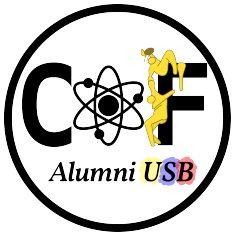
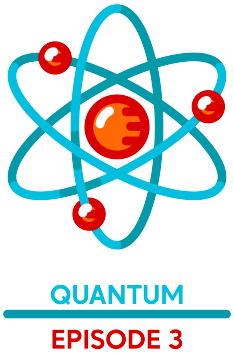




Quantum Machine Learning Project

Alan Vásquez, Ángel Álvarez, Gáby Linares, Jaisar Cammarata, Luis Viloria, Milagro Rojas

Tutor:

Alberto Maldonado Romo

April 30, 2022

Abstract

The machine learning in one of the sub fields of computing science and artificial intelligence whose goal is to develop algorithms that machines can use to mimic the human feature of "*learning*". In recent decades there have been advances in classical machine learning theory and its implementations, but recently this field has expanded into quantum computer science. In this project we used data sets with 4 features and 1 label each, classified as: **train**, **test** and **validation** sets, to run a quantum machine learning model. Using a Pauli feature map and a variational algorithm, we generated a cost function that sends feedback to the variational circuit to updates the values of the parameters of the model. The Pauli feature map set the initial conditions of 5 qubits who were sent to the variational algorithm. The output of the variational circuit was measured and fed to an optimizer, as classical bits, to perform the computation of the values of the cost function and send feedback to the variational algorithm. it was found that the best parameters for the Pauli feature map and the VAR TwoLocal are the 2 repetitions for each one giving precisions of 0.70 and 0.83 for both quantum circuits.

Key words: Quibit, Quantum machine learning, Variational algorithm, Cost functions.

1 Introduction

1.1 What is machine learning?

In the technologic world, *machine learning*, **ML**, is considered as an emerging discipline which can be applied to a large variety of research fields, such as computational biology, finances, natural language processing and many others. ML is a field of artificial intelligence that searches patterns in a data set in an empirical way and creates prediction models by applying statistical methods. ML is classified into *supervised learning*, *unsupervised learning* and *reinforcement learning*.

1.2 Supervised, unsupervised and reinforcement learning

In the supervised learning, the algorithms use "*labeled*" data as input, to find a function that assigns an output label. This function is then applied to an unlabeled data set so that it predicts the output values. It is common to use supervised learning in classification and regression problems.

In unsupervised learning there is no "tagged" data, only the input data is available, without knowing the corresponding output data. It is necessary to describe the structure of the data, to try to find some type of organization that simplifies the analysis. This learning is often used in clustering and correlation problems.

the reinforcement learning is based on *trial and error*. The idea is to improve the response of the model using a feedback process. The algorithm learns by observing the world around it. Its input is the feedback it gets from the outside world in response to its actions.

1.3 Machine learning algorithms

The most common algorithms applied in supervised learning are:

- Decision trees.
- Naïve Bayes classification.
- Least squares regression.
- Logistic regression.
- Support Vector Machines (SVMs).

- "Ensemble" methods (Sets of classifiers).

For the case of unsupervised learning they are:

- Clustering algorithms.
- Decomposition into singular values.
- Principal Component Analysis (Independent Component Analysis).

1.4 Quantum computing, what is a qubit?

A *qubit* is the smallest unit of quantum information (it is the analog of the *bit* in classical information described by 0 and 1), consisting of a two-level quantum system defined as the states $|0\rangle$ and $|1\rangle$.

The state of a qubit, being quantum in nature, complies with the postulates of quantum mechanics. As a consequence, the state of a qubit, before being measured, is a superposition of the states $|0\rangle$ and $|1\rangle$ which can be represented in a Bloch sphere. The mathematical way to express a qubit is as follows:

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

where α and β are complex numbers whose squared modules represent the probability that the qubit is in the state $|0\rangle$ and $|1\rangle$ respectively.

It is necessary (for the probabilistic notion to work) that the normalization condition is met, the latter being:

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

When measuring a qubit, it will collapse into one of two possible states and display a single result: the $|0\rangle$ state or the $|1\rangle$ state.

1.5 Quantum gates and quantum circuits

A qubit can be represented in a Bloch sphere as can be seen in fig. 2. Quantum gates are logical operations associated with possible rotations about a certain axis of the Bloch sphere that serve to change the state of the qubit.

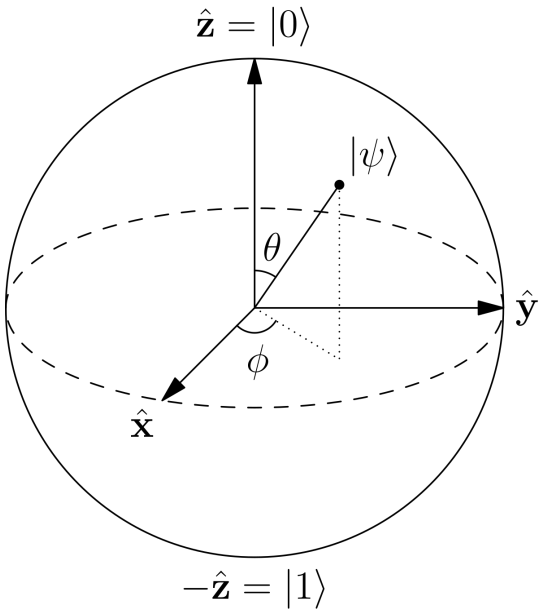


Figure 2: Geometric representation of the Bloch sphere.

The combination of these quantum gates in a set of qubits is known as a quantum circuit, whose initial states are the states in which the operator initializes the qubits (normally in the state $|0\rangle$) and the final states of the qubits depend on operations applied through the quantum gates.

1.6 Quantum machine learning features

Quantum Machine Learning (QML) arise by combining quantum computing and ML. This combining can do in four different way, depending whether the nature of the data is classical (C) or quantum (Q), or whether the algorithm is executed on a classical (C) or quantum (Q) computer.

- CC: classical data is processed by using classical computers, but this is difference from ordinary ML cause it uses algorithms inspired by quantum computing.
- CQ: classical data is processed by using quantum machine learning algorithms.
- QC: quantum data is processed by using classical machine learning algorithms.
- QQ: quantum data is processed by using quantum machine learning algorithms.

2 Motivation

Why use quantum machine learning? The implementation of classic machine learning has definitely contributed to technological advances. However, handling large amounts of data and the characteristics of the problem to be solved can require valuable computing power and time. For this reason, combining ML with quantum computing results in a significant performance gain since it can solve the same problem with fewer operations.

From another point of view, machine learning can also be key to making quantum computers more reliable. This is because, currently, one of the biggest problems in quantum computing is to manipulate qubits safely and maintain their quantum states for longer periods of time. The ML can bring new approaches to qubit error management.

2.1 Our goals

In this work, we want to meet the following goals:

1. Choose the better way to encode classical data in terms of qubits and a variational circuit proposed by the Qiskit framework that achieves a binary classification.
2. Demonstrate the knowledge to define our own variational circuit and identify the number of layers to perform against the 100% accuracy metric.

3 Methods

3.1 Data sample

We recreate the classification using a variational circuit, with which a classical value is passed to be interpreted by a quantum computer. A **training data set** and a **test data set** were worked on, each of them having 5 values corresponding to 4 features and 1 label. The first four parameters were encoded (**input**) and the last one tells us the class to which it belongs (**label**): **0** or **1**. The training file consists of a *.csv* file of 300 data and the test file, in the same format, consists of 120 data.

3.2 Qiskit

In order to meet the objectives, an open source *software development kit* (**SDK**), called *Qiskit*, was used. It's developed to work with quantum computers and was created by *IBM Research*. Qiskit is based primarily on the *Python* programming language and provides tools for creating and manipulating quantum programs and running them on quantum device prototypes in *IBM Quantum Experience* or in simulators on a local computer. It follows the circuit model for universal quantum computing and can be used for any quantum hardware (currently supporting superconducting qubits and trapped ions) that follows this model.The packages *qiskit*, *qiskit-utils*, *qiskit-aqua* were used. Additionally, we worked with the *Matplotlib*, *Pandas* and *Numpy* libraries.

3.3 From classical data to quantum data, encoding data methods

Data representation is crucial for the success of machine learning models. For classical machine learning, the problem is "'*how to represent the data numerically*", so that it can be best processed by a classical machine learning algorithm.

For quantum machine learning, this question is similar, but more fundamental: *how to represent and efficiently input the data into a quantum system, so that it can be processed by a quantum machine learning algorithm*. This is usually referred to as data encoding, but is also called data embedding or loading.

Let's say that we have a data set $\mathscr{X}$, consisting of **M** samples each of them having **N** entries or features. Our data can be written as:

$$\mathscr{X} = \left\{x^{(1)}, \dots, x^{(m)}, \dots, x^{(M)}\right\} \tag{3}$$

where each of the elements $x^{(m)}$ is an **N** dimensional vector and $m = 1,\dots,M$. The question to answer is: which embedding technique can we use to represent $\mathscr{X}$ into a quantum system?

To answer the previous question, in Quantum Machine Learning we have:

- Basis encoding.
- Amplitude encoding.
- Angle encoding.

3.3.1 Basis encoding

Basis encoding associates a classical **N**-bit string with a *computational basis state* of an **N**-qubit system. For example if $x = 5$ this can be represented as a 4-bit string, 101 and by a 4-qubit system as the quantum state $|0101\rangle$. In general, for a **N**-bit string, the corresponding **N**-qubit state is $\left|x^{(m)}\right\rangle = |b_1 b_2 \dots b_N\rangle$ with $b_n \in \{0, 1\}$ for $n = 1, 2, \dots, N$ and $m = 1, 2, \dots, M$. We can represent the entire dataset as superpositions of computational basis states:

$$|\mathscr{X}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left|x^{(m)}\right\rangle \tag{4}$$

3.3.2 Amplitude encoding

Amplitude encoding encodes data into the amplitudes of a quantum state. It represents a normalized classical **N**-dimensional datapoint, x, as the amplitudes of a **n**-qubit quantum state, $\left|\psi_x\right\rangle$:

$$\left|\psi_x\right\rangle = \sum_{i=1}^{N} x_i \left|i\right\rangle \tag{5}$$

where $n = 2^N$; x_i is the i^{th} element of x and $|i\rangle$ is the i^{th} computational quantum state.

To encode the classical data set $\mathscr{X}$ described above, we concatenate all **M** **N**-dimensional datapoints into one amplitude vector, of length $\mathbf{N} \times \mathbf{M}$:

$$\alpha = \mathbf{A}_{\text{Norm}}(x_1^{(1)}, \ldots, x_N^{(1)}, \ldots, x_1^{(m)}, \ldots, x_N^{(m)}, \ldots, x_1^{(M)}, \ldots, x_N^{(M)}) \tag{6}$$

where $\mathbf{A}_{\text{Norm}}$ is a normalization constant, such that $|\alpha|^2 = 1$. The data set $\mathscr{X}$ can now be represented in the computational basis as:

$$|\mathscr{X}\rangle = \sum_{i=1}^{N} \alpha_i |i\rangle \tag{7}$$

3.3.3 Angle encoding

Angle encoding encodes **N** features into the rotation angles of n qubits, where $N \leq n$. For example, the datapoint $x = (x_1, \ldots, x_N)$ can be encoded as:

$$|x\rangle = \bigotimes_{i=1}^{N} U(x_j^{(i)}) \tag{8}$$

where:

$$U(x_j^{(i)}) = \begin{bmatrix} \cos(x_j^{(i)}) & -\sin(x_j^{(i)}) \\ \sin(x_j^{(i)}) & \cos(x_j^{(i)}) \end{bmatrix} \tag{9}$$

Dense angle encoding is a slight generalization of angle encoding, that encodes two features per qubit using the relative phase, where the $x = (x_1, \ldots, x_N)$ datapoint can be encoded as:

$$|x\rangle = \bigotimes_{i=1}^{N/2} \cos(x_{2i-1})|0\rangle + e^{ix_{2i}} \sin(x_{2i-1})|1\rangle \tag{10}$$

3.3.4 Arbitrary encoding

Arbitrary encoding encodes **N** features as **N** rotations on parameterized gates on **n** qubits, where $\mathbf{n} \leq \mathbf{N}$. Like angle encoding, it only encodes one datapoint at a time, rather than a whole dataset. It also uses a constant depth quantum circuit and $\mathbf{n} \leq \mathbf{N}$ qubits, meaning it can be run on current quantum hardware.

3.4 Variational algorithms

Variational algorithms Uses a parameterized quantum circuit, $U(\theta)$, to prepare the state $|\psi(\theta)\rangle = U(\theta)|0\rangle$, and measure the expectation value using a quantum computer. We define a cost function $C(\theta)$, that determines how good θ is for the problem we are trying to solve. We then use a classical computer to calculate the cost function and provide updated circuit parameters using an optimization algorithm. The goal of the algorithm is to find the circuit parameters θ for the parameterized quantum circuit $U(\theta)$ that minimizes the cost function $C(\theta)$.

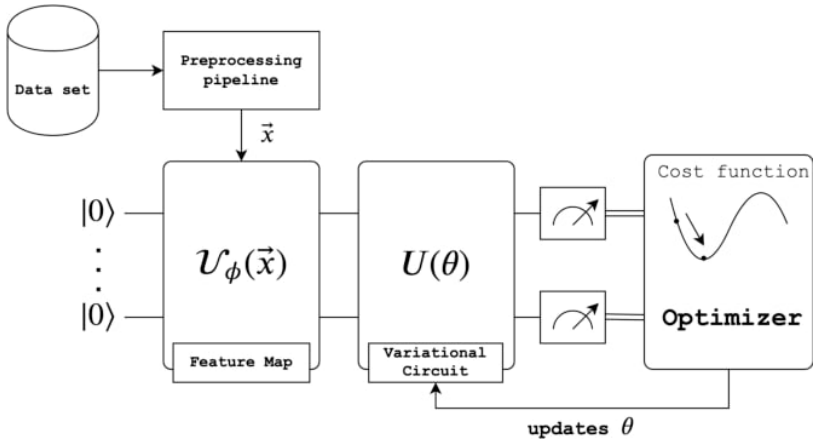


Figure 3: Scheme of a variational algorithm.

4 Our approach

For this project we perform a series of simulations using a **train set** for the purpose of training the variational algorithm, a **test set** to observe whether or not the variational model performs a good performance a **validation set** which was 10 % of the train set and whose purpose was to give an estimate of model skill while tuning model's hyperparameters.

The optimizer used to run the simulations was the **SPSA**. It was chosen to deal with our problem because it's designed to work with data of big values. Besides from the optimizer, we used the arbitrary enconding to load the data of the train, test and validation sets.

The Pauli feature map used to run the simulations was given the following parameters:

- Type of entanglement = Full
- Paulis gates = **Z**, **X**, **ZY**
- Depth: 28

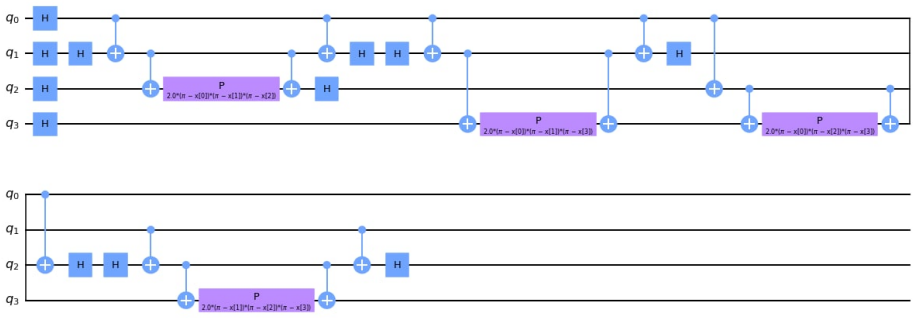


Figure 4: Pauli feature map circuit.

The Variational algorithm used to run the simulations (the Twolocal) was given the following parameters:

- Type of entanglement = Full
- List of rotations = **RY**

- List of controls = **CX** (control **X**).
- Depth: 8

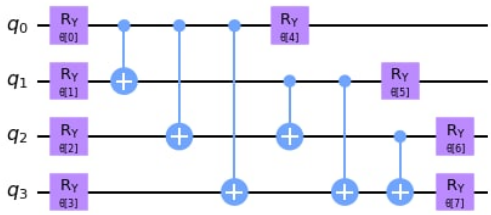
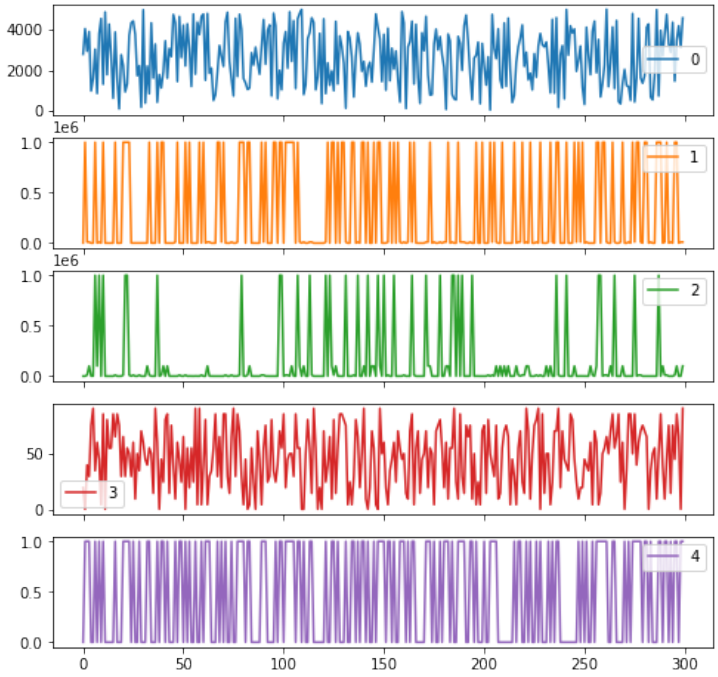


Figure 5: VAR TwoLocal circuit.

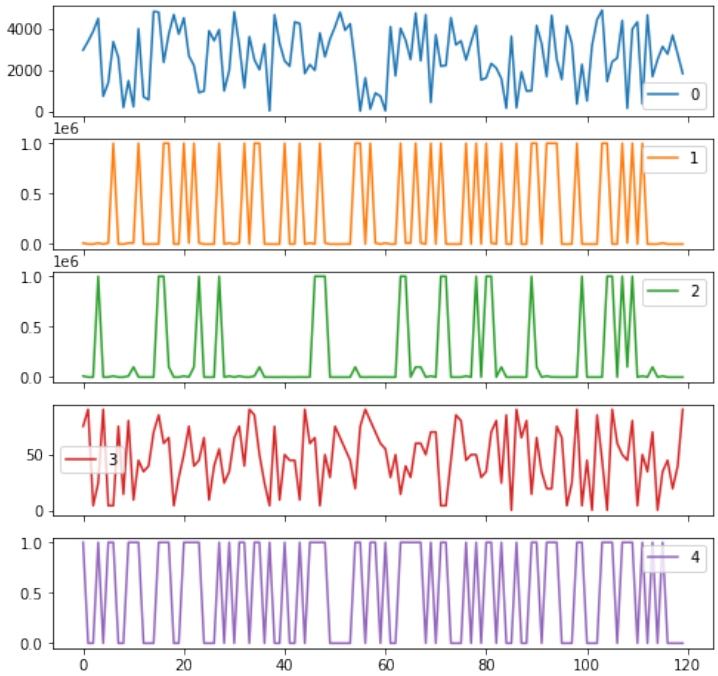
As it can be seen in fig. 3, we preprocess the data from the datasets and pass it to the Pauli feature map. The Pauli feature map sets the conditions of the qubits to be run in the circuit. Once set all qubits, they run in the variational circuit to generate an output. The output is measured and fed to an optimizer which then computes a cost function value. The optimizer sends feedback to the variational circuit so it can produce the updates of the parameters and guarantee that we reach the minimum of the cost function.

5 Results

5.1 Determining the class



(a) First look of **train data set**.



(b) First look of **test data set**.

Figure 6: Visualizations of the **train** and **test** data sets.

5.2 Testing our algorithm

After training the data, 3 tests were carried out with Pauli feature map using VAR TwoLocal.

Table 1: Confusion matrix for Pauli feature map (1 repetition) and VAR TwoLocal (3 repetition)

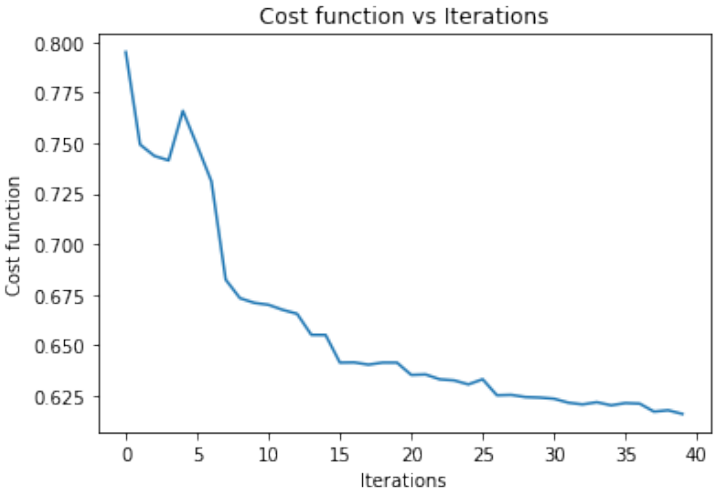
	precision	recall	f1-score	support
A	0.50	0.88	0.64	16
B	0.33	0.07	0.11	15
accuracy	-	-	0.48	31
macro avg	0.42	0.47	0.37	31
weighed avg	0.42	0.48	0.38	31

Table 2: Confusion matrix for Pauli feature map (2 repetition) and VAR TwoLocal (1 repetition)

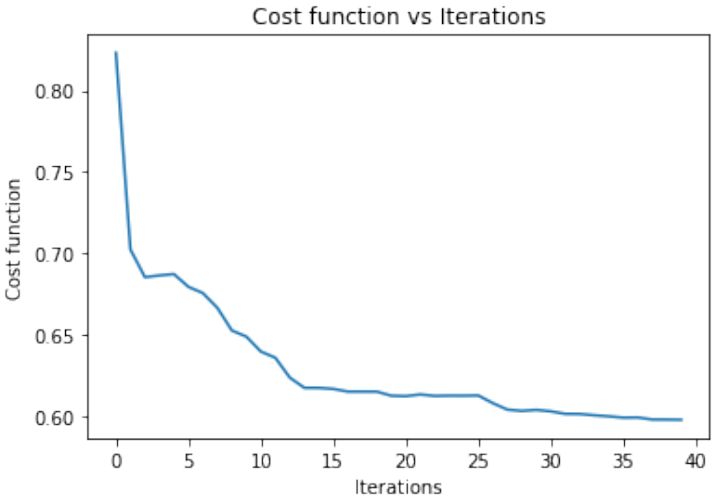
	precision	recall	f1-score	support
A	0.60	0.94	0.73	16
B	0.83	0.33	0.48	15
accuracy	-	-	0.65	31
macro avg	0.72	0.64	0.60	31
weighed avg	0.71	0.65	0.61	31

Table 3: Confusion matrix for Pauli feature map (2 repetition) and VAR TwoLocal (2 repetition)

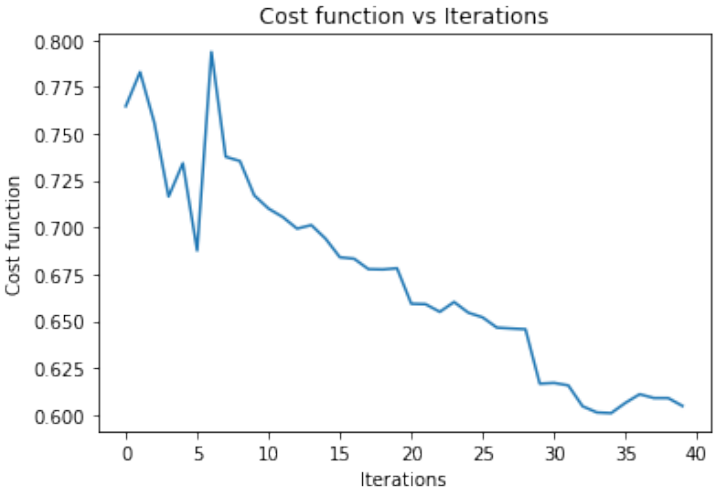
	precision	recall	f1-score	support
A	0.70	0.88	0.78	16
B	0.82	0.60	0.69	15
accuracy	-	-	0.74	31
macro avg	0.76	0.74	0.74	31
weighed avg	0.76	0.74	0.74	31



(a) Cost function for Pauli feature map (1 repetition) and VAR TwoLocal (3 repetition).



(b) Cost function for Pauli feature map (2 repetition) and VAR TwoLocal (1 repetition).



(c) Cost function for Pauli feature map (2 repetition) and VAR TwoLocal (2 repetition).

Figure 7: Graphs of the cost function Vs iterations for different parameters of the Pauli feature map and the VAR TwoLocal.

5.3 QML limitations

One of the main limitations of performing quantum machine learning is the amount of time for simulations. Performing a simulation of the algorithm on a classical system (a non-quantum PC) using qiskit can take a long time.

The time that the algorithms run the simulations could be from 30 min to 1 h or even 1.5 h+. The best way to run such algorithms is using a quantum computer rather than a normal classic computer.

6 Conclusions

According to the data in table 3, the best parameters for the Pauli feature map and the VAR TwoLocal are the 2 repetitions for each one. The criteria to establish an optimal performance is the maximization of the precision. In tables 1 to 3 it can be seen that the precision of the table 3 are the highest ones. Visually, we can observer how is the path that took the cost function with respect to the iterations in the fig. 7c.

7 Bibliography

References

[1] D. J. Griffiths, *Introduction to quantum mechanics*. Pearson International Edition (Pearson Prentice Hall, Upper Saddle River, 2005), 1962.

[2] M. Nakahara and T. Ohmi, *Quantum computing: from linear algebra to physical realizations*. CRC press, 2008.

[3] M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*. Springer Cham, 1 ed., 2018.