Programming Assignment 3

Zoheir El Houari

Markus Zimmball

Introduction:

This assignment focuses on exploring different approaches to approximately solve the k-means clustering problem while improving both accuracy and runtime performance. We begin with the implementation of Lloyd's algorithm as a baseline, followed by enhancements using Locality Sensitive Hashing (LSH) and coresets. By comparing the three variants of k-means clustering, we aim to understand the trade-offs between accuracy and computational efficiency. The results will provide valuable insights for real-world applications requiring scalable and efficient clustering algorithms.

Task 1: Lloyd's algorithm for k-Means Clustering

Implementation:

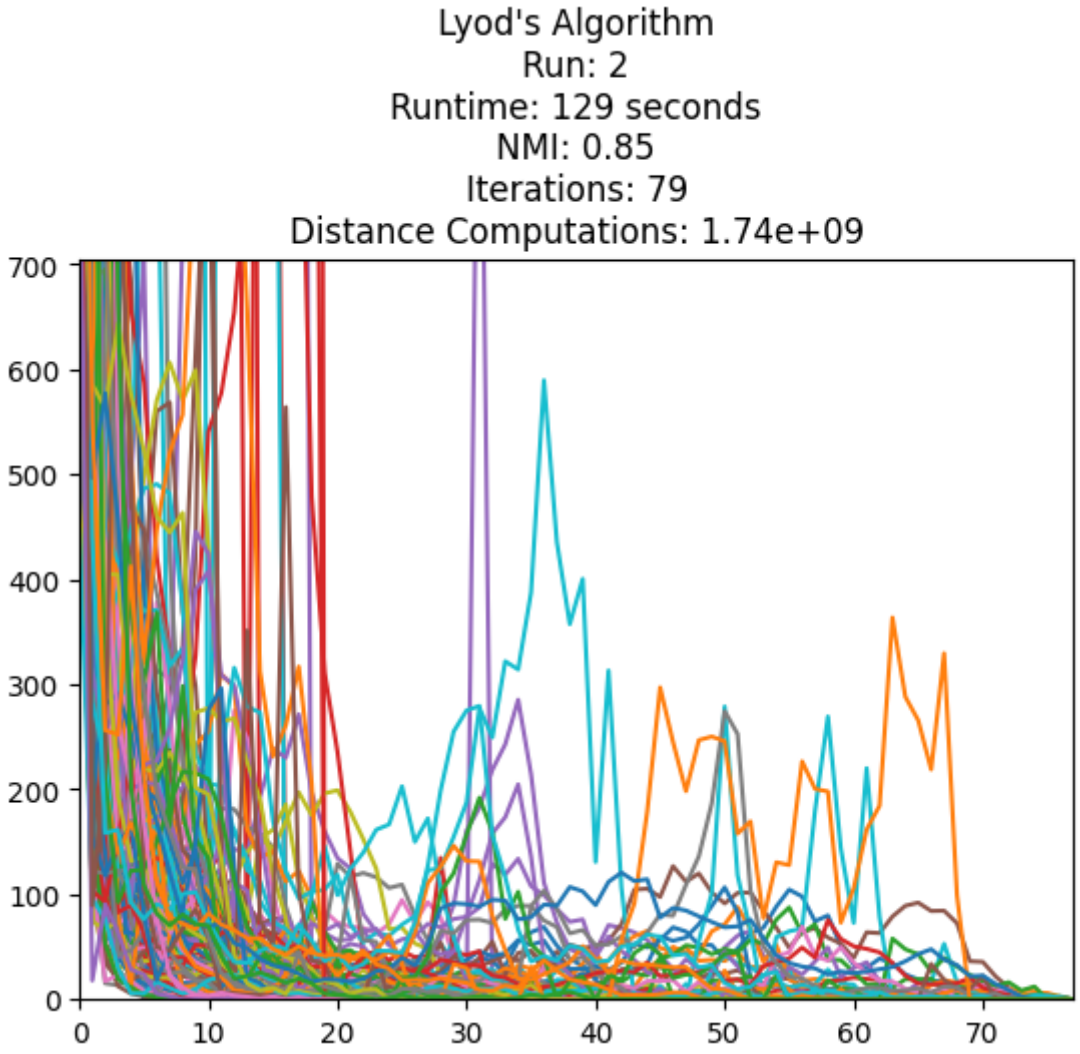
Lloyd's algorithm is a popular iterative algorithm for K-means clustering. Here's a short description of its implementation:

- Initialize the centroids: Randomly select K data points from the dataset as initial centroids.
- Assign data points to centroids: For each data point, compute its distance to each centroid and assign it to the nearest centroid.
- Update centroids: Recalculate the centroids by taking the mean of all data points assigned to each centroid.
- Repeat steps 2 and 3 until convergence: Iterate steps 2 and 3 until the centroids no longer change significantly or a maximum number of iterations is reached.

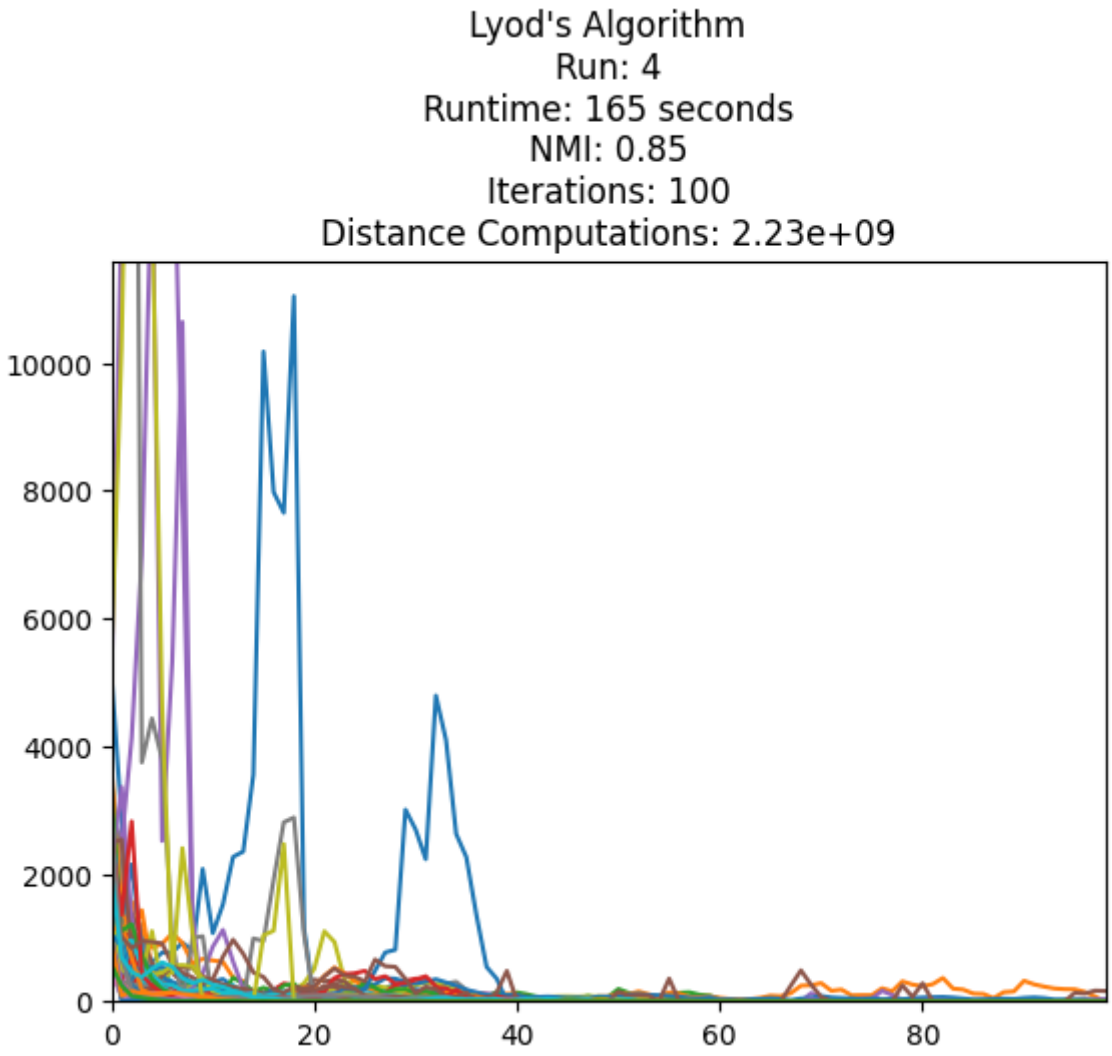
Output: The final centroids represent the K clusters, and each data point is assigned to the cluster with the nearest centroid.

Plots:

Each plotted line represents a cluster where the x-axis represents the iteration count and the y-axis is the shows the distance of cluster centroids between iterations.



One can see in the next plot that the selection inital cluster centroids matters as it seems like the algorithm is alternating centroids in a local optimum and therefore maxing out in the number of iterations.



Task 2 : k-Means with Locality Sensitive Hashing (LSH)

Implementation

Our KMeansLSH implements the kmeans clustering with Locality sensitive hashing used in the fitting process.

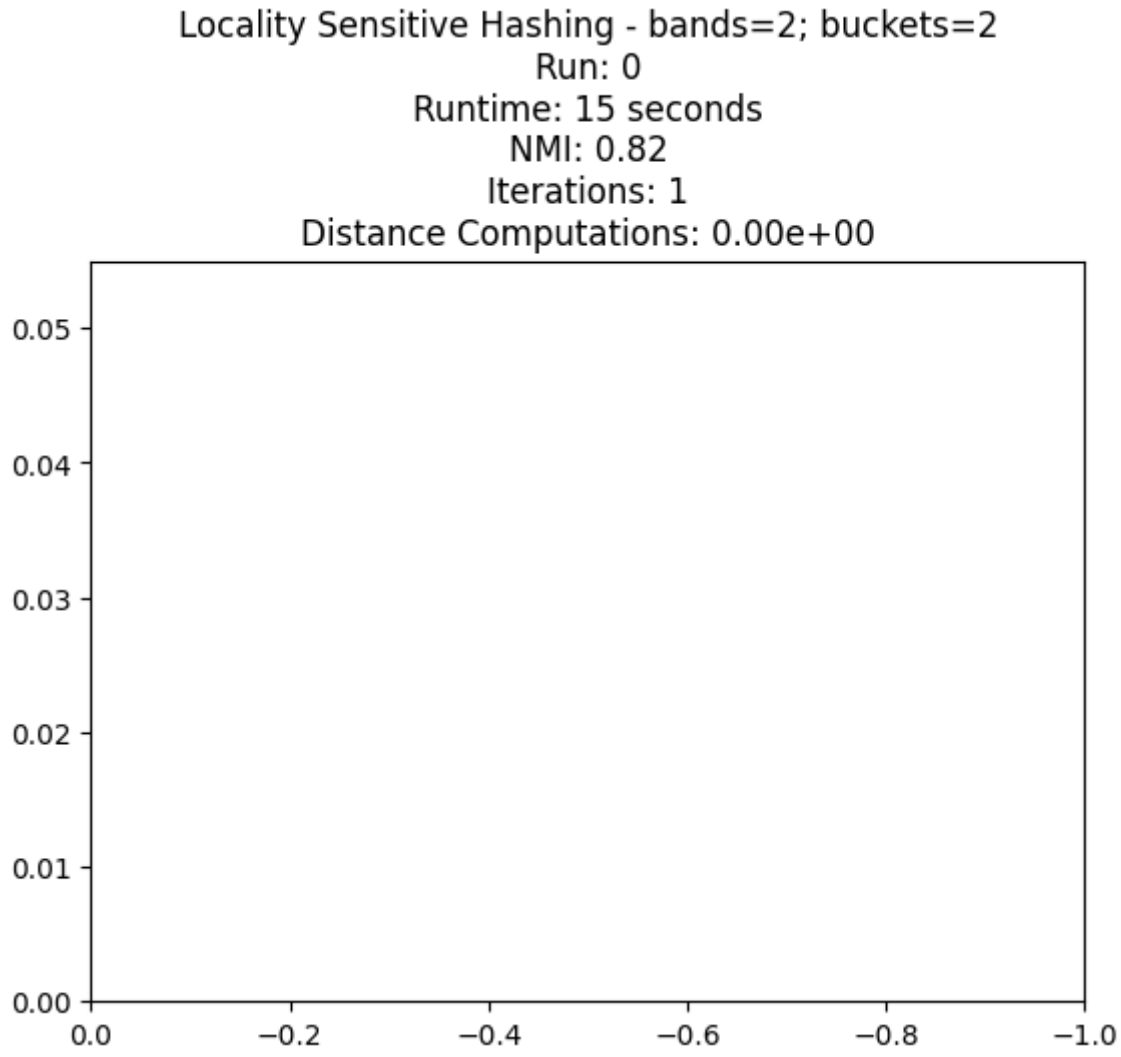
it uses hash functions and hash tables to efficently assign data points to centroids and update the centroids.

Here is a brief overview:

- The **HashFunction** class: computes hash values using random coefficients and bias.
- The **HashTabl**e class stores items in buckets based on their hash values.
- The **LocalitySensitiveHashing** class performs LSH by using multiple hash tables to find similar items.

- The **KMeansLSH** class combines K-means with LSH, assigning data points to centroids and updating centroids using LSH results.

Plots:



we can see clearly that the KmeansLSH out perform LIyod's algorithm when it comes to runtime and number of iterations needed for convergence. if convergence after only one iteration.

Task 3: k-means with coresets

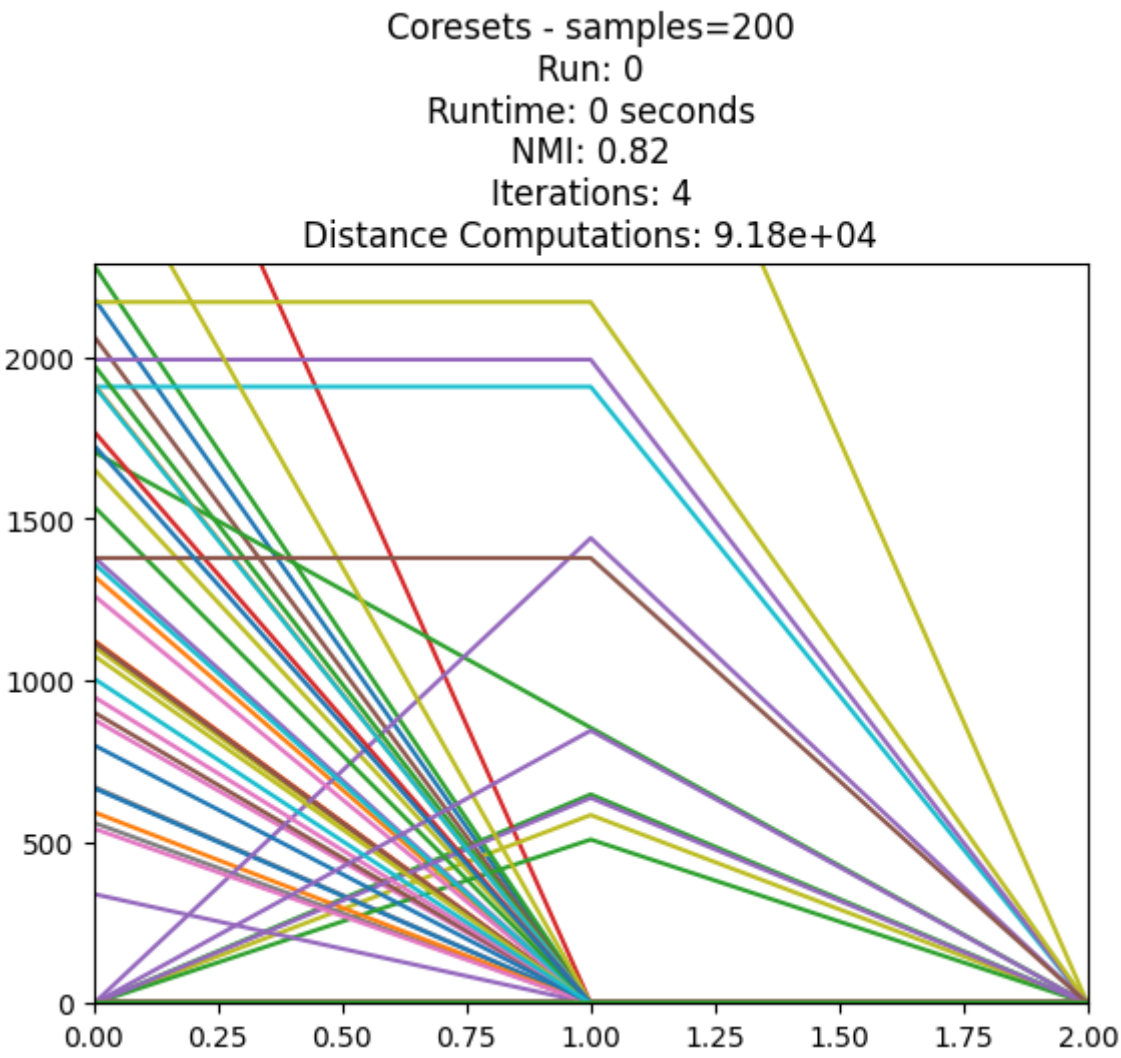
Implementation

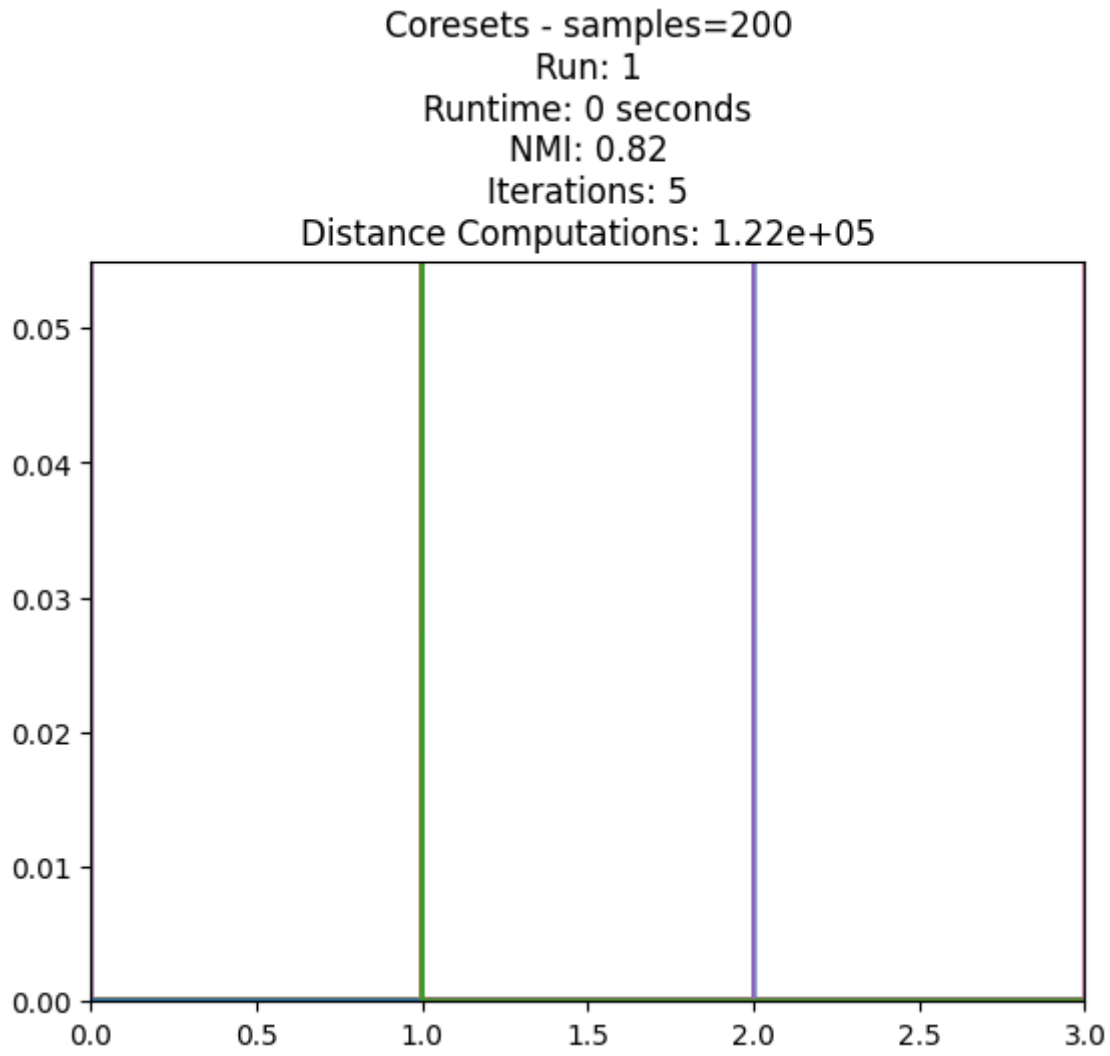
The k-means implementation with coresets works in a very similar way compared to lyod's algorithm as the only difference is the fitting phase.
Instead of the whole dataset this implementation only takes a random subset of certain sample size as training data.

Plots:

Samplesize=200

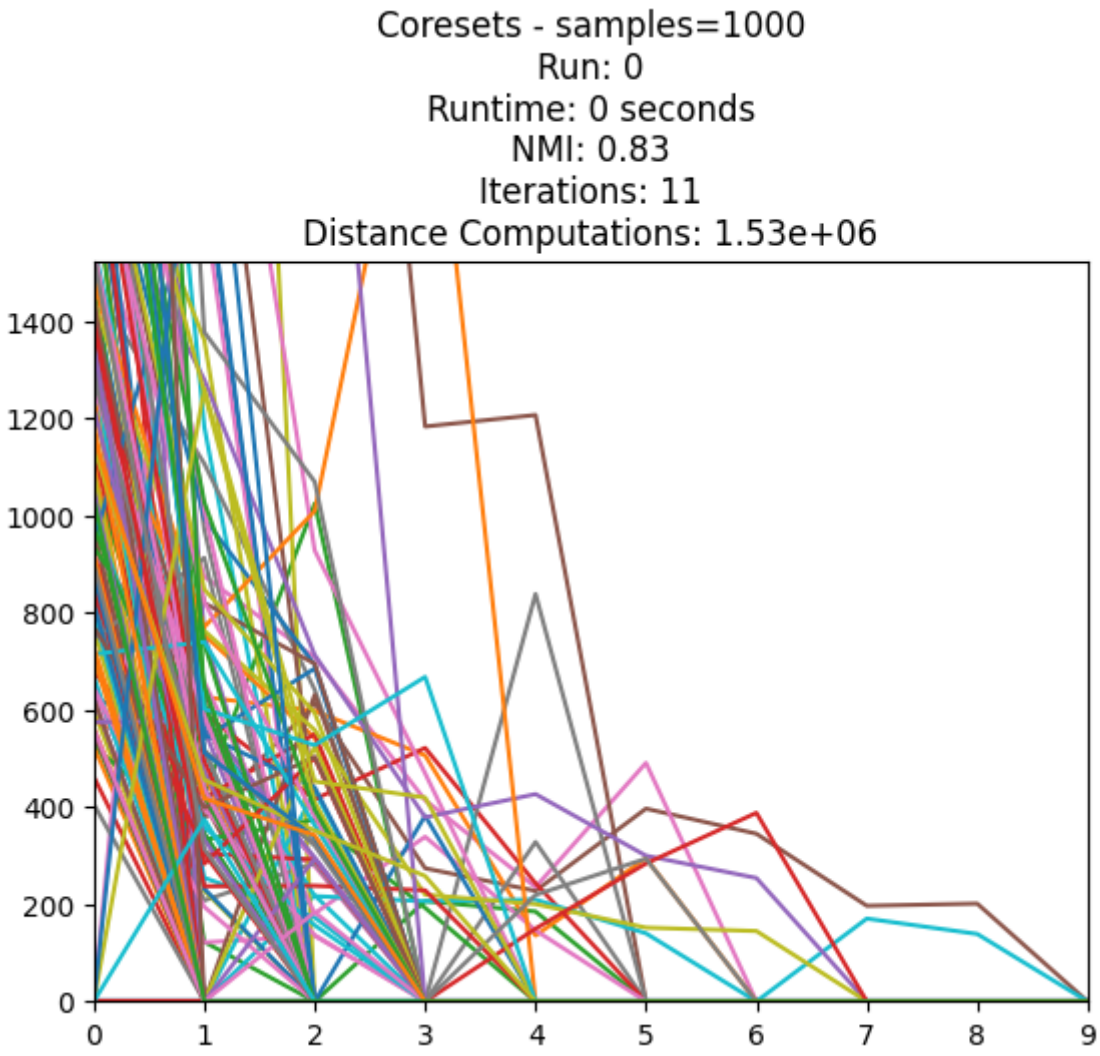
Its clear to see that with a low sample size size convergence happens amlost immediately with 2 to 3 iterations. Also one thing to note is that depending on the inital data and initial cluster centroids the centroid dostistance can be instantaneous.


Coresets - samples=200
Run: 0
Runtime: 0 seconds
NMI: 0.82
Iterations: 4
Distance Computations: 9.18e+04
2000
1500
1000
500
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



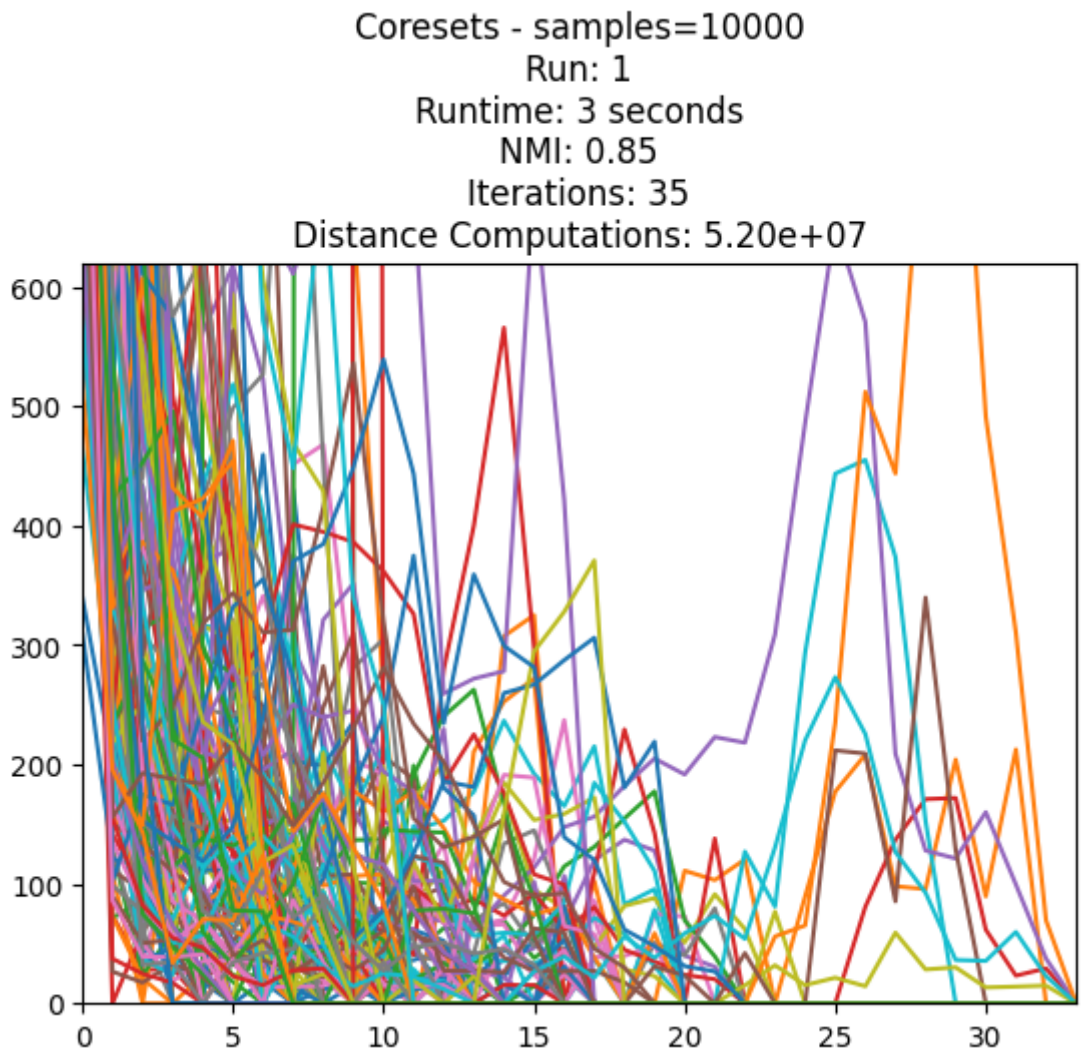
Samplesize=1000

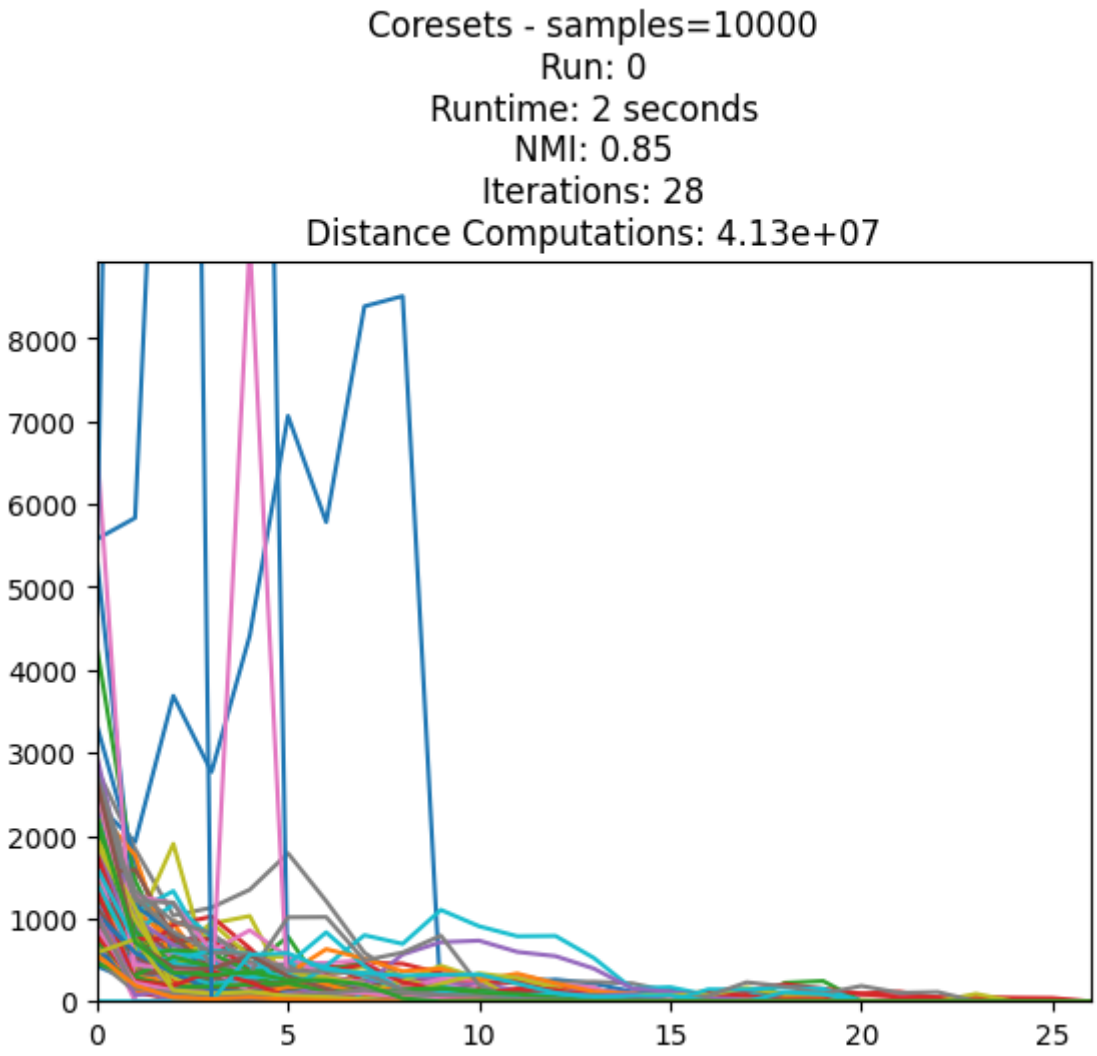
When increasing the samplesize the iterations count increases together with the



Samplesize=10000

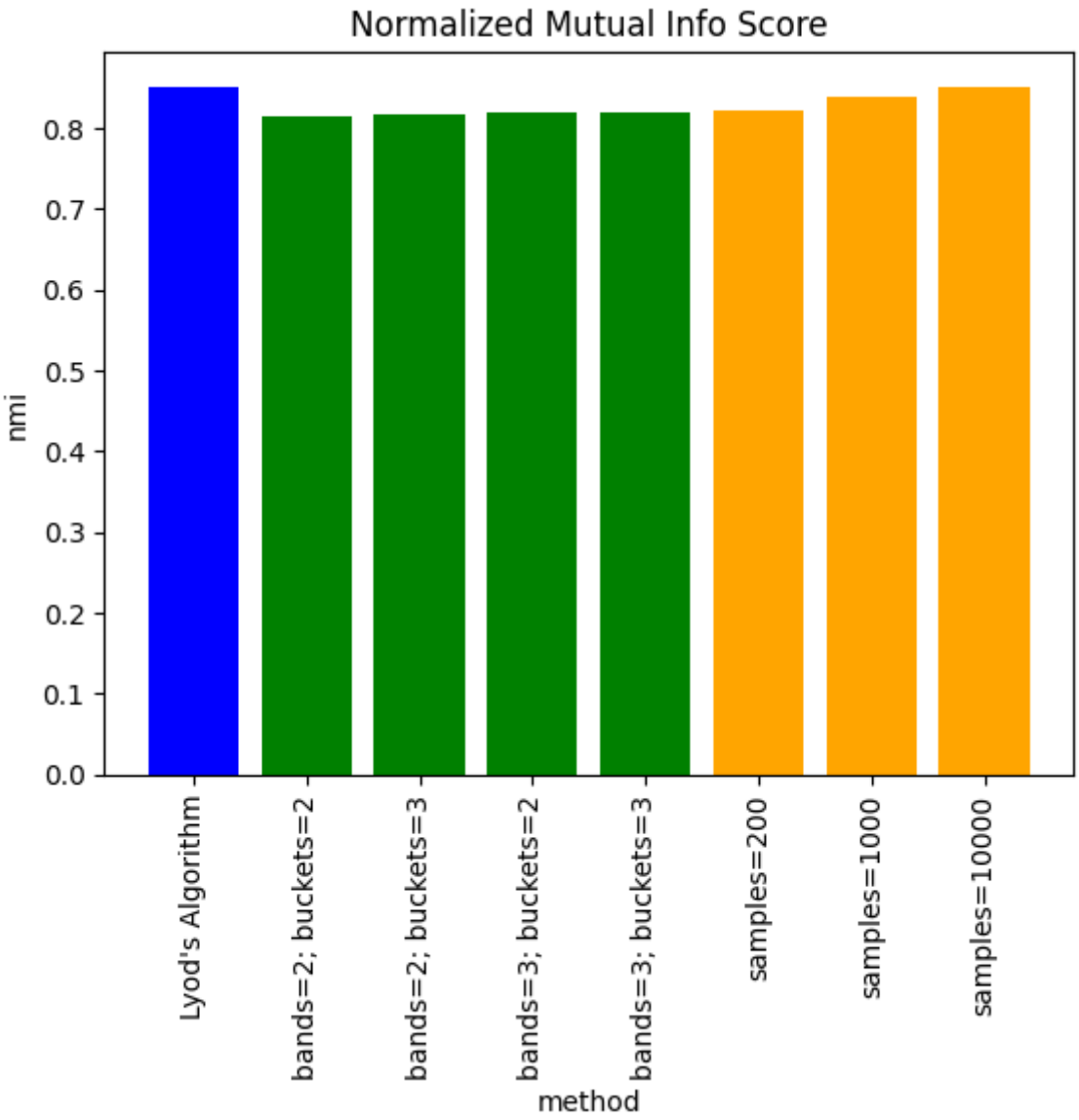
Comparing the next two plots one can again see that the initial set of centroids and sample data affects the number of iterations, runtime and also accuracy


Coresets - samples=10000
Run: 1
Runtime: 3 seconds
NMI: 0.85
Iterations: 35
Distance Computations: 5.20e+07
600
500
400
300
200
100
0
0
5
10
15
20
25
30

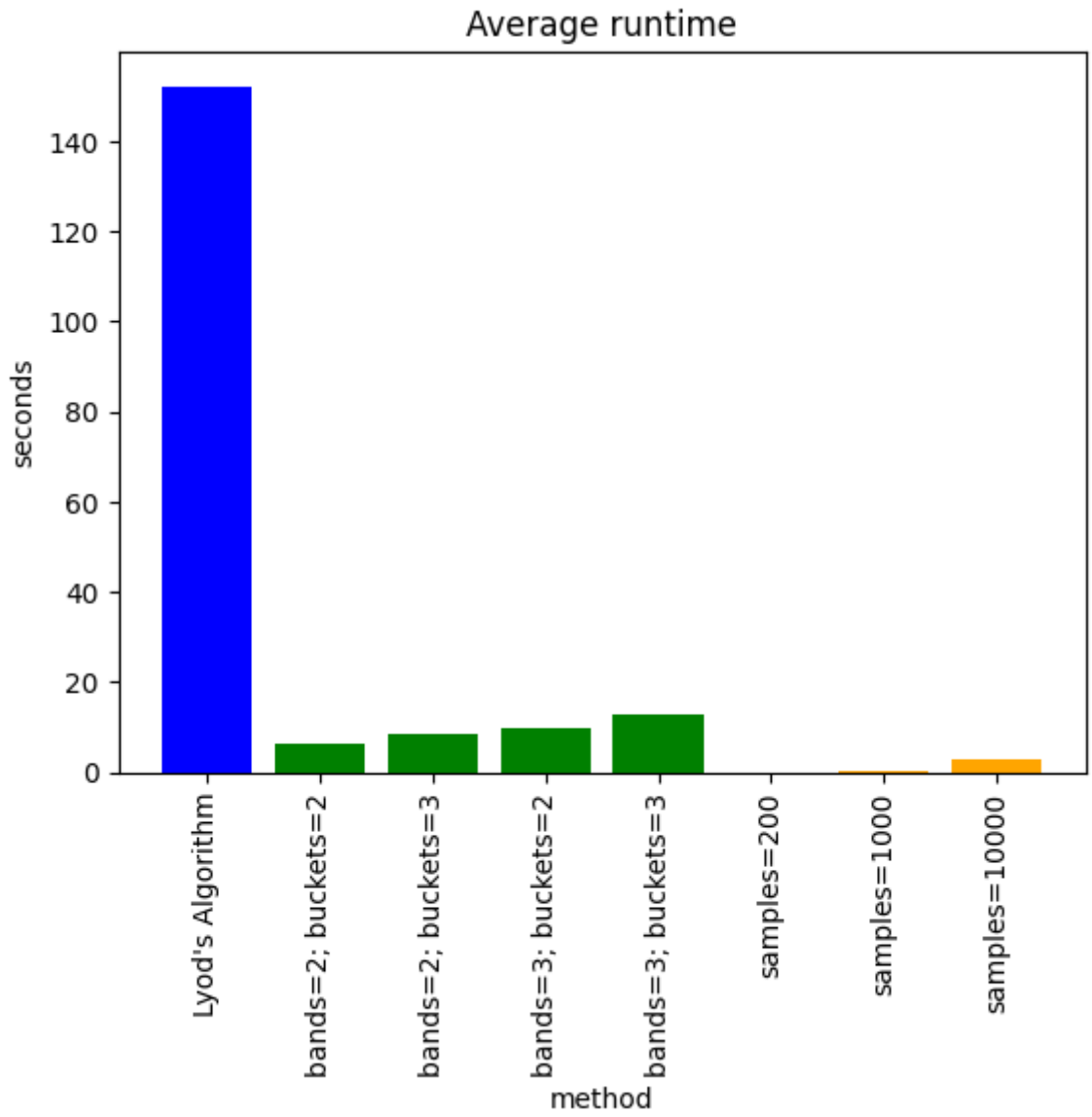

Coresets - samples=10000
Run: 0
Runtime: 2 seconds
NMI: 0.85
Iterations: 28
Distance Computations: 4.13e+07
8000
7000
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25

Benchmarks:

Normalized Mutual Information



Runtime



Interpretation

Lyods algorithm as a base line performs the best but also takes quite some time to run, as it runs through the number of maximum interations. The KMeansLSH implementation on the other hand performs about 5% worse due to it’s reliance on hash tables which may be unsufficient to accuratly capture the complex relationship between the data points.
The Coreset implementation performs better than LSH because it does not reduce the feature space and also runs significantly faster as its does not have to hash every data point.

Results:

Approach	LIyod’s Algorithm	KMeansLSH	Coresets
Runtime	73 seconds	3 seconds	1 second
NMI	0.85	0.82	0.81

Work Distribution:

We met 3 times and did pair programming since the tasks we dependent on each other and splitting the work didn’t seem feasable.

time spent by Zoheir El Houari: 20 hours
time spent by Markus Zimmball: 20 hours

References:

*https://dl.acm.org/doi/10.1145/3219819.3219973 (https://dl.acm.org/doi/10.1145/3219819.3219973)
*https://dl.acm.org/doi/10.1145/1007352.1007400 (https://dl.acm.org/doi/10.1145/1007352.1007400)
*https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134 (https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134)