Stockholms universitet

# Initialization of the k-means algorithm
# A comparison of three methods

Simon Jorstedt

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2023:1
Matematisk statistik
Januari 2023

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2023:1**
http://www.math.su.se

# Initialization of the k-means algorithm
# A comparison of three methods

Simon Jorstedt*

June 2023

**Abstract**

k-means is a simple and flexible clustering algorithm that has remained in common use for 50+ years. In this thesis, we discuss the algorithm in general, its advantages, weaknesses and how its ability to locate clusters can be enhanced with a suitable initialization method. We formulate appropriate requirements for the (batched) UnifRandom, k-means++ and Kaufman initialization methods and compare their performance on real and generated data through simulations. We find that all three methods (followed by the k-means procedure) are able to accurately locate at least up to nine well-separated clusters, but the appropriately batched UnifRandom and the Kaufman methods are both significantly more computationally expensive than the k-means++ method already for $K = 5$ clusters in a dataset of $N = 1000$ points.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: jorstedtsimon@gmail.com. Supervisor: Taras Bodnar, Lina Palmborg, Dongni Zhang.

**Acknowledgements**

Thank you to my supervisors Taras Bodnar, Lina Palmborg and Dongni Zhang for providing very useful feedback, advice, time-saving LaTeX tricks and for gently nudging me to "be efficient" and "hurry up". Thank you so much Max, for believing in me. Thank you Josefin for your advice, for cheering me on, and for the coffee. Thank you Dad, for bearing with me through my complicated parables. Thank you Erika for cheerfully proclaiming "kill your darlings!" some Sunday a long time ago. My sincere gratitude I extend to Naturvetarspexet, Monty Python and Python the programming language for the silly and simple things in life. Thank you also to the 68 cups of coffee that perished in the making of this project.

# Contents

# 1 Introduction

A core problem in the field of Unsupervised Machine Learning is the clustering problem, concerning classification of data into unnamed groups without records of the "true" groups or often not even the number of groups. Various types of data (binary, ordinal, quantitative etc) and other variations of the problem have lead to "a plethora of methods being proposed since the 1950s" according to Celebi, Kingravi, and Vela 2013. One simple brute-force type approach is to attempt to generate all possible partitions of the data, evaluate some objective function for each, and then select the partition which optimises the objective function. According to Celebi, Kingravi, and Vela 2013, the number of possible partitions of a dataset with $N$ points and $K$ groups is equal to the Stirling number of the second kind:

$$S(N,K) = \frac{1}{K!}\sum_{k=0}^{K}(-1)^{K-k}\binom{K}{k}k^N$$

This makes it impractical to generate and compare all possible partitions unless for very small datasets. This can however be circumvented using combinatorial algorithms and iterative greedy decent (as discussed briefly in Hastie et al. 2009, p. 507-509), to try to adjust the partitioning in a smart way. More often, "centroid seeking" clustering algorithm are used. They instead try to either find representative instances in the available data, or to create such instances in the dataspace. Datapoints are then assigned to the representative instances by some rule, typically based on a distance measure.

In this thesis, we will introduce the k-means algorithm, the concept of an initialization method and the technique of batching. We will use these concepts to create and evaluate a new initialization method which we will call the R-batched UnifRandom method. The following Sections are structured as follows. Section 2 introduces the k-means algorithm and its limitations. In Section 3 the idea of an initialization method is introduced, and three methods: UnifRandom, Kaufman and k-means++ are presented and discussed. In Section 4 the technique of batching is discussed and the R-Batched UnifRandom method is presented. In Section 5 we discuss the datasets and type of data we will analyse. In Section 6 we discuss the simulations and the performance measures we will use in our analysis. In Section 7 we test, compare and discuss the initialization methods. In Section 8 we summarise our conclusions and mention some recent contributions to the topic of this thesis.

# 2 The k-means algorithm

The k-means algorithm is a partitioning cluster algorithm taking a dataset $X = \{x_1, ..., x_N\}$ of $N$ vectors $x_i \in \mathbb{R}^d$, each with $d$ variables and attempts to separate the vectors into $K$ disjoint clusters, represented by their respective cluster centres $\mu_k \in \mathbb{R}^d$, $k = 1, ..., K$. Points are assigned to the closest centre (typically measured by Euclidean distance), forming so-called Voronoi cells[1] (Hastie et al. 2009, p. 510) The method starts with an initial guess (or "seed") $\boldsymbol{\mu}^{(1)} = (\mu_1^{(1)}, ..., \mu_K^{(1)})$ of the true cluster centres and a specification of $K$, and then attempts to minimize an objective function: typically the sum of squared Euclidean distances SSE between each point and its closest centre[2]. In any iteration $j$ we define $\mu(x)$ as the centre in $\boldsymbol{\mu}^{(j)}$ that is currently closest to a point $x$. We can then formulate the SSE as

$$SSE = \sum_{i=1}^{N} \|x_i - \mu(x_i)\|^2.$$

Many variations of the k-means algorithm exist, for example the Hartigan-Wong and Macqueen variations (Oti et al. 2020). The standard k-means algorithm or *Lloyd's algorithm* (Lloyd 1982) (which we will be using in this thesis) is adapted for quantitative/real-valued data, which is the usual application. Lloyd's algorithm attempts to achieve minimization of the SSE through an iterative procedure. After producing some initial centres $\boldsymbol{\mu}^{(1)}$, the procedure assigns each point in the dataset to its nearest centre, and then recalculates new updated cluster centres $\boldsymbol{\mu}^{(2)}$ as the mean of the points in each cluster.

[1] *Voronoi tesselation* is the partitioning of points or space into Voronoi cells. One Voronoi cell contains all points that are closer to that cells center-point (or *centroid*) than any other center-point.

[2] SSE is an abbreviation for "Sum of Squared Errors".

This iterative process is then repeated until either convergence is achieved, or some specified number of iterations is reached. By denoting the set of points assigned to cluster $k = 1, ..., K$ in the $j$th iteration $(j = 2, 3, ...)$ with $X_k^{(j)}$, and the number of points in $X_k^{(j)}$ with $N_k^{(j)}$, we can formulate the recalculation of each cluster center in the $j$th iteration as

$$\mu_k^{(j)} = \frac{1}{N_k^{(j)}} \sum_{x \in X_k^{(j)}} x \quad \text{for} \quad k = 1, ..., K$$

The Pseudocode for the standard k-means algorithm is described in Figure 1, inspired by Arthur and Vassilvitskii 2007. We will refer to the entire algorithm (steps 1-4) as the k-means algorithm, to differentiate from the k-means procedure or refinement, which we will use to refer to steps 2-4. This is because they are refining the initial seeding in order to produce a final clustering solution that has converged to some optima. In Figure 2 we demonstrate a simple application of the k-means algorithm on a small dataset divided into three somewhat overlapping clusters.

- **Step 1: Initialization.** The number of clusters K is selected. A seed $\boldsymbol{\mu}^{(1)}$ is selected.
- **Step 2: Partition.** All points $x$ are assigned to the nearest cluster centre, forming $K$ disjoint sets.
- **Step 3: Recalculation.** For each cluster, the arithmetic mean of the associated points is calculated, together becoming the updated cluster centres $\boldsymbol{\mu}^{(j)}$.
- **Step 4: Repetition.** Calculate the SSE, evaluate the convergence criteria and stop if it is met or some maximum number of iterations is reached. Otherwise repeat steps 2 and 3.

Figure 1: Pseudocode for the standard k-means algorithm.

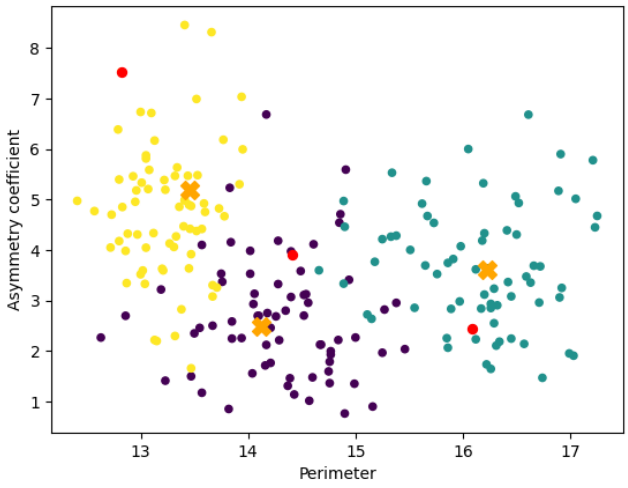


Figure 2: A simple application of the k-means algorithm on two types of measurements on three types of seeds (yellow, purple and green dots indicating true cluster belonging). Initial seeding (red dots) and final clustering solution (orange crosses) are included. See Bishop 2006, p. 426.

The k-means algorithm is guaranteed to converge (Hastie et al. 2009, p. 510). This is because both the partition and recalculation steps decrease or maintain the SSE. In the partitioning step, each point

is either assigned to a closer centroid (decreasing the contribution to the SSE), or kept in its current cluster (not changing the SSE contribution). In the Recalculation step, each cluster center is replaced such that it minimizes the SSE within its cluster. This will either decrease or maintain the total SSE. Unfortunately, this does not guarantee convergence to the global optima. This is in fact a well known disadvantage of the k-means algorithm. According to Steinley and Brusco 2007, Steinley 2003 [3] suggested that the number of local optima for already a small dataset could be in the thousands.

The k-means algorithm has several compelling advantages. Firstly it is a very simple algorithm (Fränti and Sieranoja 2019), making it understandable and easy to convey. In addition to this, it can be visualised very easily in two dimensions. Secondly it is very flexible, since most aspects of it can be modified, for example the distance measure, initialization and stopping condition (Celebi, Kingravi, and Vela 2013). Thirdly, it is fast (Arthur and Vassilvitskii 2007). Celebi, Kingravi, and Vela 2013 stated that the k-means algorithm has a time complexity that is linear in the parameters $N$, $d$ and $K$. In practice however, $N$ is typically much larger than $d$ and $K$, and therefore contributes most to the computational expense. Fourthly, as indicated by this paragraph, the k-means algorithm has been extensively studied (Fränti and Sieranoja 2019), and its properties are very well known. This makes it straightforward and reliable to work with. According to Fränti and Sieranoja 2019, p. 96, k-means also has "excellent fine-tuning capabilities", in that as long as an initial guess is locally close to the "true" centre, k-means can produce a good approximation. However, this is heavily reliant on the initialization methods ability to locate clusters globally.

k-means also has disadvantages. According to Arthur and Vassilvitskii 2007 it "offers no accuracy guarantees". The convergence, which again very easily falls into local optima is entirely up to the initialization, since the initialization is deterministic to the clustering solution (He et al. 2004). By that we mean that the k-means procedure is deterministic given an initialization and some convergence criteria, and therefore the goodness of the final clustering solution is dependent on the goodness of the initialization.

Due to the usage of the Euclidean distance, the standard k-means algorithm is best suited for "spherical" or "ball-like" clusters of data (in any dimension), with similar within-cluster variances. An example shown in Figure 3 shows the splitting of clusters which may occur when clusters have different within-cluster variances. Some variations of the k-means algorithm attempt to deal with this, for example the Fuzzy C-means (FCM). In some cases it is however more suited to use a different method entirely. K-means is specifically favoured for problems of low to medium complexity (small database and/or low dimension), because of its inability to describe more complex data structures.

According to Fränti and Sieranoja 2019, the k-means algorithm performs poorly on well-separated clusters, because this limits the algorithms ability to move centroids around globally. Rather, centroids may be bound to the clusters they were assigned to in the initialization. We are interested in this type of well separated data, and which initialization methods that can be efficiently employed to identify these well separated clusters. The definition of well-separated is a bit vague, but essentially means that the distances between clusters is much larger than the distances between points within each cluster. In Section 5 we go into more detail about how the datasets are simulated in order to get well-separated clusters.

# 3 k-means Initialization

As previously stated, the initialization is deterministic to the k-means refinement, making it crucial in achieving a good clustering. This also motivates the need for an intelligent method, that can counter the disadvantages of the k-means algorithm, specifically the numerous local optima. We will now briefly discuss the concept of an initialization method, and then present the three methods, paying particular attention to how they aim to locate clusters.

[3] Steinley 2003 was not available to the author of this thesis.

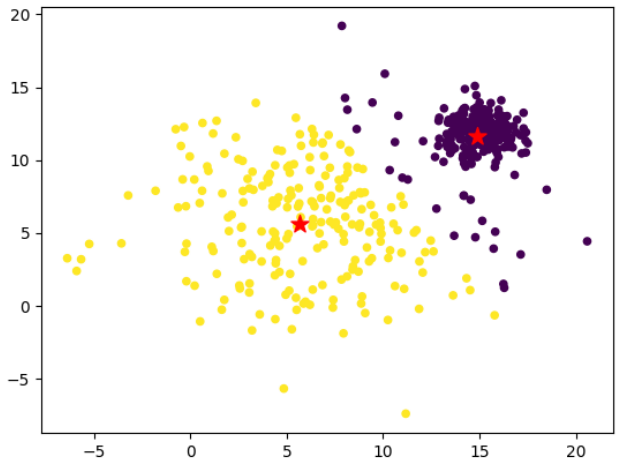


Figure 3: Two simulated clusters with different within-cluster variances placed closely with a lot of overlap. This has caused the more denser cluster to claim some of the points of the less dense cluster. The color of the points indicates cluster assignment by the k-means algorithm.

Broadly speaking, k-means initialization involves choosing a number of clusters $K$ if not provided, and a set of initial clusters $\boldsymbol{\mu}^{(1)}$. Concerning the choice of $K$, the heuristic *Elbow method* can be employed (Hastie et al. 2009, p. 519)[4]. This involves running k-means for some plausible consecutive integer values, say $(I_1, I_2, \cdots, K, K+1, \cdots, I_b)$ $(b \in \mathbb{N})$, for the true number of clusters $K$, and then looking for a value that greatly decreases the objective function, with subsequent values having a relatively small (yet still decreasing) effect on the objective function, indicating that values larger than $K$ causes overfitting. In our simulation study we will consider $K$ to be known, since in practice it will be available by the datasets and from the simulations. This is so that we can focus on the centroid initialization.

Some important comparative studies of the initialization method have been conducted previously. Peña, J. Lozano, and Larrañaga 1999 compared the empirical properties of four initialization methods, among them UnifRandom and Kaufman. Celebi, Kingravi, and Vela 2013 presented a comparison of some linear time methods, and provided a brief overview of many of the available methods, particularly emphasising time complexities. Outside these comparative studies, methods are often presented by the by while explaining some other aspect of the algorithm.

Although there are no standard requirements for what is and is not an initialization method, some sensible suggestions do exist. Fränti and Sieranoja 2019, p. 99, required initialization methods to be simple (easy to implement), have lower or equal time complexity (in terms of $N$) than k-means itself ($O(N)$), and that they not introduce more parameters. The time-complexity requirement was motivated by Celebi, Kingravi, and Vela 2013 because an initialization method should only complement the k-means algorithm, and not run the risk of inflating the computational expense.

## 3.1 UnifRandom method

The method which we will call the UnifRandom method was suggested along with the k-means algorithm by (Macqueen 1967) and remains widely used (Capó, Pérez, and J. A. Lozano 2022) (Khan and Ahmad 2004), likely because of its simplicity. It works by uniformly sampling $K$ points from the dataset as centres, as described in Pseudocode in Figure 4. The idea is that this causes the initial centres to tend to be spread out across the clusters, particularly in regions with many points. However, this allows for

[4]Hastie et al. 2009 explains the elbow method, but does not name it.

some clearly bad initializations such as placing multiple centroids in the same cluster, and consequently no centroids in other clusters (Capó, Pérez, and J. A. Lozano 2022) exemplified in Figure 7. One result of this may be the "splitting" of clusters as seen in Figure 3. These problems can however be somewhat circumvented by the technique of batching which we will discuss in Section 4. The UnifRandom method certainly fulfils the suggested requirements. It is very simple to implement and understand, has a time complexity of $O(N)$, and introduces no new parameters.

**Step 1a:** Select $K$ points uniformly at random from the dataset.

**Step 2-4:** Proceed with k-means refinement.

Figure 4: Pseudocode for UnifRandom initialization.

## 3.2 Kaufman method

The Kaufman initialization method (Kaufman and Rousseeuw 1990) is deterministic, and essentially attempts to select centres from the dataset by estimating density (He et al. 2004), or rather it selects points that are surrounded by many other points that are not already "claimed" by other centroids. The point that is closest to the global mean is selected as the first center. After that, the method successively selects a point from the dataset such that unselected points tend to lie closer to it than to the previously selected centres, until $K$ centres have been selected. The Pseudocode for this method is presented in Figure 5. The idea is that centroids will not be placed close to each other and certainly not in the same cluster in the case of well-separated clusters. This is exemplified in Figure 7.

The Kaufman method has been reviewed and found successful by Peña, J. Lozano, and Larrañaga 1999. It is certainly much more intelligent than the UnifRandom and k-means++ methods, but this also makes it somewhat more difficult to implement. This design also comes at the great cost of a time complexity (in terms of $N$) of $O(N^2)$ (Celebi, Kingravi, and Vela 2013). This is because it requires calculation of pairwise distances between points and centres for the selection of each centroid. Although no new parameters are introduced, the time complexity requirement and possibly the implementation requirement by Fränti and Sieranoja 2019 disqualifies the Kaufman method as a general method for practical applications.

**Step 1a:** Select the most central point as the first centre.

**Step 1b:** For every pair of unselected points $x_i$ and $x_j$ $(i \neq j)$, calculate $C_{ij} = \max(D_i - d_{ij}, 0)$ with $d_{ij} = ||x_i - x_j||$ being their Euclidean distance, and $D_i = \min_s(d_{si})$ being the distance between $x_i$ and its closest centre in $S$, the current set of centres.

**Step 1c:** For every point $x_i$, calculate $\sum_j C_{ij}$ and select the point $x$ that maximizes the sum as the next center.

**Step 1d:** Repeat step 1b-1c until $K$ instances have been selected.

**Step 2-4:** Proceed with k-means refinement.

Figure 5: Pseudocode for Kaufman initialization.

### 3.3 k-means++ method

The fairly recently suggested k-means++ method (Arthur and Vassilvitskii 2007) is a random procedure similar to UnifRandom that relies on stepwise selection. Centres are selected from the dataset based on probabilities proportional to the square of each points distance to its currently closest centre. The result is that centres tend to be far away from each other, but this is not guaranteed, which allows for a variation in the resulting seeds which similarly to UnifRandom can be utilized through the technique of batching which we will discuss in Section 4. The Pseudocode for the k-means++ method is presented in Figure 6. In Arthur and Vassilvitskii 2007 "k-means++" refers to the entire algorithm (steps 1-4) in Figure 6, but in this thesis we will let k-means++ refer to only the initialization method (step 1a-1c).

k-means++ is only slightly more complex than the UnifRandom method, has a time complexity of $O(N)$, and introduces no new parameters. It therefore also fulfils the requirements in Fränti and Sieranoja 2019. Additionally, it showed favourable results according to the authors. A successful application of the k-means++ method is presented in Figure 7.

**Step 1a:** Select the first center $\mu_1^{(1)}$ uniformly random from the dataset.

**Step 1b:** Out of the unselected points, select point $x$ to be the next center with the following probability, where $D(x)$ is the distance between $x$ and center that is closest to $x$. Note that $D(x) = 0$ if $x$ has already been selected.

$$\frac{D^2(x)}{\sum_{x \in X} D^2(x)}$$

**Step 1c:** Repeat step **1b** until $K$ centers have been selected.

**Step 2-4:** Proceed with k-means refinement.

Figure 6: Pseudocode for k-means++ initialization.

## 4 Batching and Delegation

Batching is a common and simple technique that can be used to improve the k-means algorithm (Fränti and Sieranoja 2019). It consists of repeating the initialization and the k-means procedure $R$ times[5], and selecting the best outcome, in terms of minimal SSE. Clearly batching is only relevant for methods that rely on some form of randomness, which includes the UnifRandom and the k-means++ but excludes the Kaufman method. This is because batching of deterministic methods simply produces multiple identical cluster seedings.

The appropriate number of repetitions $R$ in a given situation depends greatly on the dataset and to some extent the initialization method (Fränti and Sieranoja 2019, p. 98). In the literature, suggestions of constant values are made seemingly arbitrarily. Fränti and Sieranoja 2019 suggest using the expected number of repetitions $R = 1/\mathbb{P}(\text{Success})$ when viewing the k-means algorithm as a Bernoulli trial with probability $\mathbb{P}(\text{Success})$ of the k-means algorithm succeeding in finding the optimal clustering. However, they instead use various constants (e.g. $R = \{10, 100, 5000\}$) since $\mathbb{P}(\text{Success})$ is generally unknown. We will advance this reasoning in two ways.

We will let *delegation* refer to the structure of the partitioning of centroids into clusters by an initialization method. We will then say that *good* delegation is achieved when exactly one centre is placed in

[5]We consider $R$ a parameter associated with the technique of Batching, rather than with any of the initialization methods.

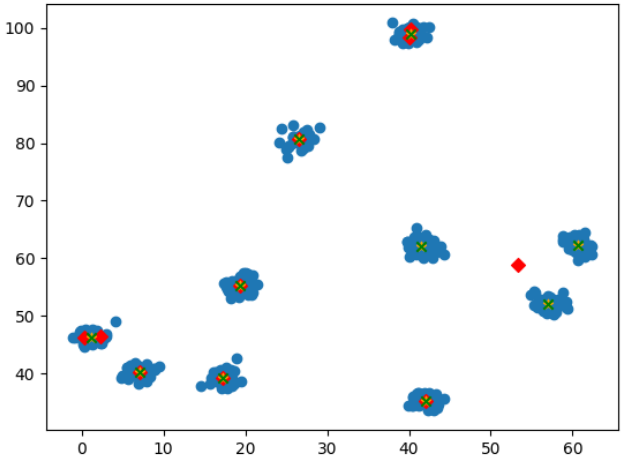


Figure 7: Demonstration of clusterings by UnifRandom (red diamonds), k-means++ (orange stars) and Kaufman (green crosses) on an artificial dataset with 10 clusters. All methods used only 1 start, and each methods clustering was refined by the k-means procedure.

each cluster. We will attempt to estimate the probability $p$ of achieving good delegation, and secondly we will select the number of repetitions $R$ such that the cumulative probability $P$ of achieving good delegation in at least one of the $R$ repetitions (or trials) surpasses some specified value. In practice, $P$ may be chosen based on accuracy requirements in a given problem, but we will use a constant $P = 0.95$ for the purpose of demonstration.

Additionally, we note that if good delegation guarantees that the optimal clustering is obtained, then we have that the total probability $\mathbb{P}(\text{Success after } R \text{ repetitions})$ of obtaining the optimal clustering after $R$ repetitions must be larger than $P$. This is because of the additional possibility that the optimal clustering is obtained even if good delegation is not achieved in any of the $R$ repetitions. We have thus a possible lower bound for $\mathbb{P}(\text{Success after } R \text{ repetitions})$ which we formulate in Equation 1. Again we must stress that this relies on the assumption that good delegation guarantees achieving the optimal clustering. This is true for data with well-separated clusters, because the k-means algorithm isolates the clusters and has no way of changing the delegation if good delegation is already achieved. This likely makes the assumption less appropriate in the case of closely placed clusters.

$$\mathbb{P}(\text{Success after } R \text{ repetitions}) \geq P. \tag{1}$$

For the UnifRandom method, the problem of estimating $p$ and thus finding a suitable $R$ is quite simple. Let us assume that we have a dataset $X$ in which the $N$ datapoints are evenly divided into the $K$ clusters, and that $N$ is sufficiently large (compared to $K$) such that UnifRandom samples approximately with replacement. In other words, the probability of drawing a point from a certain cluster remains approximately unchanged through the sampling. Note: *individual* points can still not be chosen more than once. Under these assumptions of evenly sized clusters and sampling with replacement, to achieve good delegation essentially requires selecting one cluster out of $K$ clusters, then one cluster out of $K-1$ clusters etc. The probability of achieving good delegation in any one of the independent repetitions while batching is therefore approximately

$$\frac{K}{K} \cdot \frac{K-1}{K} \cdot \frac{K-2}{K} \cdots \frac{1}{K} = \frac{K!}{K^K}. \tag{2}$$

Now let us consider a process in which we perform repetitions of the k-means algorithm until good delegation is finally achieved. Under our assumptions, the number of repetitions $R$ we perform in total

is then (approximately) geometrically distributed with parameter $p$ equal to Equation (2). We have thus in Equation (3) a method for estimating the cumulative probability $P$ given a number of repetitions $R$, which can be reversed into Equation (5). We can now use Equation (5) to produce recommendations on the number of repetitions $R$ to use for a given dataset. Formally we have that

$$R \sim \text{Geo}(p)$$

$$P = \mathbb{P}(R = r) = p \sum_{i=0}^{r-1} (1-p)^i = 1 - (1-p)^r, \quad r = 1, 2, ... \tag{3}$$

where

$$p \approx \frac{K!}{K^K} \tag{4}$$

Now we can solve for $r$ in (3), and we achieve the following result.

$$r = \frac{\log(1 - \mathbb{P}(R = r))}{\log(1-p)}, \quad K \geq 2 \tag{5}$$

| K | $\lceil r \rceil$ |
|---|---|
| 2 | 5 |
| 3 | 12 |
| 4 | 31 |
| 5 | 77 |
| 6 | 193 |
| 7 | 489 |
| 8 | 1246 |

Table 1: Evaluation of Equation (5) for increasing values of $K$, $r$ is rounded up with the ceiling function. The probability $\mathbb{P}(r) = 0.95$ is constant in all calculations.

We will call the batched UnifRandom method with the number of repetitions $R$ recommended by Equation (5) the R-Batched UnifRandom method. Interestingly Equation (5) only requires specifying $K$, $P$ and that our two assumptions are true. This is reasonable given that UnifRandom selects points randomly, so that distance between clusters, cluster variance, dimensionality and similar aspects of the dataset should have no effect on the quality of the seed. In Table 1 we illustrate the quick diverging growth of Equation (5). This indicates that under our assumptions, R-Batched UnifRandom quickly becomes very computationally expensive as $K$ increases. Already for $K = 10$ we are advised to use $R = 8254$.

The two assumptions that the R-Batched UnifRandom method rely on are not always appropriate, but in many common situations they are, including all datasets in this thesis. The assumption of equally sized clusters is typically true when one is studying the properties of some distinct subgroups in a population of objects, rather than the relative sizes of the subgroups within the population, for example in a political survey in which case cluster analysis itself may be inappropriate. In the right context these even sizes are often by design. The same number of observations are often drawn from each subgroup to avoid bias and allow a fair comparison. This is because uneven cluster sizes may lead to one cluster having a larger impact on the objective function than the other clusters and thus skewing the results.

The second assumption that the probabilities of drawing from each cluster is approximately unchanged during the UnifRandom sampling is not immediately obvious and not always appropriate. We consider a database with $N$ points evenly divided into $K$ clusters. That is, each cluster is assumed to have $N/K$ points. The greatest difference in probabilities when selecting a point from one cluster or a point from another cluster occurs in the unlikely event that one cluster ($k$) has been sampled from $(K-1)$ times.

The probability $\mathbb{P}_u$ for each of the unselected clusters to have one of its points be selected as the final centroid is then

$$\mathbb{P}_u = \frac{N/K}{N-(K-1)} = \frac{1}{K} \cdot \frac{N}{N-(K-1)} \;\rightarrow\; \frac{1}{K} \quad \text{as } N \to \infty \tag{6}$$

The probability $\mathbb{P}_k$ of the final centroid being chosen from cluster $k$ is then

$$\mathbb{P}_k = \frac{N/K-(K-1)}{N-(K-1)} = \frac{1}{K} \cdot \frac{N-K(K-1)}{N-(K-1)} \;\rightarrow\; \frac{1}{K} \quad \text{as } N \to \infty \tag{7}$$

In Equations (6) and (7) we also demonstrate how the expressions can be rewritten to show that for a constant $K$, both $\mathbb{P}_u$ and $\mathbb{P}_k$ converge to $1/K$ as $N \to \infty$. This is expected, since this is equal to the original probability for each cluster to have one of its points be selected as the first centroid. In Section 7 we will calculate and present $\mathbb{P}_u$ and $\mathbb{P}_k$ when relevant. Additionally, it is worth reiterating that while sampling the $K$ centroids, the probability of selecting a point from a particular cluster is decreased if a point has previously been selected from that cluster. This means that the true probability of achieving good delegation in a particular situation is slightly higher than Formula (4). Therefore Formula (4) provides a lower bound to the true probability of achieving good delegation in a particular situation.

Determining the probability of good delegation for the k-means++ method is much more difficult than for the UnifRandom for multiple reasons. Due to the involved distance calculations it is highly dependent on the dataset itself, not just on the values of $N$ and $K$. This along with the fact that the k-means++ method often produces a good seed with only one repetition due to its intelligent design motivates running it with only one repetition in our comparative tests, the results of which are presented in Section 7. Similarly, determining the probability of good delegation for the Kaufman method is also difficult, partly because it requires a different problem formulation due to the dataset being deterministic to the Kaufman seed. Additionally, even if acquired it could only at most serve as an advisory tool for a particular type of data since batching provides no improvement to the Kaufman method.

# 5 Datasets

In this Section, we will discuss aspects of the real and generated datasets used in our study to test the validity in our claims and to compare the initialization methods. Two real world datasets were selected, both courtesy of the UCI Machine Learning Repository (Dua and Graff 2017). The datasets were selected primarily on the criterion that true cluster belonging and the number of clusters is known, and that the k-means algorithm is appropriate for the dataset. Two additional datasets measuring wine and glass qualities respectively where eventually excluded. The first one for being too large (compromising computation time in tests), and both for requiring modifications (e.g normalizing) before application of the k-means algorithm would be appropriate.

The first dataset is the well known Iris dataset ($N = 150, d = 4, K = 3$) which measures petal and sepal width and height of 50 each of the Iris Setosa, Iris Versicolour and Iris Virginica species of the Iris family of plants. The Iris dataset can be somewhat suitably described by the k-means algorithm, but this is not ideal due to dependence between variables. The second dataset ($N = 210, d = 7, K = 3$) describes seven measurements of 70 each of three types of wheat seeds: Kama, Rosa and Canadian (Institute of Agrophysics of the Polish Academy of Sciences in Lublin. 2012).

The generated datasets are created as follows. $K$ random *true* centroids are independently generated in a $d$-dimensional hypercube with some specified side length. Each point is then created by uniformly selecting one centroid, and adding a standard normal variate to each of the $d$ components of the centroid. The random selection of centroid for each point is intended to simulate a slight variation in cluster sizes. In order to generate very well-separated clusters, we will chose a hypercube with side length 1000 in our tests. As long as this side length is much larger than the number of cluster $K$ in a dataset, clusters will very likely be located far apart, relative to the within-cluster variance.

# 6 Performance Measures

The performance of the initialization methods will be measured by three effectiveness, and two speed measures. This is a similar setup as in Celebi, Kingravi, and Vela 2013.

**Final SSE** This is the final minimal SSE across all k-means repetitions in one batching of the algorithm for a particular dataset and initialization method.

**Seed SSE** This is the SSE calculated on the seed $\boldsymbol{\mu}^{(1)}$ (called the final seed) that corresponds to the final minimal SSE and thus the final clustering in one batching of the k-means algorithm. According to Celebi, Kingravi, and Vela 2013, this indicates an initialization methods effectiveness on its own, without k-means refinement.

**Iterations** This is the average number of required iterations until convergence is reached, averaged across all repetitions of the algorithm in one batching. For the Kaufman method, this will just be the number of iterations until convergence.

**CPU time** Total elapsed time for one batch. This includes all repetitions starts in one batch (1 for the Kaufman) of both the initialization and k-means refinement. It is important to note that this measure is highly dependent on the machine that was used. It should therefore only be considered and interpreted as a relative measure.

**Accuracy** Using the true labels of each dataset, the percentage of points that have been correctly clustered is calculated. Technically this is calculated on a clustering as the percentage of points that have the same true label as the mode of the true labels in their respective assigned cluster, for practical reasons.

By looking at the effectiveness criteria Final SSE and Accuracy of all methods for a single dataset, we can potentially determine if a "correct" clustering has been discovered. Because the initialization methods are independent, if two or more achieve identical or very close Final SSE, it is very likely that they have found the same optima. Two or more methods also achieving high accuracy would indicate that their identical (or very similar) clustering is the optimal one. By considering both indicators we hope to make a more robust statement about the nature of the found optima. The speed indicators give comparative measures on how computationally costly the result of a method was.

# 7 Discussion and Results

We previously in Section 4 defined the R-Batched UnifRandom method, and in this Section we will discuss how we can verify the quality of our arguments in Section 4 and then how we can compare its performance and cost with the Kaufman and k-means++ methods. We will then perform multiple relevant tests and discuss the results. All tests and simulations are built in Python 3.10.4. The scikit-learn package was used for the k-means procedure, but all three initialization methods and the testing structure was built specifically for this thesis by the author in order to extract all the desired outputs.

For the first task, we recall our discussion about the assumptions in Section 4. From Formula (4) we are lead to believe that the probability of achieving good delegation (for UnifRandom) only depends on $K$ (given that the two assumptions are satisfied), and not $N$ or $d$. In this case, and throughout this thesis, we will however pay little attention to the dimensionality $d$, since it only affects distance calculations approximately linearly. In fact, Fränti and Sieranoja 2019 found that dimensionality has "no direct effect" on k-means. We therefore wish to investigate this plausible dependence on $K$ and possibly $N$, and assess the quality of the approximation by Formula (4). For this purpose, we perform a simulation study in which we evaluate the ability of UnifRandom to produce good delegation for datasets of varying sizes ($N$) and number of clusters ($K$). For each combination of $N$ and $K$, 10 datasets with $d = 2$ were generated, and the UnifRandom method was run 1000 times on each dataset. See Section 5 for a detailed explanation of the generation of the datasets. The results in Table 2 show the fraction of UnifRandom runs that succeeded in obtaining good delegation for all 10 datasets in each combination of $N$ and $K$.

The rightmost column shows the evaluation of Formula (4) for the respective number of clusters.

| K \N | 100 | 300 | 500 | 1000 | $\frac{K!}{K^K}$ |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0.5039 | 0.5003 | 0.5016 | 0.4988 | 0.5 |
| 3 | 0.2219 | 0.2146 | 0.2236 | 0.2234 | 0.2222 |
| 4 | 0.0895 | 0.0891 | 0.0941 | 0.0975 | 0.0938 |
| 5 | 0.0352 | 0.0397 | 0.0390 | 0.0365 | 0.0384 |
| 6 | 0.0157 | 0.0148 | 0.0170 | 0.0173 | 0.0154 |
| 7 | 0.0052 | 0.0055 | 0.0066 | 0.0070 | 0.0061 |
| 8 | 0.0026 | 0.0026 | 0.0026 | 0.0018 | 0.0024 |
| 9 | 0.0008 | 0.0011 | 0.0011 | 0.0010 | 0.0009 |
| 10 | 0.0004 | 0.0005 | 0.0003 | 0.0003 | 0.0004 |
| 11 | 0.0002 | 0.0002 | 0.0005 | 0.0001 | 0.0001 |

Table 2: Estimated and simulated probability of achieving good delegation for a variety of generated datasets.

In Table 2 we first clearly find that the number of points $N$ has no noticeable effect on the probability of achieving good delegation. We can also very clearly see that Formula (4) is a good approximation of the probability for UnifRandom to achieve good delegation, at least for small values of $K$. Indeed the accuracy of the approximation deteriorates around $K = 10$. This problem was also mentioned in Fränti and Sieranoja 2019, p. 104, who remarked that when $p$ is very small, the required number of repeats can be unreasonably high. Additionally, we can note that our statement in Section 4 regarding Formula (4) providing a lower bound for $p$ is not entirely supported by Table 2. Our statement would require that Formula (4) underestimate the proportions on each respective row. This is true for a small majority (60 %) of the observed proportions. However, the proportions are random realizations, and no additional effort has been made to quantify the variation in order to make a more robust statement. This warrants further investigation that is beyond the scope of this thesis.

Next, we wish to investigate the improvement provided by the R-Batched UnifRandom method. For this purpose we will simply run the full k-means algorithm on some real and simulated data, using the UnifRandom and R-Batched UnifRandom methods respectively. From these tests we report the performance measures in Table 3. The datasets are the two real world datasets: Iris and Seeds described in Section 5 below, and some simulated datasets with increasing number of clusters.

| Method | Perf. Measures | Iris | Seeds | (K=4) | (K=5) | (K=6) | (K=7) | (K=8) |
|---|---|---|---|---|---|---|---|---|
| UnifRandom | Accuracy (%) | 89.33 | 89.52 | 100.00 | 45.00 | 46.70 | 79.70 | 45.80 |
| | Final SSE | 78.94 | 588.92 | 3042.0 | $63 \cdot 10^6$ | $29 \cdot 10^6$ | $2 \cdot 10^6$ | $32 \cdot 10^6$ |
| | Seed SSE | 503.41 | 1269.02 | $156 \cdot 10^6$ | $182 \cdot 10^6$ | $58 \cdot 10^6$ | $65 \cdot 10^6$ | $38 \cdot 10^6$ |
| | Time (s) | 0.09 | 0.09 | 0.12 | 0.12 | 0.13 | 0.16 | 0.15 |
| | Iterations | 7.0 | 4.0 | 4.0 | 2.0 | 2.0 | 3.0 | 2.0 |
| R-Batched UnifRandom | Accuracy (%) | 89.33 | 89.52 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| | Final SSE | 78.94 | 587.32 | 3042.00 | 3048.68 | 3002.41 | 2969.46 | 3146.43 |
| | Seed SSE | 651.19 | 2154.64 | $322 \cdot 10^6$ | $101 \cdot 10^6$ | $35 \cdot 10^6$ | $34 \cdot 10^6$ | $17 \cdot 10^6$ |
| | Time (s) | 0.20 | 0.25 | 2.01 | 5.88 | 17.22 | 49.31 | 136.26 |
| | Iterations | 7.92 | 8.33 | 2.71 | 2.65 | 2.77 | 2.94 | 3.02 |
| | Rep. ($R$) | 12 | 12 | 31 | 77 | 193 | 489 | 1246 |

Table 3: UnifRandom vs R-Batched UnifRandom for datasets with increasing number of close and well-separated clusters. Note that the Iris and Seeds datasets are **not** well-separated.

In Table 3, we see clearly that the R-Batched UnifRandom method provides a much more reliable Accuracy than the regular UnifRandom method, at the cost of an extreme growth in computation time. This is further evidenced by the fact that the accuracies that the regular UnifRandom method achieved for the simulated datasets varied greatly over multiple generations of Table 3. This was not the case for the R-Batched UnifRandom method, which consistently achieved optimal accuracy. Although no extensive studies of the performance on data with closely placed clusters was done, we note that the regular UnifRandom method achieved generally higher accuracies on the Iris and Seeds datasets (with closely placed clusters) than for the datasets with well-separated clusters. This is likely because when clusters are closely placed and slightly overlapping, the k-means procedure is able to refine the clustering, which compensates for the regular UnifRandoms bad initialization.

For the second task, we again wish to test out the R-Batched UnifRandom method, but this time against other methods. In this case we will primarily dedicate our focus to simulated datasets with well-separated clusters, but we will also test the methods on the Iris and Seeds datasets with closely placed clusters. We test the three methods and report all performance measures described in Section 6 in Table 4 and in Figure 8 for the well-separated datasets, and in Table 5 for the Iris and Seeds datasets.

| Method | Perf. Measures | (K=4) | (K=5) | (K=6) | (K=7) | (K=8) | (K=9) |
|---|---|---|---|---|---|---|---|
| R-Batched | Accuracy (%) | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| UnifRandom | Final SSE | 3114.84 | 3024.29 | 2918.19 | 3000.57 | 2877.56 | 2855.21 |
| | Seed SSE | 6775.84 | $51 \cdot 10^6$ | $33 \cdot 10^6$ | $6 \cdot 10^6$ | $47 \cdot 10^6$ | $43 \cdot 10^6$ |
| | Time (s) | 1.90 | 5.60 | 16.44 | 49.10 | 136.18 | 351.84 |
| | Iterations | 2.58 | 2.88 | 2.73 | 2.69 | 2.90 | 2.94 |
| | Rep. (R) | 31 | 77 | 193 | 489 | 1246 | 3197 |
| k-means ++ | Accuracy (%) | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| | Final SSE | 3114.84 | 3024.29 | 2918.19 | 3000.57 | 2877.56 | 2855.21 |
| | Seed SSE | 9364.14 | 4807.11 | 7029.56 | 8123.27 | 6374.27 | 6313.33 |
| | Time (s) | 0.12 | 0.16 | 0.16 | 0.18 | 0.21 | 0.23 |
| | Iterations | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 |
| Kaufman | Accuracy (%) | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| | Final SSE | 3114.84 | 3024.29 | 2918.19 | 3000.57 | 2877.56 | 2855.21 |
| | Seed SSE | 6217.59 | 4448.96 | 5359.61 | 5625.68 | 5139.66 | 5465.50 |
| | Time (s) | 19.81 | 31.48 | 51.30 | 62.59 | 88.58 | 98.01 |
| | Iterations | 2.0 | 1.0 | 2.0 | 2.0 | 2.0 | 2.0 |

Table 4: Performance measures for all three initialization methods tested on simulated datasets with well-separated clusters. The largest difference between Equations (6) and (7) occurs when $K = 9$ where they evaluate to $\mathbb{P}_u \approx 0.11201$ and $\mathbb{P}_k \approx 0.10394$.

The results in Tables 4 and 5 and in Figure 8 give some insight into the considered methods. We first note that for each simulated dataset, all three initialization methods acquired the same Final SSE, and also 100 % accuracy, which indicates that all methods were able to correctly identify all clusters. This strongly motivates their applicability on this type of data. However, the methods report very different Seed SSE. The Kaufman method consistently provides the best seed SSE for all considered real world and simulated datasets. This is not surprising given that it attempts to place centres specifically such that they are surrounded by many points. Both the regular and R-Batched UnifRandom methods however produce enormous Seed Errors for the datasets with five or more clusters and consistently the largest Seed SSE. Celebi, Kingravi, and Vela 2013, p. 208 similarly found that deterministic (e.g. the Kaufman) methods produced the lowest Seed SSE, and that the UnifRandom produces high Seed SSE. The k-means++ method appears to produce Seed SSE that are slightly larger than those of the Kaufman method, but still not extreme. In fact, although k-means++ does not produce the best seeding, it is dramatically cheaper in computational expense than the Kaufman method, while still initializing the k-means algorithm appropriately. This brings up the fact that an initialization method does not need to be perfect, it only needs to provide a good-enough setup for the next method in line.

| Method | Perf. Measures | Iris | Seeds |
|---|---|---|---|
| R-Batched UnifRandom | Accuracy (%) | 89.33 | 89.52 |
| | Final SSE | 78.94 | 587.32 |
| | Seed SSE | 555.28 | 2243.31 |
| | Time (s) | 0.19 | 0.23 |
| | Iterations | 6.17 | 9.17 |
| | Rep. (R) | 12 | 12 |
| k-means ++ | Accuracy (%) | 89.33 | 89.05 |
| | Final SSE | 78.94 | 588.78 |
| | Seed SSE | 196.76 | 960.57 |
| | Time (s) | 0.07 | 0.08 |
| | Iterations | 7.0 | 6.0 |
| Kaufman | Acc. (%) | 88.67 | 89.05 |
| | Final SSE | 78.95 | 588.78 |
| | Seed SSE | 97.01 | 649.81 |
| | Time (s) | 0.36 | 0.79 |
| | Iterations | 10.0 | 3.0 |

Table 5: Performance measures for all three initialization methods tested on the Iris and Seeds datasets, both having closely placed clusters. The largest difference between Equations (6) and (7) occurs for the Iris dataset ($N = 150, K = 3$) where they evaluate to $\mathbb{P}_u \approx 0.33784$ and $\mathbb{P}_k \approx 0.32432$.

Seemingly, the k-means algorithm requires more iterations before convergence for the close cluster data than for the data with well-separated clusters. This is particularly noticeable in Table 3, although only two datasets with close clusters are included. This is also not surprising, since when clusters are far apart, they are likely to be isolated by the k-means procedure. This means that convergence is reached almost immediately. On the other hand, for data with closely placed clusters, the k-means algorithm is able to (and will) move around the centroids until it converges, which requires more iterations. This is why the reported number of iterations for the data with well-separated cluster tend to lie around 2, and why the k-means algorithm required up to around 8 iterations for the data with close clusters. Even so, the number of required iterations does not vary much.

For comparisons of the computational cost, we turn to computation time. In Figure 8 we present the reported computation times from Table 4. We see that the computation time of the Kaufman method grows approximately linearly with $K$, and for R-batched UnifRandom it grows approximately exponentially with $K$. Conversely, the k-means++ method has a much slower time complexity and therefore its computation time grows much slower than the other two methods. It is important to point out that we are now discussing time complexity in terms of $K$, not $N$.

# 8 Conclusions

We have presented the k-means algorithm along with three initialization methods, and we have argued for the R-Batched UnifRandom method as an improvement on the regular UnifRandom method in certain situations. What we have found is that the R-Batched UnifRandom method is indeed much more reliable (consistently producing optimal accuracy) than the regular UnifRandom method, particularly for simple data with well-separated clusters. However, the improvement comes with a dramatic growth in computation time as seen in Figure 8. The same reliability was reported by the Kaufman and k-means++ methods, both at significantly lower computational expenses. The Kaufman method unfortunately fails at least one of the requirements on initialization methods set out by Fränti and Sieranoja 2019, indicating that caution is advised if it is to be used in practical applications. The k-means++ method is on the other hand very suitable as an initialization method in many practical situations.

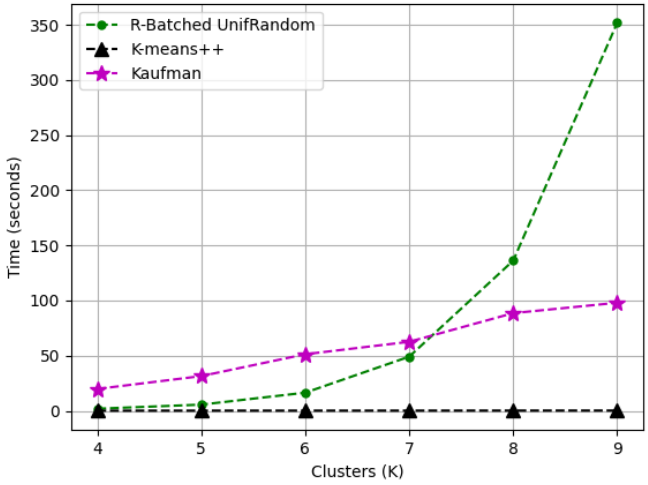


Figure 8: Computation time for all three initialization methods for the datasets with well-separated clusters analysed in Table 4.

Many recent contributions have been made on the topic of this thesis. For massive datasets, even the k-means++ method becomes computationally expensive. Bahmani et al. 2012 discussed this and presented the improved k-means|| method with favourable results. Capó, Pérez, and J. A. Lozano 2022 recently proposed the SM$K$-means algorithm which utilizes information from previous runs of the standard k-means algorithm to avoid falling into the same local minimum multiple times. Oti et al. 2020 propose the *modified* and *enhanced* k-means methods respectively.

# References


[1] David Arthur and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." In: *Symposium on Discrete Algorithms* 18 (2007), pp. 1027–1035.

[2] Bahman Bahmani et al. "Scalable k-means++". In: *arXiv preprint arXiv:1203.6402* (2012).

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2006.

[4] Marco Capó, Aritz Pérez, and Jose A. Lozano. "An efficient Split-Merge Re-Start for the K-means algorithm." In: *IEEE Transactions on Knowledge and Data Engineering* 34.4 (2022), pp. 1618–1627.

[5] M. Emre Celebi, Hassan A. Kingravi, and Patricio A. Vela. "A comparative study of efficient initialization methods for the k-means clustering algorithm". In: *Expert Systems with Applications* 40.1 (2013), pp. 200–210. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2012.07.021. URL: https://www.sciencedirect.com/science/article/pii/S0957417412008767.

[6] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[7] Pasi Fränti and Sami Sieranoja. "How much can k-means be improved by using better initialization and repeats?" In: *Pattern Recognition* 93 (2019), pp. 95–112. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2019.04.014. URL: https://www.sciencedirect.com/science/article/pii/S0031320319301608.

[8] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[9] Ji He et al. “Initialization of cluster refinement algorithms: a review and comparative study”. In: 1 (2004), pp. 297–302. DOI: 10.1109/IJCNN.2004.1379917.

[10] Institute of Agrophysics of the Polish Academy of Sciences in Lublin. 2012.

[11] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 1990.

[12] Shehroz S. Khan and Amir Ahmad. “Cluster center initialization algorithm for K-means clustering”. In: *Pattern Recognition Letters* 25.11 (2004), pp. 1293–1302. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2004.04.007. URL: https://www.sciencedirect.com/science/article/pii/S0167865504000996.

[13] Stuart Lloyd. “Least Squares Quantization in PCM.” In: *IEEE Transactions on Information Theory* 2.28 (1982), pp. 129–137.

[14] James B. Macqueen. “Some methods for classification and analysis of multivariate observations.” In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* 1.5? (1967), pp. 281–297.

[15] Eric U. Oti et al. “New k-means clustering methods that minimizes the total intra-cluster variance.” In: *African Journal of Mathematics and Statistics Studies* Vo:3.No:5 (2020), pp. 42–54.

[16] J.M. Peña, J.A. Lozano, and P. Larrañaga. “An empirical comparison of four initialization methods for the K-Means algorithm”. In: *Pattern Recognition Letters* 20 (1999), pp. 1027–1040.

[17] Douglas Steinley. “Local optima in K-means clustering: what you don’t know may hurt you.” In: *Psychological methods* 8.3 (2003), p. 294.

[18] Douglas Steinley and Michael J Brusco. “Initializing K-means batch clustering: A critical evaluation of several techniques”. In: *Journal of Classification* 24.1 (2007), pp. 99–121.