Revista Brasileira de Ensino de Física, vol. 46, e20240230 (2024)
www.scielo.br/rbef
DOI: https://doi.org/10.1590/1806-9126-RBEF-2024-0230

Artigos Gerais

Introdução ao aprendizado de máquina quântico, suas aplicações e vantagens

Introduction to quantum machine learning, its applications and advantages

Naomy Duarte Gomes[*1], Jimi Togni[2], Bruno Aparecido Cazotti Ramalho[3,4], Kleython José Coriolano Cavalcanti de Lacerda[3,4], Paulo Henrique Ferreira[5]

[1]Universidade de São Paulo, Instituto de Física de São Carlos, 13566-590, São Carlos, SP, Brasil.
[2]Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e da Computação, 13083-852, Campinas, SP, Brasil.
[3]Universidade de São Paulo, Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Departamento de Física, 14040-901, Ribeirão Preto, SP, Brasil.
[4]Universidade de São Paulo, Faculdade de Medicina de Ribeirão Preto, Departamento de Imagem Médica, Hematologia e Oncologia Clínica, 14049-900, Ribeirão Preto, SP, Brasil.
[5]Universidade Federal da Bahia, Instituto de Matemática e Estatística, 40170-110, Salvador, BA, Brasil.

Recebido em 19 de junho de 2024. Revisado em 30 de julho de 2024. Aceito em 17 de agosto de 2024.

Neste artigo, abordamos o campo emergente do aprendizado de máquina quântico (AMQ) e suas aplicações inovadoras. Exploramos uma visão geral das bases da mecânica quântica relevantes para o aprendizado de máquina, destacando como os princípios quânticos podem ser utilizados para processar informações de maneira mais eficiente em comparação às abordagens clássicas. Discutimos o passo a passo de um exemplo de algoritmo quântico utilizando Qiskit, comparando-o com seu análogo clássico. Abordamos as vantagens do AMQ, incluindo o potencial de aceleração em problemas de grande escala e a capacidade de lidar com dados altamente dimensionais. Por fim, são discutidos os desafios atuais e as perspectivas futuras do campo, enfatizando seu papel na transformação de diversos setores tecnológicos. Este artigo serve como uma introdução abrangente para aqueles interessados em explorar a interseção entre aprendizado de máquina e mecânica quântica, destacando as promissoras oportunidades que essa combinação oferece.
Palavras-chave: Aprendizado de máquina, Computação quântica, Algoritmos.

In this article, we delve into the emerging field of quantum machine learning (QML) and its innovative applications. We provide an overview of the fundamentals of quantum mechanics relevant to machine learning, highlighting how quantum principles can be harnessed to process information more efficiently than classical approaches. We discuss the step-by-step process of a quantum algorithm example using Qiskit, comparing it with its classical counterpart. We address the advantages of QML, including the potential for acceleration in large-scale problems and the ability to handle highly dimensional data. Finally, current challenges and future prospects of the field are discussed, emphasizing its role in transforming various technological sectors. This article serves as a comprehensive introduction for those interested in exploring the intersection of machine learning and quantum mechanics, highlighting the promising opportunities that this combination offers.
Keywords: Machine Learning, Quantum Computing, Algorithms.

1. Introdução

Nos últimos anos, o campo do aprendizado de máquina (AM) tem testemunhado avanços notáveis que redefiniram a forma como lidamos com dados e tomamos decisões. Originado das raízes da inteligência artificial [1, 2], o AM vem evoluindo rapidamente, movendo-se além de simples análises estatísticas para abordagens mais sofisticadas e automatizadas. Esse progresso tem sido impulsionado por um aumento exponencial na disponibilidade de dados e poder computacional, permitindo que algoritmos aprendam padrões complexos e façam previsões precisas. A habilidade de máquinas em aprender padrões e melhorar seu desempenho com o tempo, sem programação explícita, se mostra revolucionária em diversos setores, desde a medicina até a indústria.

As vantagens do AM são amplas e impactantes [3]. A capacidade de analisar grandes volumes de dados em tempo real, identificar tendências ocultas e gerar *insights* valiosos tem se traduzido em melhorias significativas na tomada de decisões empresariais e científicas. Além disso, a automação de tarefas rotineiras [4, 5] e a detecção de anomalias [6–8] têm resultado em maior eficiência operacional. O AM também tem se mostrado vital em aplicações complexas, como diagnóstico médico [9–12], previsão climática [13, 14] e desenvolvimento de veículos autônomos [15]. No entanto, à medida que

*Endereço de correspondência: naomy.gomes@usp.br

enfrentamos desafios cada vez mais complexos, surgiu um novo horizonte promissor no campo: o aprendizado de máquina quântico (AMQ) [16–18]. Este representa um avanço revolucionário que une os princípios da mecânica quântica com as técnicas do AM tradicional. Ao explorar a natureza única da computação quântica, que aproveita estados superpostos e emaranhamento, o AMQ demonstra um potencial extraordinário para lidar com problemas que desafiam a capacidade dos computadores clássicos. O poder de processamento exponencialmente mais rápido e a habilidade de explorar diversas soluções simultaneamente abrem portas para abordagens inovadoras em áreas como otimização, simulação molecular e análise de grandes conjuntos de dados [19]. Mais recentemente, foi demonstrado [20] que o AMQ pode fornecer como vantagem a capacidade de lidar com um conjunto de dados limitado, ainda assim apresentando boa performance. À medida que essa tecnologia emergente ganha destaque, é imperativo explorar suas vantagens, utilidade e perspectivas, definindo um novo paradigma no campo do AM e estendendo suas aplicações a fronteiras anteriormente inexploradas.

Neste trabalho, queremos demonstrar o passo a passo da construção de um algoritmo de AMQ para um caso simples e bem conhecido, que é o de classificação com a base de dados Iris, utilizando o *framework* Qiskit (*Quantum Information Software Development Kit*) [21–23]. O Qiskit é um conjunto de ferramentas de *software* de código aberto desenvolvido pela IBM para programação, simulação e execução de algoritmos quânticos em computadores quânticos reais ou simulados. Ele fornece uma interface de programação amigável para desenvolver e executar algoritmos quânticos em diferentes plataformas de computação quântica. Com isso, visamos proporcionar uma compreensão dos princípios da computação quântica e suas aplicações na resolução de problemas complexos. Além disso, vamos explorar as potenciais vantagens e desafios da computação quântica em relação à computação clássica.

Nosso trabalho está dividido como segue: na Seção 2 descrevemos o que é e quais os tipos de AM e definimos quais métricas podem ser utilizadas para validar a eficiência de um modelo; na Seção 3 descrevemos as bases da computação quântica e os conceitos a ela subjacentes; na Seção 4 apresentamos algoritmos de AM e AMQ, definindo o passo a passo de cada um deles, comparando suas vantagens e desvantagens. Por fim, fazemos um levantamento de tudo que foi demonstrado e as conclusões na Seção 5.

2. Aprendizado de Máquina

As técnicas de modelagem preditiva têm desempenhado um papel fundamental na análise de dados e na tomada de decisões em diversos setores. Essas técnicas envolvem a utilização de algoritmos e métodos estatísticos para criar modelos que possam inferir padrões de dados e, assim, prever o comportamento futuro de determinado fenômeno ou variável. Dentro do AM, essas técnicas podem ser divididas em três categorias principais, que são o aprendizado supervisionado [24, 25], o aprendizado não supervisionado [26] e o aprendizado por reforço [27] (ver Figura 1). A primeira abordagem consiste em fornecer dados reais rotulados, permitindo ao modelo aprender a mapear os padrões e relacionamentos entre esses dados de entrada e as saídas esperadas [28]. Essa abordagem pode ser exemplificada por técnicas como regressão linear [29, 30], regressão logística [31, 32] e redes neurais artificiais [33, 34]. Na segunda abordagem, não há rótulos nos dados fornecidos, e o objetivo é encontrar estruturas, padrões ou agrupamentos intrínsecos nos dados sem informações prévias sobre as saídas esperadas. As duas técnicas mais destacadas nessa abordagem são a associação e a clusterização [35]. A terceira abordagem envolve um *feedback* positivo ou negativo conforme o modelo aprende por meio de tentativa e erro. Os algoritmos de *Q-learning* [36, 37] e SARSA (do inglês *State-Action-Reward-State-Action*) [38] são duas das técnicas mais utilizadas no aprendizado por reforço.

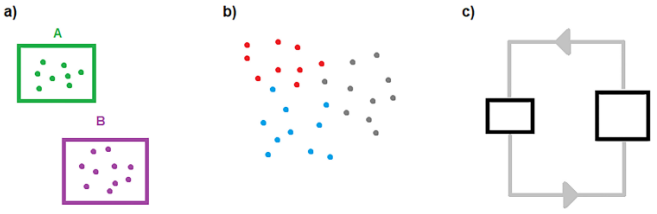

Figura 1: Tipos de aprendizado de máquina: a) supervisionado, b) não supervisionado e c) por reforço.

Dentre as técnicas de aprendizado supervisionado, destacam-se as redes neurais artificiais, que são modelos computacionais inspirados na capacidade de processamento do sistema nervoso biológico [39]. O primeiro modelo de neurônio artificial foi proposto por McCulloch e Pitts em 1943, visando englobar as principais características de uma rede neural biológica: paralelismo e alta conectividade [40]. Em 1958, Frank Rosenblatt descreveu a forma mais simples de uma rede neural artificial, o perceptron [41]. Um perceptron processa a informação da seguinte forma: recebe um vetor de entradas $\mathbf{X}$, representando as informações a serem mapeadas pelo modelo, e atribui a cada entrada um peso sináptico $\mathbf{W}$, que é atualizado durante o treinamento para otimizar a classificação. Os valores de $\mathbf{X}$ e $\mathbf{W}$ são então combinados linearmente, somando-se também um viés (*bias*). Se a soma ponderada mais o viés supera um limiar de ativação θ, a informação é passada para uma função de ativação apropriada, que gera a saída $\mathbf{y}$ do modelo. Os pesos sinápticos são atualizados com base na diferença entre a saída esperada (classe real) e a saída obtida (classe prevista).

Devido à simplicidade do perceptron, ele só consegue classificar problemas linearmente separáveis. Para

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2024-0230

resolver problemas não linearmente separáveis, foram introduzidas camadas intermediárias (ocultas) nas redes neurais artificiais, resultando nos Perceptrons Multicamadas (PMC). Assim como nos perceptrons simples, as entradas nos PMCs são associadas aos pesos sinápticos e processadas pela primeira camada neuronal, que gera uma saída. Essa saída é então associada a um novo peso sináptico e serve como entrada para a próxima camada neuronal, até chegar à camada de saída. O número de camadas ocultas e de neurônios em cada camada varia conforme o problema a ser resolvido.

Os dados fornecidos ao modelo podem conter variáveis categóricas e/ou numéricas. Quando a variável de saída é numérica, utilizam-se técnicas de regressão, que estimam um valor numérico de saída com base em um conjunto de variáveis de entrada [42]. Se a variável de saída é categórica, utilizam-se técnicas de classificação para prever a classe ou categoria da variável de saída [43]. Essas técnicas, uma vez aplicadas a um conjunto de dados, precisam de métricas de comparação de desempenho, de modo que possamos ter uma medida objetiva e quantitativa de quão bem um modelo está realizando as previsões desejadas [44, 45]. O primeiro passo é construir uma matriz de confusão, que simplifica a visualização dos resultados brutos dos algoritmos de classificação. A matriz de confusão é uma tabela que indica os erros e acertos do modelo, comparando a classificação prevista com a esperada. Além disso, as métricas de comparação de desempenho também podem ser usadas para selecionar os recursos mais relevantes para um modelo e para comparar o desempenho do modelo com diferentes configurações de hiperparâmetros, auxiliando, assim, na escolha dos melhores valores para eles. Algumas das métricas mais conhecidas para problemas de classificação são: acurácia, precisão, *recall* e F1-*Score* [46–48].

A acurácia mede a proporção de exemplos classificados corretamente em relação ao total de exemplos no conjunto de dados de teste, indicando a performance global do modelo ao mostrar a porcentagem de previsões corretas feitas. A precisão avalia quantas das classificações de uma determinada classe foram feitas corretamente pelo modelo, enquanto o *recall* (ou revocação/sensibilidade) mede quantas das ocorrências esperadas de uma classe específica foram corretamente identificadas. O F1-*Score*, uma média harmônica entre precisão e *recall*, oferece uma medida equilibrada do desempenho do modelo. Todas essas métricas variam de 0 a 1, sendo que valores mais próximos de 1 indicam melhor desempenho. Para problemas de classificação multiclasse, essas métricas são aplicadas a cada classe e depois agregadas para fornecer uma avaliação geral do desempenho do modelo. As equações a seguir trazem a definição matemática de cada uma dessas métricas citadas:

$$Acurácia = \frac{VP}{VP + FN + VN + FP} \times 100\%, \quad (1)$$

$$Precisão = \frac{VP}{VP + FP} \times 100\%, \quad (2)$$

$$Recall = \frac{VP}{VP + FN} \times 100\%, \quad (3)$$

$$F1 - Score = 2 * \frac{Precisão * Recall}{Precisão + Recall} \times 100\%, \quad (4)$$

nas quais VP, VN, FP e FN, respectivamente, representam verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos.

O R-quadrado (R^2) [49], para tarefas de regressão, mede a proporção da variabilidade dos dados que é explicada pelo modelo. Varia de 0 a 1, em que 1 indica um ajuste perfeito do modelo aos dados. Matematicamente, temos:

$$R^2 = 1 - \sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{(\hat{y}_i - \bar{y})^2}, \quad (5)$$

em que y_i são os valores reais, $\bar{y}$ é a média dos valores reais e $\hat{y}_i$ são os valores preditos pelo modelo.

3. Bases da Computação Quântica

A computação quântica representa uma abordagem revolucionária para o processamento de informações, fundamentada nos princípios da mecânica quântica [50, 51]. Ao contrário da computação clássica, que utiliza bits tradicionais como unidades fundamentais de informação, a computação quântica se baseia em bits quânticos, ou qubits [52], que podem existir em estados superpostos e emaranhados. Essa capacidade única permite que os computadores quânticos realizem cálculos exponencialmente mais rápidos do que seus equivalentes clássicos em certas tarefas específicas. As bases teóricas da computação quântica incluem, portanto, conceitos como qubits, superposição, emaranhamento, interferência quântica e decoerência, que serão exemplificados a seguir. Para uma abordagem mais completa, o leitor pode consultar o trabalho de Alves e Felipe [53] e de Jesus et al. [22].

Qubit: Enquanto a computação clássica usa bits para representar informações como 0 ou 1, a computação quântica utiliza qubits $|0\rangle$ e $|1\rangle$, que podem existir em múltiplos estados ao mesmo tempo, graças ao fenômeno de superposição [50]. Isso permite que eles realizem cálculos em paralelo, proporcionando uma vantagem potencial na resolução de certos problemas. Matematicamente, para um estado quântico puro, temos uma função de onda $|\Psi\rangle$ dada por:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle, \quad (6)$$

em que α e β são amplitudes de probabilidade, que podem ser números complexos, e possuem a seguinte relação:

$$|\alpha|^2 + |\beta|^2 = 1, \quad (7)$$

o que garante que a probabilidade de medir um estado ou outro seja 1.

Superposição: Um qubit pode estar em uma combinação linear de seus estados $|0\rangle$ e $|1\rangle$, como mostra a Equação (6), o que significa que ele pode apresentar ambos os valores simultaneamente, permitindo assim que os computadores quânticos processem informações de maneira paralela e mais ágil.

Emaranhamento: Qubits emaranhados estão correlacionados de tal forma que o estado de um qubit não pode ser descrito independentemente do estado do outro. Como os qubits estão fortemente correlacionados, mesmo que eles estejam separados por uma grande distância, as medições feitas em um afetarão instantaneamente o estado do outro. Matematicamente, temos:

$$|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \tag{8}$$

em que os estados $|00\rangle = |0\rangle \otimes |0\rangle$ e $|11\rangle = |1\rangle \otimes |1\rangle$ são estados emaranhados, o que significa que, se uma medida resultar em $|0\rangle$ ($|1\rangle$), a outra necessariamente será $|0\rangle$ ($|1\rangle$).

Portas Quânticas: Assim como as portas lógicas em circuitos clássicos, as portas quânticas são operadores que atuam nos qubits para realizar operações como inversões, rotações e combinações lineares. Cada porta quântica é representada por uma matriz unitária, que descreve a transformação linear que ela aplica ao estado do qubit. Essas matrizes unitárias são fundamentais para o cálculo das operações que podem ser realizadas em um computador quântico [54, 55].

Interferência Quântica: Esse conceito se relaciona à propriedade de interferência, na qual diferentes caminhos que um qubit pode seguir se combinam, interferindo construtiva ou destrutivamente, dependendo das fases dos caminhos. Isto ocorre devido à natureza ondulatória da função de onda e é fundamental para algoritmos como o de busca de Grover [56].

Medição Quântica: Ao medir um qubit, dizemos que sua função de onda colapsa de seu estado de superposição da Equação (6) em um estado clássico 0 ou 1, o que permite obter os resultados finais observáveis após os cálculos quânticos.

Decoerência e Correção de Erros: Qubits são suscetíveis a ruídos e decoerência, esta sendo uma espécie de perda de informação do sistema para o ambiente, o que pode levar a erros. A correção quântica de erros é uma área ativa de pesquisa para garantir a confiabilidade dos cálculos quânticos [57, 58].

4. Exemplos de Algoritmos de Aprendizado de Máquina

4.1. Aprendizado de máquina clássico

Diferentemente dos métodos tradicionais de programação, em que os algoritmos são explicitamente definidos pelo programador, no AM os algoritmos são projetados para aprender automaticamente com os dados disponíveis, adaptando-se e melhorando seu desempenho ao longo do tempo.

Um dos principais pilares do AM é a utilização de modelos estatísticos para representar e compreender os dados [59]. Esses modelos podem variar desde técnicas simples, como regressão linear [29, 30] e árvores de decisão [60, 61], até modelos mais complexos, como redes neurais artificiais [33, 34] e máquinas de vetores de suporte [62, 63]. O processo de treinamento de um modelo de AM envolve a apresentação de dados de treinamento ao algoritmo, que ajusta seus parâmetros internos com base nos padrões observados nos dados, e uma posterior apresentação de dados de teste, de modo que o modelo faça a previsão em dados desconhecidos e ela possa ser comparada aos resultados reais.

Tomemos como exemplo a base de dados Iris [64], que é bastante conhecida. Esta base contém informações sobre amostras de três espécies de íris: setosa, virgínica e versicolor, consistindo em quatro variáveis de entrada, que são o comprimento da sépala da flor, a largura da sépala, o comprimento da pétala, a largura da pétala, e a variável classificadora, possuindo um total de 150 amostras, com 50 amostras para cada uma das três espécies. Este *dataset* é geralmente considerado simples por várias razões. Primeiramente, sua pequena escala, com apenas 150 amostras divididas em três classes igualmente distribuídas, facilita a manipulação e a compreensão dos dados. O conjunto de dados é caracterizado por uma baixa dimensionalidade, contendo apenas quatro características para descrever cada amostra, o que simplifica a análise. Além disso, as classes no conjunto de dados Iris tendem a ser bem definidas e separáveis, tornando a tarefa de classificação relativamente direta. Esses atributos tornam este conjunto uma escolha comum para introdução a algoritmos de AM e demonstrações de conceitos de classificação.

Podemos escrever um algoritmo de AM que, para determinados valores das variváveis de entrada, ele preveja qual a espécie correspondente. Para isto, podemos utilizar o Google Colaboratory [65], de modo que não seja necessário para o usuário instalar nenhum pacote em seu computador. O código da Figura 2 começa importando os pacotes da biblioteca Python Scikit-learn [66] necessários para a construção do algoritmo (linhas 2 a 6). A seguir (linhas 9 a 11), o conjunto de dados Iris é carregado e separado entre variáveis de entrada e variável classificadora. Na linha 14, dividimos os dados aleatoriamente em conjuntos de treinamento (80%) e teste (20%), usando *random_state* para garantir a reprodutibilidade dos resultados do código. Nas linhas 17 a 19, usamos a função *MinMaxScaler()*, que transforma os dados para que fiquem dentro do intervalo [0,1], normalizando-os e evitando, assim, que características com valores maiores dominem o aprendizado. Utilizando como classificador uma árvore de decisão (*Decision Tree Classifier*, linha 22) [67, 68], cujo objetivo é dividir o

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2024-0230

```python
# Importando as bibliotecas necessárias
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import MinMaxScaler

# Carregando o conjunto de dados Iris
iris_data = load_iris()
X = iris_data.data  # variáveis de entrada
y = iris_data.target  # variável classificadora

# Dividindo o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalização dos dados para ficarem na mesma escala
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Inicializando o classificador de árvore de decisão
dtc = DecisionTreeClassifier(random_state=42)

# Treinando o classificador no conjunto de treinamento
dtc.fit(X_train, y_train)

# Fazendo previsões no conjunto de teste
y_pred = dtc.predict(X_test)

# Calculando as métricas do classificador
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

# Imprimindo as métricas
print(f"Acurácia: {accuracy*100:.2f}%")
print(f"Precisão: {precision*100:.2f}%")
print(f"Recall: {recall*100:.2f}%")
print(f"F1-Score: {f1*100:.2f}%")
```

Figura 2: Código Python para carregar o conjunto de dados, treinar um algoritmo de AM usando árvore de decisão (*Decision Tree Classifier*) e obter as métricas de desempenho acurácia, precisão, *recall* e F1-*Score*.

```python
# Importando as bibliotecas necessárias
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# Carregando o conjunto de dados Iris
iris_data = load_iris()
X = iris_data.data  # variáveis de entrada
y = iris_data.target  # variável classificadora

# Convertendo os rótulos para one-hot encoding
y = to_categorical(y)

# Dividindo o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalização dos dados para ficarem na mesma escala
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Inicializando a rede neural com 2 camadas
model = Sequential()
model.add(Dense(10, input_dim=4, activation='relu'))  # Primeira camada oculta com 10 neurônios
model.add(Dense(3, activation='softmax'))  # Camada de saída com 3 neurônios (uma para cada classe)

# Compilando o modelo
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Treinando o modelo no conjunto de treinamento
model.fit(X_train, y_train, epochs=50, batch_size=5, verbose=1)

# Fazendo previsões no conjunto de teste
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
y_test_classes = y_test.argmax(axis=1)

# Calculando as métricas do classificador
accuracy = accuracy_score(y_test_classes, y_pred_classes)
precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')

# Imprimindo as métricas
print(f"Acurácia: {accuracy*100:.2f}%")
print(f"Precisão: {precision*100:.2f}%")
print(f"Recall: {recall*100:.2f}%")
print(f"F1-Score: {f1*100:.2f}%")
```

Figura 3: Código Python para carregar o conjunto de dados, treinar um algoritmo de AM usando rede neural e obter as métricas de desempenho acurácia, precisão, *recall* e F1-*Score*.

conjunto de dados em subconjuntos menores, com base nas características dos dados, de forma a fazer previsões precisas, nós o treinamos no conjunto de treinamento (linha 25) e fazemos previsões no conjunto de teste (linha 28). Calculamos as métricas de desempenho do modelo de acordo com as Equações (1) a (4), que já vêm implementadas no pacote *sklearn.metrics*, usando as previsões feitas (y_pred) e os valores reais (y_test). O parâmetro *average* especifica a estratégia a ser usada para calcular a métrica a partir dos resultados de cada classe, neste caso, *average='macro'* é usado, pois as classes têm o mesmo tamanho. Obtemos os valores de 97% para todas as métricas, o que mostra o excelente ajuste do modelo clássico aos dados. Para o *dataset* Iris, onde as classes são equilibradas e o problema é relativamente simples, destacamos que a acurácia serviria bem como uma métrica principal, fornecendo uma avaliação clara e direta da eficácia do modelo de classificação. Porém, em aplicações mais complexas ou em *datasets* com classes desbalanceadas, é importante olhar também a precisão, *recall* e F1-*Score* [69, 70].

Podemos, também, usar redes neurais [33, 34] como algoritmo de AM para prever as classes/espécies de íris. Aqui, as redes neurais serão utilizadas principalmente pela sua semelhança com os algoritmos de AMQ. A Figura 3 mostra o código utilizado para isto. Observemos que o código ficou maior e mais complexo, devido às características intrínsecas de uma rede neural. Neste código, começamos importando os pacotes já utilizados anteriormente, mas também *Sequential* e *Dense* da biblioteca Keras [71] para construir a rede neural e *to_categorical* para converter os rótulos em codificação *one-hot*, que atribui o valor 1 à classe correspondente e 0 às demais, facilitando o processamento pela rede neural (linhas 2 a 8). Nas linhas 11 a 13, carregamos o conjunto de dados e separamo-lo, como feito anteriormente, e na linha 16 fazemos a conversão das classes para *one-hot*. A rede neural é inicializada na linha 27 como uma rede neural sequencial, com uma camada oculta de 10 neurônios (linha 28) cuja dimensão de entrada é 4 ($input\_dim = 4$), pois há 4 variáveis neste *dataset*, e uma camada de saída de 3 neurônios (correspondente às 3 classes no conjunto de dados Iris - linha 29). A camada oculta usa a função de ativação *relu* [72, 73], que é uma função usada para introduzir não linearidades no modelo, e a camada de saída usa *softmax* [74], que transforma os valores de saída em probabilidades. Compilamos o modelo usando a perda *categorical_crossentropy* e o otimizador *adam* [75], treinando o modelo por 50 épocas com um tamanho de lote de 5, sendo a época uma passagem completa por todo o conjunto de dados de treinamento, e o lote o número de amostras de treinamento que o modelo processa antes de atualizar seus parâmetros. Por fim, fazemos as previsões no conjunto de teste (linhas 38 a 40) e calculamos as

métricas de desempenho comparando as classes previstas com as classes reais.

O modelo com a rede neural apresenta acurácia, *recall* e F1-*Score* de 93%, e precisão de 94%, ligeiramente menor do que as métricas obtidas com a árvore de decisão. Ressaltamos aqui que pode haver ligeiras variações nos valores das métricas, pois as redes neurais inicializam pesos e parâmetros de forma aleatória.

4.2. Aprendizado de máquina quântico

O AMQ representa um domínio inovador que integra os princípios da mecânica quântica com técnicas de AM, prometendo avanços significativos em uma ampla gama de aplicações. No AMQ, os sistemas computacionais são projetados para processar e analisar dados usando qubits, as unidades fundamentais de informação quântica, que possuem as propriedades descritas na Seção 3.

Novamente, nos voltaremos à base de dados Iris e aplicaremos o AMQ utilizando o *framework* Qiskit [23]. Para rodar o código no Google Colaboratory, é importante que antes sejam instalados alguns módulos:

- !pip install pylatexenc,
- !pip install qiskit_algorithms,
- !pip install qiskit_machine_learning.

Feito isto, começaremos de maneira semelhante ao feito para o caso do AM (ver Figura 4): primeiro importamos as bibliotecas necessárias, incluindo as classes e funções específicas para AMQ, bem como bibliotecas do *scikit-learn* para trabalhar com a base de dados Iris (linhas 2 a 9). Nas linhas 12 a 14, carregamos o conjunto de dados Iris, dividindo-o em variáveis de entrada e alvo, como feito na Figura 2 e, da mesma maneira, dividimos os dados em treino e teste na mesma proporção utilizada anteriormente de 80% e 20%, respectivamente (linha 17). Para garantir que todos os dados estejam na mesma escala, novamente fizemos a normalização (linhas 20 a 22) e, a partir deste ponto, inserimos no código a configuração do circuito quântico que será usado para mapear as variáveis dos dados de entrada para um espaço quântico (linhas 28 e 29). Este mapeamento ocorre primeiro pela transformação de bits (clássicos) em qubits (quânticos), depois por uma introdução de correlações quânticas entre os qubits. Notemos que, quando da configuração do circuito quântico, precisamos definir o número de qubits necessários. Este número é calculado de acordo com a quantidade de variáveis de entrada, o que, para o caso do *dataset* Iris, são 4 (notemos a semelhança com o *input_dim* das redes neurais). Isto é feito na linha 25 calculando-se o número de colunas dos dados de entrada de treinamento.

```
1  # Importando as bibliotecas necessárias
2  from qiskit_machine_learning.algorithms.classifiers import VQC
3  from qiskit.primitives import Sampler
4  from qiskit_algorithms.optimizers import COBYLA
5  from qiskit.circuit.library import ZZFeatureMap
6  from sklearn.model_selection import train_test_split
7  from sklearn.preprocessing import MinMaxScaler
8  from sklearn.datasets import load_iris
9  from qiskit.circuit.library import RealAmplitudes
10
11 # Carregando o conjunto de dados Iris
12 iris_data = load_iris()
13 X = iris_data.data # variáveis de entrada
14 y= iris_data.target # variável classificadora
15
16 # Dividindo o conjunto de dados em treinamento e teste
17 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
18
19 # Normalização dos dados para ficarem na mesma escala
20 scaler = MinMaxScaler()
21 X_train = scaler.fit_transform(X_train)
22 X_test = scaler.fit_transform(X_test)
23
24 # Definição do número de qubits
25 n_qubits = X_train.shape[1]
26
27 # Configurando o circuito quântico
28 feature_map = ZZFeatureMap(feature_dimension=n_qubits, reps=1)
29 feature_map.decompose().draw(output="mpl", style="clifford", fold=20)
```

Figura 4: Primeira parte do código Python de AMQ para carregar o conjunto de dados e configurar o mapeamento dos dados clássicos para estados quânticos.

A Figura 5 mostra o circuito quântico criado usando o *ZZFeatureMap* e que mapeia as características dos dados clássicos de entrada em um estado quântico. Podemos ver os quatro qubits gerados (q0, q1, q2 e q3), que representam as variáveis de entrada. Os quadrados azuis com a letra H são as portas de Hadamard [76], que transformam um único estado em uma superposição de dois estados base, essencial para explorar o espaço de estados quânticos. Logo após as portas H, temos os quadrados roxos com a letra P, que são portas de fase parametrizadas [76]. Estas são aplicadas a cada qubit com parâmetros baseados nas características dos dados clássicos e têm a forma $P(2{*}x[i])$, em que $x[i]$ representa a i-ésima variável de entrada. Depois de aplicadas as portas, começam as interações entre os qubits, que são introduzidas usando portas CNOT (+) [77] e portas de fase controladas (P) do tipo $P(2{*}(\pi - x[i])(\pi - x[j]))$, que representam a interação entre os qubits qi e qj, modulada pelas características $x[i]$ e $x[j]$. Estas operações introduzem emaranhamento entre os qubits, permitindo que o circuito capture correlações entre as características dos dados de entrada. As etapas aqui descritas (Hadamard, rotações de fase e interações) podem ser repetidas ou não. Aqui, escolhemos pela não repetição (o que é configurado ao se escolher *reps = 1* na linha 28 da Figura 4). Cada repetição visa aumentar a complexidade do estado quântico, permitindo capturar relações não

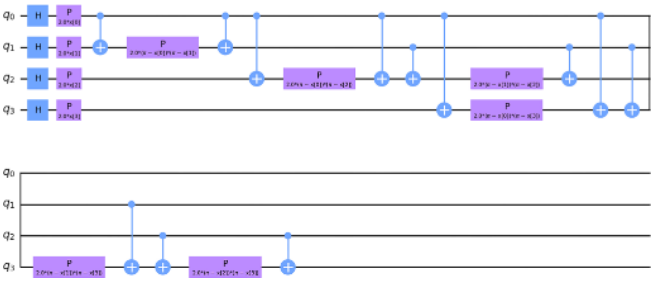

Figura 5: Esquema do circuito quântico que mapeia os dados clássicos em estados quânticos.

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2024-0230

lineares mais complexas entre as características dos dados. Porém, como foi dito anteriormente, o *dataset* é simples, de forma que escolhemos apenas uma repetição.

Na Figura 6 vemos a segunda parte do código, que foi aqui separado desta forma para fins didáticos. Nas linhas 2 e 3, definimos o ansatz usando *RealAmplitudes*, que é um circuito quântico cuja principal finalidade é a de permitir a otimização dos parâmetros para resolver problemas específicos, tais como a classificação (esquema do circuito mostrado na Figura 7). Tal circuito pode ser diretamente comparado às camadas de redes neurais [78–80], tanto em termos de estrutura quanto de funcionalidade. Suas portas quânticas R_Y têm parâmetros (que são os $\theta[i]$ na Figura 7, com $i = 0, 1, \ldots, 11$) que podem ser ajustados durante o treinamento para minimizar uma função de custo específica, capacidade essencial para resolver problemas de otimização e classificação, permitindo que o modelo quântico aprenda a partir de dados de treinamento, da mesma forma que uma rede neural ajusta seus pesos e vieses. Neste caso, escolhemos um circuito com duas repetições (*reps = 2* na linha 2 da Figura 6), o que significa que temos duas camadas com parâmetros a serem ajustados.

```
1  # Define o ansatz RealAmplitudes
2  ansatz = RealAmplitudes(num_qubits=n_qubits, reps=2)
3  ansatz.decompose().draw(output="mpl", style="clifford", fold=20)
4
5  # Define o otimizador COBYLA
6  optimizer = COBYLA(maxiter=50)
7
8  # Inicializa o classificador VQC
9  vqc = VQC(feature_map=feature_map, ansatz=ansatz, optimizer=optimizer)
10
11 # Treinamento do modelo VQC e teste
12 vqc.fit(X_train, y_train)
13 test_score = vqc.score(X_test, y_test)
14
15 # Fazendo previsões no conjunto de teste
16 y_pred = vqc.predict(X_test)
17
18 # Calculando as métricas no conjunto de teste
19 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
20
21 accuracy = accuracy_score(y_test, y_pred)
22 precision = precision_score(y_test, y_pred, average='weighted')
23 recall = recall_score(y_test, y_pred, average='weighted')
24 f1 = f1_score(y_test, y_pred, average='weighted')
25
26 # Imprimindo as métricas
27 print(f"Acurácia: {accuracy:.2f}")
28 print(f"Precisão: {precision:.2f}")
29 print(f"Recall: {recall:.2f}")
30 print(f"F1-Score: {f1:.2f}")
```

Figura 6: Segunda parte do código Python de AMQ que define o algoritmo quântico e calcula a capacidade de classificar corretamente os dados de teste do *dataset* Iris.

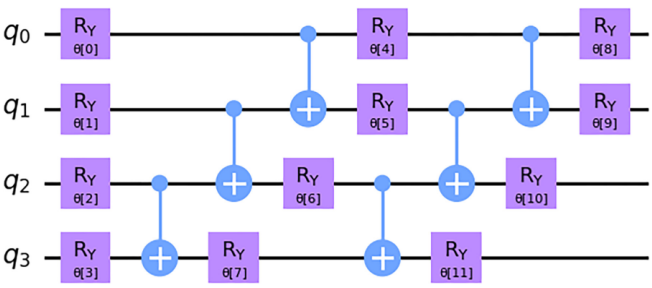

Figura 7: Circuito quântico que realiza a otimização dos parâmetros de forma a minimizar o erro.

Dentre os vários otimizadores disponíveis, tais como Adam (*Adaptive Moment Estimation*) [75], *Gradient Descent* [81–83], AQGD (*Analytic Quantum Gradient Descent*) [84], utilizamos o otimizador COBYLA (*Constrained Optimization BY Linear Approximations* - linha 6 da Figura 6) [85]. Os otimizadores são algoritmos cujos objetivos são o de otimizar uma função em um espaço de parâmetros. No caso do COBYLA, ele opera aproximando a função objetivo por uma série de poliedros lineares e ajustando iterativamente os parâmetros para encontrar a solução ótima. Por não utilizar derivadas, ele é particularmente útil em problemas onde a função objetivo não é suave ou onde as derivadas são difíceis de determinar. Para o *dataset* Iris, escolhemos o COBYLA devido à complexidade potencial da paisagem de otimização quântica e pelo seu código mais simples, no qual só definimos o número máximo de iterações (*maxiter* = 50).

Os algoritmos classificadores disponíveis no Qiskit também são variados. Temos, por exemplo, o QSVC (*Quantum Support Vector Classifier*) [86], o *NeuralNetworkClassifier* [23] e o VQC (*Variational Quantum Classifier*) [87, 88]. Escolhemos este último pela simplicidade maior do código. Na linha 9 da Figura 6 inicializamos o VQC definindo como parâmetros *feature map*, ansatz e otimizador utilizados. A função VQC utiliza algoritmos variacionais para resolver problemas de classificação, combinando técnicas de otimização clássica com computação quântica para ajustar os parâmetros de um circuito quântico de forma a minimizar uma função de custo, similar ao treinamento de redes neurais clássicas. Após a execução do circuito quântico, que é treinado utilizando um simulador de computador quântico, os resultados são medidos e comparados com os rótulos reais dos dados de treinamento, e a função de custo quantifica o erro entre as previsões do modelo e os valores reais.

Nas linhas 12 a 16, treinamos o algoritmo quântico utilizando os exemplos inéditos do *dataset* de treinamento e fazemos as previsões no conjunto de teste. Novamente, na linha 19, importamos as métricas de desempenho já pré-definidas pela biblioteca *scikit-learn* e calculamo-las (linhas 21 a 24) utilizando os dados de teste. Imprimimos os valores de todas as métricas, que para o caso do algoritmo de AMQ são acurácia, *recall* e F1-*Score* de 87% e precisão de 81%.

5. Considerações Finais

Ao observarmos os códigos utilizados para o AM e o AMQ, fica fácil notar a maior simplicidade dos primeiros em relação ao último. O AMQ, por necessitar da etapa de mapeamento dos dados clássicos em quânticos, acrescenta alguma complexidade e demanda o conhecimento básico de fundamentos da computação quântica.

O leitor pode ser levado a acreditar, em um primeiro momento, que todos os conceitos quânticos apresentados e tão promissores sejam uma garantia do sucesso do algoritmo de AMQ frente ao seu análogo clássico. Porém, o que foi aqui demonstrado e que pode ser reproduzido por cada leitor, é que os modelos de AM clássico apresentam melhor acurácia. Os motivos para este aparente paradoxo são, no entanto, bastantes claros. A codificação de dados clássicos em estados quânticos pode introduzir uma complexidade adicional, sendo que escolher o mapeamento correto é crítico e desafiador, e um mapeamento inadequado pode levar à perda de informações importantes. Além disso, os modelos quânticos ainda estão em fases experimentais e podem não escalar bem para alguns conjuntos de dados, em comparação com modelos clássicos que já são bem otimizados para determinados dados, como, por exemplo, o Iris. Também podemos citar aqui que muitos algoritmos de AMQ são conduzidos em simuladores quânticos clássicos, ou seja, computadores clássicos que simulam computadores quânticos e que podem ser lentos e limitados por recursos computacionais clássicos. Isso impacta a capacidade de realizar treinamentos extensivos e avaliações rigorosas de modelos quânticos.

Com todas essas questões levantadas, é importante notar que tanto a computação quântica quanto o AMQ estão em um estágio de desenvolvimento inicial, especialmente quando comparadas aos seus análogos clássicos já bem estabelecidos. No entanto, o campo da computação quântica está avançando rapidamente, e muitas das limitações atuais podem ser superadas à medida que a tecnologia evolui. Por exemplo, a qualidade dos qubits tem melhorado substancialmente. Empresas como IBM e Google têm desenvolvido qubits com tempos de coerência mais longos e taxas de erro mais baixas, como exemplificado pelo processador Sycamore da Google, que alcançou a "supremacia quântica" ao resolver um problema específico mais rapidamente do que o supercomputador clássico mais poderoso [89]. Além disso, o número de qubits nos processadores quânticos está aumentando; IBM, Google e outras empresas estão desenvolvendo chips com dezenas a centenas de qubits [90, 91]. Outro elemento central para os avanços computacionais habilitados por algoritmos quânticos é a exploração do espaço de estados quânticos. A capacidade de acessar um espaço de características quântico exponencialmente grande, que só é eficientemente acessível por um computador quântico, abre caminho para uma possível vantagem. Neste sentido, tem havido implementações teóricas e experimentais de algoritmos que exploram esta questão [92, 93]. Por fim, avanços significativos têm sido feitos em termos de generalização com dados limitados, demonstrando que circuitos quânticos parametrizados obtiveram uma boa generalização com um número limitado de dados de treinamento [20], o que é atualmente um desafio significativo no AM [94, 95], sendo, portanto, um grande passo de superação para o AMQ.

Portanto, enquanto o AM clássico ainda prevalece em termos de métricas de desempenho e praticidade, o futuro do AMQ é promissor, e a pesquisa contínua pode levar a avanços significativos que transformem a paisagem atual do AM.

Torna-se, portanto, crucial compreender os fundamentos da computação quântica e do AMQ. Assim como contribuímos para o avanço do AM, possibilitando o desenvolvimento de inteligências artificiais avançadas, como o ChatGPT, é imperativo impulsionar o AMQ para explorar todo o seu potencial. O AMQ tem a capacidade de resolver problemas que são intratáveis para computadores clássicos, destacando-se em áreas como otimização, simulação de sistemas quânticos e processamento de grandes volumes de dados. Portanto, ao adquirirmos conhecimento e habilidades em AMQ, estamos preparando o terreno para inovações significativas que podem moldar o futuro da computação e da ciência.

Referências

[1] P.H. Winston, *Artificial intelligence* (Addison-Wesley Longman Publishing Co., Massachusetts, 1984).

[2] I. Goodfellow, Y. Bengio e A. Courville, *Deep learning* (MIT Press, Cambridge, 2016).

[3] Z.-H. Zhou, *Machine learning* (Springer Nature, Singapore, 2021).

[4] W. Wang e K. Siau, Journal of Database Management (JDM) **30**, 61 (2019).

[5] J. Jaihar, N. Lingayat, P.S. Vijaybhai, G. Venkatesh e K.P. Upla, em: *International Conference for Emerging Technology (INCET)* (Belgaum, 2020).

[6] S. Omar, A. Ngadi e H.H. Jebur, International Journal of Computer Applications **79**, 2 (2013).

[7] A.B. Nassif, M.A. Talib, Q. Nasir e F.M. Dakalbab, IEEE Access **9**, 78658 (2021).

[8] T. Ahmed, B. Oreshkin e M. Coates, em: *Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques* (Cambridge, 2007).

[9] K.R. Foster, R. Koprowski e J.D. Skufca, Biomedical Engineering Online **13**, 1 (2014).

[10] I. Kononenko, Artificial Intelligence in Medicine **23**, 89 (2001).

[11] J.G. Richens, C.M. Lee e S. Johri, Nature Communications **11**, 3923 (2020).

[12] B.A. Ramalho, L.R. Bortolato, N.D. Gomes, L. Wichert-Ana, F.E. Padovan-Neto, M.A.A. da Silva e K.J.C. Lacerda, Journal of Medical Artificial Intelligence **7**, 1 (2024).

[13] D. Rolnick, P.L. Donti, L.H. Kaack, K. Kochanski, A. Lacoste, K. Sankaran, A.S. Ross, N. Milojevic-Dupont, N. Jaques, A. Waldman-Brown et al., ACM Computing Surveys (CSUR) **55**, 1 (2022).

[14] K. Kashinath, M. Mustafa, A. Albert, J.L. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh et al., Philosophical Transactions of the Royal Society A **379**, 20200093 (2021).

[15] M.R. Bachute e J.M. Subhedar, Machine Learning with Applications **6**, 100164 (2021).
[16] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe e S. Lloyd, Nature **549**, 195 (2017).
[17] M. Schuld, I. Sinayskiy e F. Petruccione, Contemporary Physics **56**, 172 (2015).
[18] M. Cerezo, G. Verdon, H.-Y. Huang, L. Cincio e P.J. Coles, Nature Computational Science **2**, 567 (2022).
[19] V. Chauhan, S. Negi, D. Jain, P. Singh, A.K. Sagar e A.K. Sharma, em: *2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)* (Noida, 2022).
[20] M.C. Caro, H.-Y. Huang, M. Cerezo, K. Sharma, A. Sornborger, L. Cincio e P.J. Coles, Nature Communications **13**, 4919 (2022).
[21] A. Cross, em: *APS March Meeting* (Los Angeles, 2018).
[22] G.F. de Jesus, M.H.F. da Silva, T.G.D. Netto, L.Q. Galvão, F.G. de Oliveira Souza e C. Cruz, Revista Brasileira de Ensino de Física **43**, e20210033 (2021).
[23] *Qiskit contributors, Qiskit: An open-source framework for quantum computing*, disponível em: https://github.com/Qiskit/qiskit (2023).
[24] V. Nasteski, Horizons. b **4**, 51 (2017).
[25] A. Singh, N. Thakur e A. Sharma, em: *3rd International Conference on Computing for Sustainable Global Development (INDIACom)* (New Delhi, 2016).
[26] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L.A. Yau, Y. Elkhatib, A. Hussain e A. Al-Fuqaha, IEEE Access **7**, 65579 (2019).
[27] R.S. Sutton, em: *Reinforcement Learning*, editado por R.S. Sutton (Kluwer, Boston, 1992).
[28] D. Bzdok, M. Krzywinski e N. Altman, Nature Methods **15**, 5 (2018).
[29] S. Weisberg, *Applied Linear Regression* (John Wiley & Sons, New Jersey, 2005).
[30] X. Yan e X. Su, *Linear Regression Analysis: Theory and Computing* (World Scientific, New Jersey, 2009).
[31] L.G. Grimm e P.R. Yarnold, *Reading and Understanding Multivariate Statistics* (American Psychological Association, Washington, 1995).
[32] D.G. Kleinbaum e M. Klein, *Logistic Regression* (Springer-Verlag, New York, 2002).
[33] A. Krogh, Nature Biotechnology **26**, 195 (2008).
[34] B. Yegnanarayana, *Artificial Neural Networks* (PHI Learning Pvt. Ltd., New Delhi, 2009).
[35] P. Harrington, *Machine Learning in Action* (Manning Publications Co., New York, 2012).
[36] C.J.C.H. Watkins, *Models of delayed reinforcement learning*. Tese de Doutorado, Cambridge University, Cambridge (1989).
[37] C.J.C.H. Watkins e P. Dayan, Machine Learning **8**, 279 (1992).
[38] G.A. Rummery e M. Niranjan, *On-Line Q-Learning Using Connectionist Systems* (University of Cambridge, Cambridge, 1994).
[39] I.N. da Silva, D.H. Spatti e R.A. Flauzino, *Redes Neurais Artificiais para Engenharia e Ciências Aplicadas* (Artliber Editora, São Paulo, 2010).
[40] W.S. McCulloch e W. Pitts, The Bulletin of Mathematical Biophysics **5**, 115 (1943).
[41] F. Rosenblatt, Psychological Review **65**, 386 (1958).
[42] D. Maulud e A.M. Abdulazeez, Journal of Applied Science and Technology Trends **1**, 140 (2020).
[43] S.B. Kotsiantis, I. Zaharakis e P. Pintelas, Emerging Artificial Intelligence Applications in Computer Engineering **160**, 3 (2007).
[44] D.V. Carvalho, E.M. Pereira e J.S. Cardoso, Electronics **8**, 832 (2019).
[45] O. Rainio, J. Teuho e R. Klén, Scientific Reports **14**, 6086 (2024).
[46] M. Sokolova e G. Lapalme, Information Processing & Management **45**, 427 (2009).
[47] M. Hossin e M.N. Sulaiman, International Journal of Data Mining & Knowledge Management Process **5**, 1 (2015).
[48] D.M.W. Powers, arXiv:2010.16061 (2020).
[49] G. Casella e R. Berger, *Statistical Inference* (CRC Press, Pacific Grove, 2024).
[50] E. Merzbacher, Quantum Mechanics (John Wiley & Sons, New York, 1998).
[51] R.G.G. Amorim, W.C. Santos, A.P. Ibaldo, A.E. Santana e S. Simon, Revista Brasileira de Ensino de Física **43**, e20210076 (2021).
[52] T. Hey, Computing & Control Engineering Journal **10**, 105 (1999).
[53] W.M.S. Alves e J.C.C. Felipe, Revista Brasileira de Ensino de Física **44**, e20210290 (2022).
[54] J.-L. Brylinski e R. Brylinski, em: *Mathematics of Quantum Computation*, editado por R. K Brylinski and G. Chen (Chapman and Hall/CRC, 2002).
[55] D.P. DiVincenzo, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **454**, 261 (1998).
[56] J.A. Jones, M. Mosca e R.H. Hansen, Nature **393**, 344 (1998).
[57] S.J. Devitt, W.J. Munro e K. Nemoto, Reports on Progress in Physics **76**, 076001 (2013).
[58] D.A. Lidar e T.A. Brun, *Quantum Error Correction* (Cambridge University Press, Cambridge, 2013).
[59] D.A. Freedman, Statistical Models: Theory and Practice (Cambridge University Press, Cambridge, 2009).
[60] L. Rokach e O. Maimon, em: *Data Mining and Knowledge Discovery Handbook*, editado por L. Rokach e O. Maimon (Word Scientific, New Jersey, 2005).
[61] C. Kingsford e S.L. Salzberg, Nature Biotechnology **26**, 1011 (2008).
[62] W.S. Noble, Nature Biotechnology **24**, 1565 (2006).
[63] A.C. Lorena e A.C.P.L.F. de Carvalho, Revista de Informática Teórica e Aplicada **14**, 43 (2007).
[64] R.A. Fisher, Annals of Eugenics **7**, 179 (1936).
[65] Google Colaboratory, disponível em: https://colab.research.google.com/ (2024).
[66] F. Pedregosa, G. Varoquaux. A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., JMLR **12**, 2825 (2011).
[67] P. Priyanka e D. Kumar, International Journal of Information and Decision Sciences **12**, 246 (2020).
[68] B. Charbuty e A. Abdulazeez, Journal of Applied Science and Technology Trends **2**, 20 (2021).

[69] L.A. Jeni, J.F. Cohn e F. De La Torre, em: *Humaine Association Conference on Affective Computing and Intelligent Interaction* (Geneva 2013).

[70] N. Japkowicz, em: *Imbalanced Learning: Foundations, Algorithms, and Applications*, editado por H. He, Y. Ma (Wiley, 2013).

[71] A. Gulli e S. Pal, *Deep Learning with Keras* (Packt Publishing Ltd, Birmingham, 2017).

[72] J. Brownlee, *A Gentle Introduction to the Rectified Linear Unit (ReLU)*, disponível em: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[73] D. Liu, *A Practical Guide to ReLU*, disponível em: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7.

[74] J. Ren e H. Wang, *Mathematical Methods in Data Science* (Elsevier, Amsterdan, 2023).

[75] D.P. Kingma e J. Ba, arXiv preprint arXiv:1412.6980 (2014).

[76] G.E. Crooks, *Gates States and Circuits (2020)*, disponível em: https://chunyangding.com/assets/files/on-gates.pdf

[77] W.-L. Chang e A.V. Vasilakos, *Fundamentals of Quantum Programming in IBM's Quantum Computers* (Springer, Cham, 2021).

[78] T. Khanna, *Foundations of Neural Networks* (Addison-Wesley Longman Publishing Co., Boston, 1990).

[79] J.E. Dayhoff, *Neural Network Architectures: An Introduction* (Van Nostrand Reinhold Co., New York, 1990).

[80] L. Fu, *Neural Networks in Computer Intelligence* (McGraw-Hill, Inc., New York, 1994).

[81] R. Courant, Bulletin of the American Mathematical Society **49**, 1 (1943).

[82] H.B. Curry, Quarterly of Applied Mathematics **2**, 258 (1944).

[83] B.T. Polyak, Introduction to Optimization (Optimization Software, Inc., New York, 1987).

[84] B. Koczor e S.C. Benjamin, Physical Review Research **4**, 023017 (2022).

[85] M.J.D. Powell, em: *Advances in Optimization and Numerical Analysis. Mathematics and Its Applications*, editado por S. Gomez e JP. Hennart (Springer, Netherlands, 1994).

[86] P. Rebentrost, M. Mohseni e S. Lloyd, Physical Review Letters **113**, 130503 (2014).

[87] D. Maheshwari, D. Sierra-Sosa e B. Garcia-Zapirain, IEEE Access **10**, 3705 (2021).

[88] I. Piatrenka e M. Rusek, em: *International Conference on Computational Science* (London, 2022).

[89] F. Arute K. Arya, R. Babbush, D. Bacon, J.C. Bardin, R. Barends, R. Biswas, S. Boixo, F.G.S.L. Brandao, D.A. Buell et al., Nature **574**, 505 (2019).

[90] J. Gambetta, *IBM's roadmap for scaling quantum technology*, disponível em: https://www.ibm.com/quantum/blog/ibm-quantum-roadmap

[91] A. Morvan, B. Villalonga, X. Mi, S. Mandrà, A. Bengtsson, P.V. Klimov, Z. Chen, S. Hong, C. Erickson, I.K. Drozdov et al., arXiv:2304.11119 (2023).

[92] V. Havlicek A.D. Córcoles, K. Temme, A.W. Harrow, A. Kandala, J.M. Chow e J.M. Gambetta, Nature **567**, 209 (2019).

[93] T. Goto, Q.H. Tran e K. Nakajima, Physical Review Letters **127**, 090506 (2021).

[94] V.N. Vapnik e A.Y. Chervonenkis, Th. Prob. App. **16**, 264 (1971).

[95] D. Pollard, *Convergence of Stochastic Processes* (Springer, New York Inc., 1984).