Exploring the Q-Means Algorithm - Quantum Algorithm for Unsupervised Learning

Project Report
presented to Dr. Abdullah Khalid
(abdullah.khalid@sse.habib.edu.pk)
by

Faraz Ahmed Khan	fk03983
Huda Feroz Ahmed	ha04081
Syed Ammar Ahmed	sa04050

In partial fulfillment of the course requirements for
CS/PHY 314/300 Quantum Computing

Dhanani School of Science and Engineering

Habib University
Spring 2021

Contents

List of Figures

1. Introduction

Clustering is a very important task in unsupervised Machine Learning. The aim of clustering is to group similar objects together based on a distance metric. It is used in a variety of problems and domains such as image processing, finance, market research etc. For example, clustering can be used by businesses to group their customers based on their likes/dislikes, purchase power, frequency of purchase etc. One of the most popular algorithms used for clustering is the K-means algorithm. Using it as the base, we would be introducing a Q-means algorithm and see the difference in their cluster assignment accuracy and error rate in assignment of the points to clusters.

1.1 Project Aims

The aim of this project is show the working of Q-means in comparison to its classical counterpart, K-means. The project focuses on implementing both the algorithms and show their accuracy in terms of cluster formation and centroid assignment in comparison to each other. The project also points out the issue with distance estimation in Q-means as in K-means it is fairly simple because distance between data points and centroid is measured in form of euclidean distance however in Q-means

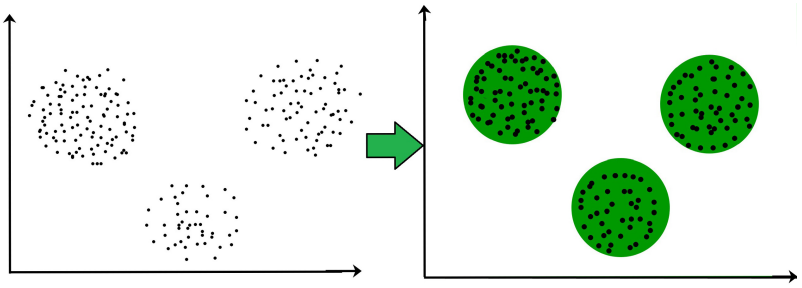


Figure 1.1: Clustering of data points into three clusters

the same method cannot be deployed efficiently. Hence the project introduces another method for distance estimation and the known Bloch Sphere representation of qubits for the said task. The project then moves forward with the comparison of both the algorithms with two types of data distribution, uniform and normal and shows the error rate in accordance to the distribution.

2. Background

2.1 Bloch Sphere Representation

This section will explain a method for representation and visualization of a qubit. This section derives its explanation mainly from [1]. We know that a qubit can be written as,

$$|\rho\rangle = c_0|0\rangle + c_1|1\rangle \tag{2.1}$$

Where, c_0 and c_1 are complex numbers. We also know that any complex number can be written as,

$$c_i = r_i e^{j\theta} \tag{2.2}$$

Putting (2.2) in (2.1) and using the fact that $c_0^2 + c_1^2 = 1$, we could rewrite the equation 2.1 as,

$$|\rho\rangle = r_0|0\rangle + r_1 e^{j\theta}|1\rangle \tag{2.3}$$

Using euler identity, we could rewrite equation 2.3 as,

$$|\rho\rangle = \cos\theta_1|0\rangle + \sin\theta_1 e^{j\theta_2}|1\rangle \tag{2.4}$$

or more conventionally,

$$|\rho\rangle = \cos\theta|0\rangle + e^{j\phi}\sin\theta|1\rangle \tag{2.5}$$

Equation (2.5) allows us visualize the state of qubit, using the spherical co-ordinates system. We know that the magnitude of any qubit is always equals to one, so the radius parameter in the spherical co-ordinates system would be equal to 1, while θ and ϕ come directly from equation (2.5). This means that all possible states of a qubit could be visualized on the surface of a unit sphere. This sphere is called the bloch sphere and this method of representation is called bloch sphere representation, named on Felix Bloch.

Another important takeaway from equation (2.5) is the fact that the probability of ρ measuring to $|0\rangle$ is equal to $\cos^2\theta$ and probability of ρ measuring to $|1\rangle$ is equal

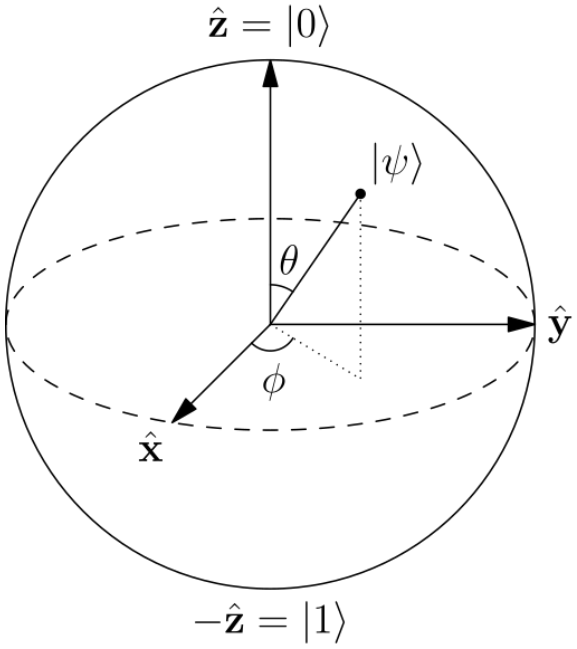


Figure 2.1: Diagram of Bloch Sphere

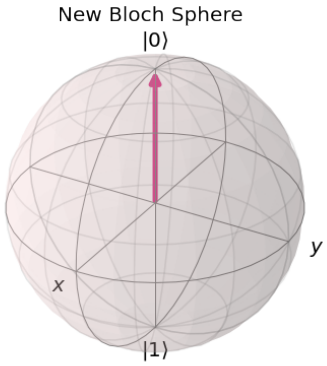
to $\sin^2\theta$ as $|e^{j\phi}| = 1$. This suggests that that as θ moves from 0 to $\frac{\pi}{2}$ the probability of $|0\rangle$ decreases from 1 to 0, which can be modelled as the rotation of a vector. In order to model a rotation from 0 to π to represent probability going from 1 to 0, equation (2.5) can be written as,

$$|\rho\rangle = \cos\frac{\theta}{2}|0\rangle + e^{j\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.6}$$

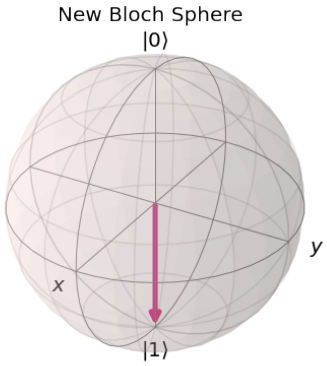
Putting all of this together, figure 2.1 shows the Bloch sphere. The positive z-axis denotes the basis $|0\rangle$ and negative z-axis denotes the basis $|1\rangle$. We can see that any vector above the x-y plane will more likely measure $|0\rangle$ than $|1\rangle$ as θ in that case will be less than $\frac{\pi}{2}$ and hence $\cos^2\frac{\theta}{2}$ will be greater than $\sin^2\frac{\theta}{2}$.

Figure 2.2(a) and 2.2(b) shows the representation of $|0\rangle$ and $|1\rangle$ respectively. We can see that the vectors point in opposite direction. $|0\rangle$ makes an of 0 with positive z-axis while $|1\rangle$ makes an angle of π with positive z-axis.

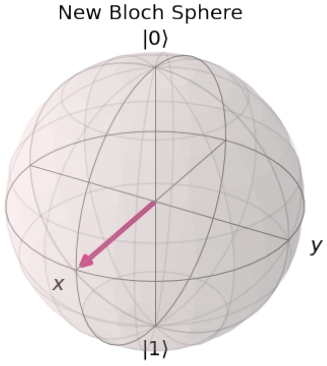
Figure 2.2(c) and 2.2(d) shows the representation of $|+\rangle$ and $|-\rangle$. This is interesting because both $|+\rangle$ and $|-\rangle$ make an angle $\theta = \frac{\pi}{2}$ with positive z-axis, but the



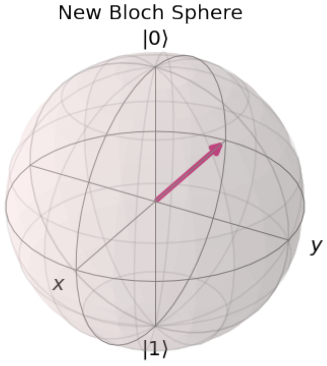
(a) $|0\rangle$ Representation in Bloch sphere



(b) $|1\rangle$ Representation in Bloch sphere



(c) $|+\rangle$ Representation in Bloch sphere



(d) $|-\rangle$ Representation in Bloch sphere

angle ϕ (angle with x-axis) is different. This shows the impact of $e^{j\phi}$ in equation (2.6)($\phi = 0$ for 2.2(c) and $\phi = \frac{pi}{2}$ for 2.2(d)). It does not impact the probabilities but it has a significant impact on the orientation.

2.2 U3 Gate

In order to convert a $|0\rangle$ into our desired vector, the U3 gate is used, which in itself is a combination of different gates. The matrix representation[2] for the gate is given as:

$$\begin{bmatrix} \cos\frac{\theta}{2} & -e^{\iota\lambda}\sin\frac{\theta}{2} \\ -e^{\iota\phi}\sin\frac{\theta}{2} & -e^{\iota(\phi+\lambda)}\cos\frac{\theta}{2} \end{bmatrix}$$

The values of θ and ϕ for our case moves the $|0\rangle$ to the desired position.

2.3 Relation between Euclidean Distance and Inner Product

The relation between euclidean distance between two vectors and their inner product is fundamental for the implementation of kmeans on a quantum circuit. The kmeans algorithm works on the basis of euclidean distances while the quantum circuit we propose estimates for the inner product. Here we claim that the euclidean distance between two unit vectors is a monotonous function of their inner product. Since the vectors that we are dealing with lie on the block sphere, we can be sure that they are unit vectors. The proof for this claim goes as follows,

We know that the euclidean distance between two vectors a and b is given by,

$$Dist = ||\vec{a} - \vec{b}||^2 \tag{2.7}$$

Using law of cosines,

$$Dist = \sqrt{||\vec{a}||^2 + ||\vec{b}||^2 - 2|\vec{a}||\vec{b}|\cos\theta} \tag{2.8}$$

Since $\vec{a}$ and $\vec{b}$ are unit vectors, $||\vec{a}||^2 = 1$ and $||\vec{b}||^2 = 1$. Hence,

$$Dist = 2 - 2|\vec{a}||\vec{b}|\cos\theta \tag{2.9}$$

We also know that,

$$|\vec{a}||\vec{b}|\cos\theta = \vec{a}\cdot\vec{b} \tag{2.10}$$

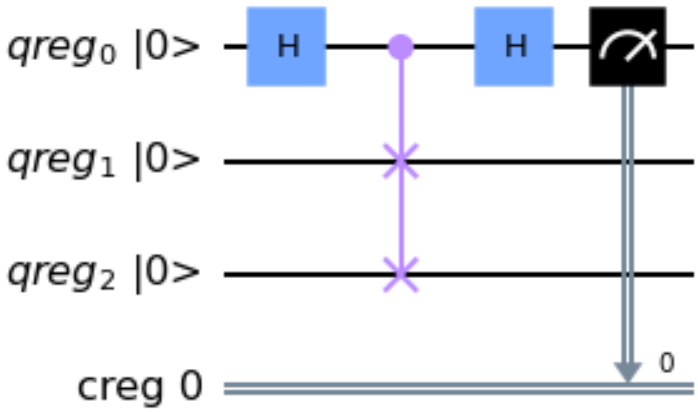


Figure 2.2: SWAP Test Circuit

Hence,

$$Dist = 2 - 2\vec{a} \cdot \vec{b} \tag{2.11}$$

Which shows Dist to be a monotonic function of the inner product between $\vec{a}$ and $\vec{b}$. Therefore an estimation for the inner product could be used as an alternate for euclidean distance.

2.4 SWAP Test For Inner Product

The swap test circuit given in Figure2.2 can be used to estimate inner product between two qubits (or vectors in Bloch Sphere). $P(|1\rangle)$ or $P(|0\rangle)$ can be used to estimate the inner product. This derivation from [5] shows why,

$$\psi_0 = |0, a, b\rangle \tag{2.12}$$

Application of Hadamard,

$$\psi_1 = \frac{1}{\sqrt{2}}(|0, a, b\rangle + |1, a, b\rangle) \tag{2.13}$$

Application of cSWAP gate,

$$\psi_2 = \frac{1}{\sqrt{2}}(|0,a,b\rangle + |1,b,a\rangle) \tag{2.14}$$

Application of Second Hadamard

$$\psi_3 = \frac{1}{\sqrt{2}}((\frac{1}{\sqrt{2}}(|0,a,b\rangle + |1,a,b\rangle) + (\frac{1}{\sqrt{2}}|0,a,b\rangle - |1,b,a\rangle)) \tag{2.15}$$

$$\psi_4 = \frac{1}{2}(|0\rangle(|a,b\rangle + |b,a\rangle) + |1\rangle(|a,b\rangle + |b,a\rangle)) \tag{2.16}$$

Probability of 1 can be measured as,

$$P(|1\rangle) = \frac{1}{4}(|(|a,b\rangle - |b,a\rangle)|^2) \tag{2.17}$$

$$P(|1\rangle) = \frac{1}{4}((\langle b|b\rangle\langle a|a\rangle - \langle b|a\rangle\langle a|b\rangle - \langle a|b\rangle\langle b|a\rangle + \langle a|a\rangle\langle b|b\rangle)) \tag{2.18}$$

here $\langle a|a\rangle = 1$ and $\langle b|b\rangle = 1$ (as unit vectors) and $\langle b|a\rangle = \langle a|b\rangle$ as inner product is commutative. Hence eq.2.18 becomes,

$$P(|1\rangle) = \frac{1}{4}((1 - \langle a|b\rangle^2 - \langle a|b\rangle^2 + 1)) \tag{2.19}$$

$$P(|1\rangle) = \frac{1}{2} - \frac{\langle a|b\rangle^2}{2} \tag{2.20}$$

Thus, $P(|1\rangle) = 0.5$ implies orthogonality (max distance in Bloch sphere) and $P(|1\rangle) = 0$ implies maximum similarity between the vectors.

3. Description

There are four important steps in the qmeans algorithm which allows us to use a Quantum Circuit for distance estimation.

1. Data Normalization
2. Transformation to Bloch Sphere
3. State Preparation in Quantum Circuit
4. Inner Product Estimation Circuit

3.1 Data Preparation

The first step for any machine learning algorithm is data preparation. For this purpose the data is scaled to a range between -1 to 1. This range is specifically chosen and reason for which will be explained in the next section. The data normalization is the standard for kmeans because it relies on crude euclidean distance which is greatly impacted by the scale of the magnitudes and hence it is imperative that the data lies within the same scale.

Figure 3.1 shows min max normalization[7] applied to randomly generated data points. We see that the relative distance between each pair of points is not affected by this normalization.

3.2 Transformation to Bloch Sphere

Each normalized data point is transformed into its Bloch sphere representation, which was explained earlier in section 2.1, i.e we extract the values of θ and ϕ for each data

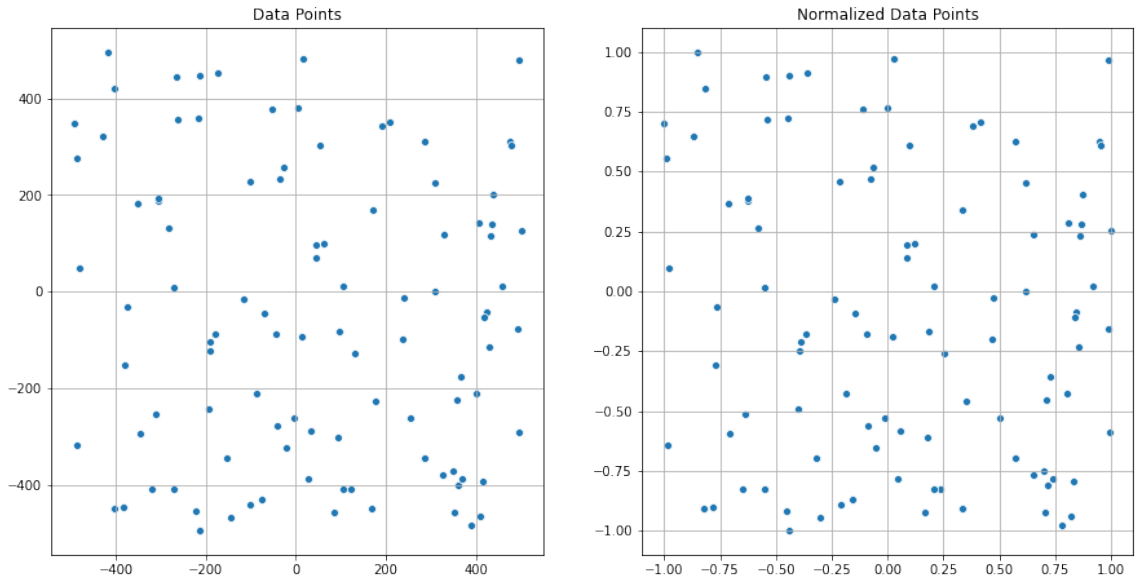


Figure 3.1: Scatter plot of sample Data Points before and after normalization

point which is used in the next step for the preparation of its qubit inside the quantum circuit.

The data-points are transformed using the following formulae from [6],

$$\phi = (feature_0 + 1) \times \frac{\pi}{2} \tag{3.1}$$

$$\theta = (feature_1 + 1) \times \frac{\pi}{2} \tag{3.2}$$

For each data point p_i, ϕ_i and θ_i are extracted, which allows the point to be transformed from the euclidean space to the Bloch Surface. The range that we specified earlier (-1 to 1) could now be explained. Any value outside this range would go beyond the time period of sin and cos which would mean that multiple points would have the same representation on the Bloch surface, which would make the distance estimation to be incorrect.

3.3 Quantum Circuit for Distance Estimation

The distance estimation circuit takes ϕ_p,θ_p,ϕ_c, θ_c as arguments where ϕ_p, θ_p are parameters for the given data point and ϕ_c, θ_c are parameters for the given centroid. Figure 3.2 shows the complete circuit used for the estimation.

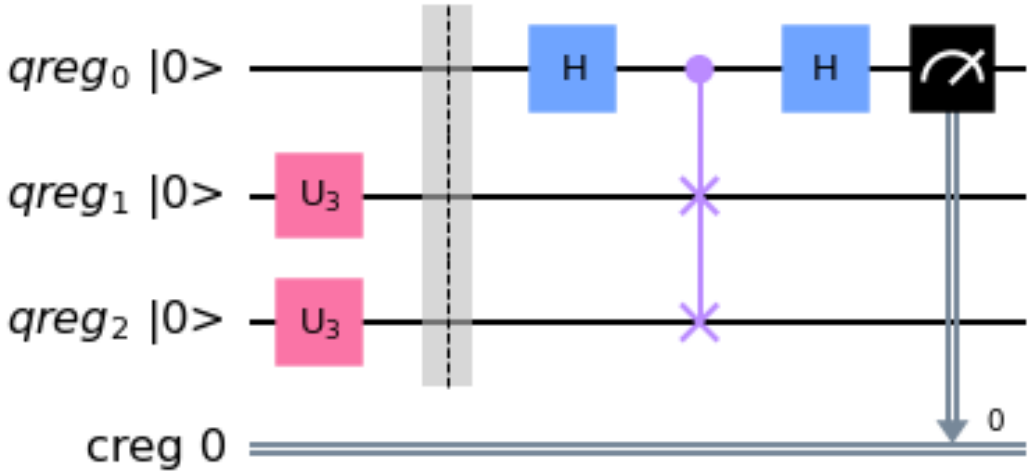


Figure 3.2: Quantum Circuit for Distance Estimation

The circuit first computes prepares the state using U3 gates with parameters ϕ_p,θ_p,ϕ_c, θ_c. The states are then sent for a controlled SWAP test which provides an estimate for the euclidean distance, which is then returned back.

Now with the states prepared for each of the point, the final step that is left is to compute or estimate the inner product of each point with the centroid. We use the SWAP test explained earlier to compute distance of each point with each centroid.

4. Algorithms

4.1 K-means

The K-means algorithm typically works in the following way: Given a set of n data points $D = d_1, d_2, ..., d_n$ in a d-dimensional space, the algorithm tries to group these points in a predefined number of clusters K, each cluster is represented by a centroid C_i ($\forall 0 \leq i < K$). Each centroid is a d-dimensional point which is the mean of all points within the cluster. The algorithm works by minimizing the Euclidean distance between each data point $d \in D$ and cluster C is minimized. The distance measure is defined as:

$$\sum_{k=1}^{K} \sum_{i \in C_k} ||d_i - C_k||^2 \qquad \text{Euclidean distance}$$

The conceptual steps in the algorithm involve the following. Some of these are also highlighted in fig. 4.4:

1. Initialize each cluster with a random centroid
2. For each data point, calculate the Euclidean distance from each cluster point. This is the most expensive operation shown in 4.1.
3. Assign each point to the cluster with the minimum distance (shown in 4.2)
4. Update the centroids by calculating the mean of all points within that cluster. (shown in 4.3)
5. Calculate change between current centroids and previous centroids. If the change > 0 OR current iteration is less than max. iterations, repeat step 2-6.

Input:
Data: An array of tuples where each tuple represents a data point. For e.g. $[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$ where n is equal to number of data points, x_i is equal to x-coordinate of ith data

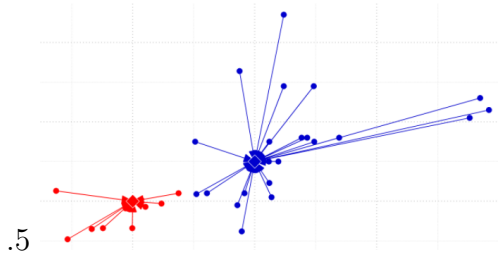


Figure 4.1: Calculate Distance: Distance to cluster from each point

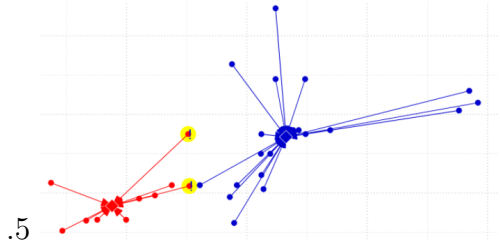


Figure 4.2: Update: Assigning each point to its closest

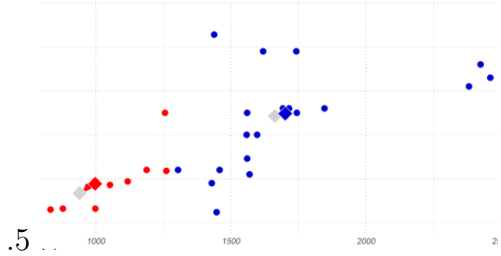


Figure 4.3: Update centroid: Mean of all points in a cluster

Figure 4.4: Operations in K-means

point and y_i is equal to y-coordinate of ith data point.
K: An integer denoting number of clusters.

Output:
Centroids: An array of tuples where each tuple represents a centroid. For e.g. $[(x_1, y_1), (x_2, y_2), ..., (x_Q, y_Q)]$ where Q is equal to number of clusters, x_i is equal to x-coordinate of ith centroid and y_i is equal to y-coordinate of ith centroid.

Complexity
The classical K-means algorithm has a complexity of $O(NMK)$ where M is the

number of data points, N is the dimension of each data point and K is the number of clusters.

4.2 Q-Means

It is understood that in most of the quantum machine learning applications there are some common algorithmic primitives that are used to build the algorithms. This can include the use of quantum procedures for linear algebra in recommendation systems or the freedom to estimate distances between quantum states through, assuming SWAP test [3]. It can be seen that procedures such as these require quantum access to the data. This can be attained by storing the data in a data structure in Quantum Random Access Memory (QRAM). Hence we assume that the data is in QRAM otherwise the performance of the algorithm would be greatly affected.
The Q-Means algorithm has the following signature:

Input:
Data: An array of tuples where each tuple represents a data point. For e.g. $[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$ where n is equal to number of data points, x_i is equal to x-coordinate of ith data point and y_i is equal to y-coordinate of ith data point.
Q: An integer denoting number of clusters.

Output:
Centroids: An array of tuples where each tuple represents a centroid. For e.g. $[(x_1, y_1), (x_2, y_2), ..., (x_Q, y_Q)]$ where Q is equal to number of clusters, x_i is equal to x-coordinate of ith centroid and y_i is equal to y-coordinate of ith centroid.

The algorithm works as follows,

1. Initialize a new array **Normal Data** having the same shape as **Data**.
2. Normalize each data point such that x_i and y_i are within the range (-1, 1) as shown in Figure3.1 and store them in **Normal Data**.
3. Initialize a new array **Transformed Data** having the same shape as **Data**.
4. Extract θ_i and ϕ_i for each data point in **Normal Data** such that $\theta_i = (x_i + 1)\frac{\pi}{2}$ and $\phi_i = (y_i + 1)\frac{\pi}{2}$[6]. Store all θ_i and ϕ_i in **Transformed Data**.
5. Initialize a new array **Centroids** which stores **Q** tuples with values distributed randomly between -1 and 1.

6. Initialize a new array **Clusters** which stores **n** integers. $\textbf{Clusters}_i$ denotes the cluster of ith data point.

7. For each data point in **Transformed Data** compute its distance with each centroid in **Centroids** using the distance estimation circuit defined in section 3.3. Assign each data point to the closest cluster by assigning **Clusters[i]** = j where ith data point is closest to jth Centroid.

8. Recompute all centroids, i.e. assigning **Centroids[j]** is equal to mean of all data points for which **Clusters** is j, i.e. mean of all data points belonging to cluster j.

9. Check if the magnitude difference between newly computed centroids and old centroids is greater than a defined threshold t. If it is, repeat step 7-9, if it is not, return the array **Centroids**.

Complexity
The Q-means algorithm has a complexity of $O(log(N)MQ)$ where M is the number of data points, N is the dimension of each data point and Q is the number of clusters.

5. Experimentation and Analysis

We performed unsupervised clustering using both k-means and q-means and performed a comparative analysis of both the algorithms. Below, we discuss the experimentation and analysis performed.

5.1 Dataset Preparation

For our experimentation, data was prepared in two difference ways. One was to use a well clusterable dataset as described in [4] using normal distribution generator of numpy with k different means, where k is the number of clusters. The other dataset was generated as a uniform random distribution. The reason for two different datasets is to see how both classical and quantum kmeans fare against each other on different types of datasets.

5.2 Cluster Formation and Visualization

We provide a visualization of our final clusters and also graphical visualization of the process of formation of clusters by plotting them at each iteration, for both uniformly distributed data and data with normal distribution. We performed both algorithms on various counts of clusters (k) and count of data points. Here we show cluster formation for k = 3, and 200 data points. Figure 5.1 shows our uniformly distributed data with the said parameters before clustering. Figure 5.2 and figure 5.3 shows k-means and q-means applied on data shown in figure 5.1. Here, we can see that both k-means and q-means identify slightly different clusters but have similar centroids. Both converge with visually similar cluster formation.

Figure 5.4 shows our normally distributed data with the said parameters before clustering. Figure 5.5 and 5.6 shows k-means and q-means applied on normally distributed data for the same parameters i.e. k = 3 and 200 rows. We can see that

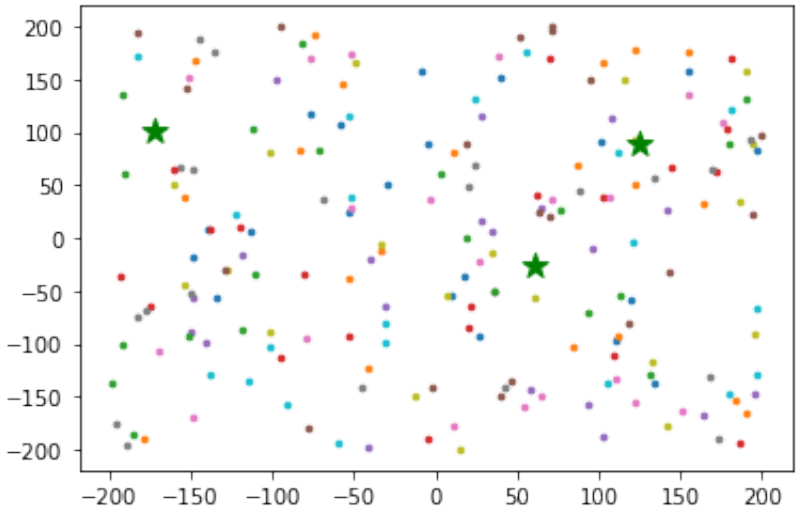


Figure 5.1: Uniform data, k=3, 200 data points

k-means takes 5 iterations and q-means 3 to converge and they converge with slightly different clusters and centroids. Possible reasons for not completely similar clusters in both normal and uniform data can be the difference of distance estimators such that q-means uses probabilistic distance estimator which is hence not accurate. Also, the randomness in both algorithm may have also impacted output which, as random values change at every run.

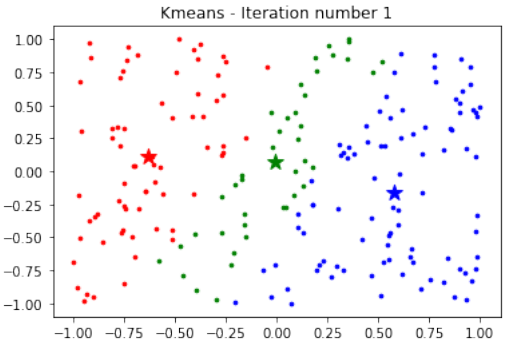
5.3 Analysis

The cluster quality was analyzed using the mean L1 norm from the centroid for each data point.

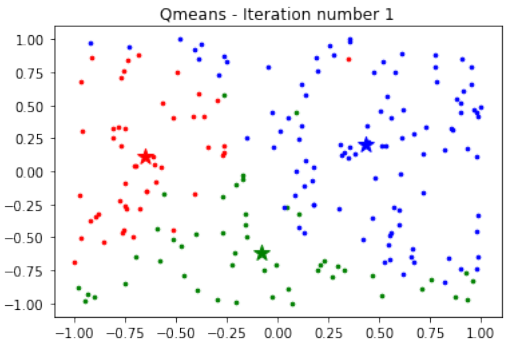
5.3.1 Uniform Distribution

A uniform random distribution was generated between. The scatter plot of which can be seen below,

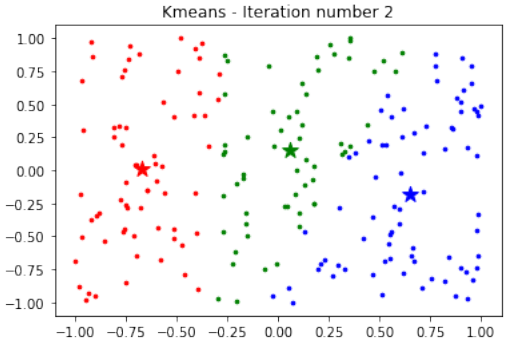
Figure 5.8(a) and 5.8(b) show cluster formed by kmeans and qmeans respectively when run with k = 3. We can see that both algorithms create completely different clusters, but this is not because of the difference in qmeans or kmeans this is because of the nature of the algorithms and random selection of centroids.



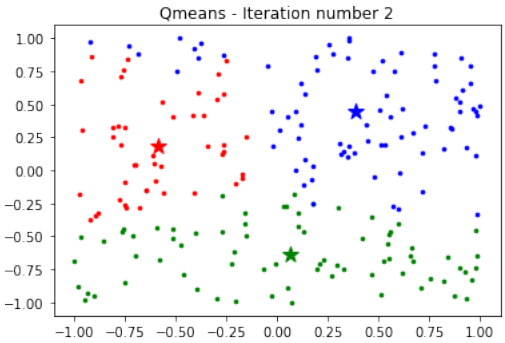
(a) K-means iteration 1



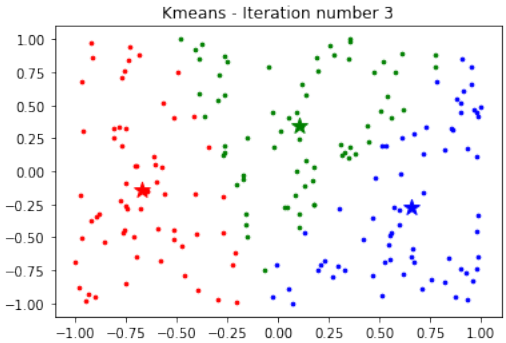
(b) Q-means iteration 1



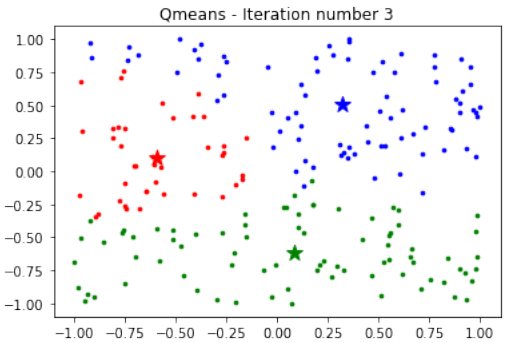
(c) K-means iteration 2



(d) Q-means iteration 2

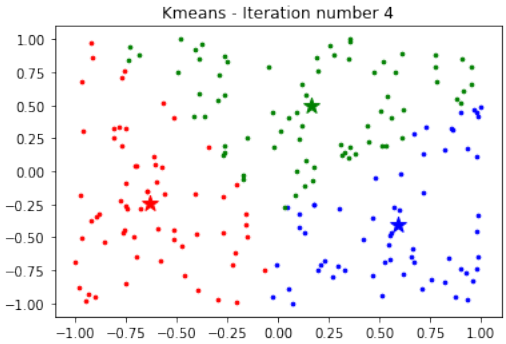


(e) K-means iteration 3

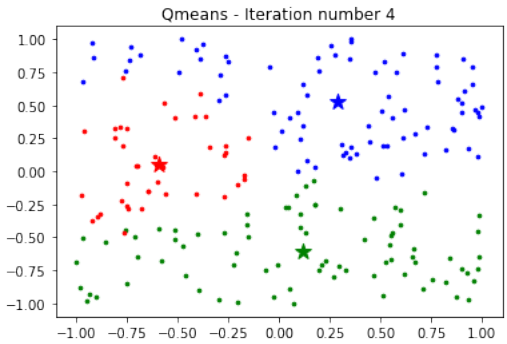


(f) Q-means iteration 3

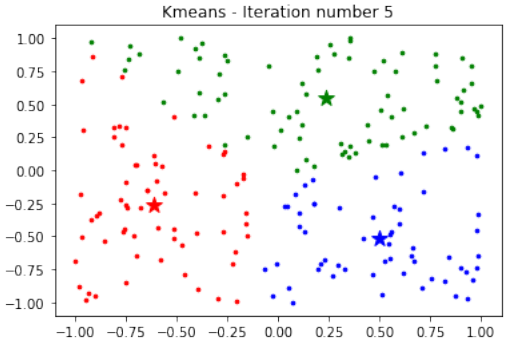
Figure 5.2: Cluster formation for uniform data. 1-3 iterations



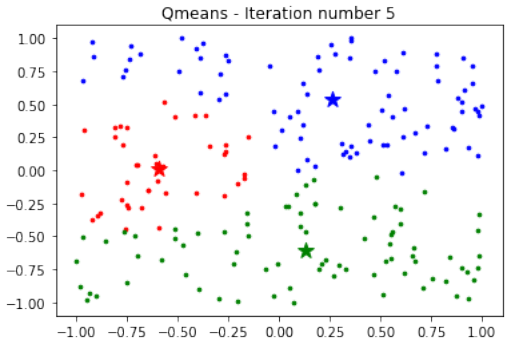
(a) K-means iteration 4



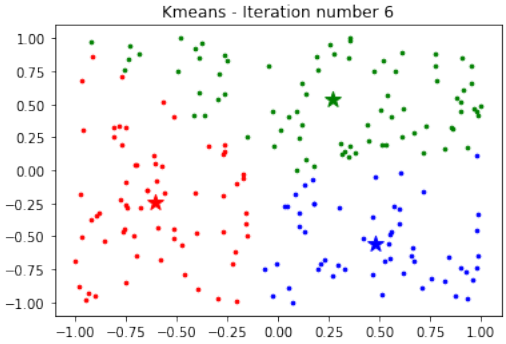
(b) Q-means iteration 4



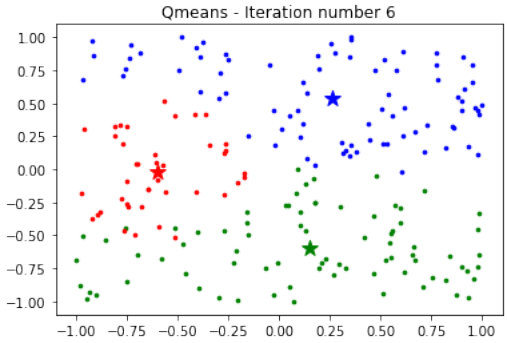
(c) K-means iteration 5



(d) Q-means iteration 5



(e) K-means iteration 6



(f) Q-means iteration 6

Figure 5.3: Cluster formation for uniform data. 1-6 iterations

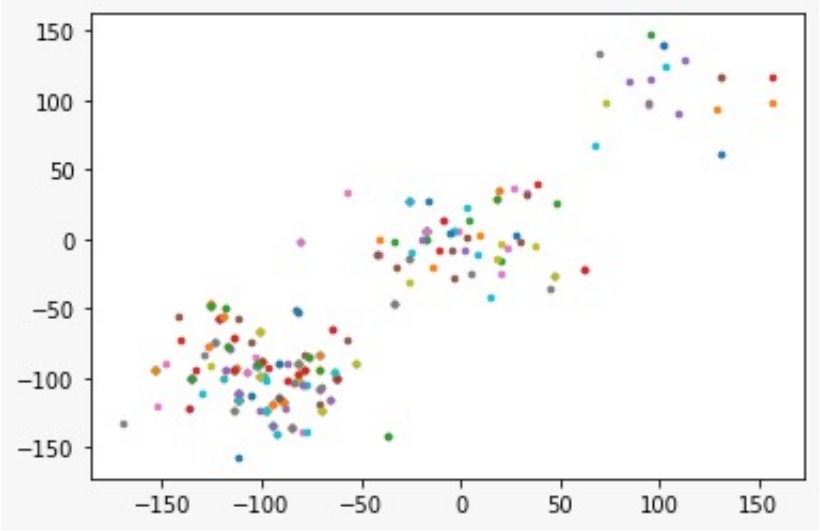


Figure 5.4: Normally distributed data data, k=3, 200 data points

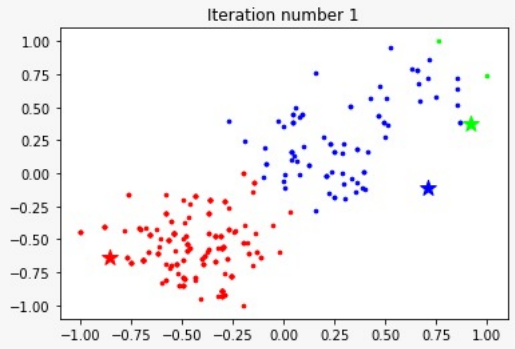
Figure 5.8(c) and Figure 5.8(d) show average L1 norm of each centroid to its data-points. While there is neglible difference, we do see that Qmeans has a slightly higher error rate. This could also be seen from Figure 5.8(b) where all the points are not assigned to the closest cluster.

5.3.2 Normal Distribution

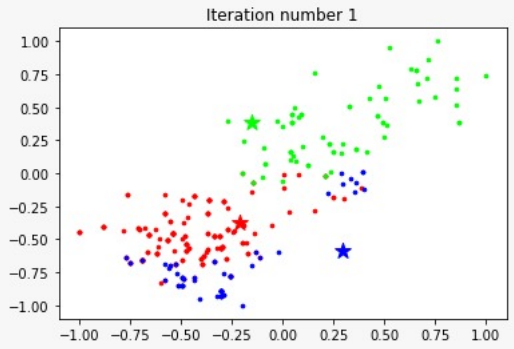
A normal random distribution was generated with 3 means. The scatter plot of which can be seen below,

Figure 5.8(a) and 5.8(b) show cluster formed by kmeans and qmeans respectively when run with k = 3. We can see that both algorithms create completely different clusters.

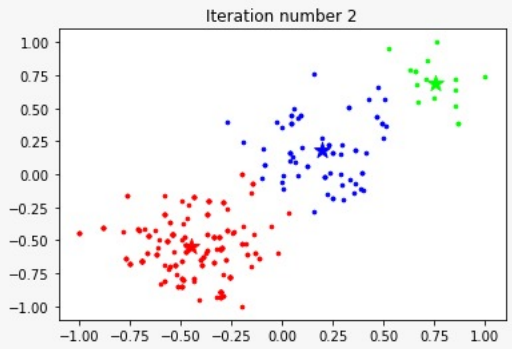
Figure 5.9(c) and Figure 5.9(d) show average L1 norm of each centroid to its data-points. Qmeans has lesser error than the uniform distribution, but there is still slight difference in performance between classical and quantum kmeans with classical providing fractionally better results.



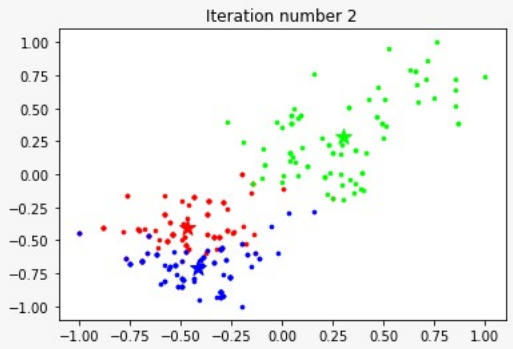
(a) K-means iteration 1



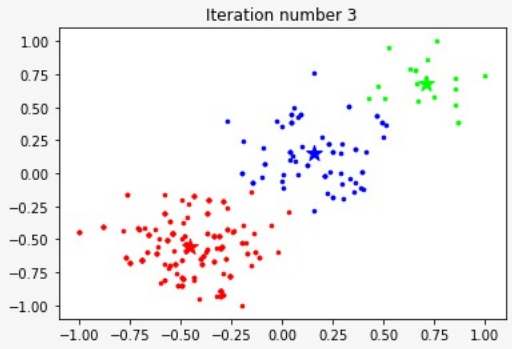
(b) Q-means iteration 1



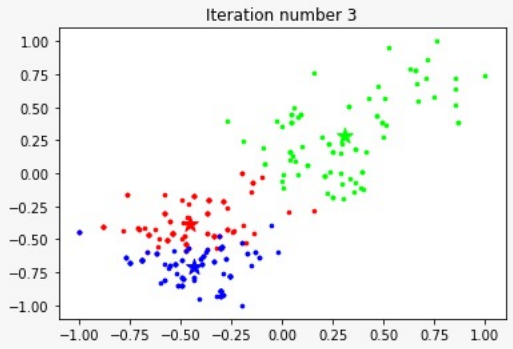
(c) K-means iteration 2



(d) Q-means iteration 2

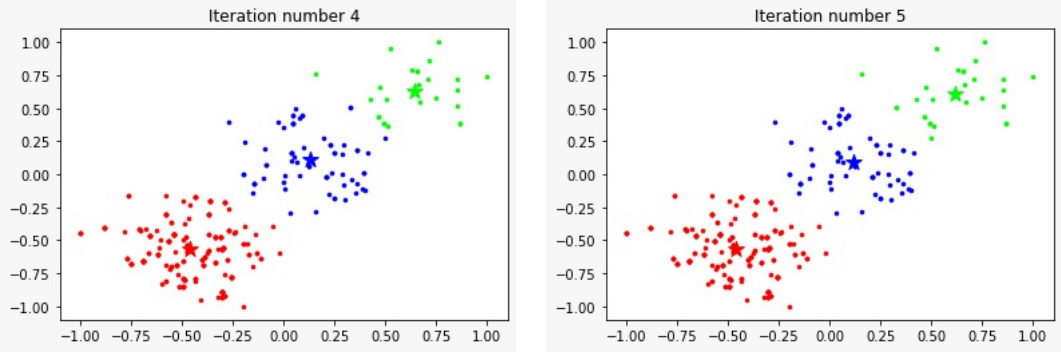


(e) K-means iteration 3



(f) Q-means iteration 3

Figure 5.5: Cluster formation for normal data.



(a) K-means iteration 4 (b) K-means iteration 5

Figure 5.6: Cluster formation for normal data.

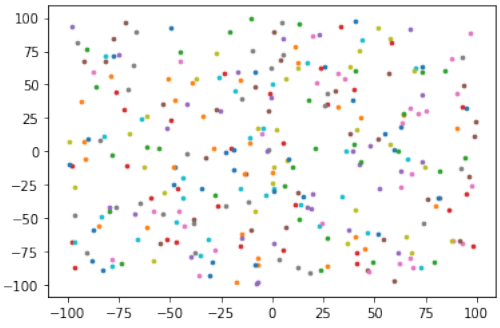
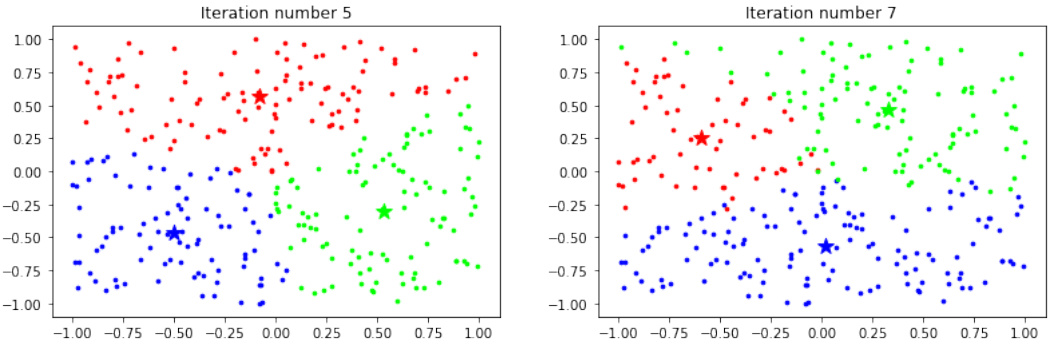
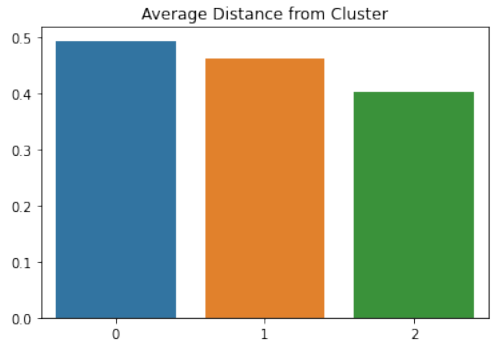


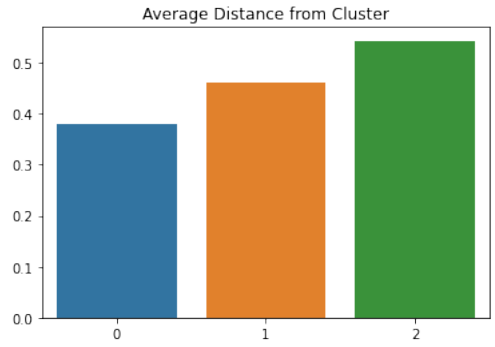
Figure 5.7: Random Uniform Distribution



(a) K-means clusters 1 (b) Q-means Clusters 1



(c) Average L1 norm of each centroid from its data points: Kmeans



(d) Average L1 norm of each centroid from its data points: QMeans

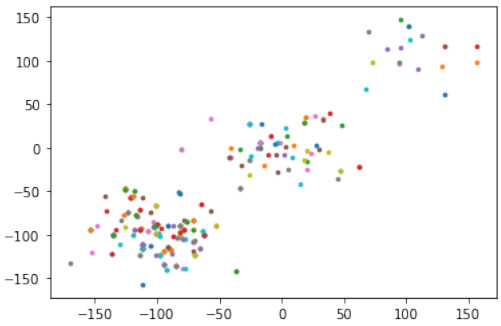
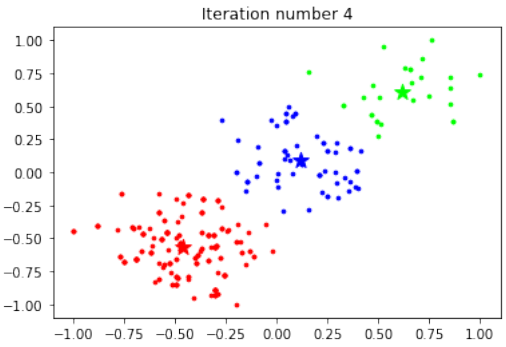
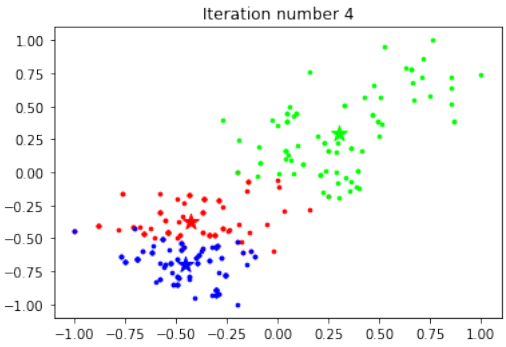


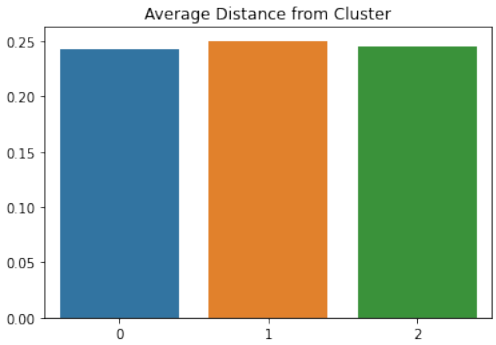
Figure 5.8: Random Multivariate Normal Distribution



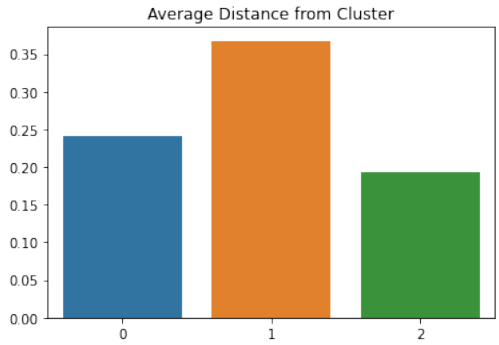
(a) K-means clusters 1



(b) Q-means Clusters 1



(c) Average L1 norm of each centroid from its data points: Kmeans



(d) Average L1 norm of each centroid from its data points: QMeans

6. Conclusion

This project aimed to highlight the possibility of K-means using quantum computing. The implemented quantum version of K-means is theoretically more efficient, and in terms of results almost similar to classical K-means with negligible difference in performance. The main takeaway from this project is the idea of representing data from real life into its quantum representation and then getting tangible results from it. The performance of this algorithm could be further optimized using QRAM which would provide a further speedup in terms of time.A possible extension of the project could be the inclusion of quantum procedures in other steps of the algorithm such as centroid update.

References

[1] *Introduction to quantum computing: Bloch sphere.* URL: http://akyrillidis.github.io/notes/quant_post_7.

[2] Abhijith J. et al. *Quantum Algorithm Implementations for Beginners*. 2020. arXiv: 1804.03719 [cs.ET].

[3] Iordanis Kerenidis et al. *A Quantum Algorithm for Unsupervised Machine Learning*. 2018. arXiv: 1812.03584 [cs.ET].

[4] Iordanis Kerenidis et al. *q-means: A quantum algorithm for unsupervised machine learning*. 2018. arXiv: 1812.03584 [quant-ph].

[5] Dawid Kopczyk. *Quantum machine learning for data scientists*. 2018. arXiv: 1804.10068 [quant-ph].

[6] David Morcuende-Cantador. *Morcu/q-means*. URL: https://github.com/Morcu/q-means.

[7] *sklearn.preprocessing.MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.