Learning and Nonlinear Models - Journal of the Brazilian Society on Computational Intelligence (SBIC), Vol. 19, Iss. 1, pp. 4-16, 2021

IMPROVING PROTOTYPES REPRESENTATIVENESS BY INTERNAL VALIDITY INDEX ANALYSIS

Alexandre Szabo
Thomaz A. Ruckl

Federal University of Grande Dourados
Faculty of Exact Sciences and Technology
e-mails: alexandreszabo@ufgd.edu.br, thomaz-araujo@outlook.com
Rodovia Dourados/Itahum, Km 12 - Unidade II
CP 364, CEP 79804-970, Dourados, MS, Brazil

Abstract- Internal validity indexes are applied to evaluate the solution of a partition, which no equally reflects the same quality for all clusters, individually, in terms of prototypes representativeness. Thus, knowing their representativeness in respective clusters, it is possible adjust them to increase the confidence in analysis of found clusters. In this sense, this paper proposes a simple and effective method to obtain the internal validity index value in every cluster in a partition, identify those with low prototypes representativeness and improve them. Experiments were carried out by sum of the squared error index, which measures the compactness of clusters. The behavior of the method was illustrated by a synthetic dataset and performed for ten datasets from the literature with *k*-Means algorithm. The results demonstrated its effectiveness for all experiments.

Keywords- Data clustering, internal validity index, sum of squared error, *k*-Means.

1 Introduction

Data clustering is an important task in Data Mining area, responsible for segmenting objects in clusters, according to similarity among them, i.e., objects in a same cluster are similar to each other. The importance of clustering is observed in different areas, such as computer vision (Bhowmik et al., 2019), business (Caruso et al., 2020), medicine (Kannan et al., 2013; Khanmohammadi et al., 2017), and others.

Objects in a dataset are represented by vectors, which dimension is given by their attributes. Therefore, the dissimilarity among them can be defined by some distance measure (Visalakshi and Suguna, 2009; Prasetyo and Purwarianti, 2014; Kapil and Chawla, 2016), such as Euclidean distance. This task is also called unsupervised classification, once objects' class is not known, *a priori*. The objective is finding such classes and describe their characteristics (Han and Kamber, 2006).

Classes can be identified by natural clusters, i.e., regions relatively densely populated, surrounded by empty regions (Carmichael and Julius, 1968). However, in real problems, the frontier among clusters is not well defined; clusters are generally overlapped and present different forms and sizes, which becomes clustering a challenging task (Nagy, 1968; de Oliveira and Pedrycz, 2007; Ruspini et al., 2019).

Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of objects to be grouped and $C = \{c_1, c_2, \dots, c_m\}$ a set of clusters, a partitional clustering algorithm aims partitioning n objects in m clusters, such that $m < n$. The goal is producing compact and/or separable clusters, i.e., minimize intra-cluster distances and maximize inter-cluster, respectively, in clustering algorithms based on prototypes.

Partitional algorithms use prototypes, vectors representatives of clusters, which can be randomly generated on dataset to be clustered, and iteratively updated for the proposal of becoming centroids in respective clusters. Using prototypes facilitates analyzing characteristics of clusters, replacing each group of objects by a representative. In this context, partitional algorithms aim to move prototypes in center of respective clusters or in region of higher density of objects, in order to generalizing their characteristics.

There are several ways to group n objects in m clusters and all solutions can be corrects, which denotes a combinatorial optimization problem (Jain et al., 1999; Mehdizadeh and Tavakkoli-Moghaddam, 2007). Differently of supervised classification problems (Umadevi and Marseline, 2017; Alzahrani and Rawat, 2019), there is no a correct model to be compared to solutions from clustering problems. Thus, internal validity indexes are commonly applied to validate found clusters (Vij and Khandnor, 2016; Rojas-Thomas et al., 2019). Such indexes consider inter-cluster separability and/or intra-cluster compactness, which value depends of prototypes position in relation to objects and/or clusters. The better the index value, the better the prototypes representativeness to clusters.

Validity indexes produce a unique value, which represents the general quality for all clusters in a partition. However, one or more clusters with low representative compromises the whole partition. The ideal would be evaluating each cluster, individually, which could allow to adjust their position, if necessary, for a better confidence on results. Improving individually prototypes representativeness, consequently the quality of entire partition will be improved too.

Thus, this paper proposes a method to obtain the internal validity index value in every cluster in a partition, identify those with low prototypes representativeness and improve them.

The paper is organized as follows: Section 2 presents a brief background to *k*-Means algorithm and internal validity indexes. Section 3 describes the proposed method by detailing the algorithm steps. The method is evaluated in Section 4 and the results are discussed. The paper is concluded in Section 5 with future works.

2 Background

This section provides basics concepts about *k*-Means algorithm and internal validity indexes, focusing the squared error index (SSE).

2.1 k-Means Algorithm

k-Means is one of the most traditional clustering algorithms from the partitional clustering literature (MacQueen, 1967; Jain, 2010). Applications are vast and include gene expression data (Liu et al., 2009), protein sequence (Elayaraja et al., 2012), image segmentation (Chen et al., 2015; Li et al., 2015), medicine (Su et al., 2017), air pollution (Govender and Sivakumar, 2020) and many others. Its popularity is given by ease of implementation and understanding, rapid convergence and produces satisfactory results. On the other hand, it is sensitive to initialization of prototypes and converges to a local minimum.

Clustering algorithms based on prototypes, such as *k*-Means, use some dissimilarity measure among objects to assign them to respective clusters. Euclidean distance generally is applied to this goal. The smaller the distance between object and prototype, the more similar they are each other:

$$d_{ij} = \sqrt{\sum_{a=1}^{dim} (\boldsymbol{x}_i^a - \boldsymbol{c}_j^a)^2} \tag{1}$$

where d_{ij} is the Euclidean distance between object $\mathbf{x}_i$ and prototype $\mathbf{c}_j$, which dataset dimension is *dim*.

As all partitioning clustering algorithms, *k*-Means produces a membership matrix $\mathbf{U}_{nxm} = \{\mu_{11}, \mu_{12}, \dots, \mu_{nm}\}$, such that $\mu_{ij} \in \{0,1\}$ represents the membership degree between object of index *i* and cluster of index *j*, i.e., the relation between objects and prototypes. The matrix is updated by the nearest neighbor's technique – objects are assigned to clusters which distance is the smallest:

$$\mu_{ij} = \begin{cases} 1, if\ d_{ij} < d_{ik}, \forall\ k \in C, k \neq j \\ 0, otherwise \end{cases} \tag{2}$$

The goal of *k*-Means is minimizing the function:

$$SSE = \sum_{j=1}^{m} \sum_{i=1}^{n} \mu_{ij}\, d_{ij}^2 \tag{3}$$

where SSE is the sum of the squared error (Javed et al., 2008; Rajee and Francis, 2013).

k-Means is characterized by updating prototypes, iteratively, by mean of objects for which they represent:

$$\mathbf{c}_j = \frac{\sum_{i=1}^{n} \mu_{ij} \mathbf{x}_i}{\sum_{i=1}^{n} \mu_{ij}} \tag{4}$$

The algorithm is presented as follows:

Algorithm 1: *k*-Means
Input: dataset, *k*
Output: membership matrix **U**, prototypes **C**, sum of squared error SSE
1.Initialize *k* clusters, randomly
2.update **U** (Eq.2)
3.while SSE is not minimized (Eq.3)
4. update **C** (Eq.4)
5. update **U** (Eq.2)
6. update SSE (Eq.3)
7.end while
8.return (**U**,**C**,SSE)

2.2 Internal Validity Indexes

Validity indexes measure the quality of a partition, i.e., the validity of found clusters (Cui et al., 2014; de Oliveira et al., 2017). In real problems, we can evaluate the quality of a partition by analyzing the position of prototypes in respective clusters, i.e., their representativeness to clusters, as well as their position to each other. The more representative the prototypes and/or separable, better the partition. Such indexes are called internal validity indexes.

Differently of external indexes, internal indexes do not use label of objects, but intra-cluster compactness and/or inter-cluster separability.

Compactness refers to cohesion intra-cluster. It measures how close objects are within a cluster; the lesser the scattered the objects into a cluster, more compact it is. Indexes as squared error and davis-bouldin (Davies and Bouldin, 1979; Xiao et al., 2017), e.g., consider intra-cluster distance. On the other hand, separability is measured by inter-cluster distances. It indicates how dissimilar or well-separated the clusters are from each other (Liu et al., 2010; Cui et al., 2014); the higher the distance between clusters, the more distinct they are. Dunn index (Dunn, 1974; Hamasuna et al., 2017) is one of that use the inter-cluster separability scheme.

Some indexes measure both compactness and separability, simultaneously, such as xie-beni (Xie and Beni, 1991; Pakhira et al., 2004), calinski-harabasz (Calinski and Harabasz, 1974; Maulik and Bandyopadhyay, 2002) and silhouette (Rousseeuw, 1987; Vendramin et al., 2010). For more details and other indexes, see (Desgraupes, 2017).

Validity indexes also had been used as objective-function to guide the clustering and as tool to determine the optimum number of clusters (Pal and Bezdek, 1995; Zhao and Fränti, 2014; He et al., 2016; Kesemen et al., 2017). In this paper, internal validity index was used to measure the quality of individual clusters in a partition.

3 Proposed Method

This section introduces the proposed method, which can identify clusters with low prototypes representativeness and adjust their position to improve the general quality of a partition. It is focused to increase the compactness of clusters in a partition. So, some intra-cluster index must be used. For that, the squared error index was used for this purpose.

The method has three steps: (i) obtaining internal validity index value of individual clusters, i.e., prototypes representativeness; (ii) measuring how representative to clusters the prototypes are, by comparing its index value with a threshold ε; this step determines whether prototypes must be adjusted; and (iii) adjust them, if necessary.

Algorithm 2: Proposed Method
Input: dataset, prototypes, threshold ε
Output: adjusted prototypes
1.obtain the index value of clusters (Eq.5)
2.normalize all values obtained in Line 1 (Eq.7)
3.generate new prototypes on clusters (Eq.9), for which *normalizedSolution* < ε(Eq.8)
4.run a partitional clustering algorithm with new prototypes added (such as Algorithm 1)
5.return adjusted prototypes

The algorithm's steps are detailed as follows.

3.1 Obtaining Internal Validity Index Value in Individual Clusters

The validity index is obtained for each cluster. By considering the squared error index, the sum of intra-cluster distances between objects and respective prototype is:

$$\boldsymbol{sse}(j) = \sum_{i=1}^{n} \mu_{ij}\, d_{ij}^2, \forall\, j = 1 \ldots m \tag{5}$$

the closer to zero, the more representative to cluster. This step allows evaluate the influence of partial solutions in composition of the global solution and it will be applied to next step.

3.2 Measuring Representativeness in Individual Clusters

This step can identify clusters with low prototypes representativeness in a partition. For that, the quality of individual clusters, obtained in previous step, must be relative to the best cluster. That is important to measure how good is a partial solution. Thus, every individual solution is normalized (0,1]; closer to 1, better the partition:

$$bestSolution = arg_{max}(\mathbf{individualSolutions}) \tag{6}$$

$$\mathbf{normalizedSolutions} = \mathbf{individualSolutions}/bestSolution \tag{7}$$

For indexes where the smaller value the better cluster, such as squared error, the normalized value requires a previous step, before to get the $bestSolution$, given by $(\mathbf{individualSolutions})^{-1}$. Thus, the smaller the value, the higher it will be and *vice-versa*. However, how far from the best solution the individual ones must be to decide adjust their prototypes? A threshold ε was used for this purpose. It determinates how representative a prototype is to a cluster:

$$if\,(\mathbf{normalizedSolution}(j) < \varepsilon), \text{adjust prototype } \mathbf{c}_j \tag{8}$$

The threshold is compared with the normalized index of each cluster, and its value is a normalized quality indicator $\varepsilon \in (0,1]$. Solution under the threshold means low prototype representativeness in a cluster. The threshold close to zero (one) minimizes (maximizes) the application of method.

3.3 Improving Representativeness in Individual Clusters

The proposed method improves the representativeness in a cluster by adding a new prototype around of the respective prototype defined by threshold ε:

$$\boldsymbol{newPrototype} = \Delta\boldsymbol{prototype} \tag{9}$$

where $\boldsymbol{\Delta prototype} = (0.1 * rand(1, dimensionObject)) * \boldsymbol{prototype} + \boldsymbol{prototype}$, i.e., its position is defined by a little perturbation on prototype to be adjusted. Thus, the new prototype will contribute to position that one considered bad. The constant 0.1 in Eq.9 was empirically defined, so that the new prototype be added closely to the bad one.

The new prototypes are added to old set of prototypes and a partitional data clustering algorithm is applied for adjusting their position in respective clusters, improving their representativeness.

4 Performance Assessment

This section presents the methodology, results and discussion. The proposed method was carried out in three analyses: (i) its behavior was evaluated in a synthetic dataset; (ii) a parametric sensitivity analysis was performed for threshold ε; and (iii) results obtained for the best threshold were compared with that before applying the method.

4.1 Methodology

The proposal is improving the clusters compactness by adjusting the prototypes position when necessary. Thus, the sum of the squared error index was used as prototype representativeness measure (Eq.3).

The behavior of the Method was illustrated by Ruspini synthetic dataset (Section 4.2.1). Right after, the method was evaluated for ten datasets available at UCI repository[1], in ten runs for each one of the ten values of $\varepsilon = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, in a parametric sensitivity analysis. That analysis was necessary to determine the most appropriated threshold value.

The *k*-Means algorithm was applied in both experiments for generating partition and adjusting of prototypes position by proposed method; *k*-Means stopping criterion was given by ten iterations. The prototypes were randomly initialized on datasets for each run of *k*-Means, and the same initialization was kept for all thresholds on respective run. The number of initialized prototypes was equal to number of classes in respective datasets. Codes and figures were produced in Octave[2].

Results for parametric sensitivity analysis were presented in three graphics (Section 4.2.2), meanwhile the results obtained by the best threshold were presented in table in terms of mean (Section 4.2.3). For the experiments, the results were computed when the application of the proposed method was necessary to one or more clusters, i.e., the solution quality was under the threshold value.

The main characteristics from dataset are presented in Table 1.

Table 1: Main characteristics from datasets

Dataset	#Classes	#Attributes	#Objects
Ruspini	4	2	75
Ionosphere	2	34	351
Statlog Heart	2	13	270
Haberman	2	3	306
Liver	2	7	345
Iris	3	4	150
Wine	3	13	178
Balance Scale	3	4	625
Soybean	4	35	307
Yeast	4	8	1299
Glass	6	9	214

For Yeast dataset, only four of more abundant classes were considered.

4.2 Results and Discussion

This section was divided in three kinds of analyses:

i. The behavior of the proposed method, illustrated for a synthetic dataset in a two-dimensional scenario (Section 4.2.1).

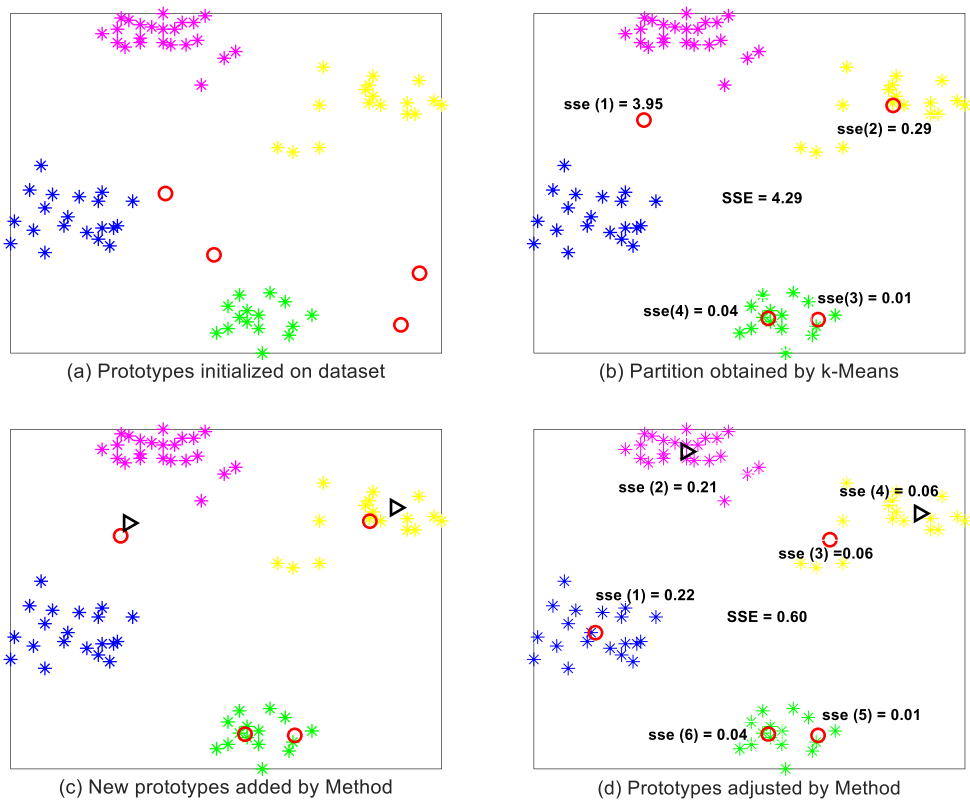
ii. A parametric sensitivity analysis for the threshold ε in terms of gain(%), number of generated prototypes(%), and how many times(%) the method was applied in ten runs, concluding the analysis with statistical tests to determine the best threshold ε (Section 4.2.2).

iii. Comparison between results obtained for the best threshold ε and ones before applying the proposed method, both experiments for number of prototypes equal to classes from datasets, as well as number of prototypes over than number of classes in respective datasets (Section 4.2.3).

4.2.1 Behavior of the Proposed Method for Ruspini Synthetic Dataset

The scenario to illustrate the behavior of proposed method is depicted in Fig.1.

[1] https://archive.ics.uci.edu/ml/datasets.html
[2] https://www.gnu.org/software/octave/



(a) Prototypes initialized on dataset
(b) Partition obtained by k-Means
(c) New prototypes added by Method
(d) Prototypes adjusted by Method

Figure 1: Ruspini dataset. The behavior of the Method in a two-dimensional scenario for $\varepsilon = 0.1$.

Fig.1(a) shows the initial scenario: dataset to be clustered (asterisks) and prototypes randomly initialized (red circles). As we can see, a bad initialization of prototypes for *k*-Means produces low prototypes representativeness (Fig.1(b)).

In Fig.1(b) Prototype 1 represents Classes 1 and 2 ($sse(1) = 3.95$), Prototype 2 represents Class 3 ($sse(2) = 0.29$), and Prototypes 3 and 4 represent Class 4 ($sse(3) = 0.01$ and $sse(4) = 0.04$). Their normalized indexes values are around 0.0 ($sse(1)$), 0.03 ($sse(2)$), 1.0 ($sse(3)$) and 0.25 ($sse(4)$), where Prototype 1 has the worst representativeness and Prototype 3 has the best one.

For this scenario (Fig.1(b)), with $\varepsilon = 0.1$, the method identified low prototypes representativeness ($sse < \varepsilon$): (i) Prototype 1 far from a centroid of a cluster; the prototype maps two natural clusters; and (ii) density of objects in Cluster 2, i.e., part of this cluster has more objects, dividing it in two regions. Thus, two new prototypes were added (black triangles) around those bad ones, as showed in Fig.1(c). For *m* prototypes in a partition, the maximum number of new prototypes added by method is *m*-1.

The solution obtained by proposed method is presented in Fig.1(d), which partition solution was 0.60 after method, against 4.29 before its application (Fig.1(b)). As we can see in Fig.1(d), Prototypes 5 and 6 held on respective positions after applying method because they fell in a local minimum. Thus, only badly positioned prototypes, influenced by new added prototypes, move on space.

The ε closer to 1.0, the more compact clusters are desired, i.e., the probability to apply the Method increases (see Eq.8). However, a sensibility analysis for ε was necessary to evaluate the effectiveness of method.

4.2.2 Parametric Sensibility Analysis

Fig.2 shows how many times the method was applied into ten runs. The gain(%), presented in Fig.3, represents how much the quality increased after applying the proposed method, on average, as well as total of prototypes added in each dataset, in Fig.4.

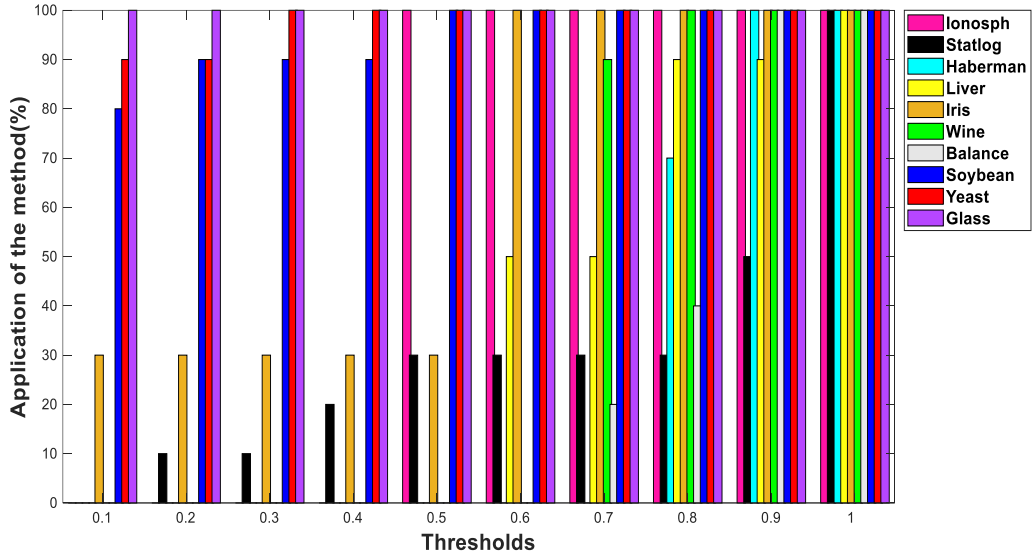


Figure 2: Application of the Method (%) for all thresholds.

For $\boldsymbol{\varepsilon = 0.1}$, the prototype representativeness was adjusted in 40% of the datasets, between 25% and 100% of runs. It means that the normalized quality of one or more clusters in those datasets was under 0.1 before the proposed method be applied. The Glass dataset presented low prototype representativeness in all runs, for all thresholds.
The need of prototypes representativeness adjustment for Ionosphere was verified to $\boldsymbol{\varepsilon \geq 0.5}$ and the method was applied 100% of runs for such thresholds.
For Soybean, the method was applied for all thresholds. When $\boldsymbol{\varepsilon \geq 0.5}$, the method was applied in 100% of runs.
For $\boldsymbol{\varepsilon >= 0.6}$, the method was applied to 80% of datasets. It means that partial solutions for these datasets are homogeneous, relatively, which does not require adjustment for very low thresholds.
For $\boldsymbol{\varepsilon \geq 0.7}$, the necessity of adjustment was to 90% of datasets, except to Haberman. Thus, for $\boldsymbol{\varepsilon \geq 0.8}$ the adjustment was observed to all datasets, and the method was applied 100% of runs to 60% of datasets.
For $\boldsymbol{\varepsilon = 0.9}$, the method was applied over than 40% and 80% of runs to Statlog and Liver, respectively, and 100% of runs to the other datasets. For $\boldsymbol{\varepsilon = 1.0}$, the method was applied 100% of runs for all datasets. As we can see in Fig.2, the higher the threshold, the more the probability to apply the proposed method.

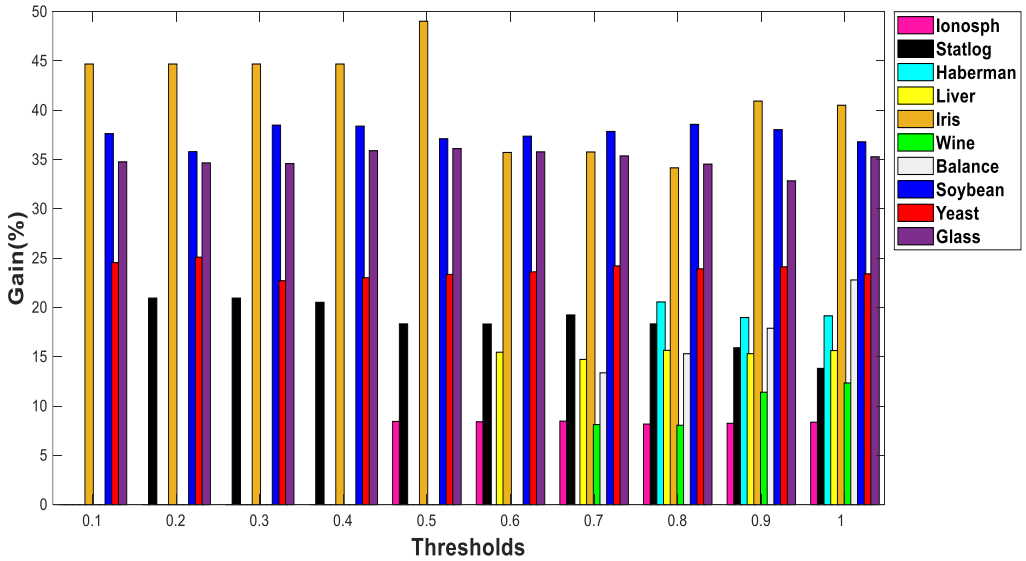


Figure 3: Growth of representativeness in partitions after application of Method.

The gain(%) depicted in Fig.3 represents how much the quality (prototypes representativeness) increased after application of proposed method:

$$gain(\%) = \left(\frac{|quality - adjustedQuality|}{quality}\right) * 100\% \quad (10)$$

where *quality* is the index value before method being applied, meanwhile *adjustedQuality* is the index value obtained by method.

For Iris, Soybean, Yeast and Glass datasets, the gain(%) is verified to all thresholds. Iris presented the higher gain(%) among datasets; 43% on average for $\varepsilon = \{0.1, 0.2, 0.3, 0.4\}$, and 49% on average to $\varepsilon = 0.5$.

For Ionosphere, the method was necessary for $\varepsilon \geq 0.5$, with a gain(%) around 8.4%, on average, to those thresholds.

For $\varepsilon \geq 0.8$, the prototype adjustment was necessary for all datasets.

It was observed that most of datasets produced homogeneous quality among clusters in a partition when ε decreases from 0.4, i.e., the method was necessary only for part of datasets (between 40% and 50%).

The partition quality was improved for all experiments when the proposed method was applied, with gain(%) over than 8%, on average, for all datasets. The gain(%) presents low variation, on average, when the threshold increases, and remains for $\varepsilon = \{0.9,1.0\}$, which suggest that a high threshold produces a partition with homogeneous quality among clusters.

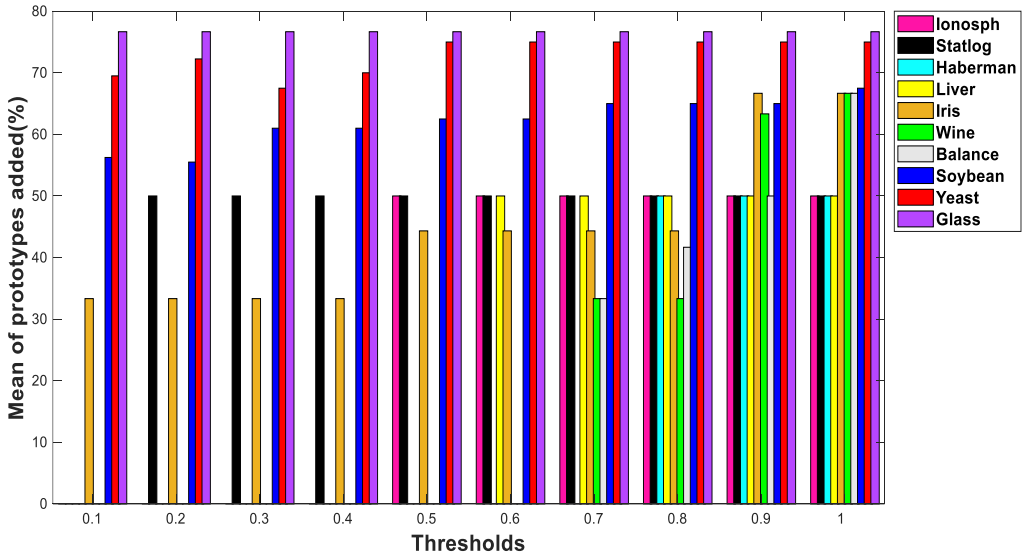


Figure 4: Percentage of prototypes added in partition after application of Method.

The mean of prototypes added by method remained to respective datasets, on average, for all thresholds, for which the method was applied. The exception was verified to Wine and Balance, for which the amount of prototypes presented some variation among thresholds, on average. For both datasets, the method was applied only for $\varepsilon \geq 0.7$.

According to Figs.2, 3 and 4, the application of method increases according to the threshold value. The higher the threshold, the more times the method trends to be applied among runs. The prototypes representativeness was improved for all datasets for which the method was applied, and the quality of partition remains among all thresholds on respective dataset, on average. Furthermore, the amount of prototypes added does not substantially vary, on average, among thresholds. Therefore, the best values for ε were for 0.8, 0.9 and 1.0, for which all datasets had one or more bad position prototypes (Fig.3).

A statistical analysis was carried out for $\varepsilon = \{0.8,0.9,1.0\}$ to identify the most appropriated. The tests were performed for gain(%) and amount of prototypes added (addPrototypes(%)), separately.

First, a normality test was applied to determine whether a parametric or non-parametric test should be used (Derrac et al., 2011). For that, Shapiro-Wilk test (Shapiro and Wilk, 1965) was performed and result showed that null hypothesis must be rejected for both gain(%) and prototypes added(%), with 95% of confidence. Thus, Wilcoxon (1945) was applied as a non-parametric test, and it concluded for both experiments that there is no significant difference in performance among three thresholds, with 95% of confidence. However, for $\varepsilon = \{0.9,1.0\}$, the probability of generating prototypes increases, on average, in relation to lower

ones, even it is not significant. Thus, $\varepsilon = 0.8$ is more interesting among three thresholds, which minimizes to add new prototypes without compromising the solution quality.

4.2.3 Performance of the Proposed Method for the Best Threshold

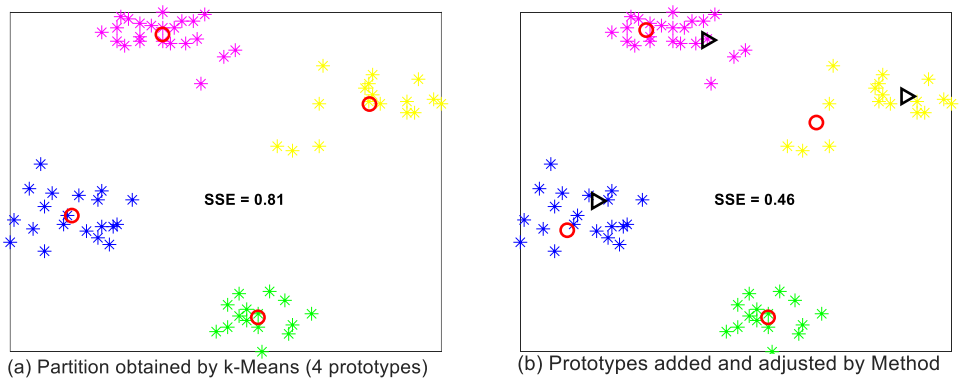
This section compares results obtained by $\varepsilon = 0.8$ with ones before applying the proposed method. We considered two kinds of analysis for this threshold: (i) number of initialized prototypes equal to respective classes in datasets, and (ii) number of initialized prototypes given by amount prototypes added by Method in experiment (i). For both experiments we performed the mean of prototypes added by Method (addPrototypes(%)), gain(%) in relation to squared error obtained before applying Method (SSE), and application of the Method in ten runs (times(%)). Results before applying Method were only computed for which the Method was applied.

Table 2: Mean of prototypes added by the proposed method (addPrototypes(%)), gain(%), times(%) and sum of the squared error (SSE)

		Proposed Method, $\varepsilon = 0.8$				*k*-Means
Datasets	#classes	addPrototypes(%)	gain(%)	times(%)	SSE	SSE
Ionosphere	2	50.00	08.17	100.00	577.51	628.90
Statlog Heart	2	50.00	18.33	30.00	282.58	346.02
Haberman	2	50.00	20.57	70.00	20.08	25.28
Liver	2	50.00	15.67	90.00	25.30	30.00
Iris	3	44.33	34.19	100.00	05.39	08.19
Wine	3	33.33	08.04	100.00	45.04	48.98
Balance Scale	3	41.67	15.31	40.00	185.85	219.45
Soybean	4	65.00	38.57	100.00	59.35	96.61
Yeast	4	75.00	23.92	100.00	62.64	82.33
Glass	6	76.67	34.56	100.00	15.49	23.67

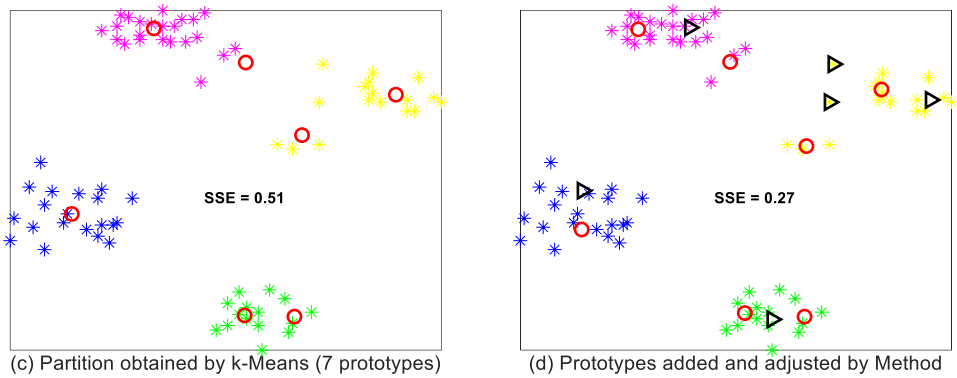
The highest gain(%) in terms of compactness was obtained for Soybean (38.57%), Glass (34.56%) and Iris (34.19%) datasets, meanwhile Ionosphere (08.17%) and Wine (08.04%) produced the smallest ones. For these datasets, the Method was applied 100% of runs. For Statlog Heart, the method was applied only 30% of runs; that means the solution of individual clusters was homogeneous in 70% of partitions generated by *k*-Means. For Yeast and Glass, the method added 75% and 76.67% of prototypes, respectively, and they represent the highest percentages among datasets. However, the amount of prototypes added is not related to gain(%). It can be noticed by comparing some datasets, such as Ionosphere (08.17%), Balance (15.31%), Iris (34.19%) and Yeast (23.92%), for which the method added 50%, 44.33%, 41.67% and 75% of prototypes, respectively.

Experiments were also performed by considering increase the number of prototypes initialized on dataset, in order to verify whether the proposed method is necessary when number of prototypes is over than number of classes in respective dataset. For that, we considered this additional amount from mean of generated prototypes by proposed method (addPrototypes(%), Table 2). A typical behavior of the Method is depicted in Figure 5, for Ruspini dataset, meanwhile Table 3 shows the results for the other datasets.



(a) Partition obtained by k-Means (4 prototypes)

(b) Prototypes added and adjusted by Method



(c) Partition obtained by k-Means (7 prototypes) (d) Prototypes added and adjusted by Method

Figure 5: Ruspini dataset. The behavior of the Method in a two-dimensional scenario for $\varepsilon = 0.8$. Partition obtained by *k*-Means with 4 (a) and 7 (c) prototypes initialized, and respective partition produced by Method (b) and (d).

As we can see in Fig.5, increasing the number of prototypes on dataset does not suppress the Method application, either by a bad initialization of prototypes, or for a disparity among individual solutions. There are two factors that contribute to add prototypes by Method:

(a) Heterogeneity among individual solutions
The quality of individual solutions is relativized by the best solution (Eq.7), which highlights the bad solutions; the more heterogeneous the solutions, the greater the probability of adding prototypes;
(b) Affinity threshold (ε)
The higher the threshold, the greater the probability of adding prototypes. It is closely linked to the quality of individual solutions.

Thus, the proposed method aims to homogenize individual solutions from the best solution, by adding prototypes close to bad clusters, adjusting them, which increases the confidence on results.

Table 3: Mean of prototypes added by the proposed method (addPrototypes(%)), gain(%), times(%) and sum of the squared error (SSE), with #prototypes over than #classes

		Proposed Method, $\varepsilon = 0.8$				*k*-Means
Datasets	#initialPrototypes	addPrototypes(%)	gain(%)	times(%)	SSE	SSE
Ionosphere	3	63.33	12.02	100.00	512.90	582.98
Statlog Heart	3	62.96	19.51	90.00	226.67	281.63
Haberman	3	60.00	31.64	100.00	14.67	21.46
Liver	3	59.26	20.03	90.00	20.71	25.89
Iris	5	78.00	40.56	100.00	03.07	05.16
Wine	4	70.00	14.25	100.00	39.47	46.03
Balance Scale	5	48.57	20.80	70.00	133.17	168.14
Soybean	7	71.43	35.79	100.00	42.98	66.93
Yeast	7	81.43	27.64	100.00	49.26	68.08
Glass	11	51.82	35.45	100.00	13.42	20.79

By increasing the number of initialized prototypes (#InitialPrototypes), relatively to number of classes in respective dataset, *k*-Means produced better SSE, as well as Method applied to respective partition. However, the number of prototypes implies in additional computational cost, which is not desirable. Besides that, *k*-Means is sensible to initialization of prototypes and can produce bad solutions. The proposed method minimizes the initialization problem (see Fig.1).

For Glass dataset, the addPrototypes(%) considerably decreased (from 76.67% (Table 2) to 51.82% (Table 3)). Differently, for Iris and Wine, the percentage of prototypes added by Method increased 75.95% and 110.02%, respectively. Therefore, the Method application was not suppressed by increasing the number of prototypes; on the contrary, its application increased (times(%)) among runs for all datasets (except to Liver): Statlog (200.00%), Haberman (42.86%) and Balance (75.00%).

The gain(%) increased in relation to that presented in Table 2, since clusters are represented by more prototypes.

5 Conclusion and Future Works

Internal validity indexes are commonly applied to evaluate a quality of a partition in clustering algorithms based on prototypes, which consider their position within clusters. According to index value in each cluster, we can adjust the prototype improving its representativeness in terms of compactness, which increases the confidence in analysis of found clusters. The proposed method goes in this sense, and uses a threshold to decide if the adjustment of prototypes is necessary. In affirmative case, a prototype is added around of it, promoting its adjustment within respective cluster.

The experiments with the proposed method were carried out to three kinds of analyses and it was verified that increasing threshold ε produces an important gain(%) in terms of compactness, for all datasets. Thresholds in $\varepsilon = \{0.8, 0.9, 1.0\}$ presented the best solutions among thresholds tested and a statistical analysis showed there is no significant difference in gain(%) and neither in addPrototypes(%) for $\varepsilon = \{0.8, 0.9, 1.0\}$. Thus, as $\varepsilon = \{0.9, 1.0\}$ the method has high probability to add prototypes for any dataset, $\varepsilon = 0.8$ was considered the most appropriate threshold value.

For future works, the partial solutions normalization function will be improved in order to minimize the emphasis from the best solution to the bad ones, and so control adding unnecessary prototypes. The Method will be applied to other internal indexes, algorithms and fuzzy partitions.

6 Acknowledgment

The authors thank to National Council for Scientific and Technological Development (CNPq) and Federal University of Grande Dourados (UFGD) for the financial support.

7 References

Alzahrani, A. and Rawat, D. B. (2019). Comparative Study of Machine Learning Algorithms for SMS Spam Detection, SoutheastCon, Huntsville, AL, USA, 1-6.

Bhowmik, M. K., Debnath, T., Bhattacharjee, D. and Dutta, P. (2019). EF-Index: Determining number of clusters (K) to estimate number of segments (S) in an image, Image and Vision Computing 88: 29-40.

Calinski, T. and Harabasz, J. (1974). A dendrite method for cluster analysis, Communications in Statistics – Theory and Methods, 3(1): 1–27.

Carmichael, J. W. and Julius, R. S. (1968). Finding Natural Clusters, Systematic Biology, 17(2): 144–150.

Caruso, G., Gattone, S.A., Fortuna, F. and Di Battista, T. (2021). Cluster Analysis for mixed data: An application to credit risk evaluation, Socio-Economic Planning Sciences 73: 100850.

Chen , Z., Qi, Z., Meng, F., Cui, L. and Shi, Y. (2015). Image Segmentation via Improving Clustering Algorithms with Density and Distance, Procedia Computer Science 55: 1015-1022.

Cui, H., Xie, M., Cai, Y., Huang, X. and Liu, Y. (2014). Cluster validity index for adaptive clustering algorithms, IET Communications 8(13): 2256-2263.

Davies, D. L. and Bouldin, D. W. (1979). A Cluster Separation Measure, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1(2): 224–227.

de Oliveira, J. V. and Pedrycz, W. (2007). Advances in Fuzzy Clustering and its Applications. John Wiley and Sons Ltd, England.

de Oliveira, J. V., Szabo, A. and de Castro, L. N. (2017). Particle Swarm Clustering in clustering ensembles: Exploiting pruning and alignment free consensus, Applied Soft Computing 55: 141-153.

Derrac, J., Garcia, S., Molina, D. and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1: 3-18.

Desgraupes, B. (2017). Clustering Indices. Package clusterCrit for R, University Paris Ouest - Lab Modal'X.

Dunn J. C. (1974). Well separated clusters and optimal fuzzy partitions, Journal of Cybernetics 4: 95–104.

Han, J. and Kamber, M. (2006). Data Mining: Concepts and Techniques, 2nd ed., Elsevier, San Francisco.

Elayaraja, E., Thangavel, K., Ramya, B. and Chitralegha. M. (2012). Extraction of Motif Patterns from Protein Sequence Using Rough- K-Means Algorithm, Procedia Engineering 30: 814-820.

Govender, P. and Sivakumar, V. (2020). Application of k-means and hierarchical clustering techniques for analysis of air pollution: A review (1980–2019), Atmospheric Pollution Research 11(1), 40-56.

Hamasuna, Y., Ozaki, R. and Endo, Y. (2017). A study on cluster validity measures for clustering network data, Joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS), Otsu, 1-6.

He, H., Tan, Y. and Fujimoto, K. (2016). Estimation of optimal cluster number for fuzzy clustering with combined fuzzy entropy index, IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Vancouver, BC, 697-703.

Jain, A.K. (2010). Data Clustering: 50 Years Beyond K-means, Pattern Recognition Letters 31(8): 651-666.

Jain, A. K., Murty, M. N. and Flynn, P. J. (1999). Data Clustering: A Review. ACM Computing Surveys (CSUR), 264-323.

Javed, K., Babri, H. A. and Saeed, M. (2008). The behavior of k-Means: An empirical study, Second International Conference on Electrical Engineering, Lahore, 1-6.

Kannan, S. R., Devi, R., Ramathilagam, S. and Takezawa, K. (2013). Effective FCM noise clustering algorithms in medical images. Computers in Biology and Medicine, 43(2), 73-83.

Kapil, S. and Chawla, M. (2016). Performance evaluation of K-means clustering algorithm with various distance metrics. IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, 1-4.

Kesemen, O., Tezel, Ö. Özkul, E., Tiryaki, B. K. and Agayev, E. (2017). A comparison of validity indices on fuzzy C-means clustering algorithm for directional data. 25th Signal Processing and Communications Applications Conference (SIU), Antalya, 1-4.

Khanmohammadi, S., Adibeig, N. and Shanehbandy, S. (2017). An improved overlapping k-means clustering method for medical applications, Expert Systems with Applications 67: 12-18.

Li, H., He, H. and Wen, Y. (2015). Dynamic particle swarm optimization and K-means clustering algorithm for image segmentation. Optik, 126(24): 4817-4822.

Liu, W., Wang, B., Glassey, J., Martin, E. and Zhao, J. (2009). A novel methodology for finding the regulation on gene expression data. Progress in Natural Science, 19(2), 267-272.

Liu, Y., Li, Z., Xiong, H., Gao, X. and Wu, J. (2010). Understanding of Internal Clustering Validation Measures, IEEE International Conference on Data Mining, Sydney, NSW, 911-916.

MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, 281–297.

Maulik, U. and Bandyopadhyay, S. (2002). Performance evaluation of some clustering algorithms and validity indices, IEEE Transactions on Pattern Analysis and Machine Intelligence 24(12): 1650-1654.

Mehdizadeh, E. and Tavakkoli-Moghaddam, R. (2007). A hybrid fuzzy clustering PSO algorithm for a clustering supplier problem, IEEE International Conference on Industrial Engineering and Engineering Management, 1466-1470.

Nagy, G. (1968). State of the art in pattern recognition, Proc. IEEE. 56(5), 836-863.

Pakhira, M. K., Bandyopadhyay, S. and Maulik, U. (2004). Validity index for crisp and fuzzy clusters, Pattern Recognition 37(3): 487-501.

Pal, N. R. and Bezdek, J. C. (1995). On cluster validity for the fuzzy c-means model. IEEE Transactions on Fuzzy Systems 3(3): 370-379.

Prasetyo, H. and Purwarianti, A. (2014). Comparison of distance and dissimilarity measures for clustering data with mix attribute types, 1st International Conference on Information Technology, Computer, and Electrical Engineering, Semarang, 276-280.

Rajee, A. M. and Francis, F. S. (2013). A Study on Outlier distance and SSE with multidimensional datasets in K-means clustering, 13th International Conference on Advanced Computing (ICoAC), Chennai, 33-36.

Rojas-Thomas, J., Santos, M., Mora, M. and Duro, N. (2019). Performance analysis of clustering internal validation indexes with asymmetric clusters, IEEE Latin America Transactions 17(5): 807-814.

Rousseeuw, P. J. (1987). Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. Computational and Applied Mathematics, 20: 53–65.

Ruspini, E. H., Bezdek, J. C. and Keller, J. M. (2019). Fuzzy Clustering: A Historical Perspective, IEEE Computational Intelligence Magazine 14(1): 45-55.

Shapiro, S.S. and Wilk, M.B. (1965). An Analysis of Variance Test for Normality. Biometrika, 52: 591-611.

Su, J., Liu, S. and Song, J. (2017). A segmentation method based on HMRF for the aided diagnosis of acute myeloid leukemia, Computer Methods and Programs in Biomedicine 152: 115-123.

Umadevi, S. and Marseline, K. S. J. (2017). A survey on data mining classification algorithms, International Conference on Signal Processing and Communication (ICSPC), Coimbatore, 264-268.

Vendramin, L., Campello, R. J. G. B. and Hruschka, E. R. (2010). Relative Clustering Validity Criteria: A Comparative Overview. Statistical Analysis and Data Mining, Wiley, 3, 209-235.

Vij, A. and Khandnor, P. (2016). Validity of internal cluster indices. International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), Bangalore, 388-395.

Visalakshi, N. K. and Suguna, J. (2009). K-means clustering using Max-min distance measure, Annual Meeting of the North American Fuzzy Information Processing Society, Cincinnati, OH, 1-6.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. Biometrics Bulletin, 1(6): 80-83.

Xiao, J., Lu, J., and Li, X. (2017). Davies Bouldin Index Based Hierarchical Initialization K-means, Intelligent Data Analysis 21(6): 327-1338.

Xie, X. L., Beni, G. (1991). A validity measure for fuzzy clustering, IEEE Trans. Pattern Anal. Mach. Intell. 13: 841–847.

Zhao, Q. and Fränti, P. (2014). WB-index: A sum-of-squares based index for cluster validity, Data & Knowledge Engineering 92: 77-89.