From First Principles: K-Means Clustering on Handwritten Digits

Trevor Joncich, Christian Nielsen

Overview

This project explores the k-means clustering algorithm by implementing it from the ground up and applying it to both a simple two-dimensional dataset and the MNIST dataset of handwritten digits. We focus on comparing two initialization strategies—random and k-means++—and show how initialization choices affect clustering performance. Along the way, we use coherence as a metric to evaluate clustering quality and apply the elbow method to estimate the optimal number of clusters.

Our results demonstrate that k-means++ initialization provides more stable and accurate clustering than random initialization. When applied to handwritten digits, the algorithm achieved modest accuracy of 52.5% but also highlighted the limitations of k-means when dealing with high-dimensional, highly variable data.

Initialization Methods

A crucial part of the k-means algorithm is choosing where to place the initial centroids. Since k-means iteratively refines clusters based on these starting points, the quality of the initialization strongly affects the final result. In this project, we explore two common initialization strategies: **k++** and **random initialization**. We then compare how these approaches influence clustering performance.

K++ Initialization

The k++ algorithm improves upon naive random initialization by spreading out the initial centroids. The idea is to reduce the chance that multiple centroids start in the same region of the dataset. The algorithm works in four main steps:

1. Randomly sample one point from the dataset χ to be the first centroid $\vec{c}_1$.
2. Find the point in the dataset farthest from $\vec{c}_1$ and set it as $\vec{c}_2$.
3. For each point $\vec{x} \in \chi$, compute its distance to the nearest chosen centroid. The point with the maximum such distance becomes the next centroid $\vec{c}_l$.
4. Repeat until all k centroids have been chosen.

Step 1: Starting Vector

We first define the dataset, cluster set, and index set. Let n be the number of points and m the number of features in each point:

$$\chi = \{\vec{x}_i \in \mathbb{R}^m \,|\, i = 1, \ldots, n\}, \quad \chi_l = \{\vec{x}_j \,|\, j \in I_l\}, \quad I_l \subset \{1, \ldots, n\}.$$

Choosing one point uniformly at random gives us the first centroid:

$$\vec{c}_1 = \chi_S, \quad S \sim U(I).$$

Step 2: Remaining Vectors

Next, we find points that are far from the existing centroids. We measure squared Euclidean distance:

$$d^2(\vec{x}_i, \vec{c}_l) = \|\vec{x}_i - \vec{c}_l\|^2 = \sum_{j=1}^{m} (\vec{x}_{i,j} - \vec{c}_{l,j})^2.$$

At each iteration, we identify the point $\vec{x}_i$ with the maximum minimum distance to the current centroid set C and make it the next centroid:

$$\vec{c}_{l+1} = \arg\max_{\vec{x}_i \in \chi} \min_{\vec{c}_l \in C} d^2(\vec{x}_i, \vec{c}_l).$$

This ensures that new centroids are as far apart as possible.

Step 3: Repeat

Steps 1 and 2 are repeated until k centroids have been chosen. The result is a set of initial centroids that are more evenly distributed across the data compared to random placement.

Random Initialization

In contrast, random initialization chooses each centroid by sampling coordinates independently from a uniform distribution (here between $[-1.2, 1.2]$ for each component). This method is simple but can lead to poor starting positions, such as centroids placed far from any actual data points.

$$C = \{\vec{c}_l = (S_{l1}, \ldots, S_{lm}) \,|\, S \sim U(-1.2, 1.2),\, l = 1, \ldots, k\}\,.$$

Although straightforward, random initialization often produces inconsistent results compared to k++.

Comparison

To evaluate the two initialization methods, we use a metric called **within-cluster coherence**. This measures how tightly clustered the points are around their assigned centroid:

$$q_l = \sum_{j \in I_l} \|\vec{x}_j - \vec{c}_l\|^2, \quad l = 1, \ldots, k.$$

Summing across all clusters gives the **overall coherence**:

$$Q(\Pi, \vec{c}_1, \ldots, \vec{c}_k) = \sum_{l=1}^{k} q_l.$$

Lower coherence means tighter, more effective clusters. In practice, we can compute coherence directly in Python without explicitly sorting into partitions. To illustrate, we applied both initialization methods to a simple two-dimensional dataset, `XData`, and ran 10 trials with $k = 5$. The results (Figure 1) show that k++ consistently produces lower, more stable coherence values than random initialization.

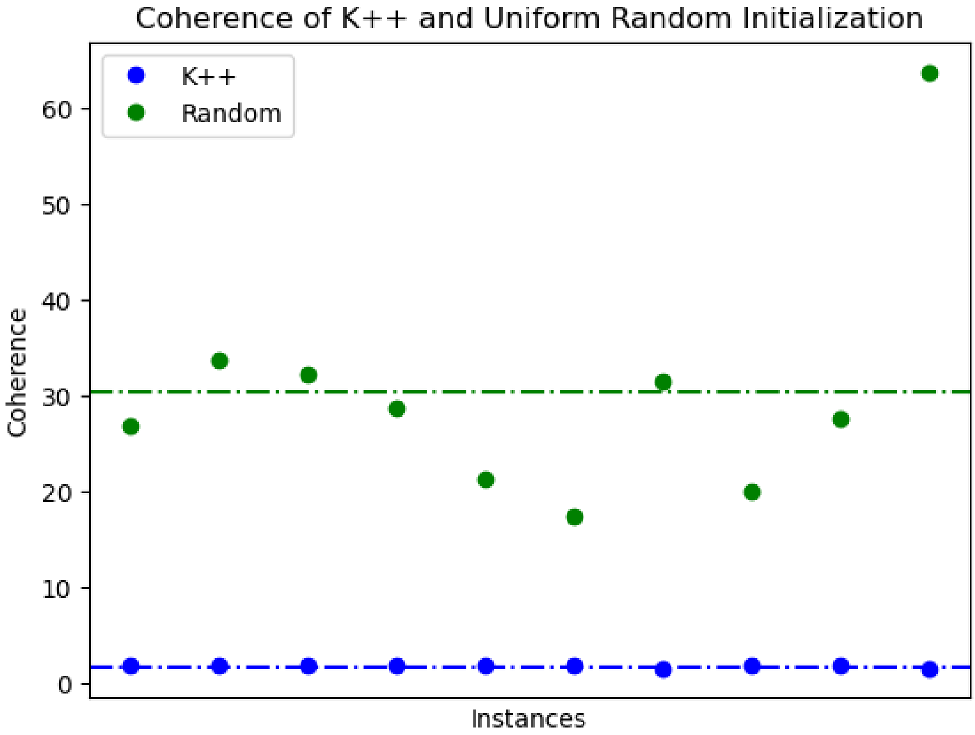


Figure 1: Initialization Coherence: K++ vs. Random

K-MEANS ALGORITHM

Now that we have defined how to initialize centroids, we can describe the k-means algorithm itself. The overall goal of k-means is to assign each point in the dataset to the nearest centroid and then update the centroids to better represent their assigned clusters. This process repeats until the algorithm converges (i.e., no further changes occur). The objective is to minimize the overall coherence defined earlier.

Algorithm

The k-means procedure alternates between two main steps:

1. **Assignment step:** Assign each data point to the cluster of the nearest centroid.
2. **Update step:** Recompute each centroid as the average of the points assigned to it.

This process iteratively reduces the within-cluster coherence until the centroids stabilize.

1) Assignment Step

We assign each data point $\vec{x}_i$ to the closest centroid $\vec{c}_l$. Formally, let τ denote the current iteration. The new index sets $I_l^{(\tau+1)}$ are:

$$I_l^{(\tau+1)} = \left\{ i \,\middle|\, d^2(\vec{x}_i, \vec{c}_l) \le d^2(\vec{x}_i, \vec{c}_p),\, \forall p \ne l,\, \vec{x}_i \in \mathcal{X} \right\}, \quad l = 1, \dots, k.$$

Equivalently, in terms of the data vectors:

$$\mathcal{X}_l^{(\tau+1)} = \left\{ \vec{x}_i \in \mathcal{X}^{(\tau)} \,\middle|\, \operatorname*{arg\,min}_{\vec{c}_j \in C^{(\tau)}} d^2(\vec{x}_i, \vec{c}_j) = \vec{c}_l^{(\tau)} \right\}, \quad l = 1, \dots, k.$$

In other words, each point is reassigned to whichever centroid is closest.

2) Update Step

Next, we update each centroid to be the mean of all points assigned to its cluster. This ensures that the centroid reflects the "center of mass" of its cluster:

$$C^{(\tau+1)} = \left\{ \vec{c}_l^{(\tau)} = \frac{1}{|\mathcal{X}_l|} \sum_{i \in I_l} \vec{x}_i \,\middle|\, l = 1, \dots, k \right\}.$$

Stopping Criterion

The algorithm continues alternating between assignment and update steps until no significant changes occur. Convergence can be detected when centroids stop moving or when the cluster memberships no longer change:

$$C^{(\tau-1)} = C^{(\tau)} \quad \text{or} \quad \mathcal{X}^{(\tau-1)} = \mathcal{X}^{(\tau)}.$$

At this point, the algorithm has reached a stable clustering solution.

Initialization Comparison

With the algorithm in place, we can revisit how the initialization strategy affects the final result. Earlier, we compared random and k++ initialization before running k-means, which gave insight into the "head start" provided by k++. Now, we compare the two strategies after running the full k-means procedure. In both cases, we ran 10 trials with $k = 5$ and measured the coherence at the end. The results are shown in Figure 2.

As shown, k++ initialization consistently leads to better results. While a few random trials achieve coherence close to k++, several others get stuck in poor local minima. This highlights the advantage of k++: by spreading out initial centroids, it gives k-means a strong starting point and reduces the risk of poor clustering outcomes.

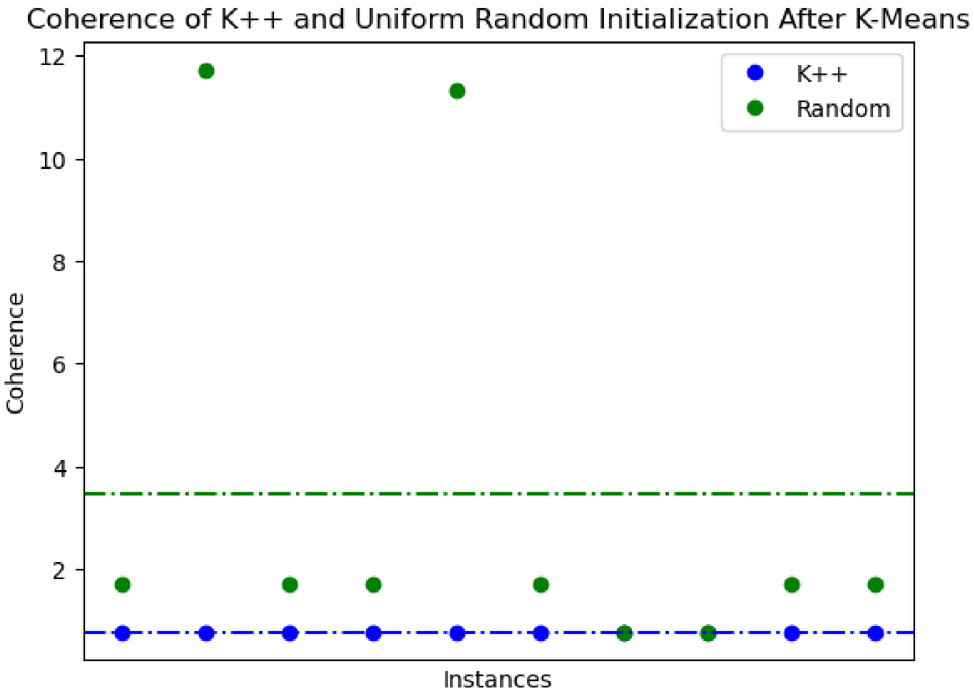


Figure 2: Post-Algorithm Coherence: k++ vs. Random

ELBOW METHOD

One of the challenges in clustering is deciding how many clusters (k) to use. If k is too small, clusters may be forced to merge dissimilar points. If k is too large, clusters may split unnecessarily, increasing complexity without much benefit.

A common heuristic for choosing k is the **elbow method**. The idea is to compute the overall coherence (within-cluster sum of squares) for different values of k and then plot the results. As k increases, coherence always decreases, but the rate of improvement slows down. The ”elbow” of the plot—where the decrease begins to level off—suggests a reasonable choice for k.

In this project, we applied the elbow method to the `XData` dataset using both random and k++ initialization. We considered values of $k = 2, \ldots, 8$ and compared the results.

Random Initialization

Figure 3 shows the elbow method applied with random initialization. Across five realizations, four of them suggest an elbow around $k = 4$. One trial (the orange line) shows abnormal behavior at $k = 4$, likely because a centroid was initialized far from any data points, leaving a cluster empty. Disregarding that case, we conclude that $k = 4$ is a reasonable choice.
However, notice the large variation between realizations. Because random initialization

introduces high variance, more trials would be needed to make confident conclusions.

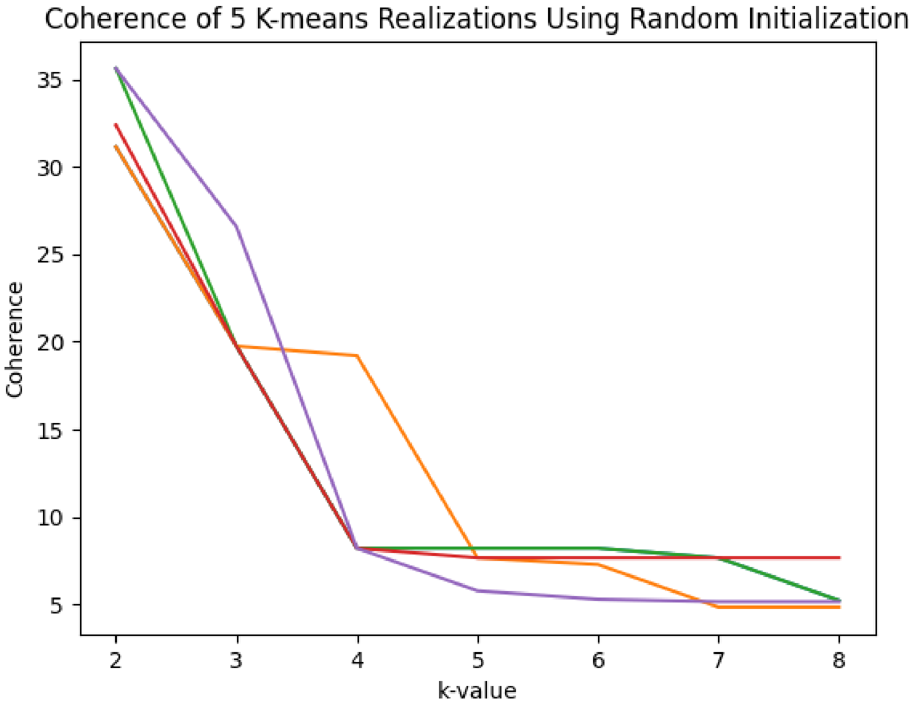


Figure 3: Elbow Method with Random Initialization

K++ Initialization

Figure 4 shows the results with k++ initialization. Unlike the random case, the curves here are much tighter and more consistent. The sharpest elbow appears at $k = 4$, though there is also a smaller bend at $k = 5$. This suggests that the dataset may naturally contain four clusters, but one of them could reasonably be split into two smaller clusters. Because k++ systematically places centroids in well-separated locations, it produces more stable results, making the elbow method easier to interpret.

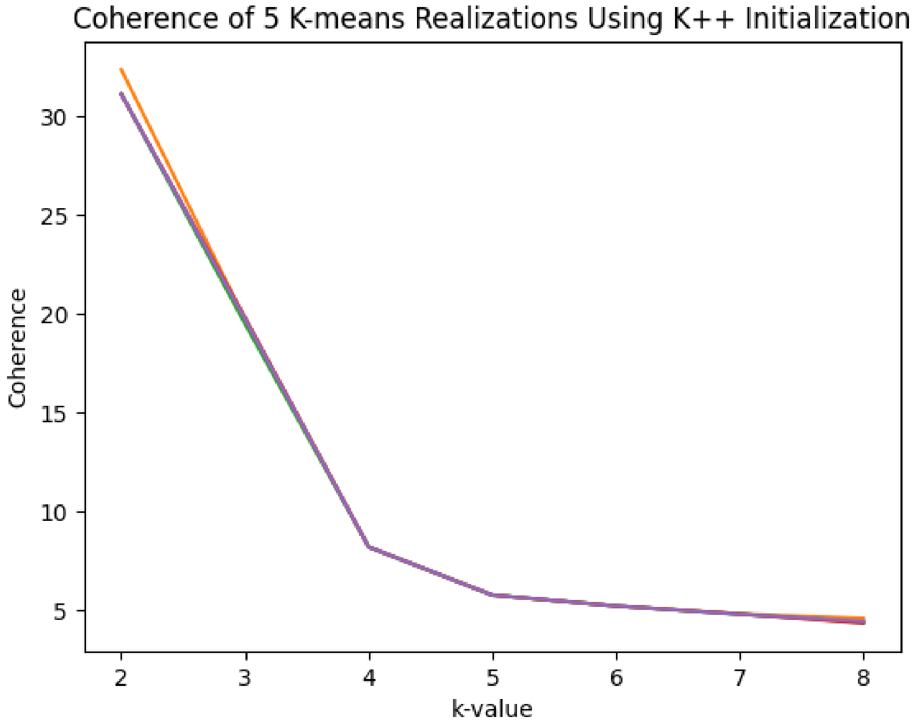


Figure 4: Elbow Method with k++ Initialization

Visual Analysis of XData

To verify our conclusions, we inspected the `XData` dataset visually (Figure 5). There are clearly four main groups, but the top-right cluster could arguably be split into two. This matches the interpretation from the k++ elbow plot, which suggested both $k = 4$ and $k = 5$ as plausible choices. In contrast, the random initialization results were too noisy to reveal this subtle structure.

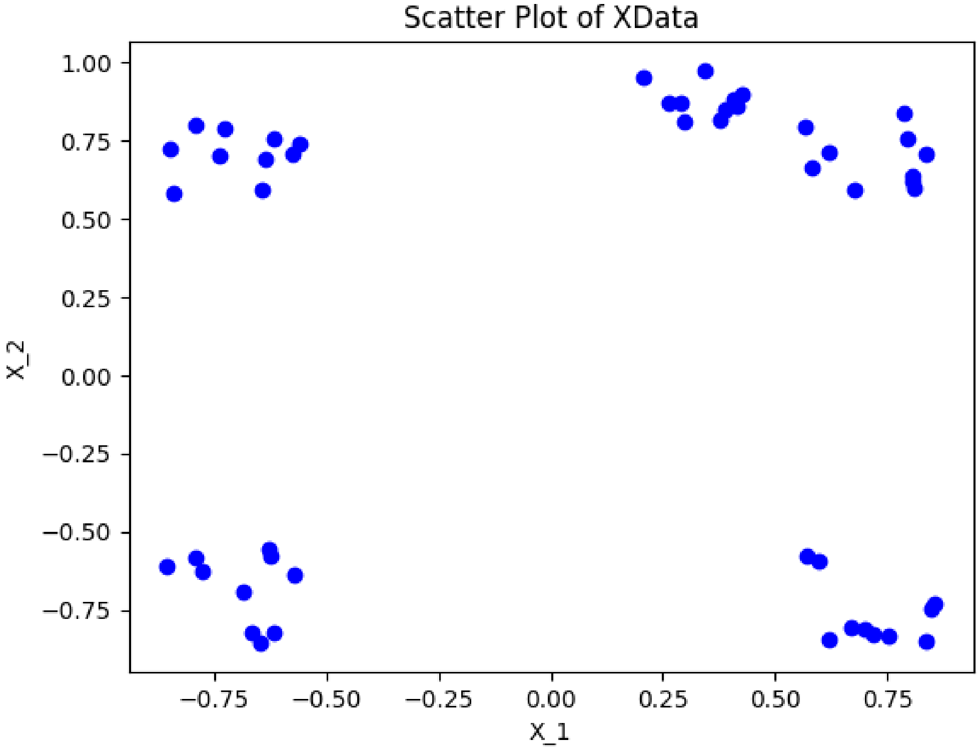


Figure 5: Scatter Plot of XData

MNIST APPLICATION

Finally, we applied our implementation of k-means to the well-known **MNIST dataset**, which contains grayscale images of handwritten digits. This dataset is a classic benchmark in machine learning because it combines high dimensionality (784 features per image) with significant variation in handwriting styles. Applying k-means to MNIST helps illustrate both the strengths and the limitations of the algorithm.

Pre-Processing

Each MNIST image is 28×28 pixels. To use them in k-means, we flatten the image into a 784-dimensional vector. This way, each pixel intensity becomes one feature, and each image can be treated as a single point in high-dimensional space.

Elbow Method

We applied the elbow method to 80 images from MNIST, testing $k = 3, \ldots, 10$. The initialization was set using the 42^{nd} image. The results are shown in Figure 6.

Unlike the earlier `XData` experiment, the elbow here is not clearly defined. The closest thing to an elbow appears at $k = 7$. This reflects the variability in handwritten digits: two people

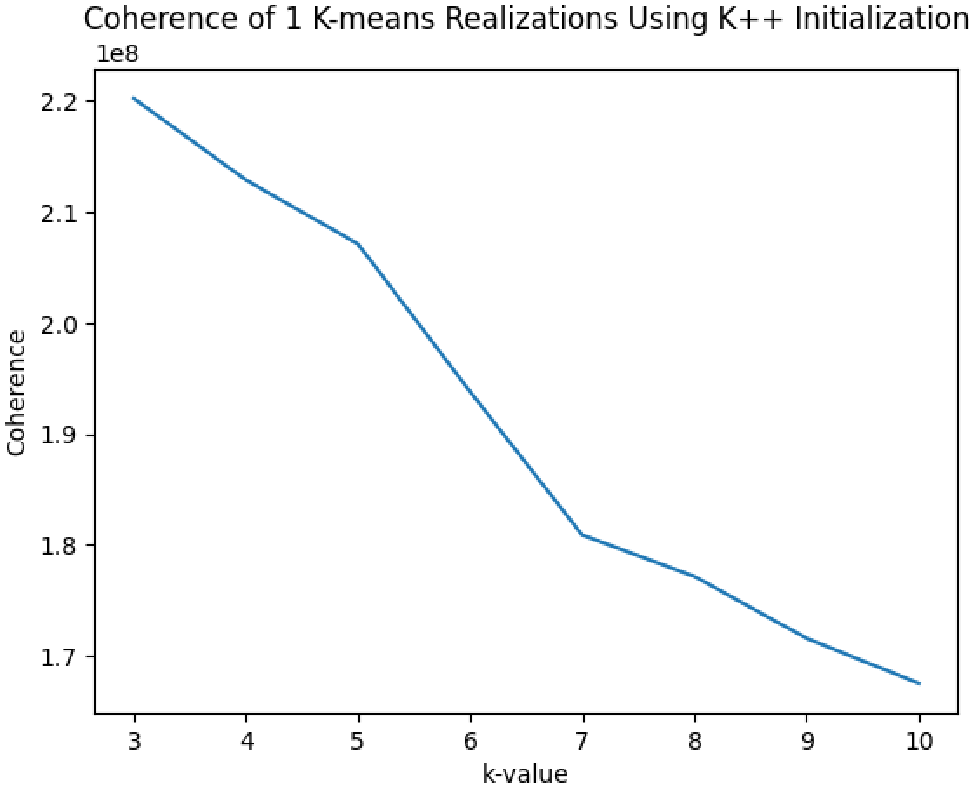
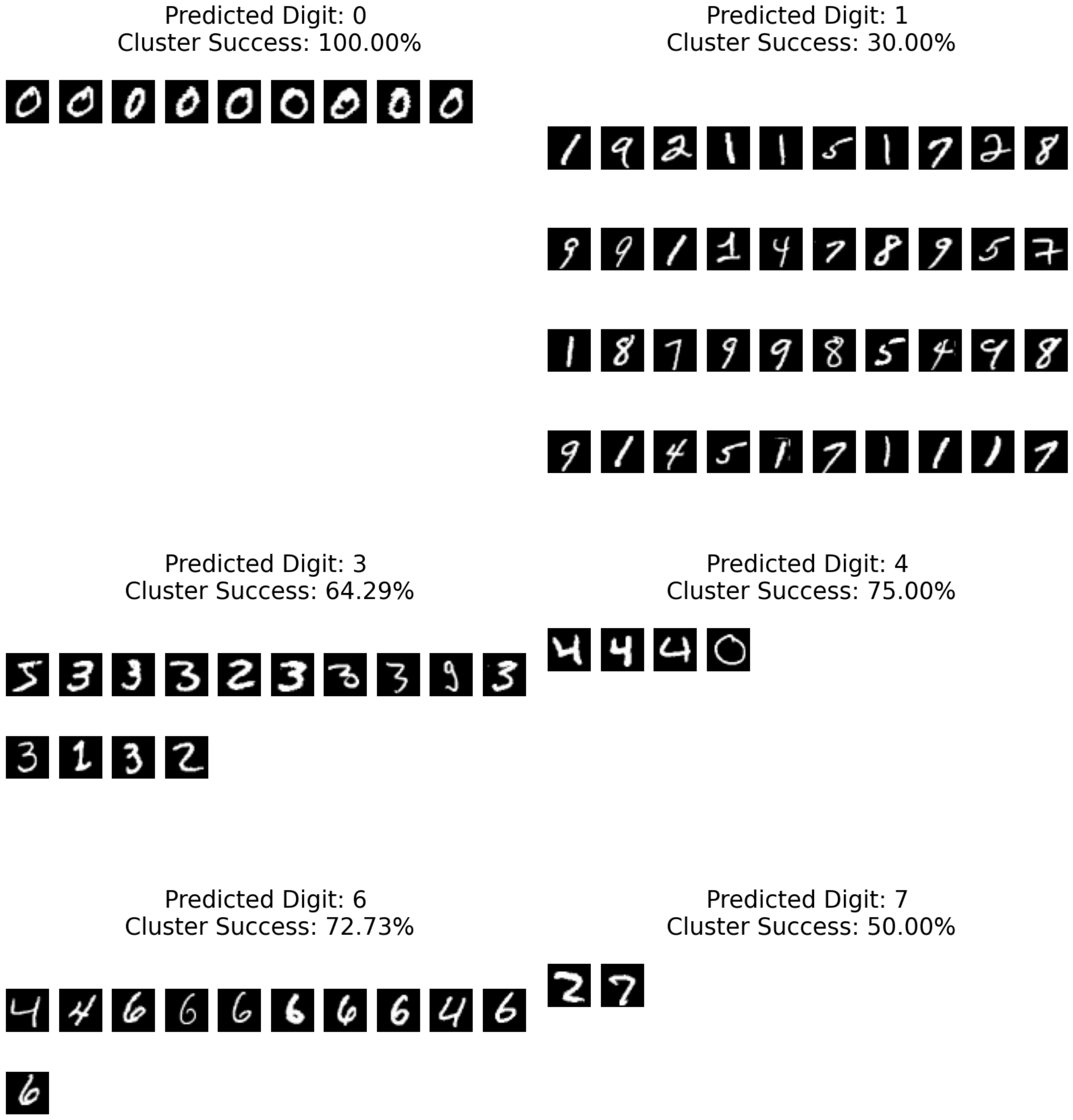


Figure 6: Elbow Method on MNIST Images

may write the same number in very different ways, and those differences cause the algorithm to split digits into multiple clusters. In other words, the overlap between instances of the same digit is weaker than we would hope, which prevents a sharp elbow from forming. For the rest of our analysis, we chose $k = 7$.

Algorithm Results

Running k-means with $k = 7$ produced an overall accuracy of 52.5%. Some digits clustered very well, while others were inconsistent. For example, the clusters for 0 and 4 performed strongly, but cluster 1 acted as a “catch-all” group for digits that did not fit neatly into other clusters. This reflects the challenges posed by handwritten data.



To better understand how clustering worked, we visualized the final centroids by reshaping them back into 28×28 images (Figure 7). These "average digits" show what the algorithm considered typical for each cluster.

For example, in the second cluster, the centroid resembles a blend of 4 and 0. Although visually distinct, these digits share many pixel intensities, so the algorithm grouped them together. This highlights a key limitation of k-means: it considers only pixel-level similarity, not semantic meaning.

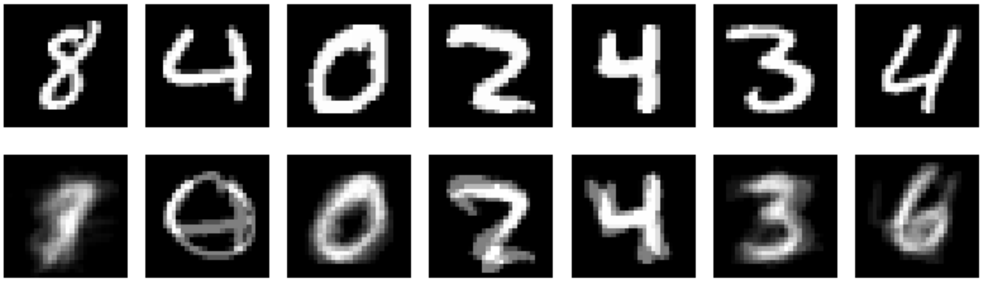


Figure 7: MNIST Initializations vs. Final Centroids

Success Rates

Instead of coherence, we can also evaluate clustering using **success rate**: the percentage of digits correctly grouped with others of the same label. Running k-means for $k = 3, \ldots, 10$ gives the following success rates:

K	Success Rate
3	33.75%
4	35.00%
5	35.00%
6	46.25%
7	52.50%
8	53.75%
9	53.75%
10	53.75%

Interestingly, higher values of k (8–10) slightly outperform $k = 7$. This makes sense: as k increases, coherence decreases and accuracy often improves, since clusters can better capture subtle variations. However, this comes at the cost of interpretability and efficiency.

As k approaches the dataset size n, each point will eventually form its own cluster, pushing the success rate toward 100%. This highlights a tradeoff: the elbow method balances performance with simplicity, making $k = 7$ a defensible choice even though slightly higher k values yield better raw accuracy.

Conclusion

In this project, we implemented the k-means clustering algorithm from the ground up and applied it to both a simple two-dimensional dataset and the more complex MNIST handwritten digits dataset. By comparing random initialization with k++, we showed that k++ consistently produces more stable and effective clustering results, reducing the likelihood of poor local minima. Through the elbow method, we explored how to choose an appropriate number of clusters and demonstrated how initialization quality can affect interpretability.

When extended to MNIST, our implementation highlighted both the strengths and limitations of k-means. While the algorithm achieved reasonable accuracy on some digits, the wide variation in handwriting styles revealed that pixel-based clustering has difficulty capturing semantic structure. This illustrates a broader lesson: even simple algorithms like k-means can provide valuable insights, but they must be applied with care, especially in high-dimensional, noisy domains.

Overall, this project provided hands-on experience with clustering, initialization strategies, and evaluation methods, reinforcing the importance of both mathematical understanding and practical experimentation in data science.