K-means Clustering Algorithm: Overview and Application

Johan Cristobal

Abstract

Given a set of objects, clustering or cluster analysis is the task of identifying subsets (called clusters) such that objects within a cluster are similar and are different from other clusters [1]. The K-means clustering algorithm is one such method. We will introduce the algorithm, highlighting the parts where the user can make choices, and lastly discuss its application in image compression. This report is done within MATLAB and we use two-dimensional numerical data sets (pre-made and randomized). Within the K-means clustering algorithm is a minimization problem, and we will discuss different heuristics that influence this minimization process.

1 Introduction

Clustering, at its core, is to find order in chaos. Its goal is to find partitions in a data set such that each cluster share similarities within its members while being distinguished from other clusters. In 1957, Stuart Lloyd of Bell Labs proposed the (naïve) K-means clustering algorithm. Essentially, it finds K representative members from a data set that will act as the benchmarks. With these benchmarks, given a new data input, it can run through at most K comparisons to decide which cluster it belongs to (and hence what attributes it has).

For this report, we referenced "Pattern Recognition and Machine Learning" by Christopher M. Bishop. We will begin by setting the stage for the problem and discussing the motivation for the name of the algorithm. Then, we introduce the algorithm and its parts, where we delve into each choice made for this project. Lastly, we discuss and showcase an application of K-means algorithm to image compression, alongside some issues we faced.

2 The Problem

For this report, we use MATLAB to program K-means to first run on two-dimensional data. Nevertheless, the code can be translated to higher dimensional data as we'll see in the application. But for the sake of visualization, we chose two-dimensional data. We have also scaled the data values to be within $[0,1] \times [0,1] \subseteq \mathbb{R}^2$.

We begin with a data set that we can cluster with observation. (See the first graph in Figure 1.) Once we confirm that the algorithm runs smoothly on this nice set, we used the **randi** function in MATLAB to create a new set that is completely randomized in its distribution on $[0,1] \times [0,1]$.

Suppose $X = \{x_1, ..., x_N\} \subseteq [0,1] \times [0,1]$ is the data set and $\{c_1, ..., c_K\}$ is the cluster centers. The objective function that we are minimizing is

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{n,k} \cdot d(x_n, c_k) \quad \text{where} \quad r_{n,k} = \begin{cases} 1 & \text{if } k = \min\{d(x_n, c_j) : 1 \leq j \leq K\} \\ 0 & \text{otherwise} \end{cases},$$

K is the number of clusters, and $d : [0,1] \times [0,1] \to [0, \infty)$ is a norm function. The algorithm proposed by Lloyd uses the squared Euclidean (L^2) norm. Here, the coefficient $r_{n,k}$ acts as the check to only include the distance between members of the same cluster.

In Pattern Recognition and Machine Learning, Bishop used $d(x,y) = \|x - y\|^2$ which allows us to compute the optimal cluster centers, assuming $r_{n,k}$ are fixed.

$$J_2 = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{n,k} \cdot \|x_n - c_k\|^2$$

Note that J_2 is convex quadratic in c_k for each $k = 1, ..., K$. Further, the c_k are independent of each other. Hence, we can set the gradient of the k-th coordinate to zero and finds its minimizer:

$$\nabla_k[J_2] = \frac{\partial J_2}{\partial c_k} = -2 \sum_{n=1}^{N} r_{n,k}(x_n - c_k) = 0 \ .$$

Thus, $c_k = \sum_{n=1}^{N} r_{n,k} x_n / \sum_{n=1}^{N} r_{n,k}$ is the mean of all the data points x_n assigned to cluster k.

3 The Algorithm

Let X be the set of objects. Let $d : X \times X \to [0, \infty]$ be a norm distance function on X. The general structure of K-means follows:

1. Pick K distinct centers: $\{c_j\}_{j=1}^{K}$.
2. For each $x \in X$, assign Cluster j such that $d(x, c_j) \leq d(x, c_i)$ for all $i = 1, ..., K$.
3. For each cluster j, find its "center" and reassign c_j to be the new center.
4. Check if $\{c_j\}_{j=1}^{K}$ is optimal. If so, terminate. Otherwise, go back to 2.

It is intentional that this general structure is vague and does not specify how each step is performed. This section will discuss each step in some depth and what choices were made to solve the problem scenario proposed in The Problem section.

In the Appendix, we have attached the MATLAB code as it is helpful to see the helper functions. However, the structure of the code is more important and follows:

1. Pick K random distinct centers: $\{c_j\}_{j=1}^{K}$ such that there is 'ample space' between the choices.
2. For each $x \in X$, assign Cluster j such that $\|x - c_j\|_2 \leq \|x - c_i\|_2$ for all $i = 1, ..., K$.
3. For each cluster j, find its statistical average in each dimension and assign c'_j to be the new center.
4. Check if the old centers $\{c_j\}_{j=1}^{K}$ and new centers $\{c'_j\}_{j=1}^{K}$ are 'close enough'. If so, terminate. Otherwise, go back to 2.

The rest of this section is dedicated to discussing each step and what choices were made, alongside considering other options and limitations that could arise.

3.1 Initial Centers

For the naive K-means, the initial centers are somewhat known ahead of time. In our report, we experimented with a randomized way of initializing the K initial centers. The program picks K distinct points from the set and uses that as the cluster centers. However, this resulted in an issue where if the chosen initial centers are too close, they compete for one cluster. Figure 1 demonstrates this challenge.

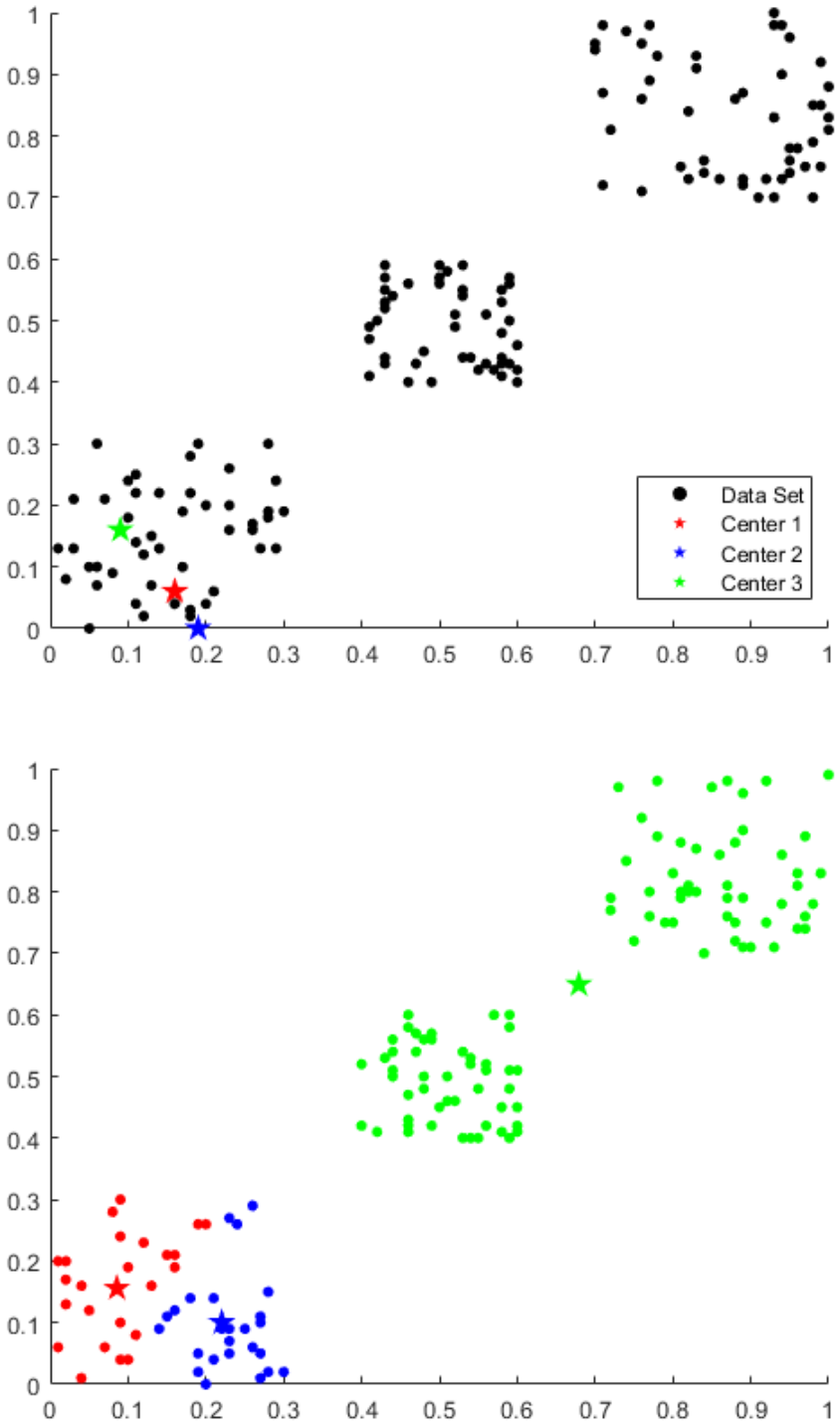

Figure 1. A challenge with using randomized initial centers. Two initial clusters centers were chosen too close to each other and could not escape the competition of one cluster.

We still wanted a randomized way of picking the initial guess. This would allow us to use the algorithm on a set which we have no prior knowledge of its structure. To answer this challenge, we specified that the algorithm randomly chooses cluster centers such that they are a certain Euclidean distance away. For this report, we found (through trial and error) that a distance of 0.1 was enough to prevent competition of one cluster.

3.2 Assigning using Norm Distance Function

As previously stated, the naïve K-means algorithm uses the Euclidean L^2 distance for its norm. This choice is what allows us to interpret that the next iteration should be at the "mean" of each currently built cluster.

However, to save on computation, previous works have also implemented this algorithm with the L^1-norm. The objective function then becomes

$$J_1 = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{n,k} |x_n - c_k| \ ,$$

where the only difference from the original is that we are now minimizing over the absolute value distance rather than the square Euclidean distance. This version is dubbed the K-medians clustering [4]. Previous work by Whelan, Harrel, and Wang have discussed that one benefit that K-medians has over K-means is that it is "incredibly resistant to outliers" as opposed to the mean-value which can skew easily with a single outlier [2].

We tried explore what using the L^p-norm (for $p \geq 3$) would yield. Applying the same stationary point analysis as in K-means, we arrive at root-finding problems for all $k = 1, ..., K$:

$$f(c_k) := \nabla_k [J_p] = \sum_{n=1}^{N} p \cdot r_{n,k} |x_n - c_k|^{p-1} = 0 \ .$$

This proved to be explicitly or analytically difficult to solve in general abstraction with the current tools we had. However, given concrete values and data, we could explore and employ techniques of finding zeros of $f(c_k)$ with iterative methods such as Newton's Method.

Therefore, we end discussion of other norms here and proceed with the industry favorite, the

Euclidean norm for the rest of the report. With that in mind, once we have chosen our initial cluster centers, we assign $x_n \in X$ to the cluster which has the least Euclidean distance from it. This is a discrete optimization problem as there are only K Euclidean distances to compare.

For our project, we added an additional row to our data to indicate the cluster it belongs to. We recognize that this is a memory intensive way. At the end of the algorithm, we will only retain the locations of the center.

3.3 New Centers

Now that we have assigned each $x_n \in X$ to some cluster k, we want to update where the new centers are. The naïve algorithm is allowed to choose a center such that it is a point outside of our set X, since the statistical average will rarely coincide with an actual data point.

However, we also explored the limitation: the center of each cluster must be an element of the set X itself. In order to do accomplish this challenge, we still used the usual algorithm, allowing it to choose the center at some point. Then, we add an additional step before reassigning elements to its new closest center: we find the closest cluster member from that new center to act as the de facto center. We will call this added limitation the "restricted" formulation, while the original is "unrestricted". We summarize our findings in Table 1.

N	k	Unrestricted	$\lvert X_k \rvert$	Restricted	$\lvert X'_k \rvert$	L^2 Distance
500	1	$(0.7539, 0.7361)$	139	$(0.76, 0.74)$	141	0.0072
	2	$(0.2266, 0.2726)$	125	$(0.22, 0.27)$	124	0.0070
	3	$(0.7407, 0.2507)$	105	$(0.73, 0.23)$	105	0.0232
	4	$(0.2544, 0.7448)$	131	$(0.27, 0.75)$	130	0.0164
80	1	$(0.7968, 0.1868)$	18	$(0.79, 0.27)$	18	0.0834
	2	$(0.0575, 0.1913)$	7	$(0.07, 0.20)$	7	0.0153
	3	$(0.3222, 0.1822)$	8	$(0.29, 0.18)$	8	0.0323
	4	$(0.4123, 0.7306)$	47	$(0.47, 0.66)$	47	0.0912

Table 1. Comparing the optimal cluster position with the restricted cluster position with 4 clusters and tolerance 0.0001. Here, X_k and X'_k are the data points in a cluster k, unrestricted and restricted formulation, respectively.

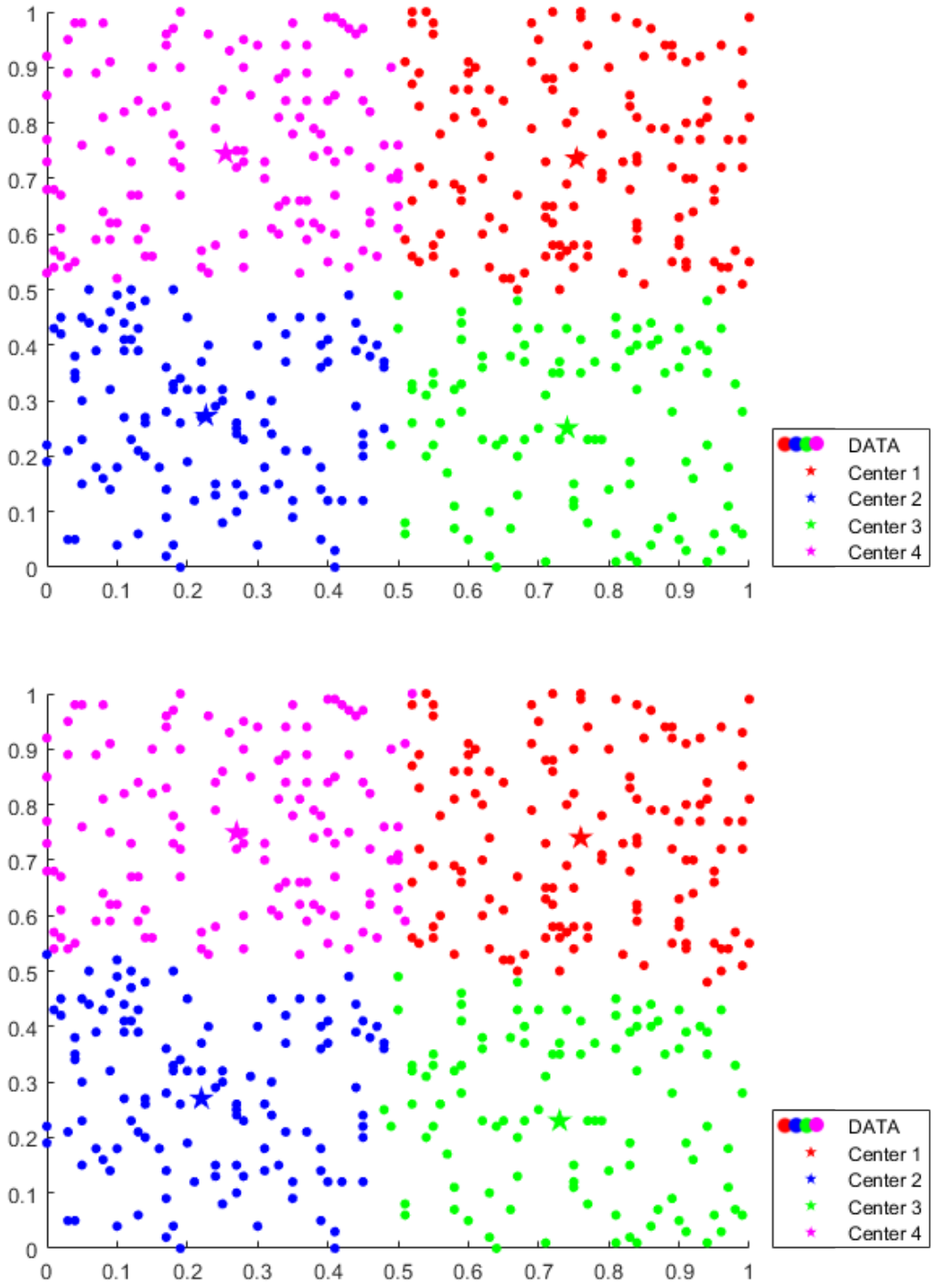

Figure 2. Top: The algorithm is allowed to choose the center to be any point in $\mathbb{R}^2$. Bottom: The algorithm is restricted to choose the center as a point within the data set.

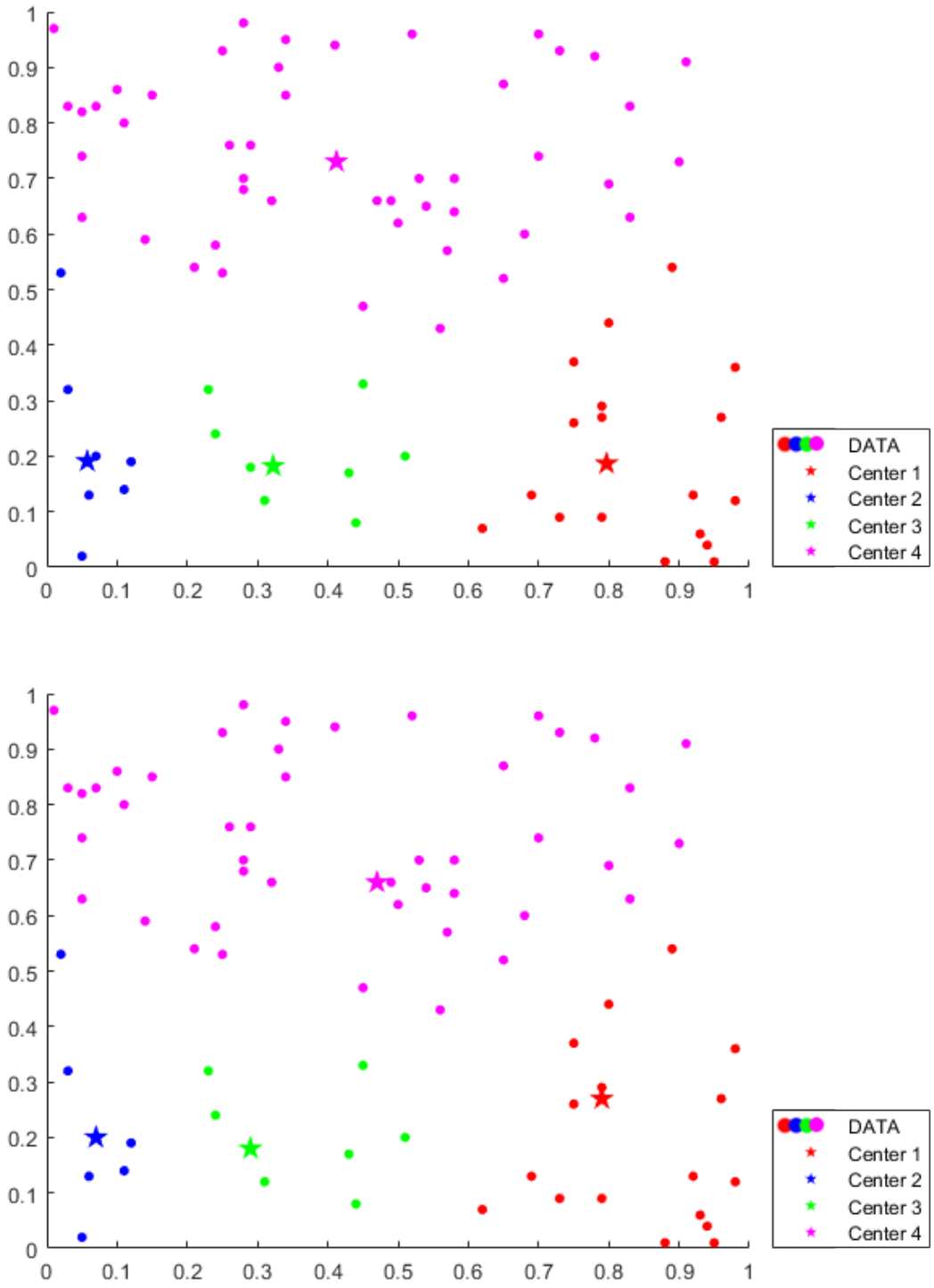

Figure 3. Top: The algorithm is allowed to choose the center to be any point in $\mathbb{R}^2$. Bottom: The algorithm is restricted to choose the center as a point within the data set.

For a larger data set, we find that there is very little difference in the positions of the center between the unrestricted and the restricted formulations. (See Figure 2, page 7.) There were 5 out of 500 elements that changed their cluster assignment between the two formulations.

However, for a smaller data set, we find that there is a larger difference in the positions of the centers. (See Figure 3, page 8.) This change stems from the nature of the denseness of the data set. The larger data set with 500 elements has a higher density in the region than the more spaced out data set with only 80. Interestingly though, there was no change in the cluster assignments between the two formulations for this set. This may be happenstance by this randomized set.

Therefore, we conclude that the nature of the problem and computation limitations will determine which formulation is best, because there would be only be a small percentage of change between the two assignments.

3.4 Optimality & Complexity

By the motivation and derivation, we know that using the statistical mean of each cluster is the most optimal choice for that iteration's cluster choice.

However, the question of optimality in this problem is whether no other configuration of clusters will yield an even smaller value for the objective function. We answered this question by implementing the following check:

> If all the Euclidean distance between the previous iteration and the current iteration are less than some tolerance, then we can terminate the algorithm. Otherwise, run the algorithm until some specified maximum number of iterations.

We run this optimality check on a completely randomized set and we summarize our findings in Table 2. We find that it takes more iterations the less number of clusters there are. As expected, the tolerance does greatly add to the number of iterations for convergence.

It is known that the general naïve algorithm is $O(NKdi)$ where N is the number of d-dimensional objects to cluster in K groups while i is the number of iterations until convergence [3]. Therefore, it is reasonable to assume that it can terminate in linear time. As we saw in Table 2, it does not take long for our program to terminate.

N	K	Tolerance	Iterations
250	3	0.001	11
250	4	0.001	8
500	3	0.0001	11
500	4	0.0001	5
1000	2	0.001	15
1000	2	0.0001	24
1000	3	0.001	23
1000	3	0.0001	15

Table 2. The number of iterations it took given the size of data, the number of clusters, and some error tolerance. These all use the unrestricted version of the algorithm.

4 Application in Image Compression

One of the biggest uses of K-means is in image compression, where K serves as the number of colors we are able to use for the image. The code of this application is in the Appendix. But, the general structure of the *image compression* algorithm follows:

1. Read the image using **imread** function.
2. Let $S = \{((x, y), (r, g, b), k)\}$ be the data set, where (x, y) is the pixel position, (r, g, b) is the RGB value at that pixel, and k is the undecided cluster group.
3. Decide the number of colors, K, and what colors each are.
4. Run K-means algorithm on the (r, g, b) portion of the data.
5. Change the (r, g, b) portion of each point in the data so there are only K colors.
6. Print out the edited pixel value as an image output using **imwrite**.

Due to the data-space focusing on (r, g, b) and not a 2-dimensional element like before, we had to make a 3-dimensional version of K-means algorithm, mimicking the structure of the original code attached in the Appendix.

We could not flesh out the intricacies of the algorithm such as the initial guess clusters could potentially be in the same region and compete for control of that section. We can see this have immediate consequence as seen in Figure 4. Further, we faced a new issue that was not present in the smaller two-dimensional case from before: the complete elimination of a cluster's appearance in the final data.

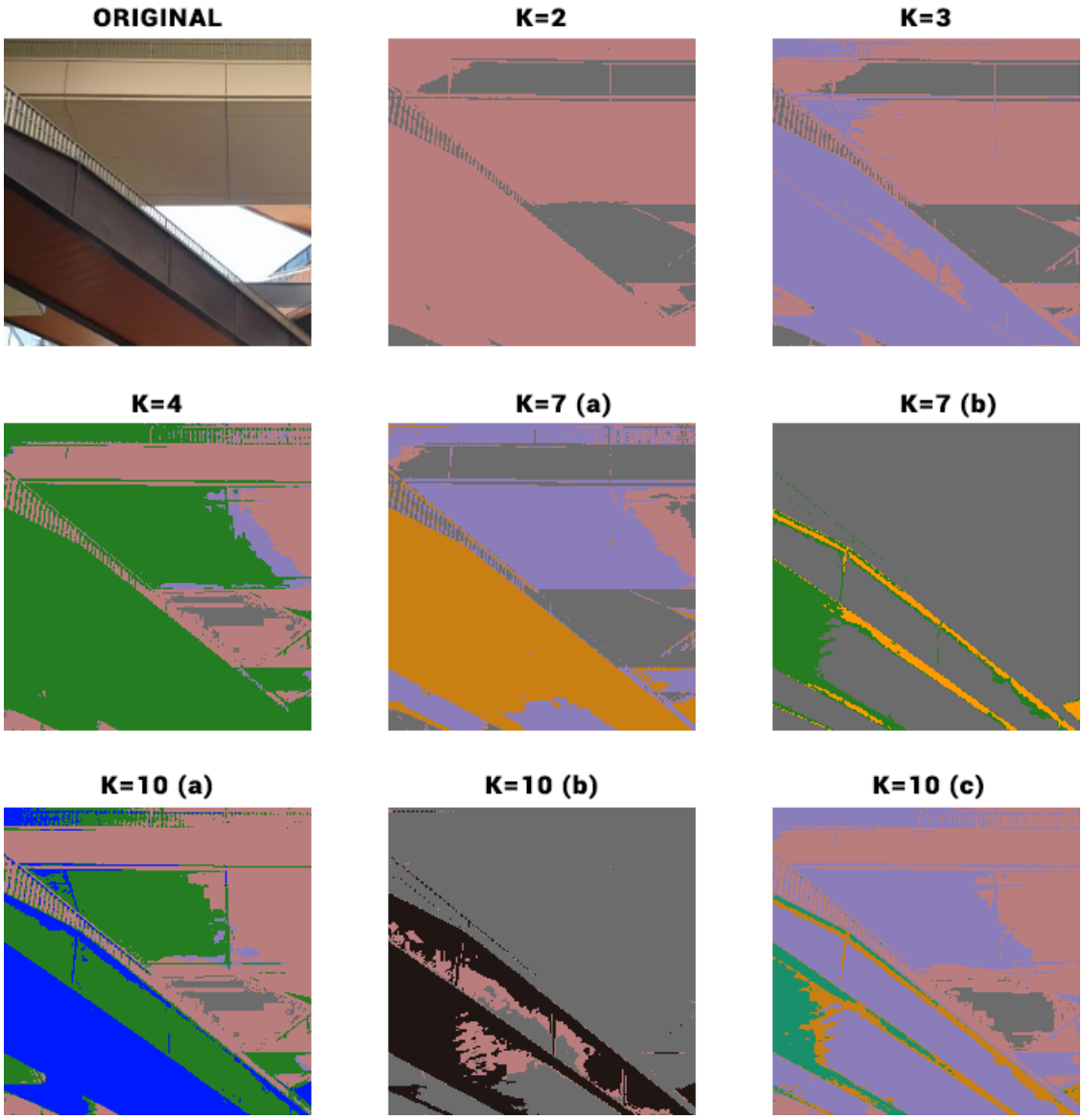

Figure 4. K-means algorithm used with image compression.
Here, K serves as the number of colors for the image.

We run the image compression algorithm on the 200-by200 pixel image and obtain the results in Figure 7. For $K = 2$, we can see that it chose to color the lighter areas in gray while the darker areas are in pink. For $K = 3$, we can see that it is able to differentiate different levels of the darker areas from $K = 2$.

We see for $K = 7$ and $K = 10$, we can have drastically different results. Note that this was an issue even with lower values of K, we only included $K = 7$ and $K = 10$. We believe that the primary cause of this issue is the lack of spacing between the initial cluster center positions. As was in Figure 1 (page 4), we get a clustering where colors compete for one region while the rest of the colors enjoy coloring multiple regions. We can see this happen in $K = 7$(b), since there can only be 3 colors seen and what was discerned by $K = 7$(a) is now just a blanket of gray. The same thing can be seen between the three results obtained with $K = 10$. It should also be noted that even though we are able to assign 7 or 10 colors, we rarely see more than 5 in the rendered image.

After running through the end result data, we can confirm that some colors are present, but in such small numbers that they cannot be seen. But, it also appears that some colors are lost or eclipsed by other colors. Therefore, we added a new check that the initial cluster centers cannot be reassigned to a different cluster. The updated results can be seen in Figure 5, where we can see more colors are present in the compression.

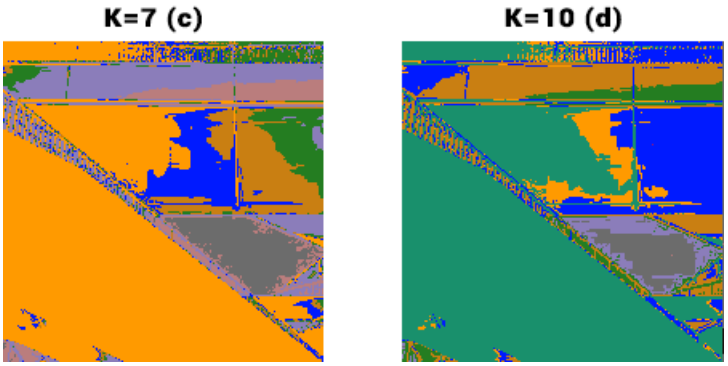

Figure 5. K-means algorithm used with image compression. Here, we add the condition that we always have at least one pixel for every cluster at every stage.

However, we do still run into the issue that multiple cluster centers begin close to each other. It is not in this documentation, but some trials with the algorithm would sometimes yield a blanket of just one or two colors. Due to the immense increase from a data set in $[0, 1] \times [0, 1] \subseteq \mathbb{R}^2$ to a larger set in $[0, 255] \times [0, 255] \times [0, 255] \subseteq \mathbb{N}_0^3$, it is not as clear as to how to discern what is a "good

enough" buffer zone between the chosen initial cluster centers. Further research will be needed to determine what is the best way to assign these initial clusters.

Another question worth noting is what tolerance level is appropriate for this more massive space. We noticed that the image compression algorithm never converged earlier than the specified maximum iterations. Further, in some instances, if the max iteration is high, then there are more chances that one or two color dominate the compression.

5 Conclusion

We saw the motivation and reasoning as to why the statistical average, the mean, of a cluster is the optimal choice for the new center. Hence why the algorithm is called K-means clustering. While the behavior of the algorithm was easily seen and vetted in the small two-dimensional world, we encounter more questions when we try to jump to a larger scope.

As we explored, it would be interesting to find better ways to initialize the cluster centers in the higher dimensions. Further, while the Euclidean distance is the standard for this algorithm, it would be an endeavor to examine the behavior of other norms, not only the L^p-norms. As stated, we only need to store the K cluster centers at the termination of the algorithm, and we can recreate the end result with ease another time. Therefore, this is an extremely memory-efficient algorithm, after the fact. Further, given a new point into the set, it can be easily categorized using just K comparison operations.

The question of optimality and what it means for K-means to converge are also left unanswered in this report, and as such, becomes another thread to follow. It would shed light as to why within the image compression application that a larger maximum iteration resulted in colors dominating other colors, when it would otherwise not if the iterations ended.

All in all, the power of K-means clustering in addressing the issue of cluster analysis should not be understated. It is still used in image compression, image segmentation, and computer vision [5].

Appendix: Codes

K-Means Main

```
K = 3; % the number of clusters
size = 1000; % number of data points
DATA = [randi([0 100],2,size)/100;ones(1,size)];
MAXITER = 100; tolerance = 0.0001;

clusterCenter = initializeCluster(K,size,DATA); % Initialize first cluster
tempCenter = clusterCenter;

for iter=1:MAXITER

    DATA = assignCluster(size,K,tempCenter,DATA); % Assign Closest
    tempCenter = updateCluster(size,K,tempCenter,DATA); % Update Center
    % Check if center is good enough
    if (checkCluster(K,clusterCenter,tempCenter,tolerance) == 1)
        clusterCenter = tempCenter;
        break;
    end
    clusterCenter = tempCenter;
end
DATA = assignCluster(size,K,clusterCenter,DATA); % Final Update
```

initializeCluster | Chooses random points of data to be center

```
function cluster = initializeCluster(K,size,DATA)
cluster = zeros(2,K); tracker = zeros(1,K); i = 1;

while (i < K+1)
   choice = randi([1 size]);
   % Check for repeated entries
   if any(tracker(:) == choice)
       continue;
   end
   % Check if `too' close
   if (i > 1) && (tooClose(DATA(:,choice),cluster,i-1) == 0)
       continue;
   end
   % Add new chosen cluster
   tracker(i) = choice;
   cluster(1,i) = DATA(1,choice); cluster(2,i) = DATA(2,choice); i = i + 1;
end
end
```

assignCluster | Assigns a cluster number to each data

```
function DATA = assignCluster(size,K,clusterCenter,DATA)

for i = 1:size
    memberOF = DATA(3,i);
    benchmark = ((clusterCenter(1,memberOF) - DATA(1,i))^2 + (clusterCenter(2,
        memberOF) - DATA(2,i))^2)^(1/2); % Euclidean distance

    for j = 1:K
        contest = ((clusterCenter(1,j) - DATA(1,i))^2 + (clusterCenter(2,j) - DATA
            (2,i))^2)^(1/2);
        if (contest < benchmark) % If better, change
            DATA(3,i) = j;
            benchmark = contest;
        end
    end
end
end
```

updateCluster | Find the mean of the new cluster

```
function clusterCenter = updateCluster(size,K,clusterCenter,DATA)

for i=1:K
    counter = 0;
    sumX = 0;
    sumY = 0;
    for j=1:size
        if (DATA(3,j) == i)
            counter = counter + 1;
            sumX = sumX + DATA(1,j);
            sumY = sumY + DATA(2,j);
        end
    end
    clusterCenter(1,i) = sumX/counter;
    clusterCenter(2,i) = sumY/counter;
end

% uncomment if you want limitation
% clusterCenter = pickDeFacto(size,K,clusterCenter,DATA);

end
```

pickDeFacto | Find the most center point of the cluster

```
function clusterCenter = pickDeFacto(size,K,clusterCenter,DATA)
for i=1:K
    candidate = 0; leastDist = 500;
    for j=1:size
        if (DATA(3,j) == i) % cluster member
            X = (clusterCenter(1,i) - DATA(1,j))^2;
            Y = (clusterCenter(2,i) - DATA(2,j))^2;
            dist = (X+Y)^(1/2);
            if (dist < leastDist) % if better, change
                candidate = j; leastDist = dist;
            end
        end
    end
    clusterCenter(1,i) = DATA(1,candidate); clusterCenter(2,i) = DATA(2,candidate);
end
end
```

checkCluster | Checks the error tolerance

```
function bul = checkCluster(K,clusterCenter,tempCenter,tolerance)
bul = 1;
for j=1:K
    distanceX = (clusterCenter(1,j) - tempCenter(1,j))^2;
    distanceY = (clusterCenter(2,j) - tempCenter(2,j))^2;
    if ((distanceX + distanceY)^(1/2) > tolerance)
        bul = 0;
        return;
    end
end
end
```

tooClose | Helps separate the randomly chosen starting center

```
function bul = tooClose(candidate,clust,size)
bul = 1;
for j=1:size
    dist = (candidate(1) - clust(1,j))^2 + (candidate(2) - clust(2,j))^2;
    if (dist^(1/2) < 0.1)
        bul = 0; return;
    end
end
end
```

Image Compression | Using a 3D K-means and imread

```
IMG = imread('original.png'); % Here the image was a 200 by 200 pixel image
% imread returns a 200 x 200 x 3 matrix
% Add another row to the 3rd dimension for cluster assignment
% It is also in unsigned integer 8-bit format, we need to type cast.
DATA = cat(3,IMG,ones(200,200,1));

K=10;
MAXITER = 10;
tol = 1;
DATA = Kmeans3D(DATA,K,200,200,MAXITER,tol);
% Code ommitted, it is essentially the same as in 2D.
% However, now we have to edit the RGB values to match its cluster.

RGB = [ 108,186,139,36,201,0,255,31,144,25;
        108,125,125,125,127,26,154,21,25,144;
        108,125,186,33,18,255,0,19,126,108]; % random colors chosen
RGB = cast(RGB,uint8);

counter = zeros(1,K); % counts the number of pixels for each cluster
for i=1:200
    for j=1:200
        memberOf = DATA(i,j,4);
        counter(memberOf) = counter(memberOf) + 1;
        IMG(i,j,1) = RGB(1,memberOf); % R-value
        IMG(i,j,2) = RGB(2,memberOf); % G-value
        IMG(i,j,3) = RGB(3,memberOf); % B-value
    end
end

imwrite(IMG,'result.png'); % Make a new image file
```

References

[1] Christopher M. Bishop. "Pattern Recognition and Machine Learning". In: Springer, 2006. Chap. Mixture Models and EM, pp. 423–435.

[2] Christopher Whelan, Greg Harrell and Jin Wang. "Understanding K-Medians Problem". In: *International Conference Scientific Computing* (2015).

[3] J. A. Hartigan and M. A. Wong. "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* (1979).

[4] A. K. Jain and R. C. Dubes. "Algorithms for Clustering Data". In: Prentice-Hall, 1988.

[5] K. Manglem N. Dhanachandra and Y. J. Chanu. "Image Segentation Using K-means Clustering Algorithm and Subtractive Clustering Algorithm". In: *Procedia Computer Science* (2015).