

Lossy compression approach to subspace clustering

Łukasz Struski*, Jacek Tabor, Przemysław Spurek

The Faculty of Mathematics and Computer Science, Jagiellonian University, Łojasiewicza 6, 30–348, Kraków, Poland



ARTICLE INFO

Article history:

Received 3 October 2016

Revised 27 December 2017

Accepted 29 December 2017

Available online 30 December 2017

Keywords:

Subspace clustering

Projected clustering

Minimum description length principle

Entropy

ABSTRACT

We present a novel subspace clustering algorithm **SuMC** (Subspace Memory Clustering) based on information theory, MDLP (Minimal Description Length Principle) and lossy compression. **SuMC** simultaneously solves two fundamental problems of subspace clustering: determination of the number of clusters and their optimal dimensions.

Although **SuMC** requires only two parameters: data compression ratio r and a number of bits that are used to code a single scalar, the optimal value of compression ratio can be estimated by the Bayesian information criterion (BIC).

We verified that in typical tasks of clustering, image segmentation and data compression, we obtain either better or comparable results to the leading methods of subspace clustering.

© 2018 Elsevier Inc. All rights reserved.

1. Introduction

Clustering techniques have been extensively studied for years in areas such as statistics [6], pattern recognition [10], big data [36,37] and machine learning. However, most clustering algorithms do not work efficiently in higher dimensional spaces because of the inherent sparsity of data [8]. Problems arise when the distance between any two data points becomes almost the same (this is one of the reasons for the so-called *curse of dimensionality* [22]); therefore, it is difficult to differentiate similar data points from dissimilar ones. Moreover, clusters are embedded in the subspaces of high-dimensional data space and they often exist in subspaces of different dimensions. Therefore, in recent years, new branches of clustering have been developed: subspace clustering [17,20,33] and projected clustering [2,24], which generalize classical clustering methods for high-dimensional data.

The main problem of standard subspace clustering methods is the determination of the number of clusters and their optimal dimensions. Most algorithms (like ORCLUS) need a fixed number of groups and the same dimensions for all clusters, while some other algorithms can fit the number of clusters (4C), but at the cost of additional parameters (in the case of 4C we have 4 parameters).

In this paper we present a subspace clustering **SuMC**¹ method which does not have the above limitations. Our idea comes from the observation that in case of coding, it is often profitable to use various compression algorithms specialized for various data types. In such a case, a code for each point consists of two parts: the identifying index of compression algorithm used and the code produced by the method. We combine this approach with ideas from [31] where, using ideas similar to rate-distortion in information theory, a subspace clustering approach based on lossy compression was presented. We aim at minimizing the squared error, while keeping the total amount of memory fixed. Thanks to the use of constrained

* Corresponding author.

E-mail addresses: lukasz.struski@uj.edu.pl (Ł. Struski), jacek.tabor@uj.edu.pl (J. Tabor), przemyslaw.spurek@uj.edu.pl (P. Spurek).

¹ An implementation of **SuMC**, together with some example data sets, is available on GitHub: <https://github.com/lstruski/SuMC>.

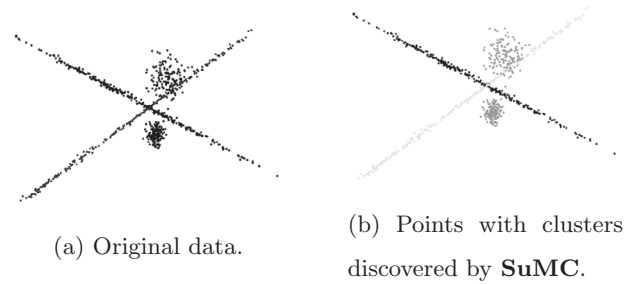


Fig. 1. Artificially generated dataset with four clusters (two zero dimensional and two one dimensional).

optimization procedure we are able to establish optimal number of clusters and dimensions in all groups simultaneously. In practice our method tends to reduce the unnecessary clusters and determine the “right” dimensions of these clusters (subspaces), see Fig. 1. Moreover, since **SuMC** is based on information theory, it can be efficiently used not only in clustering, but also in image segmentation or data compression, see Section 7.

SuMC needs only two parameters: the compression level r and the number of bits that are used to code a single scalar. However, in this case one is only interested in clustering itself, the reasonable value of r can be determined by using a version of Bayesian information criterion (BIC), see Section 4.

Conducted tests on artificial and real data show that **SuMC** obtains better results (measured by Rand index) than modern methods like ORCLUS or 4C [7], see Section 6. For example, on randomly generated datasets, **SuMC** obtains better Rand index than ORCLUS in about 80% cases, see Tables 3–4; while in the case of real data, **SuMC** won in about 65% of considered datasets, see Table 5. In terms of computational complexity, **SuMC** is comparable to the methods regarded as the best in the field, such as ORCLUS [4].

This paper is organized as follows. In Section 2 we review and discuss related works. In Section 3, we show the general theoretical framework, which in Section 4, we adapt to the subspace clustering case. In the next section we describe **SuMC** algorithm and discuss about its complexity. In Section 6, we present empirical results based on synthetic data and well-known datasets such as the datasets from the UCI Repository.

2. Related works

Traditional clustering methods are usually not applied to highly dimensional data sets due to their poor effectiveness, which is caused by the *curse of dimensionality* [22]. The problem of high dimensionality is often tackled by applying a dimensionality reduction method. These methods can be divided into techniques of feature selection or feature extraction. More information on feature selection methods can be found in [13]. The most popular feature extraction technique is the principal component analysis (PCA), also known as Karhunen–Loève method [10]. These methods in an optimal way transform the original data set into a lower dimensional space by creating dimensions that are linear combinations of given attributes. The new space has the property that the distances between points remain approximately unchanged. While these techniques may succeed in reducing the dimensionality, they have two shortcomings. Firstly, the new dimensions can be difficult to interpret, making it hard to understand clusters in relation to the original data set. Secondly, these techniques are not effective in identifying clusters that may exist in different subspaces of the original data set.

The problem of clustering of high-dimensional datasets is known as *subspace clustering* [5] or *projected clustering* [3,23]. Subspace clustering algorithms can be typically divided into soft and hard subspace clustering. The goal of hard subspace clustering is to identify the exact subspaces in which clusters are embedded, while soft subspace clustering algorithms perform clustering in high-dimensional spaces by assigning weight to each dimension to measure the contribution of individual dimensions to the formation of a particular cluster [11,35,38]. In this article, we focus on hard subspace clustering where the algorithms can be grouped into two main categories: those that use subspaces parallel to axes (this approach is especially useful for high-dimensional data) and those that are able to build arbitrary subspaces. One of the well-known methods belonging to the first category is the PROCLUS algorithm [3]. This method uses an approach similar to a k -medoid clustering, that is, the algorithm starts by choosing a random set of k medoids from a subset M of the data set and progressively improves the quality of medoid by iteratively replacing the bad medoids in the current set with a new point from M . Another method that deals with projected clustering is ORCLUS [4], which is an extension of PROCLUS that is able to find arbitrarily oriented clusters. ORCLUS is very similar to the k -means algorithm, except that it measures distances in subspaces, and not in full space. Similarly as PROCLUS, ORCLUS requires two parameters: number of clusters and dimensions. The algorithm is computationally intensive as it has a cubic complexity with respect to the dimension of the data, due to the computation of the covariance matrix and eigenvectors of each cluster. The problem with this approach is that the user has to specify the number of clusters in advance. If this guess does not correspond to the actual number of clusters, the results of ORCLUS deteriorate. In addition to these methods, we have compared our method with 4C (Computing Correlation Connected Clusters) [7]. This approach is based on density-based clustering (DBSCAN) [9] and correlation analysis (PCA) [15,29,30]. The

objective of 4C method is to find local subsets of data that exhibit strong correlations and are densely populated. In contrast to ORCLUS, this method returns the optimal number of clusters. However, 4C may not cope well with high-dimensional data, because both DBSCAN and PCA require high computational resources for such data.

3. General optimization problem

In this section we are going to present our approach, which is based on the applications of Shannons entropy [1,28], Kolmogorov complexity [18], and minimum description length principle [12] (MDLP). The first subsection presents an answers of our approach, while the second subsection explains it thoroughly in the context of two clusters.

3.1. Lossy Ockham's Razor

Our approach can be seen as an application of Ockham's Razor principle, which says that one should describe the objects as briefly as possible (this is in fact also a basic principle behind MDLP).

Compression version of Ockham's Razor. *Compress (losslessly) the given dataset/distribution, so that the total memory cost is minimal.*

Let us describe the typical application of the above approach in clustering (see also the discussion presented in [32]). We assume that we are given the family $\mathcal{F} = (f_1, \dots, f_k)$ of k lossless compression methods,² and that $\text{mem}(x, f)$ denotes the memory cost of compression of x by the coder $f \in \mathcal{F}$. Suppose that we are given data $X = (x_i)_{i=1..n}$ and that the i -th point is compressed with the coder $f_{c(i)}$, where $c(i) \in \{1, \dots, k\}$. Then, in practice, to compress/decompress the data, we first assign the identifier to the compression method and then the compressed code,³ which means that the total cost of compression of X equals

$$\text{mem}((x_i, f_{c(i)})_{i=1..n}) = \sum_{i=1}^n \text{cost of identification of the coder } f_{c(i)} + \text{mem}(x, f_{c(i)}). \quad (1)$$

To simplify the above formula, let us denote by $\text{mem}(Y, f)$ the total memory cost of compression of data set Y by a fixed compression method f (observe that in this case the cost of identification is zero as all points are compressed with one fixed method). Suppose that in our compression we use k compression methods $f_1, \dots, f_k \in \mathcal{F}$, and put

$$X_i = \{(x_j) : x_j \text{ was compressed by coder } f_i\}.$$

Now, from the Shannon theory we know that if we use $|X_i|$ times compression method f_i , then the optimal (at least asymptotically) code length needed to identify f_i equals $-\log_2(|X_i|/|X|)$. This implies that (1) reduces to (for more information we refer the reader to Section 3.2):

$$\text{mem}(X_1, f_1; \dots; X_k, f_k) = \sum_{i=1}^k (-|X_i| \log_2(|X_i|/|X|) + \text{mem}(X_i, f_i)). \quad (2)$$

Observe that as a result of application of the above formula, the model used cannot become too complicated, since using too many coders costs. Thus, the MDLP/entropy Ockham's Razor applied to clustering can be formalized as follows:

Optimization Problem 3.1 (MDLP). Assume that we are given a family $\mathcal{F}$ of compression methods. Find a splitting of the data X into $X_1, \dots, X_k$ and compression methods $f_1, \dots, f_k \in \mathcal{F}$ so that the value of $\text{mem}(X_1, f_1; \dots; X_k, f_k)$ is minimal.

Our basic idea is not only indebted to these methods, but it also uses the concept which comes from transmitting the data through a channel of fixed capacity. It occurred that in our studies concerning subspace clustering, the lossy compression became more fruitful than lossless compression. We can summarize this point of view in the following form:

Lossy Ockham's Razor. *Assume that we are given a fixed amount of memory that we can access. Compress the data, so that the (total) error is minimal.*

In this paper we show how one can apply the Lossy Ockham's Razor principle in subspace clustering. Let us remark here that in this work we generalize and modify some ideas from our earlier paper [31], in which we did not apply the MDLP approach described above (it occurs, see the end of Section 4, that the subspace clustering method constructed in [31] can be obtained as a limiting case of our model, where the amount of bits b used for saving in memory of a given scalar converges to ∞).

To apply the Lossy Ockham's Razor principle to subspace clustering one has to specify the following:

- What is understood by the amount of memory;⁴
- What is understood by the total error.

² In general those methods may have some free parameters.

³ To be able to decompress the point we need to know which method of compression was used.

⁴ This is connected to the question of what type of model we use for compression.

Explanations are provided this in [Section 4](#). However, roughly speaking, the amount of memory is proportional to the number of used scalars, while the error $\text{error}(X, f)$ is understood as the standard mean squared-error. Now, our basic problem can be stated as follows:

Optimization Problem 3.2. [Lossy-MDLP] Assume that we are given a set of compression methods $\mathcal{F}$ and a fixed amount of memory M . Find a splitting of the data X into $X_1, \dots, X_k$ and compression methods $f_1, \dots, f_k \in \mathcal{F}$ so that the value of

$$\text{error}(X_1, f_1) + \dots + \text{error}(X_k, f_k)$$

is minimal, under the constraint

$$\text{mem}(X_1, f_1; \dots; X_k, f_k) \leq M.$$

3.2. Optimization problem for two clusters

Before proceeding to the study of the general problem, we first discuss the solution to the Optimization [Problem 3.2](#) for two clusters. From the practical point of view, we ask how to construct an effective algorithm to minimize the error while keeping the compression level fixed. The main difficulty in the minimization procedure is caused by the fact that, contrary to the classical Ockham's Razor, we do not have one characteristic (the length of theoretical code) associated with a given point.

It occurs that the solution to the two following problems (see [Section 4](#)) allows to efficiently deal with the minimization problem.

Problem 3.1. Given group X and the amount of memory M , find

$$\text{error}_{\mathcal{F}}(X, M) = \min\{\text{error}(X, f) \mid f \in \mathcal{F} : \text{mem}(X, f) \leq M\}.$$

To formulate the second problem, let us assume that the solution to [Problem 3.1](#) is known. Suppose now that we are given the amount of memory M , by which we want to code disjoint groups $X_1, \dots, X_l \subset X$ (we do not assume here that those groups fill the whole of X), then by [\(2\)](#) in practice we can use only the memory

$$M - \sum_{i=1}^l (-|X_i| \cdot \log_2(|X_i|/|X|)),$$

since the sum on the above is used for the coder identification cost. Consider the case of two groups and ask how to relocate the total amount of memory M between them to minimize the error.

Problem 3.2. Assume that we are given disjoint groups $X_1, X_2 \subset X$ and the amount of memory M . Find M_1 and M_2 satisfying

$$M = M_1 + M_2 - (-|X_1| \cdot \log_2(|X_1|/|X|)) - (-|X_2| \cdot \log_2(|X_2|/|X|)) \quad (3)$$

such that the value of

$$\text{error}_{\mathcal{F}}(X_1, M_1) + \text{error}_{\mathcal{F}}(X_2, M_2)$$

is minimized.

Assuming that the above problem has been solved, we can define the following:

Definition 3.1. We define $\text{error}_{\mathcal{F}}(X_1, X_2; X, M)$ as the minimum of

$$\text{error}_{\mathcal{F}}(X_1, X_2; X, M) = \min\{\text{error}_{\mathcal{F}}(X_1, M_1) + \text{error}_{\mathcal{F}}(X_2, M_2)\}$$

where the pair M_1, M_2 satisfies the constraint [\(3\)](#).

Observe that the first of the above problems deals with finding an optimal coder for the data, under the constraint of given fixed memory. The second one deals with splitting the amount of memory between two groups so that the total error is minimized.

Now we are ready to describe the approach we use to minimize the total error under the assumption that memory level is fixed. We use a version of the Hartigan-like [\[14\]](#) algorithm with random initialization of clusters with reduction of unnecessary clusters.

Let $\mathcal{F}$ be the set of coders we are given. Assume that total amount of memory M is fixed and we search for a splitting of $X = (x_i)$ into at most k subsets so that the total error is minimized.

STEP 1. Choose an arbitrary initial (for example, constructed in a totally random way) split of X into k groups $X_1, \dots, X_k$. Then as for the previous reasoning, we have

$$M - \sum_{i=1}^k (-|X_i| \log_2(|X_i|/|X|))$$

memory free to use for “single” coding. Since we are not given any additional information, we split this memory proportionally into the groups with respect to their sizes:

$$M_i = \frac{|X_i|}{|X|} \left(M - \sum_{i=1}^k (-|X_i| \log_2(|X_i|/|X|)) \right).$$

In other words, we first use statistically equal amount of bits to code each point in the clusters. Code X_i by using memory M_i to obtain the total error:

$$\text{error}_{\mathcal{F}}(X_1, M_1) + \dots + \text{error}_{\mathcal{F}}(X_k, M_k).$$

STEP 2. We proceed consecutively over all points of X . Using Hartigan-like approach, we switch the position of point $x \in X_i$ from cluster X_i to X_j if

$$\text{error}_{\mathcal{F}}(X_1 \setminus \{x\}, X_2 \cup \{x\}; X, M) < \text{error}_{\mathcal{F}}(X_1, X_2; X, M).$$

STEP 3. Typically we assume that we reduce the cluster if its cardinality decreases under a threshold level (usually set at a fixed small percentage of the total amount of data). Then, we free the memory used by the cluster and move both the memory and the points to the randomly chosen cluster from the existing one.

4. Subspace clustering optimization problem

In this section, we apply the approach presented in previous sections to the case of subspace clustering. As a result, we obtain a clustering method, which can reduce the number of clusters on-line, while building the clusters of different dimensions.

Let us start from the general subspace clustering problem we consider.

Subspace clustering problem. Let $X \subset \mathbb{R}^N$ be a given data set and let $k \in \mathbb{N}$ denote the upper bound of the number of clusters. Our goal is to divide data set X into pairwise disjoint groups $X_1, \dots, X_k$ such that each group X_i , $i \in \{1, \dots, k\}$ is well represented by an affine subspace V_i .

To apply the method described in the previous section, we need to specify what cost function (error) we use and how we measure the amount of memory.

4.1. Cost function

Let us first discuss the error. As a criterion of fitting the data Y to the subspace V , we use the squared error

$$\text{error}(Y, V) = \sum_{y \in Y} \text{dist}(y, V)^2, \quad (4)$$

where $\text{dist}(y, V)$ denotes the distance between the point and its orthogonal projection on subspace V . We first need to ask how to find the optimal subspace for Y of a given dimension and obtain the formula for the error. Luckily, the above question has a well-known solution that is given by the PCA algorithm [15,21].

Theorem A (see [21]). Let $N \in \mathbb{N}$, $Y \subset \mathbb{R}^N$ be a given dataset and let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_N$ be ordered eigenvalues corresponding to eigenvectors $v_1, \dots, v_N$ of covariance matrix cov_Y . Then the minimum square approximation error can be expressed as

$$\text{error}_m(Y) = \text{error}(Y, V) = |Y| \cdot \sum_{i=m+1}^N \lambda_i \quad \text{for } m \leq N \quad (5)$$

where the subspace V is defined by

$$V = \text{mean}_Y + \text{span}(v_1, \dots, v_m), \quad (6)$$

mean_Y is the mean of Y and $v_1, \dots, v_m \in \mathbb{R}^N$ are the m consecutive eigenvectors of the covariance matrix cov_Y of Y (ordered decreasingly with respect to eigenvalues).

Thanks to the above theorem, we can find an optimal subspace with the given dimension. Observe that the above function is defined only for an integer m . For practical reasons (minimization of the energy), it will be more convenient for us to work with a continuous model (with respect to dimension m of the subspace) instead of the discrete one. Thus we extend our cost function $\text{error}_m(Y)$ to the (theoretical) case of continuous dimension in an affine way by the formula

$$\text{error}(Y, m) := (\lfloor m \rfloor + 1 - m) \cdot \text{error}(Y, \lfloor m \rfloor) + (m - \lfloor m \rfloor) \cdot \text{error}(Y, \lfloor m \rfloor + 1), \quad (7)$$

where $\text{error}(Y, \cdot)$ for natural numbers is defined in (5). We put $\text{error}(Y, m) := 0$ for $m \geq N$. This allows us to consider non-integer dimensions for subspaces (see Fig. 2).

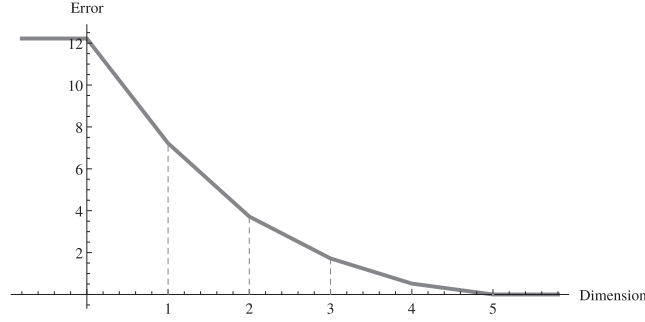


Fig. 2. This picture illustrates the plot of a function $\text{error}(Y, \cdot)$ defined by (7) with parameters $|Y| = 10$ and set of eigenvalues $\{0.5, 0.35, 0.2, 0.12, 0.052\}$.

4.2. The memory cost

Now let us proceed to the definition of the memory cost. Observe that if V is m -dimensional subspace, after projection of a point $y \in Y$ onto V , we need m parameters to identify it, while for the original y we need N parameters (scalars). Since we want to construct a subspace clustering approach, it is natural to understand the number of scalars as one of the basic parameters. However, to efficiently embed scalars in our setting, we need to specify the number of bits b we reserve for each scalar value in our theoretical memory. It occurs that this number will play an important role – the bigger it is, the relatively smaller the memory cost of cluster identification is (consequently, in the limiting case $b = \infty$,⁵ we would obtain no reduction of the number of clusters whatsoever). For example, if the point $x \in X$ belongs to a group X_i (for some $i \in \{1, \dots, k\}$), which is well represented by an affine subspace of dimension 3, and the memory for a single scalar representation is $b = 8$, then we need $8 \cdot 3 = 24$ bits to save the approximation of the point x .

Now let us estimate the amount of bits needed to store the set $X \subset \mathbb{R}^N$. Since each element of X is identified by its coordinates in the base and the base consists of N ($=$ dimension of the space) parameters (where as we have described above each real scalar needs b bits of memory), we obtain that the total number of memory needed to store X equals

$$\text{mem}_b(X) = Nb|X|. \quad (8)$$

Now, suppose that we have the splitting of a set Y into groups $Y_1, \dots, Y_k$, where Y_i is a subset of a subspace of $\mathbb{R}^N$ of dimension n_i . The amount of the memory to store a single element of Y_i is given as the sum of the identification cost of the i th cluster, which by the classical argument equals $-\log_2 \frac{|Y_i|}{|Y|}$, and the product n_i (number of base elements) by b . Consequently, we obtain that

$$\text{mem}_b(Y_1, n_1; \dots; Y_k, n_k) = \sum_{i=1}^k (n_i b - \log_2 \frac{|Y_i|}{|Y|}) \cdot |Y_i|. \quad (9)$$

If we come back to our uncompressed data X and the splitting $X = X_1 \cup \dots \cup X_k$, we apply the same notation as above to denote the memory needed to store the compressed version of X , where the elements of X_i are compressed to the optimal, with respect to the squared error, n_i -dimensional subspace given by the PCA argument.

To make our algorithm independent of the size of the dataset, we use the notion the data compression ratio r [26], which measures the rate between the total memory used before and after the compression. In our case, when we apply it to a point in $\mathbb{R}^N$, which is represented by a projection onto m dimensional subspace, this value equals

$$r := \frac{N}{m} \geq 1.$$

4.3. SuMC optimization problem

Now, by taking into account cost function (error) and memory cost, we can rewrite our optimization problem as follows.

Optimization Problem 4.1. Let dataset $X \subset \mathbb{R}^N$, the number of cluster k , the number of bits b , which represented each number, and the data compression ratio $r \geq 1$ be given. Our goal is to find the splitting of X into at most k clusters $X_1, \dots, X_k$ and numbers⁶ $n_1, \dots, n_k$, which minimize the total squared error

$$\text{error}(X_1, n_1; \dots; X_k, n_k) = \text{error}(X_1, n_1) + \dots + \text{error}(X_k, n_k), \quad (10)$$

⁵ Typically in computer languages $b \in \{8, 16, 32, 64\}$.

⁶ They correspond to the dimensions of the subspaces.

under the constraint

$$\text{mem}_b(X_1, n_1; \dots; X_k, n_k) \leq \frac{1}{r} \text{mem}_b(X). \quad (11)$$

Clearly, from (8) and (9), the formula (11) can be rewritten as:

$$\begin{aligned} \text{mem}_b(X_1, n_1; \dots; X_k, n_k) &= \left(n_1 b - \log_2 \frac{|X_1|}{|X|} \right) \cdot |X_1| + \\ &\dots + \left(n_k b - \log_2 \frac{|X_k|}{|X|} \right) \cdot |X_k| \leq \frac{1}{r} \text{mem}_b(X) = \frac{1}{r} N b |X|. \end{aligned}$$

Observe that the left hand side of (11) denotes the proportion of the total number of bits used for storage of orthogonal projection of points from clusters $X_1, \dots, X_k$ on subspaces of dimension $n_1, \dots, n_k$ with respect to the total initial number of bits needed to describe the dataset (in other words this can be seen as a compression level).

Now we show how to find a (possibly local) solution to Optimization Problem 4.1. Suppose that we have an arbitrary split of the data set X into k disjoint groups $X_1, \dots, X_k$ with associated dimensions $n_1, \dots, n_k \in \mathbb{R}$ such that the condition (11) holds. We want to modify the splitting of the data and the dimensions of the projections in such a way as to minimize the squared error given by (10) while keeping the constraint given by (11). For this purpose, we proceed in a Hartigan-like manner through all elements of the dataset X and verify whether we obtain smaller error by changing the belonging of an element (point) into cluster. Thus, for each consecutive $x \in X$, where $x \in X_i$ for some $i \in \{1, \dots, k\}$, we check if by changing the cluster belonging of x to some other X_j , where $j \neq i$, we could minimize the error. More precisely, under our memory constraint, we search for such a cluster X_j , $j \neq i$ and dimensions $\tilde{n}_i, \tilde{n}_j$ that after switching of x to cluster X_j , the total compression error is smaller, that is, we are looking for

$$\inf_{j \neq i} (\text{error}(X_i \setminus \{x\}, \tilde{n}_i; X_j \cup \{x\}, \tilde{n}_j) \mid \tilde{n}_i, \tilde{n}_j \geq 0 : \quad (12)$$

$$\text{mem}_b(X_i \setminus \{x\}, \tilde{n}_i; X_j \cup \{x\}, \tilde{n}_j) = \text{mem}_b(X_i, n_i; X_j, n_j),$$

and check whether the above value is smaller than $\text{error}(X_i, n_i) + \text{error}(X_j, n_j)$. To do this by applying the algorithm described in the previous section, we first need to solve the Problem 3.2. The following theorem shows that we can restrict ourselves to the search of the dimensions over a discrete set. However, in the proof, we will need the following easy lemma.

Lemma 4.1. Let $f_1, f_2 : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ be given continuous piecewise affine functions on intervals $[i, i+1]$ for $i \in \mathbb{N}$. We assume that $n_1, n_2 \in \mathbb{N}$ are such that

$$f_j(x) = 0 \text{ for } x \geq n_j, j = 1, 2.$$

Fix $a, b, R \in (0, \infty)$. Then

$$\inf (f_1(x) + f_2(y) \mid x, y \in \mathbb{R}_+ : ax + by = R) = \min \left(\min_{i=0..n_1} (f_1(i) + f_2(\frac{R-a \cdot i}{b})), \min_{i=0..n_2} (f_1(\frac{R-b \cdot i}{a}) + f_2(i)) \right).$$

Proof. Since f_1 and f_2 are continuous functions that are zero for sufficiently large points, the function $g : x \rightarrow f_1(x) + f_2(y)$, where y is chosen so that $ax + by = R$, attains global minimum. So let $\bar{x}$ and $\bar{y}$ be the pair satisfying the condition $a\bar{x} + b\bar{y} = R$ that realizes the minimum.

Assume that neither of these points $\bar{x}, \bar{y}$ is an integer. Then since f_1, f_2 are piecewise affine on intervals with integer endpoints, so is g in the neighborhood of $\bar{x}$. Consequently, by assumption g has to be locally constant at $\bar{x}$. Let $I = [x_0, x_1]$ denote the maximum closed interval containing $\bar{x}$ such that g is constant at I . Then by the previous reasoning, we obtain that either x_1 or $y_1 = \frac{R-ax_1}{b}$ is an integer. $\square$

Theorem 4.1. Let Y_1, Y_2 be given disjoint subsets of $\mathbb{R}^N$. Let $R \geq 0$ be a fixed positive real number. Then

$$(\text{mem}_b(Y_1, n_1) + \text{mem}_b(Y_2, n_2) \mid n_1, n_2 \in \mathbb{R}_+ : n_1|Y_1| + n_2|Y_2| = R) = \text{const}. \quad (13)$$

Moreover,

$$\begin{aligned} \inf (\text{error}(Y_1, n_1) + \text{error}(Y_2, n_2) \mid n_1, n_2 \in \mathbb{R}_+ : n_1|Y_1| + n_2|Y_2| = R) = \\ \min_{j=\{0, \dots, n\}} (\text{error}(Y_1, j) + \text{error}(Y_2, \frac{R-|Y_1| \cdot j}{|Y_2|}), \text{error}(Y_1, \frac{R-|Y_2| \cdot j}{|Y_1|}) + \text{error}(Y_2, j)), \end{aligned} \quad (14)$$

where $\text{error}(Y_i, \cdot)$ for $i = 1, 2$ is defined by (7), which means that one of the dimensions which realize the infimum taken as an integer.

Proof. Observe that (13) is a direct consequence of the definition of the memory.

To prove (14), we use Lemma 4.1. Observe that for $i = 1, 2$ function $\text{error}(Y_i, n_i) \equiv \text{const}$ for $n_i \leq 0$ and $\text{error}(Y_i, n_i) \equiv 0$ for $n_i \geq N$. Moreover, $\text{error}(Y_i, \cdot)$ for $i = 1, 2$ is non-negative piecewise linear function with integer endpoints $\{0, 1, \dots, N\}$

on the interval $[0, N]$. Hence functions $\text{error}(Y_1, \cdot), \text{error}(Y_2, \cdot)$ satisfy the assumptions of Lemma 4.1. As parameters a, b in Lemma 4.1, we put $a = |Y_1|, b = |Y_2|$. As a consequence of Lemma 4.1, we obtain new dimensions of these groups and at least one of n_1, n_2 is an integer, that is, $n_i \in \{0, \dots, N\}$ for some $i = 1, 2$. $\square$

Note that during the shift of the point between clusters we do not change the total amount of memory, which we devote to describing the points of the two considered clusters. This enables us to give a partial solution to Optimization Problem 4.1.

Observe that if we apply the above theorem to various pairs of clusters, then we can obtain dimensions of clusters which are not integer numbers. We show that we can modify these dimensions without changing the total memory cost and potentially decreasing squared error in such a way that at most one of the dimensions is non-integer.

Theorem 4.2. *Let $k, N \in \mathbb{N}$ and $X = X_1 \cup \dots \cup X_k \subset \mathbb{R}^N$ be given pairwise disjoint clusters with dimensions used for compression $n_1, \dots, n_k \in [0, N]$, respectively. Then there exist $\tilde{n}_1, \dots, \tilde{n}_k \in [0, N]$ such that at most one is non-integer and*

$$\text{error}(X_1, \tilde{n}_1, \dots, X_k, \tilde{n}_k) \leq \text{error}(X_1, n_1; \dots; X_k, n_k),$$

where $\text{error}(X_1, \cdot; \dots; X_k, \cdot)$ is defined by (10).

Proof. If $k = 1$ then thesis trivial is satisfied. Assume that $k > 1$. We use at most $k - 1$ times Theorem 4.1 for two consecutive clusters with non-integer dimensions. At the end of this procedure, we obtain that at most one of the clusters is compressed with non-integer dimension, and from Theorem 4.1, the total squared error of clustering process is no greater than before the modification. $\square$

Now we discuss the situation in which we converge with b to ∞ . Then the proportion between the memory needed to identify the cluster becomes infinitely small compared to the memory needed for saving the scalars which represent the coordinates of the point. Consequently, by going to the limit, we have to work with the amount of scalars used instead of the memory (since the memory would be infinite). Other consequence of proceeding with b to ∞ is that as the identification cost becomes negligible, the model loses the ability to remove clusters, since there would be no penalty cost for introducing new clusters. This can be useful when the number of clusters is arbitrarily fixed.

4.4. Parameter estimation

At the end of this section we present a possible method for determination of parameter r . We use a natural modification of the Bayesian information criterion [27] (BIC), which is defined as

$$BIC = -2 \cdot \ln L + m \cdot \ln(n)$$

where L is the likelihood function of the model, n is the number of data points in X and m is the number of free parameters to be estimated.

Let us first describe after [25] how to apply BIC in the case of classical k -means. To do so, we need probabilistic interpretation of the k -means cost function. In the case of one cluster X , the maximum likelihood value of generating data by the standard normal density $N(m, I)$ (the covariance is spherical and equal to identity) is obtained for $m = \text{mean}_X$ (mean value of the data) and equals

$$\ln L(X) = \sum_{x \in X} \ln(N(\text{mean}_X, I)(x)) = -\frac{1}{2} \sum_{x \in X} (\|x - \text{mean}_X\|^2 + N \ln(2\pi)).$$

In the case of k -clusters $X = X_1 \cup \dots \cup X_k$, making the identical spherical Gaussian assumption and counting the probability of each point only with respect to the Gaussian determined by its cluster, we get

$$\begin{aligned} \ln L(X) &= \ln \left(\prod_{i=1}^k \prod_{x \in X_i} N(\text{mean}_{X_i}, I)(x) \right) = \sum_{i=1}^k \sum_{x \in X_i} \ln(N(\text{mean}_{X_i}, I)(x)) \\ &= -\frac{1}{2} \sum_{i=1}^k \sum_{x \in X_i} (\|x - \text{mean}_{X_i}\|^2 + N \ln(2\pi)). \end{aligned}$$

Since constants for the minimization can be omitted, we obtain that

$$\ln L \approx -\frac{1}{2} \sum_{i=1}^k \sum_{x \in X_i} \|x - \text{mean}_{X_i}\|^2.$$

Since k -means is in fact the vectorization procedure, we can understand the above as

$$\ln L \approx -\frac{1}{2} \sum_{x \in X} \text{error}^2(x), \quad (15)$$

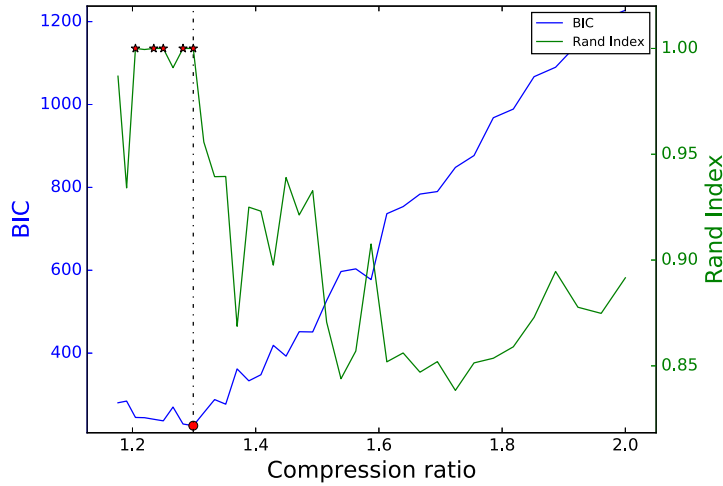


Fig. 3. The figure represents a relationship between compression ratio parameter r and BIC. There are two vertical axes: left blue axis corresponds to BIC, right green axis describes the Rand Index. Moreover, we mark points which show minimum of BIC (point) and maximum of the Rand Index (stars). The vertical dash-dot line shows the optimal value of compression ratio found by taking the minimal value of BIC. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1

Mean of results of Rand index (MIN – the mean of minimum of Rand index, MEAN – the mean of all results, BIC – the mean of Rand index for minimum value of BIC function, BEST – the mean of the best Rand index).

Dim.	No. of clusters															
	2				3				4				5			
	MIN	MEAN	BIC	BEST	MIN	MEAN	BIC	BEST	MIN	MEAN	BIC	BEST	MIN	MEAN	BIC	BEST
3	0.64	0.76	0.95	0.96	0.74	0.82	0.89	0.92	0.79	0.86	0.91	0.94	0.80	0.86	0.88	0.92
4	0.59	0.71	0.93	0.95	0.70	0.80	0.94	0.95	0.77	0.85	0.94	0.96	0.80	0.86	0.92	0.94
5	0.55	0.68	0.93	0.95	0.69	0.80	0.96	0.97	0.76	0.84	0.95	0.96	0.78	0.86	0.96	0.97
6	0.52	0.66	0.88	0.93	0.68	0.78	0.93	0.96	0.75	0.84	0.95	0.96	0.79	0.87	0.96	0.97

where $\text{error}(x)$ is the error made by replacing the point by its representation (that is its cluster center): $\text{error}(x) = \|x - \text{mean}_{x_i}\|$, where i denotes the number of a cluster to which x belongs.

In our case for the analogue of likelihood function we use exactly formula (15), but with the error determined by the difference of the point and its projection on the respective subspace. To apply the proposed modification of BIC we still need to count the number of free parameters. For our algorithm the number of free parameters for each cluster depends on dimension of subspace. To calculate the coordinates of the projection of a point onto the affine space V of dimension n in $\mathbb{R}^N$, we need to fix the center of the coordinate system m and n base vectors $v_1, \dots, v_n$. Consequently, to be able to uniquely decode the point $x \in V$ from its coefficients, we need $N + n \cdot N = (n + 1)N$ parameters.

Summarizing, for k -clusters we obtain the formula:

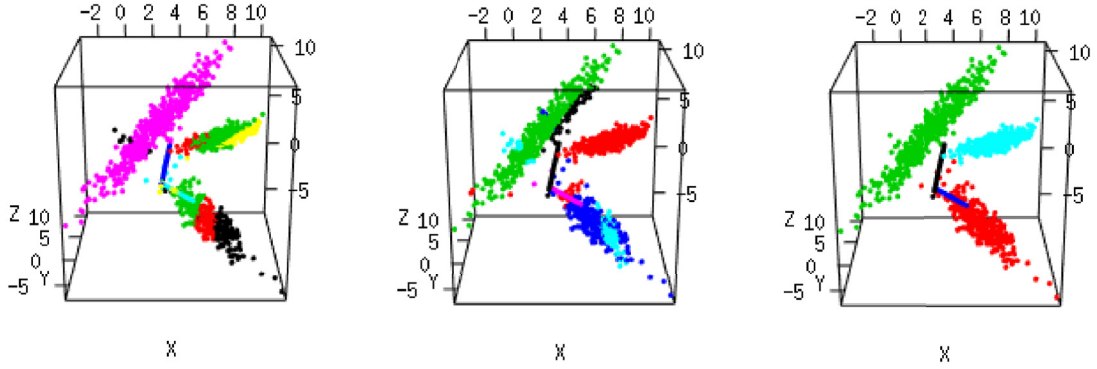
$$BIC = \sum_{i=1}^k \text{error}^2(X_i, V_i) + \sum_{i=1}^k (n_i + 1)N \cdot \ln(|X_i|), \quad (16)$$

where $n_i = \dim(V_i)$, V_i denotes the affine subspace which represents the cluster, and error^2 is the sum of squared errors for all points in the cluster.

Now we show how to find the optimal value of compression ratio parameter. For this purpose we will minimize the function BIC, see Fig. 3.

As we see the minimal value of BIC 224.96 recovers the original division of data (high Rand Index), see Fig. 4.

We made several experiments to verify our hypothesis of finding the optimal value of compression ratio r . We generated 50 random synthetic datasets for different dimensions (between 3 and 6) and numbers of clusters (from 2 to 5), see Table 1. In the experiment we used a similar approach as in Section 6.1.1. For these datasets we ran our method **SuMC_b** and calculated function BIC for each r from range $\frac{N}{N-1}, \dots, N$. We calculated the mean of: minimal Rand index, the mean of all Rand index, the mean of Rand index for minimal BIC value and the mean of the best Rand index, see Table 1.



(a) Number of clusters: 7, (b) Number of clusters: 6, (c) Number of clusters: 5,
BIC: 1089.9, Rand Index: 0.89. BIC: 451.1, Rand Index: 0.93. BIC: 224.96, Rand Index: 1.

Fig. 4. Figure shows clusters depending on the parameter r . The dataset consisted of three 2-dimensional subspaces (with 300, 400, 500 points chosen randomly from normal densities) and two of 1-dimensional subspaces (each with 200 points).

As one can see, the mean Rand index for BIC is better (in all considered cases) than mean value of Rand index, and it is very close to the optimal level. Therefore, BIC allows us to reasonably estimate parameter r .

5. Algorithm

In this section we present our algorithm, which can be divided into three phases: initialization phase, iterative phase and dimension refinement phase.

Initialization phase

At the beginning we initialize clusters by random assignment of points to clusters. Then, for each group we assign memory proportionally with respect to its size. For a detailed description of this step, suppose that we begin with k clusters $X_1, \dots, X_k$ such, that $X_1 \cup \dots \cup X_k = X \subset \mathbb{R}^N$ and parameter $r \geq 1$ (data compression ratio). For each of those clusters we reserve the amount of memory which is proportional to its number of elements

$$M_1 = \frac{N \cdot |X_1| \cdot b}{r}, \dots, M_k = \frac{N \cdot |X_k| \cdot b}{r},$$

where b is number of bits that need to represent one scalar (real number).

Iterative phase

In the second phase, we repeatedly go over all elements of the partition and switch points to those groups for which the compression error is minimal, for more details see [Algorithm 5.2](#). It is crucial to observe that if we change the position of an element from one cluster to another, we then must change the amount of memory reserved for these clusters. During this operation, we also update dimensions of these groups.

After this process, we reduce the number of groups. We usually delete clusters with a number of points below a given threshold (typically set at a fixed small percentage of the total amount of data set). In such situation we move both the memory and the points connected with the removed cluster to the randomly chosen cluster (typically we chose the first existing one). We repeat the second phase until the total error (10) is close to zero or when it stabilizes.

Now we present a more detailed description of a crucial method called “SwitchMembership”, which shows how to find the best cluster for point x of a data set (see [Algorithm 5.1](#) line 14).

Observe that in line 13 of [Algorithm 5.2](#), where we calculate the difference of errors before and after the change of position of point x , we use the same memory parameters M_i, M_j . We can do that because the total memory reserved for two clusters $M_i + M_j$ does not change during switching of the point x from X_i to X_j or vice versa. To calculate the difference of errors between two groups and update their memory allocation we use function ‘Error’ (see [Algorithm 5.3](#)), which solves Optimization Problem 4.1 for two clusters.

To find new dimensions of clusters X_1, X_2 for which we obtain minimal error (line 8 of [Algorithm 5.3](#)) we apply [Theorem 4.1](#) and equation (7). In this process we use the ordered eigenvalues corresponding to eigenvectors of covariance matrix $\text{cov}_{X_i}, \text{cov}_{X_j}$. If we find dimensions of clusters X_1, X_2 , we also have minimal error (line 9 of [Algorithm 5.3](#)) and new memory reserved for X_1, X_2 (line 10–11 of [Algorithm 5.3](#)). So [Algorithm 5.3](#) returns minimal error of two clusters, their new dimensions and memory allocations.

Dimension refinement phase

Finally, in the last phase we use [Theorem 4.2](#) in order to modify the dimensions of clusters so that all clusters, except for at most one, have integer dimension. From the previous stage we obtain k pairwise disjoint groups $X_1, \dots, X_k$, which divide

Algorithm 5.1 SuMC_b(Compression ratio: r , No. of bits: b , No. of clusters: k).

```

1:  $\{X \text{ is dataset consisting of points } x \in \mathbb{R}^N\}$ 
2:  $\{X_i \text{ is current cluster } i\}$ 
3:  $\{n_i \text{ is current dimension of } X_i\}$ 
4:  $\{M_i \text{ is current memory reserved for } X_i\}$ 

5: begin
6:   {1. Initialization Phase}
7:   Randomly generated clustering of  $X_i$ 
8:   for  $i = 1, \dots, k$  do
9:      $M_i = N \cdot |X_i| \cdot b/r$ 

10:  {2. Iterative Phase}
11:  repeat
12:    for each  $x \in X$  do
13:      if  $x$  does not belong to any cluster, assign it to  $X_1$ 
14:      SwitchMembership( $x, M_1, \dots, M_k, X_1, \dots, X_k$ )
15:      Delete clusters which have number of points below a-given threshold
16:  until (termination_criterion)

17:  {3. Refinement Phase}
18:   $L = \text{List}(1, \dots, k)$ 
19:  while ( $\text{size}(L) > 1$ ) do
20:    {take the~first two elements of  $L$ }
21:     $i = L[0], j = L[1]$ 
22:    {Modification of dimensions and memory reserved of clusters}
23:     $(M_i, M_j, n_i, n_j) = \text{Error}(M_i, M_j, X_i, X_j)$ 
24:    if  $n_i \in \mathbb{N}$  then
25:       $L.\text{remove}(i)$ 
26:    if  $n_j \in \mathbb{N}$  then
27:       $L.\text{remove}(j)$ 

28:  return ( $n_1, \dots, n_k, X_1, \dots, X_k$ )
29: end

```

data set $X = X_1 \cup \dots \cup X_k$. At this stage we do not modify this clustering, i.e. we do not change the position of any point from the data set. We modify only the dimensions (and memory, which is associated with it) of these clusters, taking successive groups, and calculate their minimum error (to calculate error we apply [Theorem 4.1](#)). Then, we obtain new dimensions of these clusters. At least one of them has non integer dimension.

At the end of this section we discuss computational complexity of our method. We estimate it separately in each phase of [Algorithm 5.1](#). In the first one – *initialization*, we need the following operations:

- Random generation of clustering – $O(|X|)$, see [\[16, Sec 3.4.1.A\]](#) and [\[34\]](#),
- Computation of basic characteristics of the clusters (covariance matrix, mean of cluster, eigenvalues, weight, memory) – $O(k \cdot N^{2+\gamma})$.⁷

In the case of space complexity we need $O(|X|)$ for remembering labels and $O(k \cdot N^2)$ for parameters, which describe clusters. Therefore, we have $O(\max(|X|, k \cdot N^{2+\gamma}))$ time complexity and $O(k \cdot N^2)$ space complexity.

In the *iterative phase* we need the following operations (*SwitchMembership* function):

- Copying of k clusters (X_i^- and X_i^+ for $i = \{1, \dots, k\} \setminus \{l\}$, see lines 5–7 [Algorithm 5.2](#)) – $O(k \cdot N^2)$ time and space complexity;
- Switching the position x to other clusters, see [Algorithm 5.2](#) lines 5–7 – updating mean of cluster $O(N)$ and covariance matrix (symmetric rank- k update to a matrix) $O(N^2)$, updating eigenvalues $O(N^{2+\gamma})$. So we need $O(k \cdot N^{2+\gamma})$ operations;
- Finding an optimal cluster for a point (function *Error*) – all operations in [Algorithm 5.3](#) (ordered eigenvalues were calculated in the previous point) expect line 8 to have $O(1)$ complexity; to find new dimensions of clusters we theoretically

⁷ The naive matrix multiplication gives $\gamma = 1$, the applied algorithms have $\gamma \approx 0.8$, there are some theoretical results which show that $\gamma \leq 0.36$, while there still is an open hypothesis whether γ can be chosen arbitrarily close to zero.

Algorithm 5.2 SwitchMembership($x, M_1, \dots, M_k, X_1, \dots, X_k$).

```

1: { $l$  is cluster in which belongs point  $x$ }
2: { $M_i$  is current memory reserved for  $X_i$ }

3: begin
4:   {Switch position  $x$  form cluster where  $a$ -belongs to other clusters}
5:    $X_l^- = X_l \setminus \{x\}$ 
6:   for  $i = \{1, \dots, k\} \setminus \{l\}$  do
7:      $X_i^+ = X_i \cup \{x\}$ 
8:    $j = l$ 
9:   differenceErrors = 0
10:  {Find  $a$ -cluster, which gives us the-smallest error}
11:  for  $i = \{1, \dots, k\} \setminus \{l\}$  do
12:    {Calculate the-difference of errors}
13:    tempError = Error( $M_i, M_l, X_i^+, X_l^-$ ) – Error( $M_i, M_l, X_i, X_l$ )
14:    if tempError < differenceErrors then
15:       $j = i$ 
16:      differenceErrors = tempError
17:  {If we obtain smaller error by shift of point  $x$ , then we update clusters and their memories}
18:  if  $j \neq l$  then
19:     $X_l = X_l^-, X_j = X_j^+$ 
20:    Changing the-amount of memory allocated to clusters  $X_l, X_j$ 
21:  return ( $M_1, \dots, M_k, X_1, \dots, X_k$ )
22: end

```

Algorithm 5.3 Error(M_1, M_2, X_1, X_2).

```

1: { $b$  is number of bits that need to represent one scalar}
2: { $n_i$  is new dimension of  $X_i$ }
3: { $X \subset \mathbb{R}^N$  is dataset}

4: begin
5:   {Calculate number of scalars}
6:    $R = (M_1 + M_2 + |X_1| \cdot \log_2 \frac{|X_1|}{|X|} + |X_2| \cdot \log_2 \frac{|X_2|}{|X|}) / b$ 
7:   Calculate ordered eigenvalues  $\lambda_i^{(1)}, \lambda_i^{(2)}$  of covariance matrices  $\text{cov}_{X_1}$  and  $\text{cov}_{X_2}$ 
8:   Find new dimensions  $n_1, n_2$  of  $X_1, X_2$  for which we obtain minimal error under condition  $n_1 \cdot |X_1| + n_2 \cdot |X_2| = R$ 
9:   minError = calculate from equation (??)
10:   $M_1 = (n_1 \cdot b - \log_2 \frac{|X_1|}{|X|}) \cdot |X_1|$ 
11:   $M_2 = R - M_1$ 
12:  return (minError,  $M_1, M_2, n_1, n_2$ )
13: end

```

need $O(N)$ operations, but since experiments showed that the dimensions of these clusters do not change more than twice, we need $O(1)$ in this step;

- Coping cluster's attributes from X_l^-, X_j^+ to X_l, X_j , respectively if we obtain smaller error by shift of point $x - O(N^2)$.

Consequently, we need $O(k \cdot |X| \cdot N^{2+\gamma})$ operations in *iterative phase* and $O(k \cdot N^2)$ space. Similarly to the k -means, the amount of iterations in our algorithm cannot be theoretically estimated, but experiments showed that to stabilize the cost function, we need at most 50 iterations. In the last phase – *refinement*, we need $O(k)$ operations.

Summarizing, the numerical complexity of the algorithm described in this section is linear due to the number of clusters, number of points in dataset and at most cubic due to the dimension of data ($O(k \cdot |X| \cdot N^{2+\gamma})$). Moreover, the space complexity is $O(k \cdot N^2)$.

6. Experiments

In this section, we evaluate our method implemented in C++ language. All experiments were run on Ubuntu 14.04 (64-bit) workstation with a 3.3 GHz Quad-Core Intel Xeon Processor and 32GB RAM.

Table 2
Methods and their parameters.

Methods	Parameters	Range
SuMC _∞	r – compression ratio	$r \in \mathbb{R} \wedge r \geq 1$
	k – number of clusters	$k \in \mathbb{N} \wedge k \geq 1$
SuMC _b	r – compression ratio	$r \in \mathbb{R} \wedge r \geq 1$
	b – number of bits	$b \in \{4, 8, 16, 32, 64\}$ We put $b = 4$.
ORCLUS	k – number of clusters	$k \in \mathbb{N} \wedge k \geq 1$
4C	l – the dimension of cluster-specific subspaces (equal for all clusters)	$l \in \mathbb{N}$
	ε – the size of the local areas	$\varepsilon \in \mathbb{R}$
	μ – the number of neighbors	$\mu \in \mathbb{N}$
	λ – the correlation dimension	$\lambda \in \mathbb{N}$
	δ – influences the tightness of the detected correlations	$\delta \in [0, 1]$

Table 3

The results of comparison between SuMC_∞ and ORCLUS on five-dimensional synthetic data sets. We randomly generated 400 datasets for which SuMC_∞ method won 331, and lost 66 times for Rand measure (330 to 67 for Jaccard measure).

Dim. of data	Similarity measure	SuMC _∞			Mean	Method
		Better	Equal	Worse		
1	Rand	97	0	3	0.650	ORCLUS
					0.860	SuMC _∞
	Jaccard	97	0	3	0.284	ORCLUS
					0.672	SuMC _∞
2	Rand	69	1	30	0.958	ORCLUS
					0.918	SuMC _∞
	Jaccard	69	1	30	0.878	ORCLUS
					0.812	SuMC _∞
3	Rand	82	2	16	0.895	ORCLUS
					0.948	SuMC _∞
	Jaccard	81	2	17	0.709	ORCLUS
					0.856	SuMC _∞
4	Rand	83	0	17	0.721	ORCLUS
					0.879	SuMC _∞
	Jaccard	83	0	17	0.411	ORCLUS
					0.682	SuMC _∞

In the literature, there exist many different subspace clustering approaches. We decided to compare our method with ORCLUS [4] and 4C [7], which have the most similar features and properties to SuMC. At the beginning, we compare SuMC_∞ with ORCLUS, which detects clusters in arbitrary oriented subspaces. In the next part of this section we present the results of comparing two density-based methods – SuMC_b and 4C. To compare these methods, we use Weka⁸ Software, which is implemented in Java and contains both algorithms ORCLUS and 4C.

All of the above methods have several parameters, see Table 2. We will describe them in more detail below.

6.1. Comparison between SuMC_∞ and ORCLUS

One of the well-known methods that deals with the projected clustering is ORCLUS [4]. We compare this method with ours on both synthetic and well-known datasets from the UCI Repository. Before we show our results, we will recall the basic information about these methods. Both methods detect clusters in arbitrarily-oriented subspaces. Moreover, their algorithms require similar parameters: number of clusters and joint dimension (ORCLUS) or total number of parameters (SuMC_∞). The difference between these methods is that ORCLUS divides a dataset into a given number of clusters of

⁸ Weka is a collection of machine learning algorithms for data mining tasks, which contains, inter alia, tools for clustering, www.cs.waikato.ac.nz/ml/weka.

Table 4

Summary results of comparison between **SuMC_∞** and ORCLUS on synthetic data sets. We randomly generated 7400 datasets for which **SuMC_∞** method won 6182, and lost 370 times for Rand measure (6184 to 373 for Jaccard measure).

Dim. of data	Similarity measure	SuMC _∞			Mean	Method
		Better	Equal	Worse		
3	Rand	157	1	42	0.765	ORCLUS
					0.876	SuMC _∞
	Jaccard	158	1	41	0.470	ORCLUS
					0.690	SuMC _∞
4	Rand	213	27	60	0.803	ORCLUS
					0.892	SuMC _∞
	Jaccard	216	25	59	0.561	ORCLUS
					0.721	SuMC _∞
5	Rand	331	3	66	0.806	ORCLUS
					0.901	SuMC _∞
	Jaccard	330	3	67	0.571	ORCLUS
					0.756	SuMC _∞
6	Rand	348	78	74	0.826	ORCLUS
					0.923	SuMC _∞
	Jaccard	347	77	76	0.610	ORCLUS
					0.805	SuMC _∞
7	Rand	537	16	47	0.814	ORCLUS
					0.945	SuMC _∞
	Jaccard	537	16	47	0.588	ORCLUS
					0.852	SuMC _∞
8	Rand	552	107	41	0.823	ORCLUS
					0.951	SuMC _∞
	Jaccard	553	107	40	0.620	ORCLUS
					0.882	SuMC _∞
9	Rand	724	52	24	0.834	ORCLUS
					0.962	SuMC _∞
	Jaccard	724	52	24	0.630	ORCLUS
					0.901	SuMC _∞
10	Rand	373	20	7	0.858	ORCLUS
					0.981	SuMC _∞
	Jaccard	372	20	8	0.673	ORCLUS
					0.950	SuMC _∞
12	Rand	373	121	6	0.873	ORCLUS
					0.985	SuMC _∞
	Jaccard	373	120	7	0.718	ORCLUS
					0.967	SuMC _∞
14	Rand	522	77	1	0.856	ORCLUS
					0.990	SuMC _∞
	Jaccard	522	77	1	0.666	ORCLUS
					0.976	SuMC _∞
16	Rand	560	139	1	0.855	ORCLUS
					0.990	SuMC _∞
	Jaccard	561	138	1	0.676	ORCLUS
					0.975	SuMC _∞
18	Rand	734	66	0	0.872	ORCLUS
					0.993	SuMC _∞
	Jaccard	734	66	0	0.704	ORCLUS
					0.981	SuMC _∞
20	Rand	758	141	1	0.868	ORCLUS
					0.988	SuMC _∞
	Jaccard	757	141	2	0.704	ORCLUS
					0.972	SuMC _∞

the same dimensions, while our method switches the memory between clusters, which results in the automatic discovery of the optimal dimension of each cluster. It should be highlighted that our method needs compression rate parameter $r \geq 1$ directly associated with the dimensions of clusters. In this part of subsection, we use method **SuMC_∞**, because we want to return a fixed number of clusters, similarly to ORCLUS.

6.1.1. Synthetic datasets

Let us start with synthetic datasets, which we randomly generated on the spaces $[0, 1]^{\text{dim}}$, where dim denotes dimension of a space. We generated 100 random synthetic datasets for each pair (dimension of data, dimension of clusters), and for them, we calculated the mean of Rand and Jaccard Index; see [Tables 3 and 4](#) (all clusters have the same dimension, because ORCLUS method looks only for groups with the same dimensions). Each time the number of clusters was given as a random integer between 2 and 5. First parameter for both of these methods is the same: numbers of clusters. The second parameter for ORCLUS is the dimension of clusters, but for **SuMC_∞** it is the data compression ratio, which is calculated by the formula [\(11\)](#):

$$r = \frac{\text{dimension of data}}{\text{dimension of cluster}}.$$

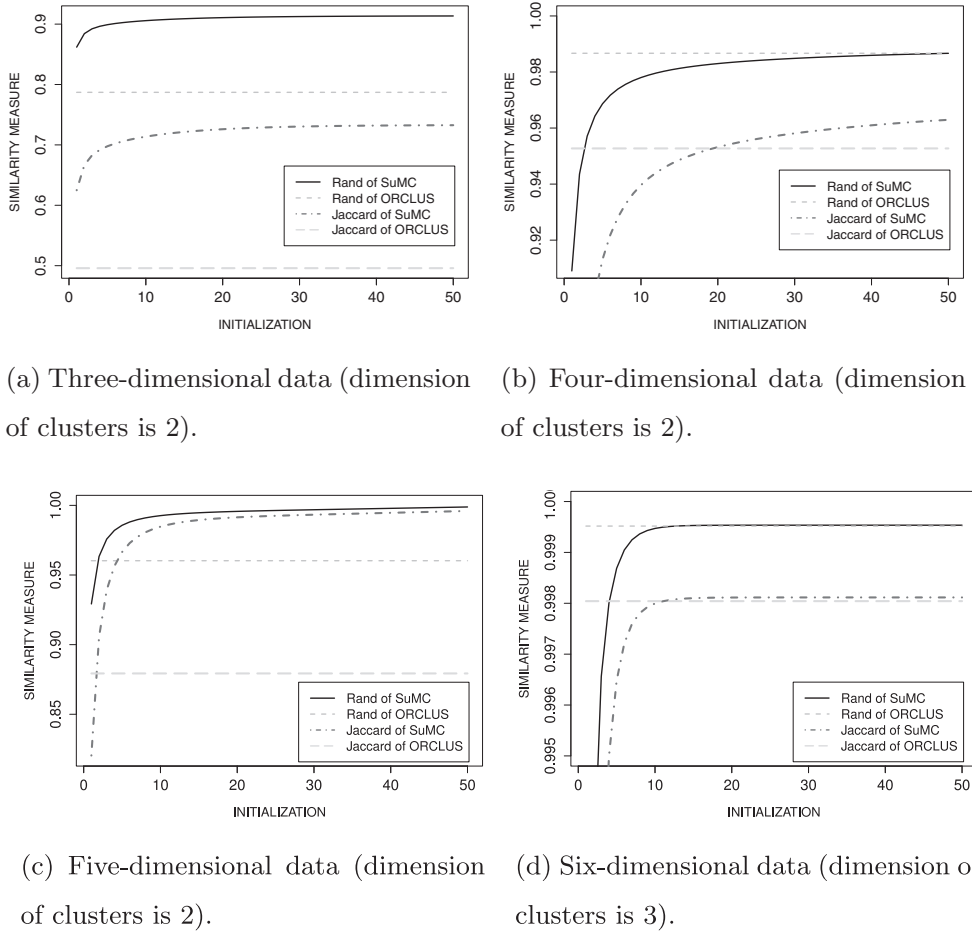


Fig. 5. Plot of mean Rand and Jaccard Index with the number of launch methods SuMC_∞ and ORCLUS on the same synthetic data sets.

In Table 3, we present the number of wins (better results than the competition) for these 100 random datasets. Due to the size of the results, only one of them is given.

In Table 4, we present the summary results for each dimensions from the set $\{3, 4, \dots, 10, 12, 14, 16, 18, 20\}$ of data sets. Similarly as before, we generate 100 random datasets for each pair (dimension of data, dimension of clusters), and for them, we calculate the mean of Rand and Jaccard Index. For dimensions less than 10, we take all dimensions of clusters (for example, for nine-dimensional data we take $1, \dots, 8$ dimensions), but for the rest, we took every second (for example, for 14-dimensional data we take $2, 4, \dots, 12$ dimensions of clusters). In addition, we performed the same experiment for a dataset with added error and its similar results to those shown in Tables 3 and 4 were returned. Therefore, in this paper we have shown only some of them.

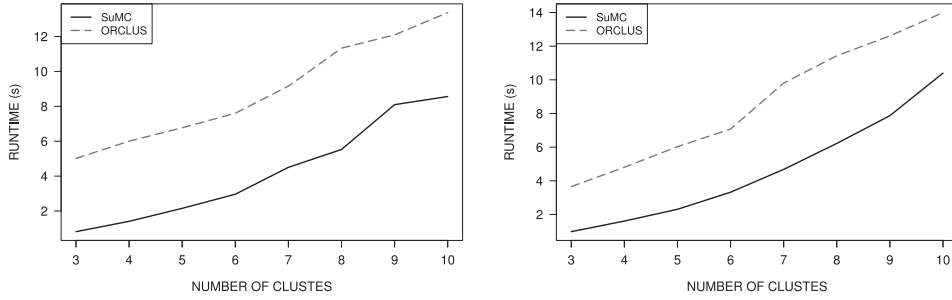
As can be seen from Table 4, our method almost always obtains better results than the ORCLUS. An additional advantage of our method is the fact that it can be run several times on the same dataset and selects the best result, because we minimize the cost function defined by (10).

In the next experiment, we investigate these properties in a more detailed way. Similarly to the previous experiments, we generated 100 datasets with the difference that for each dataset we run both methods 50 times. We consider only data of dimensions $\{3, 4, 5, 6\}$, because we obtained worse results in these dimensions than in higher dimensions. In Fig. 5 we present some typical plots of mean Rand and Jaccard Index with the number of launches of both methods on the same data sets.

The remaining cases are similar to Fig. 5(a) and (b). Observe that Rand and Jaccard Index stabilize after a certain number of runs of method SuMC_∞ .

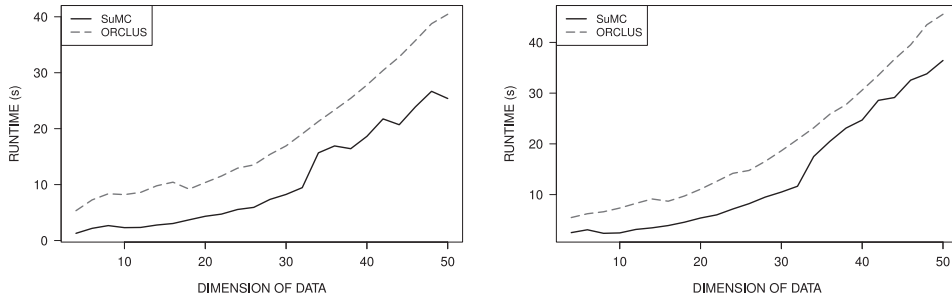
6.1.2. The computational complexity

The computational complexity with respect to the number of data points as well as the dimensionality of the data space is an important issue. Therefore, in the next experiment, we study the time complexity of the methods. We have analyzed the complexity of these methods with respect to the number of clusters, the dimension of space, and the number of points.



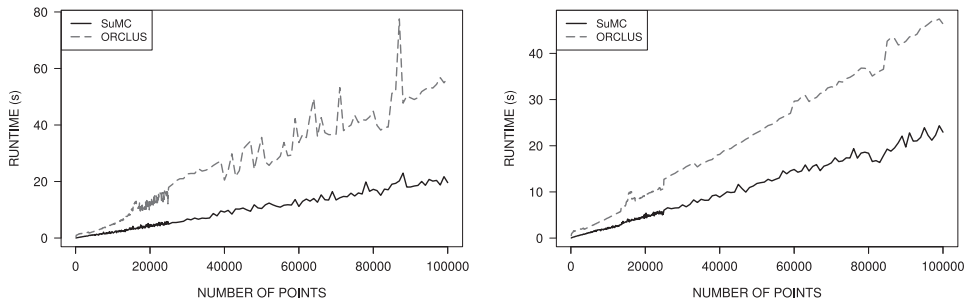
(a) Dimension of every cluster is 3.

(b) Dimension of every cluster is 6.

Fig. 6. Plot of median time complexity of methods SuMC_∞ and ORCLUS in relation to the number of clusters.

(a) Dimension of every cluster is 3.

(b) Dimension of every cluster is 6.

Fig. 7. Plot of median time complexity of methods SuMC_∞ and ORCLUS in relation to the dimension of data sets.

(a) Dimension of every cluster is 3.

(b) Dimension of every cluster is 6.

Fig. 8. Plot of median time complexity of methods SuMC_∞ and ORCLUS in relation to the number of points in data sets.

Fig. 6 presents the median time complexity of both methods in relation to the number of clusters $\{3, 4, \dots, 10\}$. For each number of clusters, we generate 100 different data sets (each consisting of 10000 points) from 10-dimensional space.

Similarly as before, we generated 100 different data sets (each consisting of 10000 points) from each $\{4, \dots, 50\}$ -dimensional space, see Fig. 7.

Fig. 8 shows time complexity according to the number of points in data sets. In this case, we generate 50 different datasets from 10-dimensional space for each point from set $\{100, \dots, 100000\}$.

As one can see, the computational complexity is similar in both methods with a small advantage of our approach.

In general, our method has the linear computational complexity in respect to number of points, and the cubic complexity in respect to data dimensions (due to the computational complexity of the problem of calculating the eigenvalues).

6.1.3. Real dataset

Now we compare both methods – SuMC_∞ – and ORCLUS on gas sensor, gesture, glass, seeds, wine, and yeast datasets from the UCI Repository as well as on the MNIST database; see Table 5.

Table 5

Comparison between **SuMC_∞** and ORCLUS on data sets from the UCI Repository and the MNIST database. For 45 cases (without MNIST dataset) **SuMC_∞** method won 30, and lost 15 times for Rand measure (22 to 23 for Jaccard measure). ORCLUS for MNIST data did not work.

Name of data	No. of clusters	Dim. of clusters	Compression ratio	Rand Index		Jaccard coefficient	
				ORCLUS	SuMC _∞	ORCLUS	SuMC _∞
Gas Sensor	5	10	12.8	0.841	0.701	0.475	0.191
		30	4.27	0.844	0.842	0.488	0.495
		50	2.56	0.782	0.837	0.408	0.481
		70	1.83	0.765	0.812	0.390	0.463
		90	1.42	0.830	0.753	0.494	0.341
		110	1.16	0.392	0.659	0.224	0.144
Gesture	5	5	6.4	0.551	0.652	0.212	0.210
		10	3.2	0.545	0.643	0.207	0.255
		15	2.13	0.579	0.669	0.198	0.193
		20	1.6	0.547	0.674	0.189	0.231
		25	1.28	0.620	0.672	0.184	0.217
		30	1.07	0.629	0.648	0.161	0.226
Glass	6	1	9	0.485	0.676	0.191	0.255
		2	4.5	0.495	0.678	0.291	0.246
		3	3	0.601	0.672	0.221	0.143
		4	2.25	0.623	0.671	0.242	0.156
		5	1.8	0.663	0.621	0.212	0.176
		6	1.5	0.663	0.660	0.186	0.154
		7	1.29	0.661	0.644	0.210	0.187
		8	1.12	0.688	0.661	0.230	0.160
Seeds	10	1	7	0.578	0.600	0.240	0.264
		2	3.5	0.514	0.624	0.260	0.277
		3	2.33	0.540	0.571	0.210	0.220
		4	1.75	0.580	0.616	0.250	0.278
		5	1.4	0.584	0.555	0.249	0.197
		6	1.17	0.618	0.733	0.271	0.433
Wine	3	1	13	0.579	0.505	0.234	0.259
		2	6.5	0.548	0.575	0.234	0.262
		3	4.33	0.565	0.649	0.236	0.360
		4	3.25	0.578	0.634	0.230	0.295
		5	2.6	0.552	0.590	0.200	0.242
		6	2.17	0.556	0.557	0.218	0.219
		7	1.86	0.552	0.569	0.237	0.216
		8	1.62	0.506	0.568	0.262	0.215
		9	1.44	0.562	0.558	0.272	0.206
		10	1.3	0.621	0.565	0.349	0.215
		11	1.18	0.644	0.568	0.306	0.233
		12	1.08	0.519	0.544	0.260	0.217
Yeast	10	1	8	0.474	0.731	0.248	0.144
		2	4	0.495	0.729	0.194	0.112
		3	2.67	0.675	0.714	0.112	0.106
		4	2	0.684	0.688	0.111	0.113
		5	1.6	0.691	0.634	0.108	0.129
		6	1.33	0.683	0.626	0.135	0.145
		7	1.14	0.715	0.661	0.123	0.122
		100	7.84	—	0.904	—	0.364
		200	3.92	—	0.878	—	0.268
		300	2.61	—	0.834	—	0.177
		400	1.94	—	0.627	—	0.109
		500	1.57	—	0.659	—	0.102
		600	1.31	—	0.820	—	0.053
		700	1.12	—	0.820	—	0.053

From the above table, we can conclude that both methods give similar results. For gas sensor set, we obtained 3 better results of Rand index and 3 worse. For gesture set, we always obtained better results than ORCLUS. For glass set, we obtained 4 better results of Rand index and 4 worse. For seeds set, we obtained 5 better results and 1 worse. For wine set, we obtained 8 better results of Rand index and 4 worse. In the last set, we obtained 4 better results and 3 worse. In the case of the Jaccard Index the results obtained, respectively, for data sets are as follows: 3 to 3, 4 to 2, 1 to 7, 5 to 1, 5 to 7, and 3 to 4. In the case of the MNIST data we did not obtain results for ORCLUS because it is not very scalable for high-dimensional data.

6.2. Comparison between **SuMC₆** and 4C

To the best of our knowledge, both concepts of density-based clustering (i.e., finding densely populated subsets of the data) and correlation analysis have not yet been addressed as a combined task for data mining. The most relevant

Table 6

The results of algorithm 4C with parameters: $\delta = 0.1$, λ is default, and ε , μ as in the table (left table) and the results of algorithm **SuMC_b** with parameters: r – data compression ratio and b – numbers of bits dedicated to represent a real number (right table). For each parameter, we calculate the mean of Rand and the average error of original number of clusters in data set.

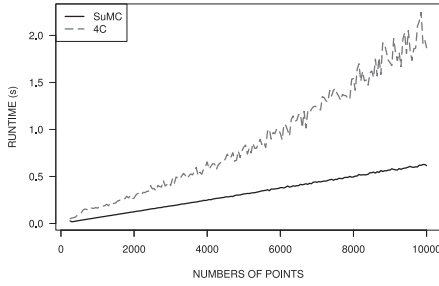
$\varepsilon \backslash \mu$	5	6	7	8	9	10		$r \backslash b$	2	4	8	16	32	64	
10	0.4856	0.4762	0.4611	0.462	0.451	0.4395	Rand	10	0.7238	0.8109	0.8279	0.8233	0.8198	0.8181	Rand
	7.165	5.233	4.422	4.09	4.155	4.233	Error		4.862	2.338	3.98	4.072	4.121	4.229	Error
20	0.6188	0.6033	0.6009	0.585	0.5868	0.5775	Rand	5	0.8218	0.8235	0.8182	0.8222	0.8263	0.8264	Rand
	4.826	3.647	3.221	3.209	3.284	3.442	Error		2.211	3.646	4.245	4.268	4.26	4.308	Error
30	0.7104	0.7034	0.7001	0.7026	0.6901	0.6853	Rand	3.333	0.8311	0.8417	0.8447	0.8506	0.8489	0.8523	Rand
	5.555	2.955	2.59	2.571	2.607	2.737	Error		2.926	3.739	3.998	4.043	4.115	4.108	Error
40	0.7886	0.7796	0.7796	0.7748	0.7746	0.76	Rand	2.5	0.8571	0.8731	0.8822	0.8869	0.8863	0.8917	Rand
	5.73	2.893	2.213	2.083	2.028	2.195	Error		2.857	3.447	3.524	3.531	3.573	3.5	Error
50	0.8238	0.8229	0.8147	0.8194	0.8189	0.8081	Rand	2	0.8944	0.9058	0.9033	0.9024	0.9032	0.9041	Rand
	3.904	2.31	1.915	1.719	1.655	1.685	Error		2.221	2.724	2.79	2.853	2.841	2.877	Error
60	0.8216	0.8202	0.8164	0.8172	0.8185	0.8172	Rand	1.667	0.9196	0.9068	0.9008	0.893	0.8882	0.8848	Rand
	2.297	1.547	1.387	1.335	1.376	1.266	Error		1.746	2.366	2.25	2.462	2.481	2.509	Error
70	0.7836	0.7891	0.7766	0.792	0.7873	0.791	Rand	1.429	0.8984	0.8773	0.8578	0.8524	0.8473	0.8484	Rand
	1.698	1.49	1.337	1.268	1.269	1.235	Error		1.625	2.031	2.202	2.325	2.519	2.478	Error
80	0.7229	0.7105	0.7165	0.7243	0.7265	0.7269	Rand	1.25	0.8576	0.8233	0.8082	0.8049	0.8009	0.8005	Rand
	1.7	1.682	1.631	1.61	1.54	1.573	Error		1.591	2.128	2.463	2.649	2.709	2.689	Error
90	0.6401	0.6415	0.6408	0.6517	0.6497	0.6569	Rand	1.111	0.7855	0.7767	0.7643	0.7524	0.7526	0.7494	Rand
	2.111	2.044	2.05	2.009	2.054	1.988	Error		1.793	2.276	2.266	2.355	2.419	2.518	Error
100	0.5575	0.5585	0.5647	0.5701	0.567	0.5781	Rand	1	0.6941	0.6896	0.66	0.6383	0.5952	0.5423	Rand
	2.614	2.611	2.559	2.553	2.555	2.465	Error		2.176	2.011	2.263	3.504	3.595	3.618	Error

approach is ORCLUS. In this part of the paper, we compare our method **SuMC_b** with 4C [7], which is capable of detecting local subsets of the data that exhibit strong correlations and are densely populated (a correlation connected cluster). In contrast to ORCLUS, in the algorithm 4C, we do not fix the number of clusters. Clusters grow from a seed as long as a density criterion is fulfilled. Otherwise, another seed is picked to start a new cluster. The density criterion is a required minimal number of points within the neighborhood of a point, where the neighborhood is based on the distance matrices computed from the eigensystems of two points. See [7] for more details.

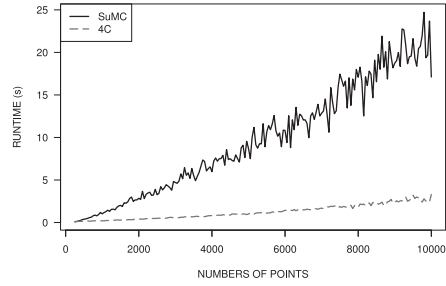
The algorithm 4C needs four input parameters: ε , μ , λ , and δ . The parameter $\varepsilon \in \mathbb{R}$ specifies the size of the local areas in which the correlations are examined, and thus determines the number of objects that contribute to the covariance matrix and consequently to the correlation similarity measure of each object. Quoting after paper [7]: “The choice of ε has two aspects. First, it should not be too small, because in that case an insufficient number of objects contribute to the correlation similarity measure of each object, and thus this measure can be meaningless. On the other hand, ε should not be too large, because then some noise objects might be correlation reachable from objects within a correlation connected cluster.” The parameter $\mu \in \mathbb{N}$ (should be greater than or equal to 5) specifies the number of neighbors an object must find in an ε -neighborhood and in a correlation ε -neighborhood to exceed the density threshold. It determines the minimum cluster size. The parameter $\lambda \in \mathbb{N}$ specifies the correlation dimension of the correlation-connected clusters to be computed. The parameter $\delta \in [0, 1]$ influences the tightness of the detected correlations, that is, how much local variance from the correlation is allowed.

Observe that our method needs only two parameters, but 4C method requires four. Moreover, both methods have completely different parameters. Thus, we present the results in two separate tables. For 4C method, we set parameter $\delta = 0.1$ and λ as default value. We change only parameters ε , μ ; see Table 6. In contrast to ORCLUS, both methods can find optimal numbers of clusters in data sets. In this example, we compare methods **SuMC_b** and 4C in terms of clustering data and the detected numbers of clusters. The error is measured by the square root of the average of the squared deviations of the values from their original value.

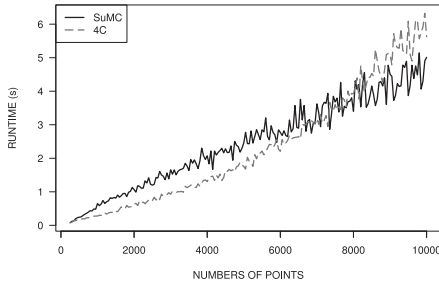
For each pair (ε, μ) , we create 50 different synthetic datasets from 10-dimensional space and run method 4C on these data. For the same datasets, we evaluate our method with parameters r and b . The results of these experiments are presented in Table 6.



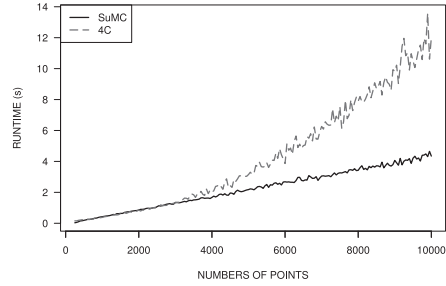
(a) For algorithm 4C parameters $\varepsilon = 10$, $\mu = 5$, and for **SuMC_b** $r = 10$, $b = 2$.



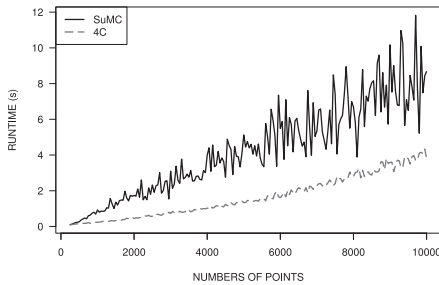
(b) For algorithm 4C parameters $\varepsilon = 30$, $\mu = 8$, and for **SuMC_b** $r = 3.3$, $b = 16$.



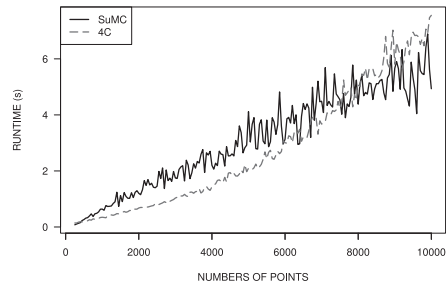
(c) For algorithm 4C parameters $\varepsilon = 70$, $\mu = 7$, and for **SuMC_b** $r = 1.42$, $b = 8$.



(d) For algorithm 4C parameters $\varepsilon = 100$, $\mu = 10$, and for **SuMC_b** $r = 1$, $b = 64$.



(e) For algorithm 4C parameters $\varepsilon = 50$, $\mu = 6$, and for **SuMC_b** $r = 2$, $b = 4$.



(f) For algorithm 4C parameters $\varepsilon = 80$, $\mu = 9$, and for **SuMC_b** $r = 1.25$, $b = 32$.

Fig. 9. Comparing time complexity between **SuMC_b** and 4C.

In Table 6 we present comparison of each pair (ε, μ) with (r, b) . We obtain 59 better results and 1 worse for Rand Index; and 42 wins to 18 losses for the average error of numbers of clusters. Looking globally at Table 6, we can conclude that our method gives in most cases better results than 4C method.



Fig. 10. Original images (481×321 , color) and segmentation results produced by our method (k is the number of clusters and r is the data compression ratio).



Fig. 11. Original images.

6.2.1. The computational complexity

Fig. 9 shows the comparison of both methods in terms of running time and number of points in dataset. We create 50 synthetic data sets from 10-dimensional space for each of the numbers of points of data set and run both methods (on the same data).

Results presented in Fig. 9 do not allow us to state which method has better running time. As one can see, the time is dependent on the parameters used for both methods.

7. Image segmentation and compression

In this section, we show some possible applications of our algorithm in image segmentation and image compression. The aim of this section is not to present some new important results, but rather to indicate a possible research direction regarding application of SuMC.

Table 7

results of images compression of Fig. 11 by algorithms: SuMC_b (with parameters $b = 4$ bits and others as in the table, where r is the data compression ratio and k is the number of clusters), PCA (with parameter r), k -means clustering method with PCA (with parameters k and r , which describe dimensions of all clusters) and JPEG compression method (with parameter r).

		k r	2	4	8	16	32	64			k r	2	4	8	16	32	64
Baboon	PCA		13.51	44.51	94.75	159.30	221.94	292.93	Lena	PCA		1.82	4.95	10.21	20.57	37.47	68.99
	JPEG		49.14	105.69	165.56	205.90	257.73	294.02		JPEG		4.10	10.83	21.29	44.67	63.80	106.03
	k -means & PCA	2	12.37	43.18	93.47	153.42	220.22	276.62		k -means & PCA	2	1.54	4.54	9.64	19.24	35.19	66.63
	5	10.48	38.17	87.23	148.24	207.93	271.53	5		1.26	4.04	8.90	18.26	33.51	64.68		
		10	8.02	33.56	81.27	140.32	199.89	257.88			10	0.92	3.36	7.80	15.98	30.94	55.95
		15	6.40	29.11	74.17	133.04	191.00	243.59			15	0.73	2.88	6.87	14.67	28.50	51.33
	SuMC _b	2	10.19	33.84	72.36	129.44	194.33	258.78		SuMC _b	2	1.43	3.65	6.52	12.08	23.55	46.57
	5	7.48	26.31	59.16	109.31	175.89	242.57	5		1.06	3.17	5.65	11.01	20.70	42.94		
	10	6.37	25.09	59.14	107.75	172.15	242.34	10		0.94	3.03	5.65	10.11	20.44	43.13		
	15	5.97	24.09	57.43	107.11	172.23	242.05	15		0.94	2.92	5.45	10.11	20.46	41.50		
	PCA		0.97	4.54	12.91	29.73	56.01	110.93		PCA		3.75	9.85	19.02	34.85	67.59	141.41
	JPEG		3.84	12.22	28.25	51.05	85.59	111.53		JPEG		7.45	16.17	32.99	57.61	105.18	156.18
Fruits	k -means & PCA	2	0.78	3.92	11.75	27.39	54.49	98.55	Peppers	k -means & PCA	2	3.27	9.01	18.01	33.05	62.48	128.74
	5	0.59	3.31	10.55	24.75	50.62	87.97	5		2.46	7.66	15.94	29.59	56.77	111.45		
	10	0.43	2.57	8.32	21.33	44.73	80.21	10		1.75	6.06	13.50	25.61	49.12	88.27		
	15	0.33	1.99	7.25	18.65	39.81	70.05	15		1.46	5.01	11.59	23.02	43.01	80.30		
	SuMC _b	2	0.64	2.39	6.12	13.10	28.19	66.59		SuMC _b	2	2.71	6.78	12.36	21.24	40.35	82.95
	5	0.54	1.88	5.21	11.42	24.68	66.58	5		1.95	5.74	11.24	18.51	34.40	74.98		
	10	0.54	1.82	4.80	11.40	24.69	63.26	10		1.68	5.72	10.67	17.72	33.15	74.97		
	15	0.54	1.82	4.79	10.67	24.70	60.78	15		1.68	5.43	10.21	17.67	32.82	75.46		

7.1. Image segmentation

In the context of image segmentation, as is common, we represent the data as vectors in $\mathbb{R}^5$, where three first coordinates correspond to a color in RGB format and the last two are pixel positions in the image. In our experiments, we use data repository [19] that contains images with two basic elements: background and foreground. It is difficult to determine these two classes directly. Our goal here is to demonstrate that our method can be applied to finding visually appealing structures in real data. However, we emphasize that it does not provide a complete solution to practical problems. Such a solution usually entails a significant amount of domain-specific knowledge and engineering. Nevertheless, based on these preliminary results with image data, we believe that **SuMC** provides an important insight into generic problem of clustering of real data sets.

Fig. 10 shows segmentation of several images from the Berkeley segmentation database via our algorithm with various parameters. As we can see the results of segmentation are promising – we are able to detect foreground objects with a single cluster, see Fig. 10(b), (e) and (j), or recognize details in the picture by increasing the number of clusters, see Fig. 10(c), (g), and (k).

7.2. Image compression

Now we proceed to discuss possible applications of **SuMC** in image compression. We present the results of numerical experiments illustrating the performance of the lossy image compression based on **SuMC** with the use of different parameters.

To compare our method with classical approaches, we consider several images, which are usually used in image compression, see Fig. 11. Images are represented in YCbCr color spaces and divided into disjoint squares of 8×8 pixels. As a result, we obtain data in $\mathbb{R}^{192}$ (first 64 coordinates are responsible for the luma component, next 64 describe the blue-difference, and last 64 coordinates are responsible for the red-difference chroma components).

We compared our approach with three classical methods. The first one is a classical Karhunen–Loève transform (PCA) [10]. In the second method is the k -means clustering which uses PCA in each cluster was applied (more precisely, we first divided the data into k -cluster, in which we later reduced the dimension by PCA). The third one is a first part of the JPEG compression method. JPEG consists of four steps: preprocessing, transformation, quantization and encoding. Since we compare the results obtained by linear transformation, we omitted the last two steps – quantization and encoding. Transformation is based on the change of the base to that given in the discrete cosine transform (according to the zig-zag order).

The quality of compression is measured by Mean Squared Error (MSE). Table 7 shows our results.

Observe that, as expected, if we increase data compression ratio with fixed number of clusters, then MSE increases. On the other hand, if we increase the number of clusters and fix the data compression ratio, then the value of MSE decreases. As one can see in the above Table, we obtain better results from almost all (98%) considered methods and cases. Typically, we lose in the case of small r and large number of initial clusters with k -means+PCA due to the fact that our method substantially reduces the initial number of clusters.

8. Conclusions

In this paper we presented a subspace projection clustering algorithm **SuMC**, which is based on information theory and lossy version of the Minimum Description Length Principle (MDLP). As a consequence, our algorithm has a penalty of using each cluster built into the cost function, which results in its ability to reduce unnecessary clusters. Moreover, thanks to strong theoretical background, **SuMC** has a well-defined cost function, and consequently the results of various clusterings can be easily compared.

In this paper we generalize and modify some ideas from our earlier paper [31], in which we did not apply the MDLP approach described above.⁹ The idea is based on minimizing the squared error, while keeping the total amount of memory fixed. Due to this modification, we needed to rebuild the typical approach used in standard clustering algorithms, since in practice, we minimize one function while keeping the other under strict constraint.

In its simplest form, to start our algorithm, we have to specify two parameters – the amount of bits reserved for the real scalar and the total amount of memory that is at our disposal. It produces subspaces that are typically of various dimensions and are not parallel to the axis of the coordinate system. Since our method uses MDLP-based idea, it can be used not only for clustering but also for image segmentation or data compression.

We compared our algorithm with ORCLUS and 4C for both – synthetic and real data. Experiments show that it outperforms both ORCLUS and 4C. Moreover, **SuMC** has lower time complexity than 4C, but similar to ORCLUS.

Our technique is applicable to clustering datasets of not very large dimensionality due to the computational complexity of the method (the highest dimension we were able to process efficiently was below 800 in the case of MNIST dataset).

In future we plan to investigate the possibility of using modified or simplified approaches to deal with data of dimensions larger than 10^6 (which is a common case in NLP).

We plan to use an approach similar to PROCLUS and consider only subspaces parallel to the main axes of the coordinate system. Such a modification allows us to cluster higher dimensional data, but with larger error.

References

- [1] J. Aczél, Z. Daróczy, On measures of information and their characterizations, Academic, New York, 1975.
- [2] C.C. Aggarwal, C.K. Reddy, Data Clustering: Algorithms and Applications, CRC Press, 2013.
- [3] C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, J.S. Park, Fast algorithms for projected clustering, in: ACM SIGMOD Record, 28, ACM, 1999, pp. 61–72.
- [4] C.C. Aggarwal, P.S. Yu, Finding Generalized Projected Clusters in High Dimensional Spaces, 29, ACM, 2000.
- [5] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, 27, ACM, 1998.
- [6] P. Arabie, L. Hubert, An overview of combinatorial data analysis, Cluster. Classif. (1996) 5–63.
- [7] C. Böhm, K. Kailing, P. Kröger, A. Zimek, Computing clusters of correlation connected objects, in: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM, 2004, pp. 455–466.
- [8] D.L. Donoho, et al., High-dimensional data analysis: the curses and blessings of dimensionality, AMS Math. Challenges Lecture (2000) 1–32.
- [9] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise., in: Kdd, 96, 1996, pp. 226–231.

⁹ It occurs, see the end of Section 4, that the subspace clustering method constructed in [31] can be obtained as a limiting case of our model, where the amount of bits b used for saving in memory of a given scalar converges to ∞

- [10] K. Fukunaga, "Introduction to Statistical Pattern Recognition", Academic press, 1990.
- [11] G. Gan, M.K.-P. Ng, Subspace clustering using affinity propagation, *Pattern Recognit.* 48 (4) (2015) 1455–1464.
- [12] P.D. Grünwald, *The Minimum Description Length Principle*, MIT press, 2007.
- [13] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, *J. Mach. Learn. Res.* 3 (2003) 1157–1182.
- [14] J.A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, Inc., 1975.
- [15] I. Jolliffe, *Principal Component Analysis*, Wiley Online Library, 2005.
- [16] D.E. Knuth, H. Saitou, T. Nagao, S. Matui, T. Matui, H. Yamauchi, *Of Book: The Art of Computer Programming.-Volume 2, Seminumerical Algorithms* (Japanese Edition), 2, ASCII, 2004.
- [17] H.-P. Kriegel, P. Kröger, A. Zimek, Subspace clustering, *Wiley Interdiscip. Rev.* 2 (4) (2012) 351–364.
- [18] M. Li, P. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer Science & Business Media, 2013.
- [19] D. Martin, C. Fowlkes, D. Tal, J. Malik, A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, in: *Proc. 8th Int'l Conf. Computer Vision*, 2, 2001, pp. 416–423.
- [20] B. McWilliams, G. Montana, Subspace clustering of high-dimensional data: a predictive approach, *Data Min. Knowl. Discov.* 28 (3) (2014) 736–772.
- [21] A.A. Miranda, L.B. Yann-Aël, G. Bontempi, New routes from minimal approximation error to principal components, *Neural Process. Lett.* 27 (3) (2008) 197–207.
- [22] G. Moise, J. Sander, Finding non-redundant, statistically significant regions in high dimensional data: A novel approach to projected and subspace clustering, in: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2008, pp. 533–541.
- [23] E. Müller, S. Günemann, I. Assent, T. Seidl, Evaluating clustering in subspace projections of high dimensional data, *Proc. VLDB Endowment* 2 (1) (2009) 1270–1281.
- [24] F. Nie, X. Wang, H. Huang, Clustering and projected clustering with adaptive neighbors, in: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014, pp. 977–986.
- [25] D. Pelleg, A.W. Moore, et al., X-means: Extending k-means with efficient estimation of the number of clusters., *ICML*, 1, 2000.
- [26] K. Sayood, *Introduction to Data Compression*, Newnes, 2012.
- [27] G. Schwarz, et al., Estimating the dimension of a model, *Annals Stat.* 6 (2) (1978) 461–464.
- [28] C.E. Shannon, A mathematical theory of communication, *ACM SIGMOBILE Mobile Computing and Communications Review* 5 (1) (2001) 3–55.
- [29] P. Spurek, M. Śmieja, K. Misztal, Subspaces Clustering Approach to Lossy Image Compression, in: *Computer Information Systems and Industrial Management*, Springer, 2014, pp. 571–579.
- [30] P. Spurek, J. Tabor, K. Misztal, Weighted Approach to Projective Clustering, in: *Computer Information Systems and Industrial Management*, Springer, 2013, pp. 367–378.
- [31] Ł. Struski, J. Tabor, P. Spurek, Subspace memory clustering, *Schedae Informaticae* 24 (2015) 133–142.
- [32] J. Tabor, P. Spurek, Cross-entropy clustering, *Pattern Recognit.* 47 (9) (2014) 3046–3059.
- [33] R. Vidal, A tutorial on subspace clustering, *IEEE Signal Process. Mag.* 28 (2) (2010) 52–68.
- [34] A.J. Walker, An efficient method for generating discrete random variables with general distributions, *ACM Trans. Math. Softw. (TOMS)* 3 (3) (1977) 253–256.
- [35] J. Wang, Z. Deng, K.-S. Choi, Y. Jiang, X. Luo, F.-L. Chung, S. Wang, Distance metric learning for soft subspace clustering in composite kernel space, *Pattern Recognit.* 52 (2016) 113–134.
- [36] Q. Zhang, L.T. Yang, Z. Chen, P. Li, High-order possibilistic c-means algorithms based on tensor decompositions for big data in iot, *Inf. Fusion* 39 (2018) 72–80.
- [37] Q. Zhang, C. Zhu, L.T. Yang, Z. Chen, L. Zhao, P. Li, An incremental cfs algorithm for clustering large data in industrial internet of things, *IEEE Trans. Ind. Inf.* 13 (3) (2017) 1193–1201.
- [38] L. Zhu, L. Cao, J. Yang, Multiobjective evolutionary algorithm-based soft subspace clustering, in: *Evolutionary Computation (CEC), 2012 IEEE Congress on*, IEEE, 2012, pp. 1–8.