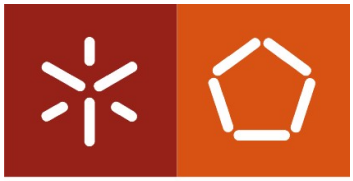
Classificação com Circuitos Variacionais Quânticos

Ciência de Dados Quântica

Mestrado em Engenharia Física
Márcio Mano, pg47446
Pedro Teixeira, a75049

14 abril 2022

Docentes:
Luís Paulo Santos
André Sequeira



Universidade do Minho

Contents

Comentário geral: Bom trabalho ! Falta apenas clarificar um pouco mais qual será a estratégia no que toca ao bloco de medição e pós -processamento. Dito isto, posteriormente será necessário ajustar a loss/cost function em função do tipo de medição que estão a fazer. A otimização dos parametros variacionais também precisa de ser melhorada, pois não ficou muito claro como pretendem otimizar o vosso circuito variacional.
Também existem umas duvidas relativamente ao amplitude encoding, no que toca à normalização dos dados. Não creio que seja ssim tão problemático. Mais abaixo têm uns comentários e sugestões para melhorarem o vosso encoding. Em suma, bom trabalho !

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/embeding*, um bloco responsável pelo algoritmo variacional e blocos de medição.

- Na parte clássica temos o pós-processamento (associa as medições a rótulos), o cálculo das *losses functions* e o otimizador dos parâmetros theta;

Estas duas partes ficam conectados em *loop* criando assim um sistema híbrido entre a computação quântica e computação clássica.

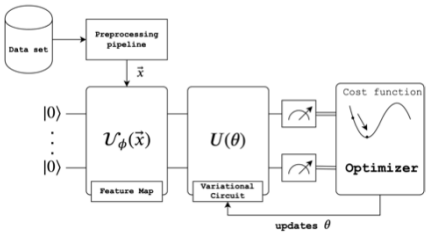


Figure 1: Exemplo de sistema híbrido.

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 dividida 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[1].

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

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

Com este trabalho pretendemos utilizar um classificador que seja *Hardware Efficient*, isto porque, pretendemos utilizar os computadores quânticos disponibilizados pela **IBM**[2] para correr o nosso sistema. Portanto, encontrámo-nos limitados em certos aspetos quando comparados com simuladores de máquinas quânticas, tais como:

- Só temos acesso a máquinas com 5 *qubits*, máquinas com maior número de *qubits* não se encontram publicamente disponíveis;

- Possuímos um tempo de coerência limitado, portanto, não podemos elaborar circuitos muito longos/profundos, ou os nossos *qubits* iram perder a informação a eles associada;

Ao longo do documento expomos outros objetivos secundários, que também pretendemos cumprir, isto, pois podemos não conseguir realizar o nosso objetivo principal de criar um sistema que consiga ser corrido numa máquina quântica real.

Contudo, o verdadeiro objetivo com este trabalho é sermos introduzidos ao mundo de *machine learning*, mais própriamente ao mundo de *machine learning* quântico, pois quem sabe se esta não será uma área onde possamos trabalhar no futuro, visto que é uma área em constante crescimento que tem alcançado excelentes resultados nos últimos anos.

3 *Data Set*

O primeiro passo antes de começarmos a desenhar qualquer sistema de *machine learning*, é a escolha do *data set* sobre o qual iremos trabalhar.

No nosso caso temos de selecionar um *data set* clássico ao qual seja possível codificar os seus elementos em estados quânticos, pois uma máquina quântica só consegue trabalhar com *qubits*, que ao contrário dos bits da computação clássica, que ora são 0 ou 1, os *qubits* podem estar num sobreposição dos dois estados, ou seja, 0 e 1 simultâneamente (grande parte da vantagem quântica advêm deste fenómeno).

Portanto, uma das dificuldades que sentimos na escolha de um *data set* foi facto de o mesmo não poder ser muito complexo, por exemplo, não podiamos escolher um *data set* onde os *data points* fossem imagens, pois transformar pixeis ou valores *RGB* em 0s e 1s não é uma tarefa fácil,

da mesma forma que também não podíamos escolher *data sets* com muitas *features*/atributos, pois não íamos ter *qubits* suficientes para codificar aos nossos dados.

3.1 *Wine Quality* Data Set

O *data set* selecionado pode ser encontrado no *link* seguinte[3].

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)

Este *data set* possui 12 atributos, sendo que um deles é, no nosso entender, um atributo subjetivo (qualidade do vinho), portanto um dos objetivos secundários do nosso projeto é verificar a influência que este atributo subjetivo tem no nosso classificador.

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[4].

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

Durante a elaboração do nosso sistema vamos aumentando progressivamente o número de *data points* que fornecemos ao nosso sistema durante a fase de treino, ou seja, vamos fazer varias sessões de treino, e de teste, aonde vamos aumentando o tamanho do *data set* de treino, e consequentemente aumentar linearmente o *data set* de teste (30% do tamanho do *data set* de treino), de modo a encontrarmos os pontos de *underfitting* e *overfitting* do nosso sistema.

3.2 Justificação para a escolha deste *data set*

Resolvemos escolher este *dataset* por 3 motivos:

1. Possui poucos atributos/features, quando comparado com os outros *data sets* que encontramos;
2. Possuímos muitos dados, portanto, em princípio, não corremos o risco de fazer *underfitting* ao nosso sistema, sendo que o *overfitting* é mais fácil de resolver que o *underfitting*.
3. Encontramos uma tese de um aluno da Holanda que tenta fazer um classificador como o nosso usando o mesmo *data set* e nós queremos comparar os nossos resultados experimentais com os que se encontram neste documento[5].

4 Encoding/Embeding

Para o *encoding* do nosso conjunto de dados avaliamos diferentes opções:

1. **Basis Encoding**;

   Não recomendado, visto que cada *data point* têm 11 ou 12 atributos com valores decimais, portanto estar a converter todos os valores para binário para codificar nas bases quânticas, iria significar que teríamos de usar vários *qubits*.

2. **Amplitude Encoding**;

   Provavelmente não será o *encoding* ideal, pois nos casos onde diferentes atributos do mesmo *datapoint* possuem uma grande diferença entre eles a normalização dos nossos atributos fará com que os atributos de menor valor sejam quase desprezados, mas como é dos encodes que utiliza menos *qubits* será o primeiro que implementaremos, pois 2^n = Num. de atributos, onde n é o número de *qubits*.

3. **Angle Encoding**;

   Provavelmente não será a melhor abordagem uma vez que para cada atributo do nosso *datapoint* precisamos de 1 *qubit*, ou seja para o nosso dataset precisaremos de 12 qubits para cada *datapoint*, o que basicamente fazemos nesta codificação (*encoding*) é fazer rotações em cada *qubit* de acordo com o valor do atributo correspondente a esse *qubit* (limitando a gama de variação de cada *feature* a um intervalo de $-\pi$ e π).

4. **Higher order Encoding**;

   Provavelmente não será a melhor abordagem dado que para cada atributo do nosso *datapoint* precisamos de 1 *qubit* e aumenta significativamente a profundidade do nosso circuito, pois não só fazemos as rotações de cada parâmetro mas também o *entanglement* dos nossos *qubits* juntamente com as rotações do produto dos atributos. A tudo isto é adicionado a possibilidade de fazer *data re-upload* aumentando ainda mais a profundidade do nosso circuito e consequentemente o ruido do nosso sistema.

5. **Feature Map**

   Seja $\mathcal{X}$ um dado *data set*. Uma *feature map* ϕ é uma função que funciona como $\phi : \mathcal{X} \to \mathcal{F}$, em que $\mathcal{F}$ é um *feature space*.

   Os resultados do mapeamento de cada elemento do *data set*, $\phi(x)$ para todo o $x \in \mathcal{X}$, são chamados de *feature vectores*.

   Em geral $\mathcal{F}$ é um espaço vetorial e ϕ é um *quantum feature map*, $\phi : \mathcal{X} \to \mathcal{F}$.

   De seguida apresentamos os diferentes tipos de *feature maps* que encontramos:

   1) **ZFeatureMap**

   "The first order Pauli Z-evolution circuit."

   É uma expansão diagonal de primeira ordem sendo implementada usando uma *ZFeature Map* onde $|S| = 1$.

   O circuito resultante não contém nenhuma interação (*entanglement*) entre as outras *features* do dado codificado.

   Data Map Function: $\phi_s : x \to x_i$

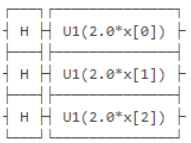


   Figure 2: Exemplo *ZFeature Map*[6]

   2) **ZZFeatureMap**

   "Second-order Pauli-Z evolution circuit."

   Este *ZZFeatureMap* permite mapenamento com $|S| \leq 2$, portanto interações entre as *features* de um dado será possível e ϕ é um função clássica não linear.

   Data Map Function:

$$\phi_s : x \to \begin{cases} x_i, & S = \{i\} \\ (\pi - x_i)(\pi - x_{xi}), & S = \{i, j\} \end{cases}$$

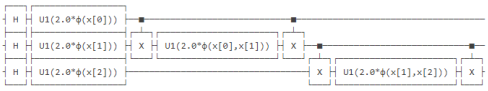


   Figure 3: Exemplo *ZZFeature Map*[6]

   3) **Pauli FeatureMap**

   Forma mais geral de um *feature map*.

   Permite a criação de *featura maps* usando diferentes gates. Podemos criar *feature maps* como o *ZFeatureMap*, ou como o *ZZFeatureMap*, entre outros.

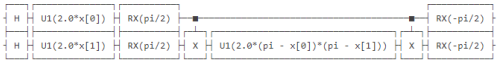


   Figure 4: Exemplo *Pauli Feature Map*[7]

4.1 Conclusão em relação ao Encode:

À primeira vista o melhor encoding para o nosso problema é o *Amplitude Encoding*, contudo não sabemos como funcionará a parte de normalização dos atributos, o nosso receio é que para atributos muito disparos/afastados tenhamos valores normalizados muito próximos de zero e isso possa ser um problema.

Após uma análise mais profunda surgiu a possibilidade de utilizar *Feature Maps* como *encoding*, sendo que temos à nossa disposição três tipos diferentes de *Feature Map*: *ZFeatureMap*, *ZZFeatureMap* e *Pauli FeatureMap*.

Contudo, utilizando uma *feature map* como encoding irá surgir o problema mencionado na secção 2, isto, pois para conseguirmos utilizar uma *feature map* iríamos precisar de 11/12 *qubits* e os computadores quânticos à nossa disposição só possuem 5 *qubits*, assim sendo temos de arranjar uma forma de reduzir a dimensionalidade do nosso *data set*, isto é, usar menos *features* mas sem perder a informação a elas associadas, é aí que entram as técnicas de *Principal Component Analysis*(PCA)[8].

Numa próxima fase de avaliação do nosso projeto serão apresentados os resultados atingidos com cada tipo de *encoding* bem com as técnicas de PCA[8].

5 Parameterized Model

Existem diferentes circuitos quânticos parametrizados que podemos aplicar no nosso sistema, alguns já se encontram predefinidos na biblioteca **qiskit**[9] como os abaixo enunciados:

- Two-local circuit;
- Hardware efficient SU(2) 2-local circuit;
- Pauli Two-Design ansatz;
- Real-amplitudes 2-local circuit;

Dentro destes circuitos enunciados o que mais nos chamou à atenção foi o ***Hardware efficient SU(2) 2-local circuit***[10] pelo facto de ser *Hardware Efficient*, contudo não temos qualquer tipo de garantias que o mesmo será uma boa opção para o nosso classificador, apesar disso existe uma quantidade de alternativas enormes para circuitos variacionais, nós podemos até mesmo desenhar o nosso próprio sistema variacional, tendo claro sempre o devido cuidado, com a profundidade dos circuios, bem como o *entenglement* para minimizar o ruído introduzido no nosso sistema, mas gratificando sempre uma boa expressibilidade do nosso sistema[11].

Aqui podemos ver um exemplo de um circuito variacional simples:

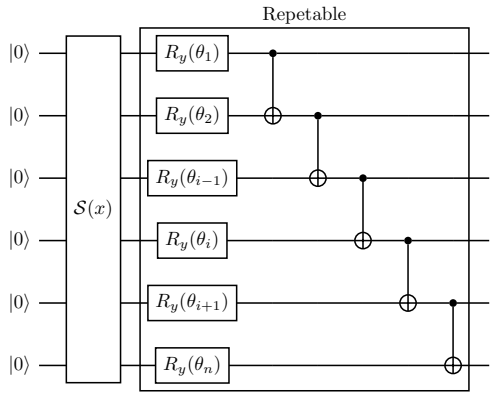


Figure 5: Exemplo de um circuito variaconal para 6 *qubits*, onde o bloco $\mathcal{S}(x)$ representa o bloco de encoding

Como podemos observar na imagem o bloco parametrizado pode ser repetido, isto obviamente aumenta a expressividade do nosso circuito[11], contudo o nosso objetivo é construir um circuito o menos profundo possível, ou seja, repetiremos o bloco parametrizado o mínimo de vezes possíveis até obtermos resultados aceitáveis tais como os do [5].

6 Circuit Measurements

Fazer medições e extrair labels das mesmas, existem pelo menos dois métodos de fazer medições e associar-lhes uma *label*/classe em problemas de *machine learning* quântico, mais propriamente problemas de **classificação binária**, como podemos observar em baixo.

- *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..

- *Mesurement of the first qubit*:

  Apenas temos em consideração a medição do primeiro *qubit*, e calculamos o seu valor expectável, $\langle z \rangle$. Se $\langle z \rangle$ for menor que 0 associamos uma classe, se for maior ou igual a 0 associamos a outra classe.

7 Loss and Cost Function

Loss Function:

$$L(x, y) = y - y^{truth}$$

Mean Squared Error (MSE).

$$C(\theta) = \frac{1}{n}\sum_{i=1}^{n}(f(x_1, \theta) - y^{truth})^2$$

Mean Absolute Error (MAE)

$$C(\theta) = \frac{1}{n}\sum_{i=1}^{n}|f(x_1, \theta) - y^{truth}|$$

8 Optimization Techniques

Para otimizar os nossos parâmetros temos de escolher uma técnica que calcule os melhores parâmetros θ. Uma dessas técnicas é o *Gradient ~~Descendent~~*.

- *Gradient ~~Descendent:~~* Para optimizar os parâmetros θ o que fazemos e a subtração entre a medição do circuito variacional $w(\theta+s)$ e a medição do circuito variacional $w(\theta-s)$, sendo que s tem que ser um valor pequeno, mas não muito pequeno.

  Assim o "gradiente" é:

$$\theta_{t+1} = \theta_t - \eta\vec{\nabla}(c(\theta))$$

No entanto, numa próxima fase de desenvolvimento do projeto outros otimizadores presentes no qiskit[12] serão testados.

9 Conclusion

"Talk is cheap. Show me the code"

-Linus Torvalds

References

[1] 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.

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

[3] 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.

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

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

[6] The first order Pauli Z-evolution circuit. https://qiskit.org/documentation/stubs/qiskit.circuit.library.ZFeatureMap.html#qiskit.circuit.library.ZFeatureMap. Accessed: 2022-04-13.

[7] The Pauli Expansion circui. https://qiskit.org/documentation/stubs/qiskit.circuit.library.PauliFeatureMap.html#qiskit.circuit.library.PauliFeatureMap. Accessed: 2022-04-13.

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

[9] Circuit Library qiskit.circuit.libary. https://qiskit.org/documentation/apidoc/circuit_library.html. Accessed: 2022-04-13.

[10] The hardware efficient SU(2) 2-local circuit efficientsu2. https://qiskit.org/documentation/stubs/qiskit.circuit.library.EfficientSU2.html#qiskit.circuit.library.EfficientSU2. Accessed: 2022-04-13.

[11] 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), 2019.

[12] Qiskit Optimizers. https://qiskit.org/documentation/stubs/qiskit.algorithms.optimizers.html. Accessed: 2022-04-14.