UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA

Corso di Laurea Magistrale in Ingegneria Informatica

Quantum Computing

Classificazione del Diabete con VQC: Un'applicazione del Quantum Machine Learning

Antonio Pio Francica 269017, Rocco Pio Vardaro 264097

ANNO ACCADEMICO 2024-2025

Indice

Capitolo 1

Introduzione

Negli ultimi anni, la classificazione automatica di dati ha assunto un ruolo centrale in numerosi ambiti applicativi, tra cui medicina, finanza e sicurezza. Nel settore sanitario, l'uso di algoritmi di Machine Learning ha mostrato un enorme potenziale per supportare le diagnosi precoci e il monitoraggio di malattie croniche. Tuttavia, l'incremento della complessità dei dati e la necessità di elaborazione sempre più efficiente richiedono soluzioni computazionali innovative.

In questo contesto si inserisce il quantum computing, una disciplina emergente che promette di rivoluzionare il paradigma computazionale tradizionale sfruttando le proprietà della meccanica quantistica. Tra i vari approcci proposti per applicare il quantum computing al machine learning, uno dei più promettenti è il **Variational Quantum Classifier** (VQC). Questo metodo combina circuiti quantistici parametrizzati con strategie di ottimizzazione classiche e fornisce una possibile via per migliorare le prestazioni dei modelli di classificazione, soprattutto, in presenza di strutture dati complesse e non lineari.

L'elaborato proposto esplora l'applicazione del VQC a un caso di studio reale: la classificazione di dati relativi al diabete, basata su un dataset pubblico comunemente utilizzato nella letteratura scientifica. Dopo una breve panoramica sui fondamenti del Machine Learning e del Quantum Computing, vengono descritti l'architettura del modello VQC, il processo di pre-processing

dei dati, la fase di addestramento e i risultati ottenuti. L'obiettivo è esaminare la fattibilità e l'efficacia di questo approccio quantistico nell'affrontare un concreto problema di classificazione in ambito medico, confrontandone le prestazione con quelle della corrispondente metodologia classica.

Capitolo 2

Descrizione e Preprocessing Dataset

2.1 Caratteristiche principali

In letteratura sono disponibili diversi dataset pubblici utilizzati per la previsione e la classificazione del diabete. L'esame dei dataset ha consentito di osservare che la letalità del diabete risulta maggiore nelle donne rispetto agli uomini. Nel 2019, infatti, i decessi attribuiti a questa malattia hanno interessato a 2,3 milioni di donne. Nello stesso periodo, invece, gli uomini deceduti sono stati 1,9 milioni. Lo stile di vita odierno, sempre più basato su alimenti processati e una ridotta attività fisica,inoltre, contribuisce ad accrescere il rischio di sviluppare la malattia.

Per analizzare in modo più accurato il rischio di diabete nella popolazione femminile, in questo studio è stato utilizzato il *Pima Indians Diabetes Dataset* (PIDD) sviluppato dal National Institute of Diabetes and Digestive and Kidney Diseases. Questo dataset è considerato tra i più affidabili e versatili per lo studio predittivo del diabete.

Il dataset PIDD comprende 768 osservazioni relative a donne di età superiore ai 21 anni. Tra queste, 500 casi sono classificati come non diabetiche, mentre le restanti 268 presentano una diagnosi di diabete. Il dataset è ampiamente impiegato per prevedere la probabilità di insorgenza del diabete

sulla base di otto variabili indipendenti ritenute altamente significative. Una descrizione dettagliata di queste caratteristiche, accompagnata da un'analisi statistica descrittiva del dataset, è riportata nella tabella sottostante.

SN	Feature	Descrizione	Intervallo	Media	Deviazione std
1	Pregnancies	Numero di gravidanze	0–17	3.85	3.37
2	Glucose	Glicemia a 2h nel test OGTT	0–199	120.90	31.97
3	Blood pressure	Pressione diastolica (mmHg)	0–122	69.11	19.36
4	Skin thickness	Spessore piega cutanea tricipite (mm)	0–99	20.54	15.95
5	Insulin	Insulina sierica a 2h (mu U/ml)	0–846	79.81	115.24
6	Body mass index	Indice massa corporea ($\frac{\text{peso (kg)}}{\text{altezza (m)}^2}$)	0–67.1	31.99	7.88
7	Diabetes pedigree	Familiarità genetica per diabete	0.08–2.42	0.47	0.33
8	Age	Età (anni)	21–81	33.24	11.76

Tabella 2.1: Descrizione e statistiche principali delle feature del dataset.

2.1.1 Grafici e statistiche

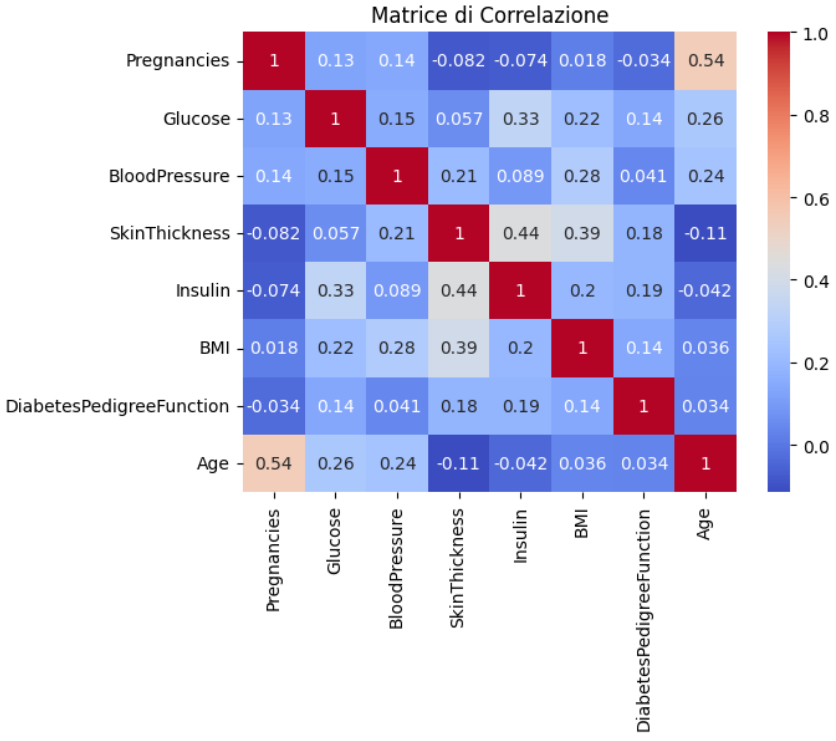

Figura 2.1: Matrice di correlazione tra features

Figura 2.2: Correlazione tra coppie di variabili

2.2 Pre-processing del Dataset

Il preprocessing del dataset Pima Indians Diabetes Dataset (PIDD) è stato strutturato in più fasi, con l'obiettivo di migliorare la qualità dei dati e preparare un sottoinsieme più informativo e bilanciato per l'addestramento del modello di classificazione. Le operazioni effettuate sono state le seguenti:

- **Rimozione della colonna "Age"**: la variabile è stata esclusa dall'analisi per ridurre la dimensionalità del dataset vista l'elevata correlazione con altre features e per minimizzare l'eventuale impatto di bias legati all'età.

- **Selezione automatica delle feature (SelectKBest)**: per ridurre il rumore informativo e migliorare le prestazioni del modello quantistico, è stato applicato il metodo SelectKBest con funzione statistica `Fclassif` (ANOVA F-test). Questo processo ha permesso di selezionare automaticamente le k variabili indipendenti, più rilevanti rispetto alla variabile target, riducendo la dimensionalità, mantenendo l'informazione discriminante.

- **Bilanciamento del dataset tramite sottocampionamento**: per affrontare il problema dello sbilanciamento tra classi (diabetici e non diabetici), è stato adottato un approccio di undersampling: è stato selezionato casualmente un uguale numero di esempi per ciascuna classe, per garantire un dataset bilanciato ed evitare che il modello sviluppi una preferenza verso la classe maggioritaria.

Nel complesso, questo approccio di preprocessing ha permesso di ottenere un dataset più pulito, bilanciato e concentrato sulle feature più significative, rendendolo più adatto all'addestramento di un classificatore basato su tecniche di machine learning, incluso l'approccio quantistico VQC.

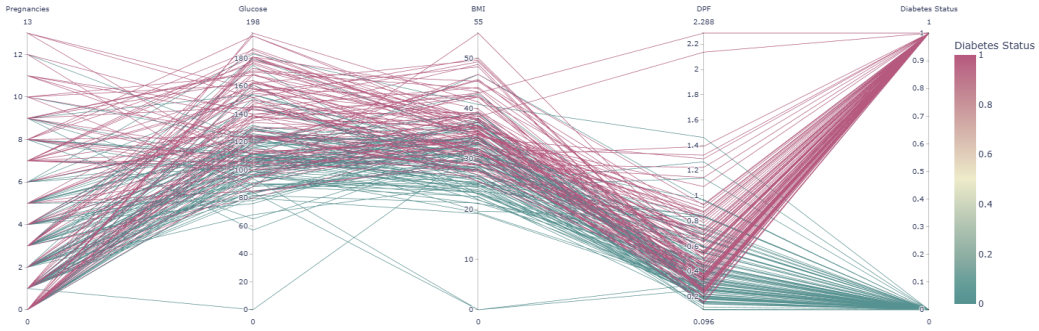

Figura 2.3: Grafico a coordinate parallele (dopo il pre-processing)

Capitolo 3

Tecnologie Utilizzate e Metriche di Performance

3.1 Librerie Python

Nel presente progetto sono state impiegate diverse librerie Python, sia per la parte di machine learning classico, sia per l'implementazione e l'ottimizzazione del modello quantistico di classificazione. Di seguito viene fornita una descrizione delle principali librerie coinvolte:

Librerie per il preprocessing e la modellazione classica:

- **Pandas**: utilizzata per la gestione e la manipolazione dei dati in formato tabellare. Ha permesso di caricare, esplorare e preprocessare il dataset, oltre ad effettuare operazioni come la sostituzione di valori anomali, la selezione delle colonne e il bilanciamento delle classi.

- **Numpy**: utilizzata per operazioni numeriche di basso livello, come la gestione di array, la computazione di statistiche descrittive e il supporto a funzioni vettoriali e matriciali essenziali per l'elaborazione dei dati.

- **Scikit-learn (sklearn)**: libreria fondamentale per l'applicazione di tecniche di machine learning classiche. In particolare, è stata utilizzata per la selezione delle feature con SelectKBest e Fclassif, la divisione del data-

set in sottoinsiemi, l'addestramento di modelli di classificazione classici e la valutazione delle performance tramite metriche standard.

- **Qiskit**: principale framework per la progettazione e l'esecuzione di algoritmi quantistici sviluppato da IBM. È stato utilizzato per costruire circuiti parametrizzati impiegati nel modello di classificazione quantistica (VQC – Variational Quantum Classifier). In particolare, per i circuiti che mappano i dati classici nello spazio di Hilbert, tramite feature map quantistiche, e per circuiti variazionali usati come ansätze per l'ottimizzazione dei parametri del classificatore. Tali componenti sono stati combinati per costruire il circuito parametrico su cui si basa il VQC.

- **Scipy**: utilizzata per l'ottimizzazione dei parametri del circuito quantistico. È stata impiegata come ottimizzatore classico, per minimizzare la funzione di costo associata al classificatore quantistico, in sostituzione o complemento agli ottimizzatori inclusi in Qiskit (es. COBYLA, SPSA, ecc.).

Queste librerie, integrate in un flusso coerente, hanno consentito la realizzazione di un confronto tra approcci classici e quantistici alla classificazione binaria, con un focus applicativo sul problema della diagnosi del diabete.

3.2 Metriche di valutazione

Le metriche scelte per valutare il modello di classificazione binaria sono le seguenti:

3.2.1 Accuracy

L'accuracy è la misura più semplice di valutazione di un modello. Indica la percentuale di previsioni corrette rispetto al totale dei campioni.

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

TP (True Positives) sono i pazienti diabetici predetti correttamente come malati.

TN (True Negatives) sono i pazienti sani predetti correttamente e classificati come non diabetici.

FP (False Positives) sono i pazienti sani che il modello ha erroneamente classificato come malati.

FN (False Negatives) sono i pazienti che presentano diabete che il modello ha erroneamente classificato come pazienti sani.

L'accuracy può essere fuorviante se le classi sono sbilanciate.

3.2.2 Precision

La precision indica quanti pazienti classificati come "diabetici" hanno effettivamente la malattia.

$$\frac{TP}{TP + FP} \tag{3.2}$$

Un valore basso di Precision indica che molti pazienti sani sono erroneamente classificati come malati.

3.2.3 Recall

Il recall indica quanti pazienti effettivamente malati vengono riconosciuti dal modello.

$$\frac{TP}{TP + FN} \tag{3.3}$$

Un valore basso di Recall indica che molti pazineti con il diabete non vengono riconosciuti.

3.2.4 F1 Score

L'F1-score è una media armonica tra Precision e Recall, utile quando le classi sono sbilanciate.

$$2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.4}$$

Un valore alto di F1-score significa che il modello ha buona precisione e buon recall contemporaneamente.

3.2.5 Confusion Matrix

La confusion matrix è una tabella che mostra il numero di predizioni corrette ed errate suddivise per classe. Consente di comprendere dove il modello sta sbagliando. Per ottenere le quattro categorie (TP, TN, FP, FN) si contano le occorrenze di ogni combinazione (target, prediction) Il risultato è una matrice del tipo:

$$\begin{bmatrix} TN & FN \\ FP & TP \end{bmatrix}$$

Capitolo 4

Modello di classificazione con alberi decisionali

Il classificatore **Decision Tree** (albero decisionale) è uno degli algoritmi più intuitivi e interpretabili nell'ambito del machine learning supervisionato. Viene utilizzato per risolvere sia problemi di classificazione che di regressione.

L'idea centrale è quella di suddividere lo spazio dei dati in sottoinsiemi più omogenei rispetto alla variabile target (Non-Diabetici, Diabetico), usando una struttura ad albero. Ogni nodo interno dell'albero rappresenta una decisione basata su una caratteristica, ogni ramo rappresenta un esito della decisione, e ogni foglia rappresenta una classe predetta. L'obiettivo finale consiste nel dividere ripetutamente il dataset in modo da massimizzare l'omogeneità delle classi all'interno di ogni nodo. L'ideale è che ogni nodo finale (foglia) contenga esempi appartenenti a una sola classe.

Uno dei principali punti di forza di questo algoritmo è la sua chiarezza logica: le decisioni vengono prese in modo simile al ragionamento umano, attraverso una sequenza di domande semplici basate sulle caratteristiche dei dati. Questo lo rende facilmente comprensibile anche da chi non ha competenze tecniche avanzate, permettendo di analizzare e giustificare in modo trasparente le predizioni effettuate. I Decision Tree sono molto flessibili, poiché possono gestire sia variabili numeriche che categoriche. Sono anche in grado di adattarsi a strutture complesse dei dati, segmentando lo spazio delle caratteristiche in

modo dinamico e gerarchico. Tale flessibilità, tuttavia, può diventare un'arma a doppio taglio: i Decision Tree tendono, infatti, ad essere molto sensibili alle variazioni nei dati di input, e in particolare sono soggetti a fenomeni di overfitting.

4.1 Implementazione del modello: Decision Tree

Dopo aver preparato i dati, si procede con la **scelta del modello di classificazione** più adatto al problema. In questo caso specifico, è stato selezionato l'algoritmo Decision Tree. La scelta dell'algoritmo non è casuale, ma dipende da diversi fattori, quali: la dimensione del dataset, la presenza di rumore nei dati, il numero di caratteristiche e la necessità di interpretabilità dei risultati.

Prima di addestrare il modello, abbiamo suddiviso il dataset in due parti: **Training set** e **Test set**. Questa suddivisione risulta fondamentale per valutare correttamente le prestazioni del modello. Il training set viene utilizzato per insegnare al modello a riconoscere le relazioni tra le caratteristiche e le etichette, mentre il test set, costituito da dati mai utilizzati durante l'addestramento, serve a valutare la capacità del modello di generalizzare e di effettuare previsioni su nuovi dati. La suddivisone utilizzata in questo caso è la seguente: 80% train e 20% test.

Una volta selezionato l'algoritmo, si passa alla fase di **addestramento del modello**. In questa fase, il modello viene esposto ai dati di addestramento. Il compito del modello è "*imparare*" a riconoscere le relazioni tra le caratteristiche e le etichette. Questo avviene attraverso un processo iterativo, in cui l'algoritmo cerca di minimizzare l'errore tra le predizioni fatte e le etichette reali.

Al termine dell'addestramento, si procede con la **valutazione del modello**, che consiste nel testarne l'efficacia su dati mai osservati in precedenza, ovvero quelli appartenenti al test set. Questo passaggio risulta fondamentale per verificare se il modello ha imparato a generalizzare oppure se si è sempli-

cemente adattato troppo ai dati di addestramento (overfitting). Per valutare le prestazioni si usano metriche come quelle descritte nel capitolo precedente.

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

Figura 4.1: Codice python sklearn.

4.2 Performance del modello

In questa sezione vengono analizzate le performance del modello Decision Tree applicato al dataset in esame. Per valutare l'efficacia del classificatore, si fa riferimento alle principali metriche di classificazione ottenute tramite il metodo `classification_report()` di Scikit-learn. A completamento dell'analisi, viene riportata anche la confusion matrix, rappresentata graficamente, che consente di visualizzare, in modo immediato, la distribuzione delle predizioni corrette e degli errori commessi dal modello per ciascuna classe. Questa combinazione di strumenti fornisce una panoramica completa e dettagliata delle prestazioni del modello.

4.2.1 Performance Dataset 3 Feature

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.76	0.82	21
1	0.77	0.89	0.83	19
accuracy			0.82	40
macro avg	0.83	0.83	0.82	40
weighted avg	0.83	0.82	0.82	40

Figura 4.2: Risultati di classificazione

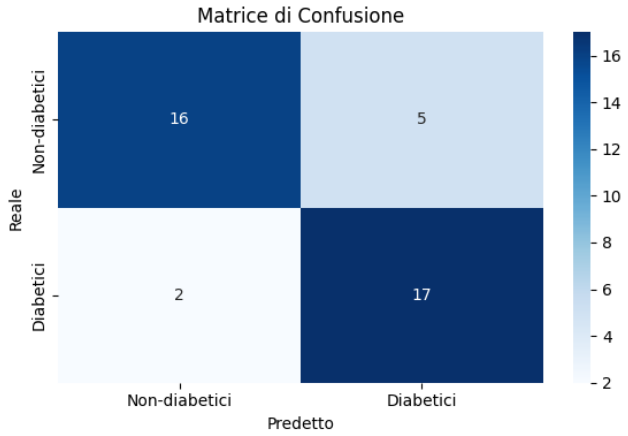

Figura 4.3: Confusion Matrix

4.2.2 Performance Dataset 4 Feature

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.81	0.85	21
1	0.81	0.89	0.85	19
accuracy			0.85	40
macro avg	0.85	0.85	0.85	40
weighted avg	0.85	0.85	0.85	40

Figura 4.4: Risultati di classificazione

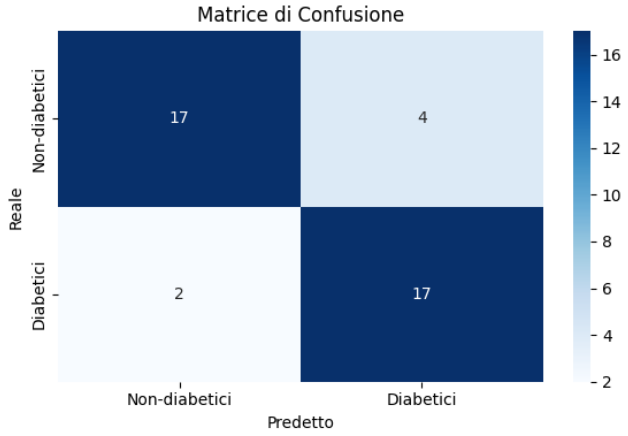

Figura 4.5: Confusion Matrix

4.2.3 Performance Dataset 5 Feature

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.71	0.79	21
1	0.74	0.89	0.81	19
accuracy			0.80	40
macro avg	0.81	0.80	0.80	40
weighted avg	0.81	0.80	0.80	40

Figura 4.6: Risultati di classificazione

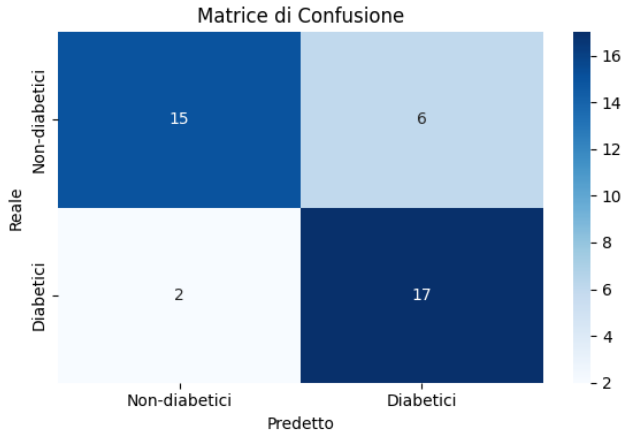

Figura 4.7: Confusion Matrix

Capitolo 5

Classificazione tramite algoritmo VQC

Il Variational Quantum Classifier (VQC) è un modello ibrido che unisce il meglio del calcolo quantistico e di quello classico, offrendo un approccio innovativo alla classificazione di dati. L'idea alla base di questo modello è quella di sfruttare il potenziale della computazione quantistica per rappresentare e trasformare i dati in modi non facilmente replicabili dai modelli classici.

Il funzionamento di un VQC si articola in quattro fasi principali:

1. **Codifica dei dati classici in stati quantistici**: si parte da dati classici, ad esempio punti in uno spazio bidimensionale o vettori in uno spazio n-dimensionale, che vengono codificati negli stati di qubit. Questa fase permette di "trasportare" l'informazione nel dominio quantistico.

2. **Circuito quantistico parametrico (Ansatz)**: Viene progettato un circuito quantistico il cui comportamento dipende da un insieme di parametri reali. La modifica di questi parametri consente di controllare la trasformazione dello stato quantistico, permettendo al modello di apprendere informazioni necessirie ai fini della classificazione.

3. **Misurazione e Previsione**: al termine del circuito, alcuni qubit vengono misurati. Tipicamente si calcola il valore atteso di determinati

operatori di Pauli. Il risultato è un valore numerico che viene interpretato come predizione del modello. Nei problemi di classificazione binaria, ad esempio, questo valore può essere convertito in una classe (ad esempio 1 o 0) applicando una funzione di soglia.

4. **Ottimizzazione classica**: sebbene l'elaborazione avvenga tramite un circuito quantistico, la fase di apprendimento si basa su tecniche di ottimizzazione classiche. In particolare, si definisce una *funzione di perdita* che misura quanto le previsioni del modello si discostano dalle etichette reali. L'obiettivo è quello di trovare i valori dei parametri quantistici che minimizzano questa funzione.

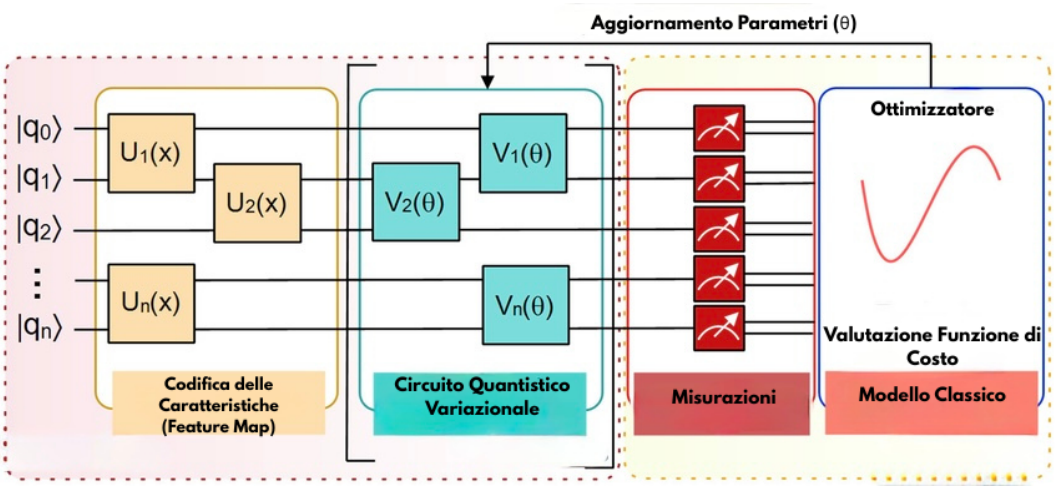

Figura 5.1: Struttura VQC

5.1 Encoding

L'obiettivo del circuito di codifica è rappresentare i dati all'interno di un circuito quantistico, rendendoli accessibili e manipolabili secondo i principi della computazione quantistica. In questa fase, sono state prese in considerazione principalmente due strategie di codifica: la codifica tramite angoli (angle encoding) e la codifica tramite ampiezze (amplitude encoding).

5.1.1 Angle Encoding

Questa modalità consiste nel tradurre i valori numerici, che rappresentano le feauture principali dei dati, in angoli che vengono utilizzati come parametri per specifiche porte di rotazione nel circuito quantistico. Ogni valore numerico viene mappato in un angolo, che, a sua volta, determina la rotazione di un qubit nello spazio di Hilbert, secondo una determinata direzione (come gli assi X, Y o Z). Le prestazioni del modello possono dipendere in maniera significativa dal tipo di porta rotazionale utilizzata per la codifica dei dati nel circuito quantistico.

5.1.2 Amplitude Encoding

Questo approccio si basa sull'idea di rappresentare le informazioni numeriche direttamente nelle ampiezze degli stati quantistici di un sistema.

In pratica, un vettore di dati classico viene normalizzato e utilizzato per costruire uno stato quantistico in cui ciascun elemento del vettore corrisponde a una delle componenti dell'ampiezza dello stato quantistico complessivo. In questo modo, un singolo stato quantistico può racchiudere, in maniera compatta, un gran numero di informazioni.

Uno dei principali vantaggi dell'Amplitude Encoding è l'efficienza in termini di risorse. In tal senso, infatti, è possibile rappresentare un vettore di dimensione 2^n utilizzando soltanto n qubit. Questo rende la tecnica particolarmente adatta in scenari in cui si ha a disposizione un numero limitato di qubit ma si vuole lavorare con dati ad alta dimensionalità.

5.1.3 Tecniche di Encoding in Qiskit

Il pacchetto Qiskit Machine Learning include un'ampia gamma di circuiti preconfigurati utilizzabili per mappare le diverse feature negli stati quantistici.

In particolare, le `PauliFeatureMap` e le `ZFeatureMap` sfruttano rotazioni derivate dalle matrici di Pauli per codificare le informazioni. La PauliFeatureMap è una mappa di feature molto flessibile, in grado di adattarsi a diversi

tipi di dati. Il circuito prevede, inizialmente, l'applicazione di porte Hadamard, seguite da porte di fase che variano in base al qubit. In seguito, viene implementata una rete di entanglement che collega tra loro i qubit. Dopo l'entanglement, vengono applicate ulteriori porte di fase che dipendono dall'interazione tra i qubit collegati. Le rotazioni utilizzate possono essere attorno agli assi X, Y o Z (rotazioni di primo ordine) oppure attorno alle coppie XX, YY o ZZ (rotazioni di secondo ordine), o anche una combinazione di queste.

La ZZ Feature Map, nello specifico, è una mappa di secondo ordine che utilizza rotazioni del tipo ZZ. Anche in questo caso, il circuito è composto da porte Hadamard e di fase seguite da entanglement e rotazioni specifiche.

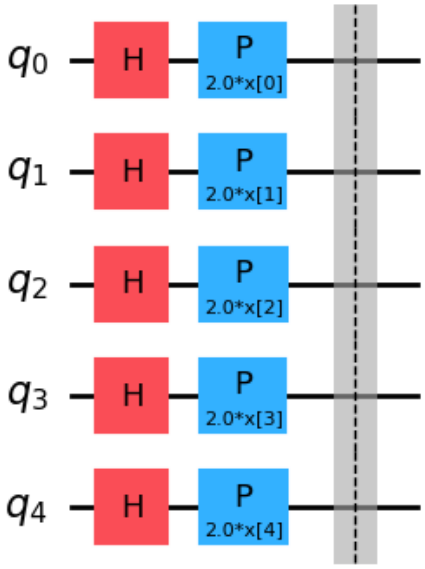

Figura 5.2: ZFeatureMap con 5 Qubit

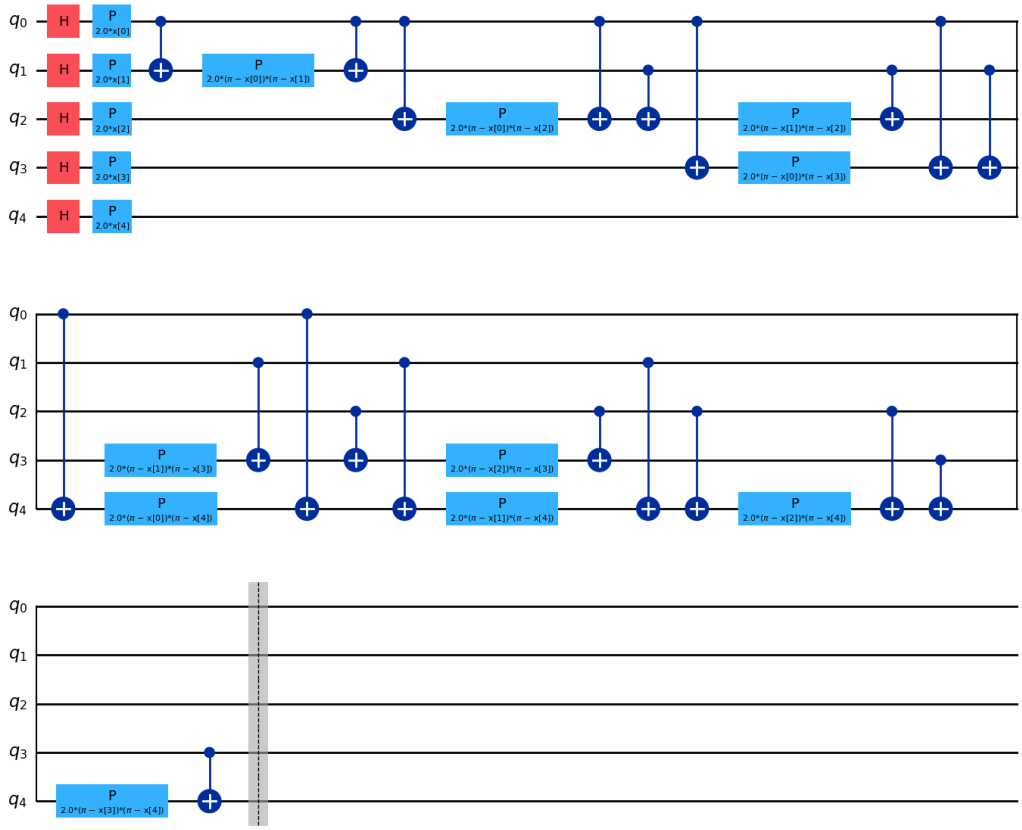

Figura 5.3: ZZFeatureMap con 5 Qubit

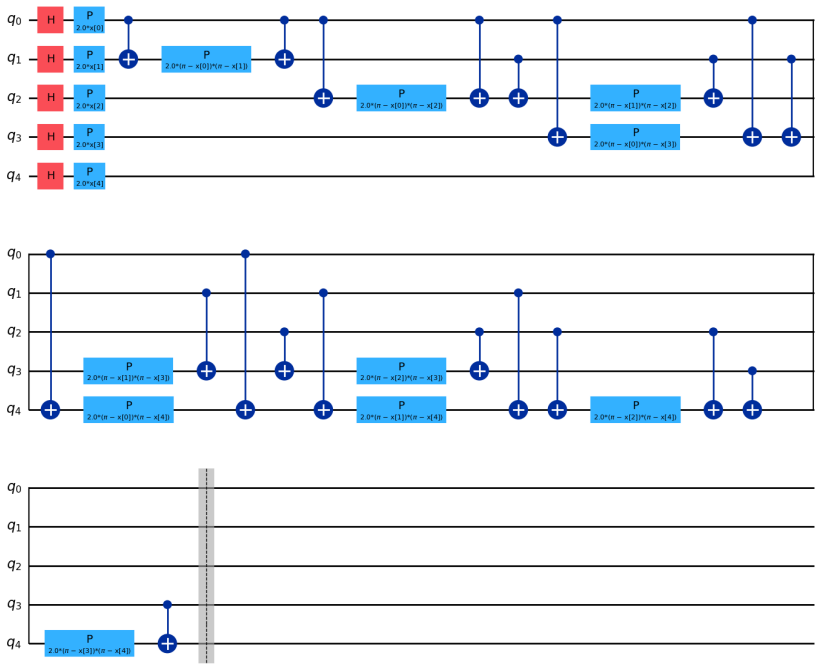

Figura 5.4: PauliFeatureMap con 5 Qubit

5.2 Ansatz

Nell'ambito dell'algoritmo Variational Quantum Classifier, l'ansatz rappresenta un circuito quantistico parametrizzato progettato per generare stati quantistici che possono essere ottimizzati per svolgere un compito specifico, come la classificazione di dati. L'ansatz è una struttura predefinita di porte quantistiche che dipendono da parametri variabili (angoli di rotazione), i quali vengono aggiornati durante l'addestramento dell'algoritmo per minimizzare una funzione di costo. La scelta dell'ansatz è cruciale: deve essere sufficientemente espressivo da catturare la complessità del problema ma non troppo profondo da diventare impraticabile per l'hardware quantistico attuale, spesso rumoroso e limitato. Esistono diverse tipologie di ansatz, come quelli hardware-efficient o quelli ispirati da circuiti teorici, e la selezione dipende dal bilanciamento tra accuratezza e fattibilità sperimentale.

5.2.1 EfficientSU2

Questo circuito è pensato per essere hardware-efficient, ovvero progettato per funzionare bene con le architetture quantistiche attuali, che sono limitate da rumore e numero di qubit.

La sua struttura si basa su sequenze di rotazioni quantistiche parametrizzate che agiscono su ciascun qubit, in genere sotto forma di rotazioni attorno agli assi X, Y o Z. Dopo queste rotazioni, vengono introdotte porte di entanglement, solitamente CNOT, che collegano i qubit secondo uno schema specificato, come l'entanglement lineare, circolare o completo. L'intero schema è organizzato in blocchi ripetuti, detti layer, il cui numero può essere controllato con il parametro `reps`, permettendo al circuito di acquisire maggiore espressività aumentando la profondità. L'ansatz è altamente configurabile e permette la personalizzazione del tipo e dell'ordine delle rotazioni, dello schema di entanglement e della disposizione complessiva del circuito, rendendolo uno strumento versatile per vari algoritmi quantistici variazionali.

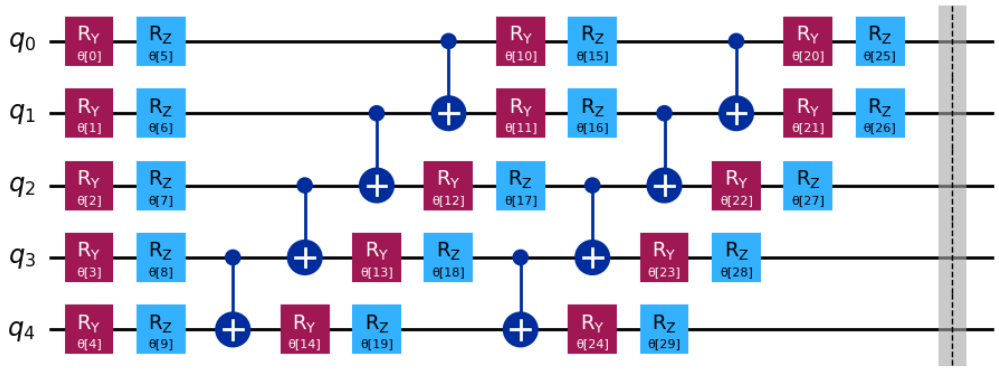

Figura 5.5: EfficientSU2 con 5 Qubit ed entanglement `reverse_linear`

5.2.2 RealAmplitudes

La sua architettura si basa esclusivamente su rotazioni Ry che utilizzano solo parametri reali, da cui deriva il nome. Questa scelta riduce la complessità del circuito e limita lo spazio di parametri da ottimizzare, rendendo l'ansatz più stabile e adatto a ottimizzatori classici.

Dopo le rotazioni sui singoli qubit, RealAmplitudes applica schemi di entanglement tra i qubit, generalmente attraverso porte CNOT. E' possibile specificare lo schema di connessione (lineare, completo, circolare, ecc.), così come il numero di ripetizioni del blocco base formato da rotazioni più entanglement, aumentando la profondità e l'espressività del circuito.

Questo ansatz è apprezzato per la sua semplicità strutturale, l'efficienza computazionale e la buona compatibilità con l'hardware attuale, oltre al fatto che evita l'uso di parametri complessi, spesso difficili da gestire numericamente. La combinazione di rotazioni reali e porte di entanglement controllate fa di RealAmplitudes una scelta solida e interpretabile per numerosi algoritmi quantistici ibridi.

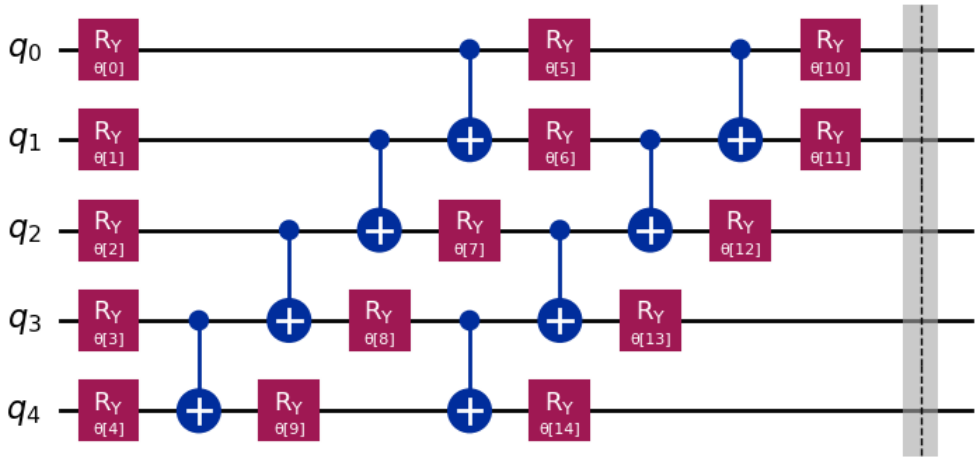

Figura 5.6: RealAmplitudes con 5 Qubit ed entanglement `reverse_linear`

5.3 Ottimizzatore Classico

Poiché i circuiti quantistici non possono, attualmente, aggiornare autonomamente i propri parametri in modo efficiente, è necessario delegare questo compito a un ottimizzatore classico. Questo componente riceve in input i valori correnti dei parametri e la valutazione della funzione costo, calcolata tramite il circuito quantistico, e restituisce una nuova configurazione di parametri che riduce il valore della funzione obiettivo. Il processo continua fino al raggiungimento della convergenza o al superamento di un limite prefissato di iterazioni.

In termini matematici, il suo compito è risolvere il problema di ottimizzazione:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} C(\theta)$$

dove:

- $\theta \in \mathbb{R}^n$ è il *vettore dei parametri variazionali* del circuito quantistico, che controllano le rotazioni e le porte parametrizzate;
- $C(\theta)$ rappresenta la *funzione costo* cioè una funzione scalare che misura la bontà delle previsioni del modello rispetto alle etichette reali del dataset.

- θ^* è il *vettore ottimale dei parametri*, cioè la configurazione che minimizza la funzione costo;
- arg min indica l'*operazione di ricerca del minimo argomentale*, ovvero il valore di θ per cui la funzione $C(\theta)$ assume il valore minimo.

La funzione di costo utilizzata è la *cross-entropy*. Essa misura la distanza tra la distribuzione delle etichette vere e la distribuzione delle probabilità previste dal modello.

L'obiettivo è minimizzare questa distanza, penalizzando le previsioni errate. La funzione assume valori più bassi quando la probabilità assegnata alla classe corretta è alta e cresce rapidamente quando la previsione si discosta dalla verità.

$$\mathcal{L}(\theta) = -\frac{1}{M}\sum_{i=1}^{M}[y_i \log(f(x_i;\theta)) + (1-y_i)\log(1-f(x_i;\theta))]$$

Dove:

- M è il numero totale di esempi nel dataset;
- x_i è il vettore delle feature dell'i-esimo esempio;
- $y_i \in \{0,1\}$ è l'etichetta vera dell'i-esimo esempio;
- $f(x_i;\theta)$ è la probabilità prevista dal VQC che l'esempio x_i appartenga alla classe 1, in funzione dei parametri θ;
- $\mathcal{L}(\theta)$ è la funzione obiettivo (funzione di perdita) da minimizzare.

5.3.1 COBYLA

Il metodo COBYLA (Constrained Optimization BY Linear Approximations), implementato in `scipy.optimize.minimize` come uno dei metodi disponibili, è un algoritmo di ottimizzazione non lineare per problemi vincolati, che non richiede la derivata della funzione obiettivo. È stato sviluppato da M. J. D. Powell. L'idea centrale è quella di risolvere il problema originale attraverso una serie di approssimazioni locali: in ogni passo, sia la funzione

obiettivo sia i vincoli vengono sostituiti da versioni lineari costruite attorno al punto corrente. Queste versioni lineari costituiscono un problema più semplice, che può essere risolto più facilmente. Una volta trovata una direzione di miglioramento sulla base di queste approssimazioni, COBYLA aggiorna la soluzione spostandosi leggermente in quella direzione, mantenendosi all'interno di una regione di fiducia, ovvero una sfera attorno alla soluzione attuale che limita quanto ci si può allontanare. Iterando questo processo e riducendo gradualmente la dimensione della regione di fiducia, il metodo converge verso un punto che soddisfa i vincoli e ottimizza la funzione. L'efficacia del metodo risiede proprio nel suo uso di approssimazioni lineari e nella capacità di operare senza conoscere la forma esatta delle derivate.

5.3.2 COBYQA

L'ottimizzatore COBYQA (Constrained Optimization BY Quadratic Approximations), disponibile nel modulo `scipy.optimize.minize` è un metodo di ottimizzazione senza derivate progettato per risolvere problemi di ottimizzazione non lineare vincolata. Il nome stesso suggerisce l'approccio utilizzato: costruisce e aggiorna approssimazioni quadratiche della funzione obiettivo per guidare la ricerca del minimo.

Questo algoritmo si distingue per il fatto che non richiede il calcolo di derivate, caratteristica che lo rende particolarmente adatto a problemi in cui la funzione obiettivo è complessa, non differenziabile, affetta da rumore o definita esclusivamente in forma numerica. COBYQA lavora costruendo un modello quadratico della funzione obiettivo all'interno di una regione di fiducia (trust region), che viene adattata dinamicamente man mano che l'ottimizzazione procede. A ogni iterazione, viene risolto un sottoproblema di minimizzazione del modello all'interno della regione di fiducia e la soluzione proposta viene accettata o rifiutata a seconda di quanto effettivamente migliora il valore della funzione originale.

Rispetto ad altri algoritmi più sofisticati (come quelli basati su derivate o metodi interior-point), può risultare meno efficiente in problemi di grande

scala, dato che la costruzione e l'aggiornamento del modello quadratico diventa onerosa man mano che aumenta la dimensionalità del problema.

5.3.3 Confronto tra ottimizzatori

Caratteristica	**COBYLA**	**COBYQA**
Nome completo	Constrained Optimization BY Linear Approximations	Constrained Optimization BY Quadratic Approximations
Tipo di metodo	Approssimazioni lineari (SLSQP-like)	Approssimazioni quadratiche (Trust region)
Derivate richieste	No	No
Gestione dei vincoli	Supporta vincoli non lineari tramite approssimazioni lineari	Supporta vincoli non lineari in modo più preciso
Vincoli ai limiti (bounds)	Non supportati nativamente	Supportati direttamente
Efficienza	Più veloce su problemi semplici o di bassa precisione	Più preciso, ma più costoso computazionalmente
Robustezza	Buona, ma può perdere precisione vicino al minimo	Elevata, con maggiore accuratezza nel trovare il minimo
Modello usato	Approssimazione lineare	Approssimazione quadratica
Tipo di problemi adatto	Problemi piccoli o con funzioni rumorose	Problemi moderati con necessità di maggiore precisione
Metodo SciPy	`method="COBYLA"`	`method="COBYQA"`
Supporto in SciPy	Presente da tempo e ben consolidato	Introdotto recentemente, in evoluzione

Tabella 5.1: Confronto tra gli ottimizzatori COBYLA e COBYQA in `scipy.optimize`.

Capitolo 6

Risultati e Conclusioni

6.1 Risultati VQC con Simulatore Ideale

Per analizzare le prestazioni del VQC in un contesto **privo** di rumore quantistico, è stata condotta una serie di test utilizzando dataset con un numero variabile di feature, selezionati tramite la tecnica di ricampionamento *SelectKBest*, fornite in input al modello. L'ottimizzazione è stata eseguita tramite l'algoritmo COBYLA. I circuiti quantistici sono stati eseguiti in simulazione ideale attraverso lo StateVectorSampler, che consente di analizzare l'evoluzione dello stato quantistico in assenza di decoerenza e rumore esterno, offrendo una visione teorica delle potenzialità del modello.

I test hanno coinvolto diverse combinazioni di Feature Map, Ansatz e tipologie di entanglement. I risultati ottenuti sono stati valutati secondo metriche standard di classificazione, oltre al valore minimo della funzione di loss, il tempo impiegato per il training (in minuti) e il numero totale di ripetizioni dell'ottimizzazione.

6.1.1 Risultati VQC con 3 Feature

La tabella che segue mostra una panoramica dei risultati ottenuti da una serie di esperimenti sul Variational Quantum Classifier (VQC) con tre variabili in input.

FeatureMap	Ansatz	Entanglement	Accuracy	Precision	Recall	F1 Score	Best Loss Value	Tempo (m)	Ripetizioni
Z	EfficientSU2	Reverse Linear	0,76	0,77	0,73	0,75	0,505	3,22	116
Z	EfficientSU2	Circular	0,64	0,65	0,54	0,59	0,512	3,11	113
Z	Real Amplitudes	Reverse Linear	0,71	0,79	0,54	0,64	0,569	1,41	107
Z	Real Amplitudes	Circular	0,6	0,6	0,55	0,58	0,671	1,12	84
Pauli	EfficientSU2	Reverse Linear	0,55	0,54	0,48	0,51	0,648	3,3	186
Pauli	EfficientSU2	Circular	0,57	0,56	0,51	0,53	0,625	3,15	181
Pauli	Real Amplitudes	Reverse Linear	0,54	0,54	0,41	0,46	0,676	1,12	80
Pauli	Real Amplitudes	Circular	0,53	0,52	0,53	0,53	0,643	1,3	97

Tabella 6.1: Statistiche principali delle performance per dataset con 3 Feature.

Le migliori performance complessive, in termini di *accuracy* e *F1 score*, si ottengono con la combinazione *ZFeatureMap*, *EfficientSU2* come ansatz e *Reverse Linear* come schema di entanglement, che raggiunge un'accuracy di 0,76 e un F1 score di 0,75, con un tempo di esecuzione moderato e un numero contenuto di ripetizioni. Ciò suggerisce che tale configurazione riesca a sfruttare efficacemente le capacità rappresentative del circuito, mantenendo un equilibrio tra complessità e capacità di generalizzazione.

Al contrario, le configurazioni basate sulla *PauliFeatureMap* mostrano in generale risultati inferiori, con *accuracy* che non supera il 0,57 e valori di *F1 score* generalmente attorno allo 0,5, indicando una minore efficacia, in questo contesto, nel codificare i dati rispetto alla *ZFeatureMap*.

Un altro aspetto rilevante riguarda la scelta dell'ansatz: l'*EfficientSU2* tende a garantire risultati migliori rispetto al *Real Amplitudes*, soprattutto se combinato con *ZFeatureMap*. La topologia di entanglement *Reverse Linear*, inoltre, sembra offrire prestazioni superiori rispetto allo schema *Circular* in quasi tutte le configurazioni, suggerendo una migliore propagazione delle correlazioni tra i qubit.

Per quanto riguarda il valore minimo della *loss*, si osservano variazioni significative tra le configurazioni, ma non sempre un valore di *loss* più basso corrisponde a una migliore performance in termini di metrica di classificazione. Questo sottolinea l'importanza di valutare i modelli oltre che sulla base della funzione obiettivo, ma anche sull'effettiva qualità della classificazione.

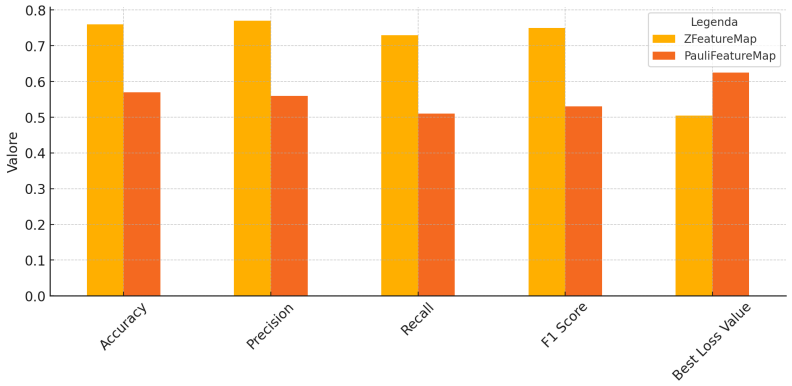

Figura 6.1: Best Feature Map

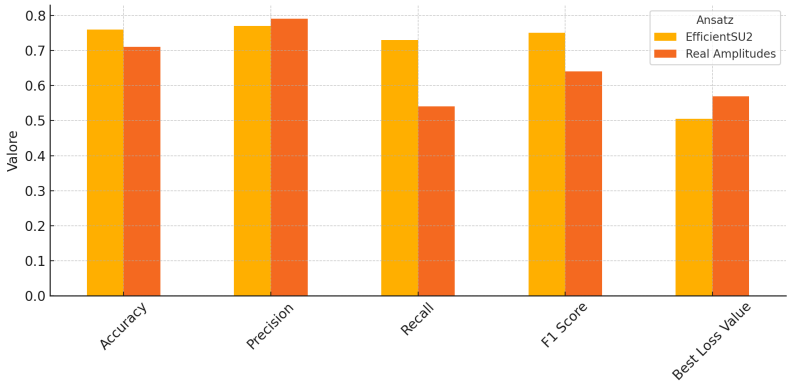

Figura 6.2: Best Ansatz

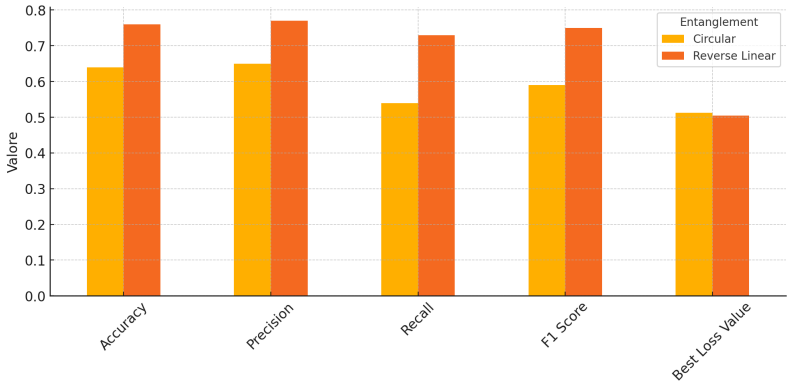

Figura 6.3: Best entanglement

6.1.2 Risultati VQC con 4 Feature

La tabella mostra i risultati ottenuti da diverse combinazioni di modelli quantistici applicati a un dataset con 4 feature. Tutte le configurazioni adottano un'entanglement di tipo *Reverse Linear*,tuttavia, differiscono per il tipo di FeatureMap e di Ansatz adottati.

FeatureMap	Ansatz	Entanglement	Accuracy	Precision	Recall	F1 Score	Best Loss Value	Tempo (m)	Ripetizioni
Z	EfficientSU2	Reverse Linear	0,78	0,75	0,81	0,77	0,508	5,38	314
Z	Real Amplitudes	Reverse Linear	0,65	0,68	0,49	0,55	0,579	2,15	133
ZZ	EfficientSU2	Reverse Linear	0,71	0,70	0,68	0,69	0,578	5,47	291
ZZ	Real Amplitudes	Reverse Linear	0,66	0,66	0,59	0,62	0,60	2,16	123

Tabella 6.2: Statistiche principali delle performance per dataset con 4 Feature.

Tra le combinazioni testate, quella che utilizza ZFeatureMap con l'Ansatz *EfficientSU2* si distingue per le migliori performance complessive: ottiene un'accuratezza del 78%, un *F1 Score* di 0,77 e un *recall* particolarmente elevato (0,81), dimostrando una buona capacità di riconoscere correttamente i casi positivi. Questo modello richiede, tuttavia, anche il maggior numero di ripetizioni (314) e un tempo di esecuzione più lungo (5,38 minuti), riflettendo un compromesso tra accuratezza e costo computazionale.

Le altre combinazioni presentano risultati inferiori, con un consumo di risorse notevolmente ridotto. L'impiego di *ZFeatureMap* con l'Ansatz *Real Amplitudes* comporta una perdita di prestazioni, ma riduce sensibilmente il tempo di calcolo (2,15 minuti) e il numero di ripetizioni (133). L'utilizzo di *ZZFeatureMap* sembra offrire una stabilità maggiore tra le metriche, pur senza raggiungere le performance massime. In particolare, la coppia *ZZ EfficientSU2* mantiene una buona accuratezza (0,71) e un F1 Score di 0,69, posizionandosi come un'alternativa bilanciata tra prestazioni e tempi.

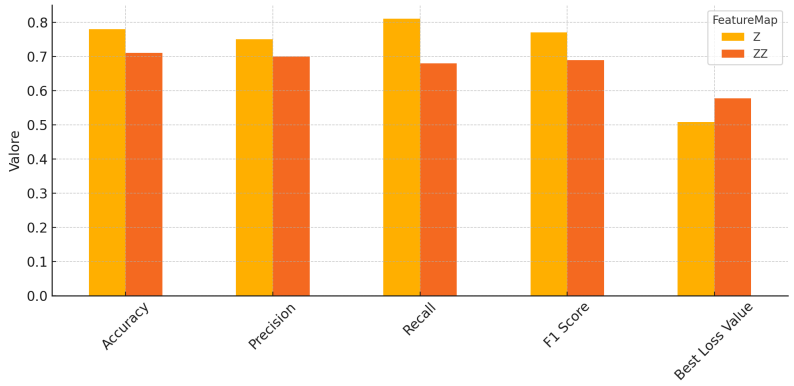

Figura 6.4: Best Feature Map

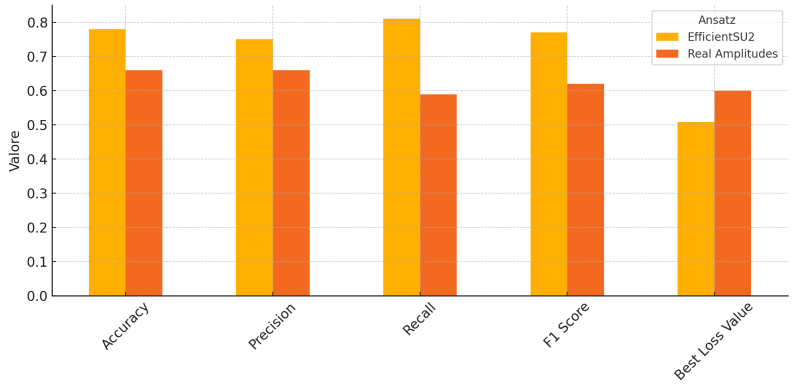

Figura 6.5: Best Ansatz

6.1.3 Risultati VQC con 5 Feature

La tabella che segue riporta i risultati ottenuti da due configurazioni quantistiche con 5 feature, entrambe basate su *ZFeatureMap*, Ansatz *EfficientSU2* ed entanglement *Reverse Linear*, ma con due ottimizzatori differenti: COBYLA e COBYQA

FeatureMap	Ansatz	Entanglement	Optimizer	Accuracy	Precision	Recall	F1 Score	Best Loss Value	Tempo (m)	Ripetizioni
Z	EfficientSU2	Reverse Linear	COBYLA	0,73	0,72	0,72	0,74	0,487	6,29	374
Z	EfficientSU2	Reverse Linear	COBYQA	0,67	0,66	0,57	0,58	0,478	12,52	610

Tabella 6.3: Statistiche principali delle performance con 5 feature.

La configurazione con COBYLA mostra una buona *accuracy* pari a 0,73, con valori equilibrati di *precision* e *recall* entrambi a 0,72 e un *F1 Score* di

0,74, il che indica una buona qualità nella classificazione. Il valore di loss si attesta a 0,487 con un tempo di esecuzione contenuto pari a 6,29 minuti e un numero di ripetizioni di 374. Questa configurazione risulta quindi efficace sia dal punto di vista delle metriche che dell'efficienza.

La configurazione con COBYQA presenta una performance inferiore con un'*accuracy* di 0,67. La *precision* è di 0,66 mentre il *recall* scende a 0,57, compromettendo il valore dell'*F1 Score* che si attesta a 0,58. Nonostante il valore di loss sia leggermente più basso a 0,478, il tempo di esecuzione raddoppia a 12,52 minuti e il numero di ripetizioni sale a 610. Questo indica che l'ottimizzatore COBYQA risulta meno efficiente e meno efficace in questo contesto.

Nel complesso, COBYLA si conferma una scelta più vantaggiosa per questa configurazione, garantendo migliori risultati in termini di performance predittiva e minore complessità computazionale.

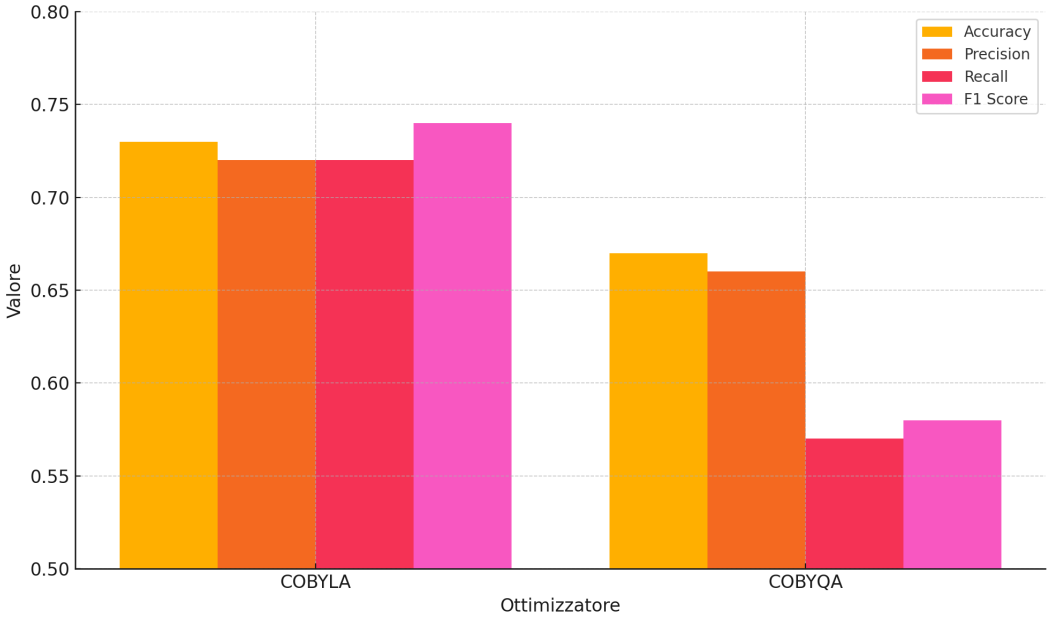

Figura 6.6: Best Optmizer

6.2 VQC utilizzando Simulatore con rumore

La simulazione con rumore nei circuiti quantistici è una tecnica usata per riprodurre con un computer classico il comportamento di un dispositivo quantistico reale, tenendo conto delle sue imperfezioni. In pratica, quando eseguiamo algoritmi su simulatori ideali, non stiamo considerando i problemi concreti che si verificano nei computer quantistici veri, come gli errori di decoerenza, le imprecisioni nelle porte quantistiche o gli errori di misurazione dei qubit. Questi problemi, che nel loro insieme chiamiamo "rumore", rendono i risultati meno affidabili e spesso diversi da quelli teorici.

Simulare con rumore significa introdurre intenzionalmente tali errori nel simulatore al fine di osservare come l'algoritmo si comporterebbe in condizioni reali su un hardware fisico. Serve soprattutto nella fase di sviluppo e testing, perché permette di progettare circuiti che siano più robusti e tolleranti agli errori. In questo modo, quando l'algoritmo sarà eseguito su un vero dispositivo quantistico, le prestazioni saranno più prevedibili.

Per ottenere questo tipo di simulazione, Qiskit offre dei backend chiamati "Fake" che rappresentano dispositivi veri, come FakeVigoV2, copiandone topologia, errori e tempi di decoerenza. Quindi usare un simulatore come FakeVigoV2 non è semplicemente simulare, ma è simulare con rumore, cioè in condizioni realistiche simili a quelle di un vero computer quantistico IBM.

Per valutare il comportamento del classificatore variazionale in un contesto più vicino alla realtà dell'hardware quantistico attuale, è stato introdotto il seguente metodo basato sulla simulazione rumorosa dei circuiti.

```python
def classification_probability_with_noisy(data, variational):
    circuits = [circuit_instance(tupla, variational) for tupla in data]

    fake_backend = FakeVigoV2()
    sim = AerSimulator.from_backend(fake_backend)

    transpiled_qc = transpile(circuits, sim)
    results = sim.run(transpiled_qc).result()

    classifications = []

    for result in results.get_counts():
        classifications.append(label_probability(result))

    return classifications
```

Figura 6.7: Metodo Python per introdurre l'esecuzione su backend con rumore

La funzione `classification_probability_with_noisy` serve a calcolare le probabilità di classificazione per un insieme di dati, utilizzando circuiti quantistici variazionali e simulando il rumore presente in un vero dispositivo quantistico. Rispetto al metodo ideale, viene istanziato un simulatore quantistico realistico utilizzando `FakeVigoV2`, un backend che riproduce le caratteristiche fisiche e gli errori del dispositivo `IBMQ Vigo`. Questo backend viene successivamente processato da `AerSimulator`, che crea una simulazione che include rumore, errori di misura e limiti di connettività tra i qubit.

In seguito, i circuiti vengono convertiti in una forma compatibile con le restrizioni hardware del dispositivo simulato. Questa operazione è fondamentale per garantire che i circuiti possano essere eseguiti correttamente nel contesto fisico simulato. Infine i circuiti vengono eseguiti sul simulatore e si raccolgono i risultati, che consistono nei conteggi delle misurazioni fatte per ciascun circuito.

Come ultima operazione, per ogni risultato, si calcola la probabilità di classificazione usando una funzione chiamata `label_probability`, che interpreta i conteggi di misura per determinare a quale classe il risultato appartiene. Tutte queste probabilità vengono raccolte in una lista chiamata classifications, che viene restituita come output della funzione.

6.2.1 Risultati

La seguente tabella riassume le performance del VQC applicato a dataset con 3, 4 e 5 feature, mantenendo costante la struttura del circuito quantistico.

Features	FeatureMap	Ansatz	Entanglement	Accuracy	Precision	Recall	F1 Score	Best Loss Value	Tempo (m)	Ripetizioni
3	Z	EfficientSU2	Reverse Linear	0,77	0,8	0,69	0,74	0,536	6,19	187
4	Z	EfficientSU2	Reverse Linear	0,82	0,75	0,93	0,83	0,556	8,44	289
5	Z	EfficientSU2	Reverse Linear	0,78	0,74	0,79	0,77	0,566	16,26	388

Tabella 6.4: Statistiche principali delle performance.

Aumentando il numero di feature da 3 a 4 si osserva un miglioramento generale delle prestazioni il che suggerisce che l'aggiunta di una feature rende il modello più espressivo e capace di catturare meglio la struttura dei dati. L'*accuracy* passa da 0,77 a 0,82 mentre la *recall* raggiunge il valore più alto a 0,93 con una buona *F1 Score* di 0,83 che rappresenta il miglior equilibrio.

Con 5 feature le prestazioni diminuiscono leggermente. L'*accuracy* scende a 0,78 e la *F1 Score* si stabilizza a 0,77 segno che l'aumento ulteriore della complessità del dataset non comporta un miglioramento netto. Il modello riesce comunque a mantenere prestazioni stabili ma con un costo computazionale significativamente più elevato. I tempi di esecuzione, infatti, aumentano da 6,19 minuti a 16,26 e le ripetizioni passano da 187 a 388 iterazioni.

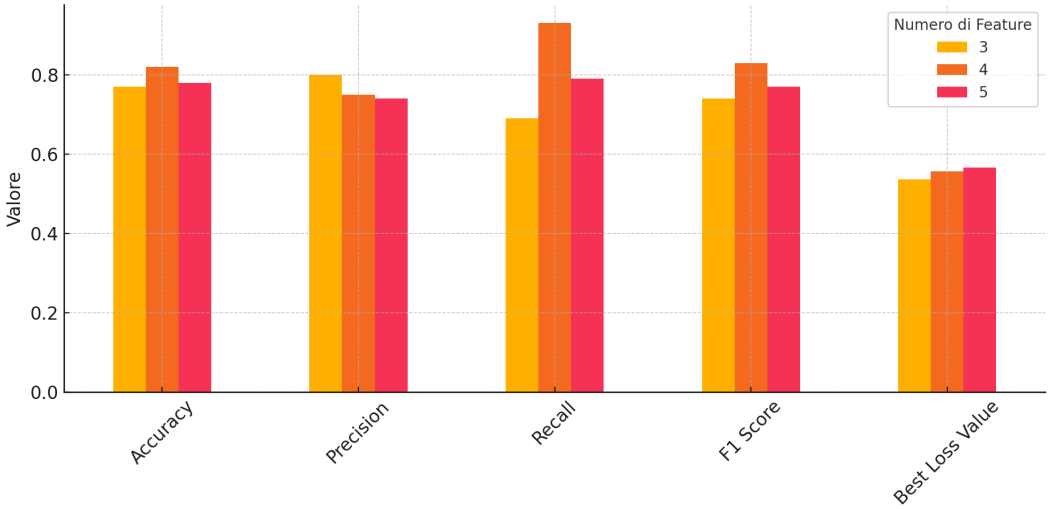

Figura 6.8: Best N features