Aix*Marseille université
Initiative d'excellence

Sujet :

Variational Quantum Classifier for Binary Classification

Master 1 Informatique - Option Informatique Calcul Quantique (ICQ)

Année universitaire : 2022 - 2023

Groupe 8 :

MOUZNI Lamara

FELLOUSSI Idriss

SMAIL Aghilas

RANGHEARD Leo

RÉSUMÉ:

Dans ce rapport, nous explorons l'utilisation des circuits quantiques variationnels (VQC) dans les tâches de classification binaire. Plus précisément, nous utilisons Qiskit, la librairie quantique à source ouverte, pour mettre en œuvre un VQC permettant de classer l'ensemble de données Iris en deux espèces. Notre approche consiste à encoder les données à l'aide d'un encodage basé sur la rotation et à optimiser les paramètres du circuit à l'aide de l'optimiseur L-BFGS-B. Nous évaluons les performances de notre approche en mesurant la précision de nos prédictions sur un ensemble de tests.

La classification binaire quantique par circuit variationnel est une application prometteuse de l'informatique quantique qui peut potentiellement surpasser les algorithmes classiques d'apprentissage automatique. Dans ce rapport , nous présentons une implémentation de la classification binaire VCQ en utilisant Qiskit. Nous commençons par expliquer les bases et les principes des circuits quantiques variationnels. Nous décrivons ensuite notre implémentation en détail, y compris notre méthode d'encodage, l'architecture du circuit variationnel et l'algorithme d'optimisation. Nous concluons en présentant nos résultats et en discutant des implications de notre mise en œuvre.

I- Encodage des données :

On a choisi l'encodage de rotation (qui est plus proche de l'encodage Pauli).
En premier lieu, nous avons déterminé le nombre de qubits nécessaires pour le circuit en fonction du nombre de colonnes dans le tableau de données d'entrée X (iris), le nombre de qubits est égale au nombre de features dans le dataset.

Avant de passer à l'encodage, nous avons besoin de normaliser les valeurs dans le dataset (***m = (x -xmin) / (xmax -xmin)***) en utilisant la méthode **minmaxscaler()** de scikit-learn.preprocessing et les rendre comprisent entre 0 et 1, pour que par la suite on les multiplient par 2π afin de déterminer l'angle de rotation.Ensuite , nous avons défini un registre quantique et un registre classique pour le circuit en utilisant les classes **QuantumRegister** et **ClassicalRegister** de **Qiskit**, respectivement.

Par la suite, on itère pour chaque qubit et chaque point de données dans le tableau d'entrée **X**. Pour chaque point de données, on applique une rotation autour de l'axe y au qubit correspondant, l'angle de rotation étant proportionnel à la valeur du point de données. Ceci est fait en utilisant la porte **Ry** de **Qiskit**, où l'angle de rotation est spécifié en radians.

II - Circuit Quantique Variationnel :

Le circuit quantique de notre code est un circuit quantique variationnel, qui est un type de circuit quantique conçu pour être optimisé en ajustant ses paramètres afin de minimiser une fonction de coût. Le circuit se compose d'un ensemble de portes quantiques, dont chacune effectue une opération spécifique sur un ou plusieurs qubits, et est paramétré par un ensemble d'angles de rotation.

Le circuit est divisé en plusieurs couches, chacune d'entre elles étant constituée d'un ensemble de portes qui agissent sur tous les qubits du circuit. La première couche de portes consiste en un ensemble de portes **Rx**, qui appliquent une rotation autour de l'axe X de chaque qubit selon un angle paramétré par la variable thêta. La deuxième couche de portes est constituée d'un ensemble de portes **Ry**, qui appliquent une rotation autour de l'axe Y de chaque qubit d'un angle paramétré par la variable thêta. La troisième couche de portes est constituée d'un ensemble de portes à deux qubits, en particulier les portes **CNOT** et **CZ**, qui appliquent une opération **NOT** ou **Z** contrôlée entre des paires de qubits, respectivement. La porte **CZ** est une porte à deux qubits qui effectue un déphasage sur le qubit cible lorsque le qubit de contrôle est dans l'état **|1**⟩. Enfin, le circuit se termine par des mesures de chaque qubit dans la base de calcul, qui sont utilisées pour obtenir la sortie finale du circuit.

Le but de ce circuit est d'effectuer une tâche de classification binaire, où les données d'entrée sont encodées sous forme de rotations des qubits, et le circuit est entraîné à produire l'étiquette de classe correcte pour chaque entrée.

III- Optimization:

Dans l'étape d'optimisation nous utilisons un algorithme d'optimisation classique pour trouver les valeurs optimales du paramètre thêta qui minimisent le coût de la fonction **cost()** . Ceci en utilisant la méthode **minimize()** du module **scipy.optimize** est utilisée à cette fin.

Nous initialisons d'abord aléatoirement la valeur de **thêta** à l'aide de la fonction **random.uniform()** qui retourne une valeur aléatoire entre 0 et 2π. Ensuite, nous passons cette valeur initiale, ainsi que d'autres paramètres d'optimisation tels que la méthode, les limites et la tolérance à la fonction **minimize()**. La méthode utilisée pour l'optimisation est **'L-BFGS-B'**, qui est une méthode quasi-Newton à mémoire limitée avec des limites. Les limites sont spécifiées à l'aide de la fonction **Bounds()** qui garantit que l'algorithme d'optimisation ne choisit pas de valeurs de thêta en dehors de la plage de 0 à 2π.

L'objet résultat obtenu à partir de la fonction **minimize()** contient la valeur optimale de **thêta** qui minimise la fonction de coût. L'attribut x de l'objet résultat contient la valeur optimale de theta., ensuite cette valeur optimale de **thêta** est liée au circuit à l'aide de la méthode **bind_parameters()** de l'objet circuit.

IV- Prédiction:

Les prédictions sont effectuées à l'aide du circuit lié. Pour chaque instance de test, les caractéristiques d'entrée codées sont utilisées pour définir l'état initial des qubits à l'aide de la porte **RY**, puis le circuit est lié avec les valeurs de paramètres optimisées.
Après avoir lié le circuit, la méthode **run()** de **Aer.get_backend('qasm_simulator')** est utilisée pour simuler le circuit sur le backend du simulateur quantique, avec un nombre fixe de tirs (10 000). La méthode **transpile()** est utilisée pour transformer le circuit en une forme qui peut être exécutée sur le backend du simulateur.
La méthode **result()** est appelée sur la sortie de la méthode **run()** pour obtenir le résultat de la simulation, qui est un dictionnaire contenant le nombre de fois où chaque résultat s'est produit.
Les valeurs attendues sont calculées à partir des nombres à l'aide de la formule mentionnée ci-dessus, et la classe prédite est celle dont la valeur attendue est la plus élevée. Enfin, la classe prédite est ajoutée à la liste ***'y_pred'*** qui sera utilisée pour évaluer le modèle.

V- Résultats:

Un taux de précision de 56 % peut être considéré comme faible ou médiocre en fonction du contexte du problème c'est là où nous avons rencontrer le plus de problème parce qu'il s'avère que le classifieur ne fait pas un bon travaille même si nous changeons la méthode d'encodage de Pauli à l'encodage de rotation , et même si nous changeons la méthode d'optimisation (Adam , adagrad…) à l'aide de scipy , nous n'avons pas pu avoir un taux de précision raisonnable .