K-means algorithm

1st Brian Crespo
School of Mathematical and Computational Sciences
Yachay Tech University
San Miguel de Urcuquí, Ecuador
brian.crespo@yachaytech.edu.ec

***Abstract*—Nowadays, Artificial Intelligence and more specifically, Machine Learning (ML) techniques are of great importance due to their many applications in fields as data security, resource optimization, natural language processing (NLP), personalized marketing, fraud detection, recommendation systems, among others. In ML, clustering turns out to be one of the most popular tasks, and k-means algorithm constitutes a famous and widely used algorithms to perform it. The present article addresses k-means algorithm and offers an explanation about its importance, applications and operation, both from a mathematical and computational point of view. Additionally, the application of k-means for image compression and its implementation in the Python 3 programming language are presented. The obtained results shows that the k-means based compression achieved satisfactory results in image size reduction. Furthermore, the mathematical and empirical analysis presented in the article allowed to classify k-means as an Exhaustive Search algorithm, with complexity $\Theta(n)$.**

***Index Terms*—k-means, clustering, machine learning, unsupervised learning, image compression, k-means++**

I. Introduction

Machine Learning is a branch of Artificial Intelligence that can be defined as a set of computational techniques capable of improving performance and precision through experience and the usage of data [1], [2]. However, this learning does not take place in a unique manner, but can be carried out in different ways, depending on the amount of human-supervision in the process. In this sense, Machine Learning can be divided into three main categories that are: *Supervised Learning*, *Unsupervised Learning*, and *Reinforcement Learning*. On one side, in Supervised Learning the learning process uses labeled data, i.e., data that contains the information that need to be predicted. On the contrary, Unsupervised Learning makes use of unlabeled data sets and the model by itself is in charge of inferring patterns from it. Finally, the algorithms in Reinforcement Learning learns by trial and error how it must act to achieve a goal, each action generates a penalty or a reward and the objective is to maximize it [3].

Clustering and dimension reduction tasks represent two of the most common applications of Unsupervised Learning. Clustering, in a basic way, is defined as the task of grouping elements from a data set, according to their similarity [4]. There are several similarity criteria used for clustering and the squared Euclidean distance is one of the most famous. An algorithm that makes use of this similarity criterion is k-means, and its objective is to minimize the sum of the squared Euclidean distances between the elements of a data set and the center of a group. This algorithm is one of the most famous and widely used techniques for clustering [5] and represents the central object of study in this work.

The rest of the article is organized as follows. Section II provides a general explanation about k-means, its importance, advantages and disadvantages, and some of its fields of application. Section III explains how the algorithm works from a mathematical and algorithmic point of view. Important methods for selecting the initial parameters of k-means are explained in section IV. Section V addresses the implementation of the algorithm in the Python 3 programming language. Section VI discusses image compression using k-means, and its implementation. The results of image compression with k-means are detailed in section VII. Section VIII analyzes the complexity of k-means through mathematical and empirical analysis. Finally, conclusions are drawn in section IX.

II. K-Means Algorithm

K-means algorithm can be defined as an iterative, unsupervised and non-deterministic algorithm [4]. The objective of this method is partitions a data set into k disjoint groups or clusters in such a way that the elements of the same cluster present similar characteristics. One of the fundamental characteristics of this algorithm is that it is part of the Unsupervised Learning techniques, and therefore, operates on an unlabeled data set. Furthermore, k-means is not predictive, i.e., different outputs can be obtained from the same input. a feature that provides this method a non-deterministic character.

K-means is one of the most famous and widely used algorithms for clustering tasks mainly due to the simplicity of its implementation and its speed. The method has proved to have high effectiveness and produce good clustering results in many practical applications [4]. Nevertheless, a correct performance of the algorithm depends largely on the value of k chosen, and the initial values of the centroids [6]. If that selection is not suitable, the algorithm could converge to a local and not a global maximum, resulting in different outcomes [7]. In addition to this, k-means is sensitive to outliers, and therefore prior processing is required to identify and remove them, before applying the clustering algorithm.

However, there are methods whose implementation helps to improve the selection of these parameters. In the case of the vakue of k, one of the most common techniques for its selection is knwons as the Jambu's elbow. In the case of the initial centroids selection for each cluster, k-means originally

uses a random selection among the elements of the considered data set, however, k-means ++ represents a famous and useful method which helps to improve this process.

Among the most common applications of k-means can be mentioned image segmentation, document classification, identification of criminal locations, customer segmentation, fraud detection, data mining, image compression, among others [8]. In particular, this article addresses the use of the algorithm for the last one.

III. How K-means works

The calculus of the squared Euclidean distances represents the basis of the operation of the k-means algorithm. Given two vectors $a, b \in \mathbb{R}^n$, its squared Euclidean distance is defined as:

$$\|a - b\|_2^2 = \sum_{i=1}^{n} (a_i - b_i)^2$$

Given a set of n observations or elements $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, with $\mathbf{x} \in \mathbb{R}^d$, and a value $k \leq n \in \mathbb{N}$, representing the number of desired clusters or groups $S = \{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_k\}$, each one with a centroid in $C = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{x}_3, \ldots, \mathbf{c}_k\}$, with $\mathbf{c} \in \mathbb{R}^d$, the objective of k-means is to minimize the sum of the squared euclidean distances between each of the elements of X and its cluster centroid. K-means operates in two basic stages: Assignment Step, and Update Step, and iteratively executes until a specified stopping criterion is met.

A. Assignment Step

In this stage, the assignment of a cluster to each element in X is carried out. Mathematically, this step can be expressed as:

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - c_i^{(t)} \right\|_2^2 \leq \left\| x_p - c_j^{(t)} \right\|_2^2 \forall j \right\}$$

where $1 \leq i, j \leq k$, $i \neq j$, $1 \leq p \leq n$ and $S_i^{(t)}$ represents the ith cluster in an specific iteration t.

In other words, each element of X is assigned to a unique cluster S_i in such a way that the distance between x_p and the centroid c_i is the smallest possible between all the centroids in C.

B. Update Step

In this part of the process, the centroids of each of the clusters are updated depending on the elements of X that were assigned in the previous step. Mathematically, this step can be expressed as follows:

$$\mathbf{c}_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

The new value of the centroid c_i corresponds to sum of all the x_j in the cluster S_i, divided by the its number of elements $|S_i|$. The obtained value is known as the *center of gravity* of S_i, or the $mean$ of its elements. Since k represents the number of clusters required, the algorithm iteratively calculates a total of *k-means*, which gives rise to its name.

C. Stop Criterion

There are three common stopping criteria for the k-means algorithm:

1) The centroids stop changing, and the algorithm has converged. Two centroids can be considered to have stopped changing if the difference between their values in two consecutive iterations is less than a tolerance value.
2) The elements stop changing clusters.
3) The algorithm has reached the maximum number of iterations.

D. K-means pseudocode

Based on the explanation provided in Section III, the pseudocode for k-means algorithm is detailed below.

Algorithm 1: K-means

Input: A dataset of points $X = \{x_1, x_2, x_3, \ldots, x_n\}$
A number of clusters K
Ouput: Centroids $C = \{c_1, c_2, \ldots, c_k\}$, dividing X into k clusters
Labels $L = \{l_1, l_2, \ldots, l_n\}$, denoting the cluster number assigned to each element in X

```
Choose k initial centroids C = {c_1, c_2, ..., c_k};
while stopping criterion has not been met do
    // Assignment Step
    for i ← 1 to n do
        Find closest centroid c_j ∈ C for x_i
        L[i] ⟵ j
    end
    // Update Step
    for i ← 1 to k do
        Set c_i to be the mean of all points in S_i
    end
end
return C, L
```

IV. Selection of k and the initial centroids

The correct functioning of k-means depends on the correct choice of two fundamental parameters:

1) k or numbers of clusters
2) The k initial centroids

A. Selection of the value for k

An optimal choice of k is of utmost importance for a correct operation of k-means. If this value is chosen incorrectly, the grouping of items might not be appropriate. To solve this, several methods are employed, but in this case, the elbow method will be explained.

The elbow method uses the $WCSS$ value as a selection criterion, i.e, the sum of the squares of the distances between the elements of a group and its centroid, or *Within-Cluster Sum of Squares*. Basically, k-means is applied using different values of k and the total $WCSS$ is obtained in each case. Finally, each $WCSS$ value is plotted against its respective k,

and the location of a curve similar to an elbow is considered an optimal choice of k. In the case of Figure 1, $k = 3$ would represent an adequate election.

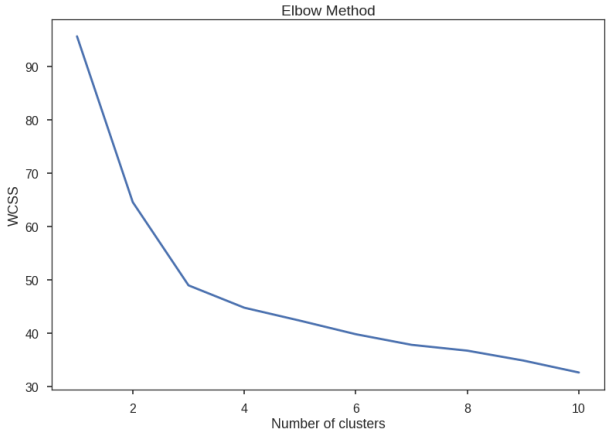


Fig. 1. WCSS vs Number of clusters

B. Selection of the initial centroids

The results obtained with k-means depend largely on the initial centroids. This selection can be done in different ways, however, two of the best known are *random initialization*, and *k-means ++*, originally proposed by Arthur and Vassilvitskii [9].

On the one hand, *random initialization* consists of the use of random elements of the dataset X as initial centroids. However, this selection could not be the most appropriate, and in many cases, could lead to non-optimal results. On the other hand, *k-means ++* seeks to choose centroids from X so that they are as far apart as possible. The pseudocode for *k-means ++* initialization is detailed below.

Algorithm 2: K-means ++ initialization

Input: A dataset of points $X = \{x_1, x_2, x_3, \ldots, x_n\}$
A number of clusters k
Ouput: Initial centroids $C = \{c_1, c_2, \ldots, c_k\}$, dividing X into k clusters
Choose the first centroid $c_1 \in X$ randomly
// Selection of the following k-1 centroids
for $i \leftarrow 2$ **to** k **do**
  for $j \leftarrow 1$ **to** n **do**
    Compute the minimum distance from x_j to a centroid of C
  end
  Select the element x with the largest distance as c_i.
end
return C

V. K-means Implementation

K-means has been implemented in Python 3. This implementation consists of seven functions:

1) `random_initialization`
2) `k_means_pp_initialization`
3) `assignment_step`
4) `update step`
5) `centroids_distance`
6) `label_comparison`
7) `k_means`

The first two functions are responsible for performing the initialization of centroids as explained in IV-B, the third and fourth function do the cluster assignment and centroids update respectively, previously treated in III-A and III-B, functions 5 and 6 correspond to the first two stop criteria addressed in III-C, and the last one perform k-means algorithm execution receiving the necessary input arguments, calling the previous functions, and returning the necessary values, according to how it is shown in the Algorithm 1 pseudocode.

A. Libraries

The library used for k-means implementation is `numpy`. The `numpy` library makes use of use of vectorization which allows to perform array operations much more efficiently than in the native Python lists.

```
import numpy as np
```

B. Centroids Initialization

The `random_initialization` receives as input a 2D array `X`, the dataset, and k, an integer representing the number of desired clusters. This function generates a random value `n` between `0` and `r-1`, with `r` as the number of items or rows in `X`. Then, the nth row of `X` is chosen as the next centroid, and is added to `C`. This process is carried out `k` times. Finally, the 2D array `C` containing `k` items is returned.

```
def random_initialization(X, k):
    # k random rows of X  are taken as initial centroids
    C = np.array([])
    # number of rows of X
    r = X.shape[0]
    for i in range(k):
        n = np.random.randint(r)
        C = np.append(C,[X[n]])
    return C
```

The `k_means_pp_initialization` function performs the choice of the first centroid similar to `random_initialization`. To define the remaining `k-1` centroids, first the `numpy` function `np.linalg.norm` calculates the distance from the jth point in `X` to the centroids in `C`. After that, the index of the smallest calculated distance is obtained with `np.argmin` and this is stored in the 1D array `distances`. When this process has been completed, the next point to be stored in `C` will correspond to the one whose minimum distance to a centroid is the maximum among all the elements `X`. This process takes place `k-1` times.

```
def k_means_pp_initialization(X,k):
    r = X.shape[0]
    # The first centroid is randomly selected
    n = np.random.randint(r)
    C = np.array([X[n]])
    # The following k-1 centroids are selected
    for i in range(1,k):
        distances = np.zeros(X.shape[0])
```

```
        for j in range(X.shape[0]):
            d = np.linalg.norm(X[j]-C, axis = 1)
            index_min = np.argmin(d)
            distances[j] = d[index_min]
        index_max = np.argmax(distances)
        C = np.append(C,[ X[index_max]], axis = 0)
    return C
```

C. Assignment step

The `assignment_step` function receives as input, the dataset `X`, and `C`, the centroids, and returns a 1D array `L` containing the label or centroid number assigned to each element in `X`. For this purpose, the distance between each element of `X` and the centroids is calculated, and the centroid assigned is the one for which the distance minimum.

```
def assignment_step(X,C):
    L = np.zeros(X.shape[0], dtype = int)
    for i in range(X.shape[0]):
        distances=np.linalg.norm(X[i]-C, axis = 1)
        index = np.argmin(distances)
        L[i] = index
    return L
```

D. Update step

The function `update_step` receives as input parameters `X`, `L`, and `C`, and is responsible for updating the value of each centroid to the mean of all the points of `X` that belong to it. This is done with the `np.mean` function. Finally, the updated array `C` is returned.

```
def update_step(X,L,C):
    for i in range(C.shape[0]):
        C[i]=np.mean(X[L==i],axis=0)
    return C
```

E. Stop Criterion 1: No change between centroids of consecutive iterations

The `centroids_distance` function receives as input `old_C` and `C`, the centroids array in two consecutive iterations `t-1` and `t`. The distance between each cetroid `old_C[j]` and `C[j]` is calculated, and the maximum is returned. This one will be later compared with a tolerance value, as explained in III-C.

```
def centroids_distance(old_C, C):
    distances = np.linalg.norm(C - old_C, axis = 1)
    return np.max(distances)
```

F. Stop Criterion 2: Elements stop changing clusters

The `label_comparison` function receives `old_L` and `L`, as input, which correspond to the arrays containing the number of assigned centroids in two consecutive iterations `t-1` and `t`. The function returns `True` if both arrays are equal, i.e. there was no change in the centroid assignment in iteration `t` relative to `t-1`, and `False` in other case.

```
def label_comparison(old_L, L):
    comparison = old_L == L
    return comparison.all()
```

G. K-means function

The `k_means` function receives as parameters `X`, the data set in 2D array format, `k`, the desired number of clusters, `iterations`, the number of iterations and `init`, the centroid initialization method, either `'random'` or `'k-means++'`, and `tol`, the tolerance value for the second stopping criterion. By default, the `iterations`, `init`, and `tol` parameters are `15`, `'random'`, and `1e-3` respectively. `k_means` calls the previously defined functions according to Algorithm 1 pseudocode, and also includes the stop criteria already discussed in III-C. As a last step, the arrays `C` and `L` are returned.

```
def k_means(X,k,iterations = 15,
            init = 'random',tol = 1e-3):
    # Label array Initialization
    L = np.zeros(X.shape[0], dtype = int)
    # Centroid Initialization
    if init == 'random':
        C = random_initialization(X,k)
    elif init == 'k-means++':
        C = k_means_pp_initialization(X,k)
    #---> Stop Criterion 3
    for i in range(iterations):
        # Assignment Step
        previous_labels = np.copy(L)
        L = assignment_step(X,C)
        #---> Stop Criterion 2
        if label_comparison(old_L,L):
            break
        # Update Step
        old_C = np.copy(C)
        C = update_step(X,L,C)
        #---> Stop Criterion 1
        if centroid_comparison(old_C, C) < tol:
            break
    return C, L
```

VI. K-means application in Image Compression

A. Digital Representation of Images

A digital image is a 2-dimensional array of pixels [10]. Specifically, colored images are represented using different color models and CMYK, sRGB and RGB are some of them. In the RGB model, colors are obtained from the combination of 3 basic ones: Red, Green, and Blue. For this reason, one more dimension in the matrix is needed for the color channel. Then, an RGB image can be represented by a three-dimensional matrix with dimensions $width \times height \times 3$.

In an RGB image, the size of each image pixel is 3 bytes or 24 bits, and each byte contains the intensity values for Red, Green, and Blue, in a range from 0, black, to 255, white. In this way, the total number of colors that can be represented in a single pixel is $256 \cdot 256 \cdot 256 = 16777216$.

B. Image Compression

Image compression represents a type of data compression used to reduce the information required to represent an image. The reduction in a file size allows more images to be stored in a specific amount of disk or memory space [11] [12].

Image compression can be classified into two types: *lossless* and *lossy*. On the one hand, *lossless compression*, does not imply loss of information since it encodes all image information in such a way that the exact original information can be

recovered when decompressed [10]. On the other hand, the objective of *lossy compression* is to increase the compression rate in exchange for less accuracy in the reconstructed image, i.e., some of the original information is lost in the compression process [12]. This last can be carried out with k-means.

C. Image compression using k-means

As explained in Section VI-A, a single pixel can take on approximately 16.7 million colors. However, just a slight variation in one of the RGB intensities produces a different color, which in most cases is imperceptible to human vision. Based on this, it results evident that a image could be represented with a smaller number of colors without the human eye perceiving a noticeable change. In this manner, the role of k-means in image compression is to group pixels with similar colors into k groups or clusters, so that each pixel acquire the RGB value of the centroid of its cluster.

As seen in VI-A, each pixel requires 3 bytes, and each byte is made up of 8 bits, taking N as the total pixels in the image, then $N \cdot 3 \cdot 8 = 24N$ bits are needed to save it. However, if pixels are grouped into k similar colors, a maximum of $N \cdot \log_2 k$ bits will be required. In the case of a 200×200 pixels image, it would need $200 \cdot 200 \cdot 24 = 960000$ bits, however, taking $k = 16$, i.e., compressing the image so that it only contains 16 colors, then it will now need $200 \cdot 200 \cdot \log_2 16 = 200 \cdot 200 \cdot 4 = 160000$ bits.

D. Implementation in Python

To perform image compression with k-means it is necessary to use 3 basic functions: `Image2array`, `array2image` and `image_compression`.

The libraries used for the implementation are `PIL`, which allows working with images, and `os`, to perform operations related to the system.

1) Image to Array conversion: The `image2array` function converts an image to a two-dimensional array of dimensions $(w \times h) \times 3$, where w and h are the width and height in pixels. In order to do it, it receives as input, `image`, the image to be compressed, and returns `X`, a bidimensional array. Each row of the array contains an RGB triplet.

```
def image2array(image):
    image = np.asarray(image)
    image = image/255 # Data Normalization
    X=np.reshape(image,(image.shape[0]*image.shape[1],3))
    return X
```

2) Array to Image conversion: The `array2image` function converts an array to an image. To do this, it receives as input a two-dimensional array `X`, and returns `compressed`, an image. To perform this conversion, first the array is transformed to the original dimensions of the image $width \times height \times 3$, with `np.reshape`. Finally, to obtain the image from the three-dimensional array, the `Image.fromarray` function from `PIL` is used.

```
def array2image(array,dimensions):
    array = array*255
    array= np.array(array, dtype = np.uint8)
    array = np.reshape(array,(dimensions[1],dimensions[0],3))
    compressed = Image.fromarray(array)
    return compressed
```

3) Image Compression function: The function `image_compression` receives as parameters `name`, a string containing the name of an RGB image and its extension, and `k` the desired number of colors. It invokes the `image2array` function to obtain an array `X` that is then passed to the `k_means` function. From `k_means`, the arrays `C`, the centroids, and `L`, the number of centroid for each pixel, are obtained. The next steps consists in the reconstruction of the array image using the new RGB combinations in `C`, and its conversion to an image format with `array2image`. Finally, the compressed image is stored in the working directory under the name `'name_comp.extension'`.

```
def image_compression(name,k):
    image = Image.open(name)
    X = image2array(image)
    #Apply K-means to the dataset
    C, L = k_means(X,k,iterations,initialization)
    # Generate an array, with the new RGB values
    compressed = C[L]
    compressed = array2image(compressed,image.size)
    # The new image is saved
    only_name = os.path.splitext(name)[0] #Image Name
    extension = os.path.splitext(name)[1] #Image Extension
    compressed.save(only_name +'_comp'+extension)
```

VII. Results

Compression using k-means has been applied to an image of size 1280×720 pixels. Figure 2 shows the result of compressing the image to 32 colors. In terms of size, it has been reduced from 1.41 MB or 1480937 bytes to 554 KB or 567388 bytes. Moreover, Figure 3 shows the color space plot for the original and the compressed images. In the case of Figure 3a, it is observed that the original image contains a total of 222992 colors. Likewise, Figure 3b shows that the compressed image is made up of a total of 32 colors, considerably less than before. Moreover, the resulting compression ratio was $2.61 : 1$, which means that the image was compressed to less than half the original size.

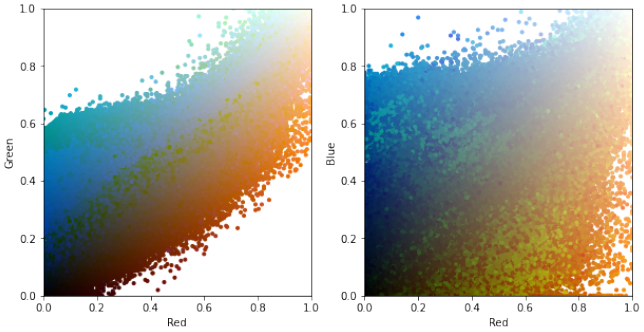
The compression algorithm has also been applied varying the number of clusters k, and its initialization method. In the first case, an image of 400×400 pixels with a size of 273 KB and 70221 different colors was selected. The clustering technique was applied with a constant number of 15 iterations varying the number of colors k in powers of 2 from 8 to 128. As Figure 4 shows, since k matches the numbers of colors available for each pixel, as k increases, the resulting image acquires more similar characteristics and colors to the original.

The resulting sizes in bytes after compression are shown in Table I. It is verified that the smaller the value of k, the smaller the space in bytes required. In addition, it is observed that the size obtained using the k-means++ initialization is smaller than that of the random initialization. While the random initialization achieved a maximum compression ratio of $5.23 : 1$, with 8 colors, and a minimum of $1.29 : 1$, with 128, using k-means++ this ratio was between $5.59 : 1$ and $1.44 : 1$.

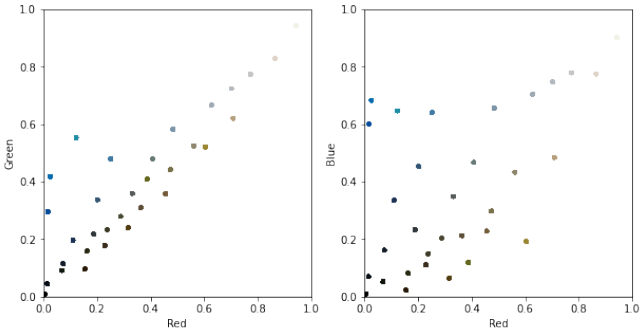
(a) Original Image

(b) 32-color Image

Fig. 2. Original and 32-color images comparison

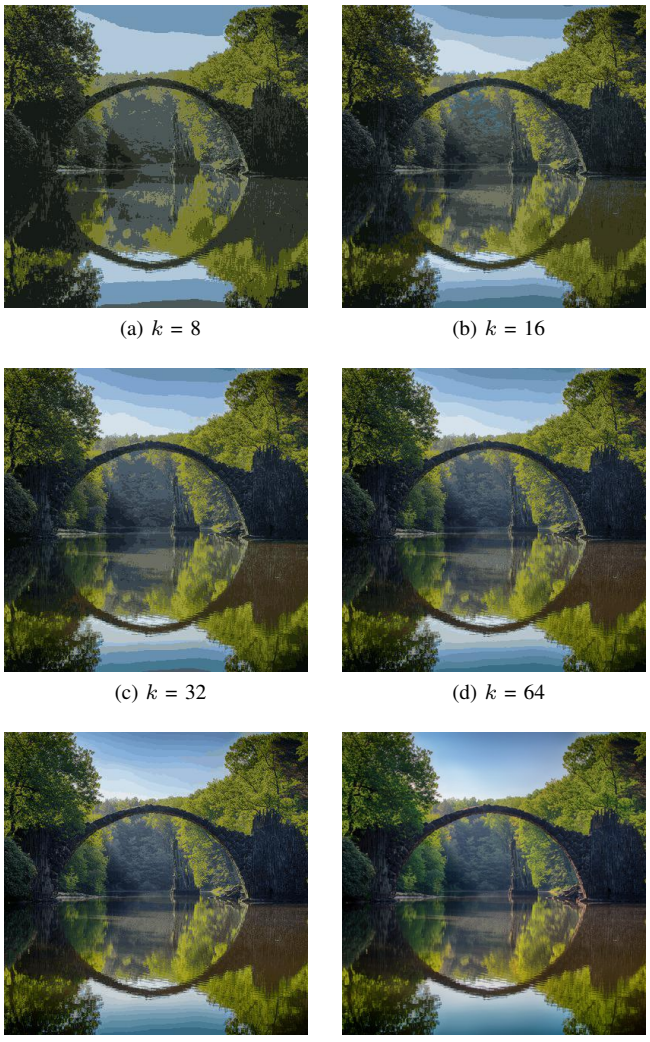


(a) Input color space: 222992 colors



(b) Reduced color space: 32 colors

Fig. 3. Color space comparison for the original and compressed images



(a) k = 8 (b) k = 16

(c) k = 32 (d) k = 64

(e) k = 128 (f) Original

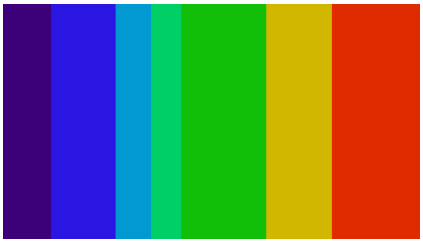
Fig. 4. Image compression for different values of k

Based on this results, the compression process has been successful for each value of k and the proposed algorithm managed to considerably reduce the size in KB of the image using only 128 colors, and preserving most of the original characteristics, as seen in Figure 4e.

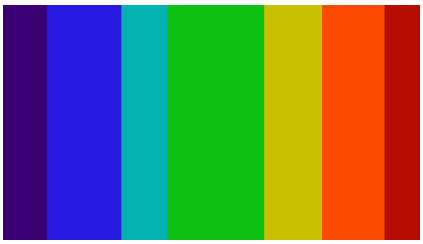
TABLE I
IMAGE SIZES AFTER COMPRESSION

k	Image Size (KB)		Compression Ratio	
	Random	***K-means ++***	***Random***	***K-means ++***
8	52.1	48.8	5.23	5.59
16	87.7	75.0	3.11	3.64
32	125	114	2.18	2.39
64	169	152	1.61	1.80
128	211	189	1.29	1.44

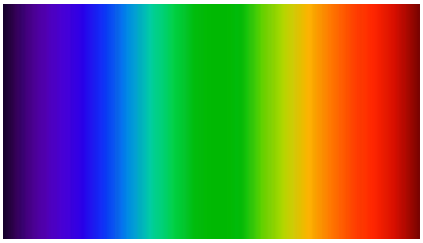
Finally, the compression process was applied to an 800×450 pixel image. Figure 5 shows a comparison of the resulting compressed image using 15 iterations and 7 colors. In terms of size, the original image required 10.0 KB to be stored. After



(a) Random Initialization



(b) K-means ++ Initialization



(c) Original

Fig. 5. Image Compression for different initialization methods

applying the compression process, the size was 2.11 KB in both cases, resulting in a compression ratio of $4.74 : 1$.

It is clearly observed that the clustering performed from the initial centroids selected with k-means++ seems to be much more adequate. The compressed image shown in Figure 5b contains stripes with colors more representative of the original image.

More tests were performed with both initialization methods. While k-means ++ obtained results quite similar in subsequent tests, although the random initialization managed to obtain similar results to k-means ++ in some cases, the randomness of the choice greatly influences the final results, and a great variation was observed in the results from one test to another.

VIII. K-means Complexity

A. Design paradigm

K-means can be classified into the Brute Force algorithmic paradigm, or more specifically, Exhaustive-Search. This is justified by the fact that this algorithm in each iteration is responsible for computation of distances between each of the points of the data set towards each of the centroids, in order to find the shortest possible distance, that is, the best candidate among those available.

B. Mathematical Analysis

To carry out the mathematical analysis it is first necessary to identify the basic operation of the algorithm, i.e. the operation that contributes the most to the total execution time. In addition, it is also necessary to take into account that in the present article two types of initializations are being considered.

For the case of *random initialization*, when looking at the `k-means` function, it in turn calls two other functions called `assignment_step` and `update_step`. The basic operation in this case is inside `assignment_step` function. Specifically the calculation of the euclidean distance between the points in `X` and the centroids is the most executed operation in the program. The `for` loop performs a total of `n` iterations, a number that matches the number of rows in `X`. Although the code has been optimized using the `np.linalg.norm` function for a much faster calculation, for the purpose of this analysis it is needed to take into account the complete process behind that function. For the calculation of the distances, the program must iterate between each one of the `k` centroids, and likewise, between each one of the `d` components of these vectors. Therefore, the real basic operation of the algorithm would be the calculation of $(x_i - c_i)^2$, i.e., the squared difference between the ith component of a point x and a centroid c.

1) Best Case: The best case of k-means considers a data set composed of only one element that has one dimension. In this case, the number of iterations required, the number of clusters, and the data dimension value `d` would be equal to 1. Therefore, the basic operation would be executed only once, and the complexity of k-means in the best case would be constant, i.e.:

$$C_{best}(n) = 1 \in \Omega(1)$$

2) Average Case: Let t be the number of iterations, n the number of points in the data set, k the number of desired clusters, and d the dimension of each data item, considering that the most common in clustering tasks is $t, k, d \ll n$, then:

$$\begin{aligned} C_{avg}(n) &= \sum_{w=0}^{t-1}\sum_{x=0}^{n-1}\sum_{y=0}^{k-1}\sum_{z=0}^{d-1} 1 = \sum_{w=0}^{t-1}\sum_{x=0}^{n-1}\sum_{y=0}^{k-1}(d-1-0+1) \\ &= \sum_{w=0}^{t-1}\sum_{x=0}^{n-1}\sum_{y=0}^{k-1} d = d\sum_{w=0}^{t-1}\sum_{x=0}^{n-1}(k-1-0+1) \\ &= d\sum_{w=0}^{t-1}\sum_{x=0}^{n-1} k = kd\sum_{w=0}^{t-1}(n-1-0+1) \\ &= kd\sum_{w=0}^{t-1} n = n\cdot k\cdot d\cdot(t-1-0+1) \\ &= t\cdot n\cdot k\cdot d \in \Theta(n) \end{aligned}$$

In the case of the *k-means++* initialization, when observing the `k_means_pp_initialization` function, it is clear that the basic operation is inside the nested `for` loop. Let k be the number of clusters to be calculated, n the number of points in the data set, cc the number of centroids calculated

until a given moment, and d the dimension of each vector, taking into account that commonly $k \ll n$, and $cc \leq k$, then:

$$C_{avg}(n) = \sum_{w=1}^{k-1}\sum_{x=0}^{n-1}\sum_{y=0}^{cc-1}\sum_{z=0}^{d-1} 1 = \sum_{w=1}^{k-1}\sum_{x=0}^{n-1}\sum_{y=0}^{cc-1}(d-1-0+1)$$
$$= \sum_{w=1}^{k-1}\sum_{x=0}^{n-1}\sum_{y=0}^{cc-1} d = d\sum_{w=1}^{k-1}\sum_{x=0}^{n-1}(cc-1-0+1)$$
$$= d\sum_{w=1}^{k-1}\sum_{x=0}^{n-1} cc = cc\cdot d\sum_{w=1}^{k-1}(n-1-0+1)$$
$$= cc\cdot d\sum_{w=1}^{k-1} n = n\cdot cc\cdot d(k-1-1+1)$$
$$= (k-1)\cdot n\cdot cc\cdot d \in \Theta(n)$$

Finally, as the complexity of *k-means* and *k-means++* is the same, the complexity of the algorithm in the average case is $\Theta(n)$.

3) Worst Case: It is known that $k \leq n$, then, the worst case for this algorithm would occur when $k = n$, and the dimension of the data is such that $d \geq n$, i.e., it is needed to group n d-dimensional data items into n clusters. Thus the complexity of k-means for both initialization methods would be:

$$C_{worst}(n) \in \mathcal{O}(n^3)$$

4) Space Complexity: The space efficiency of k-means is $\Theta((n+k)d)$. In other words, it is required to store n data and k centroids, both of dimension d.

C. Empirical Analysis

To perform the empirical analysis, the execution time in seconds of k-means, using both initialization methods, to perform compression of images of 8 different sizes was measured. For each image, a total of 10 runs were performed, and an average time was extracted. Details of the image sizes, the data size, i.e., the number of bytes required to store each image, and the average time in seconds are shown in Table II.

TABLE II
K-MEANS AVERAGE EXECUTION TIME

Image Size	**Data items** (n)	**Average time** (s)	
		Random	***K-means ++***
200 × 200	40000	9.7388	21.0083
300 × 240	72000	19.1759	41.8122
450 × 300	135000	34.0075	71.4299
400 × 400	160000	38.335	83.2811
600 × 600	360000	85.8228	186.89
800 × 500	480000	113.9574	250.7792
1024 × 576	589824	140.1652	308.3514
1280 × 720	921600	222.5237	477.6426

From the measured times, as expected, it is observed that the greater the number of elements in the data set, the greater the execution time. It is also possible to see that the times when using k-means++ are approximately double those obtained when applying a random initialization.

Fig 6 shows a graph of time in seconds vs. number of elements in the dataset. It is observed that although k-means++ has a higher execution time, the algorithm time efficiency class is linear $\Theta(n)$ for both initialization techniques, which is in agreement with the mathematical analysis.

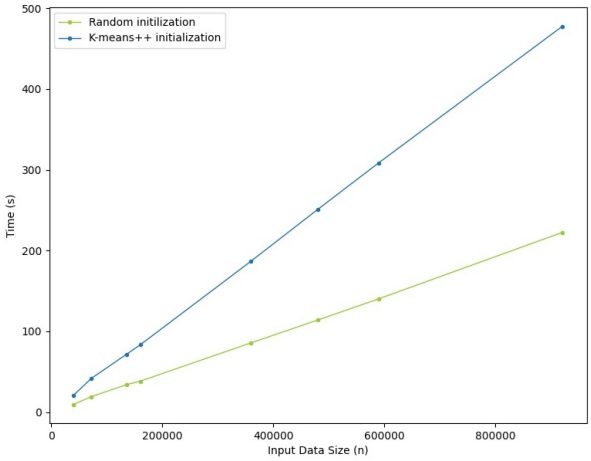


Fig. 6. Time in seconds (s) vs Number of data items (n)

IX. Conclusion

K-means is a famous and widely used Machine Learning technique for clustering tasks due to its simplicity and efficiency. The present work has presented an analysis of the algorithm including the explanation of its operation, performance, complexity, and application in clustering tasks. Additionally, it was shown that the Euclidean distance calculus constitutes the basis of k-means, and the correct choice of k and a proper initialization method are vital to its correct operation. Also, the feasibility of the algorithm in lossy compression tasks has been verified, achieving satisfactorily a reduction in the size of RGB images.

Based on the k-means operation, it could be classified as an Exhaustive Search algorithm. Furthermore, using a mathematical and empirical analysis, it was determined that the complexities of k-means in the best, average, and worst cases correspond to $\Omega(1)$, $\Theta(n)$ and $\mathcal{O}(n^3)$ respectively.

This paper has also verified that although the initialization of the centroids is one of the weaknesses of the algorithm, it can be addressed by implementing another algorithm called k-means ++, which ensures a better selection criterion, in exchange for a longer execution time. However, it was shown that the increase in time did not imply a change in the complexity of the algorithm, and it remained linear.

Finally, for the future work would be of great interest to address the application of k-means in other clustering tasks such as market segmentation, and recommendation systems, or propose modifications to the algorithm, which in turn allow improvements in its complexity, execution time, and results.

REFERENCES

[1] S. Ray, "A quick review of machine learning algorithms," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 35–39.

[2] A. Subasi, "Chapter 1 - introduction," in *Practical Machine Learning for Data Analysis Using Python*, A. Subasi, Ed. Academic Press, 2020, p. 1.

[3] T. O. Ayodele, "Types of machine learning algorithms," *New advances in machine learning*, vol. 3, pp. 19–48, 2010.

[4] S. Na, L. Xumin, and G. Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *2010 Third International Symposium on intelligent information technology and security informatics*. Ieee, 2010, pp. 63–67.

[5] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, 2003, biometrics.

[6] K. P. Sinaga and M.-S. Yang, "Unsupervised k-means clustering algorithm," *IEEE Access*, vol. 8, pp. 80 716–80 727, 2020.

[7] Y. Li and H. Wu, "A clustering method based on k-means algorithm," *Physics Procedia*, vol. 25, pp. 1104–1109, 2012, international Conference on Solid State Devices and Materials Science, April 1-2, 2012, Macao. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1875389212006220

[8] S. Ray, "A quick review of machine learning algorithms," in *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, 2019, pp. 35–39.

[9] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[10] M. Singh, S. Kumar, S. Singh, and M. Shrivastava, "Various image compression techniques: Lossy and lossless," *International Journal of Computer Applications*, vol. 142, no. 6, pp. 23–26, 2016.

[11] A. Munshi, A. Alshehri, B. Alharbi, E. AlGhamdi, E. Banajjar, M. Albogami, and H. S. Alshanbari, "Image compression using k-mean clustering algorithm," *International Journal of Computer Science & Network Security*, vol. 21, no. 9, pp. 275–280, 2021.

[12] A. J. Hussain, A. Al-Fayadh, and N. Radi, "Image compression techniques: A survey in lossless and lossy algorithms," *Neurocomputing*, vol. 300, pp. 44–69, 2018.