How Quantum clustering gets you dates, the use of entanglement to boost clustering

Abstract

Quantum Machine Learning (QML) is an emerging field with significant potential to enhance existing algorithms. This paper explores a clustering algorithm that leverages quantum principles such as embedding and entanglement to improve performance compared to classical clustering algorithms. Utilizing the Pennylane library, this study delves into the mathematical framework, logical foundations, statistical analysis, and performance metrics of the proposed algorithm. We also discuss the challenges encountered during development. Our aim is to evaluate whether quantum clustering can outperform classical clustering algorithms in specific use cases.

Introduction

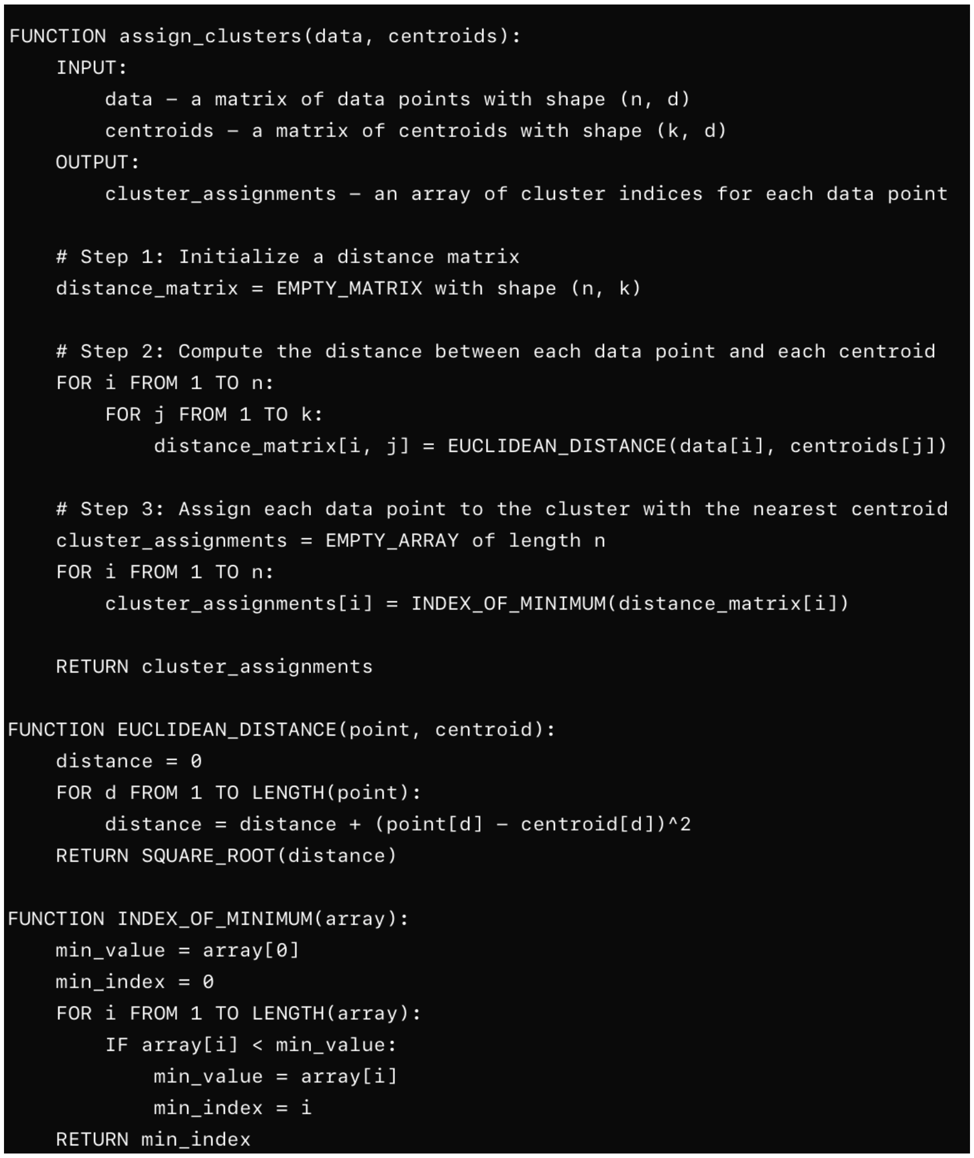
Quantum computing has demonstrated the potential to achieve speedups and enhance the performance of traditional machine learning algorithms. This study focuses on the k-means clustering algorithm, which partitions data into k clusters based on the distance between data points. The process involves initializing centroids, assigning data points to clusters, and recalculating centroid positions. While not all calculations can be performed using quantum principles, our goal is to leverage quantum mechanics as much as possible to optimize clustering.

Classical machine learning is already a robust field, offering the development of proficient automated systems. Quantum machine learning (QML), however, holds the promise of vast improvements. This research aims to determine if quantum computing can enhance machine learning algorithms like k-means clustering. We hypothesize that quantum principles can improve the efficiency and accuracy of clustering, particularly in distance calculations between data points (DiAdamo et al., 2022).

We use tools like Matplotlib and NumPy to simplify code and ensure clarity. Quantum computing principles such as entanglement and fidelity are implemented through pre-existing libraries and custom functions tailored to the algorithm's needs. We evaluate the performance of classical and quantum clustering algorithms using the Silhouette score and the Davies-Bouldin index, while also discussing the Rand index's relevance (Bhardwaj, 2020; GeeksforGeeks, 2023).

Classical Clustering

The clustering algorithm has many steps but it is quite simple. We first load in the dataset, and we then randomly initialize centroids, but the function for this initialization aims to spread the centroids apart as much as possible, the number of centroids is based on the integer variable “k” which can either be hardcoded into the program or it can be an input variable that the user can choose. The usual “k” value used for clustering is three, an attempt to make the “k” value a calculation based on the number of data points in the dataset proved to be inefficient and counterproductive. After plotting all the data and initializing the centroids, we begin to calculate the distance between a data point and all the centroids.



```
FUNCTION assign_clusters(data, centroids):
    INPUT:
        data - a matrix of data points with shape (n, d)
        centroids - a matrix of centroids with shape (k, d)
    OUTPUT:
        cluster_assignments - an array of cluster indices for each data point

    # Step 1: Initialize a distance matrix
    distance_matrix = EMPTY_MATRIX with shape (n, k)

    # Step 2: Compute the distance between each data point and each centroid
    FOR i FROM 1 TO n:
        FOR j FROM 1 TO k:
            distance_matrix[i, j] = EUCLIDEAN_DISTANCE(data[i], centroids[j])

    # Step 3: Assign each data point to the cluster with the nearest centroid
    cluster_assignments = EMPTY_ARRAY of length n
    FOR i FROM 1 TO n:
        cluster_assignments[i] = INDEX_OF_MINIMUM(distance_matrix[i])

    RETURN cluster_assignments

FUNCTION EUCLIDEAN_DISTANCE(point, centroid):
    distance = 0
    FOR d FROM 1 TO LENGTH(point):
        distance = distance + (point[d] - centroid[d])^2
    RETURN SQUARE_ROOT(distance)

FUNCTION INDEX_OF_MINIMUM(array):
    min_value = array[0]
    min_index = 0
    FOR i FROM 1 TO LENGTH(array):
        IF array[i] < min_value:
            min_value = array[i]
            min_index = i
    RETURN min_index
```

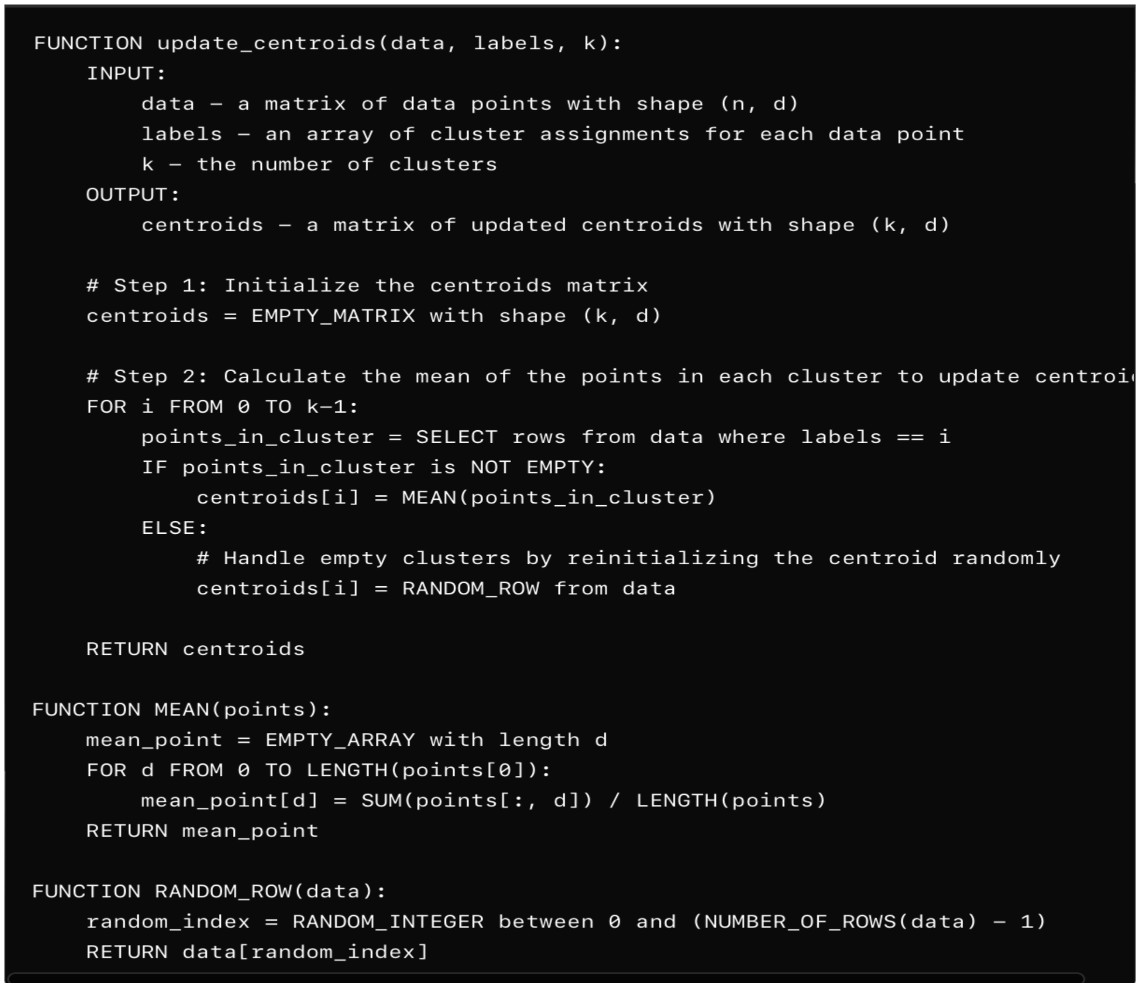
Figure 1

Figure 1 displays the entire function used to assign clusters to data points and how the distances are measured exactly, however, it is not quite clear how the math is done from Figure 1 only

$$\mathbf{D}_{ij} = \sqrt{\sum_{d=1}^{D}(X_{id} - C_{jd})^2}$$

.

X be the dataset with n data points and d dimensions: X={x1,x2,…,xn} C be the set of centroids with k centroids and d dimensions: C={c1,c2,…,ck} The Euclidean distance between each data point xi and each centroid cj is calculated as follows: distanceij= ∥ xi−cj ∥ 2 Where: xi is the i-th data point, cj is the j-th centroid, ∥ a ∥ 2 denotes the Euclidean norm (or L2 norm) of vector a, defined as ∥ a ∥ 2=∑d=1Dad2 In matrix notation, let D be the n×k matrix where each element dij represents the distance between data point ii and centroid j. A data point gets assigned to the nearest centroid, after all the data points are assigned to centroids we update the centroids by calculating the mean position of data points in a cluster.



```
FUNCTION update_centroids(data, labels, k):
    INPUT:
        data - a matrix of data points with shape (n, d)
        labels - an array of cluster assignments for each data point
        k - the number of clusters
    OUTPUT:
        centroids - a matrix of updated centroids with shape (k, d)

    # Step 1: Initialize the centroids matrix
    centroids = EMPTY_MATRIX with shape (k, d)

    # Step 2: Calculate the mean of the points in each cluster to update centroi
    FOR i FROM 0 TO k-1:
        points_in_cluster = SELECT rows from data where labels == i
        IF points_in_cluster is NOT EMPTY:
            centroids[i] = MEAN(points_in_cluster)
        ELSE:
            # Handle empty clusters by reinitializing the centroid randomly
            centroids[i] = RANDOM_ROW from data

    RETURN centroids

FUNCTION MEAN(points):
    mean_point = EMPTY_ARRAY with length d
    FOR d FROM 0 TO LENGTH(points[0]):
        mean_point[d] = SUM(points[:, d]) / LENGTH(points)
    RETURN mean_point

FUNCTION RANDOM_ROW(data):
    random_index = RANDOM_INTEGER between 0 and (NUMBER_OF_ROWS(data) - 1)
    RETURN data[random_index]
```

Figure 2

Figure 2 provides insight into how the centroids are updated after the clusters have been assigned.

$$\mathbf{C} = \left\{ \frac{1}{|\{j|l_j=i\}|} \sum_{j|l_j=i} \mathbf{x}_j \mid i = 0, 1, \ldots, k-1 \right\}$$

Here let X be the dataset with n data points and d dimensions, L={l1,l2,..., ln} be the set of labels, where li∈{0,1,...,k−1}indicates the cluster assignment for data point xi. The centroid ci for cluster i is calculated as the mean of all data points assigned to cluster i, Where ∣{j∣lj=i}∣is the number of data points assigned to cluster i. This is done for 100 iterations to guarantee that the clusters are spaced out and all the distances are accurate even after the centroid position is updated, this is the step that makes this a machine learning algorithm, the iterations done by the algorithm itself to correct and adjust any possible mistakes is extremely crucial.

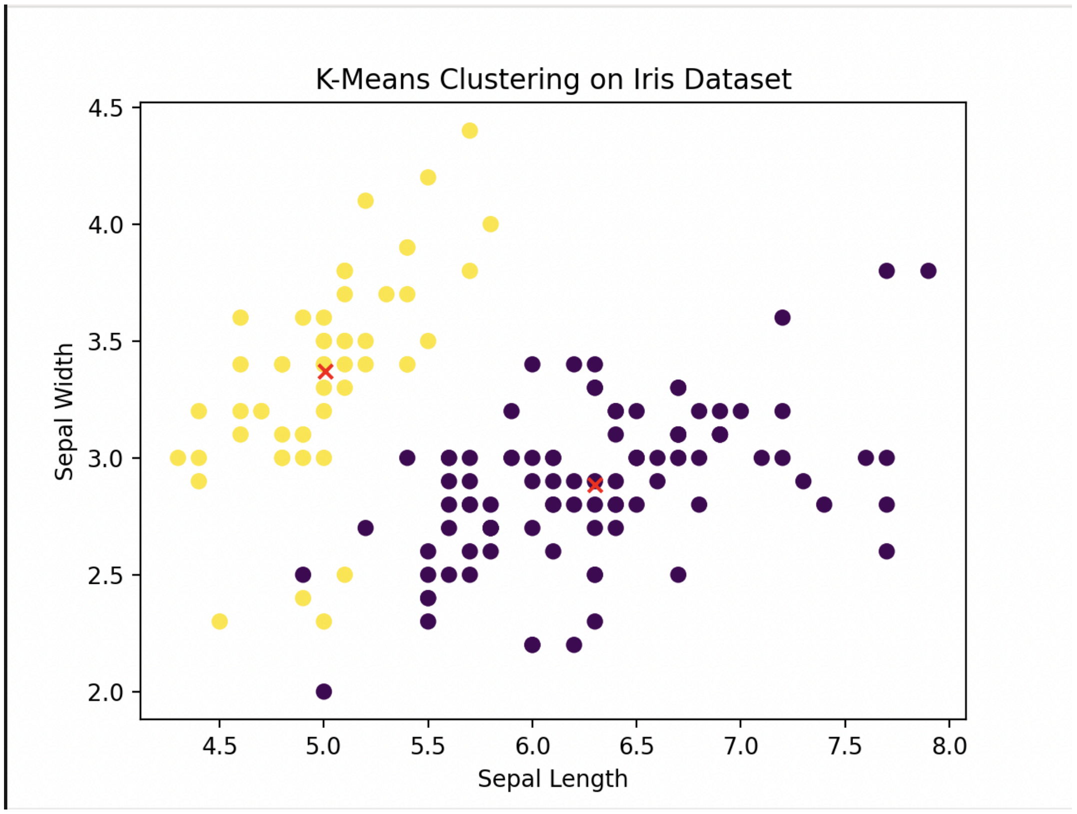


Figure 3

This is the product of using the sk.learn iris dataset that provides measurements for the sepals of plants. The data is clustered into two different clusters based on the "k" value being two. As you can see in Figure 3 the data points are separated and the centroid is in the mean position of all data points in the cluster.

There are numerous possible use cases for a clustering algorithm in various fields such as medicine, finance, sports analytics, and grade comprehension. Overall,
This algorithm is very efficient and is capable of doing clustering extremely well and scores very high in the Silhouette index and Davies-Bouldin index.

Quantum Computing

Before we begin discussing quantum clustering there is a need to lay the groundwork and introduce basic quantum principles that are essential for these algorithms. By applying quantum logic to these algorithms we can improve the efficiency significantly and even create new algorithms that are not possible to create with classical computers. Quantum computing machines are also capable of improving the efficiency of classical computer algorithms without making any alterations(Schuld, 2015).

The most important quantum tools used in our algorithms are Hadmard gates and angle Embedding. A hadmard gate is a very useful logic gate and tool in quantum computing that helps turn a |0> and |1> state into a superposition of |0> and |1> (Leung, 2020).

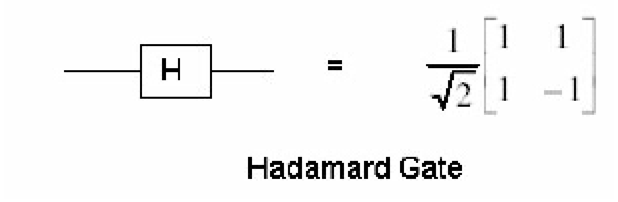

H
$= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Hadamard Gate

Figure 4

Figure 4 shows the matrix of a Hadmard gate. Angle embedding is another important concept for this work, angle embedding encodes classical data into quantum states of a qubit, this is one of the first steps done in quantum clustering algorithms and it is beneficial for making this algorithm work and keeping it simple. In most quantum libraries angle embedding is a pre-made function that is ready to use.

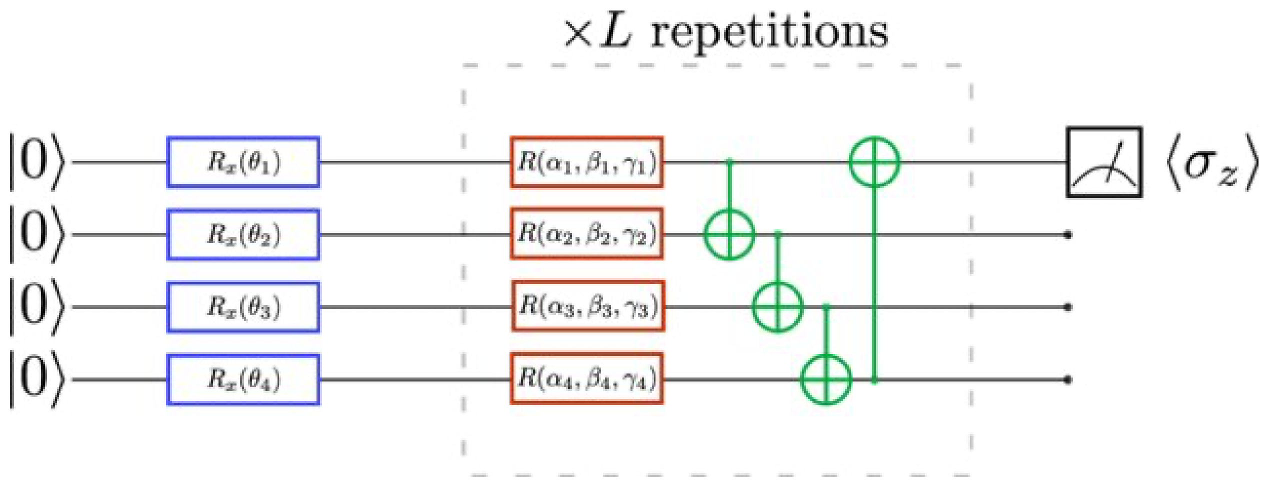

$\times L$ repetitions
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$R_x(\theta_1)$
$R_x(\theta_2)$
$R_x(\theta_3)$
$R_x(\theta_4)$
$R(\alpha_1, \beta_1, \gamma_1)$
$R(\alpha_2, \beta_2, \gamma_2)$
$R(\alpha_3, \beta_3, \gamma_3)$
$R(\alpha_4, \beta_4, \gamma_4)$
$\langle \sigma_z \rangle$

Figure 5

Circuit representation of the Angle Embedding model. In blue, x-axis rotational gates are used to embed the variables into the quantum circuit. In red, trainable generic rotational gates are to be optimized during the training phase. In green, CNOT gates entangling qubits with a circular topology (Voorhoede, n.d.).

A very important concept used is the distance calculation function using Euclidean distance, for this application, the pre-made functions could not be applied so numpy and python.math were used to create a similar function that is adjusted to fit the needs of this case.

$$\text{distance} = \sqrt{\sum_i \left(\Re(x_{1i}) - \Re(x_{2i})\right)^2 + \sum_i \left(\Im(x_{1i}) - \Im(x_{2i})\right)^2}$$

Figure 6

SWAP gates, which swap the state of 2 qubits transforming qubit A into qubit B and vice versa(Liu, 2019). The equation for it can also be written as A' = B, B' = A. A quantum circuit also requires several ancilla qubits to store partial results; the ancilla qubits are initially in a well-defined state, usually | 0⟩ , and must be returned to the same state at the end of the computation.

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Figure 7

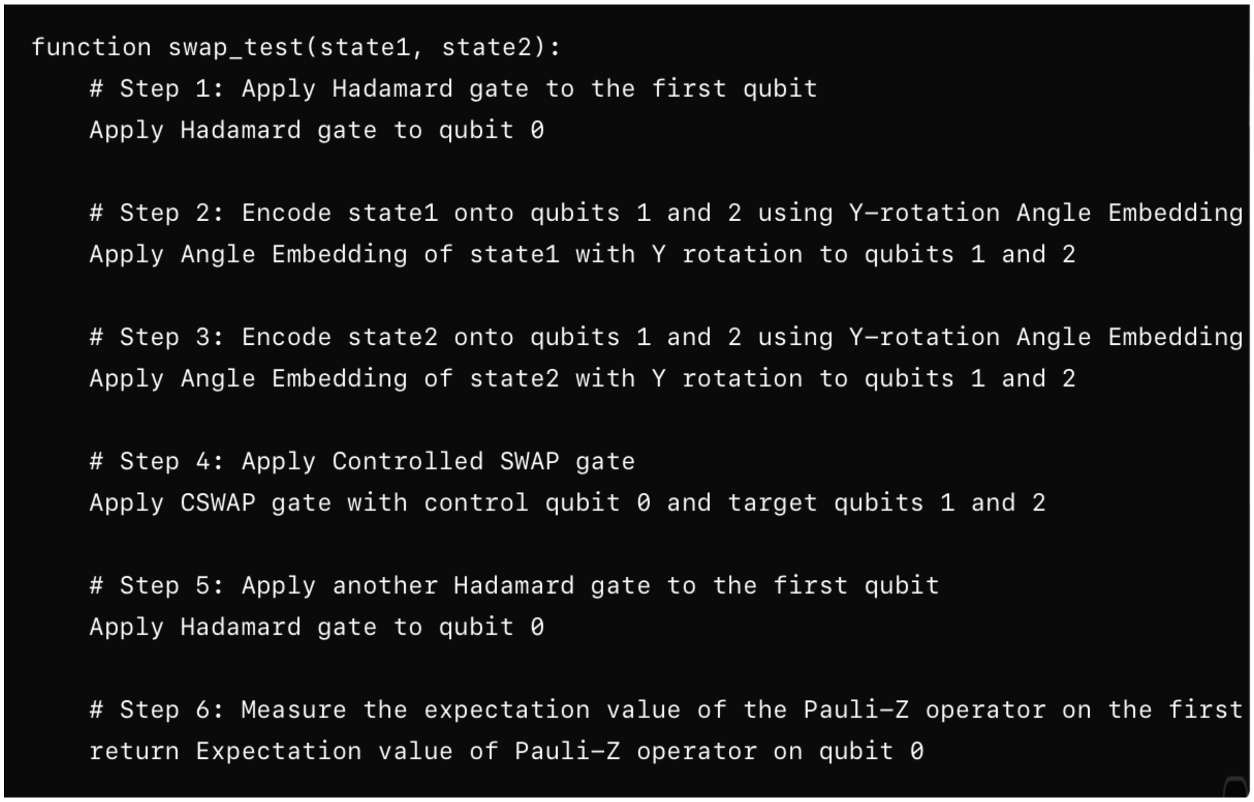
With a SWAP gate and all of our prior knowledge, we can begin to understand how quantum computing maximizes the efficiency of machine learning models. Swap gates can also be used to conduct swap tests. Also, there are controlled swap tests that rely on a third variable's status before conducting the swap.

In niche use cases SWAP gate distance calculation can be highly accurate, and possibly the only option. However, on a broader scale, euclidean distance calculation (EDC) has proven to be more consistent and highly accurate. SWAP gates are vital for quantum computing calculations, similarly Euclidean distance is the benchmark for machine learning distance calculation. When you combine both fields it is evident that Euclidean distance measurements would be more accurate, because of the empirical accuracy of EDC and the difficulty of implementing SWAP gates in higher-dimension data (Brownlee, 2020,).

Quantum Clustering

To enhance the k-means clustering algorithm, we are able to apply quantum principles to produce more accurate and efficient measurements of distance. To begin, quantum clustering can be done on any data as long as angle embedding is used to embed the data as quantum states of a qubit. The K-means clustering algorithm can be enhanced by changing the way the distance is measured, here there are two different possible functions, and it is possible to use the fidelity measurement to calculate the distance between a data point and centroid, or a swap test can be performed. A fidelity function takes the conjugate of the absolute value squared of the data point and centroid and subtracts it from 1 and compares that value with the value for every other centroid. $|<\psi|\varphi>|^2$ Here ψ represents the data point and φ represents the centroid (DiAdamo et al., 2022).

Swap tests have been shown to be more efficient than fidelity calculation. A swap test takes in the quantum state and the centroid position, it then proceeds to apply Hadmard gates and Controlled swap gates on these states to measure the distance in between them. A prominent issue with this implementation was an inaccuracy in assigning the distance values after the calculation has been done because the swap test returns a two-vector array instead of a singular value, which is why the function had to be adjusted later on to be used for clustering.



```
function swap_test(state1, state2):
    # Step 1: Apply Hadamard gate to the first qubit
    Apply Hadamard gate to qubit 0

    # Step 2: Encode state1 onto qubits 1 and 2 using Y-rotation Angle Embedding
    Apply Angle Embedding of state1 with Y rotation to qubits 1 and 2

    # Step 3: Encode state2 onto qubits 1 and 2 using Y-rotation Angle Embedding
    Apply Angle Embedding of state2 with Y rotation to qubits 1 and 2

    # Step 4: Apply Controlled SWAP gate
    Apply CSWAP gate with control qubit 0 and target qubits 1 and 2

    # Step 5: Apply another Hadamard gate to the first qubit
    Apply Hadamard gate to qubit 0

    # Step 6: Measure the expectation value of the Pauli-Z operator on the first
    return Expectation value of Pauli-Z operator on qubit 0
```

Figure 8

In Figure 8 we can see the swap test function, this alone is not enough but it is the base of turning K-means clustering algorithms into quantum algorithms. The Angle embedding is done inside of the swap test here instead of independently in a different function for the sake of efficiency and optimization. After a swap test is conducted the value is then used to assign that data point to a cluster of the K clusters available. Just like in the classical version the centroids are usually randomly initialized in random positions and then they converge into the mean of all data points in the cluster.

In other versions the data points were embedded in a separate function to the distance calculation, then the Euclidean distance of the data points to the centroids was calculated. In this version, the algorithm is not fully quantum and just relies on a qubit and embedding for plotting but it does not rely on quantum principles for calculation. This has proven to be the most efficient version so far relative to index scores, compared to classical algorithms and swap test distance calculation algorithms.

In both versions of the quantum algorithm, the clusters are clear and mathematically correct (Silhouette score is above zero).

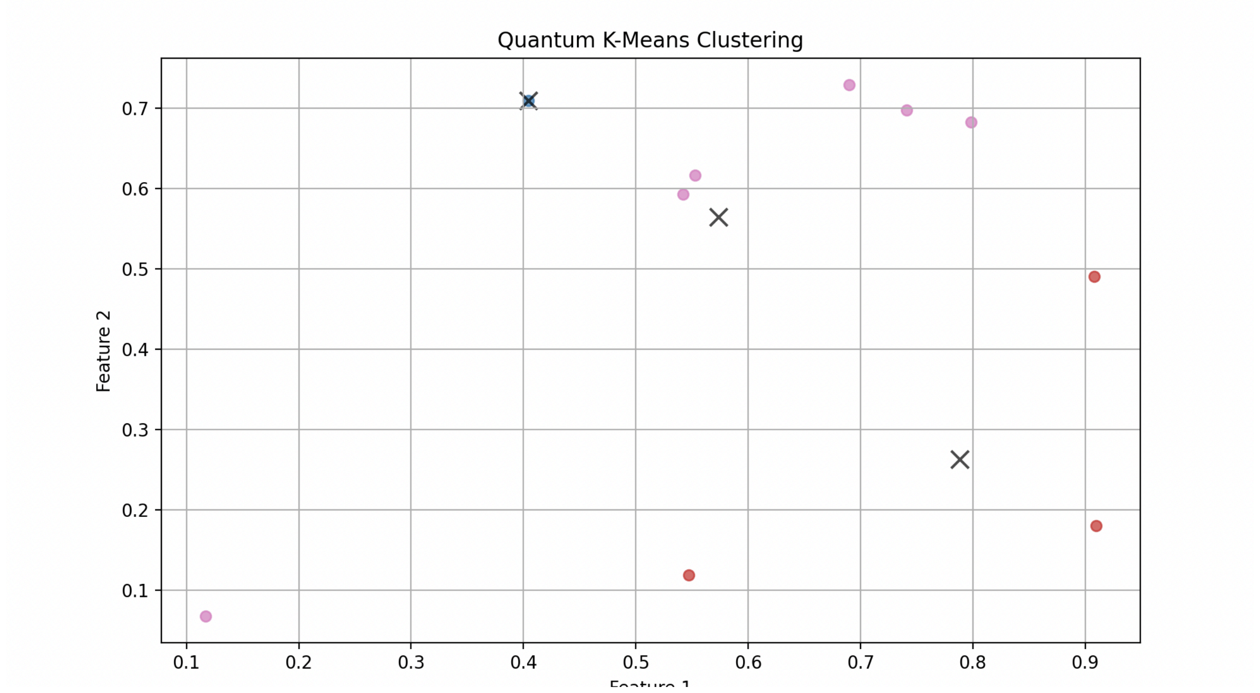


Figure 9

In this figure, the clustering is done using a swap test to calculate the distances, the centroid is in the correct position and all the data points are assigned to the correct clusters.

This algorithm is versatile and can be used in various sectors. A prominent use case is market analysis, a company can use such an algorithm to learn which products are most often bought together and how they would perform if they were bundled up together. Another use case is anomaly detection the example in Figure 8 is a great example of this, the blue datapoint with its own centroid is considered an anomaly in the dataset, and the algorithm can be used to find anomalies in much larger datasets used by companies. A positive and helpful use case is image segmentation of medical images. Here the algorithm can be used to segment images based on pixel colour or region, a great way to identify different tissues in images.

Quantum entanglement clustering

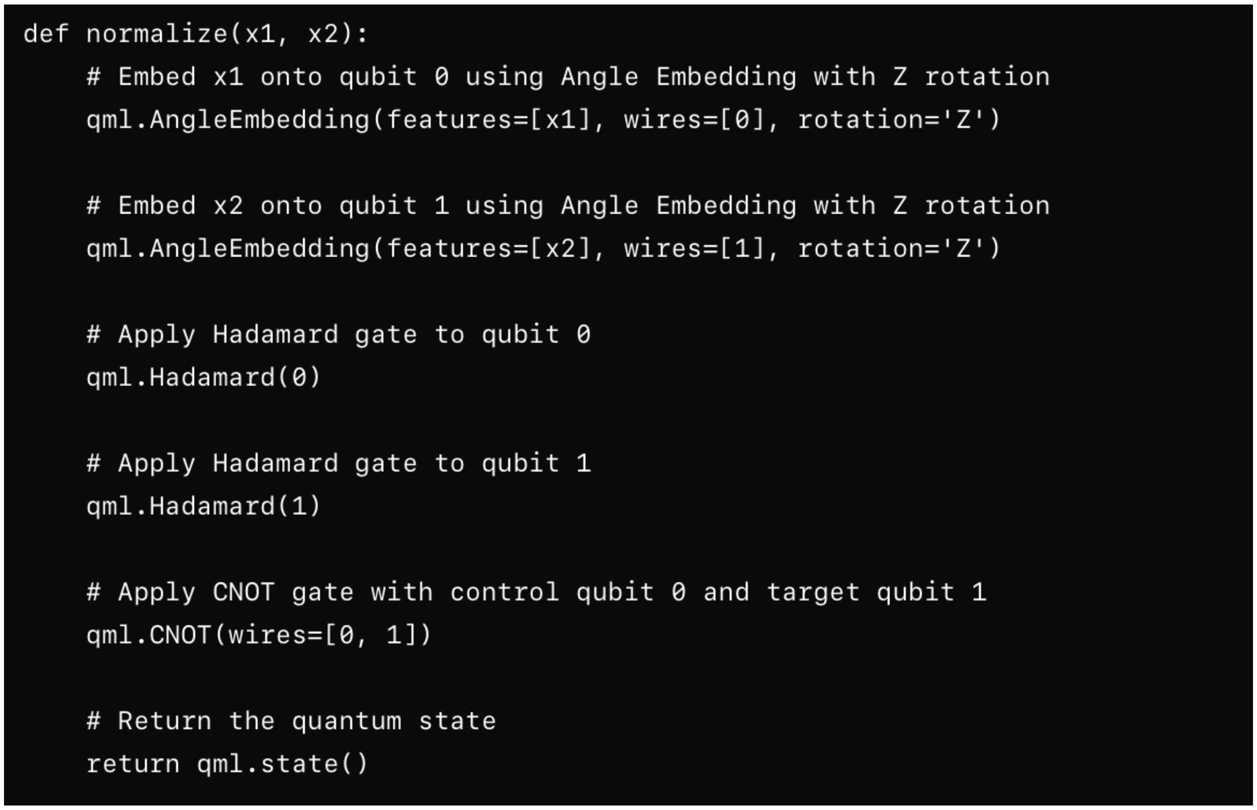
To take it a step further the quantum algorithm can be enhanced to use

entanglement. Here the data is split into two different arrays and each array is embedded into a different qubit, these two qubits are entangled using CNOT (control not) gate (Voorhoede, n.d.). That is the only adjustment made in this algorithm. The outcome however is much more interesting, the algorithm at first outputs fewer data points than it is provided, and after a closer inspection of the plot, the conclusion made is that two data points are now occupying the same position. The entanglement causes data points to converge onto each other and expose their correlation in a quantum state.

$$\text{CNOT} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 10

The CNOT gate used in this algorithm is showcased in Figure 10. It takes two different qubits and entangles them by altering the target qubit based on the state of the control qubit.



```
def normalize(x1, x2):
    # Embed x1 onto qubit 0 using Angle Embedding with Z rotation
    qml.AngleEmbedding(features=[x1], wires=[0], rotation='Z')

    # Embed x2 onto qubit 1 using Angle Embedding with Z rotation
    qml.AngleEmbedding(features=[x2], wires=[1], rotation='Z')

    # Apply Hadamard gate to qubit 0
    qml.Hadamard(0)

    # Apply Hadamard gate to qubit 1
    qml.Hadamard(1)

    # Apply CNOT gate with control qubit 0 and target qubit 1
    qml.CNOT(wires=[0, 1])

    # Return the quantum state
    return qml.state()
```

Figure 11

The pseudocode used here applies the entanglement as soon as the values from the dataset have been embedded into quantum status, to ensure that the clustering is only being done after the two qubits have been entangled. Entanglement can create new use cases for the algorithm and enhance our current use cases. Financial market modelling can benefit from the predictive nature of this algorithm, it can help predict market trends and patterns. A unique use case is the clustering of pixels in image and video resizing or processing. Entangled clustering offers significant advantages in handling complex, high-dimensional data across various domains. Its ability to capture intricate correlations through quantum entanglement can lead to more accurate and insightful clustering, enhancing the performance and reliability of various applications from healthcare to cybersecurity.

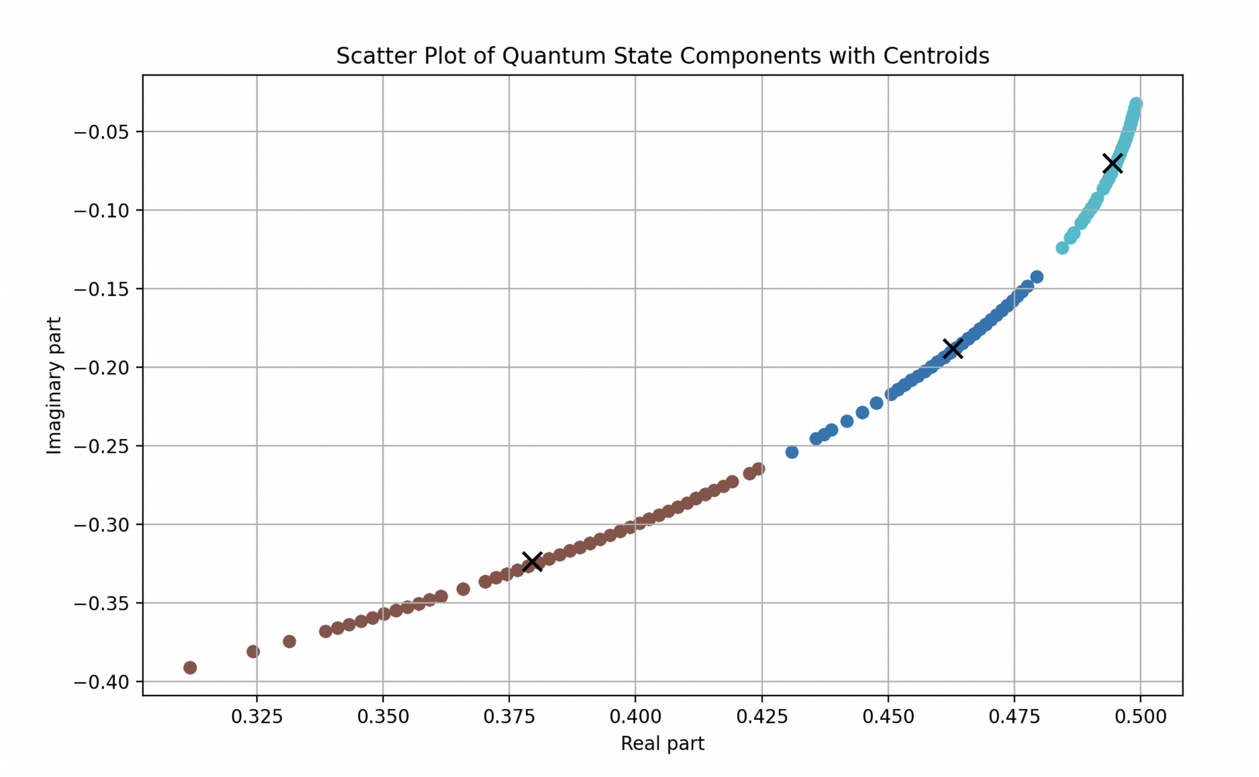


Figure 12

In this figure, the displayed graph is that of an entangled clustering algorithm on the iris dataset provided by sklearn. The dataset inputs 25 data points, but only 21 appear on the graph. This is a great example of how the entanglement works and finds the correlation between data points once they are in a quantum state.

Evaluations

Ultimately, as interesting and remarkable as these algorithms are, the real value is in the performance. To evaluate the performance of these different interpretations of the clustering algorithm, we use the Silhouette score and the Davies-Bouldin Index.

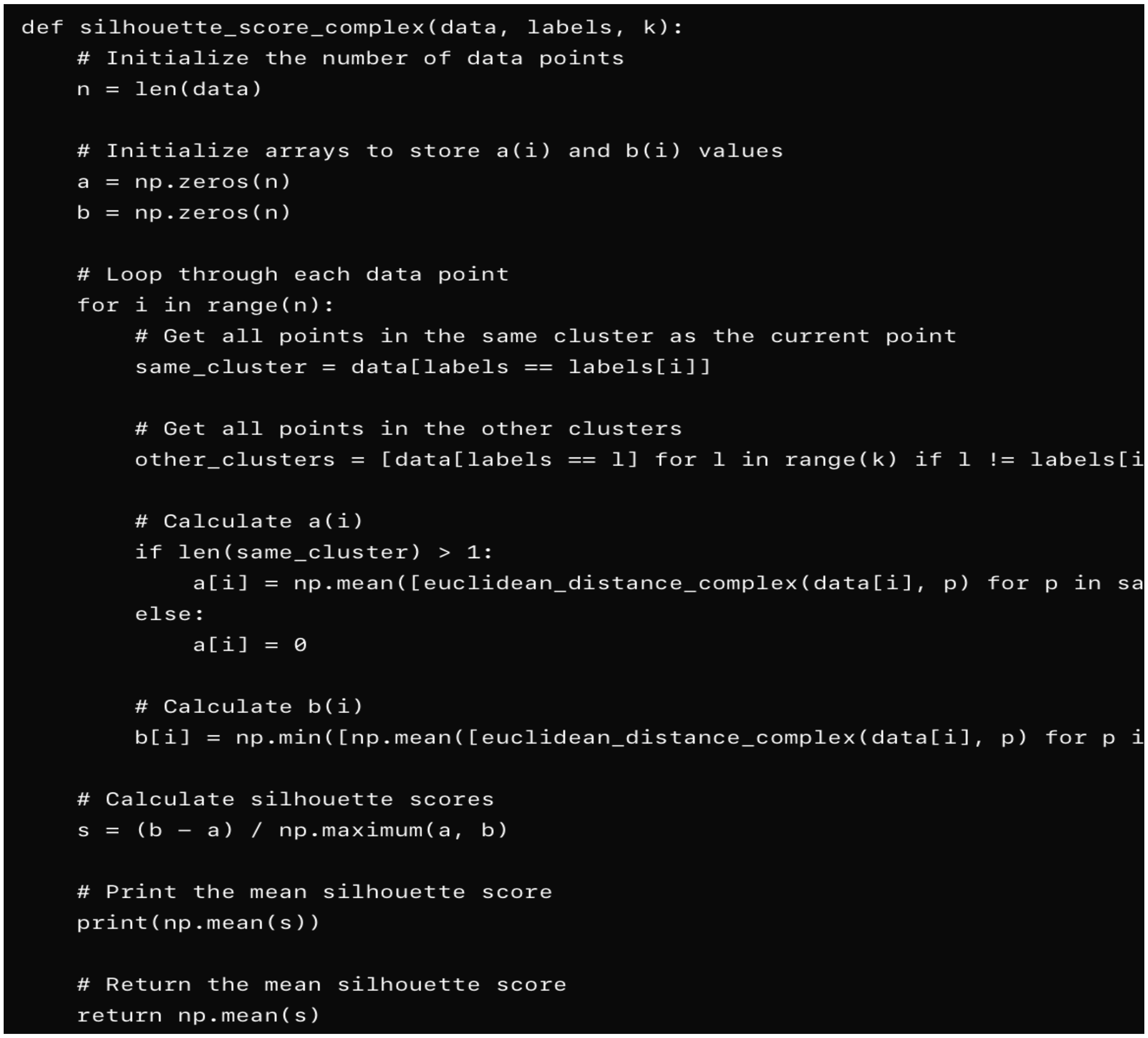
The Silhouette score is a way to evaluate how distinguished and distanced the clusters are. It measures how similar a data point is to its own cluster compared to other clusters. The score ranges from -1 to 1, with values below 0 indicating poor clustering, values around 0 suggesting overlapping clusters, and values above 0 indicating acceptable to excellent clustering. A higher Silhouette score represents better-defined clusters (Bhardwaj, 2020).

On the other hand, the Davies-Bouldin Index measures the similarity ratio between inter-cluster and intra-cluster distances. It is calculated as the average similarity ratio between each cluster and its most similar cluster. Since it is a ratio, a Davies-Bouldin

Index ideally under 1 indicates good clustering performance, while values above 1 suggest poor clustering (GeeksforGeeks, 2023).

However, we encountered an issue with using the sklearn functions that are readily available. Since our data is embedded in quantum states, the calculations for both evaluation methods needed to be adjusted to accommodate the quantum nature of the data. This involved modifying the algorithms to ensure accurate performance evaluation for the quantum clustering algorithms.

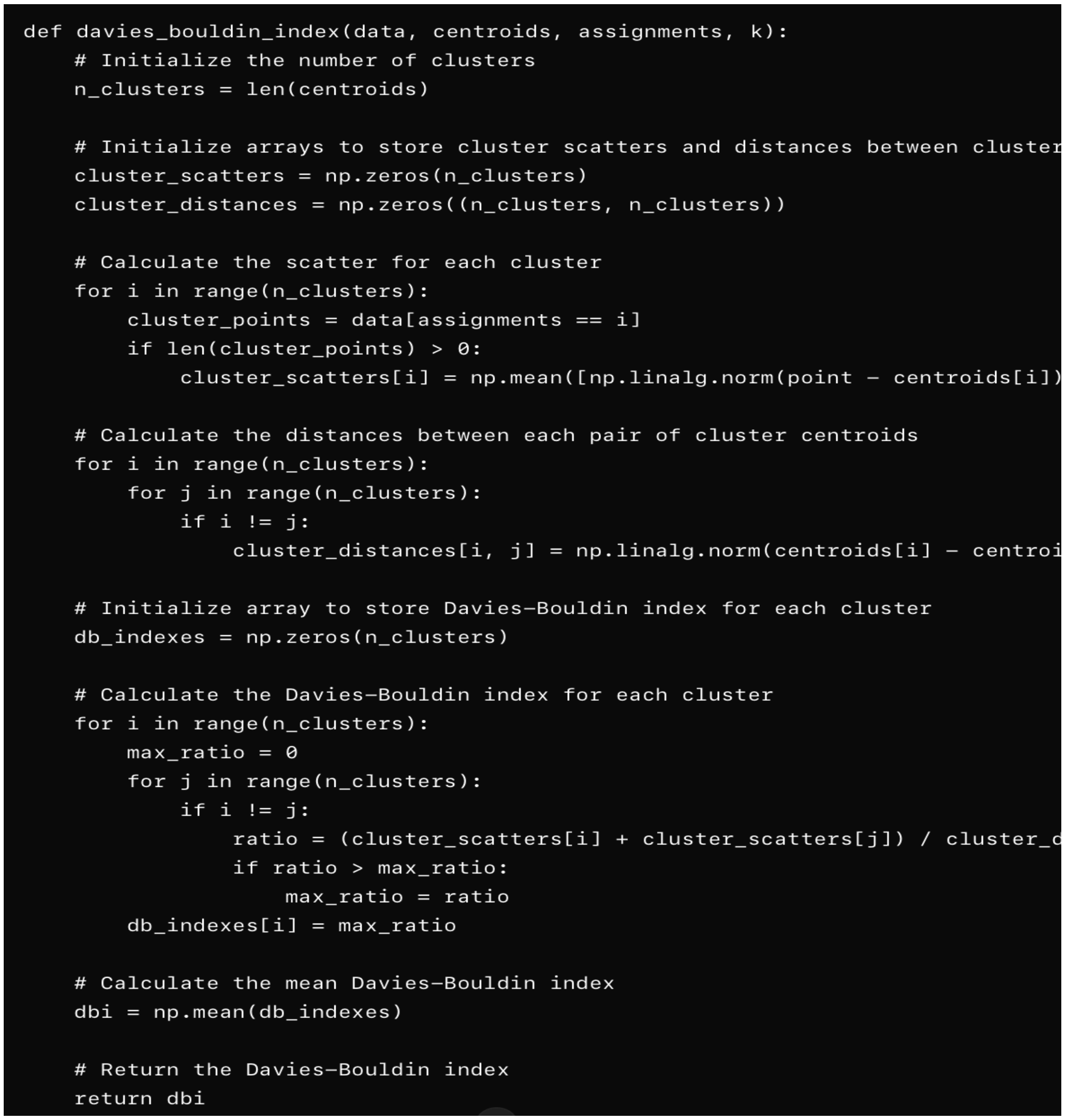
The Silhouette score and Davies-Bouldin index offer insight into clustering performance. The Silhouette score assesses the separation and cohesion of clusters. It ranges from -1 to 1, with higher values indicating better-defined clusters. The Davies-Bouldin index, on the other hand, measures the average similarity ratio between each cluster and its most similar cluster. A lower Davies-Bouldin index indicates better clustering quality. These metrics help determine the effectiveness of clustering algorithms by quantifying their ability to create distinct, well-separated clusters (Bhardwaj, 2020; GeeksforGeeks, 2023).



```
def silhouette_score_complex(data, labels, k):
    # Initialize the number of data points
    n = len(data)

    # Initialize arrays to store a(i) and b(i) values
    a = np.zeros(n)
    b = np.zeros(n)

    # Loop through each data point
    for i in range(n):
        # Get all points in the same cluster as the current point
        same_cluster = data[labels == labels[i]]

        # Get all points in the other clusters
        other_clusters = [data[labels == l] for l in range(k) if l != labels[i

        # Calculate a(i)
        if len(same_cluster) > 1:
            a[i] = np.mean([euclidean_distance_complex(data[i], p) for p in sa
        else:
            a[i] = 0

        # Calculate b(i)
        b[i] = np.min([np.mean([euclidean_distance_complex(data[i], p) for p i

    # Calculate silhouette scores
    s = (b - a) / np.maximum(a, b)

    # Print the mean silhouette score
    print(np.mean(s))

    # Return the mean silhouette score
    return np.mean(s)
```

Figure 13

This is the exact same Silhouette score as sklearn, the only adjustment is made when the arrays are initialized and how they get passed to the function.



```
def davies_bouldin_index(data, centroids, assignments, k):
    # Initialize the number of clusters
    n_clusters = len(centroids)

    # Initialize arrays to store cluster scatters and distances between cluster
    cluster_scatters = np.zeros(n_clusters)
    cluster_distances = np.zeros((n_clusters, n_clusters))

    # Calculate the scatter for each cluster
    for i in range(n_clusters):
        cluster_points = data[assignments == i]
        if len(cluster_points) > 0:
            cluster_scatters[i] = np.mean([np.linalg.norm(point - centroids[i])

    # Calculate the distances between each pair of cluster centroids
    for i in range(n_clusters):
        for j in range(n_clusters):
            if i != j:
                cluster_distances[i, j] = np.linalg.norm(centroids[i] - centroi

    # Initialize array to store Davies-Bouldin index for each cluster
    db_indexes = np.zeros(n_clusters)

    # Calculate the Davies-Bouldin index for each cluster
    for i in range(n_clusters):
        max_ratio = 0
        for j in range(n_clusters):
            if i != j:
                ratio = (cluster_scatters[i] + cluster_scatters[j]) / cluster_d
                if ratio > max_ratio:
                    max_ratio = ratio
        db_indexes[i] = max_ratio

    # Calculate the mean Davies-Bouldin index
    dbi = np.mean(db_indexes)

    # Return the Davies-Bouldin index
    return dbi
```

Figure 14

Internal evaluation methods like the Rand index, which compares the output of the clustering algorithm to the ground truth of the data set, the closer the value of the Rand index to one the better the algorithm's performance relative to the ground truth, are not applicable to these algorithms. While some quantum clustering algorithms might score well in this evaluation, it is irrelevant to this case and does not imply anything about the performance of the algorithm. This is because the ground truth is based on classical clustering of classic data, but in this case, the data is being embedded in quantum states first changing its true position on the graph.

	Silhouette Score	Davies-Bouldin index
Swap test quantum clustering	0.142402387171144	0.8239237501354221
Euclidean quantum clustering	0.5851296277182536	0.4834896309715491
Entangled quantum clustering	0.7373073521517061	0.31442990242642566
Classical clustering	0.6810461692117462	0.40429283717304246

Figure 15

Figure 15 provides a lot of significant information about the performance of all the algorithms and interpretations discussed. Clearly, the entangled quantum clustering algorithm is the best performing in the Silhouette score evaluation, however, all three quantum functions perform well, while the swap test clustering is the worst performing algorithm in the Figure, it is still above zero which is a sign of a good performing algorithm. When it comes to the Davies-Bouldin index, however, the Euclidean quantum and entangled quantum algorithms both outperform the classical clustering algorithm. This Figure provides data based on testing and evaluating the algorithms discussed and proves that quantum versions of clustering can outperform classical clustering in some areas.

Conclusion

To finish off, while the classical clustering algorithm is a great tool of machine learning that has many use cases, it is still able to be enhanced with the use of quantum computing principles. The use of angle embedding and quantum gates to compute the distance measurements has proven to be a useful and legitimate way of applying quantum principles to clustering, as well as the use of angle embedding to embed data points into quantum states of a qubit. While these methods are effective in implementing and improving the performance of clustering algorithms entanglement of qubits prior to clustering has proven to be the most effective and efficient way of applying quantum computing principles to improve the performance of clustering algorithms.

References

ChatGPT. (n.d.). Retrieved July 29, 2024, from https://chatgpt.com/c/0903de6c-739a-42e0-9bee-b85fd861205c

Fidelity. (n.d.). Quatomic. Retrieved July 29, 2024, from https://www.quatomic.com/composer/reference/state-analysis/fidelity/

DiAdamo, S., O'Meara, C., Cortiana, G., & Bernabé-Moreno, J. (2022). Practical quantum K-means clustering: Performance analysis and applications in energy grid classification. *arXiv*. https://arxiv.org/pdf/2112.08506

Voorhoede, D. (n.d.). *CNOT gate*. Quantum Inspire. Retrieved July 29, 2024, from https://www.quantum-inspire.com/kbase/cnot/#

Leung, M. (2020, April 25). What is Quantum Computing? (Part 4) - Matthew Leung. *Medium*. https://iwasnothing.medium.com/what-is-quantum-computing-part-4-313044d516ac

Bhardwaj, A. (2020, May 27). Silhouette coefficient. *Towards Data Science*. https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c

GeeksforGeeks. (2023, November 5). Davies-Bouldin index. *GeeksforGeeks*. https://www.geeksforgeeks.org/davies-bouldin-index/

Li, Z., Li, J., & Xu, Y. (2022). Hyperbolic geometry in deep learning: A survey. *arXiv*. https://arxiv.org/pdf/2212.06691

Mr. VRC. (n.d.). Evaluating clustering algorithms: A comprehensive guide. *LinkedIn*. Retrieved

July 2024, from https://www.linkedin.com/pulse/evaluating-clustering-algorithms-comprehensive-guide-mba-ms-phd-mrvrc/

Han, Y., & Kim, J. (2007). Quantum robots for teenagers. *ResearchGate*. Retrieved July 2024, from https://www.researchgate.net/publication/4248027_Quantum_Robots_for_Teenagers/figures

Ammar, O. (2024, April*) Quantum Machine Learning*

Ghose, S.(2024) Quantum Computing [CP351], Wilfrid Laurier University, Waterloo Ontario

Brownlee, J. (2020, March 24). 4 distance measures for machine learning. MachineLearningMastery.Com. https://machinelearningmastery.com/distance-measures-for-machine-learning/