POLITECNICO DI BARI

Dipartimento di Ingegneria Elettrica e dell'Informazione – DEI

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Quantum Machine Learning For Classification Using VQC And QSVM

Professor:
Prof. Floriano Scioscia

Student:
Pietro SPALLUTO

Anno Accademico 2022-2023

Contents

List of Figures

List of Tables

List of Acronyms

COBYLA Constrained Optimization By Linear Approximation

GD Gradient Descent

KL Kullback–Leibler

NM Nelder-Mead

PCA Principal Component Analysis

PQC Parameterized Quantum Circuit

QML Quantum Machine Learning

QN-SPSA Quantum Natural Simultaneous Perturbation Stochastic Approximation

QNN Quantum Neural Network

QSVM Quantum Support Vector Machine

ROC Receiving Operating Characteristic

SPSA Simultaneous Perturbation Stochastic Approximation

SVM Support Vector Machine

VQC Variational Quantum Circuit

Introduction

A quantum computer is a device which leverages quantum phenomena to perform computation. Entaglement and superposition allow a quantum computer to such computations, that are hard for a classical device, and quantum bits, or *qubits*, are used in place of a classical bit [1]. Quantum algorithms, thanks to quantum mechanics properties, are very efficient, even more than their classical counterpart, but a quantum speedup has not yet been proven, since a new classical algorithm could solve the same problem in less time [2]. A natural consequence of quantum computing is Quantum Machine Learning (QML), using quantum devices to execute faster and more complex algorithms, to accelerate data analysis. Such approach faces some challenges, since quantum computers have limited computational capabilities due to noise and classical data must be encoded into quantum states in order to be used. Both problem are active area of research [3]. Another peculiarity of quantum computation is reversibility, the ability of uniquely define the input from the output, and all operators, or gates, in a quantum circuit (except for measurement) must be reversible. Performing irreversible computations brings to a loss of information and therefore a measurement [1]. Near term QML algorithms need to perform a measurement to optimize the cost function, since the optimization is done on classical hardware, loosing reversibility [4].

This work focuses on a CQ approach, namely using classical data (text, images, tabular data) with a quantum algorithm on a quantum hardware. This approach is motivated by the volume of classical data that needs to be elaborated and the demand for an energy efficient but fast and accurate algorithm. This kind of approach is negatively influenced by the input and output problem, that is giving in input classical data and receiving classical data from a quantum device. Those two issues are bottlenecks for QML algorithms [5] and some embedding method (from classical to quantum data) are being studied to reduce time consumption. The work is structured as follows:

- Chapter 1 introduces the problem of processing classical data with quantum methods: how data is encoded, how quantum algorithms (Variational Quantum Circuit (VQC) and Quantum Support Vector Machine (QSVM)) work and how the classical optimization is done;
- Chapter 2 compares a classical classification algorithm (SVM) and the two quantum classification algorithms explained in the previous chapter.

Chapter 1

Processing Classical Data with Quantum Algorithms

This chapter will explain some methods used to process classical data, such as tabular data, images or text, with quantum algorithms, an approach called CQ (for classical data-quantum hardware) technology. This approach exploits quantum superposition and entangling, the first to simultaneously compute each output value for each input (quantum parallelism) and the second to represent complex relationships which are difficult to simulate on a classical computer [6]. In this case a hybrid loop, as shown in Figure 1.1, is necessary, in which classical data is encoded into quantum states, as explained in Section 1.1, then the function is evaluated by means of a quantum circuit (Section 1.2) or a quantum kernel (Section 1.3). In the last step, the quantum state is measured and passed to a classical device with the aim of optimizing the parameters of the circuit (Section 1.4).

1.1 Data Encoding

Encoding is a fundamental step when dealing with classical data and quantum machine learning models. Unlike classical machine learning, the loading process is a challenge in QML since it affects the computational power and the performance of the algorithm. A classical dataset $\mathcal{X}$ with M samples and N features can be written as

$$\mathcal{X} = \{x^{(1)}, ..., x^{(M)}\} \tag{1.1}$$

where each sample $x^{(m)}$ is a vector of size N for $i = 1, ..., M$. Various encoding, or embedding, techniques can be used to represent this dataset as a n-qubit quantum state $\mathcal{D}_n$. Such techniques must embed data into a Hilbert space H in a way that the metric on the Hilbert space reproduces the unknown metric of the original data [7]. Furthermore, in [8] two properties of data encoding are defined:

- **learnability**: different encodings lead to different decision boundaries, so the learnability is the ability of the classifier to predict correct labels regardless of noise;

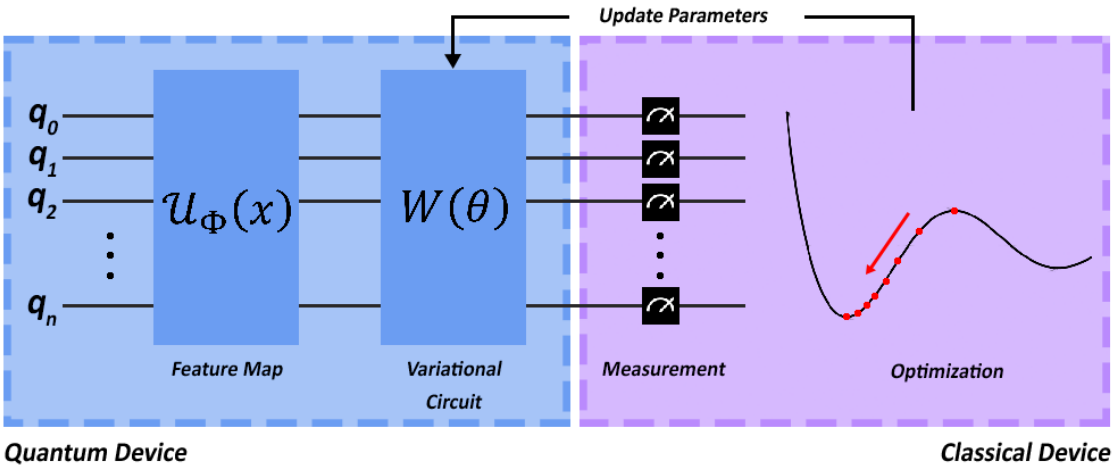


Figure 1.1: Visual representation of a quantum-classical loop. The feature map encodes the point $\mathbf{x}$, which represents a single data point. The quantum state is fed to the variational circuit and the measured result is passed to a classical device to optimize the parameters $\boldsymbol{\theta}$ in a loop.

- **robustness**: is the ability of the classifier to predict the same label (with no regard of correctness, so the label can be wrong) with and without noise. This means that the classifier should have the same behaviour in presence of noise.

Learnability and robustness give rise to a trade-off, since more complex feature maps imply higher learnability (they can find more relationships in the data) and lower robustness (a more complex circuit has more noise).

1.1.1 Basis Encoding

Basis encoding is the simplest encoding and associates a n-bit string to a computational basis state of a n-qubit system. The dataset can be represented as superposition of computational basis states

$$|\mathcal{X}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left|x^{(m)}\right\rangle \tag{1.2}$$

where $x^{(m)} = (b_1, ..., b_N)$ is directly mapped into $\left|x^{(m)}\right\rangle = |b_1, ..., b_N\rangle$. This specific encoding easily requires a large number of qubits, hence it is not suitable for most applications.

1.1.2 Amplitude Encoding

With this kind of embedding, data is encoded into the amplitudes of a quantum state. In other words a data point x is represented as the amplitudes of a quantum

state $|\psi_n\rangle$

$$|\psi_n\rangle = \sum_{i=1}^{N} x_i |i\rangle \tag{1.3}$$

where $N = 2^n$ is the size of the data point, n is the number of qubits in the system and $|i\rangle$ is the i^{th} computational basis state. The amplitude embedding of a dataset can be found by concatenating all the M samples into one amplitude vector of size $N \times M$

$$\alpha = A_{\text{norm}}(x_1^{(1)}, ..., x_N^{(1)}, ..., x_1^{(M)}, ..., x_N^{(M)}) \tag{1.4}$$

where A_{norm} is a normalization constant such that $|\alpha|^2 = 1$. The dataset can be represented as

$$|\mathcal{X}\rangle = \sum_{i=1}^{N} \alpha_i |i\rangle \tag{1.5}$$

where α_i are elements of the amplitude vector. This kind of embedding requires $n \geq \log_2(NM)$ qubits since there are 2^n amplitudes. The number of qubits is smaller but algorithms must operate on the amplitudes of a quantum state and methods to prepare and measure the quantum states tend not to be efficient.

1.1.3 Angle Encoding

This encoding embeds N features into rotation angles of n qubits ($N \leq n$). A data point can be encoded as

$$|x\rangle = \bigotimes_{i=1}^{N} \cos(x_i) |0\rangle + \sin(x_i) |1\rangle \tag{1.6}$$

The approach requires at most N qubits and a constant depth circuit (usually the size of the circuit grows exponentially with the number of the qubits) but data points are encoded one at a time. A generalization of this encoding is the **dense angle encoding**, where a data point is encoded as

$$|x\rangle = \bigotimes_{i=1}^{\lceil N/2 \rceil} \cos(\pi x_{2i-1}) |0\rangle + e^{2\pi i x_{2i}} \sin(\pi x_{2i-1}) |1\rangle \tag{1.7}$$

Those encoding use sinusoids and exponentials but they can be generalized to arbitrary functions [8].

1.1.4 General Encoding

Similarly to the angle encoding, this encoding embeds N features as rotation on N parameterized gates on n qubits, where $n \leq N$, one data point at a time. A

general map is

$$|x\rangle = \bigotimes_{i=1}^{\lceil N/2 \rceil} f_i(x_{2i-1}, x_{2i}) |0\rangle + g_i(x_{2i-1}, x_{2i}) |1\rangle \tag{1.8}$$

where $f, g : \mathbb{R} \times \mathbb{R} \to \mathbb{C}$ and $|f_i|^2 + |g_i|^2 = 1 \ \forall i$. Those transformation are called *quantum feature maps* and they are a popular way to encode classical data into a quantum system. Choosing the right feature map for a specific dataset is an open challenge but there are some methods for preparing a feature map such as genetic algorithms [9] [10], continuous time evolution [11] or a trainable kernel capable of selecting the best weights [7], a technique called quantum kernel alignment. Symmetries in data can be useful to make a feature map that can encode data points in the right way.

1.2 VQC

Encoded data, in the form of quantum state, can be passed to the next part of the circuit, called *ansatz* or *variational quantum circuit*, as shown in the hybrid loop in Figure 1.1. The ansatz is a set of gates with discrete or continuous parameters that can be optimized through a training process to solve the following task

$$\boldsymbol{\theta}^* = \underset{\theta}{\text{argmin}} C(\boldsymbol{\theta}) \tag{1.9}$$

where $\boldsymbol{\theta}$ is the parameter vector containing the parameters of the variational circuit, those parameters represent the angles of the gates, and $C(\boldsymbol{\theta})$ is the cost function [6]. According to [3], this function should meet some criteria:

- **faithfulness**: its minimum correspond to the solution of the problem;
- **efficient to estimate**: the estimation of the function should be computationally efficient on a quantum computer and possibly on a classical computer when performing post-processing. Additionally this function should be not efficiently computable with a classical computer in order to have a quantum advantage;
- **operationally meaningful**: a smaller cost implies a better solution quality;
- **trainable**: the parameters should be efficiently optimized.

The structure of an ansatz usually depends on the specific problem, in this case is a problem-inspired ansatz. Some ansatz are problem-agnostic, so they are used when no information is available. A variational circuit is unitary and it is applied to the input quantum states. The circuit $U(\boldsymbol{\theta})$ is the product of L sequentially applied unitaries

$$U(\boldsymbol{\theta}) = U_L(\boldsymbol{\theta}_1)...U_1(\boldsymbol{\theta}_1) \tag{1.10}$$

with

$$U_l(\boldsymbol{\theta}_l) = \prod_m e^{-i\theta_m H_m} W_m \tag{1.11}$$

where W_m is an unparameterized unitary and H_m is a Hermitian operator (that is the observables, namely, any physical quantity that can be measured). In literature there are different ansatze starting from problem specific to more general ones [3].

Those circuits, preceded by the embedding circuit, form a Quantum Neural Network (QNN). In a quantum computer a QNN has the following components:

- **feature maps**: is the embedding circuit that works as input layer;
- **ansatze**: are the hidden layers of the network, each layer is called a *circuit template*;
- **parameters**: are the weights of the network and are the *variational angles* used to rotate a qubit.

A Parameterized Quantum Circuit (PQC) can be evaluated by the following three descriptors defined in [12] and [13].

1.2.1 Expressibility

Expressibility is the ability to reach the full Hilbert space in a uniform way (i.e. uniformly cover the Bloch sphere [14]). To compute this value, the true distribution of the fidelities is compared to the distribution of fidelities from the ensemble of Haar random states. The fidelity is defined as the overlap of two quantum states

$$F = |\langle \psi_\theta | \psi_\phi \rangle|^2 \tag{1.12}$$

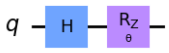
The first distribution, $\hat{P}_{PQC}(F;\Theta)$, is computed by repeatedly sampling two sets of parameters and taking the fidelity of the resulting quantum states. The second one, the ensemble of random states, can be found in an analytical way:

$$P_{Haar} = (N-1)(1-F)^{N-2} \tag{1.13}$$

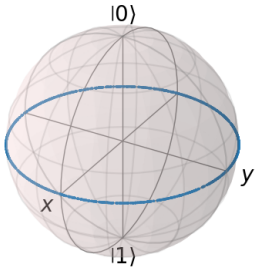
where N is the dimension of the Hilbert space. The expressibility is the Kullback–Leibler (KL) divergence between the two distributions

$$Expr = D_{\mathrm{KL}}(\widehat{P}_{PQC}(F;\Theta)||P_{Haar}(F)) \tag{1.14}$$

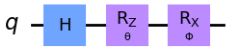
When the difference between $\hat{P}_{PQC}(F;\Theta)$ and P_{Haar} is small, so the KL divergence is small, the circuit can explore the Hilbert space in a better way. Figure 1.2 shows a simple example of how different circuits, having different expressibility, can represent the same random points using the Bloch Sphere.



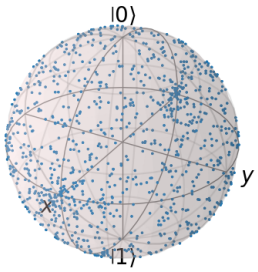
(a) One parameter circuit.



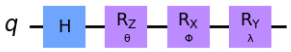
(b) Bloch Sphere.



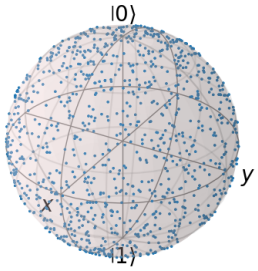
(c) Two parameters circuit.



(d) Bloch Sphere.



(e) Three parameters circuit.



(f) Bloch Sphere.

Figure 1.2: In (a) points are distributed about the equator of the sphere (b), this circuit has the lowest expressibility. The second circuit (c) has two parameters and in this case data points are distributed almost uniformly except for X and $-X$ in which there is a greater concentration (d). The last circuit (e) is the one with the greatest expressibility since points are more uniformly distributed (f).

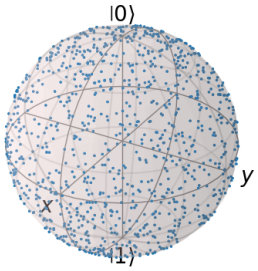
1.2.2 Entangling Capability

The entangling capability allows a circuit to capture non trivial correlation in the data. The Meyer-Wallach entangling measure [15], denoted with Q, is used due to

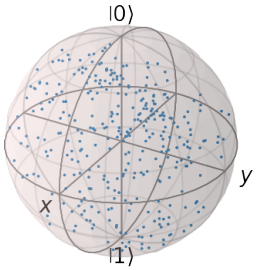
its scalability and ease of computation. The entangling capability is

$$Ent = \frac{1}{|S|} \sum_{\theta_i \in S} Q(|\psi_{\theta_i}\rangle) \tag{1.15}$$

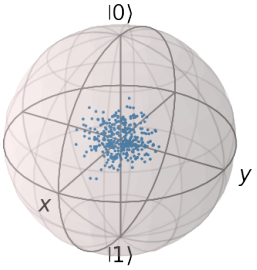
where S is the set of sampled parameter vectors. A circuit that produces always highly entangled quantum states as a value of Ent close to 1. Figure 1.3 shows how quantum states are closer to the center of the sphere when the value of entangling capability is higher.



(a) Low entangling capability.



(b) Medium entangling capability.



(c) High entangling capability.

Figure 1.3: Figure (a) represent a Bloch sphere in which there are only pure states, so entangling capability of 0. In Figures (b) and (c) the entangling capability increases, a high entangling capability value bring the mixed states closer to the center of the sphere.

1.2.3 Accuracy

Is the number of correctly classified samples over the total number of samples. In [13] a strong correlation between accuracy and expressibility is measured, at the same time the correlation between accuracy and entangling capability is weak. This means that a good expressibility increases the accuracy but the entangling capability does not influence it a lot.

1.3 QSVM

Support Vector Machines (SVMs) use kernel functions to map the input dataset into a higher dimensional feature space. In classification tasks a hyperplane is found capable of separating points of different classes with the largest possible distance [16]. The kernel is a measure of similarity and, for quantum computers, feature maps are used to generate quantum kernels. Quantum feature maps apply the transformation $\mathbf{x} \to |\phi(\mathbf{x})\rangle$, transforming the input vector in a quantum state, and the resulting quantum kernel is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_j)^\dagger \phi(\mathbf{x}_i) \tag{1.16}$$

where $\phi(\mathbf{x}_j)^\dagger$ represents the complex conjugate transpose of $\phi(\mathbf{x}_j)$. Like a classical kernel, the value a quantum kernel is large if $\mathbf{x}_i$ and $\mathbf{x}_j$ are close. A QSVM provide an advantage with respect to classical SVM if the feature map, and consequently the quantum kernel, are hard to simulate on a classical device [17]. Figure 1.4 shows some ad-hoc data being encoded into 2 qubits, it is clearly visible that data can be separated by a hyperplane.

1.4 Parameters Optimization

The last step, the optimization process, is done entirely on a classical device. The result, measured on a quantum computer, is passed to a classical one which computes the loss and tries to minimize it by means of classical optimization algorithms. Optimization algorithms use **numerical methods** or **gradient-based methods**.

1.4.1 Gradient-based algorithms

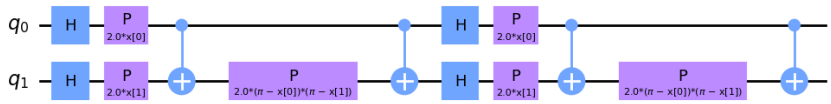
Gradient-based optimization methods are iterative algorithms that, starting from a initial value, try to iteratively approximate a solution by minimizing the cost function at each step. Those methods are faster than the more precise numerical methods but they suffer from vanishing gradient if the complexity of the quantum circuit, depth (layers executed in parallel), number of qubits (width) and number of gates, increases.

Gradient Descent (GD)

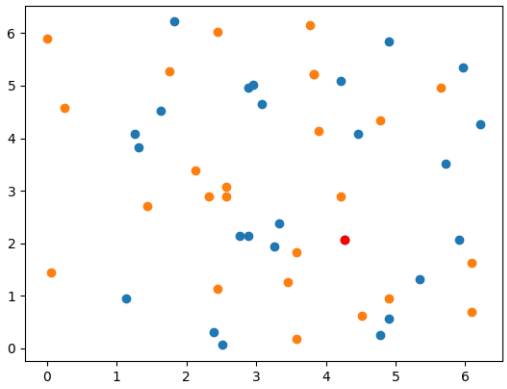
GD, or steepest descent, is a simple iterative method in which the parameter vector at the step $k+1$ is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \gamma_k \nabla f(\boldsymbol{\theta}_k) \tag{1.17}$$

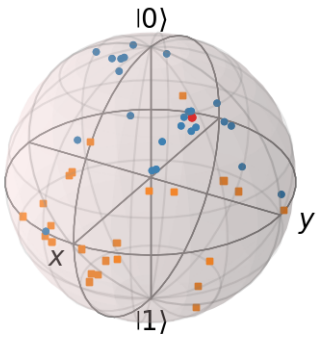
where f is the objective function and γ_k is the **learning rate** at each step. At each update the parameters move toward the point in which f decreases faster, ending in a local or global optimum [18].



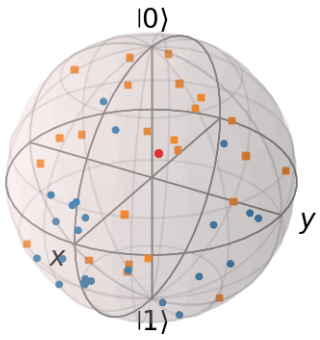
(a) Quantum kernel.



(b) Classical data points with 2 features.



(c) Qubit encoding the first feature.



(d) Qubit encoding the second feature.

Figure 1.4: Classical data points (b) cannot be divided by a hyperplane but the quantum kernel (a) encodes data into the two qubits in (c) and (d) allowing them to be divided by the hyperplane computed by a QSVM with an accuracy score of 1.

Adam

It is an extension of stochastic GD in which the learning rate is adapted at each step. The algorithm updates exponential moving averages of the gradient (m_k) and the squared gradient (v_k) where the hyperparameters β_1 and $\beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages [19]. The algorithm includes the

momentum strategy, useful to avoid oscillation and to have a faster convergence. The update rule has more steps

- update biased first and second moment estimate:

$$m_{k+1} = \beta_1 m_k + (1 - \beta_1)\nabla f(\boldsymbol{\theta}_k)$$

$$v_{k+1} = \beta_2 v_k + (1 - \beta_2)(\nabla f(\boldsymbol{\theta}_k))^2$$

- correct the bias in the moments

$$\hat{m}_k = \frac{m_k}{1 - \beta_1^{k+1}}$$

$$\hat{v}_k = \frac{v_k}{1 - \beta_2^{k+1}}$$

- update parameters

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1}} + \epsilon}$$

where α is the learning rate and ϵ is used to avoid a zero division [20].

SPSA and QN-SPSA

Both methods are based on finite difference to approximate gradients

$$\nabla f(\boldsymbol{\theta}) = \frac{f(\boldsymbol{\theta} + \epsilon) - f(\boldsymbol{\theta} - \epsilon)}{2\epsilon} \tag{1.18}$$

where ϵ is a small distance from the parameter vector. The problem is the number of evaluations to compute the gradient. In fact, with N parameters, $2N$ evaluation are necessary each time. Simultaneous Perturbation Stochastic Approximation (SPSA), differently from classical finite differences, applies perturbation simultaneously to all elements of the parameter vector. On average this works equally well, the only difference is that SPSA will not follow a smooth line when searching for the minimum [21]. The update rule is the same as a simple GD algorithm:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla f(\boldsymbol{\theta}_k) \tag{1.19}$$

Quantum Natural Simultaneous Perturbation Stochastic Approximation (QN-SPSA) is similar but it uses natural gradients. Natural gradients are useful when the loss landscape do not varies at the same rate for each parameter. Differently from standard gradient, that uses the Euclidean distance $d = \|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|_2$ to update the parameters, natural gradients use a distance that depends on the model: $d = \|\langle\psi(\boldsymbol{\theta}_k)|\psi(\boldsymbol{\theta}_{k+1})\rangle\|^2$, which is the fidelity. The update rule becomes

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k g^{-1}(\boldsymbol{\theta}_k)\nabla f(\boldsymbol{\theta}_k) \tag{1.20}$$

where g^{-1} is the quantum Fisher information [22]. QN-SPSA then, not only approximates the gradient, but it also approximates the quantum Fisher information matrix using different methods depending on the complexity of the unitary and the structure of the available hardware [23].

1.4.2 Numerical optimization algorithms

Numerical, or direct, methods follow a sequence of operations and, in absence of rounding errors, they give an exact solution of the problem but are more expensive as the number of variables increases. The following algorithms are based on the simplex method, a popular method used in linear programming to solve equation in the canonical form

$$\begin{aligned} &\max \mathbf{c}^T\mathbf{x} \\ &\text{subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq 0 \end{aligned} \tag{1.21}$$

where $\mathbf{c} = (c_1, ..., c_n)$ are the coefficient of the objective function, $\mathbf{A}$ is a $p \times n$ matrix containing the coefficients of the p equations, $\mathbf{b} = (b_1, ..., b_p)$ are the known terms and $\mathbf{x} = (x_1, ..., x_n)$ are the variables of the problem.

COBYLA

COBYLA is a derivative-free method, that is a method that do not use the derivative to find the optimal solution. The reason behind this approach is that the objective function is not smooth and the evaluation of its derivative could be time-consuming. It works by iteratively approximating the constrained optimization problem. The process starts with an initial guess and then it makes linear approximations of the objective function. The function is evaluated to have information about the direction in which the loss is decreasing. This is done iteratively until a termination criteria is met [24].

Nelder-Mead (NM) Algorithm

It is an optimization algorithm to find the minimum of a n-dimensional function using $n + 1$ points. At each point $P_0, ..., P_n$ correspond a function value denoted with y_i. Two particular values are $y_h = \max_i(y_i)$ and $y_l = \min_i(y_i)$ respectively in the points P_h and P_l. The point $\overline{P}$ is the centroid of the points with $i \neq h$. At each stage the point P_h is replaced by a new point by means of one of the following operations:

- **reflection**: the worst vertex (P_h) is reflected through $\overline{P}$ using

  $$P^* = (1 + \alpha)\overline{P} - \alpha P_h$$

  where α is a positive constant called reflection coefficient. If $y_l > y^* > y_h$ then P_h is replace by P^*;

- **expansion**: if $y^* < y_l$ then y^* is a new minimum and P^* is expanded to P^{**} by the relation

$$P^{**} = \gamma P^* + (1 - \gamma)\overline{P}$$

  where $\gamma > 1$ is the reflection coefficient. This operation allows the simplex to stretch toward that direction;

- **contraction**: if $y^* > y_l$ for all $i \neq h$ a new P_h is defined to be either the old P_h or P^* whichever has the lower y and find

$$P^{**} = \beta P_h + (1 - \beta)\overline{P}$$

  where $0 < \beta < 1$ is the contraction coefficient. With this operation the worst vertex is moved closer to the centroid.

The algorithm stops when a criteria is met [25].

Chapter 2

Implementation and Results

In this chapter classical and quantum methods are compared. The dataset used contains 333 samples (after removing null values) divided into three classes with the following proportions: "Adelie": 44%, "Chinstrap": 20% and "Gentoo": 36%. In Section 2.1, a baseline model is built using a classical SVM to classify samples. This is useful to compare the two approaches in terms of performance. In Sections 2.2 and 2.3 quantum algorithms are used to solve the same classification problem. Section 2.4 summarizes results.

2.1 Baseline

First, a classical device is used to train a SVM on the dataset. This model is used as a baseline to evaluate the quantum algorithms. Since circuit complexity (number of qubits, number of gate and circuit depth) influences noise and training time (being a simulation of a real quantum device the advantage of exploring more states simultaneously greatly reducing computation time is not existent), a dimensionality reduction technique, Principal Component Analysis (PCA) in this case, is applied. As shown in Figure 2.1, two principal components are enough to express more than the 99% of the variance of the dataset. A simple SVM with the linear kernel

$$k(x, y) = x^T y \tag{2.1}$$

is trained on the dataset giving an accuracy of 0.88. Figure 2.2 gives more detailed information about the performance and illustrates the decision boundaries generated by the kernel. While PCA decreases the accuracy of the model, from 1 to 0.88, a reduced number of features implies less qubits. Table 2.1 shows the differet complexities of various quantum circuits used in the next sections when the features are 6 or 2.

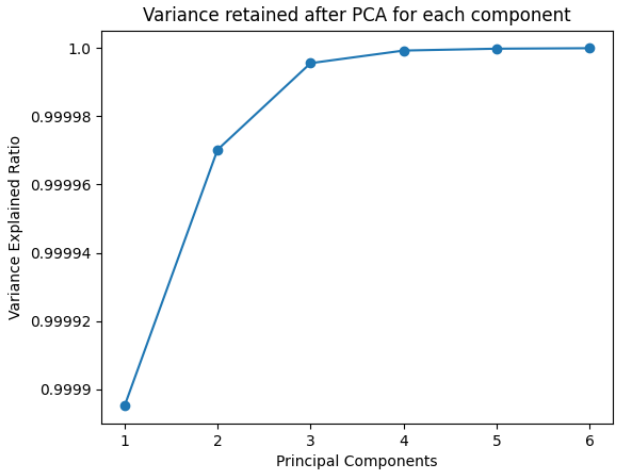


Figure 2.1: Out of six maximum components, only the first two can describe the majority of the variance.

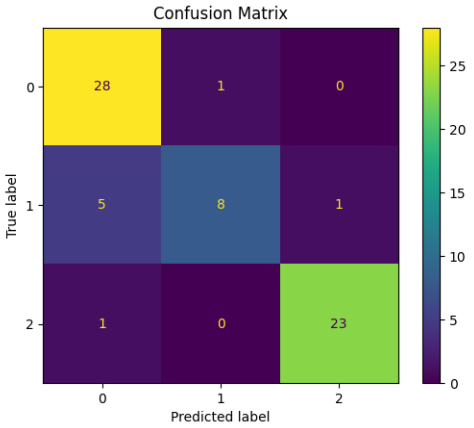
Table 2.1: Comparison between circuits used as feature map or ansatz when the dataset has 6 and 2 features. In this case the width is the same as the dimensionality of the dataset.

Circuit	Width	#parameters	#gates	Depth
ZZFeatureMap (1 repetition)	6	6	57	29
ZZFeatureMap (3 repetitions)		6	171	69
TwoLocal (rotation: RX, RZ, entanglement: CZ, 3 repetitions)		48	93	29
RealAmplitudes (3 repetitions)		24	39	13
EfficientSU2		12	30	10
ZZFeatureMap (1 repetition)	2	2	7	5
ZZFeatureMap (3 repetitions)		2	21	15
TwoLocal (rotation: RX, RZ, entanglement: CZ, 3 repetitions)		16	19	11
RealAmplitudes (3 repetitions)		8	11	7
EfficientSU2		4	9	5

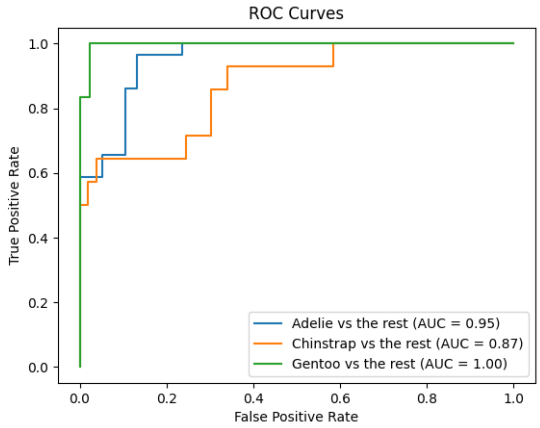
2.2 VQC Implementation

For this implementation, different combinations of the three main components of the hybrid loop (embedding feature map, ansatz and optimizer) are tested, for a total of 36 configurations (see Appendix A). Being a classification, the cross-entropy loss is used as metric for each optimization process to train the variational circuit. Figure 2.3 shows the loss curves for each configuration divided according to the optimizer used. There is no a particular correlation between the optimization methods and the losses, but COBYLA usually converges faster than other methods.

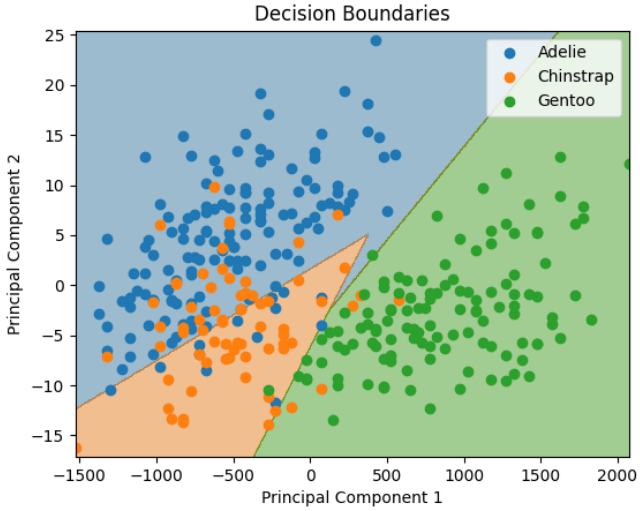
Figure 2.4 illustrates mean accuracy (number of correct predictions over number of total predictions) and training times of all configurations. As expected



(a) One-vs-rest confusion matrix.



(b) One-vs-rest Receiving Operating Characteristic (ROC) curves.

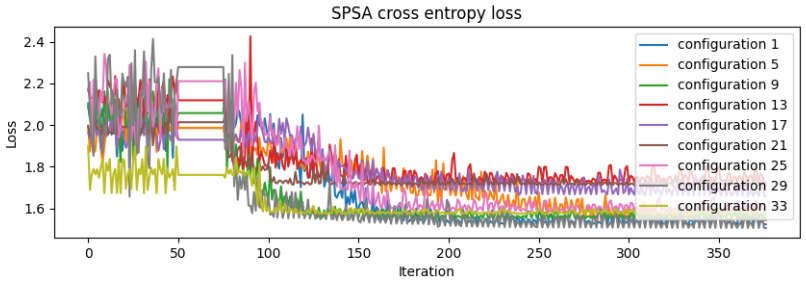


(c) Decision boundaries with ground truth points.

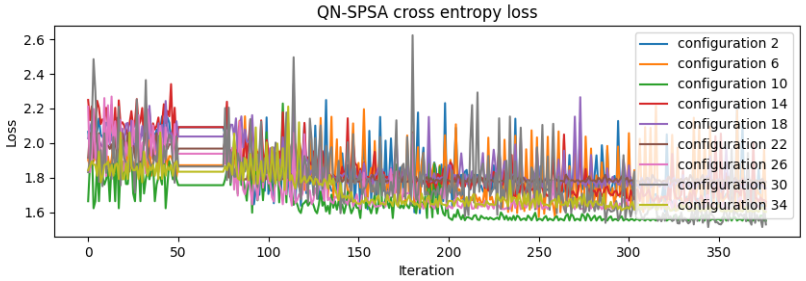
Figure 2.2: Dataset is well balanced in fact the model can correctly classify most data points of all three classes as shown in (a) and (b). In (c) the colored areas represent the decision boundaries.

from the previous plots, mean accuracy has approximately the same value for each configuration (0.396 ± 0.051 with a maximum value of 0.552 and a minimum of 0.284). On the other hand, training time heavily depends on circuit complexity. In fact, the bar chart of the training time can be divided into three sections, one for each feature map, where ansatze and optimization method remain unchanged. The same is true for the ansatze, they contain the trainable parameters so they mostly influence training time (with EfficientSU2 being faster than the other two).

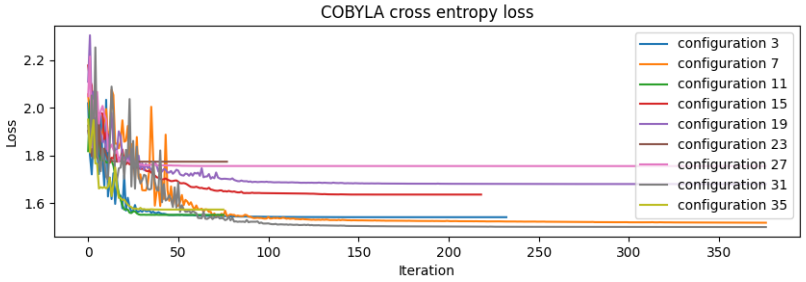
The best configuration, taking into account only the accuracy and not the training time, is the configuration 18 (see Appendix B for the complete circuit



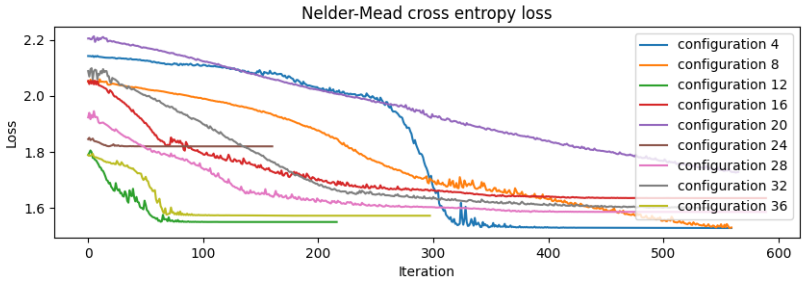
(a) SPSA losses.



(b) QN-SPSA losses.



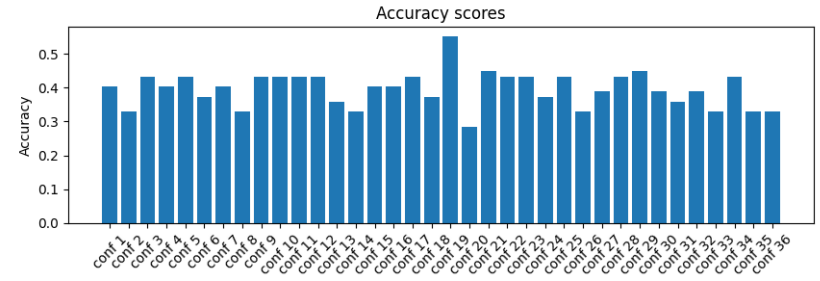
(c) COBYLA losses.



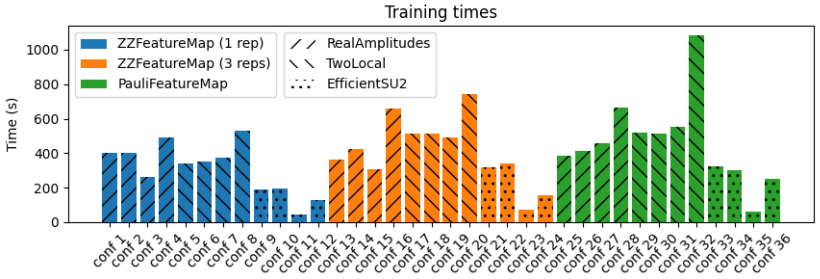
(d) NM losses.

Figure 2.3: Four different optimization methods are used to train the parameters of the variational circuit. Gradient-based methods (a) and (b) approximate the gradient of the function so they do not have a smooth line. Numerical methods in (c) and (d) usually converge faster but they take more iteration or a longer training time.

and the encoding result), which gives an accuracy score of 0.55 and is trained in 491.81s. This is worse than the value obtained by the classical SVM nor there is



(a) Accuracy scores.



(b) Training times divided for feature map and ansatz.

Figure 2.4: Feature maps, ansatze and optimization methods influence training time but they do not influence accuracy. This means that the circuit should be problem specific to improve performance.

an advantage in terms of training time.

2.3 QSVM Implementation

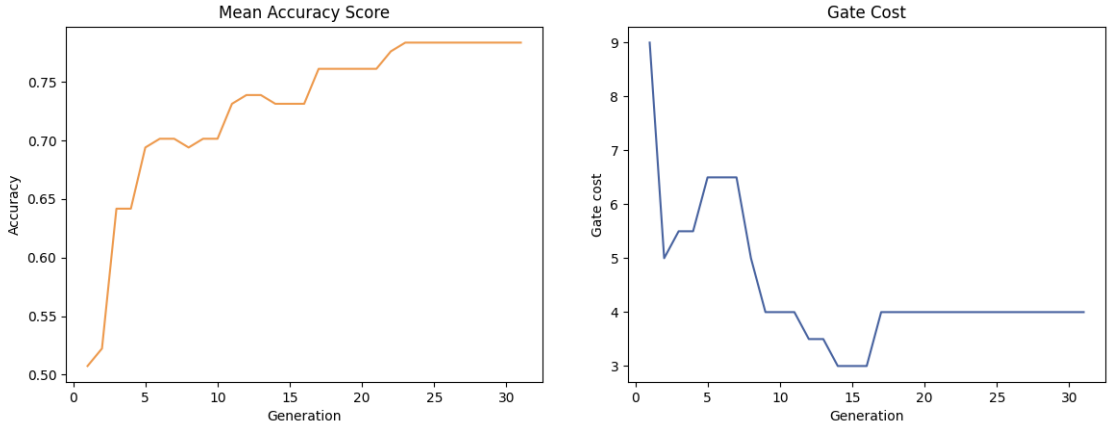
Given the poor performance of the VQC, a QSVM can be a better solution since there is no ansatz to train. The problem is to find a good data embedding, from classical to quantum, using a quantum kernel generated from a quantum feature map. A quantum feature map should be specific for the dataset and one method to find a suitable embedding is through a genetic algorithm.

In [10] a genetic algorithm is proposed in which each gate is determined by six bits: the first three determine the type of gate and the last three determine the rotation coefficient if the gate is a rotation gate. The i^{th} gate is applied to the $i \bmod (N)$ qubit or to the $i \bmod (N)$ and $i+1 \bmod (N)$ qubits if the gate is a CNOT, where N is the number of qubits. So the size of the string determining the quantum circuit is $M \times N \times 6$, where M is the maximum depth of the circuit. The gates used are rotation, Hadamard, CNOT and identity which are a universal set (except for the identity gate which has no effect on the circuit but is added if the 6 bits combination do not correspond to any of the other gates), namely they can approximate any quantum circuit [2]. A multi-objective fitness function is proposed to maximize the accuracy and simultaneously minimize gate cost

$$\text{Fitness} = \text{GateCost} + \frac{w}{\text{Accuracy}^2} \tag{2.2}$$

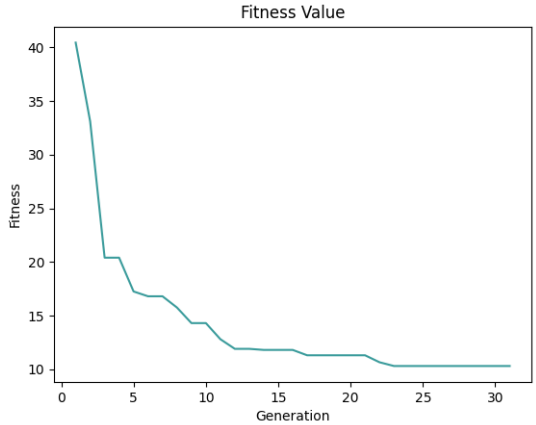
where GateCost = Rgate + 2Hgate + 5CNOTgate as proposed in [26] and w is a positive constant depending on the dataset.

To optimize the fitness function the algorithm is run for 31 generations and a population of 12 individuals. Figure 2.5 illustrates the fitness value and the two components (accuracy and gate cost) used to compute it.



(a) Mean accuracy scores of the pool of selected individuals.

(b) Mean gate cost of the pool of selected individuals.

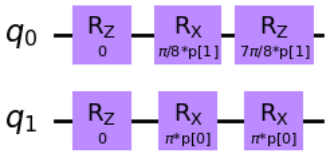


(c) Fitness value.

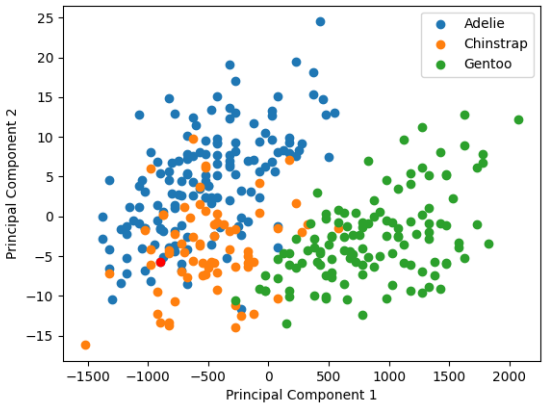
Figure 2.5: The algorithm is capable of reducing the gate complexity and increasing the accuracy at each generation until it reaches the (local) minimum. The best fitness value is 10.3 with a gate complexity of 4 and an accuracy of 0.79.

The QSVM is capable of reaching an accuracy of 0.79 with a training time of 131.4s. This is still worse than the performance of SVM, in terms of time and accuracy, but it is an improvement with respect to the previous method. Figure 2.6 shows the resulting feature map and how data points are projected into the

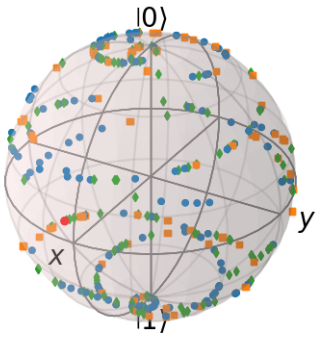
higher dimensional Hilbert space.



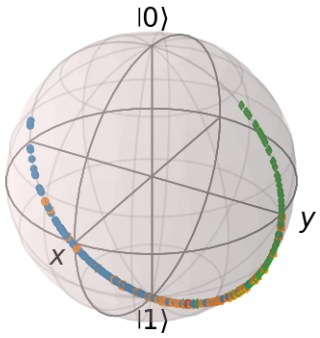
(a) Best Feature Map.



(b) Data points projected into two dimensions with PCA.



(c) Qubit 1 encoding the first feature.

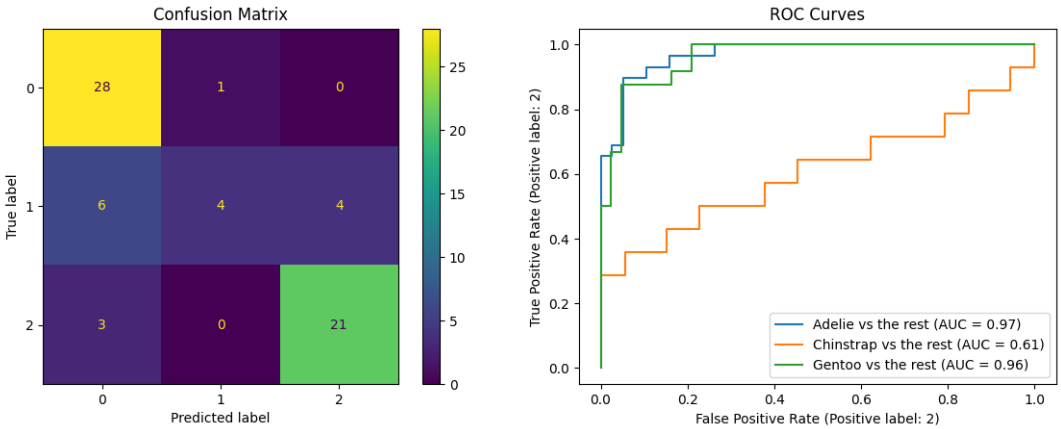


(d) Qubit 2 encoding the second feature.

Figure 2.6: The resulting feature map (a) is very simple, as expected, and do not contain Hadamard or CNOT gates which contribute the most to the total gate cost. The scatter plot in (b) shows the points divided by class and the two Bloch spheres in (c) and (d) show the points projection into the Hilbert space. Each point is on the surface of the sphere, so it is a pure state, and in the second qubit two hyperplanes can divide the points of the three classes. The red point is used to show how the projection works.

In Figure 2.7 there are the confusion matrix and the ROC curves showing a

behaviour similar to the SVM except for the second class "Chinstrap", for which the majority of predictions during the test phase are wrong.



(a) Confusion matrix.

(b) ROC curves.

Figure 2.7: The model performs well on the first and the last class with only a few misclassified samples but bad on the second class. This may be caused by the imbalanced distribution of the samples.

This new feature map can also be applied to the variational approach reducing the complexity of the circuit and improving the accuracy (see Appendix B).

2.4 Results

The classical approach provides better results compared to the two quantum algorithms. This is due to the fact that feature maps and ansatze are specific for a dataset. Testing more parameters for the genetic algorithm and using a problem-specific ansatz could give better performance in terms of accuracy. Table 2.2 summarizes the results of the three models.

Table 2.2: Comparison between the three model used in the classification problem. The custom feature map is the one found by the genetic alorithm.

Model	Accuracy	Training Time (s)
SVM (baseline)	**0.88**	**6.1**
QSVM	0.79	131.4
VQC	0.55	491.8
VQC+custom feature map	0.70	304.8

Conclusion

QML algorithms could be an actual improvement to classical methods, since the ability of quantum computers to be in multiple states at the same time and to entangle qubits provide an advantage in terms of time and can represent complex relationships difficult to model with classical hardware. The problem of finding a suitable feature map is still an open challenge but some methods have arisen able to automatically compute such a circuit. Critical points of this approach are:

- finding a feature map and an ansatz to properly encode data and discriminate data points;
- making circuits with a trade-off between robustness and expressibility in order to avoid the effects of noise and, at the same time, represent complex relationships;
- classical hardware is still necessary.

To increase classification accuracy various things can be tested:

- changing the parameters of the genetic algorithm, for example finding the right weight penalty w for the dataset or increasing the depth of the circuit;
- the optimized quantum feature map has not Hadamard gates nor CNOT gates, so the advantages of superposition and entangling are not exploited. This can be avoided by changing the fitness function;
- adding trainable parameters to the quantum feature map that can be optimized;
- finding a suitable ansatz.

Appendix A

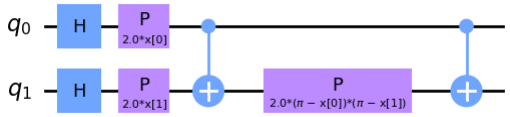
VQC configurations

Table A.1 shows the 36 configurations tested during the training process of the VQC.

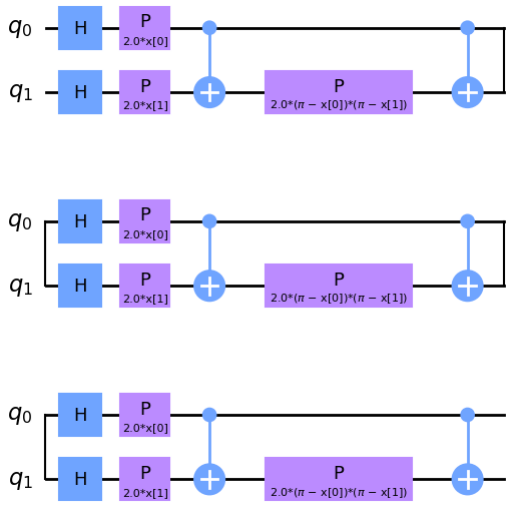
Table A.1: Configurations tested using a two qubits quantum circuit.

Configuration	Feature map	Ansatz	Optimization	#gates	#parameters	#trainable parameters	Depth
1	ZZFeatureMap (1 rep)	RealAmplitudes	SPSA	18	10	8	12
2			QN-SPSA				
3			COBYLA				
4			Nelder-Mead				
5		TwoLocal	SPSA	26	18	16	16
6			QN-SPSA				
7			COBYLA				
8			Nelder-Mead				
9		EfficientSU2	SPSA	16	6	4	10
10			QN-SPSA				
11			COBYLA				
12			Nelder-Mead				
13	ZZFeatureMap (3 rep)	RealAmplitudes	SPSA	32	10	8	22
14			QN-SPSA				
15			COBYLA				
16			Nelder-Mead				
17		TwoLocal	SPSA	40	18	16	26
18			QN-SPSA				
19			COBYLA				
20			Nelder-Mead				
21		EfficientSU2	SPSA	30	6	4	20
22			QN-SPSA				
23			COBYLA				
24			Nelder-Mead				
25	PauliFeatureMap	RealAmplitudes	SPSA	33	10	8	26
26			QN-SPSA				
27			COBYLA				
28			Nelder-Mead				
29		TwoLocal	SPSA	41	18	16	30
30			QN-SPSA				
31			COBYLA				
32			Nelder-Mead				
33		EfficientSU2	SPSA	31	6	4	24
34			QN-SPSA				
35			COBYLA				
36			Nelder-Mead				

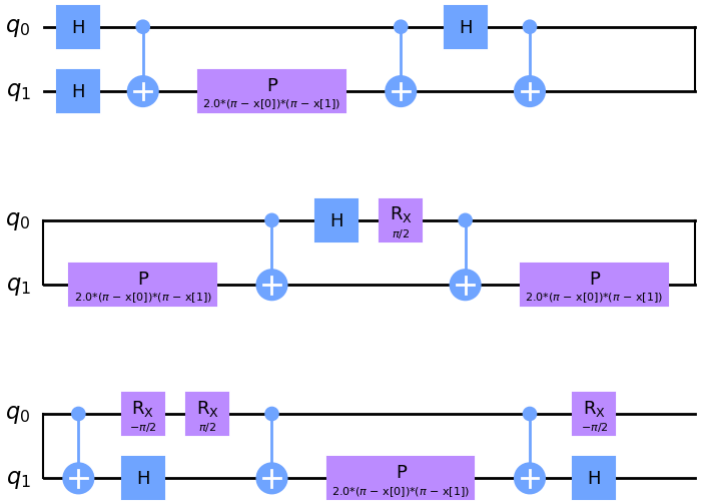
Figure A.1 shows the feature maps used for the VQC to encode data. Figure A.2 shows the three ansatze, RealAmplitudes ($Expr = 0.013$, $Ent = 0.668$), TwoLocal ($Expr = 0.012$, $Ent = 0.779$) and EfficientSU2 ($Expr = 0.014$, $Ent = 0.743$), used as variational circuit.



(a) ZZFeatureMap.

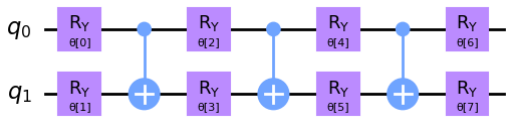


(b) ZZFeatureMap (3 repetitions).

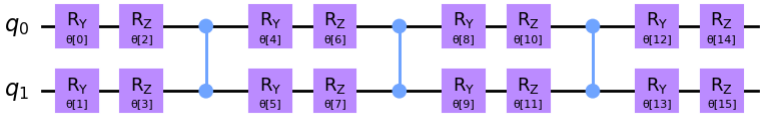


(c) Custom PauliFeatureMap.

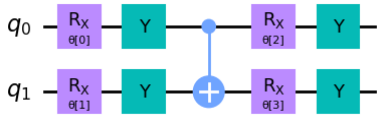
Figure A.1: Feature maps used to encode data. They are not capable of correctly represent data in the Hilbert space producing a poor embedding.



(a) RealAmplitudes (3 repetitions).



(b) TwoLocal (3 repetitions).



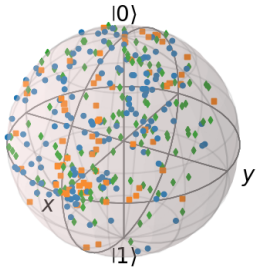
(c) EfficientSU2.

Figure A.2: Circuits with trainable parameters used for the classification task. They all have a high entangling capability and a low expressibility. Hence, they are able to produce higly entangled quantum states and to explore the full Hilbert space.

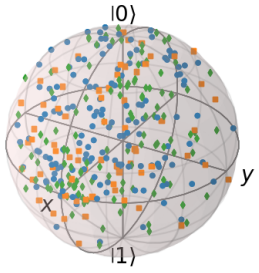
Appendix B

VQC Circuits and Quantum Encodings

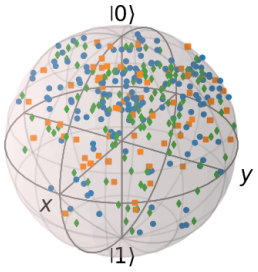
Figure B.1 shows how data point are embed into the Hilbert space after the encoding and after the training. Due to entangling there are also mixed state (inside the Bloch sphere). Figure B.2 shows the feature map, the ansatz and their composition. This circuit reaches an accuracy of 0.55.



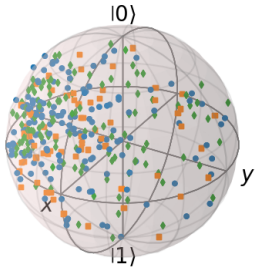
(a) First qubit.



(b) Second qubit.

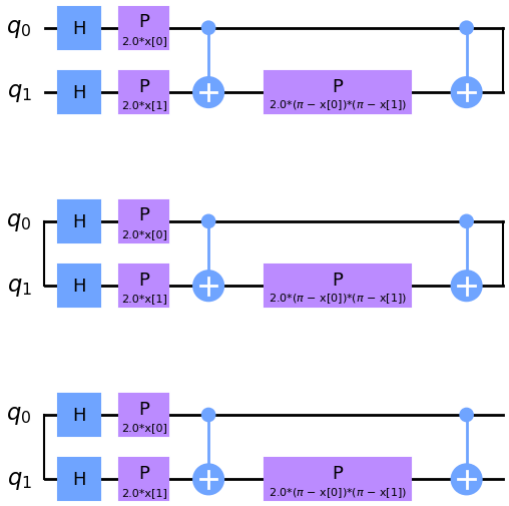


(c) First qubit after training.

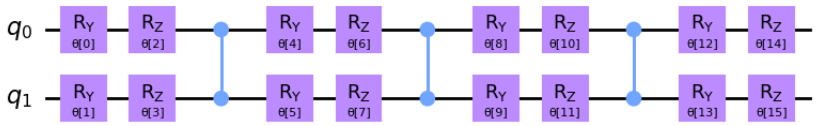


(d) Second qubit after training.

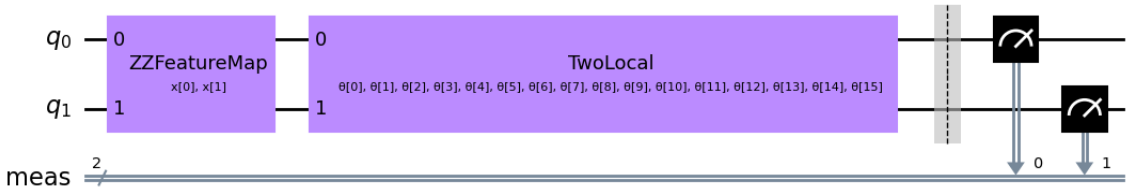
Figure B.1: Qubits (a) and (b) represent the result of quantum encoding. Qubits (c) and (d) are the resulting representations of the point after the training process.



(a) ZZFeatureMap used as quantum encoder.



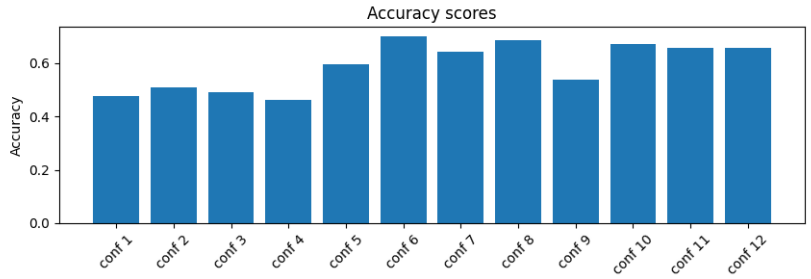
(b) TwoLocal ansatz.



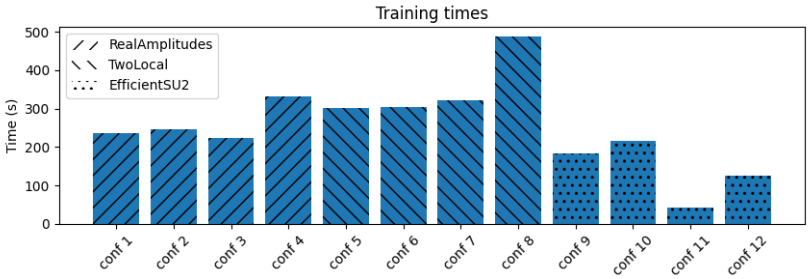
(c) Complete variational circuit.

Figure B.2: The complete circuit has 16 variational parameters that are optimized through COBYLA.

Data can be encoded into the variational circuit using the feature map obtained from the genetic algorithm. With this custom feature map the accuracy reaches 0.7 and the training time is lower than the previous quantum circuit of almost 200s (Figure B.3).



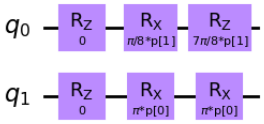
(a) Acccuracy scores.



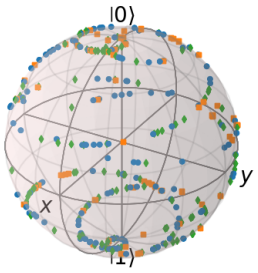
(b) Training times.

Figure B.3: Training times and accuracy scores of the 12 combinations tested. The configurations are the same except for the feature map which is the one generated by the genetic algorithm.

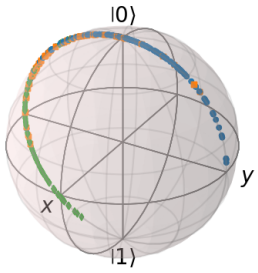
The new feature map and the data encoded in the Hilbert space are in figure B.4. After the training process, data points are easier to classify since, in the second qubit, they are grouped per class, differently from the previous projection in Figure B.1, in which they are not easiliy separable.



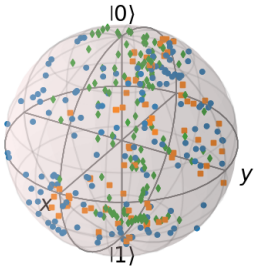
(a) Custom feature map.



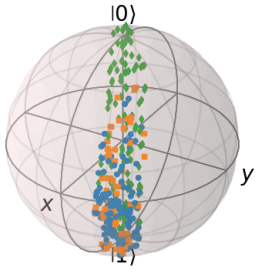
(b) First qubit.



(c) Second qubit.



(d) First qubit after training.



(e) Second qubit after training.

Figure B.4: The feature map in (a) generates the same embedding (b) and (c) as the one in Figure 2.6 since the two algorithms use the same encoding. Data points after the training (d) and (e) are easier to classify with respect to the previous case.

Bibliography

[1] Jack D. Hidary. *Quantum Computing: An Applied Approach*. Springer Publishing Company, Incorporated, 1st edition, 2019.

[2] Fabio Valerio Massoli, Lucia Vadicamo, Giuseppe Amato, and Fabrizio Falchi. A leap among quantum computing and quantum neural networks: A survey, 2022.

[3] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, aug 2021.

[4] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, oct 2014.

[5] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, sep 2017.

[6] S. Ganguly. *Quantum Machine Learning: An Applied Approach: The Theory and Application of Quantum Machine Learning in Science and Industry*. Apress, 2021.

[7] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning, 2020.

[8] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3), sep 2020.

[9] Sergio Altares-López, Angela Ribeiro, and Juan José García-Ripoll. Automatic design of quantum feature maps. *Quantum Science and Technology*, 6(4):045015, aug 2021.

[10] Bang-Shien Chen and Jann-Long Chern. Generating quantum feature maps for svm classifier, 2022.

[11] Junyu Liu, Changchun Zhong, Matthew Otten, Anirban Chandra, Cristian L Cortes, Chaoyang Ti, Stephen K Gray, and Xu Han. Quantum kerr learning. *Machine Learning: Science and Technology*, 4(2):025003, apr 2023.

[12] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12):1900070, oct 2019.

[13] Thomas Hubregtsen, Josef Pichlmeier, Patrick Stecher, and Koen Bertels. Evaluation of parameterized quantum circuits: on the relation between classification accuracy, expressibility and entangling capability, 2020.

[14] F. Bloch. Nuclear induction. *Phys. Rev.*, 70:460–474, Oct 1946.

[15] David A. Meyer and Nolan R. Wallach. Global entanglement in multiparticle systems. *Journal of Mathematical Physics*, 43(9):4273–4278, aug 2002.

[16] Corinna Cortes and Vladimir Naumovich Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 2004.

[17] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, mar 2019.

[18] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins Apl Technical Digest*, 19:482–492, 1998.

[22] James Stokes, Josh Izaac, Nathan Killoran, and Giuseppe Carleo. Quantum natural gradient. *Quantum*, 4:269, may 2020.

[23] Julien Gacon, Christa Zoufal, Giuseppe Carleo, and Stefan Woerner. Simultaneous perturbation stochastic approximation of the quantum fisher information. *Quantum*, 5:567, oct 2021.

[24] M. J. D. Powell. A view of algorithms for optimization without derivatives 1. 2007.

[25] John A. Nelder and Roger Mead. A simplex method for function minimization. *Comput. J.*, 7:308–313, 1965.

[26] Soonchil Lee, Seong-Joo Lee, Taegon Kim, Jae-Seung Lee, Jacob Biamonte, and Marek Perkowski. The cost of quantum gate primitives. *Journal of Multiple-Valued Logic and Soft Computing*, 12:561–573, 01 2006.