequenza di transizione tra livelli energetici poiché quest'ultimi non sono equispaziati come nel caso in cui si utilizzasse un'induttanza lineare. In questo modo si realizza un sistema che presenta due livelli distinguibili a cui associo allo stato fondamentale lo stato $|0\rangle$ e al primo stato eccitato $|1\rangle$. [21, 15, 2]

Tutti i tipi di qubit sono soggetti ad un fenomeno detto decoerenza. La decoerenza è dovuta dal fatto che il qubit interagendo con l'ambiente circostante perde la stabilità degli stati quantistici. A seguito di queste interazioni potrebbe transire ad uno stato quantistico indesiderato introducendo errori nella computazione. Le cause che possono portare a fenomeni di decoerenza sono varie, per esempio il rumore termico delle componenti del circuito che può portare a fluttuazioni di corrente oppure fluttuazioni casuali di campi elettrici e magnetici nell'ambiente in cui si trova il qubit.

Esistono anche delle fonti di rumore che vengono definite sistematiche, queste fonti di errore generalmente vengono identificate e corrette attraverso una corretta calibrazione dell'hardware, per esempio delle fonti di rumore sistematico sono gli impulsi a microonde che controllano lo stato del qubit che possono essere mal calibrati. [7]

2.3 Quantum Gates

Classicamente un gate è uno strumento matematico che permette di svolgere un'operazione logica su un bit. Nel caso del quantum computing è possibile definire lo stesso tipo di strumento per svolgere operazioni sui qubit, che prende il nome di quantum gate. Un quantum gate è dotato di una rappresentazione matriciale e l'unica condizione che deve rispettare una qualsiasi matrice che definisce quest'ultimo è che deve essere unitaria. Quindi data U una generica matrice che definisce un quantum gate, deve essere tale che:

$$UU^{\dagger} = I \tag{2.12}$$

dove I è l'identità e $U^{\dagger}$ è l'aggiunto di U (il trasposto coniugato). Il fatto che la matrice sia unitaria implica anche che sia invertibile e questa proprietà porta delle conseguenze importanti. Nella computazione classica le operazioni logiche non sono completamente reversibili poiché parte dell'informazione viene persa quando si svolge la stessa operazione in maniera inversa. Nel quantum computing invece i quantum gates permettono operazioni inverse consentendo il cosiddetto "reversible computing", quindi si possono ripercorrere i calcoli in direzione opposta senza perdere informazione. [15]

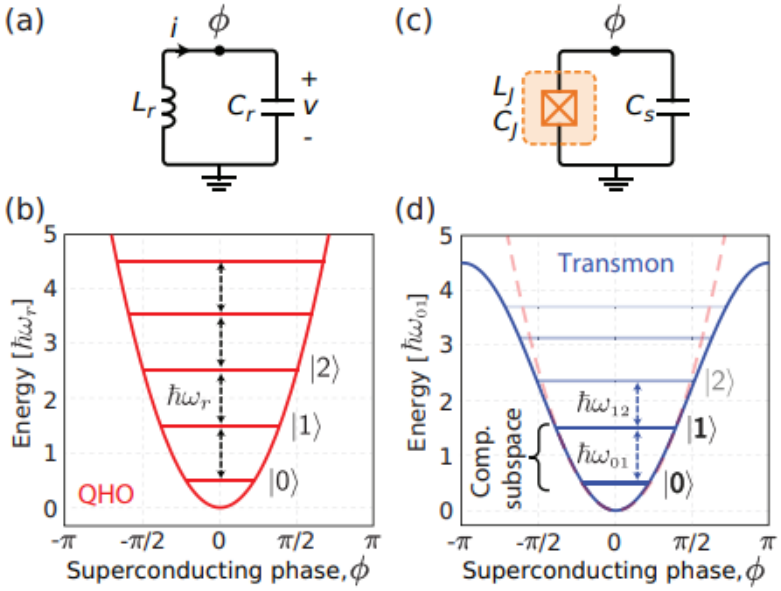

Figura 2.2: la figura rappresenta in (a) un circuito LC con induttanza lineare, in (b) è rappresentata l'energia potenziale del circuito LC dove i livelli energetici sono equispaziati. In (c) è rappresentato il transmon qubit con la giunzione Josephson al posto di L con la relativa energia potenziale in (d) dove si può notare il fatto che i livelli energetici non siano più equistapziati permettendo di isolare gli stati $|0\rangle$ e $|1\rangle$.

2.3.1 Gates a singolo qubit

In questa sezione verranno approfonditi alcuni tipi di gate che agiscono solo su un qubit.

In generale un gate a singolo qubit è definito da una matrice del tipo:

$$U = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{2.13}$$

dato uno stato generico a singolo qubit:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.14}$$

l'azione del gate sarà:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a\alpha + b\beta \\ c\alpha + d\beta \end{pmatrix} \tag{2.15}$$

quindi ora lo stato $|\psi\rangle$ è diventato:

$$|\psi\rangle = (a\alpha + b\beta)\,|0\rangle + (c\alpha + d\beta)\,|1\rangle \tag{2.16}$$

Pauli gates

I Pauli gates sono descritti da matrici di dimensione 2×2 unitarie. Le più importanti e più utilizzate sono le matrici di Pauli:

$$\sigma_x \equiv X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.17}$$

$$\sigma_y \equiv Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.18}$$

$$\sigma_z \equiv Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.19}$$

stato iniziale	X	Y	Z
$\|0\rangle$	$\|1\rangle$	$i\|1\rangle$	$\|0\rangle$
$\|1\rangle$	$\|0\rangle$	$-i\|0\rangle$	$-\|1\rangle$

Tabella 2.1: nella tabella viene riportata l'azione di ogni Pauli Gate sul singolo qubit.

Si può notare che per esempio l'X gate è l'equivalente quantistico del NOT gate cui azione consiste nel prendere uno stato e negarlo, quindi rimpiazzarlo con l'altro. Lo Z gate invece lascia invariato lo stato ma provoca un cambiamento di fase.

Hadamard gate

L'Hadamard gate è definito dalla seguente matrice:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.20}$$

L'azione del gate sul singolo qubit è la seguente:

stato iniziale	H
$\|0\rangle$	$\frac{\|0\rangle+\|1\rangle}{\sqrt{2}} := \|+\rangle$
$\|1\rangle$	$\frac{\|0\rangle-\|1\rangle}{\sqrt{2}} := \|-\rangle$

I due stati ora sono in sovrapposizione e la probabilità di ottenere o $|0\rangle$ o $|1\rangle$ è per tutti e due del 50%. Applicando due volte l'Hadamard gate si lascia invariato lo stato poiché $H^2 = I$.

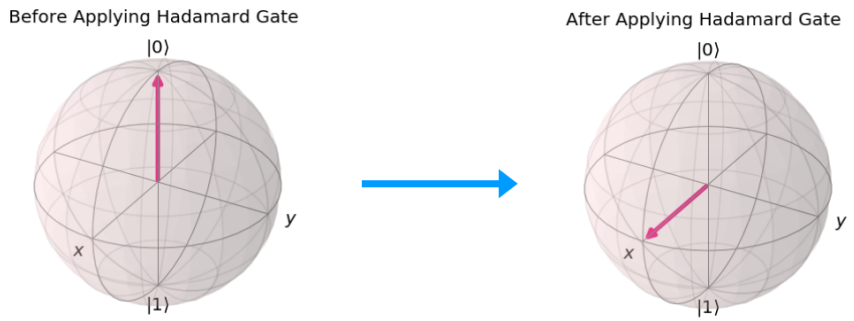

Figura 2.3: Rappresentazione sulla sfera di Bloch dell'azione dell'Hadamrd gate sullo stato $|0\rangle$.

Rotation gates

I rotation gates sono tre gates che descrivono una rotazione attorno ad uno specifico asse della sfera di Bloch attraverso un parametro θ. La loro rappresentazione matriciale è la seguente:

$$R_x(\theta) = e^{-i\frac{\theta}{2}X} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right) X = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{2.21}$$

$$R_y(\theta) = e^{-i\frac{\theta}{2}Y} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right) Y = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{2.22}$$

$$R_z(\theta) = e^{-i\frac{\theta}{2}Z} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right) Z = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{\frac{\theta}{2}} \end{pmatrix} \tag{2.23}$$

dove X, Y, Z sono rispettivamente (2.17), (2.18), (2.19) e I è la matrice identità.

2.3.2 CNOT gate

Si possono definire anche dei quantum gate che agiscono su più di un qubit. Un esempio comune di gate a multi qubit è il "controlled-NOT" o "CNOT gate". Questo gate prende in input due qubit dove uno è definito control qubit ed il secondo target qubit ed esso è il qubit sul quale si agisce. L'azione del gate è la seguente:

$$|A,B\rangle \longrightarrow |A,A \oplus B\rangle \tag{2.24}$$

dove $\oplus$ è il simbolo che indica l'addizione modulo 2 che corrisponde all'operazione XOR tra i qubit. Questo significa che se il control qubit è lo $|0\rangle$ il target qubit rimane immutato se invece il control qubit è $|1\rangle$ il target qubit viene invertito.

La forma matriciale del CNOT gate è data dalla seguente matrice 4×4:

$$U_{CN} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.25}$$

stato iniziale	CNOT
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|11\rangle$
$\|11\rangle$	$\|10\rangle$

Tabella 2.2: la tabella mostra l'azione del CNOT gate nei diversi casi.

Questo gate è molto importante perché permette di creare coppie di qubit entangled, ciò è dovuto dal fatto che la matrice (2.25) non deriva dalla fattorizzazione di matrici a singolo qubit [15, 20].

2.3.3 Circuito quantistico

Un circuito quantistico è formato da un insieme di qubits e quantum gates. Un esempio di circuito quantistico è riportato nella figura 2.6. Un circuito quantistico viene letto da sinistra verso destra e ogni linea è associata all'evoluzione di un qubit ed è detta "registro". I gates in un circuito vengono rappresentati con un quadrato e una lettera che indica il gate scelto, la rappresentazione del CNOT gate invece è la seguente: col punto nero e una linea viene rappresentato il control qubit e con la croce il target qubit.

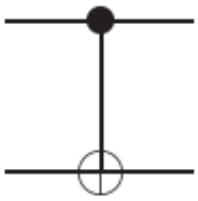

Figura 2.4: rappresentazione grafica del CNOT gate.

L'operazione di misura invece viene rappresentata col simbolo in figura 2.5.

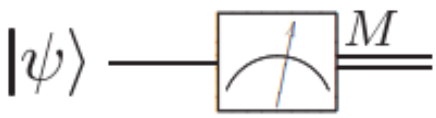

Figura 2.5: simbolo che indica l'operazione di misura di un qubit.

Convenzionalmente si assume che inizialmente ogni qubit presente nel circuito si trovi nello stato $|0\rangle$.

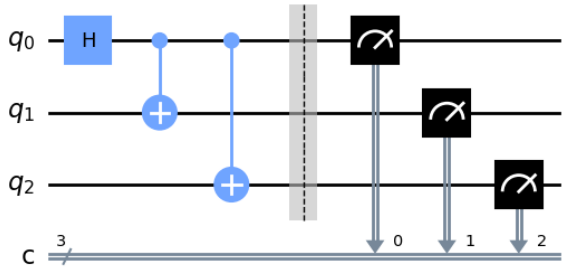

Figura 2.6: in figura è rappresentato un circuito quantistico a 3 qubit che presenta un Hadamard gate e 2 CNOT gates, i simboli in fondo indicano l'operazione di misura.

2.4 Quantum Machine Learning

Il Quantum machine learning è un'area del quantum computing che si propone potenziare modelli già esistenti di ML e crearne di nuovi sfruttando i princìpi fondamentali della meccanica quantistica [6]. Ci possono essere 4 differenti approcci che permettono di unire il machine learning ed il quantum computing derivati dalla combinazione dei modi in cui vengono generati ed elaborati i dati. I dati possono essere generati da sistemi classici (C) oppure quantistici (Q) e analogamente possono essere elaborati da dispositivi classici (C) oppure quantistici (Q). I casi sono i seguenti:

- CC è il caso in cui i dati vengono sia generati che processati classicamente ed è l'approccio convenzionale al machine learning. Nel contesto del quantum computing ci si riferisce a metodi di machine learning ispirati al quantum computing.

- QC è il caso in cui si ricerca come il machine learning possa portare vantaggi nel quantum computing. Per esempio alcuni algoritmi di machine learning possono essere utili per fare in modo di determinare stati interni ad un computer

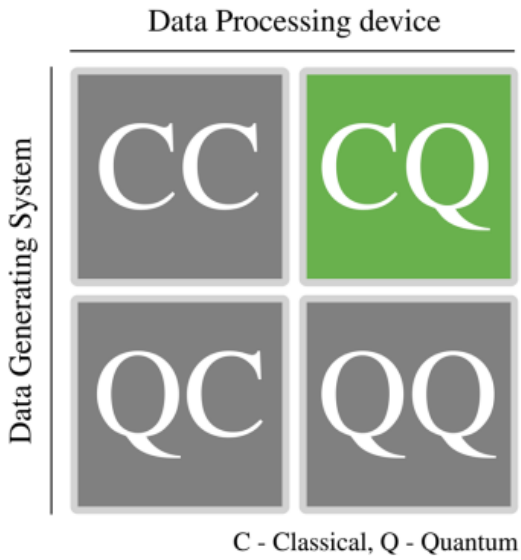

Figura 2.7: Questa tabella mostra i 4 differenti approcci del quantum machine learning che differiscono a seconda di come i dati vengono generati e processati.

quantistico con meno misurazioni possibili, questo problema è chiamato quantum state tomography e tradizionalmente richiede un numero di misurazioni che cresce esponenzialmente col numero di qubit [1].

- CQ è il caso in cui si utilizza il quantum computing per processare dati generati classicamente.
- QQ è il caso in cui dati generati quantisticamente vengono processati da dispositivi quantistici.

L'algoritmo di QML utilizzato in questo elaborato è il Quantum Support Vector Machine. In questo lavoro di tesi, la QSVM viene utilizzata nell'analisi di un dataset classico, si ricade quindi nell'insieme CQ.

2.4.1 Quantum Support Vector Machine e quantum Kernel

Il Quantum Support Vector Machine (QSVM) è considerato un modello ibrido che richiede la cooperazione tra processori classici e quantistici. La fase di training è la stessa della SVM classica descritta nella sezione 1.3 dove la parte quantistica è data dalla definizione di quantum kernel.

La feature map $\phi(\vec{x})$ in questo caso è uno stato quantistico $|\phi(\vec{x_i})\rangle\langle\phi(\vec{x_i})|$ contenuto nello spazio di Hilbert $H = \mathbb{C}^{2^n \times 2^n}$, con $i \in \{1,...,n\}$ dove n è il numero di sample del dataset. Lo stato si può considerare come lo stato iniziale $|0^n\rangle\langle 0^n|$ a cui applico

la quantum feature map che è l'operatore unitario $U(\vec{x}_i)$:

$$|\phi(\vec{x}_i)\rangle \langle\phi(\vec{x}_j)| = U(\vec{x}_i)\,|0^n\rangle \langle 0^n|\,U(\vec{x}_j)^\dagger \tag{2.26}$$

Le quantum feature map sono in grado di mappare i dati in uno spazio di Hilbert la cui dimensione cresce esponenzialmente all'aumentare del numero di qubit che la compongono.

Il quantum kernel è definito dal modulo quadro del prodotto scalare delle feature maps

$$K(\vec{x}_i,\vec{x}_j) = |\langle\phi(\vec{x}_i)|\phi(\vec{x}_j)\rangle|^2 = |\langle 0^n|U(\vec{x}_i)U(\vec{x}_j)^\dagger|0^n\rangle|^2 \tag{2.27}$$

Quindi il kernel ci dà la probabilità, dopo la misurazione degli stati di qubit, di ottenere di nuovo lo stato iniziale $|0^n\rangle$ in seguito all'applicazione della feature map descritta da $U(\vec{x}_i)U(\vec{x}_j)^\dagger$.

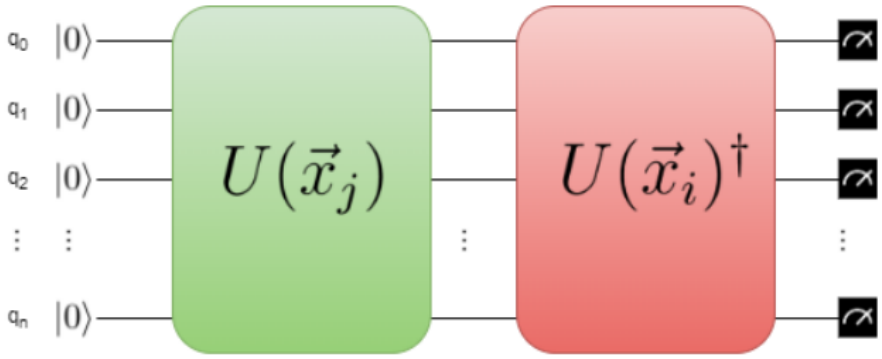

Figura 2.8: Rappresentazione di un circuito quantistico a n qubit a cui viene applicato un kernel arbitrario.

Il quantum kernel è in grado di determinare se i dati in input sono compatibili o no poiché $K(\vec{x}_i,\vec{x}_j) = 1$ se $i = j$ e $K(\vec{x}_i,\vec{x}_j) = 0$ se $\vec{x}_j$ e $\vec{x}_i$ sono ortogonali nello spazio di Hilbert ed è simmetrico e semidefinito positivo [21].

Il kernel descritto fin'ora è anche conosciuto come Fidelity Quantum Kernel ed è considerato il kernel standard per lo sviluppo di QSVM.

Una seconda tipologia di Kernel è il Projected Quantum Kernel. Esso è definito nel seguente modo: [5, 8]:

$$K(\vec{x}_i,\vec{x}_j) = \exp\left(-\gamma \sum_{k=1}^{n} ||\rho_k(\vec{x}_i) - \rho_k(\vec{x}_j)||_2^2\right) \tag{2.28}$$

In (2.28) si misura la matrice di densità ridotta sul k-esimo qubit $\rho_k(\vec{x}) = \mathrm{Tr}_{j\neq k}[\rho(x_i)]$. $\rho(\vec{x})$ è la matrice di densità ed è definita nel seguente modo [10]:

$$\rho(\vec{x}) = |\phi(\vec{x})\rangle \langle\phi(\vec{x})| \tag{2.29}$$

γ è un iperparametro e $||\cdot||_2$ è la Schatten 2-norma definita come: $||M||_2 = [\mathrm{Tr}(|M|^2)]^{\frac{1}{2}}$.

Il Projected quantum kernel è costruito misurando il sottosistema del singolo qubit permettendo una riduzione dello spazio che contiene lo stato finale del sistema a più qubit.

2.4.2 Exponential Concentration

Il Fidelity Quantum Kernel è particolarmente soggetto a un problema conosciuto come "exponential concentration" (o concentrazione esponenziale) [8]. Sotto certe condizioni, per una generica feature map, all'aumentare del numero di qubit, il valore della funzione di kernel si concentra esponenzialmente attorno ad un valore fissato, ossia la varianza del quantum kernel valutato sugli elementi di un dataset $(\vec{x_i}, \vec{x_j})$ con $i \neq j$ tende a zero. Questo problema va a ostacolare il processo di apprendimento del modello di QSVM. La ragione è che lo spazio di Hilbert del sistema a multi qubit cresce, quindi la probabilità di avere tutti i qubit in 0 dopo le operazioni in figura 2.8 è sempre più bassa. In presenza di exponential concentration, i quantum support vector machine si rivelano inefficaci poiché il rumore di un dispositivo quantistico NISQ sovrasta le differenze del valore della funzione di kernel per elementi del dataset diversi. Il lavoro di tesi è volto a studiare il projected quantum kernel, descritto nella sezione precedente, per verificare la mitigazione dell'exponential concentration.

Il problema dell'exponential concentration viene formalizzato nel seguente modo: Sia $X(\alpha)$ una quantità che dipende da un set di variabili α e che può essere misurata da un computer quantistico come valore di aspettazione di un'osservabile. $X(\alpha)$ si concentra esponenzialmente in maniera deterministica nel numero n di qubit attorno ad un certo valore fissato μ se:

$$|X(\alpha) - \mu| \leq \beta \in O\left(\frac{1}{b^n}\right) \tag{2.30}$$

Per alcuni $b > 1$ e per ogni α. Analogamente probabilisticamente $X(\alpha)$ si concentra epsonenzialmente se:

$$\mathrm{Pr}_\alpha[|X(\alpha) - \mu| \geq \delta] \leq \frac{\beta^2}{\delta^2}, \;\; \beta \in O\left(\frac{1}{b^n}\right) \tag{2.31}$$

per $b > 1$. Questa è la probabilità che $X(\alpha)$ si discosti da μ di una piccola quantità δ; questa probabilità decresce esponenzialmente per tutti gli α.

Nel contesto dei kernel quantistici: $X(\alpha) = k(x,x')$ con $\alpha = \{x,x'\}$. La concentrazione esponenziale della funzione di kernel si può visualizzare nella matrice di kernel che presenta tutti gli elementi sulla diagonale pari a 1 e quelli fuori dalla diagonale che si concentrano attorno μ.

Capitolo 3

Classificazione di pattern di rumore

In questo capitolo viene descritta l'implementazione dell'algoritmo Quantum Support Vector Machine, descritto nella sezione 2.4.1, con l'utilizzo di vari tipi di kernel, per classificare il pattern di rumore di due computer quantistici. Il Fidelity Quantum Kernel, ossia il tipo di kernel più utilizzato nell'implementazione del QSVM è limitato perché presenta un problema, descritto nella sezione 2.4.2, conosciuto come "exponential concentration". Si è osservato che sotto certe condizioni il valore della funzione di kernel valutata su elementi del dataset diversi si concentra attorno ad una costante μ. Questo capitolo mostra come l'utilizzo del Projceted Quantum Kernel, possa portare a risultati significativamente migliori mitigando il problema della concentrazione esponenziale. I programmi utilizzati per svolgere l'analisi dati sono stati scritti in linguaggio di programmazione Python e utilizzando le librerie Qiskit e scikit-learn. La libreria Qiskit è una libreria sviluppata da IBM che permette di eseguire tipologie di algoritmi quantistici, nonché la simulazione di circuiti quantistici arbitrari. La libreria scikit-learn è una libreria di python che fornisce una serie di strumenti per l'analisi dati e il machine learning ed è stata utilizzata per implementare il Support Vector Machine.

3.1 Descrizione del dataset

il dataset su cui è stato svolto il lavoro contiene un insieme di distribuzioni di stati finali ottenuti in seguito all'esecuzione di un circuito quantistico arbitrario su due diversi computer quantistici simulati, soggetti a diverse tipologie di errore, tramite il modulo `fake_provider` di Qiskit. Le distribuzioni dei due backend sono diverse perché ogni processore quantistico è soggetto a diverse tipologie di errore che portano ad effetti sistematici. I due computer quantistici si chiamano Perth e Manila, rispettivamente da 7 e 5 qubits, originariamente di proprietà di IBM ma ora dismessi. Il circuito è rappresentato in figura 3.1. Vengono misurati 2 qubit, e dato che un qubit si può trovare in 2 stati differenti $|0\rangle$ e $|1\rangle$, la misurazione potrà dare 4 possi-

bili risultati: $|00\rangle$, $|01\rangle$, $|10\rangle$ e $|11\rangle$. Il numero di conteggi dei 4 possibili risultati rappresentano le 4 features del dataset.

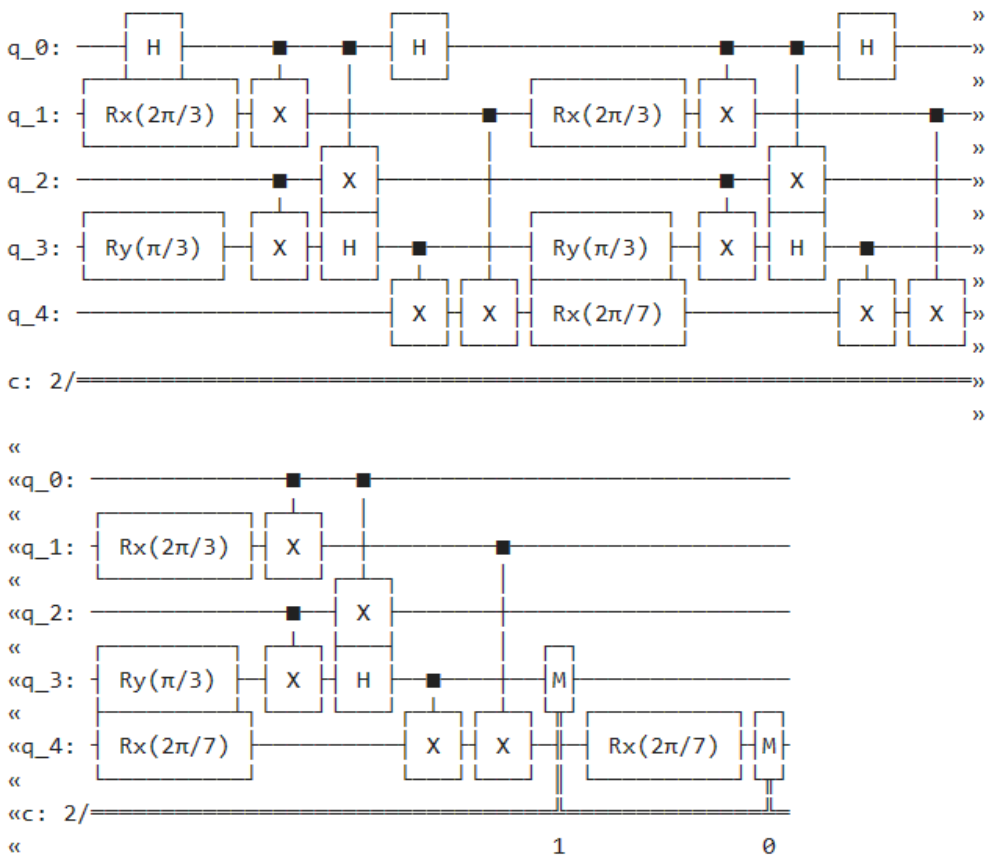

Figura 3.1: in figura è rappresentato il circuito quantistico utilizzato per generare il dataset dove è stato deciso arbitrariamente di misurare il quarto ed il terzo qubit.

Per la creazione del dataset sono state utilizzate 2 funzioni: una è servita per l'esecuzione del circuito sul dispositivo quantistico e l'altra per la costruzione del dataset. La funzione utilizzata per l'esecuzione del circuito è mostrata in figura 3.2. La funzione `transpile` è una funzione che permette di costruire un circuito equivalente al circuito in figura 3.1 che un dato computer quantistico è in grado di eseguire. Questo è necessario perché ogni computer quantistico ha i suoi gates "nativi", ossia gli unici gate che possono essere utilizzati. L'operazione di transpiling, quindi, scompone il circuito dato in input in modo tale da ottenere solo gate nativi.

La funzione utilizzata per costruire il dataset invece è illustrata in figura 3.3.
Per generare il dataset si sceglie il numero di samples, il circuito da eseguire, i backend dei due computer quantistici e il numero di conteggi.

```python
def run_circuit(circuit, backend, shots = 1000, measured_qubits = 5):
    transpiled_circuit = transpile(circuit, backend)
    job = backend.run(transpiled_circuit, shots = shots)
    counts = job.result().get_counts()
    all_possible_keys = [format(i, '0'+str(measured_qubits)+'b') for i in range(2**measured_qubits)]
    for key in all_possible_keys:
        if key not in counts:
            counts[key] = 0.0
    sorted_counts = dict(sorted(counts.items()))
    return sorted_counts
```

Figura 3.2: In figura è mostrato il codice che permette di simulare l'esecuzione di un circuito quantistico.

```python
def make_dataset(num_samples, circuit, backend1, backend2, shots = 1000, measured_qubits = 5):
    dataset = np.zeros((2*num_samples, 2**measured_qubits))
    for i in range(num_samples):
        counts_array = np.zeros(0)
        counts = run_circuit(circuit, backend1, shots, measured_qubits)
        for key in counts.keys():
            counts_array = np.append(counts_array, counts[key])
        dataset[i] = counts_array
    for i in range(num_samples, 2*num_samples):
        counts_array = np.zeros(0)
        counts = run_circuit(circuit, backend2, shots, measured_qubits)
        for key in counts.keys():
            counts_array = np.append(counts_array, counts[key])
        dataset[i] = counts_array
    labels = np.concatenate([np.zeros(num_samples), np.ones(num_samples)])
    return dataset, labels
```

Figura 3.3: In figura è mostrato il codice della funzione usata oer la generazione del dataset.

Per un dataset da 2000 samples (1000 generati con il computer quantistico Manila e 1000 con Perth) con 100.000 shots ciascuno, si ottengono le distribuzioni nelle figure dalla 3.5 alla 3.7. I grafici nelle figure rappresentano come si distribuiscono le feature appartenenti a due classi diverse, l'una in rispetto all'altra. Le features sono collegate poiché la feature 0 è il numero di volte che esce $|00\rangle$, la feature 1 è il numero di volte che esce $|01\rangle$, la feature 2 è il numero di volte che esce $|10\rangle$, la feature 4 è il numero di volte che esce $|11\rangle$. Siccome la somma dei loro conteggi deve essere 100.000 le features sono legate tra di loro da un vincolo; questo permetterebbe di ricondurre il problema ad un caso a 3 features, ciò però non è stato investigato in quanto oltre lo scopo dell'elaborato. Inoltre osservando le figure si può notare che la separazione tra le due classi non è lineare affermando il fatto che si tratti un problema di classificazione interessante in quanto non banale.

Eseguendo dei plot per diversi dataset con un numero crescente di conteggi si è osservato che la separazione tra le classi risulta sempre più evidente; un esempio è riportato nella figura 3.4. Un un numero elevato di conteggi permette la soppressione della fluttuazione statistica lasciando solo la componente sistematica di errore dei due backend. Per questa ragione per le fasi successive dell'analisi dati sono stati usati dataset con un elevato numero di shots, in modo da ottenere accuratezze di classificazione più alte.

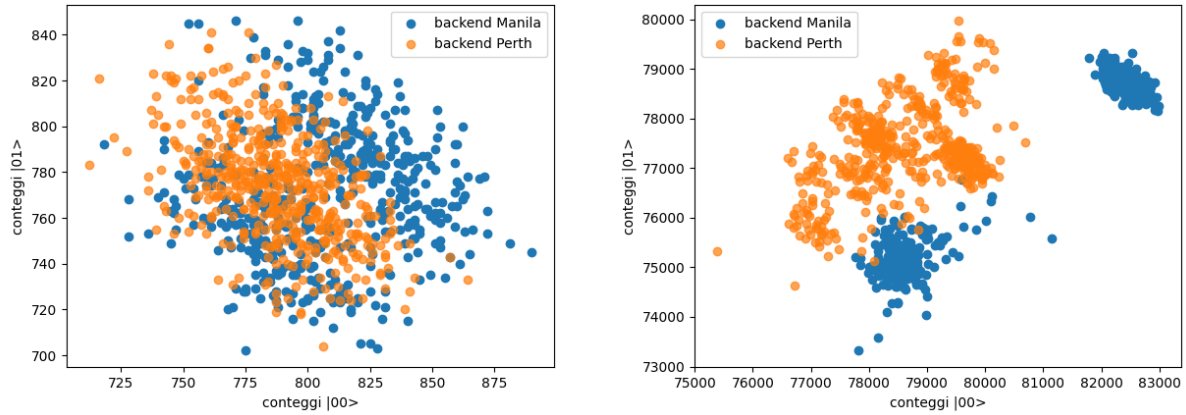

Figura 3.4: Questi due grafici mostrano la distribuzione della prima feature rispetto alla seconda per entrambe le classi con a sinistra il caso di un dataset da 1000 shots per sample e a destra uno da 100.000 shots.

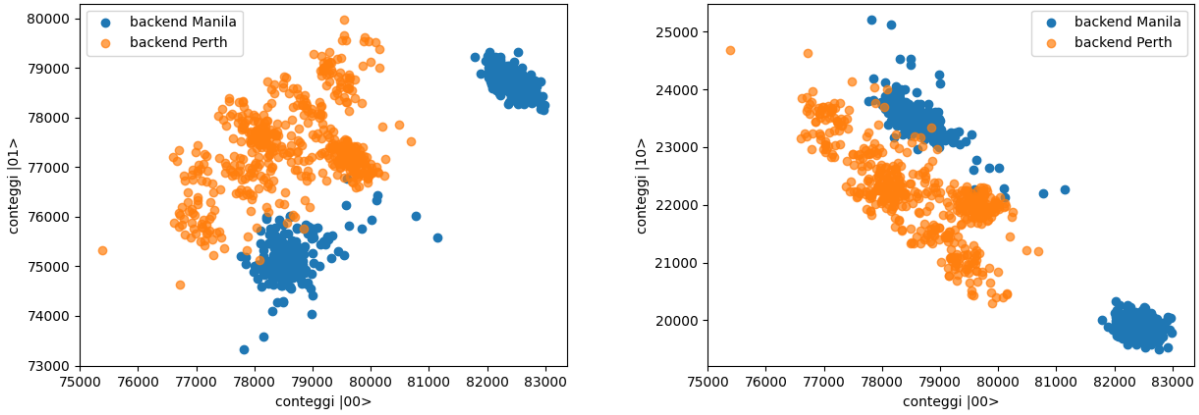

Figura 3.5: Nella figura a sinistra è mostrata la distribuzione della prima feature in rispetto alla seconda, in quella a destra la prima feature rispetto alla terza.

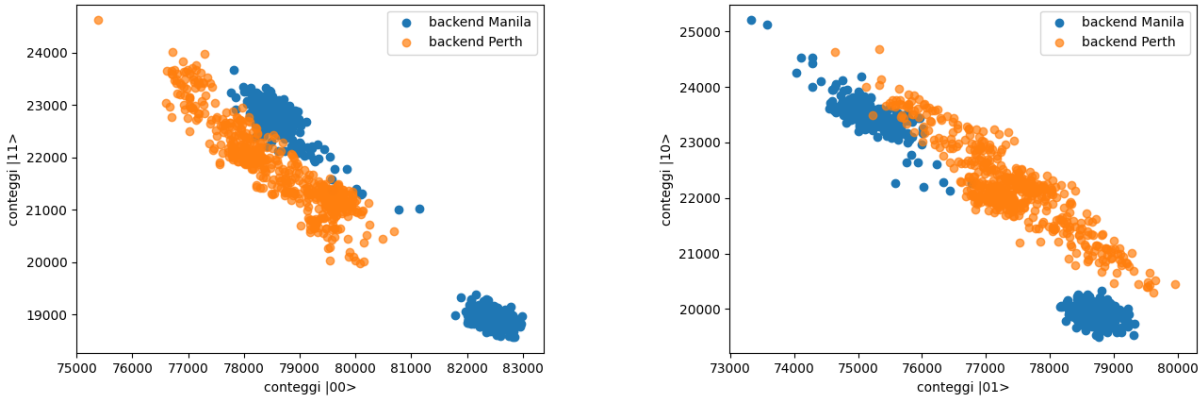

Figura 3.6: Nella figura a sinistra è mostrata la distribuzione della prima feature rispetto alla quarta, in quella a destra la seconda feature rispetto della terza.

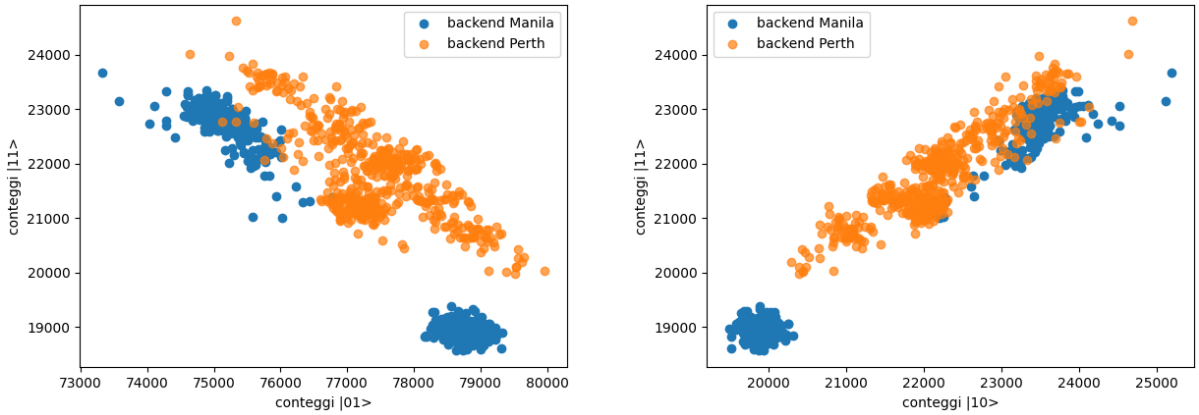

Figura 3.7: Nella figura a sinistra è mostrata la distribuzione della seconda feature in rispetto alla seconda, in quella a destra la terza feature rispetto alla quarta.

3.2 Ottimizzazione di SVM e QSVM

Scegliere il kernel da utilizzare è un aspetto di cruciale importanza nello sviluppo di un modello di SVM e QSVM, poiché ne determina l'accuratezza di classificazione, ossia la frazione di elementi del dataset che è stata classificata correttamente dal modello. In questa sezione viene mostrato il confronto tra le accuratezze ottenute con diversi modelli di SVM classico e di QSVM. Per il SVM i modelli differiscono in base al kernel utilizzato, in questo caso sono stati utilizzati il kernel lineare (1.21) ed il kernel gaussiano (1.23); mentre i modelli di QSVM differiscono per le feature map utilizzate per definire il Fidelity Quantum Kernel. In questo caso sono state utilizzate 4 quantum feature map, due già fornite dalla libreria Qiskit, denominate Z e ZZ feature map, e due create appositamente per questo elaborato, in modo che una presenti

un elevato numero di gates a due qubit, per impementare entanglement, e l'altra non ne presenti. In una prima parte della sezione verrà descritta l'implementazione dei due algoritmi, mentre nella seconda verranno commentati i risultati ottenuti.

3.2.1 Implementazione del SVM classico

Per poter implementare il SVM per prima cosa è stato necessario dividere il dataset in 2 parti: il 70% di esso ha costituito il training set ed il restante 30% il test set. L'operazione è stata svolta tramite la funzione di scikit learn `train_test_split`.

Per definire il modello di Support Vector Machine è stata utilizzata la classe SVC, scegliendo il kernel desiderato e specificando il valore desiderato per gli iperparametri, e poi per la fase di addestramento è stata utilizzata la funzione `fit` sul set di training. Successivamente è stato effettuato il test tramite la funzione `predict` sul test set per poi andarne a valutare l'accuratezza utilizzando `accuracy_score`.

```
# SVM definition and training
svm_classifier = SVC(kernel='linear', C=10)
svm_classifier.fit(training_set, training_label)
y_pred = svm_classifier.predict(test_set)
accuracy = accuracy_score(test_label, y_pred)
```

Figura 3.8: Nella figura è mostrato un esempio di implementazione di SVM nel caso in cui il kernel sia lineare, dove C è un iperparametro.

Per scegliere gli iperparametri ottimali (definiti in 1.1) da utilizzare per definire il modello di SVM è stata usata la funzione `GridSearchCV`, che permette di ricercare in una griglia di iperparametri specificati il migliore da utlizzare.

3.2.2 Implementazione del QSVM

In questa sezione viene descritta l'implementazione del QSVM con il Fidelity Quantum Kernel. Per prima cosa si definisce il Fidelity Quantum Kernel associato alla feature map quanstistica che si va ad utilizzare, tramite la funzione `FidelityStatevectorKernel` valutandolo sul dataset prima di addestrare il modello, procedendo poi in maniera a simile a come è stato fatto per il SVM come illustrato nella figura 3.9.

In questo caso l'unico iperparametro da selezionare è il l'iperparametro C per ogni feature map utilizzata, il quale viene scelto sempre utilizzando `GridSearchCV`. Per quanto riguarda l'accuratezza viene definita una matrice di kernel per il test e poi

```
qker = FidelityStatevectorKernel(feature_map=fmap)
qker_matrix = qker.evaluate(x_vec=training_set)
qsvm_classifier = SVC(kernel="precomputed", C=1)
qsvm_classifier.fit(qker_matrix, training_label)
```

Figura 3.9: codice per l'implementazione del QSVM.

viene valutata l'accuratezza del modello tramite la funzione `score` nella seguente maniera:

```
qker_matrix_test= qker.evaluate(x_vec=validation_set, y_vec= training_set)
qsvm_classifier.score(qker_matrix_test, test_label) #accuratezza
```

Figura 3.10: codice utilizzato per valutare l'accuratezza del modello

Di seguito vengono presentati alcuni esempi di feature map testate nell'elaborato. Le tipologie di feature map sono le strategie utilizzate per definire i gate nel circuito.

Z Feature Map

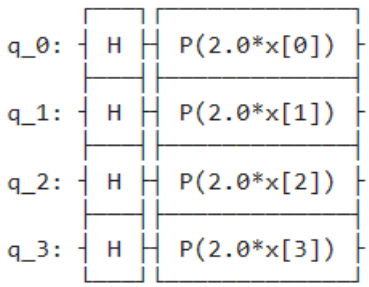

Figura 3.11: circuito che rappresenta la Z Feature map.

In questo caso la strategia è quella di applicare un H gate ed un P gate ad ogni qubit. H è l'Hadamard gate, P è un gate chiamato Phase Gate ed è equivalente ad una rotazione R_Z definita in (2.23) moltiplicata per un fattore di fase $P(\theta) = e^{i\frac{\theta}{2}} R_Z(\theta)$. $x[i]$ con $i \in \{0,1,2,3\}$ sono delle features del dataset che vanno a definire l'angolo di rotazione θ del Phase gate.

ZZ Feature Map

È una feature map a 2 qubit, in questo caso viene applicato un H e un P gate ad ogni

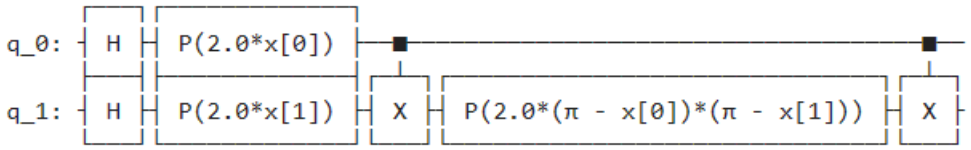

Figura 3.12: circuito che rappresenta la ZZ Feature map.

qubit e poi un CNOT gate con affiancato un Phase gate (descritto in sezione 2.3.2) che permette di creare entanglement.

Feature map priva di entanglment

In figura 3.13 U2 corrisponde all'hadamard gate, U1 corrisponde invece a una rotazione R_z poi ci sono delle rotazioni R_y ed R_x rappresentate dai gate R che prendono come primo argomento il parametro che definisce la rotazione lungo x ed il secondo lungo y. In questo caso ogni gate viene applicato ad ognuno dei qubit presenti nel circuito utilizzando gli stessi parametri.

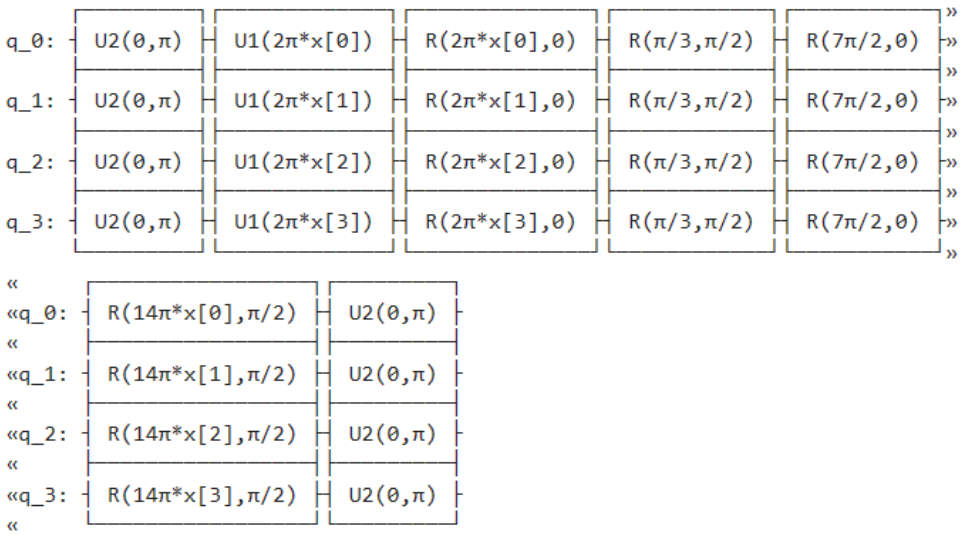

Figura 3.13: circuito che rappresenta la feature map senza entanglement.

Feature map con entanglement

In questa feature map sono stati utilizzati dei cnot gate a cui viene anche affiancato un R_z e un Phase gate.

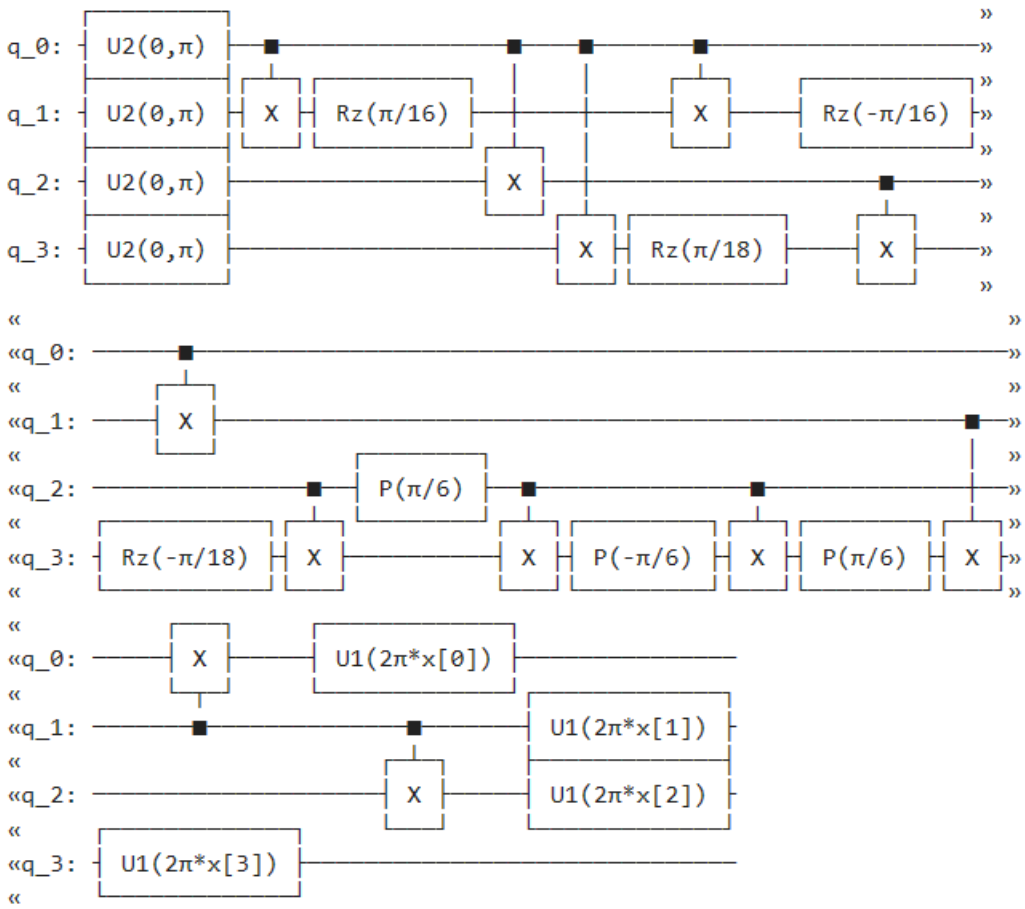

Figura 3.14: circuito che rappresenta la feature map con entanglement.

3.2.3 Accuratezze di classificazione

In questa fase dell'analisi dati sono stati utilizzati diversi dataset con un numero di samples per computer quantistico crescente da 10000 shots ciascuno.

Le accuratezze di classificazione ottenute per i SVM classici sono indicate nella tabella 3.1. Il fatto che il kernel gaussiano dia percentuali molto più alte del kernel lineare indica che la distribuzione dei dati che si hanno a disposizione non è banale e c'è bisogno di kernel più complessi per ottimizzare il SVM, come affermato nella sezione 3.1. Si conferma così che la non linearità del kernel apporta un vantaggio significativo alla performance del modello.

I risultati ottenuti per i QSVM sono inidicati nella tabella 3.2. Le percentuali ottenute con la Z, la ZZ e la feature map con entanglement si considerano paragonabili sia tra di loro che con il miglior kernel classico (il kernel gaussiano), poiché non differiscono mai di più del 2%. La feature map senza entanglment non permette di raggiungere accuratezze elevate, al contrario della Z feature map (anch'essa sprov-

samples	kernel gaussiano	kernel lineare
500	89,95%	76,90%
600	89,52%	74,72%
700	89,51%	76,23%
800	88,78%	76,19%
900	90,63%	74,60%
1000	88,83%	75,58%

Tabella 3.1: In questa tabella vengono riportate le accuratezze di classificazione ottenute per SVM classici con kernel lineare e gaussiano.

vista di entanglement), questo mostra come la strategia euristica di aggiungere delle rotazioni che non siano lungo Z ha portato ad uno svantaggio. Questo prova come il problema di ottimizzazione delle feature map sia un problema complesso. In generale non esistono delle strategie che permettono di raggiungere con certezza la feature map ottimale. Dalle tabelle 3.1 e 3.2 si osserva inoltre come all'aumentare dei samples non corrisponda un miglioramento dell'accuratezza.

samples	Z	ZZ	fmap 1	fmap 2
500	90,09%	89,88%	83,43%	89,88%
600	89,20%	89,20%	81,90%	89,72%
700	89,28 %	89,18%	81,56%	89,38%
800	88,36%	88,12%	82,23%	88,69%
900	90,29%	90,29%	83,49%	90,29%
1000	88,59%	87,84%	84,17 %	88%

Tabella 3.2: In questa tabella vengono riportate le accuratezze di classificazione ottenute per QSVM con i diversi kernel che differiscono dalle feature map introdotte nella sezione precedente. In questo caso con fmap 1 ci si riferisce alla feature map priva di entanglement mentre con fmap 2 a quella che invece lo presenta.

Siccome i risultati della feature map con entanglement sono confrontabili con quelli delle feature map già fornite dalla libreria qiskit, quest'ultima è stata scelta per affrontare la fase successiva dell'analisi dati.

3.3 Studio dell'exponential concentration

Per studiare il problema dell'exponential concentration (esposto in sezione 2.4.2) del Fidelity quantum kernel, è stato aumentato il numero di qubit che compongono la feature map illustrata in figura 3.14 scelta a seguito dell'ottimizzazione del QSVM descritta nella sezione precedente. Partendo da 4 qubit seguendo lo stesso schema logico presentato in figura 3.14 sono state costruite altre feature map da 6, 8, 10, 12 e 14 qubit. Andando a osservare la matrice di kernel si è potuto notare che, all'aumentare dei qubit essa tendeva sempre di più alla matrice identità, come si può vedere nelle figure 3.15 e 3.16, indicando che in questo caso le entrate al di fuori della diagonale si addensano ad un valore $\mu = 0$.

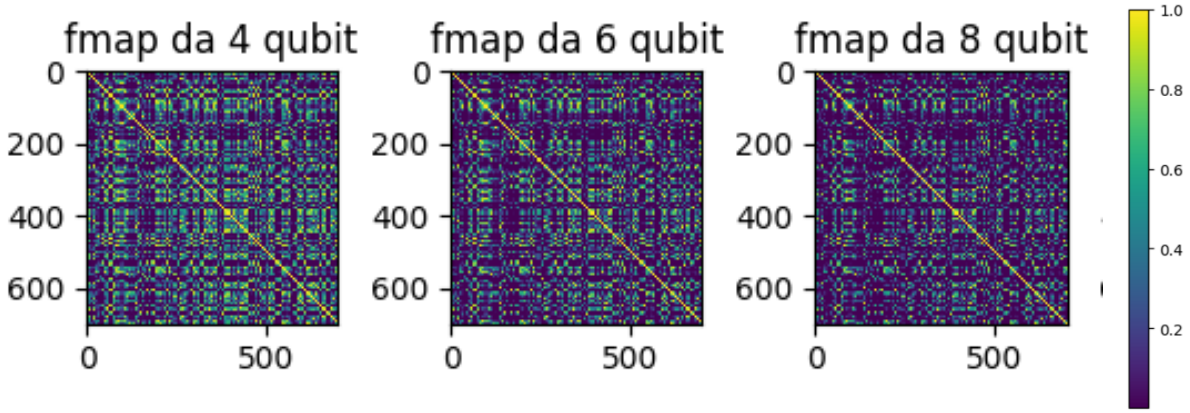

Figura 3.15: matrice di kernel per feature map da 4 a 8 qubit.

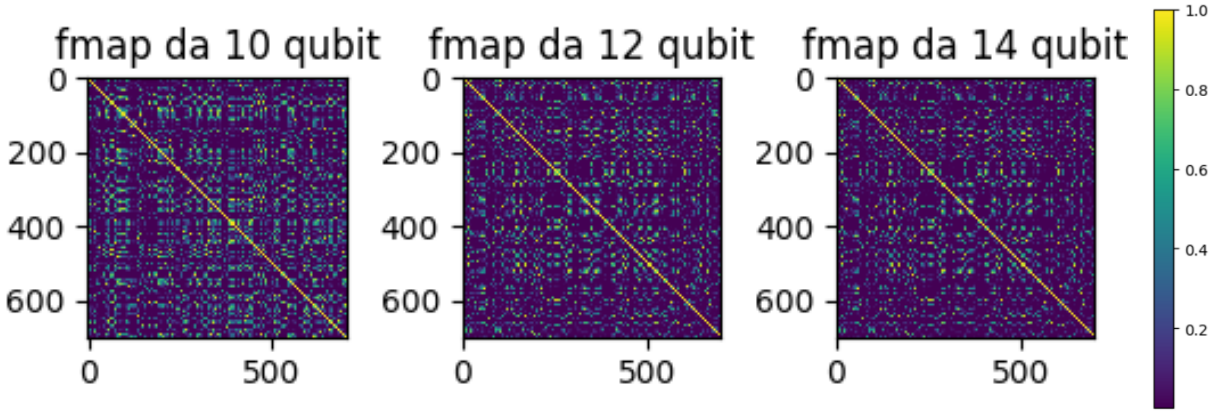

Figura 3.16: matrice di kernel per feature map da 10 a 14 qubit.

Per quantificare il fatto che la matrice si avvicini sempre di più all'identità è stata valutata la varianza degli elementi fuori dalla diagonale, ottenendo l'andamento

mostrato in figura 3.17, verificandone l'andamento qualitativamente esponenziale all'aumentare del numero di qubit.

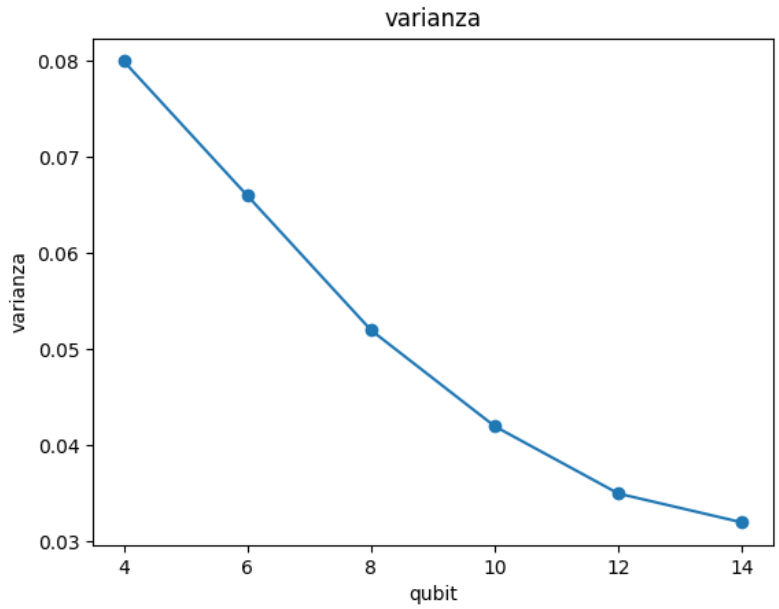

Figura 3.17: plot della varianza della matrice di kernel in funzione del numero dei qubit.

Come descritto in sezione 2.4.1, il fidelity quantum kernel dà la probabilità di ottenere di nuovo lo stato iniziale del qubit a seguito dell'applicazione della feature map. Per fattori statici, andando ad aumentare il numero di qubit che la compongono aumenta anche la dimensione dello spazio di Hilbert in cui si trovano, quindi in uno spazio di Hilbert di dimensione sempre maggiore la probabilità di riottenere lo stato iniziale è sempre più bassa e tende a zero, questa è la ragione per cui $\mu = 0$. Come spiegato in sezione 2.4.2 questo porta ad un aumento di precisione e accuratezza richieste al computer quantistico per stimare la funzione di classificazione (1.10).

3.3.1 Mitigazione dell'exponential concentration

L'operazione di mitigazione dell'exponential concentration si è svolta tramite l'utilizzo del Projected quantum kernel, che come descritto in sezione 2.4.1 permette una riduzione dello spazio che contiene lo stato finale del sistema. La funzione 2.28 che definisce il quantum kernel è stata implementata come mostrato in figura 3.18.

La simulazione di calcolo del projected quantum kernel si è mostrata molto meno efficiente di quella del fidelity quantum kernel, comportando un significativo rallentamento nell'analisi. Per questo motivo per ottimizzare la simulazione è stata calcolata solo la prima riga della matrice di kernel su un dataset da circa 600 samples allo

```python
def projected_quantum_kernel(fmap: QuantumCircuit, dataset: np.ndarray, gamma: float)-> np.ndarray:
    if not fmap.parameters:
        kernel_matrix = np.ones((dataset.shape[0], dataset.shape[0]))
        return  kernel_matrix
    kernel_matrix = np.zeros((dataset.shape[0], dataset.shape[0]))
    for i in range(dataset.shape[0]):
        for j in range(i):
            statevector_i_dm = DensityMatrix(fmap.assign_parameters(dataset[i]))
            statevector_j_dm = DensityMatrix(fmap.assign_parameters(dataset[j]))
            exp_term = 0
            for q in range(fmap.num_qubits):
                summed_qubits = [k for k in range(fmap.num_qubits) if k != q]
                exp_term = exp_term + np.linalg.norm(partial_trace(statevector_i_dm, summed_qubits)
                                                     - partial_trace(statevector_j_dm, summed_qubits))**2
            kernel_matrix[i, j] = np.exp(-gamma * exp_term)
    kernel_matrix =  kernel_matrix +  kernel_matrix.T + np.identity(dataset.shape[0])
    return  kernel_matrix
```

Figura 3.18: funzione che implementa il projected quantum kernel.

scopo di avere comunque una buona statistica per stimare la varianza in modo accurato. Lo studio è stato svolto confrontando la varianza della prima riga della matrice di kernel del projected quantum kernel con la prima del fidelity quantum kernel, con lo stesso dataset in entrambi i casi, ottenendo l'andamento mostrato in figura 3.19. Si è quindi osservato un miglioramento significativo per feature map con un numero di qubit superiore a 8, l'aumento percentuale è riportato nella tabella 3.3. Lo studio si è fermato a feature map da 12 qubit, anziché 14 come in sezione 3.3, sempre per problemi di efficienza computazionale.

n	varianza fidelity	varianza projected	aumento percentuale
4	0,108	0,109	0,92 %
6	0,09	0,102	13,33 %
8	0,076	0,096	26,32 %
10	0,066	0,09	36,36 %
12	0,056	0,075	33,93 %

Tabella 3.3: In questa tabella è riportato l'aumento percentuale della varianza delle matrici di kernel ottenute con il Projected quantum kernel rispetto a quelle ottenute con il Fidelity quantum kernel, dove n indica il numero di qubit che presenta la feature map.

Per quanto riguarda le accuratezze di classificazione in questo caso i risultati non sono conclusivi poiché è stata presa in considerazione solo una riga della matrice.

Uno studio su un dataset di dimensioni molto ridotte ha permesso di confrontare anche le accuratezze ottenute coi due diversi quantum kernel, oltre che le varianze delle matrici, fino a feature map da 10 qubit. In questo caso il dataset presenta 50

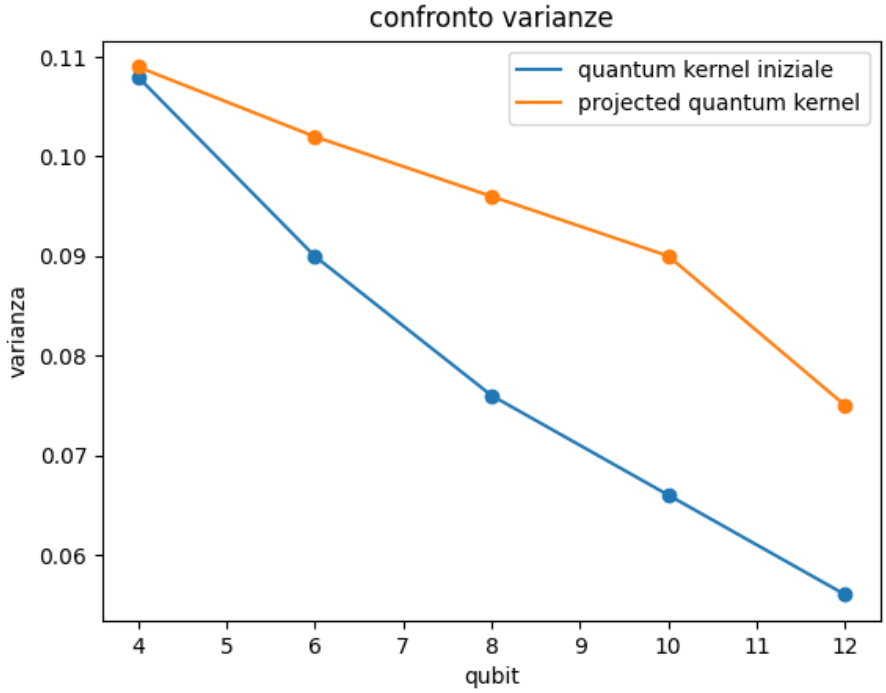

Figura 3.19: in figura è mostrato l'andamento della varianza del projected quantum kernel in arancione e del fidelity quantum kernel in blu all'aumentare dei qubit che compongono la feature map.

samples da 10000 shots ciascuno. Nelle figure 3.20 e 3.21 si possono osservare le matrici di kernel e l'andamento della varianza al variare del numero di qubit. In questo caso si osserva un miglioramento della varianza per le feature map da 8 e 10 qubit del $3,7\%$ e $29,27\%$ rispettivamente.

Il ridotto numero di dati comporta fluttuazioni statistiche importanti nell'accuratezza, quindi per valutarla è stato utilizzato il metodo della cross validation. Nella cross-validation il dataset viene suddiviso in k parti uguali (o quasi uguali), nel caso di questo studio k=5, dove k-1 sono destinate al training mentre una al test e si calcola l'accuratezza, questo processo viene iterato k volte cambiando la parte di dataset destinata al test per poi fare una media delle accuratezze ottenute. In questo modo il modello sfrutta tutto il dataset a disposizione per stimare la performance del modello [22]. Le accuratezze ottenute con projected quantum kernel, illustrate nella tabella 3.4, sono paragonabili a quelle ottenute con il Fidelity quantum kernel.

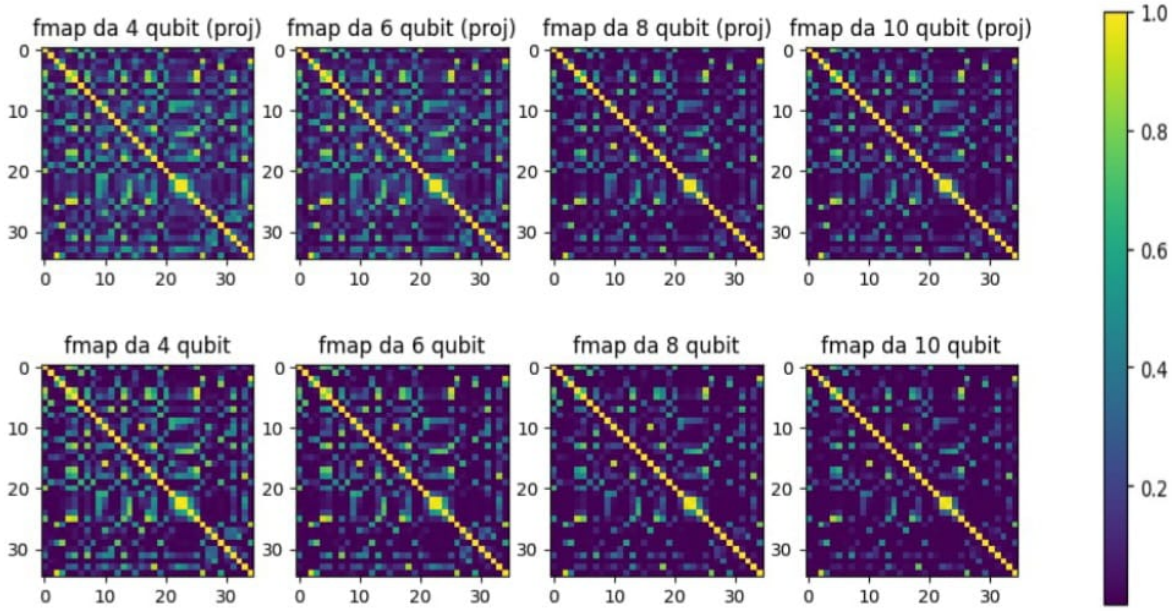

Figura 3.20: Le matrici nella riga superiore sono le matrici ottenute con il projected quantum kernel, quelle nella riga inferiore sono le matrici ottenute col fidelity quantum kernel.

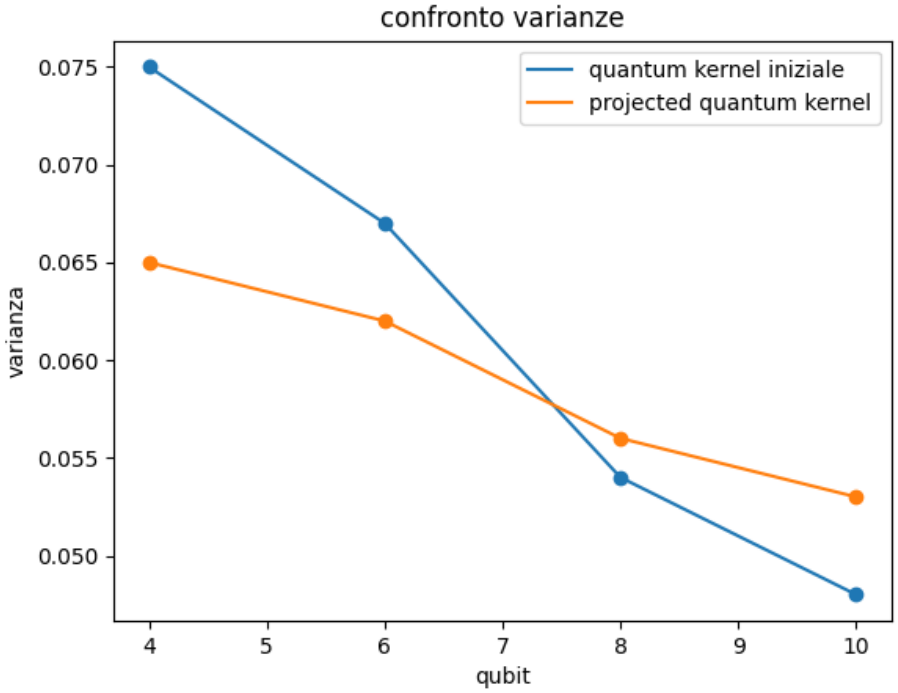

Figura 3.21: in figura è mostrato l'andamento della varianza del projected quantum kernel in arancione e del fidelity quantum kernel in blu all'aumentare del numero di qubit che compongono la feature map.

n	accuratezza fidelity	accuratezza projected
4	86,97%	86,67%
6	86 %	86,67 %
8	86,67 %	86,67%
10	88,57%	86,67%

Tabella 3.4: Nella tabella vengono mostrate le accuratezze ottenute tramite cross validation nel caso del fidelity quantum kernel e projected quantum kernel dove n è il numero di qubit che compongono la feature map.

Capitolo 4

Conclusioni

Il Quantum Machine Learning è un settore di ricerca in rapido sviluppo che ha le potenzialità di essere utilizzato in numerose aree scientifiche. Questa tesi affronta una problematica tipica del Quantum Support Vector Machine (QSVM), ossia la controparte quantistica di un noto algoritmo di classificazione supervisionata chiamato Support Vector Machine (SVM), nella classificazione di distribuzioni estratte da due computer quantistici Noisy Intermediate Scale Quantum (NISQ). I SVM e i QSVM si basano sul concetto di kernel e quantum kernel, delle funzioni che permettono di valutare la similarità tra coppie di elementi di un dataset, quindi cruciali nel determinare la performance del modello. La prima parte dello studio si è concentrata sul problema di ottimizzazione sia di SVM classiche che QSVM. Per il support vector machine sono stati utilizzati il kernel lineare ed il kernel gaussiano mentre per il QSVM è stato utilizzato il fidelity quantum kernel, il quale dà la probabilità di ottenere di nuovo lo stato iniziale del sistema a multi-qubit in seguito all'applicazione della feature map. Al variare del numero di samples nel dataset, è stato mostrato che le accuratezze di classificazione del Quantum Support Vector Machine con fidelity quantum kernel per 3 feature map diverse: la Z feature map, la ZZ feature map e la feature map con entanglement creata appositamente per l'elaborato, sono paragonabili a quelle ottenute col miglior support vector machine classico, ossia quello col kernel gaussiano, con accuratezze di classificazione vicine al 90%, mostrando inoltre che l'aumento del numero di samples non porta ad un miglioramento delle performance del modello. Il SVM con kernel lineare ha presentato delle accuratezze di classificazione intorno al 75%, quindi molto più basse del kernel non lineare, provando che le distribuzioni delle features nelle due classi non possono essere separate al meglio da un iperpiano senza ricorrere a feature map non banali. Nel caso del QSVM l'unica feature map a non aver dato risultati paragonabili al kernel gaussiano è stata una feature map priva di entanglment tra i qubit, che presenta dei gate di rotazione lungo x e y anziché averli solo lungo z, come nel caso della Z feature map, mostrando che aggiungere questi due tipi di rotazione alla feature map porta un calo della performance del modello. La seconda fase dell'elaborato si è concentrata sullo

studio del problema a cui è soggetto il fidelity quantum kernel, ossia l'exponential concentraton. È stato scelto il QSVM con la feature map con entanglment a cui è stato aumentato gradualmente il numero di qubit. All'aumentare del numero di qubit è stato verificato che il valore della funzione di kernel valutata tra elementi diversi del dataset si avvicina a zero indipendentemente dalla coppia di sample in esame. Per quantificarlo è stata calcolata la varianza degli elementi di matrice ottenuta valutando il kernel su ogni coppia di sample, escludendo la diagonale, mostrando che questa tende a zero con un andamento esponenziale. Questo studio ha confermato che l'aumento della dimensione dello spazio di Hilbert del sistema a multi-qubit utilizzato per calcolare il kernel porta ad una probabilità sempre più bassa di ottenere di nuovo lo stato iniziale in seguito all'evoluzione dello stato regolata dalla feature map. In presenza di exponential concentration, i QSVM si rivelano inefficaci poiché il rumore di un dispositivo quantistico NISQ sovrasta le differenze del valore della funzione di kernel per elementi del dataset diversi. Utilizzando al posto del fidelity quantum kernel un tipo di kernel chiamato projected quantum kernel si è osservato un miglioramento significativo del valore della varianza quando il numero di qubit utilizzati era superiore a otto. La varianza è aumentata del 26,32% per otto qubit, del 36,36% per dieci qubit e 33,93%. Questa parte dello studio, per problemi di simulazione del projected quantum kernel, è stata fatta confrontando la varianza solo di una riga della matrice di kernel e per questo motivo non è stato possibile ottenere risultati conclusivi sulle accuratezze di classificazione del projected quantum kernel. Sono state comunque ottenute delle stime più approssimative, su di un dataset contenente solo 50 elementi. Anche in questo caso la varianza, calcolata tutti gli elementi del dataset ad esclusione di quelli sulla diagonale, è aumentata per feature map superiori a 8 qubit. Inoltre sono state ottenute delle accuratezze superiori all'86%. Si può concludere affermando il successo del projected quantum kernel nel mitigare l'exponential concentration e che la performance del QSVM è pragonabile a quella del SVM classico. Lo studio e l'ottimizzazione di nuovi ed efficienti algoritmi di QML per l'analisi dati, assieme allo sviluppo di nuove strategie per mitigarne gli aspetti critici, potrebbero portare a nuovi algoritmi in grado di affiancare oppure sostituire quelli classici già esistenti, applicandoli in numerose aree della Fisica.

Bibliografia

[1] Scott Aaronson. ≪The learnability of quantum states≫. In: *The Royal Society* (2007).

[2] Thomas E. Roth et al. ≪An ntroduction to the Transmon Qubit for Electromagnetic Engineers≫. In: *arXiv.org* (2021).

[3] Abiodun M. Ikotun et al. ≪K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data≫. In: *Information Science* (2022).

[4] Frank Arute et al. ≪Quantum supremacy using a programmable superconducting processor≫. In: *Nature* (2019).

[5] Hsin-Yuan Huang et al. ≪Power of data in quantum machine learning≫. In: *Nature communication* (2021).

[6] Leonardo Alchieri et al. ≪An introduction to quantum machine learning: from quantum logic to quantum deep learning≫. In: *Springer* (2021).

[7] P. Krantz et al. ≪A Quantum Engineer's Guide to Superconducting Qubits≫. In: *arXiv.org* (2021).

[8] Supanut Thanasilp et al. ≪Exponential concentration and untrainability in quantum kernel methods≫. In: *arXiv.org* (2022).

[9] Schiazza Filippo Antonio. *Applicazioni di machine learning alla fisica delle alte energie*. AA 2019-2020.

[10] Samuele Campaniello. *Paradosso EPR, teoremi di Bell e misure in meccanica quantistica*. AA 2022/2023.

[11] Davide Castelvecchi. ≪IBM releases the first-ever 1000-qubit quantum chip≫. In: *nature* (2022).

[12] scikit-learn developers. *1.4. Support Vector Machines*. URL: `https://scikit-learn.org/stable/modules/svm.html`.

[13] Paolo Dotti. *Supervised learning, cos'è, esempi di apprendimento supervisionato*. 2022. URL: `https://www.ai4business.it/intelligenza-artificiale/supervised-learning-cose-esempi-di-apprendimento-supervisionato/`.

[14] Richard P. Feynman. ≪Simulating Physics with Computers≫. In: *International Journal of Theoretical Physics* (1982).

[15] Federcio Shin'ichi Finardi. *Algoritmi qunatistici per la risoluzione di equazioni differenziali*. AA 2022-2023.

[16] Marti A. Hearst. ≪Support Vector Machines≫. In: *IEEE Intelligent System* (1998).

[17] IBM. *Che cos'è il machine learning*. URL: https://www.ibm.com/it-it/topics/machine-learning.

[18] IBM. *Cos'è Quantum Computing?* URL: https://www.ibm.com/it-it/topics/quantum-computing.

[19] H.-T. Lin C.-J. Lin. ≪A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods≫. In: *Neural Computation* (2003).

[20] Isaac L. Chuang Michael A. Nielsen. *Quantum Computation and Quantum Information*. Cambridge Univeristy press, 2000.

[21] Roberto Moretti. *Feature Extraction Neural Networks for Quantum Kernel classifiers: low-energy background rejection in Xenon-Doped LArTPC detector*. AA 2021-2022.

[22] Liu H. Refaeilzadeh P. Tang L. *Encyclopedia of Database Systems*. Springer, 2009.

[23] Isha Salian. *SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning?* URL: https://blogs.nvidia.com/blog/supervised-unsupervised-learning/.

[24] Maura Sandri. *Primi passi verso l'intelligenza artificiale quantistica*. 2021. URL: https://www.media.inaf.it/2021/10/28/quantum-ai/.

[25] G. James D. Witten T. Hastie R. Tibshirani. *An Introduction to Statistical Learning*. 2ª ed. 2023.

[26] Alan M. Turing. ≪Computing Machinery and Intelligence≫. In: *Mind* (1950).

[27] Zhi-Hua Zhou. *Machine Learning*. Springer, 2021.