Clustering Experimental Analysis

First we have to note that the experimental analysis has been perfomed with the MNIST Digit Dataset.

Performance

The results presented below are averaged over 25 runs for each algorithm separately. The main function that implements those experiments can be found here.

- Lloyds Algorithm (classic Brute-Force approach)
  - Average Clustering Time: **6.09 seconds**
  - Average Total Silhouette (s_total): **0.0988**
- Reverse Assignment with LSH Range Search (L = 4, k = 4)
  - Average Clustering Time: **4.92 seconds**
  - Average Total Silhouette (s_total): **0.953**
- Reverse Assignment with Hypercube Range Search (M = 10, k = 14, probes = 3)
  - Average Clustering Time: **5.54 seconds**
  - Average Total Silhouette (s_total): **0.0988**

Hyperparameters

- Number of Clusters **K**

  We know a priori that the Dataset consists of 10 different "groups" of data points (digits). Therefore, an optimal value for the number of clusters **K** would be 10. Though it would be a nice idea to experiment with this number in order to see how the silhouette changes. This experiment in implemented is this main cpp file here.

  - **K** = 8

    - Classic
      - Clustering time: **5.66 seconds**
      - Total Silhouette: **0.100**
    - LSH
      - Clustering time: **3.97 seconds**
      - Total Silhouette: **0.097**
    - Hypercube
      - Clustering time: **4.60 seconds**
      - Total Silhouette: **0.100**

  - **K** = 9

    - Classic
      - Clustering time: **4.67 seconds**
      - Total Silhouette: **0.092**
    - LSH
      - Clustering time: **3.01 seconds**
      - Total Silhouette: **0.089**
    - Hypercube
      - Clustering time: **3.38 seconds**
      - Total Silhouette: **0.092**

  - **K** = 10

    - Classic
      - Clustering time: **7.16 seconds**
      - Total Silhouette: **0.103**
    - LSH
      - Clustering time: **3.809 seconds**
      - Total Silhouette: **0.098**
    - Hypercube
      - Clustering time: **5.67 seconds**
      - Total Silhouette: **0.103**

  - **K** = 11

    - Classic
      - Clustering time: **6.35 seconds**
      - Total Silhouette: **0.105**
    - LSH
      - Clustering time: **3.93 seconds**
      - Total Silhouette: **0.102**
    - Hypercube
      - Clustering time: **5.78 seconds**
      - Total Silhouette: **0.105**

  - **K** = 12

    - Classic
      - Clustering time: **6.62 seconds**
      - Total Silhouette: **0.111**

        - LSH
            - Clustering time: **4.01 seconds**
            - Total Silhouette: **0.106**
        - Hypercube
            - Clustering time: **5.05 seconds**
            - Total Silhouette: **0.111**

By examining this results, we can see that one thing stands out; By increasing the number of clusters **K**, the silhouette increases. This makes sense, because for more clusters, there are more centroids for the various data points to be assigned to. Even though we have 10 digits, some people could make a "7" look like a "1", or a "4" look like a "9". Adding more clusters, helps identify these corner cases, and therefore reduces the sihlouette. Adding too many clusters though would not make much sense, as one could argue that it would result in "overfitting", that is, to learn the data points in the training set too well, and in the end not being able to generalize.

Apart from this, The truth is that from these experiments not many conclusions can be drawn, as the random initialization plays a big role in the performance of any algorithm. Probably, if some parameters get tweaked e.g.: *range* of the range search, *tolerance* when considering if a centroid has "changed", window size W, better performance can be achieved. Below we list some of the parameters that can change.

- Range of the Range Search
    - By increasing the range in the Range Search, on the one the hand we manage to get more data points for every ball, which leads to less time comparing each data point with every cluster. On the other hand many points lie in the same balls, so we have to resolve many conflicts which leads to quite some overhead. In the end, the improvement is minimal.
    - By decreasing the range in the Range Search, on the one hand we reduce the number of conflicts to resolve, but on the other hand not many points are "found" by the balls, so we have to spend some time after in order to add those manually using Lloyds Algorithm. We could argue that decreasing the radius actually makes the algorithm run slower.
    - Hence, a balance in the radius should be found, and this is why it is hard to tune this parameter.

- Tolerance when updating centroids
    - By increasing the tolerance in any algorithm, convergence actually becomes faster, since many centroids tend to not change a lot after some iterations. The drawback is that we lose in accurracy (silhouette value actually becomes smaller).
    - By decreasing the tolerance, we make the algorithm more "harsh" demanding that the clusters actually converge near their minimum (either local or global), which increases the accurracy (silhouette), but requires more iterations, and therefore more time.
    - Thus, again the values for this paramater are limited, and it should be picked with care. In our implementation, we have assigned *tolerance* = 3000, which actually makes sense considering that if only 15 out of 784 pixels change completely, we have reached the 3000 mark.

- LSH and Hypercube Hyperparameters
    - LSH L: By increasing the number of Hash Tables, we improve the accurracy but we make the algorithm run slower. The opposite happens if we decrease it.
    - LSH k: By increasing the number of hash functions up until a point, we gain some accuraccy.
    - Hypercube k: By increasing the dimension of the projection, we increase quite much the accuraccy but there is a big penalty on the running time.
    - Hypercube probes: By increasing the number of probes, again the accuraccy is increased, up until a point.
    - Hypercube M: By increasing the number of candidate points to be checked, we gain a boost in accurracy, but lose a bit of speed perfomance.
    - Basically, almost all of them if we increase them we gain accurracy but lose speed and vice-versa.