K-means and medoids clustering

Axel Orrhede

November 2024

1 K-means clustering the MNIST dataset

The MNIST dataset consists of 70000 handwritten digits in 28x28 pixel images as shown on the title page[1]. This paper aims to cluster them into one cluster for each digit by incorporating k-means with differing amounts of self-supervision. Firstly, we will flatten the images into 784 length vectors, and set distances to be calculated with Euclidian norm.

K-means clustering with Lloyd's algorithm includes iteratively partitioning into different clusters, calculating a new cluster center, and then repeating the procedure. In this paper we run the algorithm until an iteration changes the $L1$ distance between the new and the previous cluster centers by less than 1×10^{-4}. Before calculating accuracy, the clusters will be mapped to the digits by the Hungarian algorithm [2]. Running K-means once without any supervision resulted in cluster centers showed in figure 1.

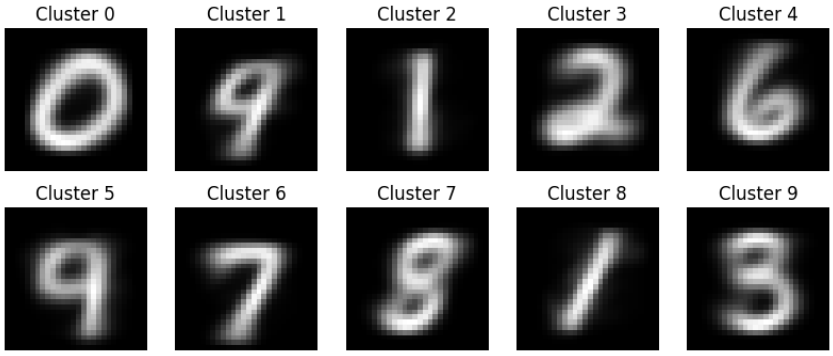

Figure 1: Cluster centers for K=10 ran on 70000 unlabeled MNIST images, each cluster center was initialized on a randomly sampled MNIST image.

The cluster centers above look like digits and it reaches an accuracy of 0.580. In practice, you would often initialize this algorithm several times, run the algorithm, and pick the best clustering. We are instead going to stick with the one-shot clustering, but start to incorporate some labeled data into the procedure.

Having one labeled image for each digit means we can initialize each cluster on each labeled datapoint, meaning that we get a decent initialization every time. We also stick the labeled point in the correct cluster, hoping the cluster will be assigned a lot of images of the same digit, and then stray towards the mean of all the images of this digit during the iteration.

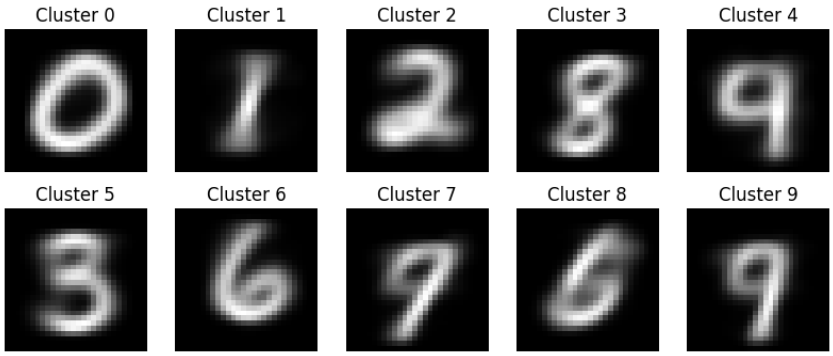

Figure 2: Clusters were initialized on one labeled datapoint per digit. The remaining 69990 images were then clustered while the labeled datapoints were never reassigned.

In figure 2 we can see that 0,1,2,6 and 9 have not strayed too far from the assigned digit, which is not the case for the rest of the clusters. It even looks like 3, 8 and 5 have switched places. Still, this achieves 0.584 accuracy, which is just slightly better than the unsupervised clustering.

Having more labeled points allows us to improve even further. More labeled digits allow us to use the mean of all the associated labeled datapoints to initialize. With 12 digits of each label we get the following cluster centers.

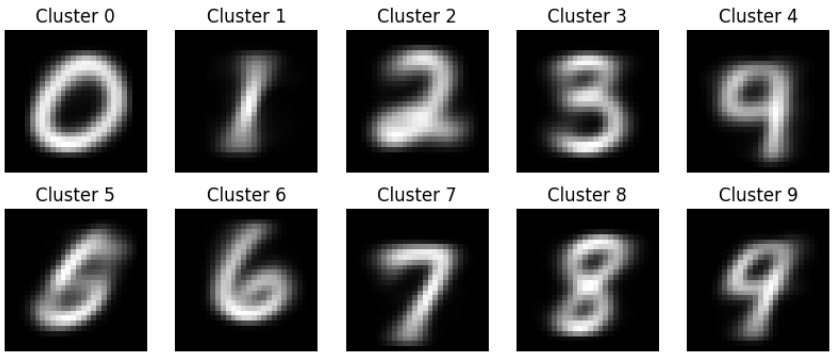

Figure 3: Clusters were initialized as the mean of 12 labeled datapoints per digit. The remaining 69880 images were then clustered while the labeled datapoints were never reassigned.

The clusters from figure 3 achieve 0.626 accuracy, and the cluster centers now look like the associated digits, even though the 5 is a bit blurry while 9 and 4 are too alike. Trying the same with an even higher proportion of labeled datapoints gives us better accuracy, which is shown in figure 4. However, doing K-means clustering for

a dataset with this many labels might be redundant. Instead, having a few labels and focusing on a better distance metric, running the algorithm multiple times and weighting the labeled images to be more prominent when calculating the mean might be the way to go.

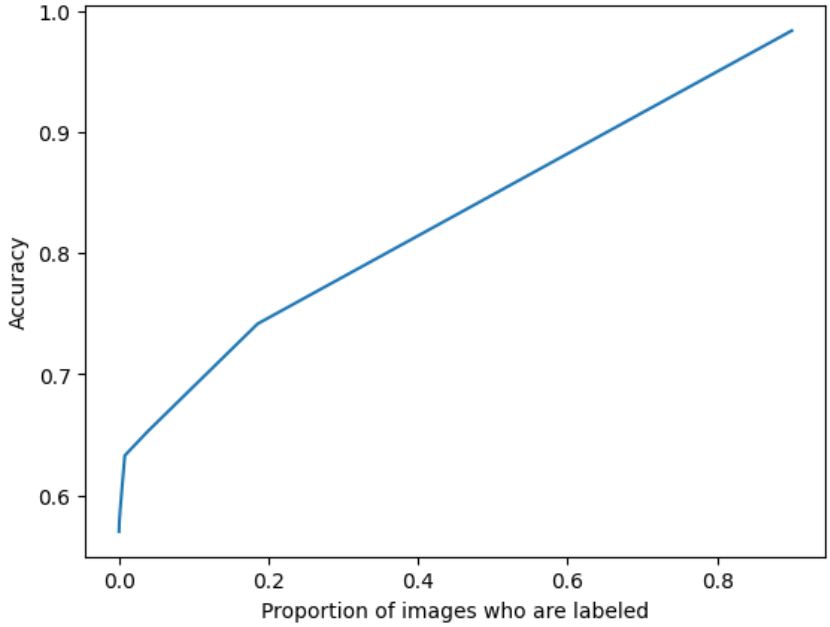

Figure 4: Accuracy over the proportion of images that were already labeled for K-means clustering in a similar way to how it was done in figure 3.

2 A greedy K-medoids clustering algorithm

An algorithm that iteratively adds the medoids that reduces the set distance $d(x_n, S) = \min_{\mu_k \in S} d(x_n, \mu_k)$ as a cluster center until K clusters have been formed is described in pseudocode below:

```
function greedy_k_medoids(X, K, d):
    N = length of dataset X

    # Precompute all pairwise distances and store in an adjacency matrix
    adjacency_matrix = NxN matrix of zeros
    for i = 1 to N:
        for j = i+1 to N:
            distance = d(X[i], X[j])
            adjacency_matrix[i][j] = distance
            adjacency_matrix[j][i] = distance
```

```
# Initialize variables
S = set  # Set of medoids
min_distances = vector with length N containing infinite numbers

for k = 1 to K:
    best_point = None
    best_set_distance = infinite

    for i = 1 to N:  # Evaluate all candidate medoids
        if i in S then:
            skip this i

        # Compute the new set distance incrementally
        new_set_distance = 0
        for n = 1 to N:
            # Find the new minimum distance for point n
            current_dist = min_distances[n]
            new_dist = adjacency_matrix[n][i]
            new_set_distance += min(current_dist, new_dist)

            # Stop if new_set_distance exceeds best_set_distance
            if new_set_distance >= best_set_distance then:
                quit this inner loop and move on to the next candidate i

        # Update the best candidate
        if new_set_distance < best_set_distance then:
            best_set_distance = new_set_distance
            best_point = i

    Add best_point to S

    # Update the minimum distances for all points
    for n = 1 to N:
        min_distances[n] = min(min_distances[n],
                               adjacency_matrix[n][best_point])

return S
```

Precomputing an adjacency matrix allows us to only calculate the distances once, storing the distance of every point to its closest medoids allows us to look up fewer distances, and stopping once the set distance exceeds the best set distance allows

us to disqualify bad candidate points quicker. However, as is apparent by the three nestled for loops, the time complexity is still $O(kN^2)$.

References

[1] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[2] Wikipedia contributors. Hungarian algorithm — Wikipedia, the free encyclopedia, 2024. [Online; accessed 26-November-2024].