The k-means-u* algorithm: non-local jumps and greedy retries improve k-means++ clustering

Bernd Fritzke fritzke@web.de
Friedrichsdorf
61381 Germany

Editor:

Abstract

We present a randomized clustering algorithm called k-means-u* which in many cases is able to improve the clusterings found by the widely used k-means++ algorithm, frequently by a significant margin (i.e. several percent lower squared error). We initially introduce the k-means-u algorithm which - starting off from a clustering obtained by k-means++ - performs a non-local jump of the "least useful" center towards the center with largest summed error of its associated data points. A small random offset is applied to its new position. Afterwards regular k-means is performed. These two steps (non-local jump followed by k-means) are repeated as long as the error measure decreases strictly with each iteration. Once the error stops falling, the previous solution is returned. Occasionally k-means-u terminates despite a large remaining optimization potential but the randomized nature of the algorithm makes it possible to further enhance it by adding a greedy retry step. This helps in escaping local minima and frequently improves the solutions found. The resulting algorithm which we call k-means-u* dominates k-means-u which again dominates k-means++ w.r.t. solution quality. Empirical results for various data sets are provided and demonstrate a significant quality improvement of k-means-u* over k-means++ with only moderate additional computation.

Keywords: Clustering, Vector Quantization, Optimization

1. Introduction

In machine learning various sub-fields can be distinguished based on the amount of feedback given to the learner. On one side there is *supervised learning* where the given data consists of pairs of input and corresponding output. In this case the learner is trained to approximately reproduce this mapping with the expectation that it can afterwards generalize to unseen patterns. Typical examples of supervised learning are classification (e.g. image or speech recognition) and regression (e.g. time-series prediction). Considerably less information is given in *reinforcement learning* where a system is expected to learn how to perform long sequences of actions (e.g. in game play) while only receiving occasional feedback (e.g. "game won" or "game lost"). Finally there is *unsupervised learning* where a system is expected to learn purely from unlabeled data. It has been argued recently that due to the number of connections in the human brain and the life span of humans a large part of learning must be unsupervised (G. Hinton, AMA on Reddit.com, 2015). This paper is concerned with *clustering* which is considered to be a fundamental unsupervised learning problem.

In particular we propose a method to improve clusterings generated by the `k-means++` algorithm which currently can be seen as a de facto standard for clustering numerical data.

The remainder of this article is organized as follows: In section 2 the particular kind of clustering problem is described which the new algorithm (as well as `k-means` and `k-means++`) deal with.

In section 3 the classical `k-means` algorithm is introduced and illustrated by examples which also highlight its problematic dependency on initialization.

In section section 4 we describe the more recent `k-means++` algorithm which enhances `k-means` by a stepwise initialization which in general leads to much better results than e.g. the common random initialization from the data set.

Section 5 investigates why `k-means++` is so effective and shows for a simple example problem that under certain conditions `k-means++` always finds the optimal initial placement.

Section 6 gives examples of problems which are hard for `k-means++` in the sense that `k-means++` is in general not able to find a solution close to the optimum (which is known for these specific problems by construction). It is investigated why this is the case and that for certain examples `k-means++` has difficulties if k is larger than the number of clusters in the data.

In section 7 the `k-means-u` algorithm is introduced which considerably improves many solutions found by `k-means++` via a sequence of non-local jumps alternated with `k-means` phases.

In section 8 an occasional problem of `k-means-u` is described and illustrated by an example: too early termination due to a poor (but normally still better than `k-means++`) local minimum.

Section 9 introduces the `k-means-u*` algorithm which enhances `k-means-u` by allowing greedy retries. This allows in many cases to reach better local minima than with `k-means-u` alone.

In section 10 the results of systematic simulations with various data sets are presented covering large ranges of k for each data set. In a nutshell the conclusion is: w.r.t. solution quality `k-means-u*` dominates `k-means-u` which again dominates `k-means++`.

Section 11 provides a summary of the article.

2. Clustering

Clustering in general can be described as the problem of finding groups (clusters) in a data set such that the similarity of data items within a group is large and data items from different groups are dissimilar. The used measure of similarity can vary from a subjective perceptual criterion to precise mathematical definitions. In the context of this paper we choose the goal of clustering to be a computational one, in particular the minimization of a distance-based error function:

We assume an integer k and a set of n data points $\mathcal{X} \subset \mathbb{R}^d$. The goal is to select k centers $\mathcal{C}$ such that the error function

$$\phi(\mathcal{C}, \mathcal{X}) = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2 \tag{1}$$

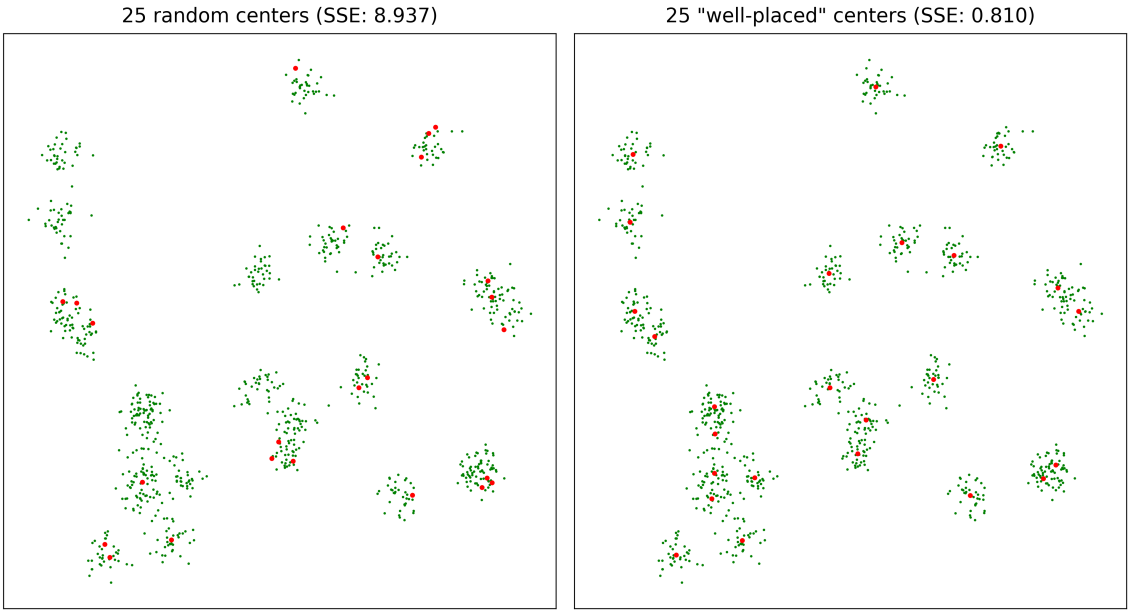


Figure 1: Illustration of a poor (left) and a good (right) distribution of centers for a two-dimensional data distribution generated from a mixture of equal-variant Gaussians. The Summed Squared Error (SSE=ϕ) is considerably smaller for the right distribution of centers.

is minimized. We thus strive to position the centers in such a way that the sum of squared distances between each data point and its respective closest center is minimized. In this paper we will refer to $\phi(\mathcal{C}, \mathcal{X})$ also as *Summed Squared Error* or SSE.

The resulting set of centers $\mathcal{C}$ can be used to group (cluster) the original data set but also to encode the data for transmission or storage in the sense of vector quantization. Finding the optimal solution for this problem is known to be NP-hard, so in practice we need to use approximation algorithms. In figure 1 an example of a data set is given with two clusterings which vary strongly with respect to the error criterion in equation (1).

3. `k-means`

`k-means` (Lloyd, 1982) is a classical method for clustering or vector quantization. Starting from an initial set of centers $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ a sequence of so-called *Lloyd Iterations* is performed. Each Lloyd iteration consists of two steps:

- determine for each center c_i its so-called *Voronoi Set* C_i, which is the set of data points for which c_i is the closest center:

$$C_i := \{x \in \mathcal{X} \mid \|x - c_i\| < \|x - c_j\| \, \forall j \neq i\} \tag{2}$$

- move all centers to the center of gravity of their Voronoi set:

$$c_i := \frac{1}{|C_i|} \sum_{x \in C_i} x$$

The complete `k-means` algorithm is specified in figure 2. It consists of an initialization ("Seeding") followed by a sequence of Lloyd iterations.

Seeding: Choose k initial centers $\mathcal{C} = \{c_1, c_2, ..., c_k\}$;
repeat /* Lloyd iterations */
  foreach $i \in \{1, ..., k\}$ **do**
    $C_i \leftarrow \{x \in \mathcal{X} \mid \|x - c_i\| < \|x - c_j\| \, \forall j \neq i\}$; `/* C_i is assigned the set of all points in X having c_i as their closest center */`
  end
  foreach $i \in \{1, ..., k\}$ **do**
    $c_i \leftarrow \frac{1}{|C_i|} \sum_{x \in C_i} x$; `/* modify c_i to be the center of mass of C_i */`
  end
until *no more change of* $\mathcal{C}$;
return $\mathcal{C}$;

Figure 2: The `k-means` algorithm

A common seeding method is to select the initial centers equiprobable at random from the data set $\mathcal{X}$. This approach ensures that there are no unused centers ("dead units") which could occur if centers were selected from random locations not contained in X.

Possible ties in defining C_i can be resolved arbitrarily. If a deterministic resolution method is used, the whole algorithm is deterministic (after the seeding) and leads to reproducible results. In summary the `k-means` algorithm performs one Lloyd iteration after another as long as the SSE decreases with each iteration[1].

`k-means` in this form is very simple and known to converge in a finite number of steps. The quality of its solutions measured by the above error function ϕ can however vary strongly depending on the seeding used. Let us illustrate this by applying `k-means` to the data set A shown in figure 3 (left). It consists of 36 quadratic clusters. Let us specifically consider the problem to distribute 36 centers over A, i.e. the same number as there are clusters in A. In the following we denote this particular problem as A-1 (variations of this problem where the number of centers is n times as large as the number of clusters in A will accordingly be denoted as A-n).

Due to the regular structure of the data set and the fact that the number of centers is the same as the number of clusters it is obvious in this case what the optimal arrangement of centers is: One center in each cluster, positioned in the middle of the cluster (see figure 3, right side). The corresponding summed squared Error (SSE) for this optimal solution is 1.458 (numerically computed).

Let us now apply `k-means` with equiprobable random initialization from the given data set A to this problem. In figure 4 a typical solution is shown. It has a SSE of 3.883 which is 166% percent above the known optimum (so this solution can be considered rather poor).

A rather obvious way to achieve better results is to run `k-means` several times with different random seedings and return the best result of all runs. However, even if we run it

1. An alternative stopping criterion is to terminate as soon as the error improvement after any Lloyd iteration falls below a threshold.

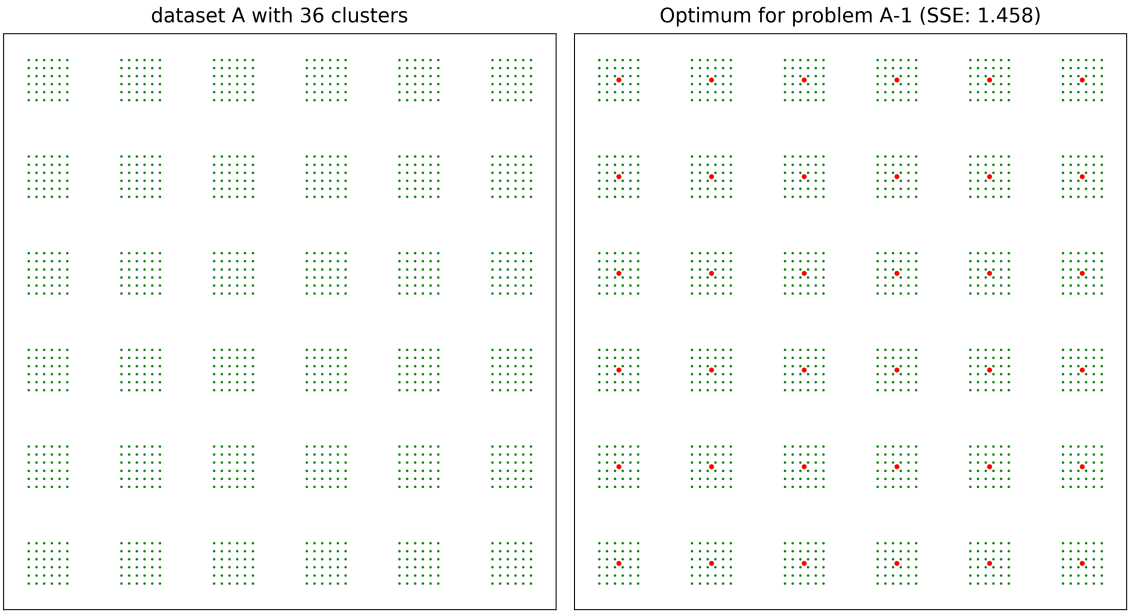


Figure 3: left: Data set A consisting of 36 identically-shaped clusters. We denote with problem A-1 the task to distribute 36 centers such that the SSE wrt. A is minimized. Right: Optimal solution for problem A-1: one center positioned in each cluster, SSE: 1.458

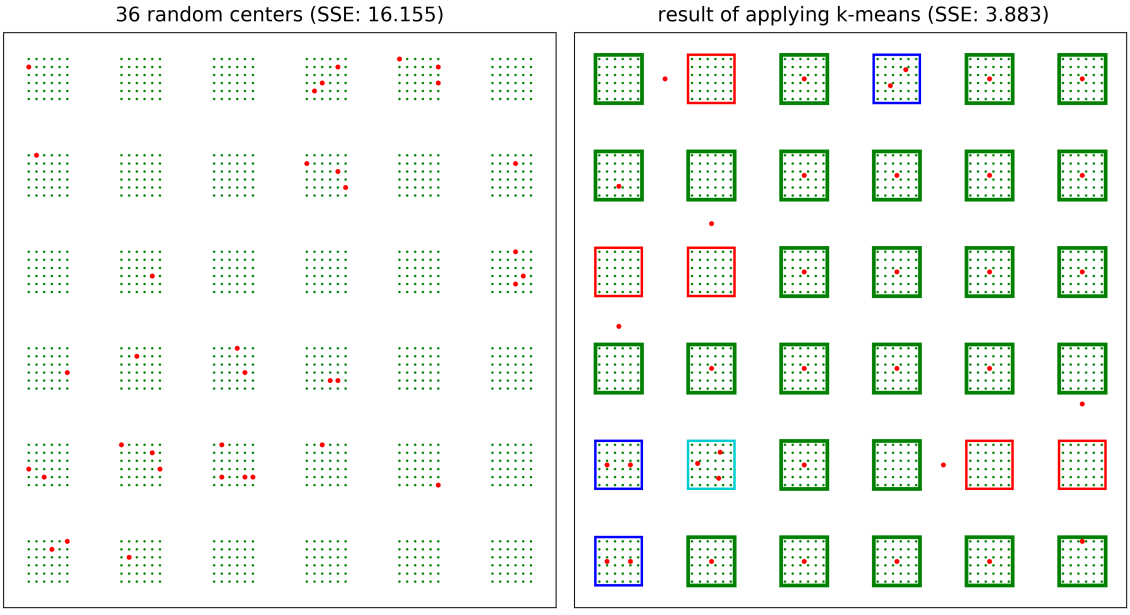


Figure 4: `k-means` approximately solves clustering problem A-1. Left: $k = 36$ centers have been chosen at random from A. Right: Result of `k-means` applied to the initial configuration shown in the left figure. Many clusters correctly receive one center (optimal, coded green), some clusters however gets 2 or 3 centers (too many, coded dark and light blue) and some clusters do not get any center at all (too few, coded red)

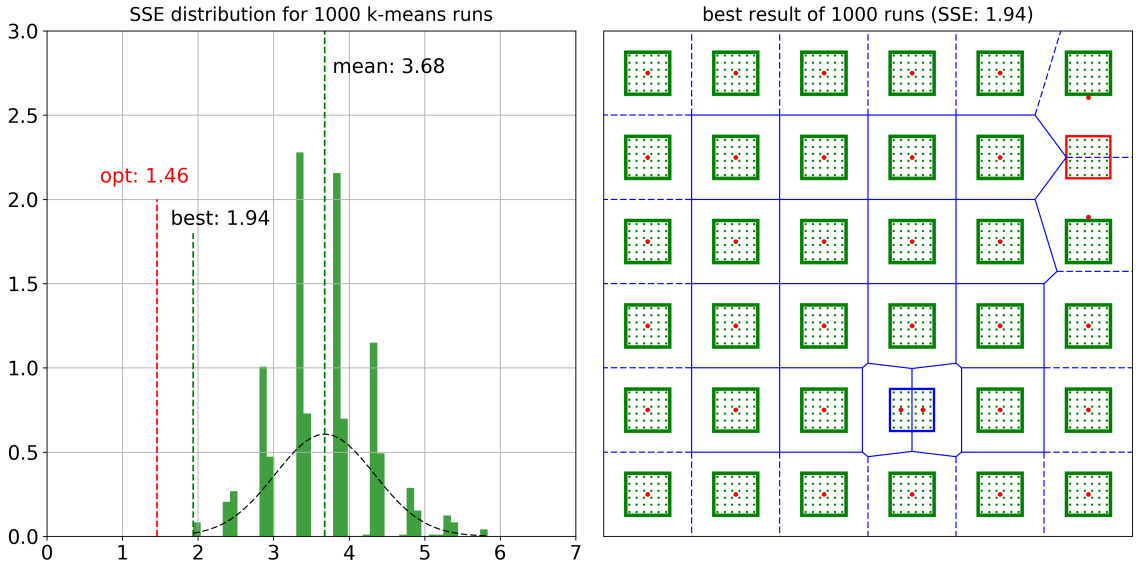


Figure 5: left: normalized distribution of SSE values for 1000 runs, optimal solution and Gaussian fitted to the measured errors. right: best result obtained during the 1000 runs. The SSE of this solution is still 32.9% higher than that of the known optimum.

1000 times, the best results are usually still far from the optimum (see figure 5). If we model the distribution of SSE values obtained from the simulation with a normal distribution, the optimal SSE is more than 3.4σ away from the mean indicating an occurrence probability of less than 0.0003. Even if it is not clear how well a normal distribution can model the `k-means` SSE distribution, it seems that a brute-force method based on a large number of random seedings is a costly way to find a low-SSE configuration with `k-means`.

4. `k-means++`

25 years after the original `k-means` algorithm an improvement was proposed by Arthur and Vassilvitskii (2007), the `k-means++` algorithm which today can be seen as a standard way of doing `k-means` (one specific indication being that the default implementation of `k-means` in the popular `scikit-learn` package (Pedregosa and Varoquaux, 2011) for scientific computation *is* `k-means++`). The `k-means++` algorithm augments `k-means` by a stepwise seeding phase which takes into account the distance of the data points from the centers placed so far. Specifically the probability that a data point x is selected as the position of the next center is chosen to be proportional to the minimum squared distance to any of the already placed centers. In contrast to choosing the next center equiprobable from the remaining data points this method favors regions which are far from the existing centers. Placing the next center in such a region likely results is a large reduction of the overall error. The complete algorithm is depicted in figure 6.

Notable details in the `scikit-learn` implementation of `k-means++` are that the algorithm is always run several times (default: 10) and only the best result is returned. Moreover, when placing new centers during seeding several candidates are tried out and

Initialization: $\mathcal{C} \leftarrow \{c_1\}$ with c_1, chosen uniformly at random from $\mathcal{X}$;
$i \leftarrow 1$;
while $i < k$ **do** /* choose next center */
  $i \leftarrow i + 1$;
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{x\}$ with x drawn at random from $\mathcal{X}$ with probability

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

  whereby $D(x) = \min_{c \in \mathcal{C}} \|c - x\|$
end
Perform `k-means` using $\mathcal{C}$ as the initial set of centers;
return *resulting value of* $\mathcal{C}$;

Figure 6: The `k-means++` algorithm

the one reducing the overall error most is taken. Since no attempt is made to backtrack this can be seen as a kind of "greedy" algorithm and is already described in Arthur and Vassilvitskii (2007) without further analysis but the remark, that "it helps".

In figure 7 the way `k-means++` works, is illustrated. One can see there how the distance-based probability guides the selection of the respective next center.

`k-means++` is often very effective in finding good seedings for the following `k-means` phase. According to Arthur and Vassilvitskii (2007) (but also in accordance with our own experiments) the resulting solutions are mostly significantly better than those obtained by equiprobable random seeding from the data set. There is even a proven lower bound for the quality of the solution $\mathcal{C}$ constructed by `k-means` compared to the optimal solution:

$$E[\phi] \leq 8(\ln k + 2)\phi_{\mathrm{OPT}} \tag{3}$$

For practical purposes this bound may often not be tight enough (for k=100, e.g., the bound guarantees that the error of the solution found differs from the optimum by not more than a factor of 52.8), but the proof of the bound as such is quite remarkable since there is no bound at all for the solutions of `k-means`, i.e. they can be arbitrarily poor.

To illustrate the high quality of `k-means++` we show - after having performed 400 simulations several times - the two most frequent results for the problem A-1 where the original `k-means` with random seeding had problems: In 48-50% of the cases the optimal solution was found (figure 3). Also in 48-50% of the cases one cluster remained empty which is actually very good given that for `k-means` this was the best solution resulting from 1000 runs and did only occur in about 0.3% of the simulations (figure 5). Rarely (0-4%) two clusters remained empty. One should note here that In every simulation run we performed `k-means++` exactly once. As mentioned above the implementation of `k-means++` in `scikit-learn` is configured such that per default 10 runs are performed and the best result is returned. Given the described probability of about 50% for finding the optimal solution of A-1, it is

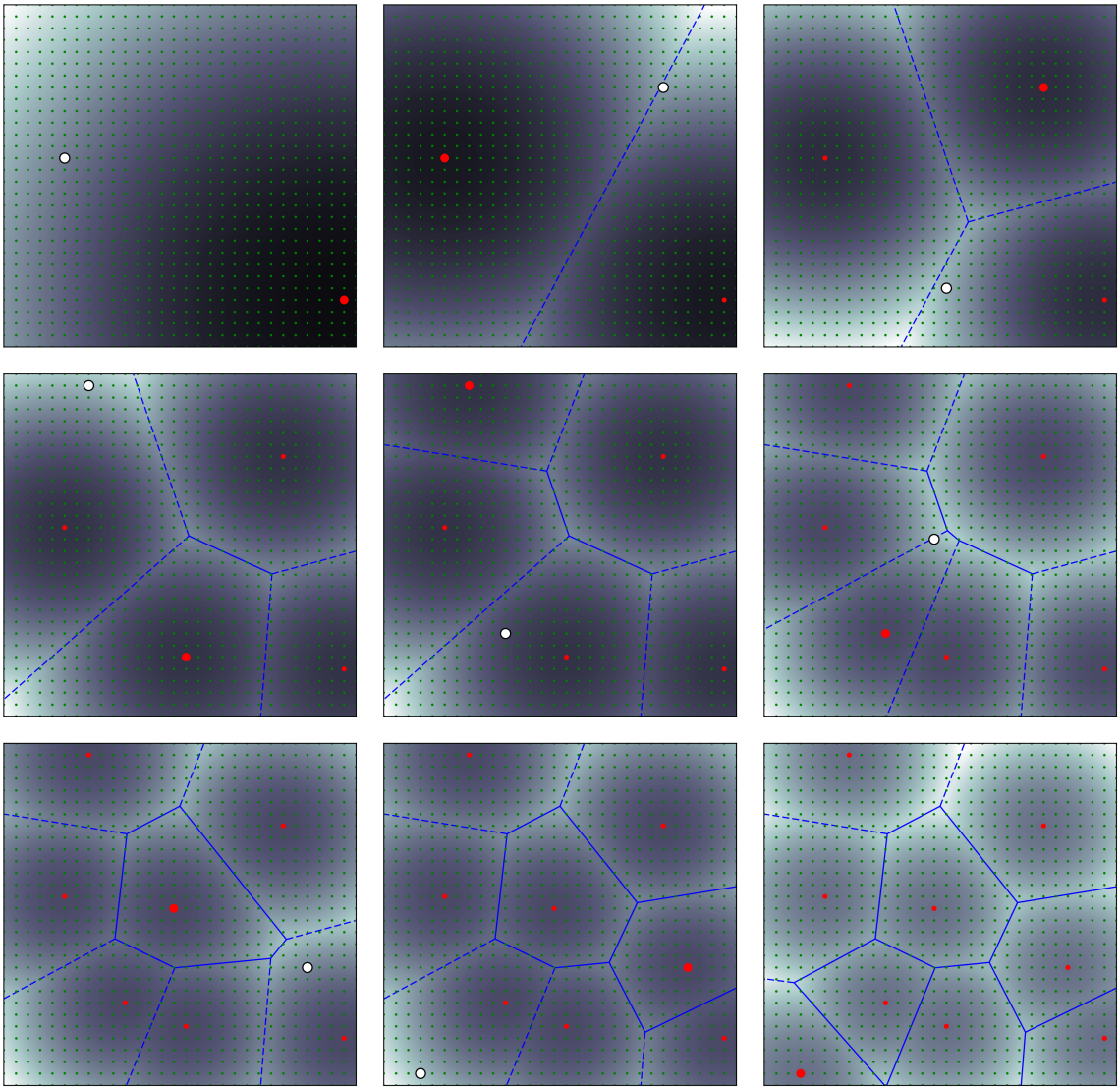


Figure 7: `k-means++` seeding example: the data consists of 900 points arranged in a 30x30 grid. The sequence of images shows the insertion of $k = 9$ centers depicted as red circles. The most recently inserted center is always enlarged. The shading indicates the squared distance of the data points (dark=low, light=high) from the current set of centers and therefore also the probability that `k-means++` choses the next center from that location. The blue segments show the Voronoi diagram corresponding to the current set of centers. The Voronoi segments coincide with light-colored regions since they are defined by neighboring centers to which they have equal (and thus large) distance at each position. The white dot indicates the position of the next center chosen. It is noticeable that the white dot usually lies in a region which is colored lightly (indicating high distance resp. probability).

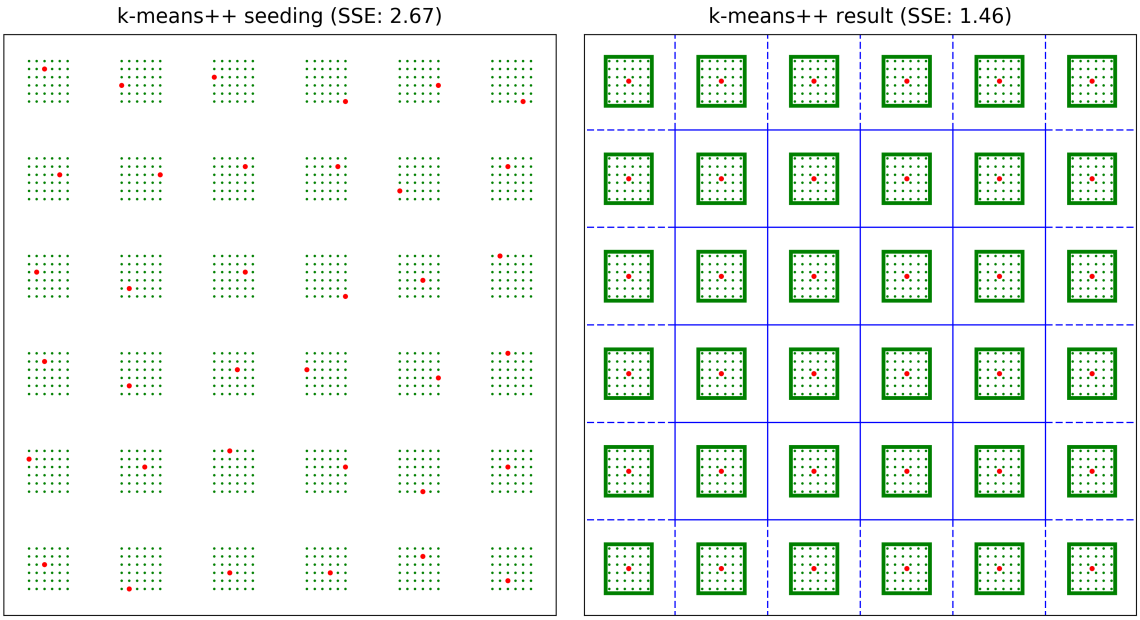


Figure 8: `k-means++` optimally solves a given clustering problem A-1: 6x6 clusters of 6x6 data points each, k=36. All clusters correctly receive one center (coded green). In 48-50% of the simulations this optimal solution was found. Equally often solutions similar to figure 9 were found with one empty cluster. In such situations it helps to perform several runs of `k-means++` as done per default in `scikit-learn` (see text)

trivial to see that the `scikit-learn` implementation would find the optimal solution with about $P(opt) = 1 - (\frac{1}{2})^{10} = 0.999$.

5. Why does `k-means++` work so well?

Why does `k-means++` work so well? Let us analyze this for the case of a data distribution in 1-D. The calculations are readily generalized to the higher-dimensional case.

Let us consider a data set A^1 consisting of $n = g * h$ data point in 1-D space distributed in g separate regions of high density (see figure 10). Each region has a length of a and contains $h = n/g$ points, equally distanced. The distance between neighboring high density regions is $a\eta$.

Let us first consider a cluster represented by one center which is optimally placed in the center of the cluster. What is the sum of distances in this cluster? If we let n grow towards infinity we can - instead of computing a discrete sum of distances for all points in the cluster - compute the following integral expression:

$$F_1 = 2 \int_0^{\frac{a}{2}} x^2 \, dx = \frac{a^3}{12} \tag{4}$$

F_1 is proportional to the mean square distance in the cluster and to the cluster width a.

For a cluster not yet covered by a center, all points have distances of at least $a\eta$ to the nearest center. Let us further assume that η is so large that we can neglect the difference

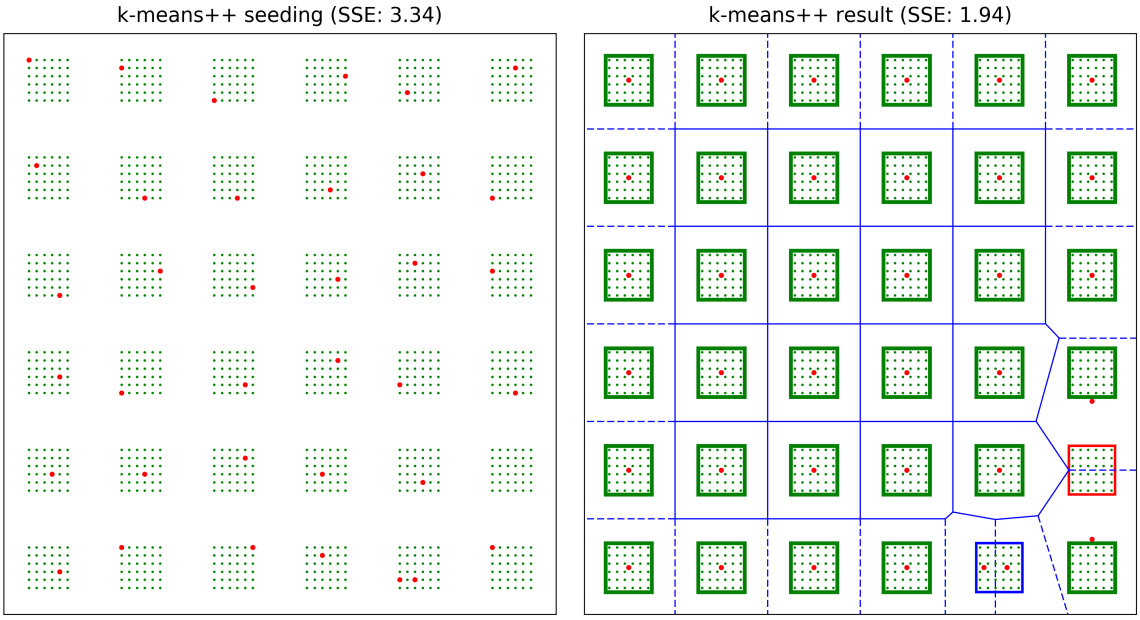


Figure 9: `k-means++` near-optimally solves a given clustering problem A-1: 6x6 clusters of 6x6 data points each, k=36. 34 clusters correctly receive one center (optimal, coded green), one cluster gets 2 centers (too many, coded in shades of blue) and one cluster gets no center at all (too little, coded red). Such a solution was found in 48-50% of our simulations. Equally often the optimal solution was found as shown in figure 3. Looking at this particular simulation one can get an idea why `k-means++` failed to find the optimum here: In the seeding phase one cluster remained empty (the one colored red on the right-hand figure). Let us denote this cluster as E. The reason that E remained empty probably was that the centers placed in the clusters directly above and below E both were positioned quite close to E leading to relatively small probabilities for the data points in E to be selected as the next center during the `k-means++` seeding phase.

Figure 10: One-dimensional data set A^1 with g clusters

in distance of points within that cluster and assume all points to have a distance of exactly $a\eta$ to the nearest cluster center. Therefore, the term corresponding to equation (4) for the squared distances is

$$F_2 > 2 * \int_0^{\frac{a}{2}} (a\eta)^2 \, dx = a^3\eta^2 \tag{5}$$

F_2 is proportional to the mean square distance of points in this uncovered cluster to the nearest center and proportional to the cluster width a.

If we assume that $i,\ 1 < i < g$ clusters are already covered with one center each, the probability $P_f(i)$ that the next center is placed in one of these "false" clusters can be conservatively estimated as

$$P_f(i) < \frac{i\,F_1}{i\,F_1 + (g-i)\,F_2} = \frac{i\frac{a^3}{12}}{i\frac{a^3}{12} + (g-i)a^3\eta^2} = \frac{1}{1 + 12(g/i - 1)\eta^2} = c\frac{1}{\eta^2} \tag{6}$$

Therefore, for any fixed i the following holds:

$$\lim_{\eta\to\infty} P_f(i) = 0$$

In other words, if the ratio η of inter-cluster distance and intra-cluster distance keeps growing the probability of a "wrong" seeding vanishes. Accordingly `k-means++` will almost always place the first k centers optimally in the assumed scenario (data set $\mathcal{X}$ consisting of k well-separated clusters of equal size).

The above explains - for our data set $A1$ with a number of well-separated clusters of similar size - the effectiveness of `k-means++` for the case that k is no larger that the number of clusters in A. A similar argument can be made for two two-dimensional data sets A (see figure 4) or higher-dimensional versions of it. In all cases a growing inter-cluster distance leads to dominating positioning probabilities in so far uncovered clusters.

Perhaps surprisingly the situation changes a lot if one tries to position more centers than there are clusters in the given data set which actually is a typical scenario in applied data analytics where the number of true clusters is usually not known in advance.

6. Clustering problems which are hard for `k-means++`

Consider again the data set A from figure 3 with 36 high-density regions. The corresponding clustering problem A-1 with $k = 36$ has been well solved by `k-means++`. Let us now consider the problem A-4 where the task is to distribute $k = 144$ centers over A (i.e. 4 times as many as there are clusters in A). The structure of this data set makes it obvious that for the optimal solution one should place exactly four centers in each cluster. In figure 11 you find a typical result from `k-means++`.

It can be seen that for many clusters the number of centers placed in them differs from the optimal value (4 in this case). Why is it that `k-means++` can reliably place centers for problem A-1 but not so well for problem A-4?

As defined in section 4 `k-means++` uses the (normalized) distance of a data point x from existing centers as probability that x will be the next center. If the number g of clusters is larger than the number n of centers placed so far, this ensures that data points in uncovered clusters have a much larger probability of getting chosen due to their large squared distance

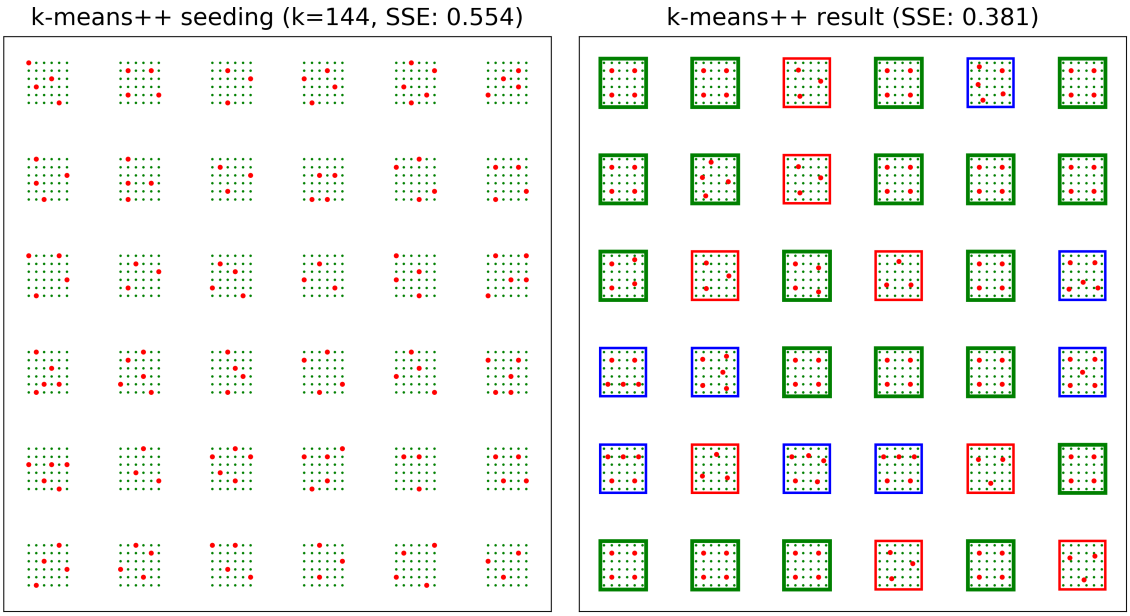


Figure 11: `k-means++` approximately solves clustering problem A-4: the data is the same as for A-1 (see figures 4 and 9), but in this case k=36*4=144 is chosen. The optimal number of centers per cluster is 4 for this problem. Left: Initialization found by `k-means++`. 14 of 36 clusters do not contain the optimal number of centers. Right: Final result. The centers have been locally re-distributed by `k-means`, but the number of centers in each particular cluster has not changed.

to the nearest center. However, if $k > g$ then `k-means++` initially positions one center in each cluster and suddenly the situation changes. Now every data point has a center in its relative vicinity. If further centers are placed they necessarily end up in a cluster already covered by one (later possibly several) centers. Accordingly, they reduce the local error only moderately. This leads to a much higher chance that `k-means++` positions a center sub-optimally and usually leads to improvable results (compared to the optimal distribution of centers, which is unknown in most cases).

In the following we analyze this behavior for the case of the one-dimensional signal distribution A^1 already used in section 5. This data set consists of $n = g * h$ data point in 1-D space distributed in g separate regions of high density (see figure 10). Each region has a length of a and contains $h = n/g$ points, equally distanced. The distance between neighboring high density regions is $a\eta$. We like to investigate the problem to place $2 * g$ centers such that the SSE is minimized. It is easy to see that the optimal solution for this problem consists of 2 centers per cluster placed in the centroid of the first and second half of each cluster. How likely is it that `k-means++` does find this configuration?

If we assume η to be large we can - according to the analysis in section 5 assume that each of the first g centers will be placed in a different cluster. The next center will necessarily be placed in one of the existing clusters. This cluster then has two centers which is correct

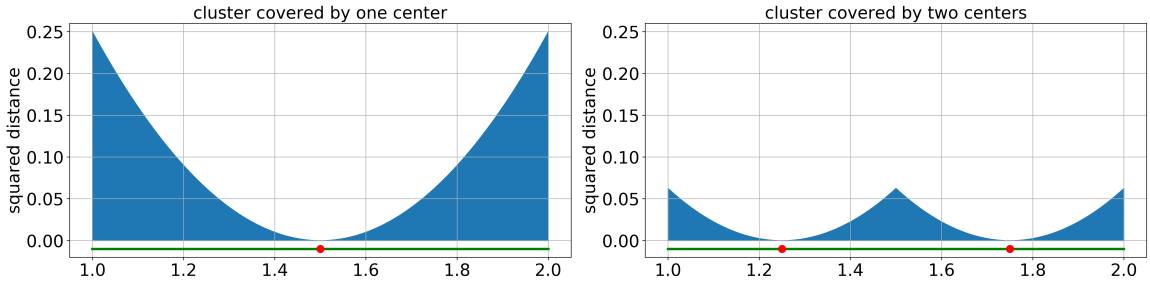


Figure 12: Squared distance within a cluster of length $a = 1$. Left: cluster covered by one center. Right: cluster covered by two centers

at that point. How likely however is it, that also the remaining $g-1$ centers will be placed correctly, i.e. such that no cluster contains more than 2 centers?

For simplicity we will assume that in a cluster with one center this center is placed in the centroid of the cluster and that in a cluster with 2 centers these centers are already placed to minimize the SSE in this cluster, i.e. each in the centroid of one half of the cluster. As in section 5 we will consider the continuous case by interpreting each cluster as finite segment of length a and we use integrals instead of discrete sums to compare error values for different configurations.

Let us first compute the integral F_1 of the squared distance for a cluster of length a covered by one single center positioned in the middle of the cluster (see figure 12, left side):

$$F_1 = 2\int_0^{\frac{a}{2}} x^2 dx = 2\left[\frac{1}{3}x^3\right]_0^{\frac{a}{2}} = 2 * \frac{a^3}{24} = \frac{a^3}{12} = c \quad \text{(for some } c\text{)} \tag{7}$$

If a cluster is covered by two centers we assume that they are optimally positioned at 25% and 75% of its length (see figure 12, right side). The corresponding integral F_2 is the following:

$$F_2 = 4\int_0^{\frac{a}{4}} x^2 dx = 4\left[\frac{1}{3}x^3\right]_0^{\frac{a}{4}} = 4 * \frac{a^3}{192} = \frac{a^3}{48} = \frac{c}{4} \tag{8}$$

As defined in section 4 `k-means++` uses the distances as probabilities for placing further centers so F_1 is proportional to the probability that a new center is placed in a particular cluster with one center and F_2 is proportional to the probability that a new center is placed in a particular cluster with two centers. Since for the following only the relative sizes of F_1 and F_2 are needed, we can replace F_1 by c and F_2 by $\frac{c}{4}$.

Let us assume that $g+i$ centers have been placed correctly by `k-means++` among the g clusters (this means that i clusters now have 2 centers). How probable is it that the next center will be placed correctly as well, i.e. in a cluster having only one center so far? To compute this we have to compare the probabilities of the $(g-i)$ "correct" cases to those of all cases:

$$P_{corr}(i,g) = \frac{c*(g-i)}{c*(g-i)+\frac{c}{4}*n} = \frac{g-i}{g-\frac{3}{4}i} \tag{9}$$

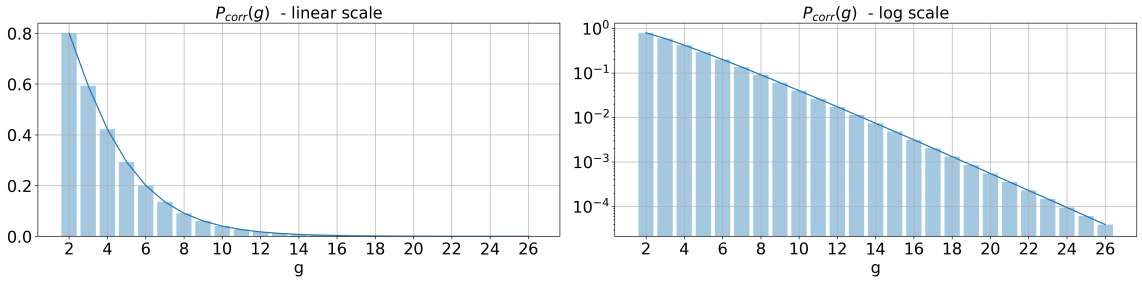


Figure 13: Probability $P_{corr}(g)$ that `k-means++` correctly places centers g+1 to 2*g for the clustering problem A^1-2 (position $2 * g$ centers while the data set is 1-D and has g clusters as shown in figure 10)

To compute the probability that all of the $2 * g$ centers are placed optimally (assuming $g + 1$ centers were placed correctly already) we have to compute the product of values $P_{corr}(i, g)$ for $i \in \{1, ..., g - 1\}$:

$$P_{corr}(g) = \prod_{i=1}^{g-1} P_{corr}(i, g) = \prod_{i=1}^{g-1} \frac{g - i}{g - \frac{3}{4}i} \tag{10}$$

$P_{corr}(g)$ decays exponentially quickly with g. In figure 13 the values of $P_{corr}(g)$ are graphically displayed and already $P_{corr}(24)$ is less than 10^{-4}. This means that w.r.t. data set A^1 - a one-dimensional data set with g clusters - `k-means++` is very unlikely to position the $2 * g$ centers such that the optimal configuration with 2 centers per clusters is achieved. This is the case for all values of g which are not trivially small.

This result is in sharp contrast to the result for the seemingly similar problem to distribute exactly g centers for data sets like A^1. In this case by choosing a large distance ηa among the clusters it can be made arbitrarily probable that `k-means++` finds the optimal configuration (one center in each cluster).

Once there is a center in each cluster, however, the inter-cluster distance is rendered meaningless. A further center in any cluster reduces the local error only moderately (e.g. by a factor of 4 if we add a second center to a cluster having one center so far, see figure 12). This again means that if we already have many clusters with two centers, it becomes very probable that a third center is placed in one of them by `k-means++` leading to a sub-optimal configuration.

The analyzed behavior for 1-D could be extended to other data distribution and to higher data dimensions. Completely general statements however are probably difficult to establish since there is a large dependency on the particular structure of the given data set which for real data sets is usually not known. Therefore we will rather concentrate on presenting a method to improve upon the results of `k-means++` and demonstrate its effectiveness by systematic comparative simulations with data sets of varying size and dimensionality.

7. k-means-u

How can the results obtained with `k-means++` be further improved? `k-means++` employs a careful seeding step followed by standard `k-means` and thus ends in a (usually local) minimum of the error function $\phi(\mathcal{C}, \mathcal{X})$ from equation (1). One idea to improve such a solution is to move single centers to other positions and afterwards let `k-means` find a - hopefully better - local minimum. To make this computationally efficient and guarantee convergence, however, one needs to carefully select both the centers to be moved and their respective target positions. Also a stopping criterion needs to be defined.

Which center should we move? Following the approach proposed by Fritzke (1997) we investigate how "useful" each center is for error reduction. This can be quantified by removing this center and comparing the resulting error with the current error. Thus we define the *utility* $U(c_i)$ of a center c_i as

$$U(c_i) = \phi(\mathcal{C} \setminus \{c_i\}, \mathcal{X}) - \phi(\mathcal{C}, \mathcal{X}) \tag{11}$$

The utility of a center is a measure how much we need this center or how easy its "task" of reducing error can be overtaken by neighboring centers. For example in the pathological case where two centers have the same position they both have a utility of zero since one could remove either one of them without increasing the error: the remaining center would cover the associated data points as well as the two centers did before. This case will not occur with `k-means` but all other things being equal one can expect the utility of centers with close neighbors being lower than those of more isolated centers.

We now define the "least useful" center λ as follows:

$$\lambda = \arg\min_{c_i \in \mathcal{C}} U(c_i) = \arg\min_{c_i \in \mathcal{C}} \phi(\mathcal{C} \setminus \{c_i\}, \mathcal{X}) = \arg\min_{c_i \in \mathcal{C}} \sum_{x \in \mathcal{X}} \min_{c_j \in \mathcal{C} \setminus \{c_i\}} ||x - c_j||^2 \tag{12}$$

Where should we move the center λ? Since our goal is to reduce the overall error function one rather straightforward approach is to move λ to the vicinity of that center μ currently having the maximal error sum for its Voronoi set.

Recall that in section 3, equation (2) we defined for each $i \in \{1, ..., k\}$ the Voronoi set C_i to be the set of all points in $\mathcal{X}$ that are closer to c_i than they are to c_j for all $j \neq i$. We now define μ as the center c_i having the largest summed squared distance over its Voronoi set C_i:

$$\mu = \arg\max_{c_i \in \mathcal{C}} \sum_{x \in C_i} ||x - c_i||^2 \tag{13}$$

It is advisable to place λ not exactly at the position of μ since that would cause all points in C_μ to have identical distances to μ and λ. We therefore place center λ at the position of μ plus some small random offset. A simple approach to define what "small" means for a given center μ is to consider the mean distance of the data points in μ's Voronoi set C_μ. We therefore define d_μ as

$$d_\mu = \sqrt{\frac{1}{|C_\mu|} \sum_{x \in C_\mu} \|x - \mu\|^2} \tag{14}$$

d_μ gives an indication of the spatial extension of μ's Voronoi set. By choosing a small fraction of d_μ, e.g. ϵd_μ with $\epsilon = 0.01$, as the length of our offset vector we can be confident that the new position of λ will be "near" μ.

Having a length, we still have to choose a direction for our offset vector. A principled and informed choice would be the direction of largest variance in the Voronoi set of μ, i.e. the unit eigenvector corresponding to the largest eigenvalue of the covariance matrix of C_μ.

Instead - both to have some non-determinism and to save the eigenvector computation in each step - we simply use a random vector from the d-dimensional unit hypersphere and rely on the following `k-means` phase to find good configurations. Choosing such a vector with uniform probability density is not completely trivial, however. For example normalizing a random vector from the d-dimensional hypercube would cause probability peaks in the directions of the corners of this hypercube. But it has been shown (Marsaglia, 1972) that a uniformly distributed unit random vector from the d-dimensional unit hypersphere can be constructed as follows:

1. Generate d Gaussian random variables x_1, x_2, ..., x_d
2. Return the vector
$$\xi = \frac{1}{\sqrt{x_1^2,\, x_2^2,\, ...,\, x_d^2}} [x_1,\, x_2,\, ...,\, x_d]^T$$

Let u be such a random vector. Then we define our offset vector o as

$$o = \epsilon\, d_\mu\, u \tag{15}$$

The `k-means-u` algorithm which we now define, starts with `k-means++`. Thereafter repeatedly the centers μ and λ are determined and the least useful center λ is moved to the position of the center μ with maximum error (plus a small random offset o which is also applied to μ itself, but with opposite sign). After each such move standard `k-means` is performed and the resulting error is measured. The algorithm terminates as soon as there is no improvement of the error measure. The complete `k-means-u` (`k-means` with utility) algorithm is specified in figure 14.

What does locally happen during a jump in `k-means-u`? An example is shown in figure 15. At the previous position of λ the data points so far belonging to λ's Voronoi set are now partitioned among the neighboring centers whose Voronoi regions are correspondingly enlarged. Assuming that o is very small the centers μ and λ now both have positions very close to the previous position of μ but offset in opposite directions. The previous Voronoi set of μ is therefore partitioned into two subsets C_μ and C_λ divided by a $(d-1)$-dimensional hyperplane. This hyper plane contains the previous position of μ and is also a normal plane to the offset vector o. Since o is created from a random vector, the orientation of this hyperplane and the resulting partitioning is random as well. This observation will become relevant in section 9.

In figure 16 a simple one-dimensional example illustrates typical results of `k-means`, `k-means++` and `k-means-u`. The data set used is a variant of data set A^1 which was used in section 5 and section 6 to illustrate easy and hard problems for `k-means++`.

Figure 17 contains a 2-dimensional example where `k-means-u` found a considerable improvement over the result of `k-means++`. In figure 18 the sequence of non-local jumps

Seeding: Perform `k-means++`;
$\phi_{\text{best}} \leftarrow \phi(\mathcal{C}, \mathcal{X})$; `/* store lowest error so far (from k-means++) */`
$\mathcal{C}_{\text{best}} \leftarrow \mathcal{C}$; `/* store best` $\mathcal{C}$ `so far (from k-means++) */`
Loop
  $\lambda \leftarrow \arg\min_{c_i \in \mathcal{C}} \phi(\mathcal{C} \setminus \{c_i\}, \mathcal{X})$; `/* find least useful center */`
  $\mu \leftarrow \arg\max_{c_i \in \mathcal{C}} \sum_{x \in C_i} ||x - c_i||^2$; `/* find center with max. local error */`
  $u \leftarrow$ (random vector from d-dimensional unit hypersphere);
  $d_\mu \leftarrow \sqrt{\frac{1}{|C_\mu|} \sum_{x \in C_\mu} \|x - \mu\|^2}$; `/* mean distance around` μ `*/`
  $o \leftarrow \epsilon\, d_\mu\, u$; `/* offset vector,` $\epsilon = 0.01$ `*/`
  $\lambda \leftarrow (\mu + o)$; `/* position` λ `near` μ `*/`
  $\mu \leftarrow (\mu - o)$; `/* position` μ `opposite to` λ `w.r.t. old` μ `value */`
  Perform `k-means` using the current $\mathcal{C}$ as initial set of centers;
  if $\phi(\mathcal{C}, \mathcal{X}) < \phi_{\text{best}}$ **then**
    $\phi_{\text{best}} \leftarrow \phi(\mathcal{C}, \mathcal{X})$; `/* store new lowest error */`
    $\mathcal{C}_{\text{best}} \leftarrow \mathcal{C}$; `/* store new best` $\mathcal{C}$ `*/`
  else
    break ; `/* exit loop */`
  end
EndLoop
return $\mathcal{C}_{\text{best}}$;

Figure 14: The `k-means-u` algorithm

leading to the result in figure 17 is displayed. Figure 19 demonstrates that `k-means-u` is also able to find improvements for more natural data sets. In this case the data is from a mixture of overlapping Gaussians. The number of data points and the number of centers is exactly as in figure 17. The improvement over `k-means++` is nearly 4% in this example, even though it is hard to see the difference between the two solutions.

Note: In principle also another seeding method than `k-means++` can be used (e.g. standard `k-means` or even random seeding) but in simulations `k-means++` led to the best results for the following `k-means-u` algorithm.

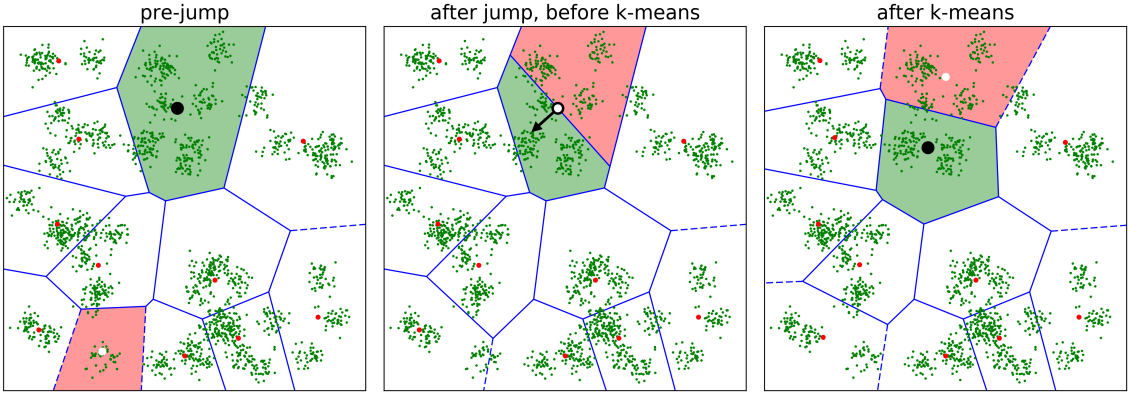


Figure 15: Detail view of a jump. The left figure shows a configuration where the initial run of `k-means++` (the first phase of `k-means-u`) has converged. The center μ with largest local summed squared error (SSE) is shown black (with a green Voronoi region). The center λ with the lowest utility is shown in white (with a reddish Voronoi region). The middle figure shows the configuration directly after performing a jump from λ to μ but before applying `k-means`. A small random offset vector has been applied to both μ and λ but in opposite direction. The direction of the offset vector is shown as an arrow. The offset vector is a normal vector of the $(d-1)$-dimensional hyperplane (line for 2-D data) which divides the previous Voronoi region of μ into the new Voronoi regions of μ (green) and λ (reddish). The right figure shows the result of the `k-means` run directly following the jump. μ and λ are now separated again each being in the center of gravity of their respective Voronoi set.

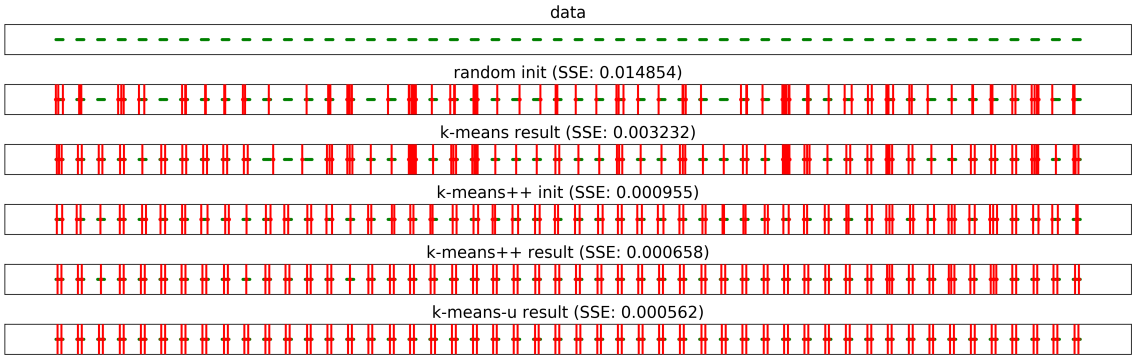


Figure 16: Results of `k-means`, `k-means++` and `k-means-u` for a one-dimensional distribution consisting of 50 clusters. The task is to distribute 100 centers among the data such that the SSE is minimized. `k-means` suffers from the poor (but typical) random seeding. The `k-means++` seeding is able to distribute the centers much better but still makes some "errors" wrt. the given data set (in accordance with the analysis from section 6). These errors (positioning either one or three centers in a cluster) cannot be corrected by the following `k-means` phase. `k-means-u`, starting from the displayed result of `k-means++`, is able to generate a result with two centers in each cluster, a necessary condition for the optimum in this case. The error achieved by `k-means-u` is 14.6% lower than that from `k-means++`.

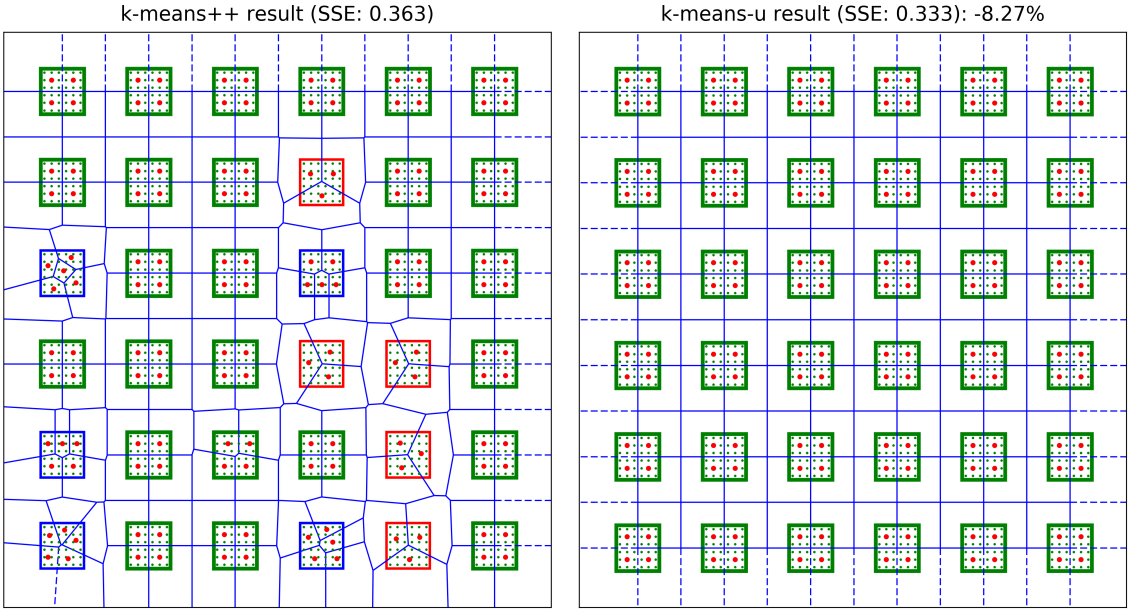


Figure 17: `k-means-u` optimally solves clustering problem A-4, Left: Result produced by `k-means++`. 10 of 36 clusters do not contain the optimal number of centers. Right: Result of `k-means-u`. The non-local jumps lead in this case to the optimal solution

Figure 18: `k-means-u` performs non-local jumps: starting from the result of `k-means++` the derived sequence of jumps is displayed. Each sub figure shows a converged configuration of `k-means`(resp. `k-means++` in the initial figure) and the non-local jump to be performed based on this configuration. The displayed SSE is always "pre-jump". Jumps are performed until the error stops dropping or even raises. At that point the last-but one local minimum is returned as the final result of the algorithm.

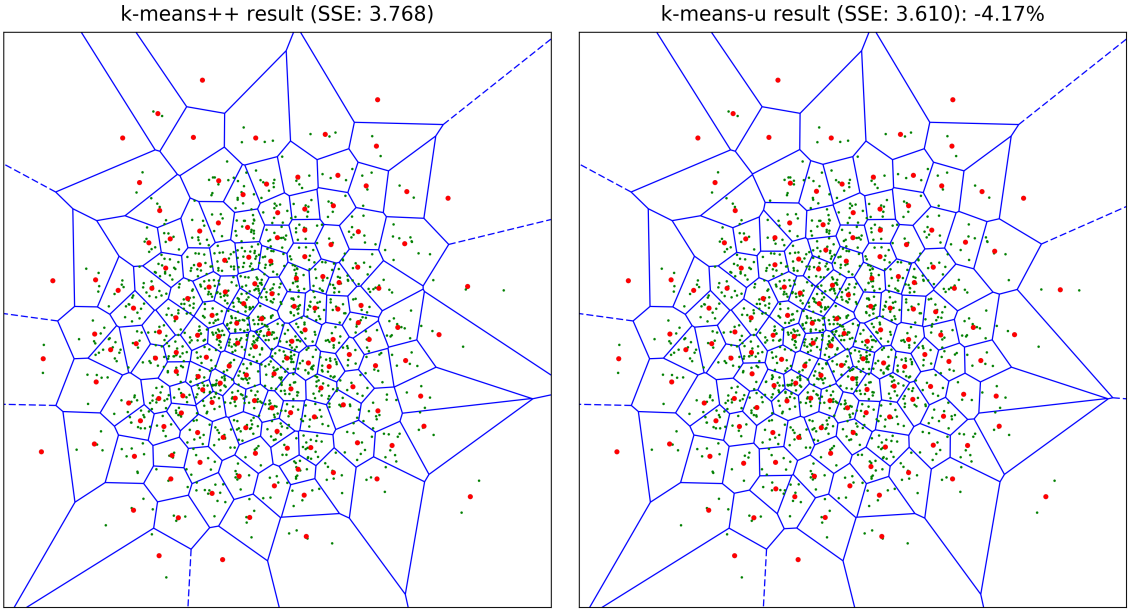


Figure 19: `k-means++` and `k-means-u` applied to mixture of overlapping Gaussians with 1296 data points and 36 centers, Left: Result produced by `k-means++`. Right: Result of `k-means-u`. The relative SSE improvement is 4.17%

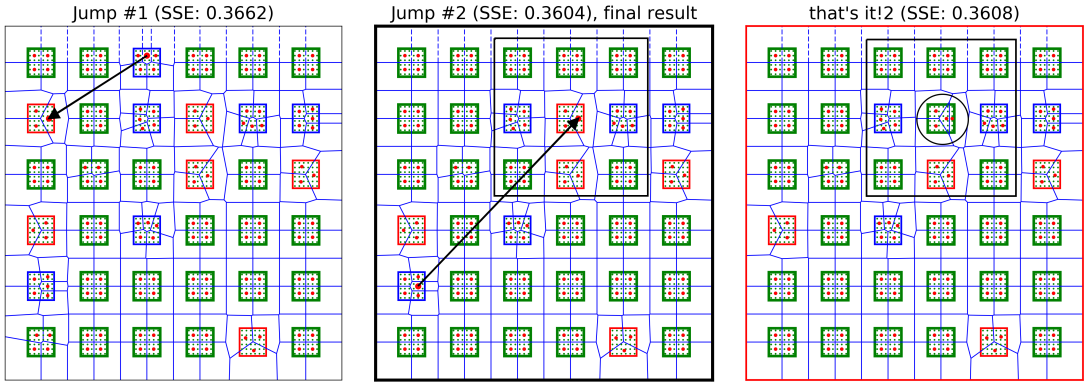


Figure 20: Too early stopping of `k-means-u`: Since already the second jump led to an increase of the SSE (right-most figure), the algorithm returns a result with many non-optimally covered clusters (center figure). The relevant region of the input space is indicated by a box and is shown enlarged in figure 21.

8. An occasional problem of `k-means-u`: too early termination

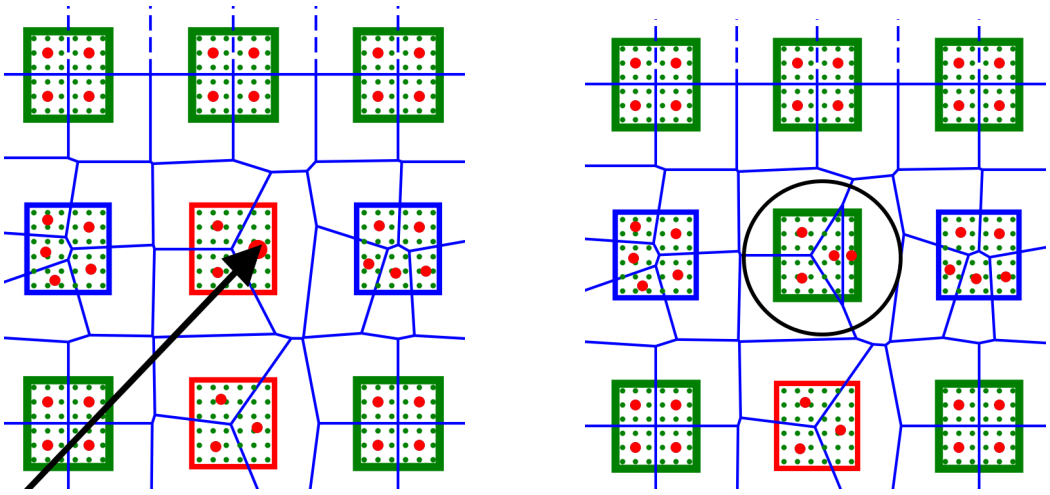
Sometimes we observed in simulations that `k-means-u` finished very early, i.e. at a point in time when there were seemingly many optimization opportunities left. Initially we suspected a programming error, but indeed in all investigated cases the most recent jump had led to a particular poor - but stable - configuration causing a relative increase of the SSE and thus a termination of `k-means-u` according to its definition.

In figure 20 such a simulation sequence is depicted. `k-means-u` already terminated after two jumps because the SSE had increased after performing the second jump. In figure 21 a detail view of the cluster causing the error increase is shown. It does contain the optimal number (4) of centers for this problem, but their arrangement determined by `k-means` is such that two centers are very close to each other and both have elongated Voronoi regions and therefore a relatively high distance to the member points of their respective Voronoi sets.

Fortunately these poor configurations seem to be relative rare. Due to the associated high SSE values, however, they have the potential to deteriorate mean performance statistics. Given that in `k-means-u` the re-positioning of the two centers affected from a jump (μ and λ) is based on a random vector, one relatively easily comes up with the idea to re-do the positioning in such cases. This leads to an extension of our original algorithm described in the next section.

9. `k-means-u*`

As exemplified in section 8 runs of `k-means-u` may end too early due to poor local configurations `k-means` runs into after a jump. In section 7 it was discussed what happens at a jump. Let us analyze this here in more detail: Directly after the repositioning the centers μ and λ divide the Voronoi set C_μ (see equation (2)) of μ among them using a



(a) Before jump 2 from figure 20. Center μ with largest local error is enlarged.

(b) Result of `k-means` phase: the encircled cluster has a very high local error

Figure 21: Details of the simulation sequence from figure 20 where `k-means-u` terminated with a quite sub-optimal result

$(d-1)$-dimensional hyperplane going through the previous position of μ and having the offset vector o (equation (15)) as its normal vector.[2]

The orientation of o depends on a random vector drawn from a d-dimensional hypersphere and determines how the data points previously associated with μ are distributed between μ and λ.

Each random choice leads with probability one (choosing two collinear vectors from a continuous distribution on the hypersphere has probability zero) to a different orientation of the hyperplane and likely to a different partitioning of the affected data points. The `k-means` phase following every jump leads to results depending on these partitionings. Different partitionings likely lead to different results of `k-means`.

Based on the above observations we propose the following simple extension of the `k-means-u` algorithm which we call `k-means-u*` (see figure 23). Instead of immediately terminating after an error increase we allow a small finite number $retry_{max}$ of retries of the most recent jump. Due to the random choice of the offset vector these retries possibly end up in a configuration with lower error and allow a continuation of `k-means-u`, sometimes for many steps. Once a retry was successful we "reset" the retry counter so the specified number of retries is again available at a later stage which will be at a lower error level than the previous retry sequence (since we just improved our "best solution"). This leads to a strictly monotone sequence of error values of the respective best solution after every retry sequence until the algorithm terminates. Since we never try to improve a configuration with

2. Strictly speaking this is true only in the limiting case when the length of o goes to zero, since with a non-vanishing vector o there may be some data points previously associated with other centers for which now either μ or λ is the closest center. If o goes to zero however, the combination of the Voronoi regions of μ or λ approaches the old Voronoi region of μ with arbitrary precision.

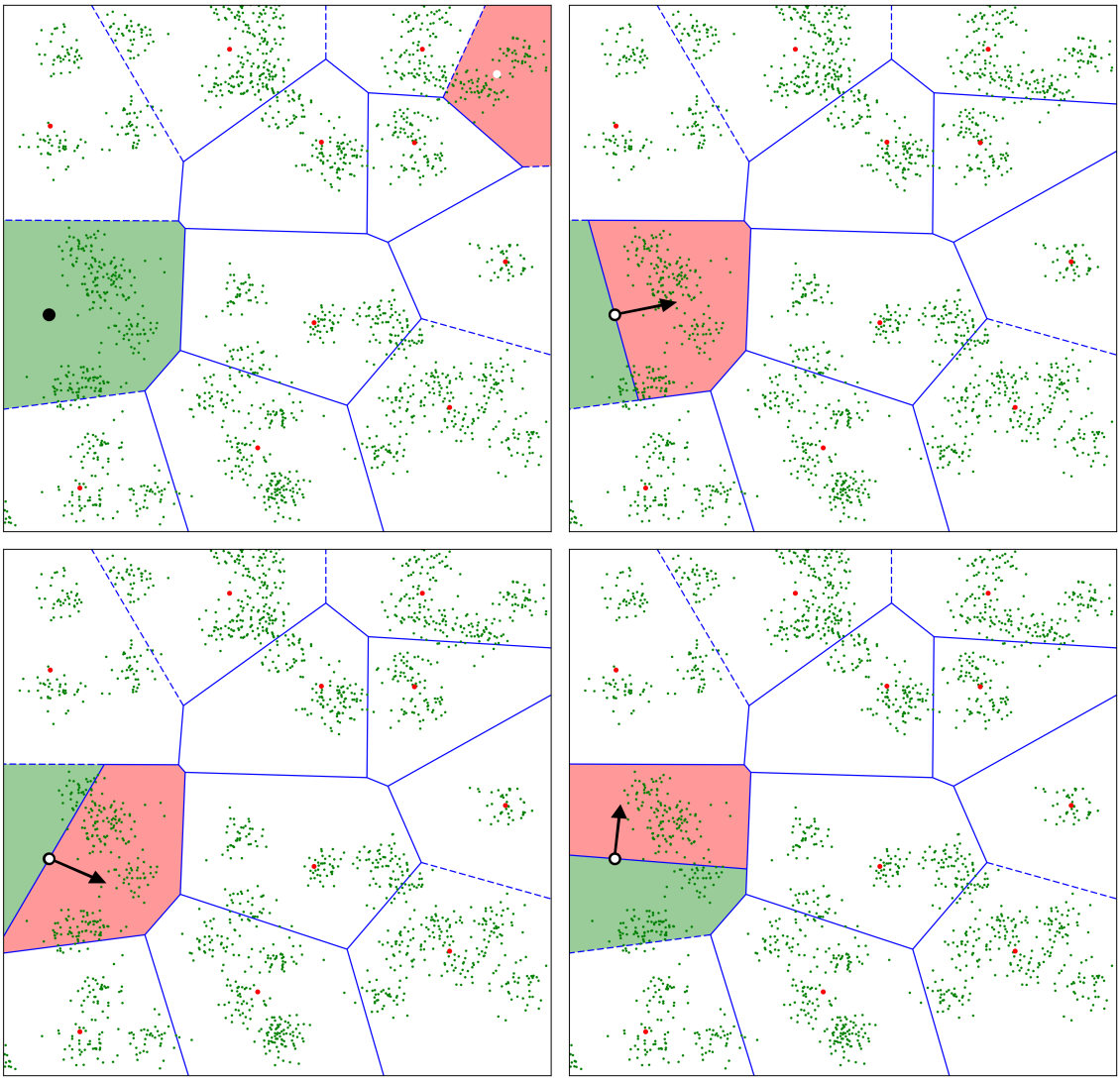


Figure 22: Different results of a jump. The upper left figure shows a configuration where `k-means` has converged. The center μ with the largest local summed squared error (SSE) is shown black (with a green Voronoi region). The center λ with the lowest utility is shown in white (with a reddish Voronoi region). The other three images show three different configurations obtained by performing a "jump" from λ to μ and applying a small random offset vector to both μ and λ but in opposite direction. The direction of the offset vector is shown as an arrow in each case. The offset vector is a normal vector of the $(d-1)$-dimensional hyperplane (line for 2-D data) which divides the previous Voronoi region of μ into the new Voronoi regions of λ (reddish) and μ (green).

a higher error than the best solution found so far, this retry procedure can be interpreted as randomized "greedy search".

Seeding: Perform k-means++;
$\phi_{\text{best}} \leftarrow \phi(\mathcal{C}, \mathcal{X})$; /* store lowest error so far (from k-means++) */
$\mathcal{C}_{\text{best}} \leftarrow \mathcal{C}$; /* store best $\mathcal{C}$ so far (from k-means++) */
$retry_{\max} \leftarrow n$; /* $n \in \{0, 1, 2, ...\}$ */
$retry \leftarrow 0$; /* initialize retry counter */
repeat
  Loop
    $\lambda \leftarrow \arg\min_{c_i \in \mathcal{C}} \phi(\mathcal{C} \setminus \{c_i\}, \mathcal{X})$; /* find least useful center */
    $\mu \leftarrow \arg\max_{c_i \in \mathcal{C}} \sum_{x \in C_i} ||x - c_i||^2$; /* find center with max. local error */
    $u \leftarrow$ (random vector from d-dimensional unit hypersphere);
    $d_\mu \leftarrow \sqrt{\frac{1}{|C_\mu|} \sum_{x \in C_\mu} \|x - \mu\|^2}$; /* mean distance around μ */
    $o \leftarrow \epsilon\, d_\mu\, u$; /* offset vector, $\epsilon = 0.01$ */
    $\lambda \leftarrow (\mu + o)$; /* position λ near μ */
    $\mu \leftarrow (\mu - o)$; /* position μ opposite to λ w.r.t. old μ value */
    Perform k-means using the current $\mathcal{C}$ as initial set of centers;
    if $\phi(\mathcal{C}, \mathcal{X}) < \phi_{\text{best}}$ **then**
      $\phi_{\text{best}} \leftarrow \phi(\mathcal{C}, \mathcal{X})$; /* store new lowest error */
      $\mathcal{C}_{\text{best}} \leftarrow \mathcal{C}$; /* store new best $\mathcal{C}$ */
      $retry \leftarrow 0$; /* improvement! reset retry counter */
    else
      break ; /* exit loop */
    end
  EndLoop
  $retry \leftarrow retry + 1$;
  $\mathcal{C} \leftarrow \mathcal{C}_{\text{best}}$; /* rewind to best solution so far (=previous) */
until $retry > retry_{\max}$;
return $\mathcal{C}_{\text{best}}$;

Figure 23: The k-means-u* algorithm. Additions to k-means-u shown red

In figure 24 a typical simulation sequence of k-means-u* is shown. While k-means-u would have stopped after 7 jumps, k-means-u* continues, in this case to the optimum.

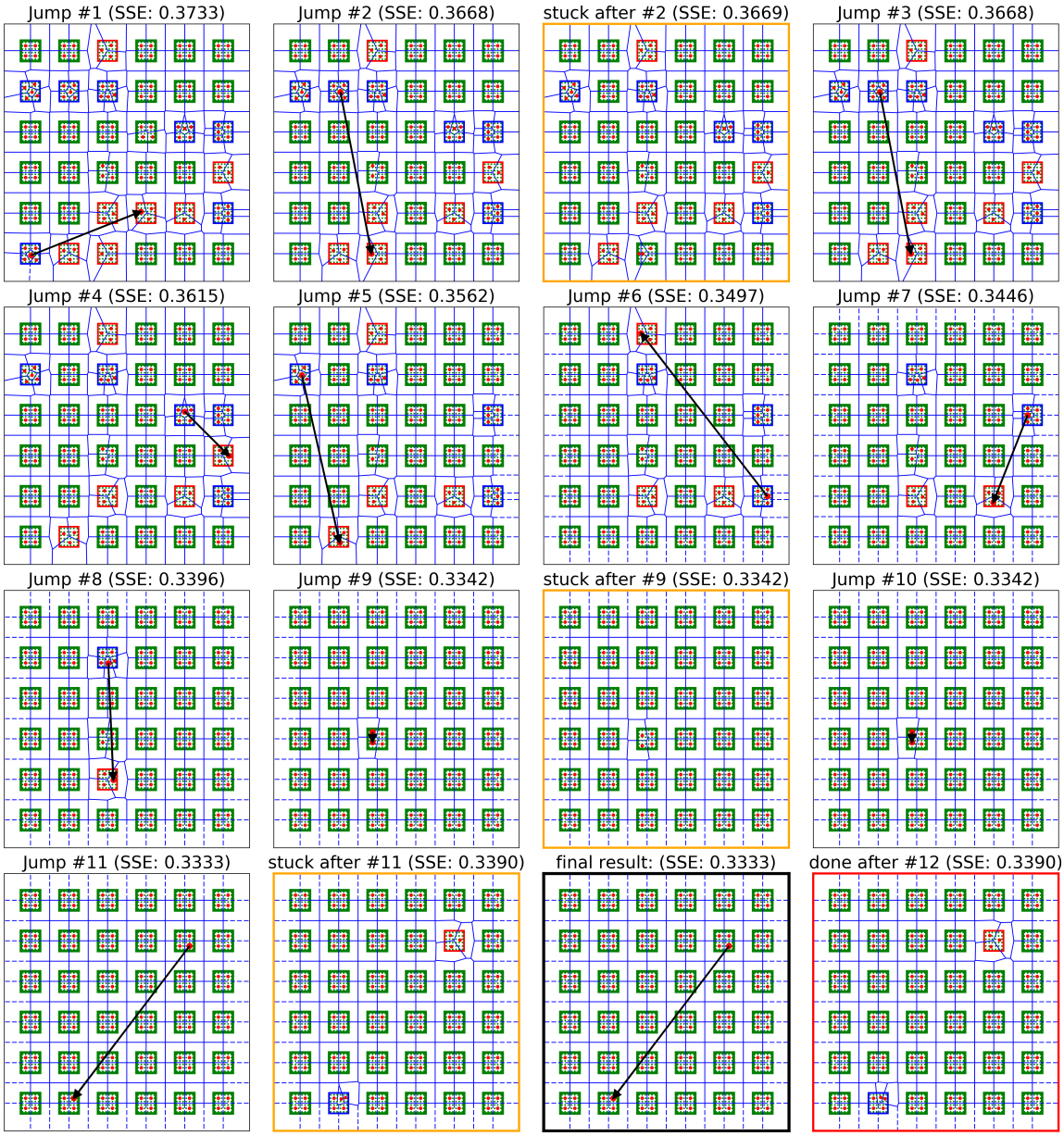


Figure 24: `k-means-u*` performs non-local jumps. We allow one retry per error level ($retry_{max} = 1$). Already after jump #2 the error increases, so `k-means-u` would stop here (SSE: 0.3668). We write "stuck after ..." above those cases and frame them in orange. The problematic constellations is actually very similar to the one shown in figure 21 b). The performed retry luckily results in a lower error and the `k-means-u` algorithm can continue for 5 successful jumps (reaching an SSE of 0.3342). Jump #9 leads to an error increase again but also in this case the retry is successful and directly leads to the final solution with an SSE of 0.3333 which happens to be the optimum for this particular clustering problem. Further jumps and retries cannot improve this result, so `k-means-u*` terminates shortly after. From the definition of the algorithm it follows that every `k-means-u*` simulation must end with a sequence of $retry_{max}$ unsuccessful retries (one in this case).

10. Empirical results

We performed systematic tests of k-means++, k-means-u and k-means-u* with 5 different data sets. For each data set a large range of values for k was investigated. For each of these k-values 10 different simulation runs were performed. Each single simulation consisted of three phases:

1. k-means++ (i.e. the k-means++ seeding followed by k-means)
2. k-means-u, starting from the result of the k-means++ run and continuing until the SSE did not fall anymore (the stopping criterion of k-means-u)
3. k-means-u* starting from the result of k-means-u and allowing 2 retries for each time k-means-u came to a stop.

For each data set we show an illustration of the data set itself and a performance chart (figures 25 to 34). If the dimension d of the data set is larger than two, we display all d^2 pairs of dimensions, each in a separate sub plot. The scaling of each sub plot is chosen such that the whole available area is used to display data points. Therefore different subplots may have different scalings, but the general nature of the data should be more visible this way.

The performance chart takes the performance of k-means++ as the baseline and indicates for both k-means-u and k-means-u* by how many percent they did reduce the SSE obtained by k-means++. No improvement would correspond to a data point on the k-axis and any actual improvement to data points above the k-axis. Per construction the new algorithms can not deliver anything worse than k-means++ so there are no values below the k-axis. For both k-means-u and k-means-u* the mean improvement (main chart) as well as the minimum and maximum improvements (error bars) are shown. While the mean indicates what to expect from one algorithm run, the maximum is an indication of what one could achieve by picking the best result of several runs.

The figure captions of the performance charts contain specific remarks regarding the simulation results. In general the new algorithms were able to improve a clear majority of the k-means++ results and often by a large margin. k-means-u* in particular was not only able to raise the mean improvement compared to k-means-u but also to obtain in many cases much higher maximum values (often more than 2 times as high as the maximum values of k-means-u).

With the exception of data set A (figure 25) which - as we know from earlier sections - is challenging for k-means++and data set B (figure 27)which was included as an example of an unstructured data set the other data sets have not been constructed or chosen with any result in mind but rather to provide a certain variety. Two data sets (cloud and propulsion) were taken from the UCI Machine Learning Repository as an established source of well-kept data sets. The simulations were performed in python using the optimized implementation of k-means++ contained in the scikit-learn package and a (non-optimized) numpy-based implementation of k-means-u and k-means-u*. Since on this base the comparison of running times was difficult, we compared the number of Lloyd iterations.

In figures 35 to 39 we display for all performed simulations the relative overhead of k-means-u and k-means-u* in terms of Lloyd iterations as well as the fraction of k-means++

solutions which `k-means-u*` was able to improve. The error reduction shown earlier is repeated for reference as well.

The overhead of `k-means-u*` over `k-means++` measured as described ranged between 10% and 230%. In computing this we considered that `k-means++` is executed 10 times in `scikit-learn` before the best result is returned. This means that for all our experiments the effort for the more complex one of our algorithms (`k-means-u*`) was within a constant factor (3.3) of `k-means++`. Given that `k-means-u*` provides significant solution improvements for an NP-complete problem this can be seen as a very moderate effort.

With few exceptions (mainly for small values of k or values of k near the number n of data points) the percentage of `k-means++` solutions improved by `k-means-u*` is at 100%. Thus according to our experiments it is highly likely that an arbitrary solution found by `k-means++` can be further improved by `k-means-u*`.

Figure 25: Data set A from figure 3, dimensionality $d = 2$, number of data points $n = 1296$ $(= 36 * 36)$, number of clusters $g = 36$.

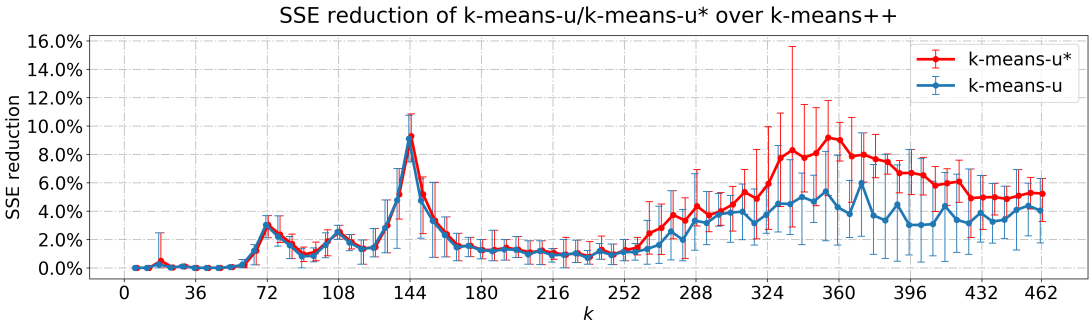


Figure 26: Simulation results for data set A (see figure 25). In accordance with our analysis `k-means++` finds good results until $k = 36$ but solution quality degrades (shown by the large improvements of `k-means-u` and `k-means-u*`) if k is increased to 72 or further multiples of 36. The problem A-4 illustrated in figures 11, 17, 18, 20, 21 and 24 corresponds to $k = 144$. For values of $k > 250$ the improvements obtained by `k-means-u` seem to grow independently of k being an integral multiple of 36 and the effect of adding a greedy search (`k-means-u*`) becomes very prominent, occasionally doubling the already significant improvements obtained by `k-means-u`. 10 runs per k-value

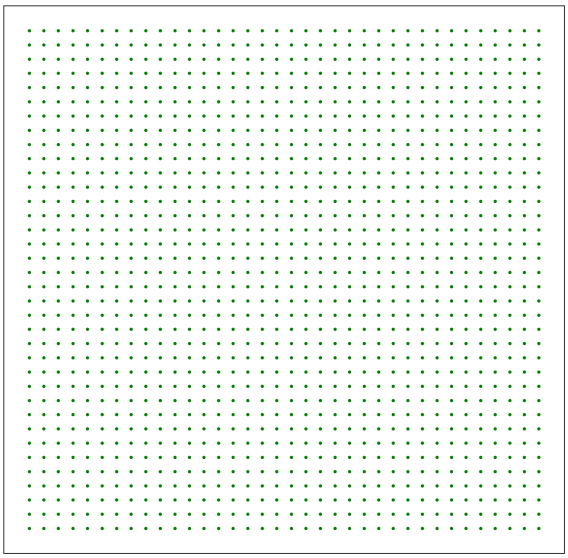


Figure 27: Data set B. This dataset which has exactly the same number of points as dataset A has been included as an example of a very simple unstructured data set with the expectation that there would be no large improvements of the `k-means++` results by our algorithms, a wrong assumption as the simulation results (see figure 28) show. dimensionality $d = 2$, number of data points $n = 1296$ $(= 36 * 36)$, number of clusters $g = 1$ (or $g = 1296$ depending on interpretation).

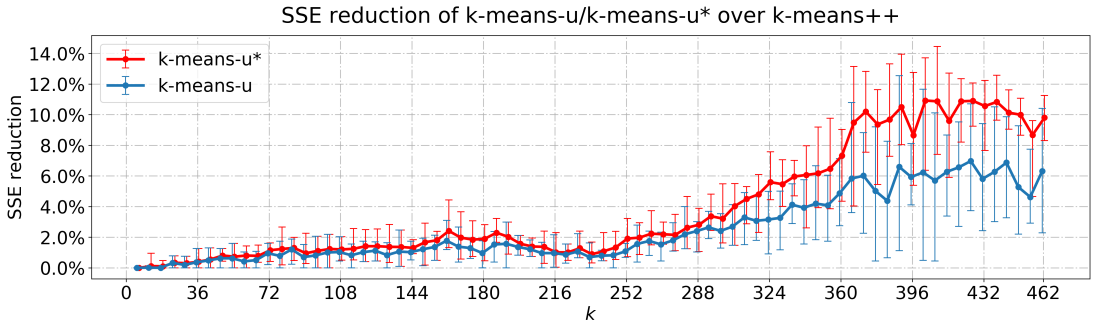


Figure 28: Simulation results for data set B (see figure 27). For smaller values of k the improvements over `k-means++` are moderate (up to 2%) and `k-means-u*` does not deliver a large advantage over `k-means-u`. Starting approximately with $k = 288$ and increasingly with larger k-values however, `k-means-u` is able to find solutions up to 6% better (in the mean) than `k-means++` and `k-means-u*` even finds solutions up to 11% better (in the mean) than those of `k-means++`. 10 runs per k-value

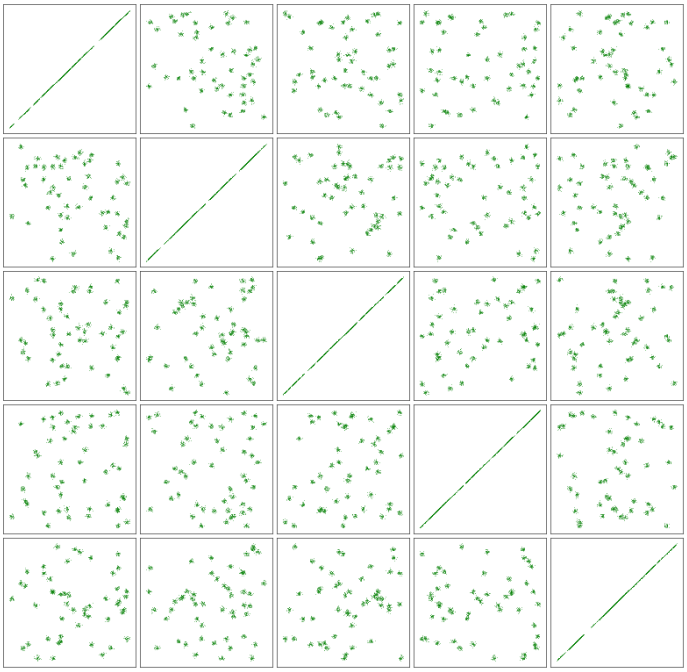


Figure 29: 5-dim. mixture of 50 Gaussians (all pairs of dimensions displayed), normally distributed with $\sigma = 0.00001$ and unit covariance matrix, $d = 5$, $n = 2000$, $g = 50$

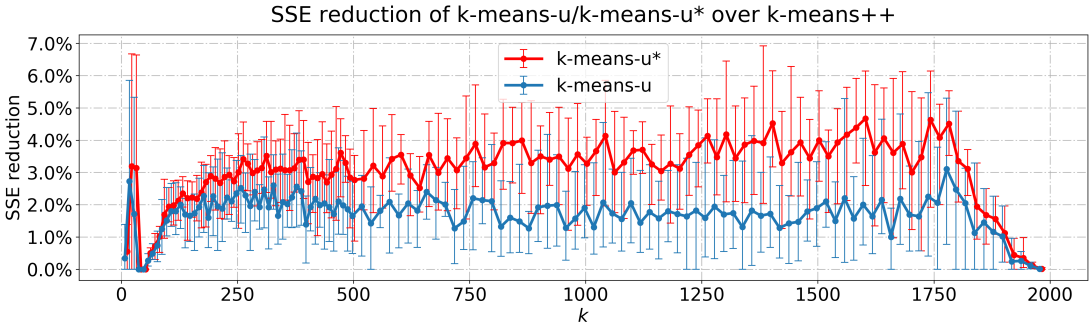


Figure 30: Simulation results for 5-dim. mixture of 50 Gaussians (see figure 29). One can note that both `k-means-u` and `k-means-u*` are unable to find improvements for $k = 50$ (50 is also the number of clusters in the data set) but for some smaller values of k for all larger values considerable improvements are found. 10 runs per k-value

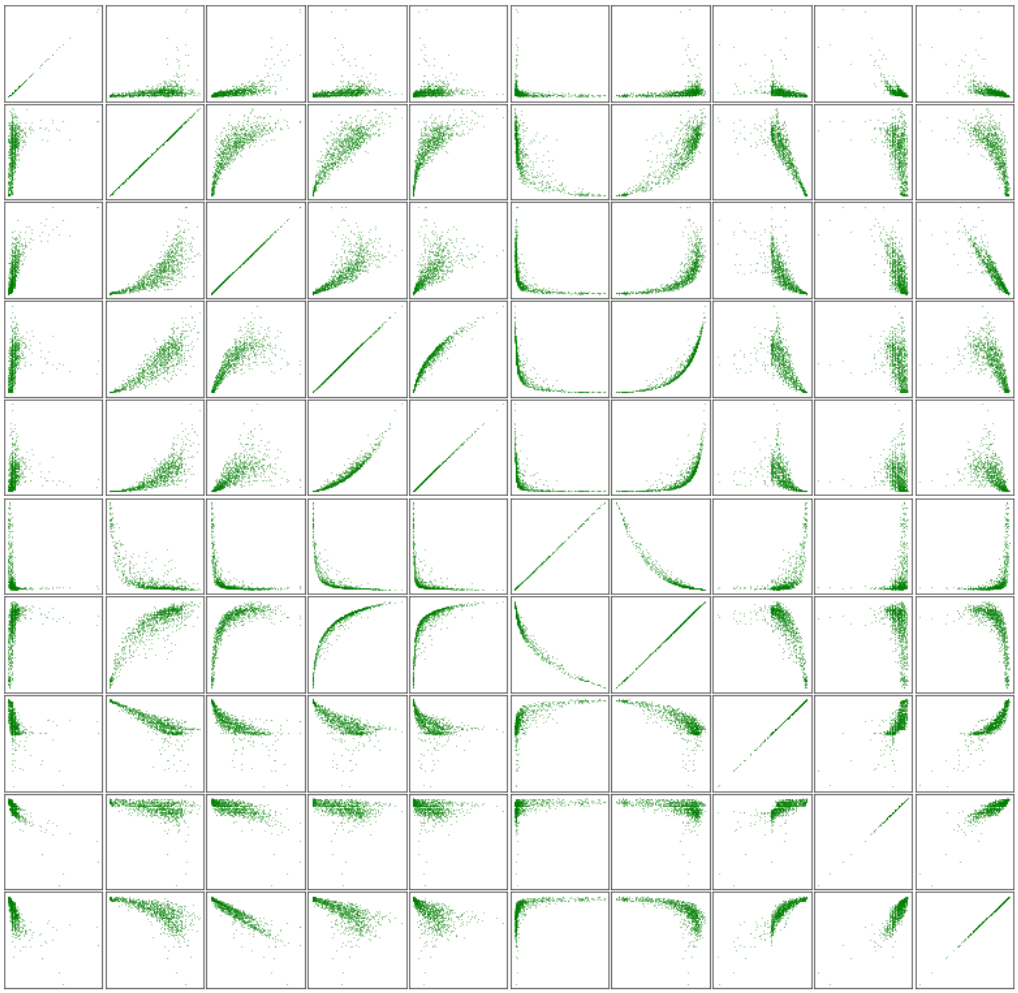


Figure 31: Cloud data from UCI (https://archive.ics.uci.edu/ml/data sets/Cloud), (all pairs of dimensions displayed), data preprocessed with `sklearn.preprocessing.StandardScaler` to have unit variance in each direction, $d = 10$, $n = 1024$, $g =$(unknown)

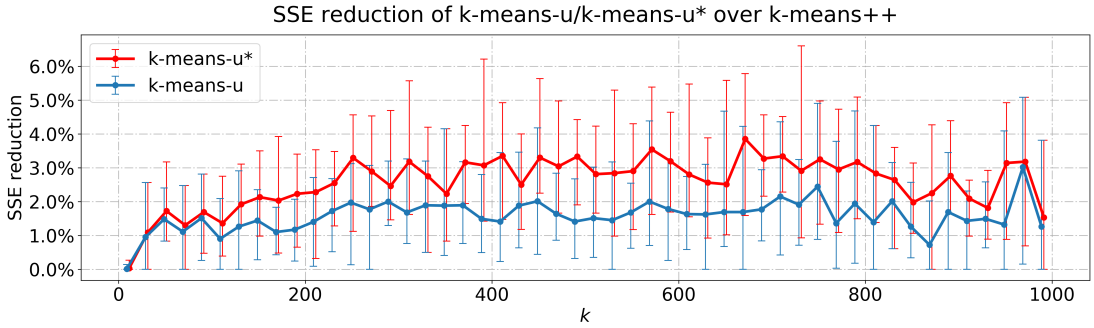


Figure 32: Simulation results for Cloud data from UCI (see figure 31), 10 runs per k-value

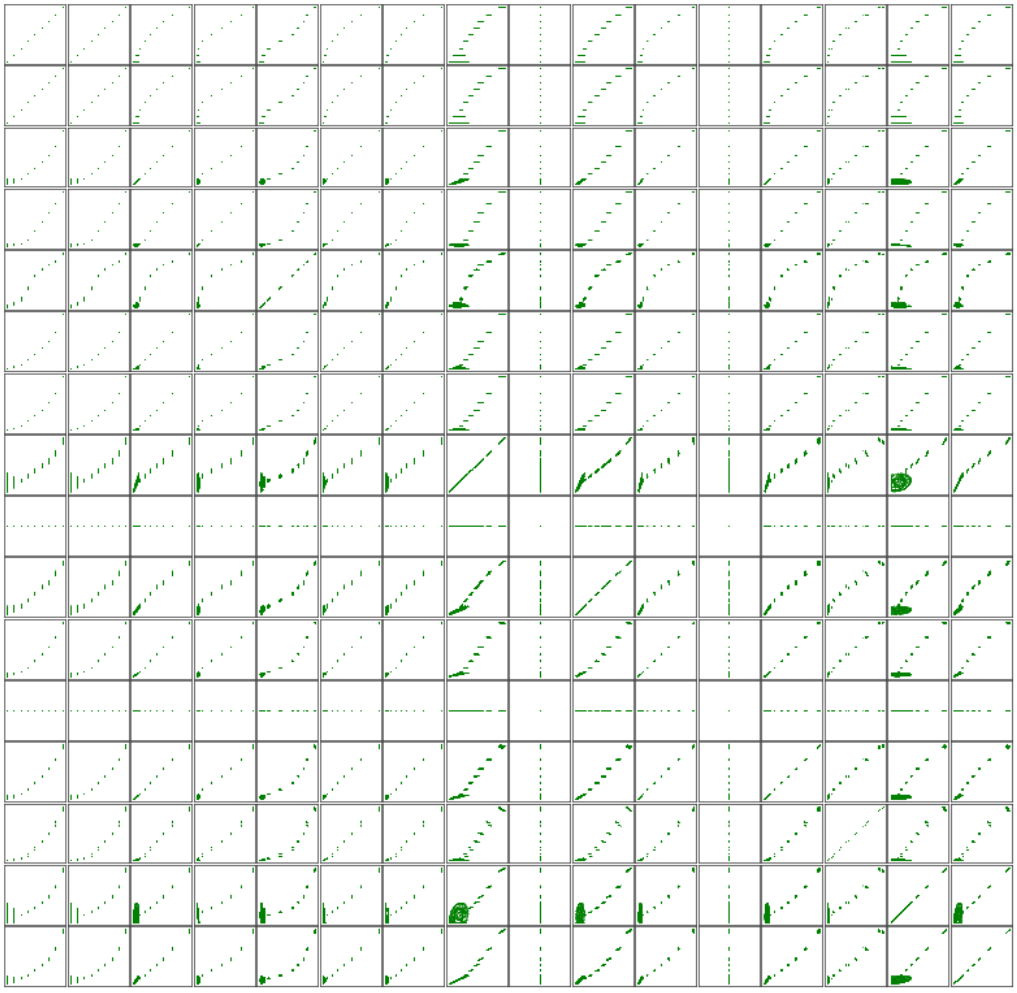


Figure 33: Propulsion data from UCI (https://archive.ics.uci.edu/ ml/data sets/Condition+ Based+Maintenance+ of+Naval+Propulsion+Plants), (all pairs of dimensions displayed), data preprocessed with `sklearn.preprocessing.StandardScaler` to have unit variance in each direction, $d = 16$, $n = 11934$, $g =$(unknown)

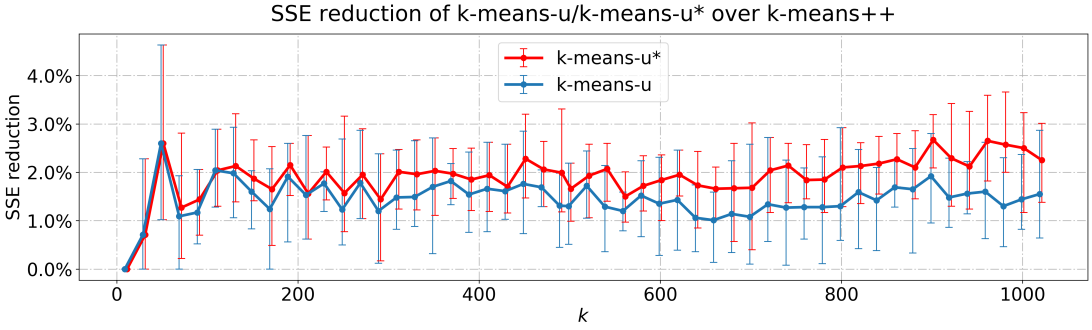


Figure 34: Simulation results for propulsion data from UCI (see figure 33), 10 runs per k-value

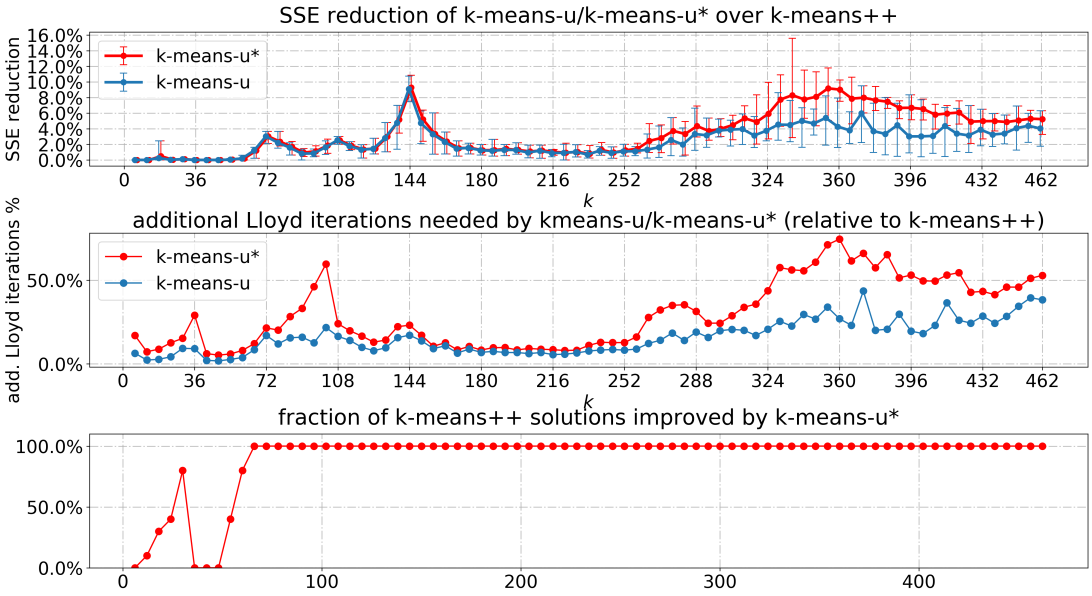


Figure 35: Data set A: SSE reduction achieved (top), additional Lloyd iterations needed (center), and fraction of `k-means++` solutions improved by `k-means-u*` (bottom). For large values of k there seems to be a correlation between achieved SSE reduction and additional Lloyd iterations needed.

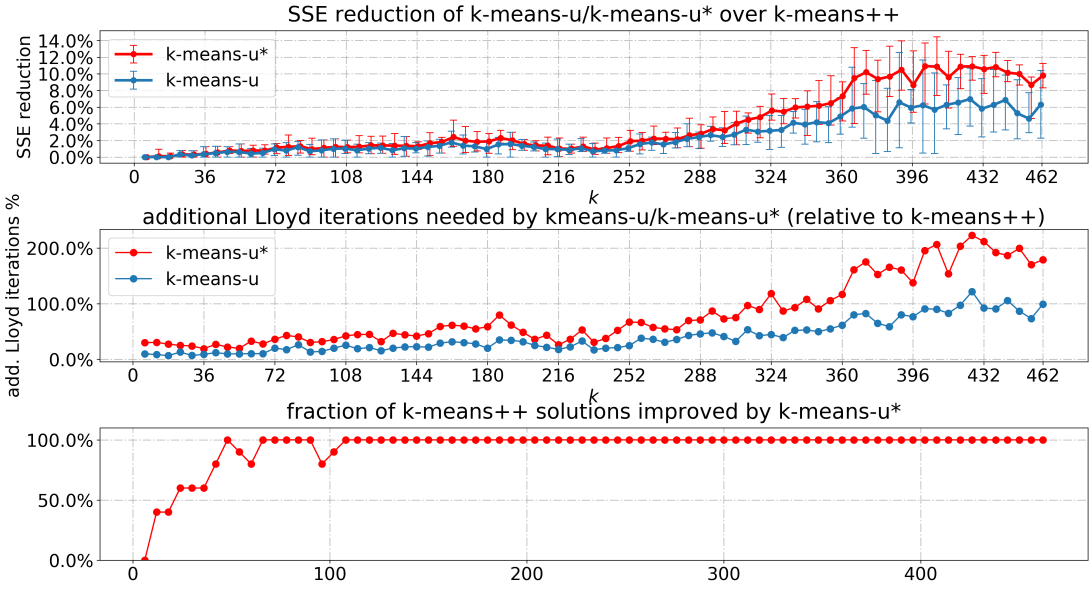


Figure 36: Data set B from figure 27: SSE reduction achieved (top), additional Lloyd iterations needed (center), and fraction of `k-means++` solutions improved by `k-means-u*` (bottom). One can not that the curves are similar to those for dataset A (figure 35) but lack the peaks where k is a low multiple of 36. The correlation between achieved SSE reduction and additional Lloyd iterations needed seems to be present for the whole range of k.

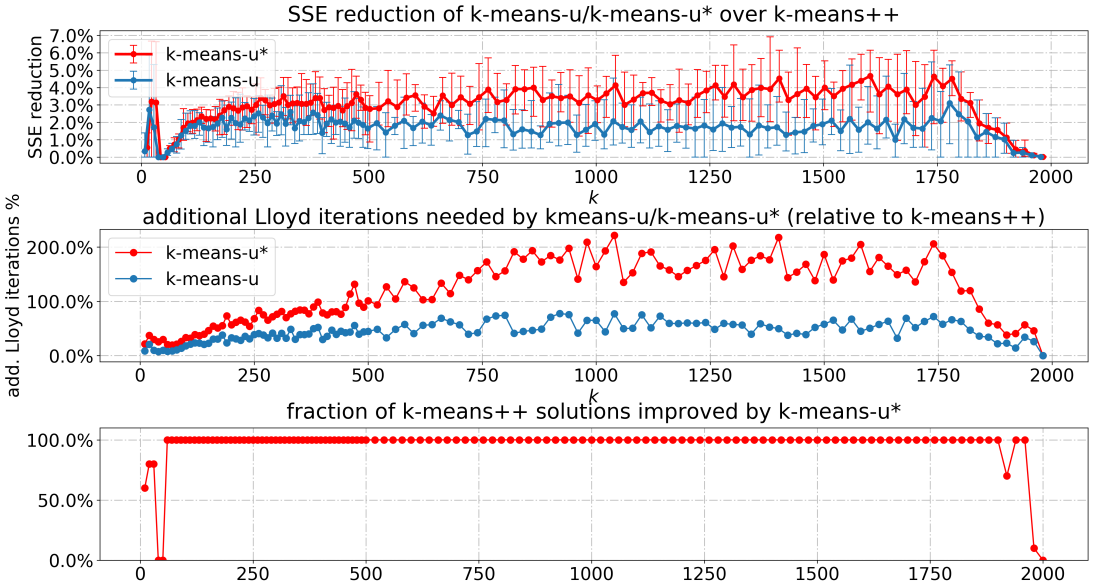


Figure 37: 5-dim Mixture of Gaussians: SSE reduction achieved (top), additional Lloyd iterations needed (center), and fraction of `k-means++` solutions improved by `k-means-u*` (bottom). For this problem the computation overhead compared to `k-means++` is particularly large. `k-means-u*` achieves an improvement around 3% over `k-means++` in most cases.

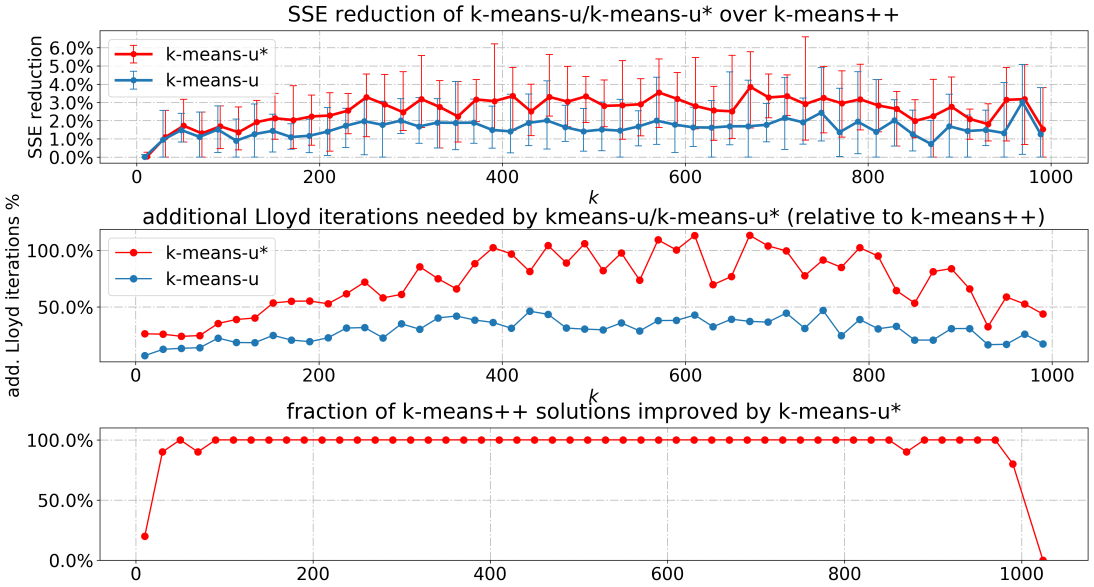


Figure 38: Cloud data: SSE reduction achieved (top), additional Lloyd iterations needed (center), and fraction of `k-means++` solutions improved by `k-means-u*` (bottom).

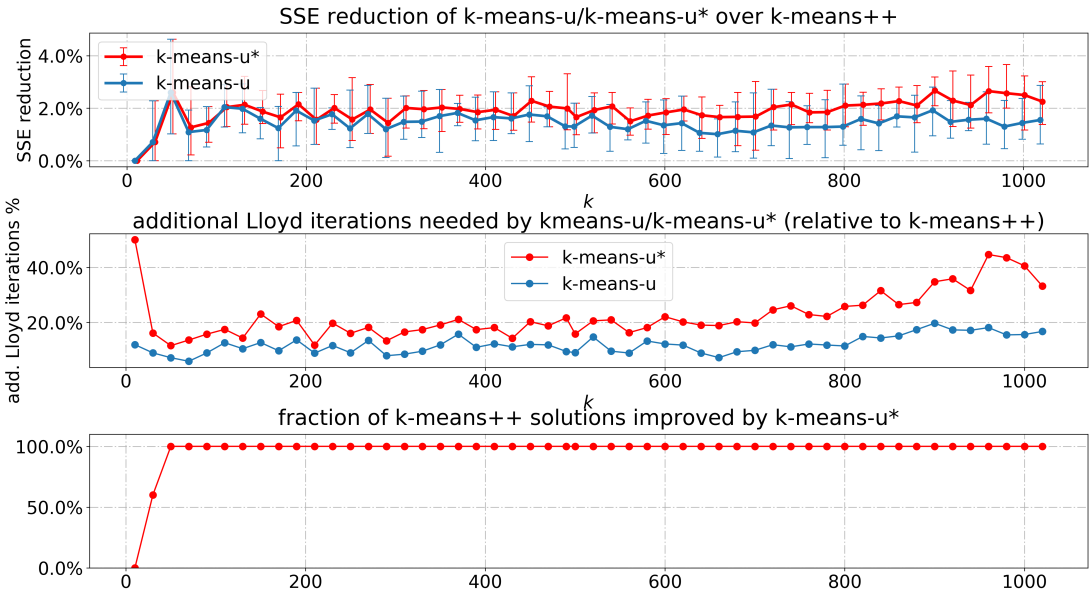


Figure 39: Propulsion data: SSE reduction achieved (top), additional Lloyd iterations needed (center), and fraction of `k-means++` solutions improved by `k-means-u*` (bottom). One can note that both the SSE improvements over `k-means++` (mostly around 2%) as well as the additionally needed Lloyd iterations compared to `k-means` (less than 50%, mostly even below 25%) are quite low. One could speculate that for this particular data set `k-means++` already finds quite good solutions which are difficult to improve. Nevertheless - as the bottom chart shows - starting with $k = 50$ every single solution found by `k-means++` is improved by `k-means-u*`.

11. Summary

In this paper we proposed the k-means-u algorithm which performs non-local jumps based on a simple utility criterion to improve the clustering results obtained with the current standard method k-means++. In some cases however we observed k-means-u stopping too early due to an error increase caused by a poor local minimum in which the k-means phase of k-means-u ended. This behavior was largely overcome by allowing a small and finite number of retries (e.g. 2) for the most recent jump. Due to the randomized local positioning during a jump, the resulting configuration after a retry is often different and possibly leads to a lower error which allows the algorithm to continue with the next jump. Further retries are then possible, but only on lower error levels which leads to a strictly monotone decreasing sequence of error values until the algorithm terminates. The resulting extended version of k-means-u is called the k-means-u* algorithm.

Simulations with a variety of data sets (partially from the UCI Machine Learning Repository) demonstrate that k-means++ is dominated w.r.t. solution quality (SSE) by k-means-u which from a certain value of k very often generates significantly better solutions and that k-means-u itself is dominated by k-means-u*, again significantly in a large number of cases. The observed improvements over k-means++ depended on the structure of the data distribution and the number k of centers and ranged from zero to about 8% mean reduction of the summed squared error. Our method incurs only a moderate computational overhead compared to k-means++ (below 230% in all of our experiments, less than 50% for the propulsion data, the largest of the data sets we used). Since the problem of minimizing SSE is NP-complete for data dimensions of two and larger, these additional costs for achieving often several percent error reduction appear to be quite reasonable. In clonclusion we consider k-means-u* a potential replacement of k-means++ in all cases where the quality of the clustering is of high importance.

Appendix A. Example Implementation

At the time of this writing the complete Python code used for the experiments in this paper is available from https://github.com/gittar/k-means-u-star.

References

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007. ISBN 9780898716245. doi: 10.1145/1283383.1283494.

Adam Coates, Ann Arbor, and Andrew Y Ng. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. *Aistats 2011*, 2011. ISSN <null>. doi: 10.1109/ICDAR.2011.95.

Bernd Fritzke. The {LBG-U} method for vector quantization – an improvement over {LBG} inspired from neural networks. *Neural Processing Letters*, 5(1):35–45, 1997.

Stuart P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 1982. ISSN 15579654. doi: 10.1109/TIT.1982.1056489.

George Marsaglia. Choosing a Point from the Surface of a Sphere. *The Annals of Mathematical Statistics*, 1972. ISSN 0003-4851. doi: 10.1214/aoms/1177692644.

Fabian Pedregosa and G Varoquaux. *Scikit-learn: Machine learning in Python*, volume 12. 2011. ISBN 9781783281930. doi: 10.1007/s13398-014-0173-7.2. URL `http://dl.acm.org/citation.cfm?id=2078195`.