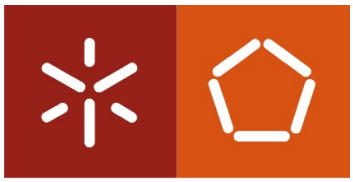
Binary Classification with Quantum Variational Circuit

Quantum Machine Learning

Márcio Mano
Pedro Teixeira
Master in Physics Engineering

Professors:
Luís Paulo Santos
André Sequeira



Universidade do Minho

Contents

1 Introduction

O foco deste trabalho é a construção de um circuito variacional quântico capaz de classificar dados clássicos. Dito isto, o nosso sistema será então constituído por uma parte quântica e uma parte clássica.

- Na parte quântica (circuito quântico) temos um bloco responsável pelo *encoding/embedding*, um bloco responsável pelo algoritmo variacional e blocos de medição;

- Na parte clássica temos o pós-processamento (associa as medições a rótulos), o cálculo da *loss function* e o otimizador dos parâmetros variacionais (θ) Fig.1, também poder ser preciso, em alguns casos, um pré-processamento especial (clássico), como a redução da dimensionalidade do *data set* através das técnicas de ***PCA***[1].

Estas duas primeiras partes (circuito quântico e pós-processamento) ficam conectados em *loop* criando assim um sistema híbrido entre a computação quântica e computação clássica, podemos observar na Fig.1 um exemplo genérico desse *loop*.

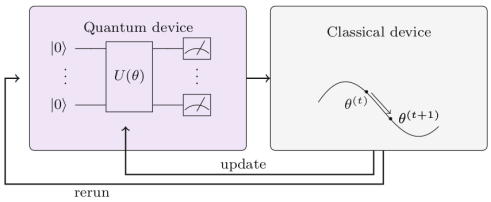


Figure 1: Exemplo de sistema híbrido[2]

$$norm = \sum_i (x_i)^2 \tag{1}$$

$$\tilde{x} = \frac{x}{\sqrt{norm}} \tag{2}$$

2 Overview

Passos para a criação do nosso sistema:

1. ***Data Set***: Definir o conjunto de dados a utilizar;

   Posteriormente esse conjunto de dados será dividido em dois subconjuntos, um subconjunto de treino e um subconjunto de teste.

   Normalmente em problemas de *machine learning* clássicos o conjunto de dados de teste tem cerca de 30% do tamanho do conjunto de dados de treino[3].

2. ***Embedding/Encoding***: Definir o tipo de codificação a utilizar;

   De modo a transformar os dados clássicos do nosso conjunto de dados em estados quânticos, para serem posteriormente processados por uma máquina quântica;

3. ***Parameterized Model***: Aplicar um modelo parametrizado;

4. ***Circuit Measurements***: Medição do circuito para extrair as *labels*;

5. ***Optimization Techniques***: Usar técnicas de otimização para encontrar os melhores parâmetros para o sistema parametrizado, de modo a encontrarmos padrões dentro do nosso conjunto de dados.

2.1 Objectives with this project

Estes foram os objetivos a que nos propusemos a cumprir numa primeira abordagem ao problema:

- Utilizar os computadores quânticos disponibilizados pela **IBM**[4] para correr o nosso sistema;

- Verificar a influência do atributo subjetivo (*quality*) tem no nosso classificador;

- Encontrar os pontos de *underfitting* e *overfitting* do nosso sistema;

- Comparar os nossos resultados experimentais com os resultados experimentais apresentados em[5];

Infelizmente não conseguimos realizar com sucesso todos os objetivos que propusemos numa primeira instância, contudo ao longo do desenvolvimento do trabalho, fomos pensando em alternativas a estes objetivos.

Um dos objetivos onde tivemos pouco sucesso, foi no de tentar correr o nosso classificador nas máquinas quânticas disponibilizadas ao público pela **IBM**, isto, pois ao tentar aceder à máquina quântica iterativamente, ou seja, correr os nossos circuitos a cada iteração do otimizador obriga nos a esperar em *queue* (onde o tempo de espera podem facilmente chegar aos 10 minutos) cada vez que precisamos de testar os novos parâmetros variacionais retornados pelo otimizador (Fig.1) o que faz com que o processo de treino possa levar dias até ser concluído.

A outra abordagem que tentamos foi usar o *TorchRunTime*[6] para tentar correr a parte de otimização nos servidores da IBM, fazendo com que não tenhamos de ficar à espera na *queue* a cada iteração do otimizador, contudo esta abordagem possui as suas desvantagens, como o tempo máximo de execução de 8 horas (segundo o *site* da IBM[7]), fazendo com que o número

máximo de iterações do otimizador com que nós conseguimos trabalhar seja 25 iterações o que é muito pouco se pretendemos retirar alguma conclusão sobre os nossos resultados.

```
train execution time:  11498.90387415886
id:  cafnv75toase3u9lgiug
execution time:  159.3931486606598
score:  0.4583333333333333
```

Figure 2: Tempo de execução para apenas 20 épocas

Como podemos ver o servidor da IBM demorou mais de 3 horas para fazer um treino com apenas 20 épocas.

Portanto, como alternativa a este problema decidimos implementar um ambiente de simulação onde usamos o ruído de uma máquina quântica (*IBMQ Manilla[8]*), para simular um uso mais realista de uma máquina quântica.

Outro objetivo apresentado na primeira avaliação do projeto era verificar a influência da *feature* qualidade na classificação e após testar classificar o nosso *data set* com a *feature* incluída e compara com a situação em que a *feature* não é tida em consideração podemos verificar que o uso da mesma não mostrou nenhuma mudança significativa.

Outro objetivo onde não tivemos sucesso, foi no de encontrar os pontos de *overfitting* e *underfitting* pois inicialmente acreditávamos que seria uma análise mais fácil do que é, contudo, como alternativa, tentamos usar um número elevado de *data points* para fazer a classificação, como forma de otimizar o tempo de execução decidimos testar para apenas 2 *qubits*, contudo, esta tentativa de classificação não nos gerou bons resultados e devido ao elevado tempo de execução não conseguimos repeti-lo vezes o suficiente para conseguirmos tirar algum tipo de ilação.

Infelizmente apesar de no *paper*[5] ser mencionado que eles usam um *data set* de vinho português, o que nos fez acreditar que estávamos a usar o mesmo *data set*, provou não ser o caso, pois após uma leitura mais cuidada podemos reparar que o *data set* utilizado no *paper* em questão é sobre vinhos do norte da Itália e não sobre vinhos portugueses, contudo, apesar de termos usados diferentes *data set* tentaremos à mesma comparar resultados, tendo em consciência de que apesar de serem problemas parecidos, os *data sets* em questão são completamente distintos.

3 *Data Set*

3.1 *Wine Quality* Data Set

O *data set* selecionado pode ser encontrado no *link* seguinte[9].

Este *data set* está dividido em dois *data sets*, um *data set* associado a vinho verde tinto e outro *data set* associado ao vinho verde branco, ambos os vinhos com origem no norte de Portugal.

O nosso objetivo é usar os dois *data sets* como um só, isto é, juntaremos os dois *data sets* de modo que ao retirar um *data point*, da junção dos dois *data sets*, ao mesmo será associado um *label* de "vinho verde tinto" ou "vinho verde branco" dependendo do *data set* original ao qual pertenciam, ou seja, com este *data set* pretendemos fazer um **Classificador Binário** que consiga diferenciar "vinho verde tinto" de "vinho verde branco" tendo apenas a informação disponível na subsecção seguinte.

3.1.1 Data points features

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol
12. quality (atributo subjetivo)

3.1.2 Classes

Evidentemente a primeira diferença que nos vem à cabeça quando pensamos em vinho branco e vinho tinto é a cor, contudo, a real diferença advém do modo como é feito daí provém a cor[10].

1. Vinho Tinto têm 1599 *data points* associados;
   - Mais álcool;
   - Menos açúcar.
2. Vinho Branco têm 4898 *data points* associados.
   - Menos álcool;
   - Mais açúcar.

4 Data pre-processing and Encoding

4.1 Data pre-processing

Antes de podermos usar os nossos dados como *inputs* de um circuito quântico temos de realizar algum pré-processamento, como a normalização do *data set* e em alguns casos a redução da dimensionalidade do próprio *data set*.

Para tentar enquadrar a dimensão do nosso *data set* com as limitações presentes nas máquinas quânticas (como o número de *qubits*), tivemos que procurar por formas de reduzir a dimensionalidade como as técnicas de *Principal Component Analysis* (PCA), esta técnica é conhecida por identificar as componentes principais dos nossos dados reduzindo assim a dimensão, sem muita perda de informação importante[1].

4.2 Data Encoding

Para o *encoding* do nosso conjunto de dados usamos duas diferentes abordagens:

1. **Amplitude Encoding**;

   Com esta técnica pretendemos dar *encode* dos nossos dados nas amplitudes de um estado quântico por isso, temos que normalizar os nossos dados para que a soma do quadrado das *features* seja igual a 1, é de notar que o nosso *data set* tem 12 *features* e como não é uma potência de 2 teremos que fazer *padding* ao resto das amplitudes do estado quântico, nos casos em que usemos o PCA, se o número de componentes não for uma potência de 2 também que teremos de aplicar um *padding*.

$$|\psi_x\rangle = \sum_{i=0}^{N-1} x_i |i\rangle \tag{3}$$

2. **Feature Map**

   Para usarmos este tipo de encoding, temos primeiro que aplicar uma técnica de PCA, pois fazer otimizações com 13 *qubits* iriam ser precisas muitas épocas de treino (o número de parâmetros variacionais aumenta com o número de *qubits*), já para não falar que o processo de otimização ia ser mais lento do que para um número inferior de *qubits*, lembrando que para este *encoding*, o número de *qubits* é igual ao número de *faeatures*. Os inputs para este tipo de *encoding* são ângulos por isso temos que garantir que cada *feature* se encontra normalizada entre $[-\pi, \pi]$.

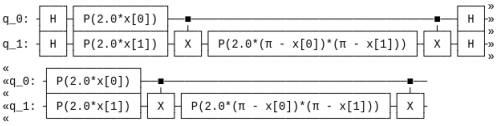


Figure 3: Para 2 *qubits* e 2 repetições

Ao usarmos estes 2 *encodes* diferentes temos como objetivo observar as diferenças de cada um. Uma das primeiras diferenças com que nos deparamos é o tempo de execução associado ao treino dos parâmetros (θ), onde podemos concluir que ao usar o *amplitude encodding* o treino dos parâmetros pode demorar duas vezes mais do que usando *ZZ Feature Map*

5 Parameterized Model

Para este trabalho usaremos como *ansatz* (circuito parameterizado) o seguinte circuito:

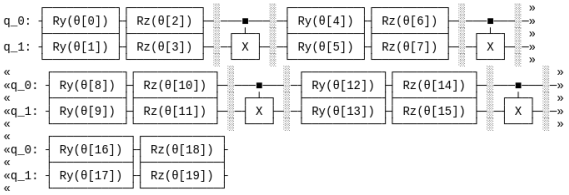


Figure 4: Hardware efficient SU(2) 2-local circuit, para 2 *qubits* com 4 repeticoes

A escolha deste *ansatz* é simplesmente explicada pelo facto de este ser o circuito variacional melhor otimizado para ser corrido em *hardware* quântico.

6 Circuit Measurements

Em problemas de classificação binária é usual utilizarmos métodos de medição e classificação, estes métodos permitem atribuir uma *label*/classe à medida que realizamos as medições. Neste projeto utilizamos dois métodos para realizar tal tarefa:

- *Parity post-processing:*

  Ao medir todos os *qubits* do nosso sistema na base computacional obteremos uma *string* binária (p.e 101000111010) o tamanho da *string* será igual ao número de *qubits* e basicamente o que fazemos é verificar a paridade da nossa *string* binária, isto é, contamos o número de $1s$. Quando esse número for par associamos a uma classe, quando for impar associamos à outra classe.

- *Expected value of the Pauli string* $\langle \sigma_z \otimes ... \otimes \sigma_z \rangle$:

  Neste *encoding* medimos todos os *qubits* e extraímos o valor da string de pauli que é dado pela seguinte fórmula:

$$\sum_{i=0}^{2^n-1} (-1)^{H(i) mod 2} P_i$$

  Onde $H(i)$ e $P(i)$ são o *Hamming weight* e a probabilidade do estado $|i\rangle$ respetivamente, se o valor expetável for positivo associamos a uma classe, quando for negativo associamos à outra classe.

7 Cost Function

Como recorremos a duas formas diferentes de fazer o *circuit measurement* teremos que usar duas *cost function* diferentes:

- Mean Squared Error (MSE) para o *expected value of the Pauli string*

$$C(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i,\theta) - y^{truth})^2$$

- Cross Entropy (CE) para o *parity post-processing*

$$C(\theta) = \frac{1}{N}\sum_{i=1}^{N} y^{truth}\ln\left(f(x_i,\theta)\right)$$

8 Optimization Techniques

Como técnica de otimização optamos por usar um otimizador clássico chamado **SPSA**, que se encontra definido no *qiskit*, existem vários motivos que justificam a nossa escolha de otimizador, um dos motivos é proveniente dos vários testes usando outros otimizadores como o *GradientDescent*, *QNSPSA* e o *ADAM*, sendo que o *SPSA* foi o otimizador para o qual conseguimos obter melhores resultados, outro motivo pelo qual o escolhemos foi o facto de ter sido o otimizador usado num problema de classificação parecido com o nosso[5], contudo no nosso possuímos, em alguns casos, problemas de maior dimensionalidade do que aqueles que foram abordados no *paper*.

9 Results

Para uma melhor compreensão dos resultados obtidos foi utilizado uma metrica denominada "matriz de confusão", a mesmo é uma tabela que permite visualizar o desempenho do nosso algoritmo de classificação após terminar a sua tarefa.

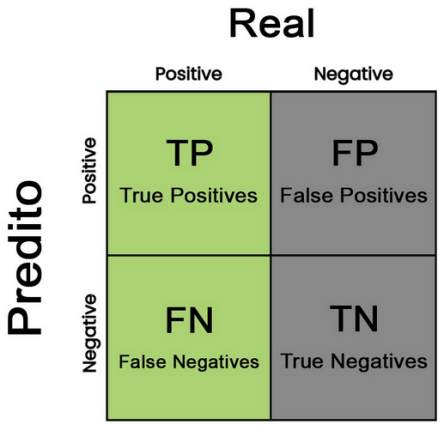


Figure 5: Diagrama da tabela de confusão.

9.1 Noiseless Simulation

9.1.1 2 Qubits

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	200	1024	70.00 %	27.7(7) %
2	4	400	1024	65.5(6) %	22.2(2) %
2	4	600	1024	69.4(4) %	24.07 %
2	4	800	1024	69.4(4) %	31.48 %
2	4	1000	1024	71.1(1) %	25.92 %

Table 1: Cross Entropy (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 7 & 19 \\ 20 & 8 \end{bmatrix}$	$\begin{bmatrix} 5 & 20 \\ 22 & 7 \end{bmatrix}$	$\begin{bmatrix} 8 & 20 \\ 19 & 7 \end{bmatrix}$	$\begin{bmatrix} 7 & 20 \\ 20 & 7 \end{bmatrix}$	$\begin{bmatrix} 7 & 20 \\ 20 & 7 \end{bmatrix}$

Table 2: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	2	200	1024	69.4(4) %	66.6(6) %
2	4	2	400	1024	69.4(4) %	62.96 %
2	4	2	600	1024	70.00 %	91.1(2) %
2	4	2	800	1024	71.1(1) %	61.1(2) %
2	4	2	1000	1024	71.1(1) %	64.81 %

Table 3: Cross Entropy (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 19 & 10 \\ 8 & 17 \end{bmatrix}$	$\begin{bmatrix} 18 & 12 \\ 9 & 15 \end{bmatrix}$	$\begin{bmatrix} 19 & 12 \\ 8 & 15 \end{bmatrix}$	$\begin{bmatrix} 20 & 14 \\ 7 & 13 \end{bmatrix}$	$\begin{bmatrix} 19 & 10 \\ 8 & 17 \end{bmatrix}$

Table 4: Matrizes de confusão.

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	200	1024	68.33 %	37.04 %
2	4	400	1024	66.6(6) %	33.3(3) %
2	4	600	1024	67.7(7) %	35.19 %
2	4	800	1024	67.7(7) %	37.04 %
2	4	1000	1024	67.7(7) %	35.19 %

Table 5: Mean Square Error (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 9 & 14 \\ 18 & 13 \end{bmatrix}$	$\begin{bmatrix} 9 & 15 \\ 18 & 12 \end{bmatrix}$	$\begin{bmatrix} 8 & 16 \\ 19 & 11 \end{bmatrix}$	$\begin{bmatrix} 8 & 16 \\ 19 & 11 \end{bmatrix}$	$\begin{bmatrix} 8 & 16 \\ 19 & 11 \end{bmatrix}$

Table 6: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	2	200	1024	72.7(7) %	66.6(6) %
2	4	2	400	1024	72.2(2) %	66.6(6) %
2	4	2	600	1024	74.4(5) %	66.6(6) %
2	4	2	800	1024	72.2(2) %	68.5(2) %
2	4	2	1000	1024	72.2(2) %	66.6(6) %

Table 7: Mean Square Error (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 19 & 8 \\ 8 & 19 \end{bmatrix}$	$\begin{bmatrix} 17 & 7 \\ 10 & 20 \end{bmatrix}$	$\begin{bmatrix} 18 & 7 \\ 9 & 20 \end{bmatrix}$	$\begin{bmatrix} 19 & 8 \\ 8 & 19 \end{bmatrix}$	$\begin{bmatrix} 17 & 7 \\ 10 & 20 \end{bmatrix}$

Table 8: Matrizes de confusão.

9.1.2 3 Qubits

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	200	1024	78.8(8) %	37.04 %
3	4	400	1024	78.3(3) %	33.3(3) %
3	4	600	1024	79.4(4) %	37.04 %
3	4	800	1024	80.5(5) %	35.18 %
3	4	1000	1024	78.3(3) %	42.59 %

Table 9: Cross Entropy (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 12 & 19 \\ 15 & 8 \end{bmatrix}$	$\begin{bmatrix} 7 & 17 \\ 20 & 10 \end{bmatrix}$	$\begin{bmatrix} 9 & 18 \\ 18 & 9 \end{bmatrix}$	$\begin{bmatrix} 11 & 18 \\ 16 & 9 \end{bmatrix}$	$\begin{bmatrix} 14 & 19 \\ 13 & 8 \end{bmatrix}$

Table 10: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	2	200	1024	70.5(5) %	57.41 %
3	4	2	400	1024	73.8(8) %	53.70 %
3	4	2	600	1024	70.5(5) %	61.1(1) %
3	4	2	800	1024	72.2(2) %	59.26%
3	4	2	1000	1024	71.1(1) %	61.1(1) %

Table 11: Cross Entropy (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 17 & 15 \\ 10 & 12 \end{bmatrix}$	$\begin{bmatrix} 19 & 16 \\ 8 & 11 \end{bmatrix}$	$\begin{bmatrix} 20 & 15 \\ 7 & 12 \end{bmatrix}$	$\begin{bmatrix} 21 & 16 \\ 6 & 11 \end{bmatrix}$	$\begin{bmatrix} 19 & 13 \\ 8 & 14 \end{bmatrix}$

Table 12: Matrizes de confusão.

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	200	1024	83.8(8) %	40.74 %
3	4	400	1024	79.4(4) %	35.19 %
3	4	600	1024	81.1(1) %	35.19 %
3	4	800	1024	80.00 %	33.3(3) %
3	4	1000	1024	82.2(2) %	38.8(8) %

Table 13: Mean Square Error (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 11 & 17 \\ 16 & 10 \end{bmatrix}$	$\begin{bmatrix} 9 & 19 \\ 18 & 8 \end{bmatrix}$	$\begin{bmatrix} 9 & 19 \\ 18 & 8 \end{bmatrix}$	$\begin{bmatrix} 9 & 19 \\ 18 & 8 \end{bmatrix}$	$\begin{bmatrix} 10 & 16 \\ 17 & 11 \end{bmatrix}$

Table 14: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	2	200	1024	71.6(7) %	51.85 %
3	4	2	400	1024	73.8(8) %	55.5(5) %
3	4	2	600	1024	71.6(7) %	64.81 %
3	4	2	800	1024	71.1(6) %	53.7(0) %
3	4	2	1000	1024	73.3(3) %	57.41 %

Table 15: Mean Square Error (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 9 & 8 \\ 18 & 19 \end{bmatrix}$	$\begin{bmatrix} 11 & 8 \\ 16 & 19 \end{bmatrix}$	$\begin{bmatrix} 14 & 7 \\ 13 & 20 \end{bmatrix}$	$\begin{bmatrix} 9 & 8 \\ 18 & 19 \end{bmatrix}$	$\begin{bmatrix} 11 & 7 \\ 16 & 20 \end{bmatrix}$

Table 16: Matrizes de confusão.

9.1.3 4 Qubits

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
4	4	200	1024	77.2(2) %	79.63 %
4	4	400	1024	76.1(1) %	70.37 %
4	4	600	1024	79.4(4) %	74.07 %
4	4	800	1024	77.7(8) %	77.7(8) %
4	4	1000	1024	81.6(7) %	83.3(4) %

Table 17: Cross Entropy (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 17 & 5 \\ 10 & 22 \end{bmatrix}$	$\begin{bmatrix} 9 & 19 \\ 18 & 8 \end{bmatrix}$	$\begin{bmatrix} 21 & 5 \\ 6 & 22 \end{bmatrix}$	$\begin{bmatrix} 19 & 3 \\ 8 & 24 \end{bmatrix}$	$\begin{bmatrix} 21 & 6 \\ 6 & 21 \end{bmatrix}$

Table 18: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
4	4	2	200	1024	71.6(7) %	51.85 %
4	4	2	400	1024	73.8(8) %	55.5(5) %
4	4	2	600	1024	71.6(7) %	64.81 %
4	4	2	800	1024	71.1(6) %	53.7(0) %
4	4	2	1000	1024	73.3(3) %	57.41 %

Table 19: Cross Entropy (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 9 & 8 \\ 18 & 19 \end{bmatrix}$	$\begin{bmatrix} 11 & 8 \\ 16 & 19 \end{bmatrix}$	$\begin{bmatrix} 14 & 7 \\ 13 & 20 \end{bmatrix}$	$\begin{bmatrix} 9 & 8 \\ 18 & 19 \end{bmatrix}$	$\begin{bmatrix} 11 & 7 \\ 16 & 20 \end{bmatrix}$

Table 20: Matrizes de confusão.

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
4	4	200	1024	76.6(7) %	79.63 %
4	4	400	1024	78.3(3) %	81.48 %
4	4	600	1024	82.7(7) %	75.92 %
4	4	800	1024	78.8(9) %	81.48 %
4	4	1000	1024	77.7(7) %	81.48 %

Table 21: Mean Square Error (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 21 & 6 \\ 6 & 21 \end{bmatrix}$	$\begin{bmatrix} 25 & 9 \\ 2 & 18 \end{bmatrix}$	$\begin{bmatrix} 22 & 9 \\ 5 & 18 \end{bmatrix}$	$\begin{bmatrix} 22 & 5 \\ 5 & 22 \end{bmatrix}$	$\begin{bmatrix} 22 & 4 \\ 5 & 23 \end{bmatrix}$

Table 22: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
4	4	2	200	1024	76.6(7) %	37.04 %
4	4	2	400	1024	73.8(9) %	37.0(4) %
4	4	2	600	1024	73.8(9) %	37.04 %
4	4	2	800	1024	76.1(1) %	40.74 %
4	4	2	1000	1024	70.00 %	31.48 %

Table 23: Mean Square Error (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 7 & 13 \\ 20 & 14 \end{bmatrix}$	$\begin{bmatrix} 5 & 12 \\ 22 & 15 \end{bmatrix}$	$\begin{bmatrix} 7 & 15 \\ 20 & 12 \end{bmatrix}$	$\begin{bmatrix} 7 & 14 \\ 20 & 13 \end{bmatrix}$	$\begin{bmatrix} 7 & 16 \\ 20 & 11 \end{bmatrix}$

Table 24: Matrizes de confusão.

9.2 Simulation with Noise

9.2.1 2 Qubits

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	200	1024	68.3(3) %	35.19 %
2	4	400	1024	68.8(8) %	35.19 %
2	4	600	1024	70.00 %	37.04 %
2	4	800	1024	70.00 %	40.74 %
2	4	1000	1024	70.00 %	35.19 %

Table 25: Cross Entropy (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 10 & 15 \\ 17 & 12 \end{bmatrix}$	$\begin{bmatrix} 9 & 16 \\ 18 & 11 \end{bmatrix}$	$\begin{bmatrix} 9 & 16 \\ 18 & 11 \end{bmatrix}$	$\begin{bmatrix} 9 & 16 \\ 18 & 11 \end{bmatrix}$	$\begin{bmatrix} 8 & 15 \\ 19 & 12 \end{bmatrix}$

Table 26: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	2	200	1024	67.2(2) %	35.19 %
2	4	2	400	1024	64.4(5) %	38.8(9) %
2	4	2	600	1024	63.8(8) %	42.59 %
2	4	2	800	1024	66.1(1) %	38.8(8) %
2	4	2	1000	1024	64.4(5) %	42.59 %

Table 27: Cross Entropy (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 6 & 13 \\ 21 & 14 \end{bmatrix}$	$\begin{bmatrix} 6 & 12 \\ 21 & 15 \end{bmatrix}$	$\begin{bmatrix} 6 & 9 \\ 21 & 18 \end{bmatrix}$	$\begin{bmatrix} 6 & 12 \\ 21 & 15 \end{bmatrix}$	$\begin{bmatrix} 6 & 9 \\ 21 & 18 \end{bmatrix}$

Table 28: Matrizes de confusão.

Qubits	Ansatz depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	200	1024	68.8(9) %	38.8(9) %
2	4	400	1024	67.7(8) %	35.18 %
2	4	600	1024	67.7(8) %	37.04 %
2	4	800	1024	68.3(3) %	38.8(9) %
2	4	1000	1024	69.4(4) %	37.04 %

Table 29: Mean Square Error (Cost Function) & Amplitude Encoding

200	400	600	800	1000
$\begin{bmatrix} 9 & 15 \\ 18 & 12 \end{bmatrix}$	$\begin{bmatrix} 10 & 16 \\ 17 & 11 \end{bmatrix}$	$\begin{bmatrix} 8 & 15 \\ 19 & 12 \end{bmatrix}$	$\begin{bmatrix} 10 & 16 \\ 17 & 11 \end{bmatrix}$	$\begin{bmatrix} 8 & 17 \\ 19 & 10 \end{bmatrix}$

Table 30: Matrizes de confusão.

Qubits	Ansatz depth	Feature Map depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	2	200	1024	66.6(6) %	42.59 %
2	4	2	400	1024	65.5(6) %	42.92 %
2	4	2	600	1024	66.1(1) %	38.8(9) %
2	4	2	800	1024	63.8(8) %	40.74 %
2	4	2	1000	1024	64.4(5) %	40.74 %

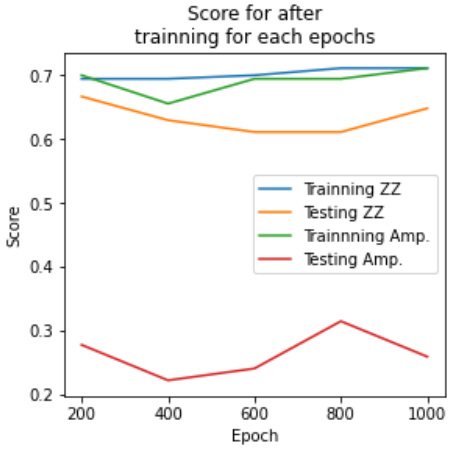
Table 31: Mean Square Error (Cost Function) & ZZFeatureMap

200	400	600	800	1000
$\begin{bmatrix} 6 & 10 \\ 21 & 17 \end{bmatrix}$	$\begin{bmatrix} 6 & 10 \\ 21 & 17 \end{bmatrix}$	$\begin{bmatrix} 6 & 12 \\ 21 & 15 \end{bmatrix}$	$\begin{bmatrix} 6 & 12 \\ 21 & 15 \end{bmatrix}$	$\begin{bmatrix} 6 & 12 \\ 21 & 15 \end{bmatrix}$

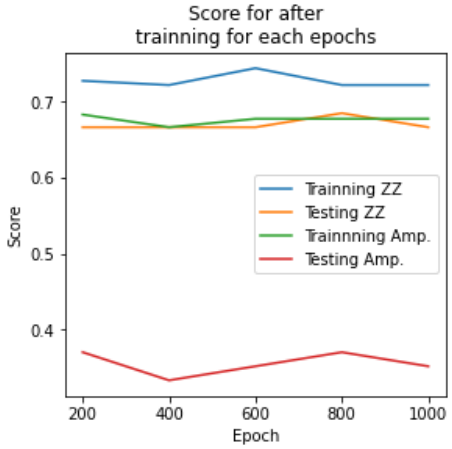
Table 32: Matrizes de confusão.

9.3 Graphs

9.3.1 Noiseless Simulation

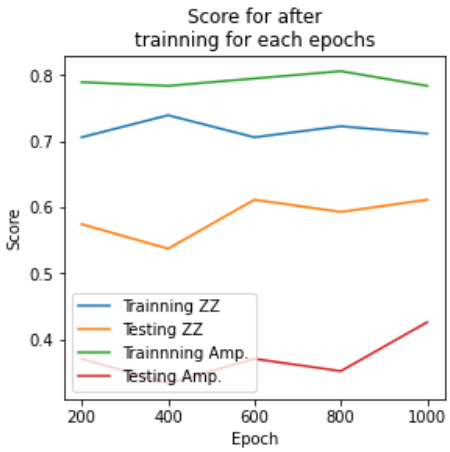


(a) *Cross Entropy Cost Function*



(b) *Mean Squared Error Cost Function*

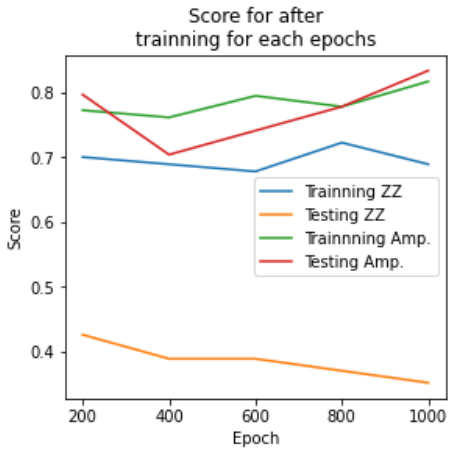
Figure 6: Valores de acc para 2 *qubits*



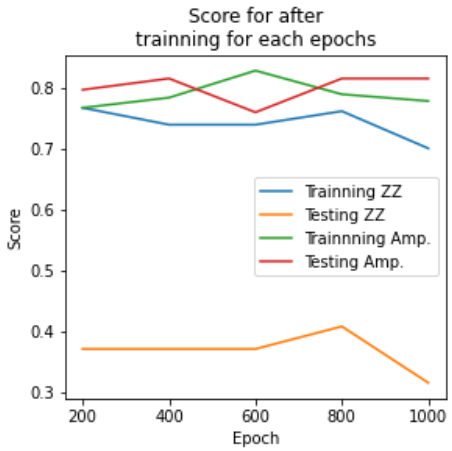
(a) *Cross Entropy Cost Function*

(b) *Mean Squared Error Cost Function*

Figure 7: Valores de acc para 3 *qubits*



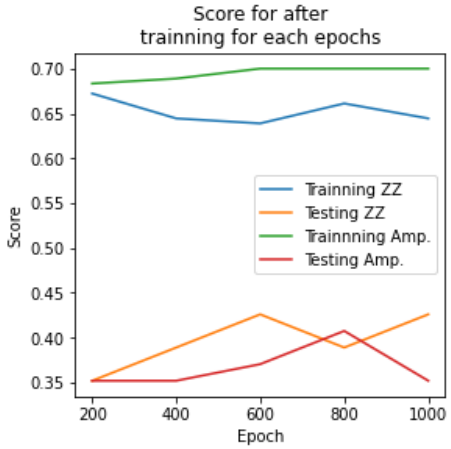
(a) *Cross Entropy Cost Function*



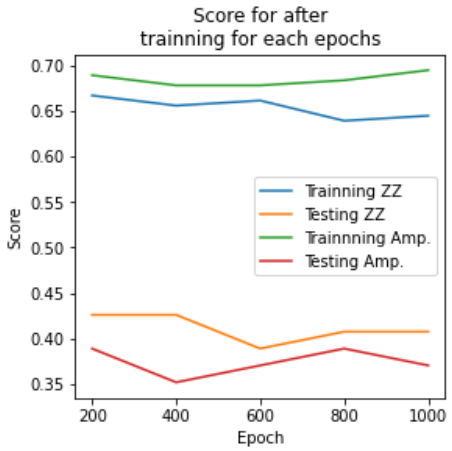
(b) *Mean Squared Error Cost Function*

Figure 8: Valores de acc para 4 *qubits*

9.3.2 Simulation With Noise



(a) *Cross Entropy Cost Function*



(b) *Mean Squared Error Cost Function*

Figure 9: Valores de acc para 2 *qubits*

10 Conclusion

Como podemos ver na secção anterior, a técnica de medição que utilizamos, bem como a *cost function* associada, não diferem muito uma da outra quando comparamos a *accuracy* de cada modelo, ver Figs. 6, 7 e 8, contudo o mesmo não pode ser dito sobre o encode escolhido.

Para dois e três *qubits*, obtemos claramente uma melhoria na classificação quando utilizamos a *ZZFeatureMap* como *enconding*, contudo isto deixa de ser verdade quando atingimos os 4 *qubits*, a partir dos 4 *qubits* a *ZZFeatureMap* diminui significativamente o seu desempenho, indo um bocado contra aquilo que esperávamos, pelo menos ao utilizar um simulador, que era obter melhor desempenho com o aumento do número *qubits*, devido ao aumento da dimensão do problema e também devido ao aumento do número de parâmetros variacionais (θ) cuja a sua combinação deveria resultar num modelo com mais expressividade, a única explicação que nos conseguimos encontrar para este fenómeno, tem a haver com a possibilidade de os nossos dados serem ou não linearmente separáveis, ou seja, é provável que para dois e três *qubits* a técnica de PCA, consiga de alguma forma torna os nossos dados linearmente separáveis, pelo menos parcialmente, contudo, o mesmo parece não acontecer quando aplicamos a regra de PCA para 4 *qubits*.

Por outro lado, o modelo em que recorremos ao *amplitude encoding* (com 4 *qubits*) originou melhores resultados que aquilo que nos esperávamos, nós acreditamos que o motivo pelo qual obtivemos maus resultados para dois e três *qubits* tem a ver com o facto de termos usado a técnica de PCA, acreditamos que de alguma forma, ao usar em conjunto com o *amplitude encoding* origina perdas

de informação inesperadas ou introdução de ruído no nosso sistema diminuindo a *performance* do mesmo.

Se comparar mos os nossos resultados com os resultados em[5], podemos ver que o autor possui melhores resultados que nos, especialmente para o caso de 2 *qubits* onde a diferença é mais significativa (para o ZZFeatureMap), contudo, se compararmos os resultados para o caso em que usamos 4 *qubits* e *amplitude encodding* como encoding, podemos reparar que possuímos melhores resultados que o autor consegue para 3, para 2 os resultados são parecidos, mas os resultados do autor são ligeiramente melhores, apesar das diferenças óbvias entre os modelos, o autor apenas usa 2 e 3 *qubits* enquanto nos usamos 4, e os *data sets* em questão também são diferentes.

Após analisar os gráficos, tabelas e as matrizes de confusão, podemos facilmente ver que apenas para o *amplitude encode* com 4 *qubits* é que os resultados sao coerentes, isto é, apenas para este caso é que a matriz de confusão e a tabela das *accuracy* batem certo, levando nos a querer que para os outros modelos, o classificador não classificava com um grau de certeza significativo.

Nas simulações com ruído como seria de esperar temos um impacto significativo no desempenho do nosso modelo, o objetivo com esta simulação é tentar obter resultados que sejam o mais parecidos com os que obteríamos usando uma máquina real.

De uma forma geral, apesar de os resultados não serem os melhores, julgamos que o trabalho foi bem desenvolvido, infelizmente devido a nossa má gestão do tempo da nossa parte não conseguimos apresentar todos os resultados que pretendíamos, contudo, com os resultados que temos já e possível formular algumas hipóteses sobre o funcionamento do nosso modelo.

References

[1] A Step-by-Step Explanation of Principal Component Analysis. `https://builtin.com/data-science/step-step-explanation-principal-component-analysis`. Accessed: 2022-04-14.

[2] Maria Schuld and Francesco Petruccione. *Supervised learning with quantum computers*, volume 17. Springer, 2018.

[3] Building a Quantum Variational Classifier Using Real-World Data. `https://medium.com/qiskit/building-a-quantum-variational-classifier-using-real-world-data-809c59eb17c2`. Accessed: 2022-04-09.

[4] IBM Quantum Services quantum machines. `https://quantum-computing.ibm.com/services?services=systems`. Accessed: 2022-04-13.

[5] Sevak Mardirosian. *Quantum-enhanced Supervised Learning with Variational Quantum Circuits*. PhD thesis, PhD thesis. Leiden University, 2019.

[6] TorchRunTime. `https://qiskit.org/documentation/machine-learning/tutorials/06_torch_runtime.html`. Accessed: 2022-06-14.

[7] IBM Quantum Services RunTime. `https://quantum-computing.ibm.com/services?services=runtime`. Accessed: 2022-06-13.

[8] IBM Quantum Services System ibmq_manila. `https://quantum-computing.ibm.com/services?services=systems&system=ibmq_manila`. Accessed: 2022-06-13.

[9] Wine Data Set red and white wine. `https://archive.ics.uci.edu/ml/datasets/Wine+Quality?spm=a2c4e.11153940.blogcont603256.15.333b1d6fYOsiOK`. Accessed: 2022-04-13.

[10] Red Wine vs White Wine: The Real Differences. `https://winefolly.com/tips/red-wine-vs-white-wine-the-real-differences/`. Accessed: 2022-04-10.