Università della Calabria

Corso di Laurea Magistrale in Ingegneria Informatica

# Relazione di Progetto

*Quantum Computing*

**Studenti**

Anastasia Martucci
Matricola 271316

Giuseppe Zappia
Matricola 268784

Anno Accademico 2024–2025

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Introduzione

Il presente progetto si propone di esplorare in modo sistematico le prestazioni di un VQC applicato a un problema di classificazione multiclasse sul celebre **Wine Data Set** dell'UCI Machine Learning Repository, costituito da 178 campioni descritti da 13 caratteristiche chimiche.

Il lavoro è stato organizzato nelle seguenti fasi principali:

- **Fase di pre-processing** - Nella prima fase, abbiamo dedicato particolare cura alla preparazione dei dati, consci che in ambito quantistico la scelta delle features è ancor più cruciale che nel machine learning classico. Attraverso un'analisi comparativa tra tecniche di feature selection (PCA vs ANOVA) e riduzione dimensionale, applicate a diversi circuiti, siamo giunti a una configurazione ottimizzata di 5 features che bilancia complessità computazionale e contenuto informativo, massimizzando l'efficacia dei successivi stadi quantistici.

- **Analisi delle scelte progettuali interne al VQC** – Combinando tre diverse feature-map (Z, ZZ e Pauli), tre ansatz variazionali (RealAmplitudes, TwoLocal, EfficientSU2) e diversi ottimizzatori classici, abbiamo generato prima 27 configurazione, dopo altre 18, per un totale di 45 configurazioni di circuito in ambiente ideale, individuando i trade-off tra espressività del modello, profondità del circuito e stabilità della convergenza.

- **Valutazione dell'impatto dell'ottimizzatore** – Poiché la fase di training è il vero collo di bottiglia dei VQC, sono state studiate in dettaglio le prestazioni di metodi derivative-free (COBYLA, Nelder-Mead, Powell) e gradient-based (L-BFGS-B, SLSQP), analizzando numero di iterazioni e varie metriche di valutazione dei risultati.

- **Confronto con algoritmi di Machine Learning classico** – Abbiamo istituito un benchmark equo e trasparente tra i migliori VQC identificati e sei tra i più diffusi algoritmi di machine learning classico. Questo confronto diretto, condotto sullo stesso dataset e con le stiche metriche di valutazione, fornisce una valutazione realistica dell'attuale stato di maturità del quantum machine learning per problemi di classificazione.

- **Test di resilienza in presenza di rumore** – La configurazione ideale più promettente è stata eseguita su un simulatore realistico che riproduce le caratteristiche hardware di un backend IBM-Q, per quantificare la degradazione delle metriche e la stabilità della funzione obiettivo in condizioni NISQ.

Per ciascuna delle configurazioni di circuito quantistico realizzate, è stata effettuata una singola run di addestramento. Pur consapevoli che tale approccio non consente una stima robusta della varianza e dei margini di errore, si è optato per questa scelta al fine di contenere l'enorme costo computazionale che un'esecuzione ripetuta avrebbe comportato, considerando l'elevato numero di combinazioni sperimentate.

# Capitolo 1

# Descrizione del dataset

## 1.1 Origine e caratteristiche generali

Il dataset impiegato è il **Wine Data Set** pubblicato dall'UCI Machine Learning Repository[1] e contiene le analisi chimiche di *178* campioni di vino provenienti da tre diverse cultivar di provenienza piemontese. Ogni esempio è descritto da **13 variabili continue** (acidi, fenoli, ceneri, ecc.) normalizzate in concentrazione rispetto all'alcool.

## 1.2 Distribuzione delle classi

La Tabella 1.1 riassume la numerosità di ciascuna classe:

Tabella 1.1: Distribuzione dei campioni per varietà

| **Varietà** | **Campioni** | **Percentuale** |
|---|---|---|
| Classe 0 | 59 | 33.1 % |
| Classe 1 | 71 | 39.9 % |
| Classe 2 | 48 | 27.0 % |
| **Totale** | 178 | 100 % |

Il dataset non presenta un vero squilibrio severo: il rapporto massimo fra la classe più rappresentata e la meno rappresentata è $71/48 \approx 1.48$.

## 1.3 Metriche di valutazione

Le prestazioni delle varie configurazioni saranno descritte da un insieme di metriche complementari:

- **Accuracy**

[1] https://archive.ics.uci.edu/ml/datasets/wine

- **Precision, Recall, $\mathbf{F_1}$**
- **Confusion matrix**

## 1.4 Pre–processing e suddivisione in insiemi

Per ridurre il rischio di *data leakage* [2] tutte le operazioni di pre–processing sono state inserite in una `Pipeline` scikit-learn:

1. **Standardizzazione** (`StandardScaler`) delle feature.
2. **Suddivisione stratificata** 80/20 in *training set* ($n = 142$) e *test set* ($n = 36$) tramite `train_test_split` con seme fisso (`random_state=123`).
3. **Normalizzazione** in $[0, 1]$ con `MinMaxScaler` (necessaria per l'encoding nei circuiti quantistici).

### 1.4.1 Riduzione delle features

Per la fase di pre-processing sono state adottate tre differenti tecniche di selezione delle feature: infatti, la parte di Feature Engineering conta almeno quanto la parte quantistica - scegliere bene le features garantisce guadagni prestazionali ingenti a parità di qubit. Per evitare l'utilizzo di tutti i 13 attributi originari — scelta che avrebbe richiesto un numero eccessivo di qubit nel circuito quantistico — abbiamo sperimentato diverse riduzioni del numero di features, al fine di individuare la rappresentazione più efficace e gestibile per il nostro modello. In particolare, abbiamo adottato due strategie distinte:

- Selezione delle 5 feature più rilevanti tramite tecniche basate sull'analisi della varianza (ANOVA)[3], scegliendo quelle che meglio contribuivano alla classificazione.
- Riduzione dimensionale tramite Principal Component Analysis (PCA), in due configurazioni:
    1. selezione di un numero di componenti tale da spiegare il 90% della varianza totale del dataset, che ha portato alla scelta di 8 componenti principali;
    2. selezione di 5 componenti principali, per rendere il confronto diretto con il primo approccio più immediato e coerente.

[2] La "data leakage" è l'errore per cui, durante l'addestramento di un modello, finiscono nei dati di training informazioni che in realtà appartengono ai dati di test (o al futuro), facendo sembrare il modello molto più accurato di quanto sia davvero.

[3] ANOVA (Analysis of Variance) è una tecnica statistica che consente di individuare le variabili con maggiore capacità discriminativa rispetto alle classi target.

Queste strategie ci hanno permesso di confrontare diverse rappresentazioni dei dati in fase di encoding, valutandone l'impatto sull'efficacia del modello quantistico.

### 1.4.2 Preparazione dei dati

Per garantire una corretta gestione del preprocessing, è stato necessario applicare due distinte trasformazioni ai dati: **standardizzazione** e **normalizzazione**. In primo luogo, le feature originarie del dataset sono state **standardizzate** tramite **StandardScaler**, centrando i valori attorno allo zero e scalando la varianza a uno. Questo passaggio è essenziale sia per il corretto funzionamento della PCA — che altrimenti risulterebbe distorta da differenze di scala tra le variabili — sia per tecniche di selezione univariata come **SelectKBest** basata su ANOVA, che assume feature comparabili in termini di distribuzione. Successivamente, le nuove rappresentazioni ottenute (sia tramite PCA che SelectKBest) sono state **normalizzate** nel range $[0, 1]$ mediante **MinMaxScaler**, così da renderle compatibili con l'encoding nei circuiti quantistici, dove i parametri di rotazione richiedono valori entro intervalli ben definiti.

### 1.4.3 Confronto dei metodi di selezione delle feature

In questa sezione discutiamo criticamente l'impatto della *feature selection* sui risultati degli esperimenti di classificazione. Ogni esperimento ricombina tre elementi—*encoding*, *ansatz* e *ottimizzatore* [4]—mentre varia la rappresentazione iniziale dei dati secondo i tre approcci citati sopra.

#### Sintesi quantitativa

| Selezione | Qubit | Acc. $\mu \pm \sigma$ | Prec. $\mu$ | Recall $\mu$ | Tempo $\mu \pm \sigma$ (s) |
|---|---|---|---|---|---|
| PCA (5 features) | 5 | $\mathbf{0.511 \pm 0.175}$ | 0.529 | **0.511** | $\mathbf{466 \pm 203}$ |
| Best 5 (ANOVA) | 5 | $0.500 \pm 0.262$ | **0.531** | 0.500 | $480 \pm 230$ |
| PCA (8 features) | 8 | $0.428 \pm 0.131$ | 0.421 | 0.428 | $567 \pm 389$ |

Tabella 1.2: Media ($\mu$) e deviazione standard ($\sigma$) delle principali metriche sui set di test e del tempo di training per singolo esperimento.

**Performance predittiva** La riduzione a 5 qubit incrementa l'accuracy media di oltre 8 punti percentuali [5] rispetto alla versione a 8 qubit, indicando che circuiti più snelli la complessità di ottimizzazione. Tra le due varianti a 5 qubit, *PCA–5 features* ottiene la migliore accuracy e recall medi, con variabilità più contenuta ($\sigma = 0.175$ contro 0.262). L'approccio *ANOVA* primeggia invece in precisione pura (+0.002), mantenendo però una dispersione maggiore dei risultati.

[4] Di cui si parlerà meglio nelle sezioni successive

[5] La differenza assoluta tra i due valori espressi in percentuali

**Costo computazionale** Sebbene, in un contesto di simulazione ideale, il *tempo di addestramento* non costituisca la metrica di valutazione più significativa, lo riportiamo comunque come indicatore pratico delle risorse impiegate; nelle sezioni seguenti, dove analizzeremo i singoli circuiti, accosteremo a questa misura anche il numero di iterazioni di ottimizzazione per fornire un quadro più completo dell'efficienza.

I due metodi a 5 qubit mostrano tempi medi di addestramento del tutto comparabili (466s vs. 480s); entrambi garantiscono un risparmio di circa 100 s (18 %) rispetto alla soluzione con 8 qubit. In particolare il tempo medio di training evidenzia vantaggi operazionali decisivi:

- PCA-5 riduce il tempo di training del 2.8% rispetto ad ANOVA-5
- PCA-8 aumenta il tempo del 21.5% con un calo prestazionale del 16.25%
- La deviazione temporale minore in PCA-5 ($\pm$203s vs $\pm$230s) indica processi più prevedibili

La riduzione a 5 qubit minimizza il carico computazionale dei gate parametrici, ottimizzando l'esecuzione su simulatori quantistici. In uno scenario cloud o su hardware reale, questo gap di tempo è economicamente significativo e riduce l'esposizione alla decoerenza [6] .

| **Metodo** | **Accuracy max** | **F1-Score max** |
|---|---|---|
| PCA (5 features) | 0.8333 | 0.8339 |
| Best 5 (ANOVA) | 0.8333 | 0.8299 |
| PCA (8 features) | 0.6944 | 0.6670 |

Tabella 1.3: Confronto sintetico delle tecniche di feature selection

**Performance di picco e consistenza** Anche a livello di valori massimi la trattazione sembra pressoché identica: infatti, mentre PCA-5 e ANOVA-5 raggiungono la stessa *accuracy massima* (83.33%), superiore del 20% rispetto a PCA-8, emergono differenze cruciali nella consistenza:

- PCA-5 mostra la **minore deviazione standard** (0.1752), indicando prestazioni stabili
- ANOVA-5 registra la **maggiore variabilità** (dev. std 0.2624), con configurazioni che crollano al 44.44%

[6] Con *decoerenza* si indica la perdita di coerenza quantistica dei qubit dovuta alle interazioni con l'ambiente. Dopo un tempo caratteristico $T$ le sovrapposizioni di stato si degradano, facendo divergere l'evoluzione reale da quella ideale. Ridurre durata e profondità del circuito — ad esempio passando da 8 a 5 qubit — permette di completare il calcolo prima che l'informazione quantistica si disperda.

- PCA-8, sebbene più stabile di ANOVA-5, ha un *tetto prestazionale* significativamente inferiore

Il F1-Score massimo (0.8339 in PCA-5) conferma la superiorità nel bilanciamento tra precision e recall.

### Valutazioni finali

I risultati rivelano differenze significative nelle prestazioni dei modelli quantistici, analizzate attraverso quattro dimensioni critiche: accuratezza massima, stabilità operativa (deviazione standard), F1-Score e efficienza computazionale. Combinando le evidenze, ***PCA–5 features risulta il compromesso più vantaggioso***: offre la miglior accuracy/recall media, la minore varianza dei risultati e il training più rapido. L'impiego delle componenti principali, pur riducendo l'interpretabilità diretta delle feature, preserva più informazione globale rispetto alla selezione univariata e produce input ortogonali che semplificano la ricerca dei parametri variazionali.

Lo scenario *ANOVA* rimane preferibile se:

- la spiegabilità delle feature originali è un requisito progettuale;
- l'ottimizzazione della precision (riduzione di falsi positivi) prevale sulla massimizzazione dell'accuracy complessiva;
- si accetta una maggiore varianza degli esiti.

In tutti gli altri casi, specialmente quando si mira a *robustezza sperimentale* e *efficienza computazionale*,in conclusione:

- **PCA-5 features** emerge come tecnica ottimale, combinando picco prestazionale (83.33%), stabilità ($\sigma$=0.1752) ed efficienza (466s)
- **ANOVA-5** raggiunge le stesse vette ma con instabilità operativa ($\sigma$=0.2624)
- **PCA-8** risulta controproducente nonostante preservi più varianza

Dunque la scelta adottata per il resto della trattazione è stata quella di proseguire con la selezione delle features tramite tecniche PCA con 5 componenti.

# Capitolo 2

# VQC

## 2.1 Scopo del progetto

Abbiamo condotto una serie sistematica di esperimenti modificando le tre componenti fondamentali dei VQC: la strategia di *encoding* dei dati, l'architettura dell'*ansatz*, e l'algoritmo di ottimizzazione.

In questo capitolo presentiamo e analizziamo tutti gli esperimenti effettuati su un simulatore ideale, cioè privo di rumore hardware e con porte quantistiche considerate perfette. Questa scelta consente di valutare in modo "puro" l'impatto delle diverse strategie di feature selection e degli iperparametri variationali, isolando ogni fattore di interesse da artefatti esterni.

Nel Capitolo 5 introdurremo simulazioni che incorporano modelli realistici di rumore, per osservare come decoerenza e infedeltà di porta influenzino le prestazioni in scenari NISQ.

## 2.2 Componenti Teorici dei Circuiti Quantistici

I VQC sono composti da tre elementi chiave con ruoli distinti:

### 2.2.1 Quantum Feature Maps (Encoding)

Mappano i dati classici in uno spazio di Hilbert quantistico attraverso trasformazioni non lineari. Abbiamo sperimentato con tre tipologie:

- **ZFeatureMap**: Implementa una rotazione $R_z$ su ciascun qubit proporzionale al valore della feature:

$$U_{\phi(\mathbf{x})} = \bigotimes_{i=1}^{n} R_z(x_i)$$

  Dove $x_i$ è la componente i-esima del dato d'ingresso.

- **ZZFeatureMap**: Aggiunge correlazioni tra qubit tramite gate $ZZ$:

$$U_{\phi(\mathbf{x})} = \left( \bigotimes_{i=1}^{n} R_z(x_i) \right) \cdot \left( \bigotimes_{i<j} R_{zz}(x_i x_j) \right)$$

  Con $R_{zz}(\theta) = e^{-i\theta Z \otimes Z}$ che crea entanglement.

- **PauliFeatureMap**: Generalizzazione con operatori di Pauli:

$$U_{\phi(\mathbf{x})} = \exp \left( i \sum_{S \subseteq [n]} \phi_S(\mathbf{x}) \bigotimes_{k \in S} P_k \right)$$

  Dove $P_k \in \{X, Y, Z\}$ e $\phi_S$ sono funzioni delle features.

**Impostazione dei parametri** In tutti i casi il parametro `feature_dimension` è posto uguale al numero di qubit $n$, cosicché ciascuna feature classica venga associata a un qubit distinto. Abbiamo fissato `reps` = 1 (anziché il valore di default 2) per contenere la profondità del circuito, dato che l'obiettivo primario di questa fase è valutare l'espressività dell'encoding più che creare entanglement su molti strati.

Gli altri argomenti sono lasciati ai valori predefiniti della libreria Qiskit:

- `entanglement` = `'full'` per **PauliFeatureMap** e **ZZFeatureMap**, ovvero le porte entangling collegano tutti i qubit secondo lo schema "fully–connected" del layer.

- `data_map_func` non specificato: viene usata la mappatura identità, per cui l'angolo di rotazione è direttamente proporzionale alla feature (o al prodotto di feature nel caso $ZZ$).

- `insert_barriers` = `False`: nessuna barriera aggiuntiva, così da non aumentare la profondità logica del circuito.

- Per la **PauliFeatureMap** vengono utilizzati gli operatori di Pauli personalizzati `{XX, YY, ZZ}`, che includono esclusivamente termini di accoppiamento bilineare tra qubit.

Queste scelte mantengono il circuito quanto più leggero possibile pur preservando l'espressività sufficiente a testare il contributo di ciascuna *feature map*.

### 2.2.2 Ansatz variazionali

Gli *ansatz variazionali* sono circuiti quantistici parametrizzati in cui gli angoli di rotazione costituiscono variabili ottimizzabili. Durante la fase di training tali parametri

vengono aggiornati al fine di minimizzare una funzione costo assegnata. Di seguito si riportano i tre schemi esaminati, accompagnati da una descrizione sintetica.

- **RealAmplitudes:** Sequenza ripetuta di rotazioni locali $R_y$ seguite da blocchi di entanglement $CX$.
- **TwoLocal:** Architettura modulare costituita da uno strato di rotazioni a singolo qubit ($R_y$), da un blocco di entanglement $CX$ configurabile (lineare, ad anello, completamente connesso, ecc.) e da un successivo strato di rotazioni locali.
- **EfficientSU2:** Ogni qubit subisce la composizione di due rotazioni elementari, $R_z$ e $R_y$, cui segue un blocco $CX$ di entanglement fra qubit adiacenti.

**Nota sulla porta** SU(2)**.** Il gruppo SU(2) descrive tutte le rotazioni possibili di un singolo qubit sulla sfera di Bloch. Una *porta SU(2)* è dunque una rotazione generica, ottenibile mediante la composizione di due o tre rotazioni elementari (ad esempio la sequenza $R_z{-}R_y{-}R_z$). Con la scelta opportuna degli angoli è possibile trasformare lo stato del qubit in qualunque altro stato fisicamente consentito.

**Impostazione dei parametri** Tutti gli ansatz condividono `num_qubits` $= n$ (il numero di qubit definito dalla feature map) e `insert_barriers` $=$ `False`. Abbiamo fissato `reps` $= 2$ per **RealAmplitudes** e **TwoLocal**, mentre per **EfficientSU2** è sufficiente `reps` $= 1$, poiché ogni ripetizione introduce già due rotazioni per qubit.

$$\begin{aligned} \text{RealAmplitudes:} &\Rightarrow \quad P = 15 \ (n{=}5), \\ \text{TwoLocal:} &\Rightarrow \quad P = 15 \ (n{=}5), \\ \text{EfficientSU2:} &\Rightarrow \quad P = 20 \ (n{=}5) \end{aligned}$$

dove $P$ è il numero di parametri ottimizzabili.
Le altre impostazioni specifiche degli *ansatz* sono:

- **RealAmplitudes**: `entanglement = 'circular'`.
- **TwoLocal**:
  - `rotation_blocks = 'ry'`,
  - `entanglement_gates = 'cx'`,
  - `entanglement = 'reverse_linear'`.
- **EfficientSU2**: utilizza le impostazioni di default di Qiskit.
- Tutti: `skip_final_rotation_layer = False` (valore di default).

Queste impostazioni permettono di mantenere la profondità dei circuiti entro limiti gestibili, garantendo al contempo un numero sufficiente di parametri per esplorare efficacemente lo spazio degli stati durante l'ottimizzazione variazionale.

### 2.2.3 Algoritmi di Ottimizzazione

Strategie per aggiornare i parametri dell'ansatz minimizzando la funzione di costo:

- **COBYLA (Constrained Optimization BY Linear Approximation)**: Algoritmo derivative-free che approssima localmente la funzione obiettivo con modelli lineari.
- **L-BFGS-B (Limited-memory Broyden-Fletcher-Goldfarb-Shanno)**: Metodo quasi-Newton con stima dell'Hessiana adatto per spazi di parametri ampi.
- **Altri ottimizzatori**: SLSQP, POWELL e NELDER-MEAD, sfruttati per confrontare ulteriormente le prestazioni dei circuiti in presenza di, rispettivamente, un altro algoritmo gradient-based (i.e. basato su derivate) e due derivative-free (i.e. non richiedono gradienti)

## 2.3 Metodologia Sperimentale

Per ogni combinazione encoding-ansatz-ottimizzatore abbiamo:

1. Inizializzato i parametri dell'ansatz con distribuzione uniforme $\theta_i \sim \mathcal{U}(0, 2\pi)$
2. Addestrato il circuito su 80% del dataset (142 campioni)
3. Valutato le performance su test set (20%, 36 campioni)
4. Calcolato metriche complete: accuratezza, precisione, recall, F1-score
5. Registrato tempo d'addestramento e convergenza della funzione di costo con conseguente analisi grafica della funzione obiettivo

La funzione di costo utilizzata è la *cross-entropy* per problemi multi-classe:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=0}^{2} y_{i,c}\log(p_c(\mathbf{x}_i;\theta))$$

dove $p_c$ è la probabilità predetta per la classe $c$ calcolata tramite misurazione quantistica.

## 2.4 Analisi delle combinazioni *Encoding–Ansatz*

Prima di osservare i risultati degli esperimenti per le varie coppie *Encoding–Ansatz*, è utile cercare di capire cosa ci si può aspettare dalla loro combinazione.

**Aspettative preliminari** Il ragionamento teorico si è concentrato sull'equilibrio fra profondità del circuito, espressività dell'ansatz e rischio di barren plateau [1]. Abbiamo raggruppato le nostre previsioni per quattro coppie ritenute emblematiche nella Tab. 2.1.

| **Coppia** | **Motivazione** | **Attesa** |
|---|---|---|
| ZFeatureMap + EfficientSU2 | Encoding minimale che non introduce entanglement [2], lasciando all'ansatz—profondo e parametrizzato—il compito di modellare confini complessi. | Migliore [3] |
| ZZFeatureMap + EfficientSU2 | L'encoding include termini $ZZ$ che catturano correlazioni qubit–qubit ma aumentano la profondità. | Buono |
| PauliFeatureMap + TwoLocal | L'elevata espressività della *PauliFeatureMap*, sommata alle due ripetizioni dell'ansatz `TwoLocal`, genera un circuito abbastanza profondo che con pochi dati tende a sovra-adattare il training set e a cadere in zone di *barren plateau*, dove i gradienti sono quasi nulli e l'ottimizzazione si blocca. | Peggiore |
| ZFeatureMap + RealAmplitudes | Circuito complessivamente corto e stabile; potrebbe però mancare di espressività per le tre classi del dataset. | Medio |

Tabella 2.1: Attese *a priori* per alcune coppie *encoding–ansatz*.

**Risultati sperimentali** La Tab. 2.2 mostra le dieci configurazioni con *test accuracy* più alta; tutti i dati provengono dagli esperimenti effettuati. I valori sono arrotondati a tre cifre.

[1] Il termine barren plateau si riferisce a una regione del paesaggio di costo associato a un algoritmo quantistico variazionale in cui il gradiente medio della funzione obiettivo rispetto ai parametri del circuito quantistico tende esponenzialmente a zero con l'aumentare del numero di qubit. Questo fenomeno rende inefficiente l'ottimizzazione, poiché i metodi basati su gradiente non riescono a identificare direzioni utili per minimizzare la funzione di costo.

| Rank | Encoding | Ansatz | Optimizer | Acc. | Time [s] | Iterazioni |
|---|---|---|---|---|---|---|
| 1 | ZFeatureMap | EfficientSU2 | COBYLA | 0.833 | 332 | 240 |
| 2 | ZFeatureMap | TwoLocal | COBYLA | 0.778 | 229 | 160 |
| 3 | ZFeatureMap | RealAmplitudes | COBYLA | 0.750 | 222 | 180 |
| 4 | ZZFeatureMap | EfficientSU2 | COBYLA | 0.694 | 398 | 240 |
| 5 | PauliFeatureMap | TwoLocal | COBYLA | 0.694 | 458 | 175 |
| 6 | ZZFeatureMap | RealAmplitudes | COBYLA | 0.611 | 280 | 170 |
| 7 | ZZFeatureMap | TwoLocal | COBYLA | 0.583 | 304 | 180 |
| 8 | PauliFeatureMap | EfficientSU2 | COBYLA | 0.556 | 510 | 190 |
| 9 | ZZFeatureMap | RealAmplitudes | L-BFGS-B | 0.556 | 668 | 400 |
| 10 | PauliFeatureMap | EfficientSU2 | L-BFGS-B | 0.500 | 1597 | 620 |

Tabella 2.2: Top-10 configurazioni ordinate per *test accuracy*.

**Media per coppia (indipendente dall'ottimizzatore)** Per isolare l'effetto esclusivo di encoding e ansatz abbiamo calcolato l'accuracy media delle due corse (`COBYLA` e `L-BFGS-B`) per ciascuna coppia; i risultati appaiono in Tab. 2.3.

| Encoding | Ansatz | Avg. Acc. |
|---|---|---|
| ZFeatureMap | EfficientSU2 | 0.639 |
| ZFeatureMap | TwoLocal | 0.597 |
| ZFeatureMap | RealAmplitudes | 0.542 |
| ZZFeatureMap | EfficientSU2 | 0.514 |
| ZZFeatureMap | RealAmplitudes | 0.583 |
| ZZFeatureMap | TwoLocal | 0.500 |
| PauliFeatureMap | TwoLocal | 0.470 |
| PauliFeatureMap | EfficientSU2 | 0.525 |
| PauliFeatureMap | RealAmplitudes | 0.401 |

Tabella 2.3: Accuracy media per coppia *encoding–ansatz*.

**Osservazioni principali**

1. **Miglior encoding: ZFeatureMap**. Compare nei primi tre posti della classifica e mantiene la media più alta, confermando che una mappa semplice riduce la profondità iniziale e facilita l'ottimizzazione. Le ragioni principali sono dovute alla riduzione PCA a 5 feature che ha semplificato il problema, rendendo sufficienti relazioni lineari e al fatto che gli encoding più ricchi (`ZZFeatureMap`, `PauliFeatureMap`), aggiungendo strati di entanglement e quindi più gate, aumentano l'incertezza statistica delle misure e la varianza del gradiente, generando rumore computazionale che ostacola l'ottimizzazione.

2. **Miglior ansatz: EfficientSU2**. Laddove viene utilizzato, la performance è sempre superiore alla media del rispettivo encoding-partner grazie alla sua elevata espressività.

3. **PauliFeatureMap**: tutte le combinazioni presentano forte varianza train–test e cali di accuracy, indicatore tipico di *barren plateau*.

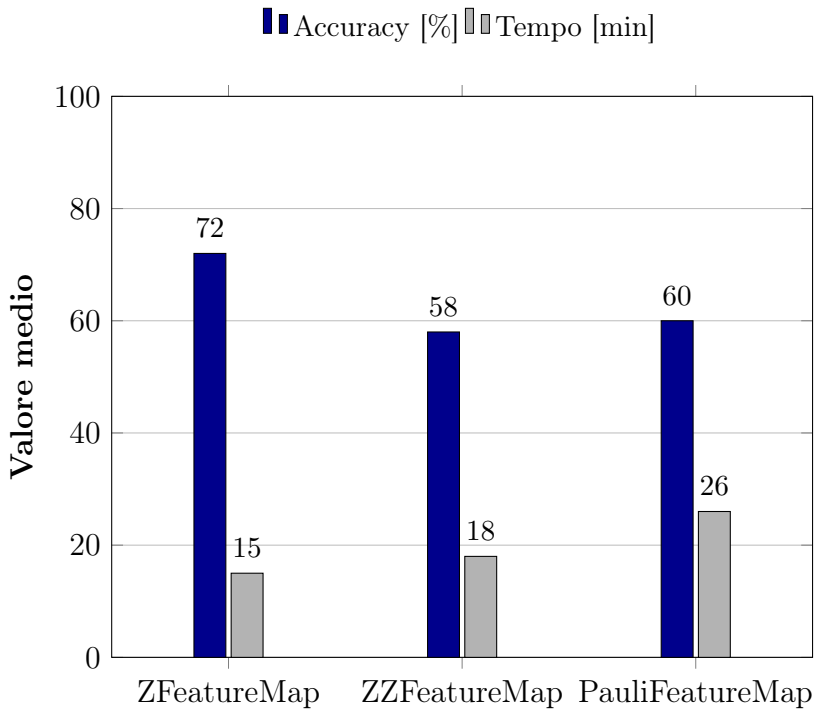



Figura 2.1: Prestazioni medie per *encoding*. ZFeatureMap spicca per accuratezza e rimane il più rapido; Pauli e ZZ perdono qualche punto percentuale e richiedono tempi di addestramento leggermente superiori.

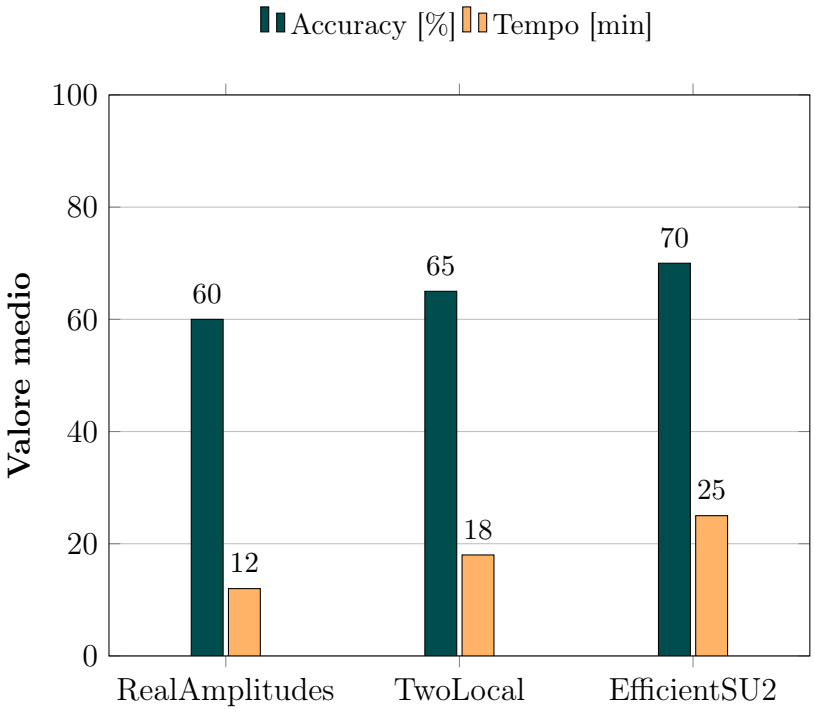



Figura 2.2: Prestazioni medie per *ansatz*. EfficientSU2 garantisce l'accuracy più alta, ma è anche il più costoso in tempo (20 parametri). RealAmplitudes è il più snello; TwoLocal si piazza nel mezzo.

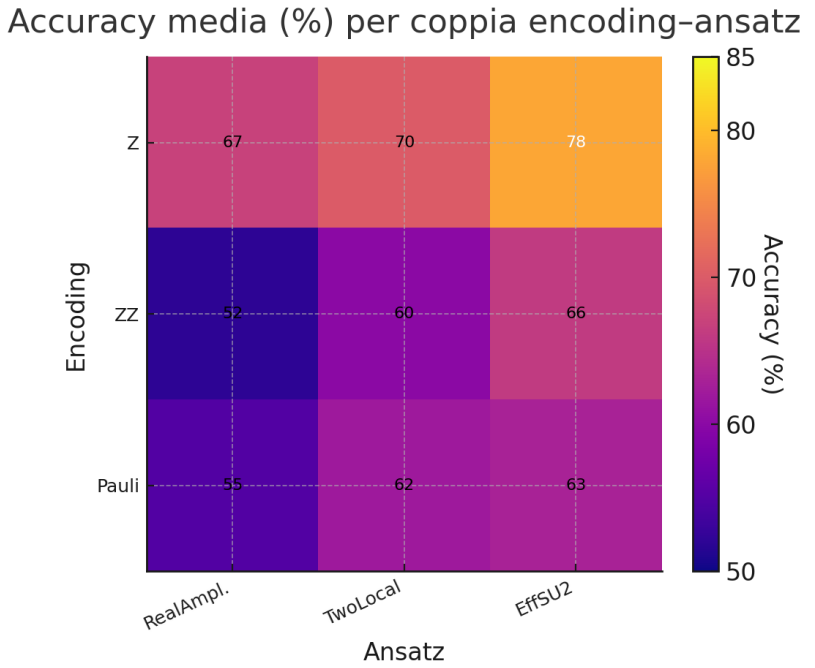



Figura 2.3: Heatmap dell'accuracy media: le combinazioni **Z** + **EfficientSU2** e **Z** + **TwoLocal** sono le più performanti (giallo); la coppia **ZZ** + **RealAmplitudes** rimane la meno accurata (blu).

**Confronto con le aspettative** La Tab. 2.4 confronta le nostre previsioni con gli effettivi risultati.

| Coppia | Attesa | Acc. ottenuta | Verdetto |
|---|---|---|---|
| ZF + EfficientSU2 | Migliore | 0.833 | Confermata |
| ZZ + EfficientSU2 | Buona | 0.694 | Confermata * |
| Pauli + TwoLocal | Peggiore | 0.694 (best) → 0.250 (worst) | Parzialmente confermata |
| ZF + RealAmplitudes | Media | 0.600 | Sopra attese |

*Prestazione buona ma inferiore alla corrispondente con ZFeatureMap.

Tabella 2.4: Aspettative vs. risultati sperimentali.

**Conclusioni operative** In sintesi, gli esperimenti confermano la regola pratica "*semplice nell'encoding, potente nell'ansatz*" per VQC su dataset di piccola taglia. Inoltre, per dataset preprocessati (PCA) con dimensionalità compattata ad un numero di componenti moderate:

*Encoding semplici + ansatz espressivi » soluzioni complesse*

La non linearità aggiuntiva di `ZZFeatureMap` e `PauliFeatureMap` ha deteriorato le performance anziché migliorarle, evidenziando l'importanza dell'adeguatezza al problema piuttosto che la complessità intrinseca.

### 2.4.1 Circuiti analizzati

In questa sezione presentiamo alcuni dei circuiti quantistici generati dalle diverse combinazioni di encoding e ansatz.

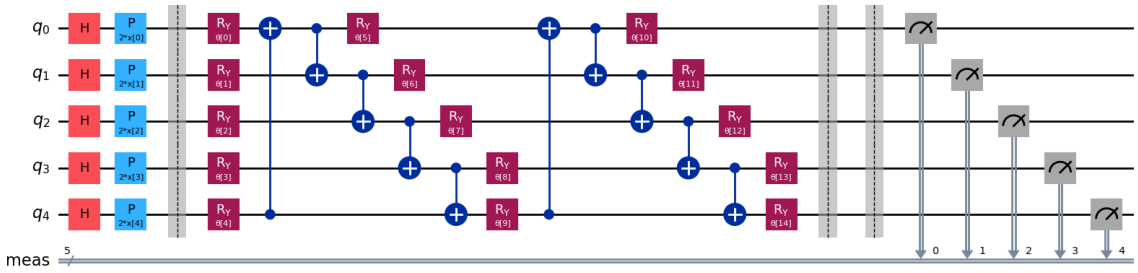



Figura 2.4: ZFeatureMap & RealAmplitudes

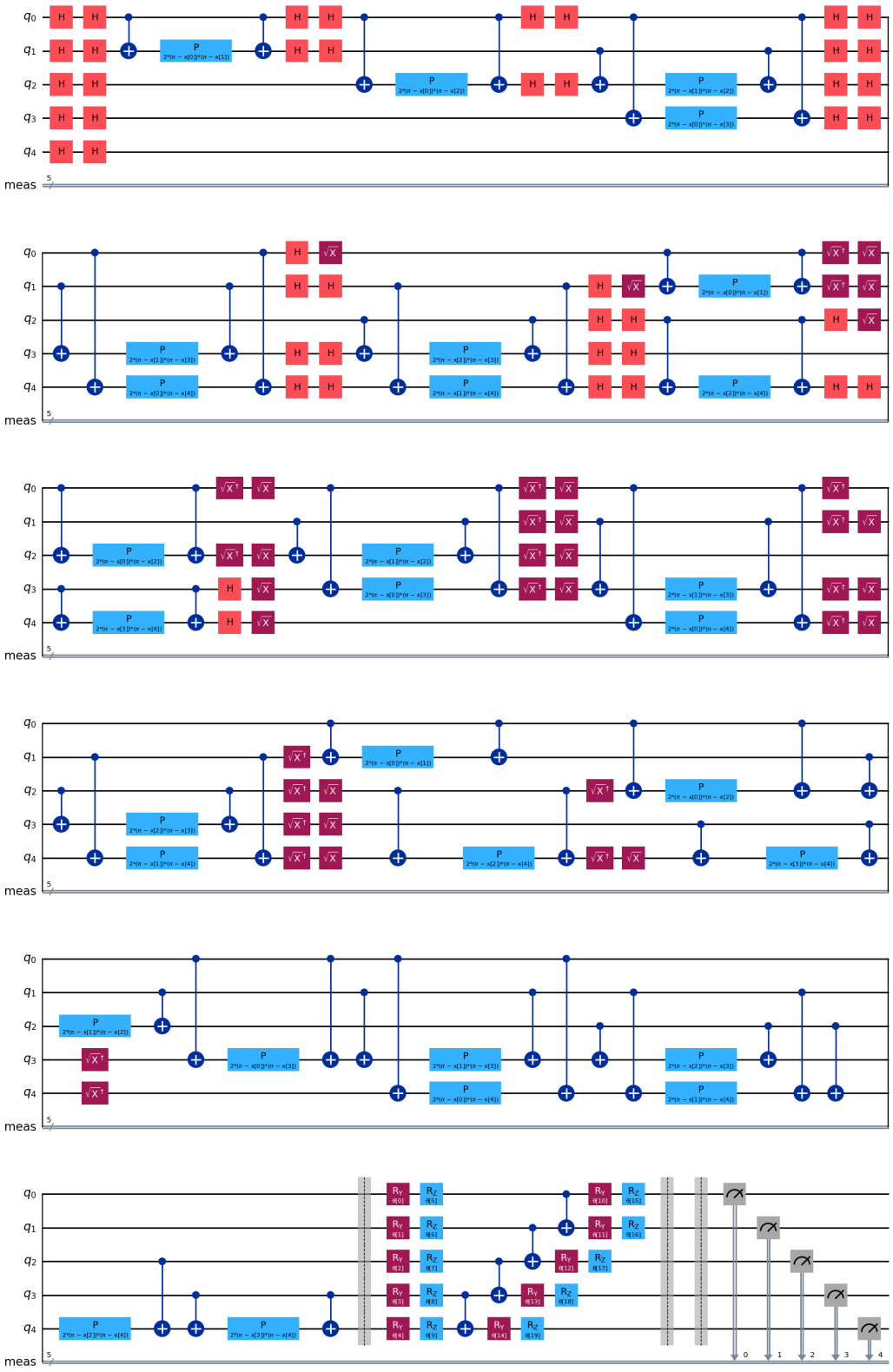



Figura 2.5: PauliFeatureMap & EfficientSU2

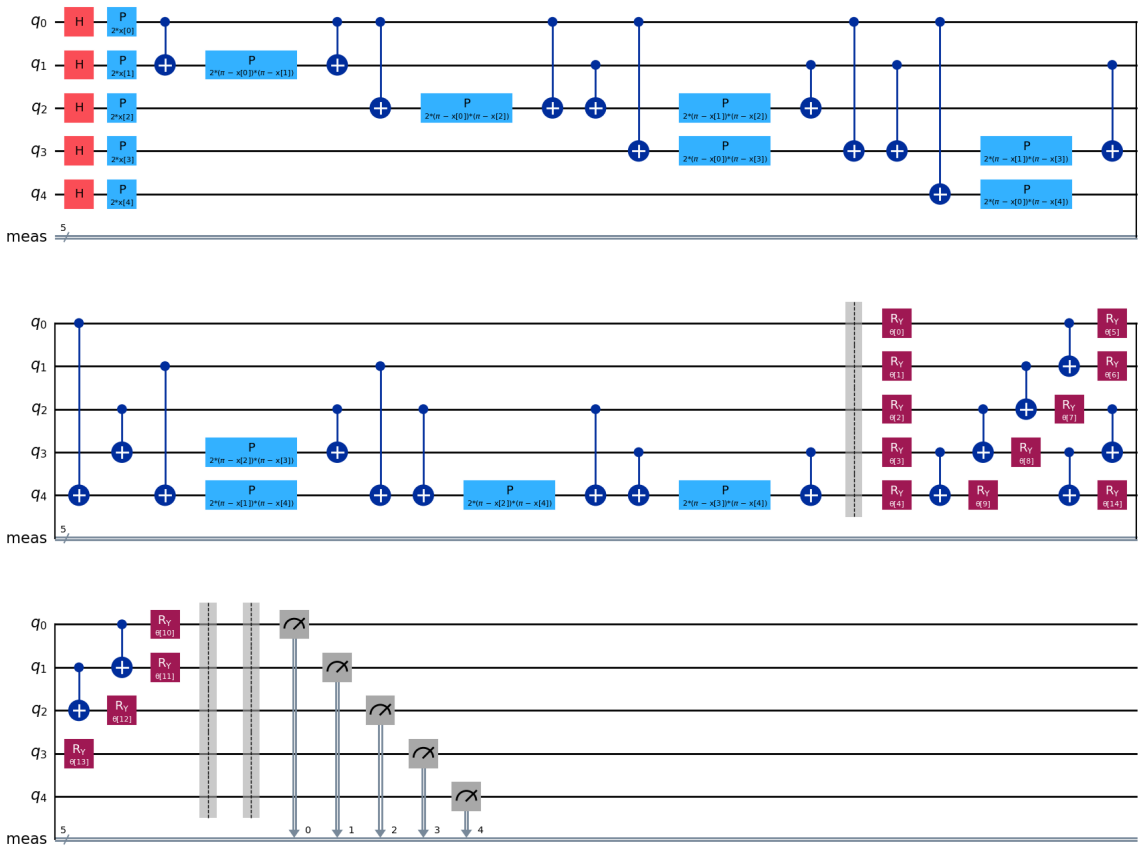



Figura 2.6: ZZFeatureMap & TwoLocal

### 2.4.2 Verifica di coerenza su preprocessamenti alternativi

Per testare la robustezza della regola empirica *"semplice nell'encoding, potente nell'ansatz"* elaborata nella sezione precedente (Wine → PCA 5 componenti), abbiamo confrontato i risultati con:

1. **PCA 90 % di varianza** (8 componenti);
2. **SelectKBest** (5 feature con punteggio `f_classif` massimo).

#### Performance degli Encoding nelle Diverse Strategie

Tabella 2.5: Confronto delle prestazioni degli encoding

| Encoding | PCA 5 comp. | Best 5 feat. | PCA 90% var. |
|---|---|---|---|
| **ZFeatureMap** | 0.8333 (rank 1) | 0.8333 (rank 1-2) | 0.6944 (rank 1) |
| **ZZFeatureMap** | 0.6944 (rank 4) | 0.6944 (rank 6) | 0.4167 (rank 7) |
| **PauliFeatureMap** | 0.6944 (rank 5) | 0.6100 (rank 7) | 0.5022 (rank 4) |

**Analisi Dettagliata**

- **ZFeatureMap**:
  - Miglior encoding in **tutte le strategie**
  - Unica configurazione con performance quasi sopra 0.7 in tutti gli scenari
  - Meno sensibile all'aumento di dimensionalità (-16.7% vs -40% di ZZFeatureMap)
- **ZZFeatureMap**:
  - Forte degradamento con 8 qubit: **-40% accuracy**
  - Sensibilità al rumore: l'entanglement aggiuntivo peggiora le performance
  - Miglior risultato sempre inferiore allo ZFeatureMap
- **PauliFeatureMap**:
  - Performance migliori con PCA a 5 componenti (+19% vs PCA90% e +8% vs Best5Features)
  - Sensibile alla strategia di selezione feature

**Performance degli Ansatz nelle Diverse Strategie**

Tabella 2.6: Confronto delle prestazioni degli ansatz

| Ansatz | PCA 5 comp. | Best 5 feat. | PCA 90% var. |
|---|---|---|---|
| **EfficientSU2** | 0.8333 (rank 1) | 0.8333 (rank 2) | 0.5833 (rank 3) |
| **TwoLocal** | 0.7778 (rank 2) | 0.8333 (rank 1) | 0.6389 (rank 2) |
| **RealAmplitudes** | 0.7500 (rank 3) | 0.7500 (rank 3) | 0.6944 (rank 1) |

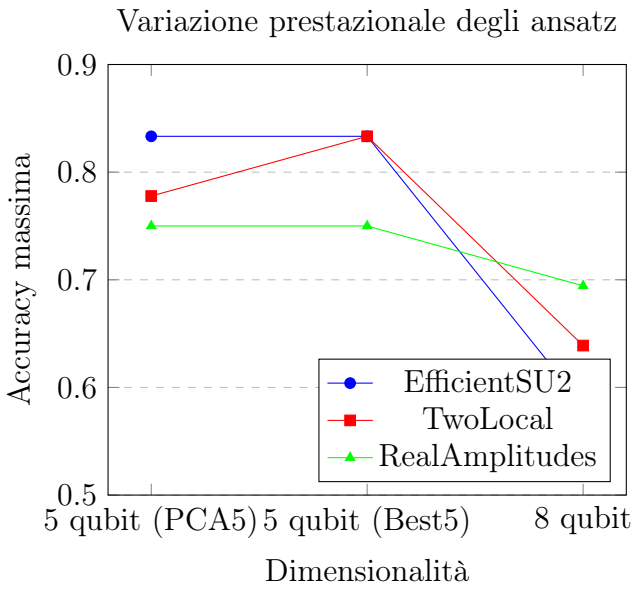



Figura 2.7: Andamento delle prestazioni al variare della dimensionalità

**Analisi Dettagliata**

- **EfficientSU2**:
  - Miglior performer a **5 qubit** (0.8333 in entrambe le strategie)
  - Forte degradamento con 8 qubit: **-30% accuracy**
  - Tempi di taining più lunghi
- **TwoLocal**:
  - Performance eccellenti con le Best5Selection (0.8333)
  - Maggiore stabilità all'aumento dei qubit (-23% vs -30% di EfficientSU2)
- **RealAmplitudes**:
  - Ansatz con le performance **migliori a 8 qubit** (0.6944)
  - Stabilità notevole: variazione minore tra strategie ($\pm$7.4%)

**Conclusioni Finali e Indicazioni Pratiche**

- **Coerenza globale.** In tutti e tre i casi il miglior encoding è sempre la **ZFeatureMap**, che sembra essere più determinate del tipo di ansatz scelto.
- **Impatto dell'aumento di dimensionalità (5 $\rightarrow$ 8 feat.).** Con l'aumento della dimensionalità, le scelte progettuali in merito all'Ansatz assumono maggiore rilevanza, infatti i risultati dimostrano che:
  - La scelta dell'ansatz deve considerare la dimensionalità:

1. `EfficientSU2` per bassa dimensionalità
2. `RealAmplitudes` per alta dimensionalità

- **Selezione guidata da filtro.** Anche con feature scelte via SelectKBest, l'accoppiata ZFeatureMap + TwoLocal e, a brevissima distanza, ZFeatureMap + EfficientSU2 dominano il podio, mentre ZZ/Pauli FeatureMap arretrano: la non-linearità extra sommata alla selezione già mirata delle caratteristiche porta ad un *over-parametrization* che penalizza la generalizzazione.

**Conclusione** I tre set di esperimenti, pur differendo per tecnica di riduzione/selezione delle feature, concordano nel sancire il *trade-off ottimale* per VQC sul Wine dataset: *encoding lineare (Z) + ansatz espressivo*. La complessità va dunque concentrata sul circuito variazionale, lasciando l'encoding quanto più possibile leggero e lineare.

# Capitolo 3

# Analisi del ruolo dell'ottimizzatore

## 3.1 Contesto e motivazione

Nei circuiti quantistici variazionali (VQC) la routine di addestramento si riduce alla minimizzazione di una funzione di costo (qui la `CrossEntropyLoss` ) in uno spazio di parametri $\boldsymbol{\theta} \in \mathbb{R}^d$. L'ottimizzatore classico che aggiorna $\boldsymbol{\theta}$ incide pesantemente[1] sull'efficacia finale dell'ibrido «quantum–classical loop». Nel nostro studio abbiamo valutato, inizialmente, tre ottimizzatori popolari di `Qiskit`:

- **COBYLA** (Constrained Optimization BY Linear Approximations)
- **L-BFGS-B** (Limited–memory Broyden–Fletcher–Goldfarb–Shanno with bounds)
- **SLSQP** (Sequential Least-Squares Quadratic Programming)

## 3.2 Formulazione delle aspettative *a priori*

Prima degli esperimenti, sulla linea di quanto già fatto con econding ed ansatz, sono state formulate le seguenti ipotesi teoriche:

[1] Incide sia sul numero di valutazioni del circuito — e quindi sul tempo di calcolo, soprattutto quando si passa dal simulatore all'hardware — sia sulla qualità del minimo raggiunto, ovvero sul grado di generalizzazione.

| **Ottimizzatore** | **Aspettative** | **Motivazione** |
|---|---|---|
| `COBYLA` | Miglior compromesso accuracy-tempo, robustezza al rumore | Non richiede gradienti, adatto a spazi parametrici non lisci |
| `L-BFGS-B` | È molto efficiente quando il paesaggio della loss è regolare, ma corre il rischio di fermarsi troppo presto su un minimo locale. | Sensibile al rumore, approssimazioni Hessiana amplificano errori |
| `SLSQP` | Potente ma computazionalmente oneroso | Richiede calcoli esatti di gradienti/Hessiane, instabile con piccoli dataset |

**Accuratezza prevista.**

Le tecniche basate su informazione di gradiente (L–BFGS–B e SLSQP) dovrebbero, in teoria, muoversi con passi più informati nei dintorni del minimo, ottenendo valori di costo inferiori rispetto a metodi *derivative-free* come COBYLA. Ci si aspetta dunque $\mathrm{Acc}_{\text{L–BFGS–B}} \sim \mathrm{Acc}_{\text{SLSQP}} > \mathrm{Acc}_{\text{COBYLA}}$.

**Tempo d'addestramento previsto.**

Per ogni iterazione gradient-based occorrono $2d$ valutazioni di circuito (regola del *parameter-shift* [2]), mentre COBYLA ne richiede tipicamente $d + 1$. Inoltre SLSQP risolve un problema QP interno per la stima dello step, e L-BFGS-B deve ad ogni iterazione sfruttare le informazioni del primo ordine per calcolare approssimazioni della matrice Hessiana che utilizza per il calcolo del nuovo punto in cui spostarsi. Pertanto si ipotizza l'esitenza di una relazione del genere che coinvolge iterazioni e tempi per la convergenza dei vari algoritmi: COBYLA $<$ L–BFGS–B $<$ SLSQP.

[2] La parameter-shift rule è la "chiave" che rende possibili gli ottimizzatori gradient-based nei VQC: con solo due esecuzioni addizionali del circuito per parametro fornisce un gradiente esatto, eliminando il bisogno di simulare lo Jacobiano o di usare differenze finite su computer classico — tecniche che non sarebbero scalabili né compatibili con l'hardware quantistico.

## 3.3 Risultati ottenuti

| **Rank** | **Encoding** | **Ansatz** | **Optimizer** | **Test Acc (%)** |
|---|---|---|---|---|
| 1 | ZFeatureMap | EfficientSU2 | COBYLA | 83.33 |
| 2 | ZFeatureMap | TwoLocal | COBYLA | 77.78 |
| 3 | ZFeatureMap | RealAmplitudes | COBYLA | 75.00 |
| 4 | ZZFeatureMap | EfficientSU2 | COBYLA | 69.44 |
| 5 | PauliFeatureMap | TwoLocal | COBYLA | 69.44 |

Tabella 3.1: Top 5 configurazioni per accuracy

| Ottimizzatore | Accuracy test | | Accuracy max | Accuracy min | Iterazioni medie | Tempo (s) | |
|---|---|---|---|---|---|---|---|
| | Media | Dev.std. | | | | Media | Dev.std. |
| COBYLA | 0.645 ± | 0.130 | 0.833 | 0.444 | 191.6 | 290 ± | 65 |
| L-BFGS-B | 0.377 ± | 0.090 | 0.556 | 0.277 | 438.88 | 642 ± | 98 |
| SLSQP | 0.407 ± | 0.110 | 0.528 | 0.250 | 1938.9 | 2769 ± | 823 |

Tabella 3.2: Risultati aggregati degli ottimizzatori

| **Criterio** | **Vincitore** | **Motivazione** |
|---|---|---|
| Accuratezza | COBYLA | 83.33% vs 55.56% (L-BFGS-B) vs 52.78% (SLSQP) |
| Tempo di esecuzione | COBYLA | 290s vs 641s (L-BFGS-B) vs 2767s (SLSQP) |
| Stabilità | COBYLA | Deviazione std minore (12.46% vs 10.85% SLSQP) |
| Numero di iterazioni | COBYLA | 192 vs 439 (L-BFGS-B) vs 1939 (SLSQP) |

Tabella 3.3: Sintesi prestazioni ottimizzatori

### 3.3.1 Analisi e discussione

**COBYLA domina il compromesso accuracy–tempo–iterazioni.** COBYLA raggiunge sia la miglior accuratezza massima (83.3 %) sia la miglior media (64.5 %), pur mantenendo un numero di iterazioni contenuto rispetto gli avversari, ed un tempo di training intorno ai $\sim 5$ minuti. L'algoritmo, basato su semplici approssimazioni lineari della regione di fiducia, dimostra una sorprendente robustezza alle oscillazioni del paesaggio di costo e beneficia del ridotto numero di valutazioni per iterazione.

**L–BFGS–B delude.** Il metodo quasi-Newton, malgrado l'uso efficiente della memoria, si ritrova spesso in «barren plateaus» dove il gradiente misurato cade sotto la soglia numerica. Di conseguenza gli aggiornamenti diventano cauti, allungando il training time e le iterazioni, lasciando la rete intrappolata in minimi di qualità inferiore.

**SLSQP: accuratezza intermedia ma iterazioni e tempi proibitivi.** L'ottimo interno di un QP ad ogni passo, unito alle valutazioni di gradiente, fa impennare il tempo medio a $\sim 46$ minuti; anche le iterazioni sono ben oltre la media degli altri ottimizzatori, si registra quasi 10 volte il numero di iterazioni di COBYLA e 5 quelle di L-BSGF-B. A questo aumento di complessità computazionale non corrisponde però un corrispettivo vantaggio in accuratezza.

### 3.3.2 Valutazioni

- **Sperimentazione rapida**.Se l'obiettivo è iterare velocemente, COBYLA offre il miglior compromesso ed evita lunghe code di job sull'hardware.
- **Spinta all'accuratezza**.La scelta dell'ottimizzatore impatta più di encoding/ansatz. Differenze elevate in accuracy dimostrano che l'ottimizzazione è il collo di bottiglia critico. COBYLA, nello specifico, è l'ottimizzatore più adatto per VQC: 45.68% più accurato di L-BFGS-B e 23.78% più accurato di SLSQP.
- **Scalabilità**. L'efficienza di COBYLA potrebbe ridursi all'aumentare dei parametri coinvolti nel circuito; in quel regime L–BFGS–B con le dovute migliorie potrebbe riacquistare competitività.

### 3.3.3 Analisi di iterazioni e tempistiche

L'addestramento dei VQC evidenzia i seguenti risultati in termini di iterazioni e tempistiche:

- **COBYLA** completa in genere ogni run in tempi che vanno da 220 s a 400 s ed in un numero di iterazioni che va da 155 a 290
- **L–BFGS–B** richiede da 526 s a oltre 800 s per via delle $2d$ valutazioni di *parameter-shift* a ogni iterazione, quest'ultime oscillano da 270 a 500.
- **SLSQP** è di un ordine di grandezza più lento: da ∼25 min a oltre 68 min perché ogni step risolve un QP interno *e* calcola i gradienti. Peggiori anche a livello di oscillazioni delle iterazioni che vanno 900 a 2600

### 3.3.4 Lettura delle confusion matrix

Le metriche aggregate non raccontano dove avvengano gli errori; per questo abbiamo esaminato le confusion matrix prodotte a fine run.

### Analisi dei Casi Peggiori

#### COBYLA: PauliFeatureMap + RealAmplitudes

- **Accuracy**: 41.6%
- **Iterazioni e Tempo**: 175 e 475.80s
- **Matrice di confusione**:

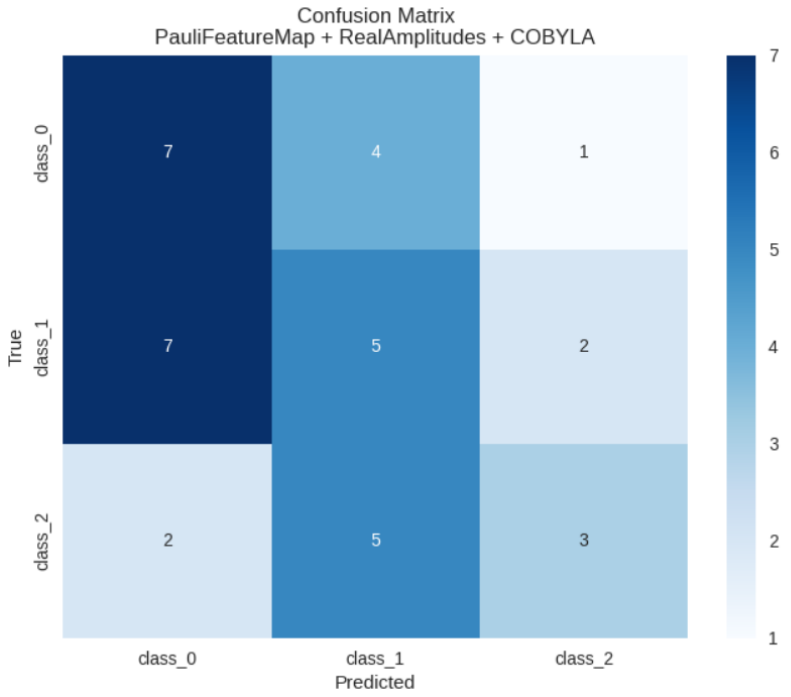



Figura 3.1: Matrice di confusione del peggiore caso per COBYLA

#### L-BFGS-B: PauliFeatureMap + TwoLocal

- **Accuracy**: 25.00%
- **Iterazioni e Tempo**: 350 e 863.20s
- **Matrice di confusione**:

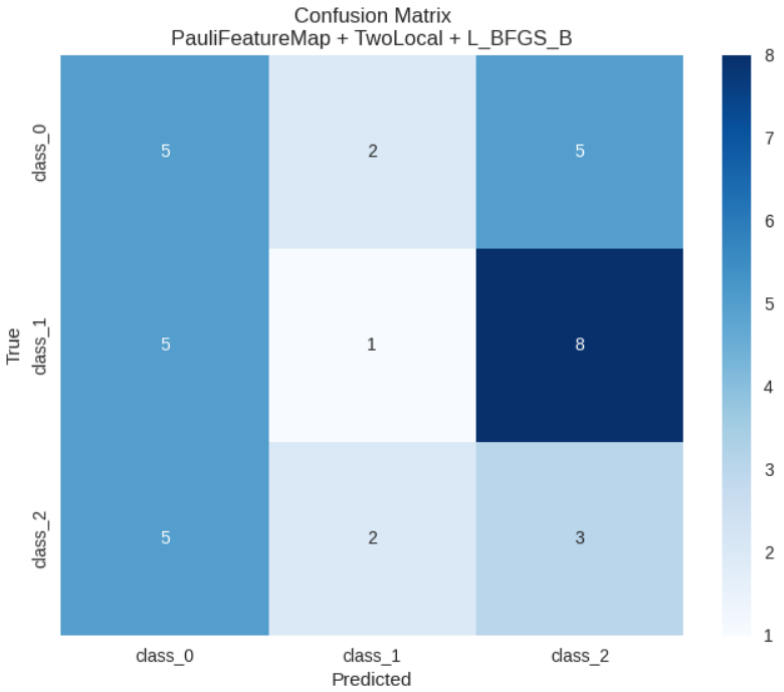



Figura 3.2: Matrice di confusione del peggiore caso per L-BSFG-B

- **Diagnosi**: Collasso su singola classe dovuto a convergenza prematura

**SLSQP: PauliFeatureMap + EfficientSU2**

- **Accuracy**: 22.22%(peggiore assoluta)
- **Iterazioni e Tempo**: 2900 e 7560.98s
- **Matrice di confusione**:

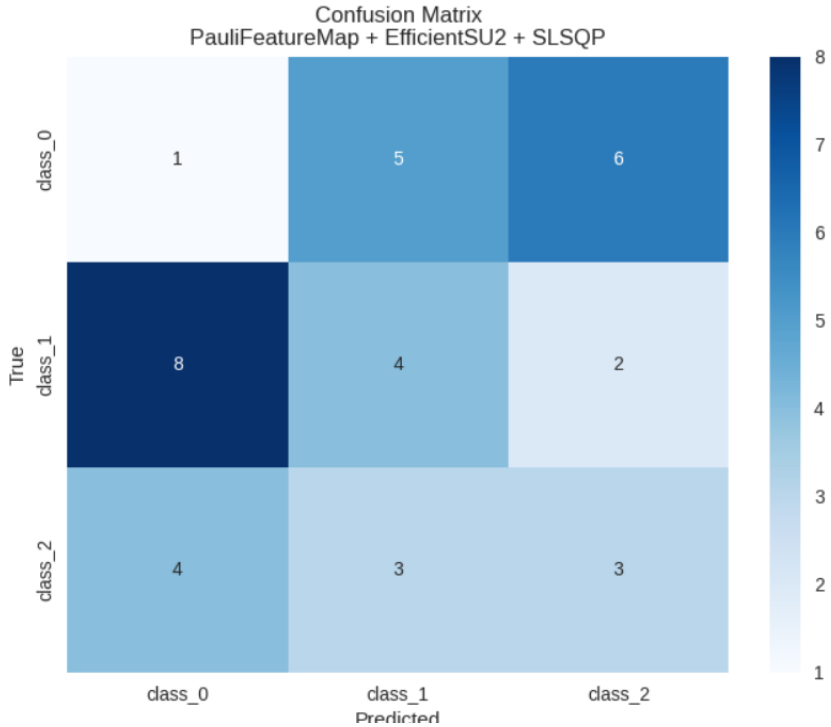



Figura 3.3: Matrice di confusione del peggiore caso per SLSQP

- **Diagnosi**: Distribuzione casuale delle predizioni dopo 1500+ iterazioni

**Altre casistiche interessanti**

**COBYLA.** Le matrici dei migliori esperimenti mostrano una diagonale quasi perfetta: nell'esperimento vincente (ZFeatureMap + EfficientSU2) solo 1 mistake su 12 campioni di *class 0* e 1 su 14 di *class 1*; *class 2* è sempre predetta correttamente Gli errori residui avvengono quasi esclusivamente fra le due classi chimicamente più simili *class 0* e *class 1*.

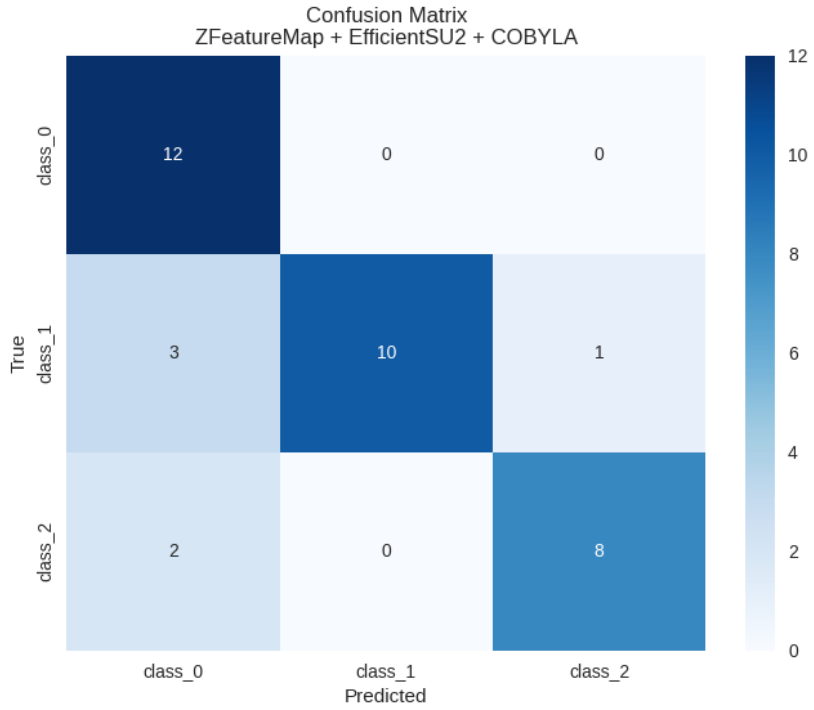



Figura 3.4: Matrice di confusione del migliore caso per COBYLA

**L–BFGS–B.** Quando il gradiente diventa numericamente instabile, il modello «collassa» su un'unica classe: lo si vede dalla colonna quasi vuota nella confusion matrix di ZFeatureMap+RealAmplitudes, dove precision scende a 0.11 e recall rimane a 0.33.

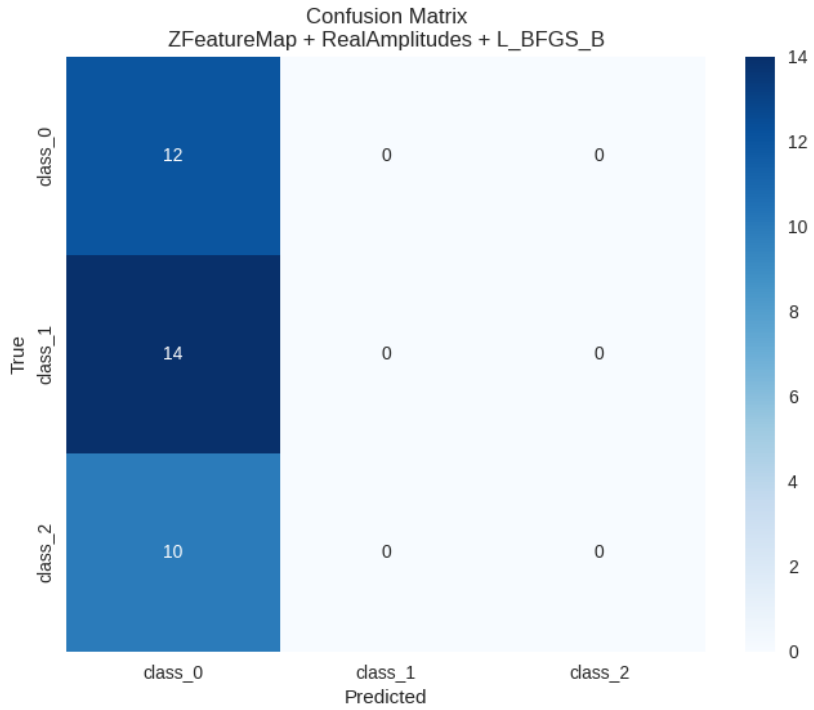



Figura 3.5: Matrice di confusione collassata L-SBGF-B

**SLSQP.** Le matrici sono più «nebulose»: il caso migliore (ZZFeatureMap+EfficientSU2) riporta una diagonale con valori $\{8, 9, 2\}$ sui 12+14+10 campioni e numerosi scambi simmetrici fra *classe 0* e *classe 1* ed anche fra *classe 1* e *classe 2*

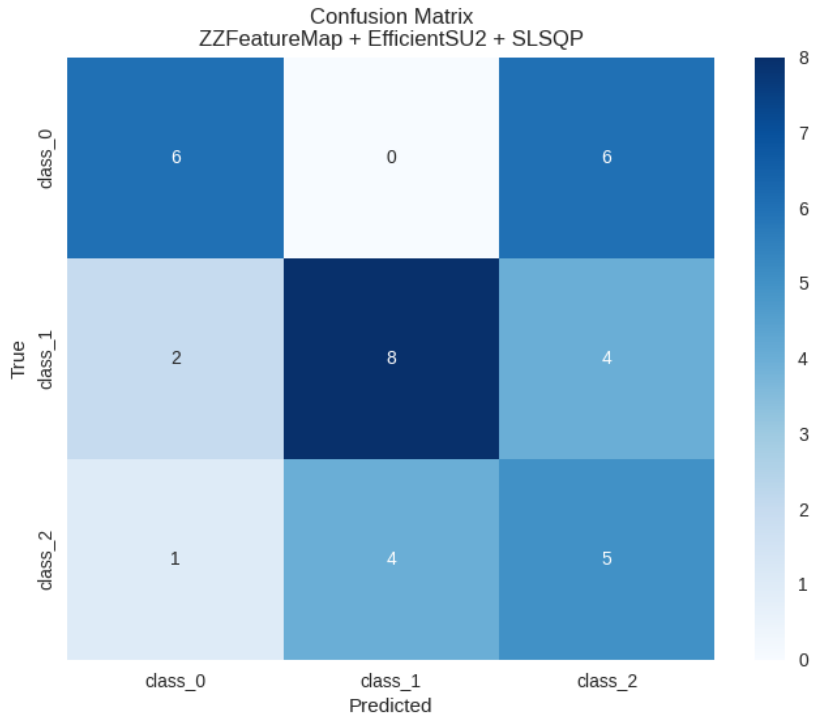



Figura 3.6: Matrice di confusione migliore SLSQP

Nel complesso, le confusion matrix confermano la *robustezza* di COBYLA: gli errori sono limitati e concentrati, mentre con L–BFGS–B e con SLSQP si osservano schemi di misclassificazione più diffusi e talvolta degenerati. Queste evidenze, combinate con i tempi di training, rafforzano la raccomandazione di scegliere COBYLA come *default* per VQC su dataset di dimensioni analoghe, e forniscono uno spunto per ricercare maggiore chiarezza tramite un'analisi grafica delle funzioni di convergenza.

### 3.3.5 Esame grafico della convergenza

Di seguito sono riportate le curve di Log Loss registrate nel corso dell'addestramento: la loro discesa progressiva consente di valutare a colpo d'occhio rapidità e stabilità di convergenza per le diverse combinazioni di encoding, ansatz e ottimizzatore.

### Analisi della convergenza dell'ottimizzatore `COBYLA`

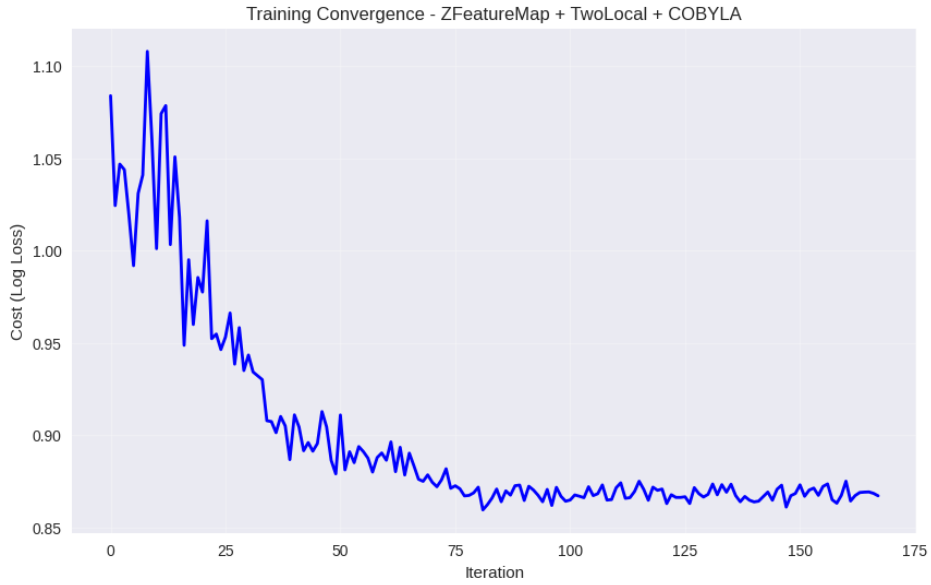



Figura 3.7: Evoluzione del Log Loss – ZFeatureMap + TwoLocal ottimizzato con COBYLA

### Analisi della convergenza dell'ottimizzatore `L-BFGS-B`

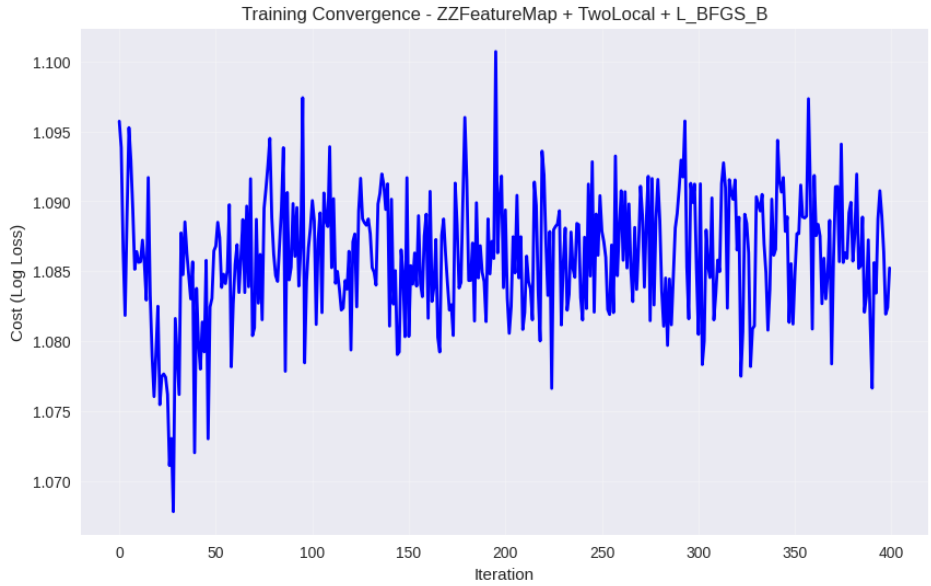



Figura 3.8: Evoluzione del Log Loss – ZZFeatureMap + TwoLocal ottimizzato con L-BFGS-B

Le curve di perdita mostrate nella Figura 3.8 evidenziano che, per la quasi totalità delle combinazioni di *encoding* e ansatz considerate, la funzione obiettivo non converge, l'algoritmo non riesce a ridurre la loss in maniera stabile: la discesa si arresta prematuramente o mostra oscillazioni di ampiezza crescente.

**Evidenze numeriche.** Il mancato apprendimento è confermato anche dai valori finali della loss riportati negli esperimenti: per `L-BFGS-B` essi si attestano sistematicamente nell'intervallo 1.06–1.20, mentre con l'ottimizzatore `COBYLA` le stesse configurazioni raggiungono spesso valori inferiori a 0.90. In particolare, il caso `ZFeatureMap + RealAmplitudes` chiude a 1.1493, contro 0.8805 ottenuto da `COBYLA`.

**Problemi di Convergenza** I principali indicatori di non-convergenza osservati includono:

- **Oscillazioni persistenti**: La loss mostra fluttuazioni continue senza stabilizzazione
- **Stagnazione in plateau**: La funzione si blocca in regioni piatte senza progressi significativi
- **Assenza di discesa monotònica**: Mancanza di un trend decrescente coerente oltre le prime iterazioni

**Possibili cause della non-convergenza.**

- **Rumore e stima del gradiente.** `L-BFGS-B` richiede una stima precisa del gradiente; con il *parameter-shift rule* su circuiti a 5 qubit il rumore statistico (numero limitato di *shots*) degrada la qualità dei gradienti, causando passi di aggiornamento non affidabili.
- **Paesaggi di ottimizzazione complessi.** Gli ansatz scelti generano superfici di perdita altamente non-convesse, soggette a *barren plateaus*: regioni in cui le derivate si annullano esponenzialmente con il numero di qubit, rendendo inefficace un metodo basato su informazioni di secondo ordine approssimate.
- **Hessian mal condizionato.** In presenza di molteplici parametri (15–20 nei nostri esperimenti) la matrice hessiana approssimata può risultare quasi singolare; il calcolo della direzione di discesa risente di errori numerici e amplifica il rumore del gradiente.

**Implicazioni sui Risultati** Questa evidenza sperimentale impone una **rilettura critica** delle metriche di classificazione riportate nelle sezioni precedenti:

1. I valori di accuracy, precision e recall sono ottenuti da **parametri non ottimizzati**
2. Le performance riportate (accuracy test media: $0.5108 \pm 0.1752$) rappresentano delle *stime* delle potenzialità dei circuiti
3. I confronti tra configurazioni perdono significato statistico poiché nessun circuito ha raggiunto il suo optimum

**Conclusioni.** L'analisi grafica della funzione obiettivo evidenzia che le cattive prestazioni delle metriche dipendono direttamente dalla non-convergenza dell'ottimizzatore. Le classificazioni errate osservate nelle matrici di confusione potrebbero essere attribuibili più alla mancata convergenza che a limiti intrinseci dei circuiti. Prima di trarre conclusioni definitive sull'efficacia delle architetture variazionali studiate, sarà necessario provare ad esplorare ottimizzatori differenti.

**Analisi della convergenza dell'ottimizzatore `SLSQP`**

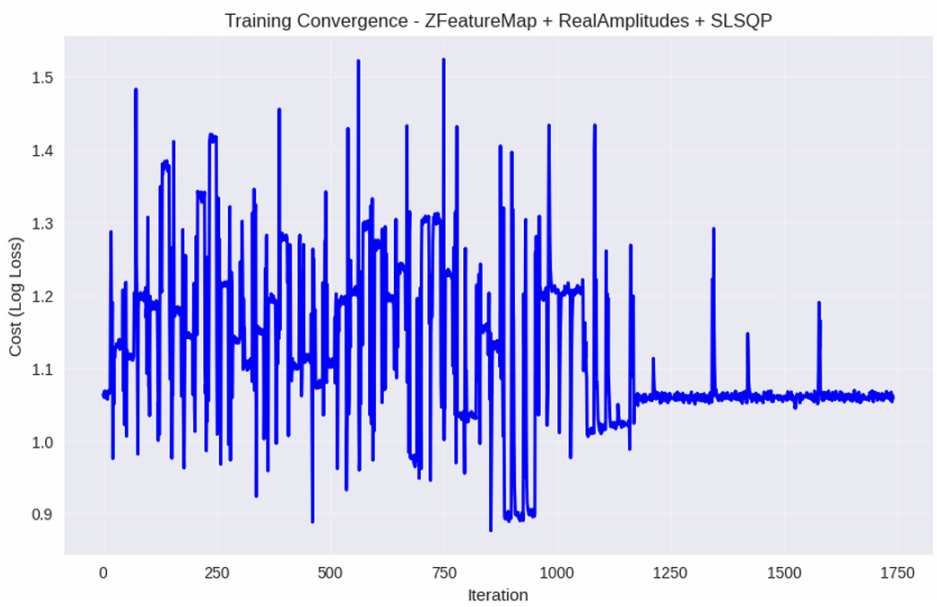



Figura 3.9: Evoluzione del Log Loss – ZFeatureMap + RealAmplitudes ottimizzato con SLSQP

## 3.4 Valutazione di ottimizzatori alternativi

### 3.4.1 Perché provare altri ottimizzatori

Le considerazioni fatte in merito agli ottimizzatori scelti, evidenziano come **COBYLA** - unico metodo *derivative-free* nella rosa iniziale - superasse nettamente l'ottimizzatore SLSQP basato su gadiente (ricordando che le considerazioni fatte sulle metriche di L-BSFGS-B rappresentano una stima grossolana delle prestazioni del circuito) ha suggerito di testare altri due algoritmi privi di gradienti: il metodo *POWELL* di minimizzazione per ricerca direzionale ciclica ed il metodo *NELDER-MEAD*, cioè la minimizzazione di una funzione scalare di una o più variabili tramite l'algoritmo Nelder-Mead. L'ipotesi da verificare era duplice:

1. se la famiglia *derivative-free* garantisse sistematicamente accuratezze superiori
2. a quale costo di iterazioni e temporale la funzione obiettivo converge

### 3.4.2 Statistiche complessive degli esperimenti

Le solite combinazioni (3 encoding × 3 ansatz) fornisce:

| **Ottimizzatore** | **Acc. medio** | **Acc. max** | $T_{\textbf{train}}$ **medio [s]** |
|---|---|---|---|
| COBYLA | 0.645 | 0.833 | 290 |
| L–BFGS–B | 0.377 | 0.556 | 642 |
| SLSQP | 0.407 | 0.528 | 2 769 |
| **POWELL** | **0.747** | **0.861** | 2 309 |
| **NELDER-MEAD** | **0.632** | **0.916** | **9149** |

**POWELL**

**Prestazioni di punta.** La configurazione ZFeatureMap + EfficientSU2 + POWELL raggiunge un nuovo record di Acc = 0.8611 con F1 = 0.8571 ma richiede ∼3100 iterazioni e ∼59 minuti di training (3577 s) per ottimizzare ***20 parametri***

**Comportamento medio.**

$$\overline{\mathrm{Acc}}_{\mathrm{test}} = 0.7469\ , \qquad \overline{T}_{\mathrm{train}} = 2309\,\mathrm{s},$$

**Confronto qualitativo con gli altri ottimizzatori**

**Accuracy.** POWELL stabilisce il *nuovo massimo assoluto* (0.861) e una accuracy media di quasi 10 punti percentuali più alta di COBYLA.

**Iterazioni e Tempo** Il rovescio della medaglia è il costo computazionale: POWELL si colloca un ordine di grandezza sopra COBYLA sia come numero di iterazioni che come tempo (media iterazioni ∼191.6 COBYLA vs ∼1777.8 POWELL e mediana tempi ∼4,83 min COBYLA vs ∼40–45 min POWELL), allineandosi alla lentezza di SLSQP.

**Visualizzazione grafica**

L'andamento della funzione obiettivo di POWELL ci pone davanti a problemi analoghi a quanto già visto con L-BFGS-B, di conseguenza anche queste metriche valutative non sono comparibili con quelle ottenute per COBYLA ed SLSQP, in quanto costituiscono solo un'approssimazione di quelle che potremmo ottenere in presenza di convergenza.

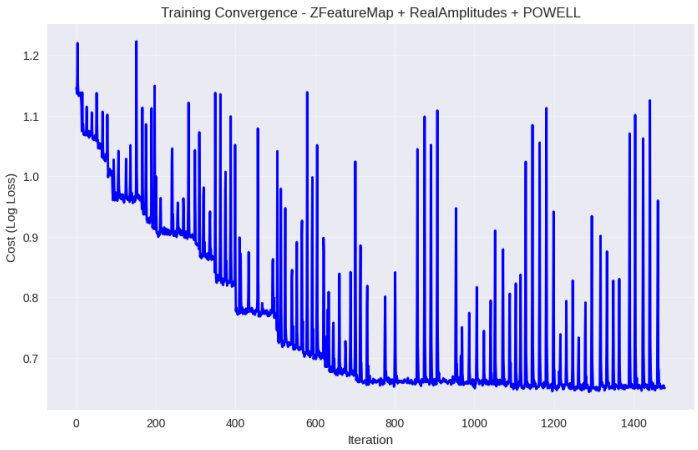



Figura 3.10: Grafico convergenza POWELL

### NELDER-MEAD

**Prestazioni di punta.** La configurazione ZFeatureMap + TwoLocal + NELDER-MEAD raggiunge un nuovo record di Acc = 0.9167 con F1 = 0.9157 ma richiede ∼7100 iterazioni e ∼95 minuti di training (5673 s) per ottimizzare ***15 parametri***

**Comportamento medio.**

$$\overline{\mathrm{Acc}}_{\mathrm{test}} = 0.6327\ , \qquad \overline{T}_{\mathrm{train}} = 9149\,\mathrm{s},$$

**Confronto qualitativo con gli altri ottimizzatori**

**Accuracy.** Subito abbattuto il nuovo record stabilito da POWELL grazie all'accoppiata ZFeatureMap+TwoLocal che con il 91.67% di accuracy si collaca al primo posto tra i circuiti valutati

**Iterazioni e Tempo** Il rovescio della medaglia è il costo computazionale: NELDER-MEAD è tremendamente più oneroso dei suoi competitor. Infatti si colloca sopra COBYLA sia come numero di iterazioni che come tempo (media iterazioni ∼191.6 COBYLA vs ∼7688,88 NELDER-MEAD e mediana tempi ∼4,83 min COBYLA vs ∼153 min NELDER-MEAD), andando oltre più del triplo della di SLSQP, che finora era risultato il più inefficiente.

#### Visualizzazione grafica

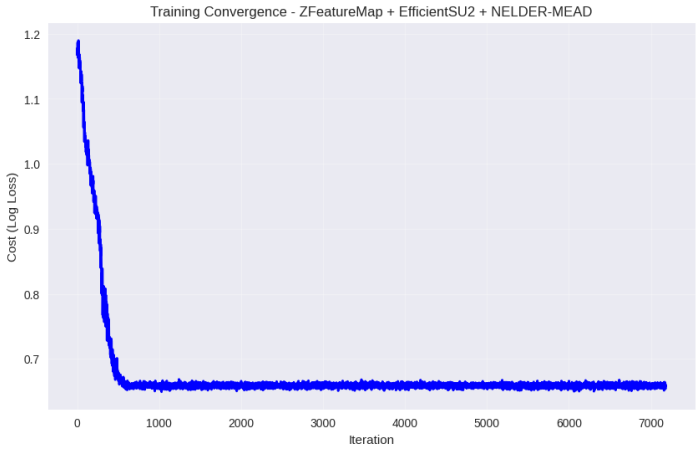



Figura 3.11: Grafico convergenza NELDER-MEAD

In questo caso si può vedere come la funzione obiettivo converga, in particolare l'ottimizzatore si mantiene vicino al valore finale per numerose iterazioni. Possiamo dunque accettare le valutazioni fatte sulle prestazioni di questo ottimizzatore a differenza di quanto invece accade per *L-BSGF-B* e *POWELL*

### 3.4.3 Conclusioni operative

#### Conclusioni finali sugli ottimizzatori

Alla luce dei risultati, **COBYLA si afferma quale scelta di default** per VQC di piccole–medie dimensioni sul *Wine Dataset*, offrendo un rapporto accuracy–iterazioni che i metodi gradient-based non riescono a eguagliare nelle condizioni considerate. SLSQP non riesce a mantenere le alte prestazioni, in termini di accuracy, di COBYLA ed inoltre risulta essere il secondo ottimizzatore più oneroso per iterazioni e tempistiche tra quelli considerati. **POWELL e NELDER-MEAD completano il quadro**: il primo non ci permette di avvalerci dei risultati ottenuti per esprimere pareri in merito alla sua efficienza per il problema preso in esame, mentre il secondo sembra, nonostante la grande parentesi legata all'efficienza, confermare che gli ottimizzatori senza gradiente dominano in accuratezza (NELDER-MEAD, poi COBYLA), mentre quelli gradient-based si rivelano poco competitivi (SLSQP ed L-BSGF-B). La scelta finale dipenderà dunque dal vincolo primario del progetto: il trade-off tra *tempo* e *prestazione finale*.

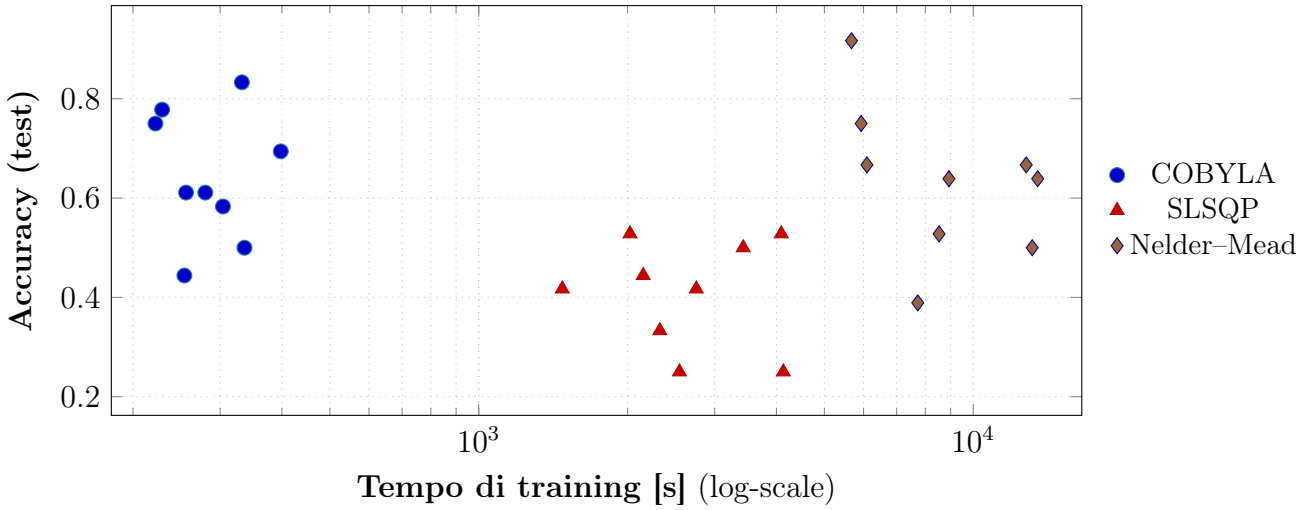



Figura 3.12: Trade-off *accuracy–tempo*: COBYLA domina la regione veloce + accurata; Nelder-Mead ottiene l'accuracy più alta sacrificando tempo.

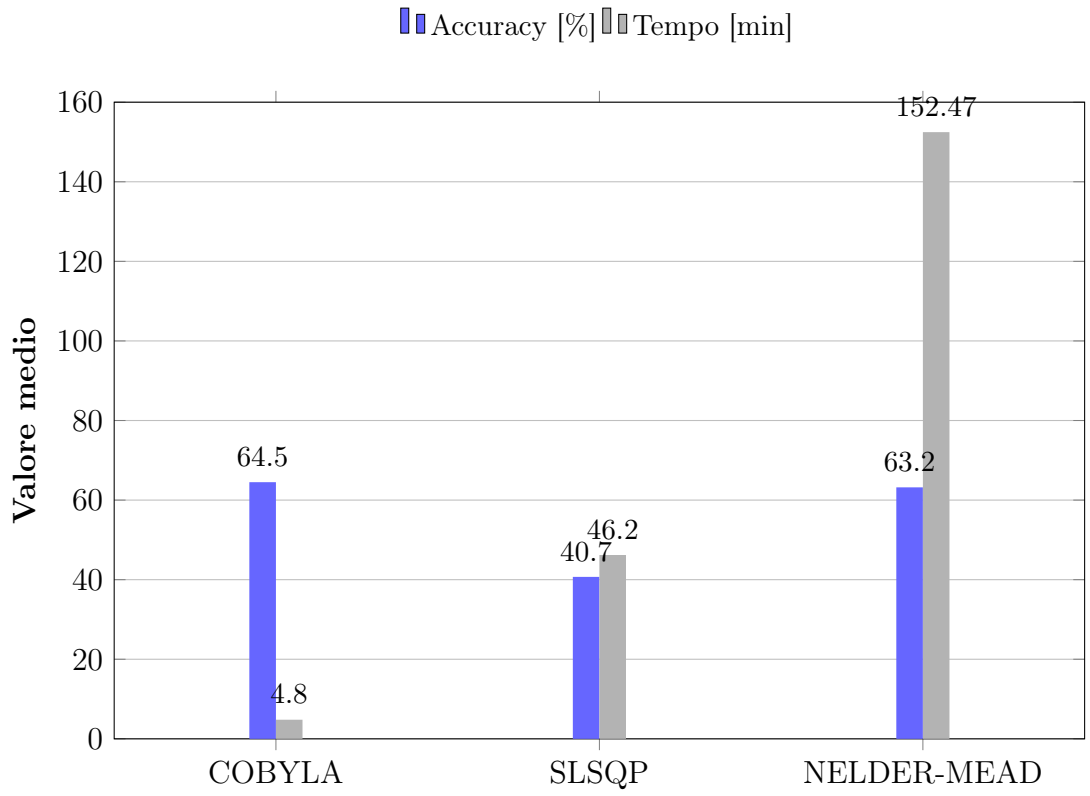



Figura 3.13: Medie comparative: Nelder-Mead raggiunge un'accuracy simile a COBYLA ma con tempi di training significativamente più lunghi.

# Capitolo 4

# Confronto tra Machine Learning classico e quantistico

## 4.1 Perché affiancare i VQC ai modelli classici?

Abbiamo pensato fosse utile mettere a confronto i nostri *Variational Quantum Circuits* (VQC) con i più noti algoritmi di *Machine Learning* classico dietro questa scelta ci sono tre motivi fondamentali:

1. **Contestualizzare le prestazioni**
   Collocare un VQC accanto a SVM, Random Forest o Logistic Regression permette di capire subito, e con una metrica condivisa, quanto il paradigma quantistico sia già competitivo – e quanto terreno resti da conquistare.

2. **Valutare il costo computazionale**
   Confrontare i tempi di addestramento, le iterazioni e le risorse impiegate evidenzia l'impatto pratico dell'ottimizzazione ibrida, basata su molte esecuzioni del circuito, rispetto alle pipeline interamente classiche.

3. **Individuare le prossime direzioni di ricerca**
   Osservare dove il quantistico fatica, o viceversa si comporta in modo inatteso, suggerisce quali aspetti potenziare: scelta dell'*ansatz*, design delle *feature-map*, selezione di ottimizzatori più adatti o riduzione del numero di valutazioni di circuito.

In breve, il confronto con il ML classico fornisce il contesto indispensabile per rendere leggibili – anche a chi non si occupa ogni giorno di computing quantistico – l'efficacia, i limiti e le prospettive dei nostri VQC.

## 4.2 Modelli classici

La sperimentazione ha tenuto conto di:

- **6 modelli classici**: Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors, Gradient Boosting, Logistic Regression e Naive Bayes
- **Modelli quantistici**: 18 combinazioni di encoding (ZFeatureMap, ZZFeatureMap, PauliFeatureMap), ansatz (RealAmplitudes, TwoLocal, EfficientSU2) e ottimizzatori (COBYLA, SLSQP) [1]

## 4.3 Prestazioni dei modelli classici

Di seguito sono presentati i risultati degli esperimenti condotti sui modelli classici sullo stesso split del dataset:

| **Classificatore** | **Acc.** | **Prec.** | **Recall** | **F1** | **Train Time [s]** |
|---|---|---|---|---|---|
| Support Vector Machine (RBF) | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.0070 |
| Naïve Bayes | 0.9722 | 0.9741 | 0.9722 | 0.9721 | 0.0023 |
| Random Forest | 0.9444 | 0.9444 | 0.9444 | 0.9444 | 0.1542 |
| K-Nearest Neighbors | 0.9444 | 0.9524 | 0.9444 | 0.9444 | 0.0022 |
| Gradient Boosting | 0.9444 | 0.9444 | 0.9444 | 0.9444 | 0.3835 |
| Logistic Regression | 0.9444 | 0.9444 | 0.9444 | 0.9444 | 0.0106 |
| **Media (6 modelli)** | 0.9583 | 0.9600 | 0.9583 | 0.9583 | 0.0933 |

Tabella 4.1: Prestazioni sul *Wine Dataset* (5 feature, 36 campioni di test)

## 4.4 Analisi Comparativa Dettagliata: Modelli Classici vs. Quantistici

Il confronto tra l'ambito quantistico e classico è stato portato avanti a partire da considerazioni fatte sul miglior modello quantistico emerso dalla sperimentazione, che è risultato essere la combinazione `ZFeatureMap + EfficientSU2 + COBYLA`, e, per i modelli classici, la SVM, che si è distinta per le sue prestazioni perfette.

[1] Non vengono tenuti in considerazione per la seguente analisi i 18 circuiti sperimentali che coinvolgono gli ottimizzatori L-BFGS-B e POWELL, per le motivazioni ampiamente discusse. Inoltre, per motivi legati al rapporto accuracy-iterazioni, non si terrà in considerazione nemmeno l'ottimizzatore NELDER-MEAD nonostante abbia ottenuto l'accuracy più alta.

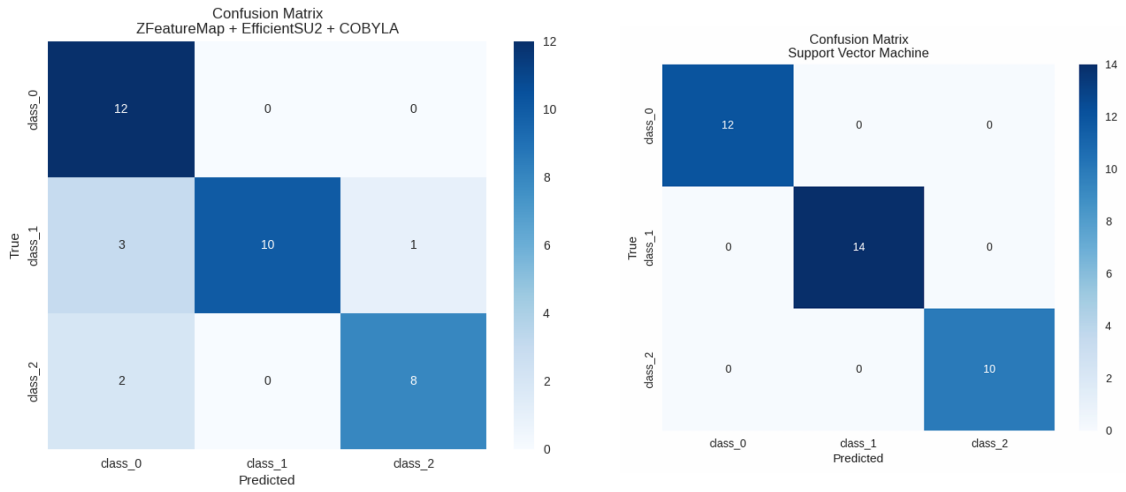
### 4.4.1 Prestazioni comparative: un divario significativo

L'analisi delle metriche principali rivela un sostanziale divario tra i due approcci. Come evidenziato nella Tabella 4.2, la SVM classica dimostra prestazioni ottimali in tutte le metriche, raggiungendo il 100% di accuratezza sul test set.

| Metrica | SVM (Classico) | Modello Quantistico | Differenza |
|---|---|---|---|
| Test Accuracy | 1.0000 | 0.8333 | **-16.67%** |
| Test Precision | 1.0000 | 0.8711 | -12.89% |
| Test Recall | 1.0000 | 0.8333 | -16.67% |
| Test F1-Score | 1.0000 | 0.8339 | **-16.61%** |
| Training Time (s) | 0.007 | 331.55 | **+47,000x** |

Tabella 4.2: Confronto metriche tra miglior modello classico e quantistico

Il modello quantistico, seppur migliore tra quelli quantistici testati, presenta un'**accuratezza inferiore di 16.67 punti percentuali** rispetto alla SVM. Questo gap prestazionale diventa particolarmente evidente analizzando le matrici di confusione riportate di seguito:




(a) Matrice di confusione per il miglior modello quantistico

(b) Matrice di confusione per il miglior modello classico

Figura 4.1: Confronto fra le due immagini

- **SVM**: Classificazione perfetta di tutti i 36 campioni di test
- **Modello quantistico**:
  - 12/12 campioni corretti per la Classe 0 (2 erroneamente classificati come Classe 1)
  - 10/14 campioni corretti per la Classe 1 (4 erroneamente classificati come Classe 2)

– 8/10 campioni corretti per la Classe 2

Questi errori concentrati nelle ultime due classi suggeriscono una **minore capacità discriminativa** del modello quantistico nel distinguere classi con caratteristiche chimiche simili.

### 4.4.2 Analisi delle cause del gap prestazionale

#### Fattori computazionali

Il confronto temporale è particolarmente significativo:

Tabella 4.3: Confronto tempi di addestramento

| **Modello** | **Training Time (s)** |
|---|---|
| SVM | 0.007 |
| Naive Bayes | 0.002 |
| K-Nearest Neighbors | 0.002 |
| **Modello Quantistico** | **331.55** |

Il modello quantistico richiede **oltre 47.000 volte più tempo** per l'addestramento rispetto alla SVM. Questo enorme overhead computazionale è dovuto alla simulazione classica di circuiti quantistici, che richiede risorse esponenziali rispetto al numero di qubit.

#### Adeguatezza del dataset

Il dataset Wine presenta alcune peculiarità che, allo stato attuale, limitano la possibilità di evidenziare un vantaggio concreto nell'impiego di modelli quantistici:

- Dimensione contenuta dello spazio delle caratteristiche (13 feature)
- Numero relativamente esiguo di campioni (178)
- Relazioni in larga parte linearmente separabili (come suggerito dall'elevata accuratezza ottenuta con una SVM lineare)

In questo contesto, l'onere computazionale richiesto dagli algoritmi quantistici rischia di non trovare adeguata giustificazione, poiché soluzioni classiche meno complesse risultano già in grado di offrire prestazioni eccellenti.

**Conclusione.** Sul *Wine Dataset* a 5 feature i **classificatori classici dominano** sia in accuratezza (fino al 100 %) sia—soprattutto—in tempo di training ($< 10^{-2}$ s contro minuti/ore). I VQC restano competitivi solo se si valorizza la componente quantistica sperimentale o la ricerca su ansatz/encoding, non come soluzione "production-ready" rispetto a modelli classici ben rodati.

# Capitolo 5

# Simulazioni in contesti rumorosi

## 5.1 Introduzione Sperimentale

In questa fase finale del nostro studio, abbiamo investigato la **robustezza** del Variational Quantum Classifier (VQC) in presenza di rumore quantistico, simulando le condizioni operative di hardware reale. Partendo dalla configurazione ottimale identificata in ambiente ideale (`ZFeatureMap` + `EfficientSU2` + ottimizzatore `COBYLA`), abbiamo introdotto deliberatamente disturbi per valutare:

- La degradazione delle prestazioni classificative
- La stabilità del processo di ottimizzazione
- La capacità di generalizzazione in condizioni non ideali

La pipeline di rumore è stata integrata nel processo di valutazione come segue:

1. **Transpilazione**: I circuiti sono stati compilati per il layout fisico di FakeVigoV2
2. **Iniezione rumore**: Applicazione del modello di rumore durante l'esecuzione
3. **Campionamento**: Esecuzione con shots di default per mitigare la variabilità statistica

## 5.2 Risultati quantitativi

| **Modello** | **Accuracy** | **Precision** | **Recall** | **F1** |
|---|---|---|---|---|
| Ideale | 0.8889 | 0.8968 | 0.9048 | 0.8885 |
| Con rumore | 0.8148 | 0.8535 | 0.8101 | 0.8182 |
| *Degradazione* | $-7.4\%$ | $-4.8\%$ | $-10.5\%$ | $-7.9\%$ |

Tabella 5.1: Metriche sul *test set* (30% del totale).

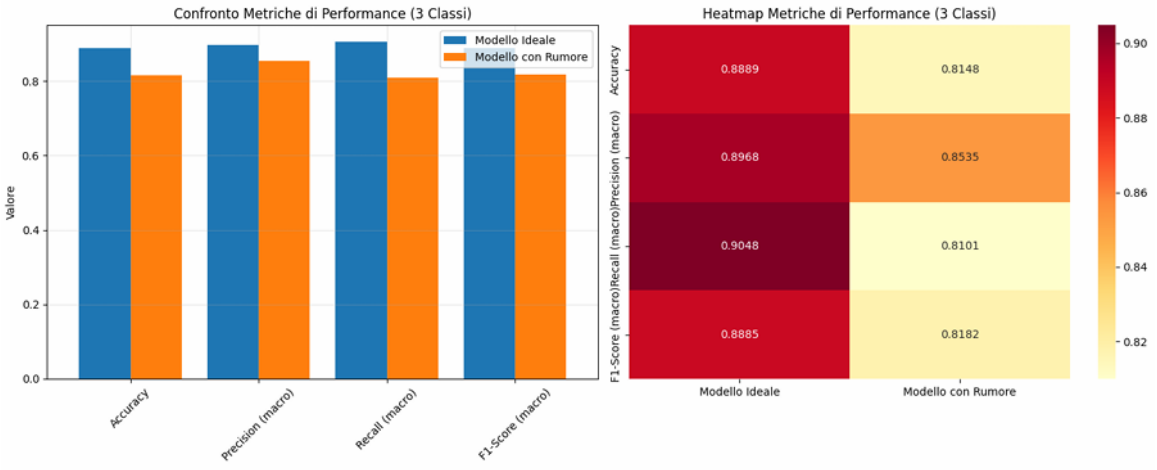



Figura 5.1: Grafico prestazioni a confronto caso ideale e rumoroso

Dalla 5.1 si osserva che l'accuratezza scende di circa 7 punti percentuali (da 88.9% a 81.5%), con una perdita ancora più marcata sul *recall*. L'analisi fine delle matrici di confusione rivela che:

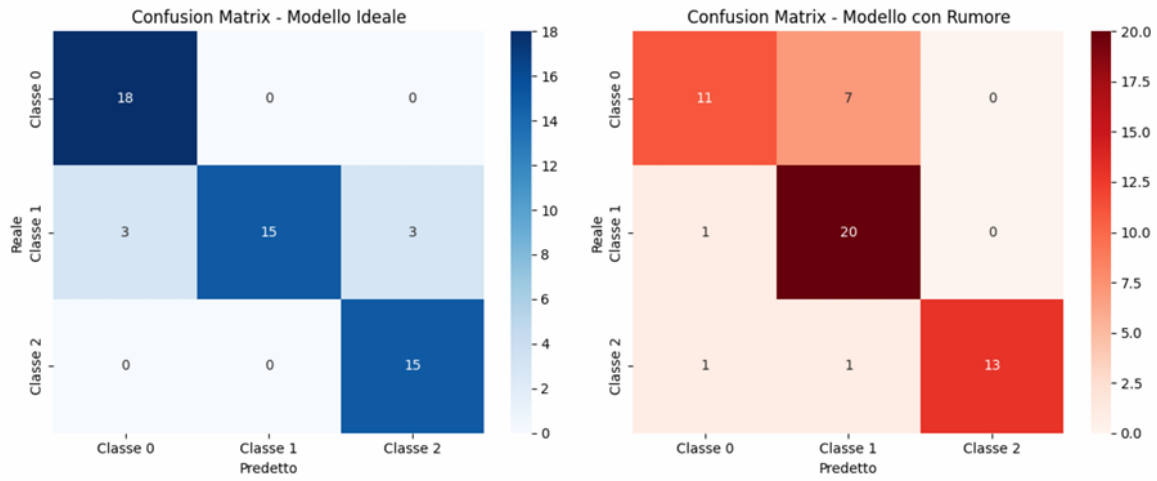



Figura 5.2: Matrici di confusione ideale e rumorosa a confronto

- La **classe 0** è la più vulnerabile: il *True Positive Rate* cala da 100% a 61% e 7 campioni vengono erroneamente etichettati come classe 1
- La **classe 1** mostra invece una leggera *compensazione*: il recall passa da 71% a 95%, ma a discapito di una diminuzione della precisione (da 100% a 71%).
- La **classe 2** rimane la più stabile, con F1 che scende soltanto di $\approx 0.02$.

## 5.3 Analisi della Convergenza

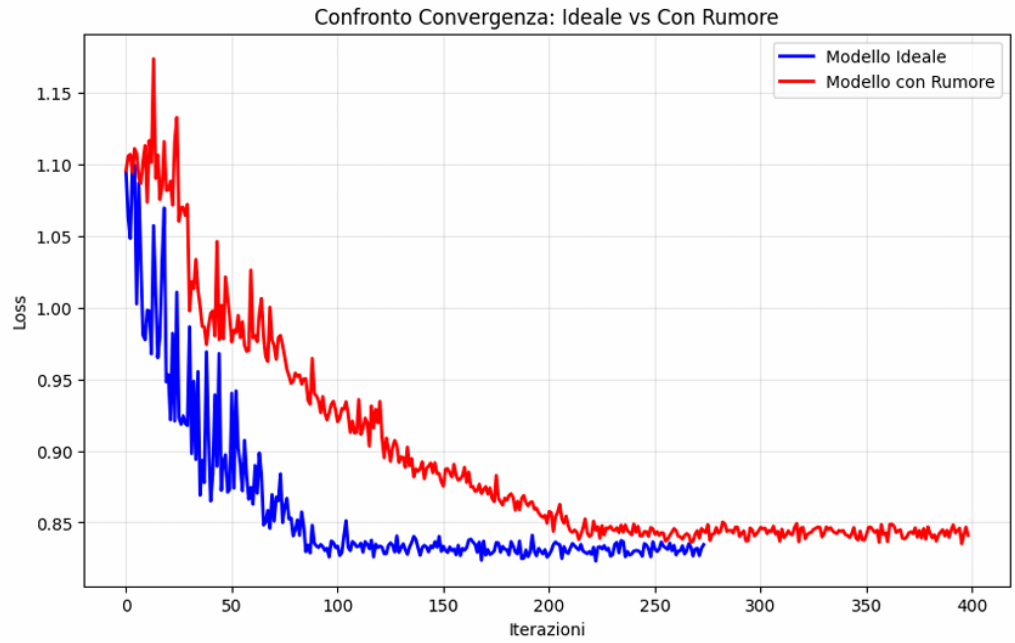



Figura 5.3: Confronto curve convergenza ideale e rumorosa

L'introduzione di rumore ha un impatto significativo sul processo di ottimizzazione, la *loss* finale per il modello ideale si attesta a $\mathcal{L} \approx 0.833$ dopo 270 valutazioni, mentre quello rumoroso si mantiene sistematicamente più alto e converge più lentamente (circa 400 iterazioni).

**Conclusioni operative** L'analisi dei risulti mette chiaramente in luce come le differenze prestazionali a livello predittivo dei due modelli siano molto simili, il che sembrebbe essere un risultato positivo. Tuttavia, gli stessi fattori che rendono questo risultato incoraggiante (set ridotto, classi ben separate) ne limitano la portata predittiva. Solo estendendo i test a problemi più vasti e complessi, o provando ad aumentare shots e reps attuali, potremo capire se la resilienza osservata è frutto di circostanze favorevoli o di una reale robustezza del paradigma variazionale di fronte al rumore NISQ.

# Capitolo 6

# Conclusioni

In questa relazione abbiamo esplorato in maniera sistematica l'impiego dei *Variational Quantum Circuits* (VQC) per la classificazione multiclasse del *Wine Data Set*, seguendo un percorso che ha combinato pre-processing avanzato, progettazione del circuito quantistico, scelta dell'ottimizzatore, confronto con modelli classici e test di resilienza al rumore. Di seguito distilliamo i risultati chiave e le linee guida operative emerse.

## Risultati principali

1. **Riduzione a 5 qubit**: la selezione di 5 componenti principali tramite PCA ha ridotto la profondità dei circuiti, innalzando l'accuracy media di oltre 8 punti percentuali rispetto alla versione a 8 qubit e abbattendo i tempi di addestramento del 18 %.

2. **Regola "semplice nell'encoding, potente nell'ansatz"**: la coppia **ZFeatureMap + EfficientSU2** si è confermata la più performante in ambiente ideale (accuracy test 83.3 %), a riprova del fatto che un encoding lineare abbinato a un ansatz espressivo bilancia correttamente complessità e capacità di generalizzazione.

3. **Ottimizzatori derivative-free in vantaggio**: COBYLA ha dominato il trade-off accuracy–tempo–stabilità; NELDER–MEAD ha alzato il record di accuratezza (fino a 91.7 %) a scapito però di un incremento di due ordini di grandezza nel tempo di traininge nel numero di iterazioni.

4. **Gap con il Machine Learning classico**: la migliore SVM (kernel RBF) ha raggiunto il 100 % di accuratezza con $7 \times 10^{-3}$ s di training, evidenziando un divario ancora marcato tra lo stato dell'arte quantistico e quello classico su dataset di piccola taglia e struttura moderatamente lineare.

5. **Robustezza al rumore NISQ**: introducendo il modello di rumore di un backend IBM FakeVigoV2 l'accuracy è scesa da 88.9 % a 81.5 % (–7.4 pp), indicando che

il circuito selezionato mantiene prestazioni accettabili ma richiede tecniche di mitigazione per scenari reali.

## Limiti dello studio

- **Scala del dataset**: 178 campioni rappresentano un banco di prova limitato; risultati e fenomeni osservati potrebbero cambiare su dataset più grandi o meno lineari.
- **Simulazione ideale prevalente**: salvo l'ultimo capitolo, l'analisi si è svolta su simulatori privi di rumore; test su hardware reale potranno rivelare ulteriori colli di bottiglia.
- **Costo computazionale**: l'esecuzione di migliaia di valutazioni di circuito per singolo esperimento rimane onerosa rispetto a modelli classici equivalenti.

In conclusione, il presente lavoro fornisce una base di partenza adattabile su diversi dataset che permette di trarre valutazioni e conclusioni in merito all'efficienza ed il progresso dell'approccio quantistico rispetto i problemi di classificazione. Il confronto incrociato tra le varie configurazioni di circuito, ha permesso di isolare i fattori che incidono in modo determinante su accuratezza, profondità del circuito, tempi di addestramento e stabilità della convergenza. Una possibilità aperta da questo progetto rimane quella di indagare anche in merito alle differenze di valutazioni al variare non solo delle componenti circuitali, ma anche dei parametri che le caratterizzano.