Improved Sparse Prototyping for Relational K-means

Safouane Cherki, Parisa Rastin, Guénaël Cabanes, and Matei Basarab
LIPN-CNRS, UMR 7030, Université Paris 13, 99 Avenue J-B. Clément, 93430 Villetaneuse, France
cherki@lipn.univ-paris13.fr, rastin@lipn.univ-paris13.fr, cabanes@lipn.univ-paris13.fr, matei@lipn.univ-paris13.fr

***Abstract*—This study investigates the possibility of improving K-means algorithm for non-vector data. It calls attention to the changes a similarity measure enforce compared to vectorial K-means, and aims to understand and take advantage of the opportunities offered by sparse prototyping for K-means. We propose here a new algorithm of clustering for relational data, i.e. data described by their relations to each other (usually their similarities). This algorithm computes a set of sparse prototypes to represent the data structure. The results are promising : the clustering quality of the sparse variation is similar to the traditional K-means, and the processing cost for high dimensions is lower than other relational algorithms.**

***Index Terms*—K-means, Dissimilarity, Relational, Prototype, Sparse.**

I. Introduction

K-MEANS is one of the most widely used clustering algorithm [1], because it has been used for a long time, is very well understood and achieve acceptable results within reasonable time. However, input data is not always available in vectorial form. For example pictures, texts, networks, trajectories or navigation patterns on a web site [2] cannot be reduced to vector without losing information. A common solution is then to define an adapted similarity measure to compare these objects. The data are thus defined from their relations to each other and are called ”relational data”. However, in that case, we don’t have points in a vectorial space any more, nor can we convert them to fit into one, and most clustering algorithms are inappropriate.

Relational clustering algorithms form a family of method adapted to relational data. Few work have been done yet in this domain, but some authors have worked on using K-means with relational data, such as R.J. Hathaway’s [3] relational duals. The idea studied by Hathaway was to consider the K prototypes as a linear combination of the input data. But as far as processing power and memory usage are concerned, this implementation is very expensive ($\mathcal{O}(N^2)$ for N samples), making it unusable for large sets of data. F. Rossi [4] proposed to enforce sparser prototyping by considering the prototypes as linear combinations of support points, a subset from the data set, succeeding in improving the complexity. However, the support points selected by Rossi are specific to the cluster represented by the prototype, and we believe it is possible de decrease complexity and memory cost even more. The main idea is to use one set of fixed prototype representatives, that we will call support points, instead of a multiple cluster-dependent sets. This can help us decrease the quantity of calculations made and also the need of keeping the full distance matrix in memory.

In this work, we will make our study based on the basis laid by F. Rossi in [4], to achieve an alternative method to select the support points to train a relational K-means : The objective is to use one unique set of representatives through the whole learning process, independently from the clusters, in a way that all the prototypes are represented by the same subset of observations, improving the cost of computing new support points and more importantly, the cost of computing the new prototypes. On the memory side, working with fixed support points, make saving the full dataset distance matrix unnecessary. That means instead of keeping a memory a full $N\times$ matrix, we only keep a $N \times (D+1)$ matrix, where D is the dimension of the space containing the objects.

We will start this paper by introducing, in Section II, the K-Means algorithm and the naive implementation of the relational version. Then we will examine in Section III the sparse variation that lay ground for the support points version of the algorithm, followed by the fixed representatives alternative (Rossi’s version). Next, we study in Section IV the usage of fixed support points and we propose a new algorithm. Section V presents and discuss the results of experimental implementations of the methods. We conclude in section VI.

II. K-means and its relational variant

We recall below the K-means algorithm and we define dissimilarity data. Additionally, we will present the relational K-means algorithm used as a frame for the sparse variation presented later. In the following, N is the number of observations $x^{(1)}, ..., x^{(N)}$ from a $D-dimensional$ pseudo-Euclidean input space E_*.

A. K-means

K-means is a prototype based algorithm [5], which aims to minimize the within-cluster sum of squares. The data in each cluster are represented by a prototype (i.e. a centroid).

Given the set of observations, K-means clustering compute a partition S of the N observations into K ($\leq$ N) sets, so as to find :

$$\arg\min_S \sum_{k=1}^{K} \sum_{x \in S_k} \|x - \mu_k\|^2 \tag{1}$$

where μ_k is the prototype of cluster k.

The most used method for this minimization is Lloyd's algorithm [5], and it uses an iterative refinement technique. After an initialization of K-means, we assign each point to the closest prototype (*Assignment step*), then we compute the prototypes taking into account the new clusters (*Update step*). We iterate this two steps until the assignments stop changing.

An alternative version of K-means uses points from data as prototypes, using medoids instead of centroids. Partitioning Around Medoids (PAM) [6] is more robust to noise and outliers than regular K-means, mainly because it works directly with pairwise dissimilarities. On the other side, PAM update step can cause a cluster overlapping, that cannot be untangled in the following iterations, because of the limited number of potentials centers compared with K-means.

B. K-means++ initialization variation :

K-means algorithm is sensitive to the initial set-up of centroids, so instead of random starting centers, Arthur's and Vassilvitskii's K-means++ [7] propose a variant for choosing the initial values. Let $D(x^{(i)}, \mu_j)$ denote the distance from a data point $x^{(i)}$ to the center μ_j (the prototypes). Then, the algorithm is defined as follow :

1) Choose one center μ_1 uniformly at random from among the data points.
2) Choose the farthest data point from μ_1 (highest $D(x^{(i)}, \mu_1)$) as a new center μ_2.
3) For each data point $x^{(i)}$, store the distance separating it from the closest centroid ($\min_j D(x^{(i)}, \mu_j)$) for all j).
4) Choose the point with the highest stored distance as the new center μ_j
5) Repeat Steps 3 and 4 until K centers have been chosen.

It means that at each iteration, the next centroid is the furthest point, selected from the set of the closest points from the previous centroids.

C. Lloyd Algorithm implementation:

The standard formulation of K-means is presented in Algorithm 1, where μ_k represent cluster centroids for i in $(1, .., K)$, and $c^{(i)}$ represent labels for each sample.

Algorithm 1 K-means Algorithm

```
1: Initialize cluster centroids $\mu_1, ..., \mu_K$ from $E_*$
2: while Not Convergence do
3:     for $i \leftarrow 1, N$ do
4:         $c^i \leftarrow \arg\min_j \|x^{(i)} - \mu_j\|^2$
5:     for $j \leftarrow 1, K$ do
6:         $\mu_j \leftarrow \frac{\sum_{n=1}^{N} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{n=1}^{N} 1\{c^{(i)} = j\}}$
```

To initialize the cluster centroids (first step), we could choose K training examples randomly, and set the cluster centroids to be equal to the values of these examples. Another method would be using the K-means++ initialization described previously. The inner-loop of the algorithm repeatedly comprises out two steps:

- Assignment : The sample i is affected to the closest cluster centroid μ_j
- Update : The cluster centroid μ_j is moved to the mean of the corresponding points.

D. Dissimilarity data

Not all data is or can be described by a set of variables with known values. Another way of representing data is by the relations within and between samples of a dataset. They can be quantified with dissimilarity measures amongst every two observations, conventionally resulting in a positive real number, for which higher is the value, more different the observations are, and respectively small values represent very similar data. The minimal constraints on a dissimilarity measure $d : (x, y) \longrightarrow d(x, y)$ are [8] :

- Non negativity : $d(x, y) \geq 0$ for all x and y
- Symmetry : $d(x, y) = d(y, x)$ for all x and y
- Reflexivity : $d(x, x) = 0$ for all x

Thus, the input dataset for such data, in our case, is a dissimilarity matrix, for which each cell represent the disparity between two samples. Consequently, the dissimilarity matrix D for an N elements dataset, is :

- Square : $N \times N$ matrix
- Hollow : $d(i, i) = 0$ for all i
- Symetric : $d(i, j) = d(j, i)$ for all i and j
- Non-negative : $d(i, j) \geq 0$ for all i and j

E. Relational K-means formulation

Relational K-means cover the case of relational datasets, where non-vector points are represented as a similarity (or dissimilarity) matrix D. The difference lies in the definition of the prototypes, which are now defined as a linear combination of data instead of a vector in the data space. Considering a normalized linear combination of the observations, namely, $y = \sum_{i=1}^{N} \alpha_i \cdot x^{(i)}$ with $\alpha = (\alpha_1..\alpha_N) \in \mathbb{R}^N$, $x^{(i)} \in \mathbb{R}^N$ and $\sum_{i=1}^{N} \alpha_i = 1$, we can deduce the dissimilarity [8] [9] :

$$d(\alpha, x^{(i)}) = (D \cdot \alpha)^{(i)} - \frac{1}{2}\alpha^T \cdot D \cdot \alpha \tag{2}$$

The idea behind this formula, is that we can compute the distance of a point $x^{(i)}$, from a point represented by the coefficients α $d(\alpha, x^{(i)})$, without having to compute the vectorial form (which may exist or not) of each point of the space. All we need in this case is the coefficients α and the distance matrix D.

The relational K-means follows the same concept of the vector variant, i.e., alternating assignment and update, while minimizing the within class variance. A solution to the underlying minimization problem in K-means would be [4] :

$$\alpha^{(k)} = \frac{1}{|C_k|}(\delta_{k,1}, ..., \delta_{k,N}) \tag{3}$$

where $|C_k|$ is te number of points in the cluster k, and $\delta_{k,1}$ correspond to the Kronecker delta ($\delta_{i,j} = 1$ for $i = j$ and $\delta_{i,j} = 0$ otherwise)

A standard formulation would give us the Algorithm 2.

Algorithm 2 Naive relational K-means algorithm

```
1: Initialize cluster centroids
2: Compute the $\alpha^{(K)}$ using (3)
3: while Not Convergence do
4:     for $i \leftarrow 1, N$ do
5:         for $k \leftarrow 1, K$ do
6:             Compute $d(\alpha^{(k)}, x^{(i)})$ using (2)
7:             Assign $x^{(i)}$ to $C_k$ for minimal $d(\alpha^{(k)}, x^{(i)})$
8:     Update $\alpha^{(K)}$ using (3)
```

III. Sparse prototypes representation for K-means

F. Rossi proved that the sparse variation of relational K-means is more efficient than an instinctive adaptation of K-means for dissimilarity data [4]. As such, we will build our version of the algorithm over foundations laid by Rossi, to enforce sparser prototypes for K-means, using a center of mass approximation. This grounding is presented in the following section.

A. Sparse prototypes representation

For a sparse representation, the prototypes are represented only by a limited number of samples, which means that prototypes are a linear combination of a relatively small portion $J_k = (j_{k,1}, ..., j_{k,P})$ of the corresponding cluster, where $x^{(j_{k,p})} \in C_k$ for all p, P being the number of support points chosen.
We are looking for a representation of μ_k, the center of mass of C_k, as a normalized linear combination of the support points $x^{(j_{k,p})}$. That can be translated into :

For any centroid μ_k, there is $\beta^{(k)} \in \mathbb{R}^P$

$$\text{such as } \mu_k = \sum_{i=1}^{P} \beta_i^{(k)} \cdot x^{(j_i)} \text{ and } \sum_{i=1}^{n} \beta_i = 1 \quad (4)$$

B. Centroid representation

Considering that the general principle of K-means is the same for this sparse variation, the minimization problem for each cluster can be written as :

$$(P_k) \min_{\beta \in \mathbb{R}^n} \sum_{x^{(i)} \in C_k} d(\beta, x^{(i)})$$

$$where \quad \sum_{i=1}^{n} \beta_i = 1; \quad \forall j \notin J_k \quad \beta_j = 0 \quad (5)$$

By considering $s_{k,j} = \sum_{i \in C_k} d_{ij}$, $D_J = (d_{uv})_{u \in J, v \in J}$ and $s_{k,J} = (s_{k,j_{k,1}}, ..., s_{k,j_{k,P}})^T$, then applying the method of Lagrange multipliers to find the local minima for (5), the problem is reduced to the equation [4] :

$$\nabla L_k = s_{k,J} - |C_k| D_J \cdot \beta + \lambda \vec{1} = 0 \quad (6)$$

where $\vec{1} = (1, ..., 1)^T \in \mathbb{R}^P$ and $\sum_{i=1}^{n} \beta_i = 1$. The problem for which we are looking for a local minimum can be represented by the linear system below [4] :

$$\begin{bmatrix} |C_k| D_J & -\vec{1} \\ \vec{1}^T & 0 \end{bmatrix} \begin{bmatrix} \beta \\ \lambda \end{bmatrix} = \begin{bmatrix} s_{k,J} \\ 1 \end{bmatrix} \quad (7)$$

C. Algorithm

Using the sparse representation presented above, we obtain the Algorithm 3.

Algorithm 3 Sparse relational K-means - with cluster specific support points

```
1: Initialize clusters by affecting a cluster to observations
2: while Not converged do
3:     Select $P$ support points
4:     for $k \leftarrow 1, K$ do
5:         Compute $s_{k,J} = \sum_{i \in C_k} d_{ij}$
6:         Solve equation (7) to get $\beta$
7:     Compute $(D\beta^{(k)})$ for all $k$ and $i$
8:     Compute $-\frac{1}{2}\beta^{(k)T} D \, \beta^{(k)}$ for all $k$
9:     for $k \leftarrow 1, K$ do
10:        for $i \leftarrow 1, N$ do Compute $d(\beta^{(k)}, x^{(i)})$
11:        Assign $x^{(i)}$ to it's closest prototype
```

IV. Sparse Relational K-means

This section present the new improved sparse relational K-means using a single set of support points for all the clusters.

A. Support points selection

In an Euclidean space with a dimension D, we need $D+1$ non-aligned samples to determine the position of any point in the space using only distances. In our case, we want a good representation of the clusters prototypes based on their distances to the support points. Of course, choosing $d+1$ support point requires the ability to compute the intrinsic dimension of the data from a similarity matrix (see for example [10]). It is also possible to choose a fixed number of support points, depending on the available memory and processing time.

While selecting random support points can give good results in practice, we need to minimize the probability of getting aligned points within the chosen observations. Here, we propose to use the same principle as K-means++ initialization. We select the first support point randomly, the second representative being the furthest one from the first. We continue by pinning down, at each iteration, the closest points from the previous support points, then we select the furthest point selected from this set. Preliminary tests have shown that this initialization increase the quality of the clustering for relational algorithms. We use this initialization for all of the algorithms tested here requiring support points.

B. Centroid representation

Taking into account the reasoning presented in the previous section, the equation (6) is a simplified form of the minimization problem.

$$\frac{1}{|C_k|} s_{k,j} - D_J \cdot \beta + \frac{1}{|C_k|} \lambda \vec{1} = 0 \quad (8)$$

where $\vec{1} = (1, ..., 1)^T \in \mathbb{R}^P$.

By keeping in mind that the support points are not cluster dependent in this situation, the problem can be simplified to the linear system below :

$$\begin{bmatrix} D_J & -\vec{1} \\ \vec{1}^T & 0 \end{bmatrix} \begin{bmatrix} \beta \\ \frac{1}{|C_k|}\lambda \end{bmatrix} = \begin{bmatrix} \frac{1}{|C_k|} s_{k,J} \\ 1 \end{bmatrix} \tag{9}$$

C. Algorithm

Using the sparse representation presented above, we obtain the following algorithm :

Algorithm 4 Sparse relational K-means - with fixed support points

```
1: Initialize clusters by affecting a cluster to observations
2: Select $P$ support points
3: while Not converged do
4:     for $k \leftarrow 1, K$ do
5:         Compute $s_{k,J} = \sum_{i \in C_k} d_{ij}$
6:         Solve equation (9) to get $\beta$
7:     Compute $(D\beta^{(k)})$ for all $k$ and $i$
8:     Compute $-\frac{1}{2}\beta^{(k)T} D \ \beta^{(k)}$ for all $k$
9:     for $k \leftarrow 1, K$ do
10:        for $i \leftarrow 1, N$ do Compute $d(\beta^{(k)}, x^{(i)})$
11:        Assign $x^{(i)}$ to it's closest prototype
```

A variation of the algorithm would consist into separating the system solving step, into inversion of the first term, which can be extracted from the loop considering it is k-independent, and a product of the resulted inverse with the right side term, which will be kept inside the loop.

In other terms, it means that D_J is now independent from k, and the system (9) will become :

$$\begin{bmatrix} \beta \\ \frac{1}{|C_k|}\lambda \end{bmatrix} = \begin{bmatrix} D_J & -\vec{1} \\ \vec{1}^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{|C_k|} s_{k,J} \\ 1 \end{bmatrix} \tag{10}$$

And the resulting algorithm would be as follow :

Algorithm 5 Optimized sparse relational K-means - with fixed support points

```
1: Initialize clusters by affecting a cluster to observations
2: Select $P$ support points
3: while Not converged do
4:     Compute the inverse used in (10)
5:     for $k \leftarrow 1, K$ do
6:         Compute $s_{k,J} = \sum_{i \in C_k} d_{ij}$
7:         Compute the product (10) to get $\beta$
8:     Compute $(D\beta^{(k)})$ for all $k$ and $i$
9:     Compute $-\frac{1}{2}\beta^{(k)T} D \ \beta^{(k)}$ for all $k$
10:    for $k \leftarrow 1, K$ do
11:        for $i \leftarrow 1, N$ do Compute $d(\beta^{(k)}, x^{(i)})$
12:        Assign $x^{(i)}$ to it's closest prototype
```

V. Experimentation and discussions

A. General setting

For the experiments, we used some representative vector datasets, to allow us a comparison with vector K-means. The samples are injected into the relational algorithms after the calculation of corresponding similarity matrices using Euclidean distance. Considering the conclusion of Rossi [4] that sparse algorithms are generally more effective than their naive counterpart, we will analyze the differences between the proposed methods and the previously presented sparse implementations, in addition to the regular vectorial K-means. Thus, we compare the overall performance of Algorithms 4 and 5 (the sparse relational K-means with fixed support point studied in section IV) with the following methods :

- Regular K-means for vector data (Algorithm 1)
- Sparse K-means without support points [4]
- Sparse K-means with cluster-specific support points (Algorithm 3)

All the algorithms are implemented using python 2.7.11. Mathematical computation are performed using the python library numpy 1.10.4, in addition to the module linalg from scipy package for matrix inversion and solving.

We evaluate the clustering results using two external quality index, Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI), and the internal Silhouette Coefficient. The results reported for processing time, and index values are the means of twenty iteration of the algorithms, with a new initialization for each run. It is worth noting, that for each specific iteration, the initialization is common for all the algorithms tested.

The datasets used, while not sufficiently massive to simulate real life problems, should provide enough insights in order to examine an evolution for real problems.

Artificial convex dataset :
A set of 8750 observation artificially generated using sklearn and mdp python libraries. The data is two dimensional and the 5 clusters making the set are convex and well separated.

Artificial non-convex dataset :
A set of 10000 observation artificially generated using sklearn and mdp python libraries. The data is two dimensional and the 5 clusters making the set are non-convex but well separated.

Iris dataset :
The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other. The predicted attribute is the class of iris plants [11].

Digits dataset :
Each datapoint of the set is a rectangular 8x8 box in a gray scale of 16 values representing of a handwritten digit. There is 1797 observations, with 64 features, separated into 10 clusters [11].

Glass dataset :
The data set contains 6 classes for a total of 214 observations, where each custer correspond to a type of glass. A sample is described by 10 attributes [11].

Wine dataset :
These data are the results of a chemical analysis of 178 wines (samples) grown in the same region in Italy but derived from 3 different cultivars (clusters). The analysis determined the quantities of 13 constituents found in each of the three types of wines [11].

For the results reported next, we use the abbreviations defined in table I for each dataset, and the names in table II for algorithms names.

TABLE I: Datasets abbreviations

Artificial convex dataset	Convex
Iris dataset	Iris
Digits dataset	Digit
Glass dataset	Glass
Wine dataset	Wine

TABLE II: Algorithms nomination

Vectorial K-means	K-means
Relational K-means using sparsity propriety without support points	Rel KM w/ Sparsity
Sparse relational K-means with cluster specific support points	Sparse KM w/ C-SP
Sparse relational K-means with fixed support points	Sparse KM 1
Optimized sparse relational K-means with fixed support points	Sparse KM 2

B. Complexity summary :

K-means problem was shown to be NP-hard in Euclidian space [12] and for the plane [13]. It's computational complexity for a dataset of N input is [14]: $\mathcal{O}(N^{DK+1} \cdot \log N)$.
Lloyd algorithm [5], which is the standard approach for K-means clustering, is often described as of linear complexity, and is given as $\mathcal{O}(N \cdot K \cdot D)$ per iterations.
The naive relational K-means, whose formulation was presented in the algorithm 2, has a complexity $\mathcal{O}(K \cdot N^2)$, which can be expensive for big datasets.
But, considering (3), the matrix α is sparse. That means, we can reduce complexity to $\mathcal{O}(N^2)$ by computing only nonzero terms in (2) [4]. Because for each cluster k, $\alpha^{(k)}$ contains only $|C_k|$ non-zero terms, and $\sum_{k=1}^{K}|C_k| = N$.
For the sparse relational K-means with cluster-dependent support points in algorithm 3, the most expensive operation is the computation of β in $\mathcal{O}(N \cdot K \cdot P + K \cdot P^3)$. Calculation of each $(D \cdot \beta^{(k)})$ and each $-\frac{1}{2}\beta^{(k)T} \cdot D \cdot \beta^{(k)}$ cost $\mathcal{O}(P)$. Finally, the complexity of the assignment step is $\mathcal{O}(N \cdot K)$.
For our variation of Rossi's sparse K-means, the sparse K-means with fixed support points, solving the system (9) cost $\mathcal{O}(P^3)$ for each cluster k. Therefore, algorithm 4 complexity is $\mathcal{O}(K \cdot P^3)$. Finally, when we use the equation 10, we replace the k solve operation (costing $\mathcal{O}(K \cdot P^3)$), with one inverse and k matrix multiplications (asymptotically costing $\mathcal{O}(K \cdot P^{2.4})$ for large matrices [15], [16], [17]). In addition, the number P of support points is lower in our method (where $P = D + 1$) than in regular sparse algorithm (where $P = K \times D$ [4]). The complexity has therefore been improved to $\mathcal{O}(N \cdot K \cdot P + K \cdot P^{2.4})$, which is beneficial when the number of clusters K get higher.

Table III shows the complexity of the different algorithms. Relational method are usually more complex than vectorial method, but among them KM2 is the less complex. In addition, in KM1 and KM2, P is around K time smaller than in the other relational algorithms.

TABLE III: Computational complexity

Algorithm	Complexity
K-means	O(NKD)
Rel KM w/ Sparsity	O(N^2)
Sparse KM w/ C-SP	O($NKP + KP^3$)
Sparse KM 1	O($NKP + KP^3$)
Sparse KM 2	O($NKP + KP^{2.4}$)

C. Theoretical Memory usage :

For vectorial K-means, we notice that we need to commit to memory the training dataset ($N\ timesD$) and the prototypes vectors ($K \times D$). And unless the number of object in the set is inferior to the number of features ($N < D$), vectorial K-means is the less memory hungry of all the derived algorithms. For the naive implementation of relational K-means, we need to store the distance matrix in order to compute the prototype-samples distances, plus coefficients of the K prototypes ($N \times K$). The full dataset is also needed for sparse K-means with cluster specific support points, but to compute the prototypes coefficients, we need a representatives-samples distance matrix for each cluster ($N \times P \times K$), and finally we need to store the K support points.

As for the sparse variants with fixed support points, we do not need the whole distance matrix to select support points, since they are fixed at the start, and the only condition is they must not be aligned to be able to represent all the points in the space. Considering that we only need a set of $D + 1$ representatives instead of a set for each cluster, the gain can become substantial for high dimensional datasets with many clusters.

The stored matrices are reported in the table IV.

TABLE IV: Memory usage

Algorithm	Stored Matrices Dimensions
K-means	$N \times D + K \times D$
Rel KM w/ Sparsity	$N \times N + N \times K$
Sparse KM w/ C-SP	$N \times N + N \times K^2 \times D + K^2 \times D$
Sparse KM 1	$N \times K + K \times D$
Sparse KM 2	$N \times K + K \times D$

D. Experimental Results

We computed the processing time, memory consumption and clustering quality for each method and each dataset.

1) Processing time (Table V) : Each algorithm was implemented in a different function, but the prototypes initialization is common for all the functions, and is not included in the duration computed. The time needed for each method/function to complete the learning was calculated using the package

timeit : we start a timer exactly before running an algorithm script, and we stop it after saving the return results into a variable.
The computations are tested on a Windows 7(x64) machine, with a dual-core CPU clocked at 2.50Ghz (i5-2450M). The program is not multi-threaded.

TABLE V: Processing durations results per iteration (ms)

	Convex	Iris	Digit	Glass	Wine
K-means	475.20	3.98	39.52	4.62	3.78
Rel KM w/ Sparsity	119321.45	59.50	3971.75	53.80	38.25
Sparse KM w/ C-SP	353.93	5.43	392.40	13.26	10.95
Sparse KM 1	409.25	3.80	340.16	9.03	8.98
Sparse KM 2	238.80	3.58	351.54	8.41	8.40

2) Memory usage (Table VI): For memory monitoring, we used python package *memory_profiler*. It provides us with a line-by-line analysis of memory usage fors the designated function. Considering that python memory management is OS dependent, and cannot always be predicted as it does not always release freed memory, we decided to launch each script independently from the other, which mean that unlike the other measures reported, memory usage numbers are based on a different initialization, and are the mean of 5 executions.

TABLE VI: Memory usage results (Gb)

	Convex	Iris	Digit	Glass	Wine
K-means	0.4950	0.0080	0.0940	0.0136	0.0088
Rel KM w/ Sparsity	2.1234	0.0160	0.1680	0.0238	0.0176
Sparse KM w/ C-SP	4.1706	0.4220	1.9148	0.5164	0.6898
Sparse KM 1	2.8736	0.3360	0.8386	0.4626	0.4432
Sparse KM 2	3.1558	0.3224	0.9472	0.4642	0.4760

3) Adjusted rand index (Table VII): ARI [18] is an external measure of the similarity between two data clusterings, by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings. The adjusted Rand index have a value close to 0.0 for random labeling and exactly 1.0 when the clusterings are identical.

TABLE VII: Adjusted rand index results
Mean±Standard Deviation

	Convex	Iris	Digit	Glass	Wine
K-means	.99 ± .00	.72 ± .00	.57 ± .03	.46 ± .02	.35 ± .00
Rel KM w/ Sparsity	.99 ± .00	.70 ± .02	.39 ± .03	.45 ± .03	.32 ± .01
Sparse KM w/ C-SP	.99 ± .03	.74 ± .01	.57 ± .04	.45 ± .02	.34 ± .01
Sparse KM 1	.99 ± .02	.73 ± .01	.57 ± .04	.47 ± .03	.34 ± .03
Sparse KM 2	.99 ± .00	.73 ± .04	.56 ± .03	.46 ± .04	.35 ± .04

4) Normalized Mutual Information (Table VIII): NMI [19] is an external clustering quality index, that quantify the mutual information between two clustering, which is a measure of the similarity between two labels of the same data. The results are then normalized to scale the results between 0 (no mutual information) and 1 (perfect correlation).

5) Silhouette (Table IX): Silhouette [20] is an internal method of interpretation and validation of consistency within clusters of data. Silhouette coefficients are measures of how similar a sample is to its own cluster (cohesion) compared to other clusters (separation). The mean coefficient of all samples is then computed. The silhouette ranges from -1 to 1, where a high value indicates that the observation is well matched to its own cluster and poorly matched to neighboring clusters.

TABLE VIII: Normalized Mutual Information Results
Mean±Standard Deviation

	Convex	Iris	Digit	Glass	Wine
K-means	.99 ± .00	.75 ± .00	.70 ± .01	.69 ± .02	.42 ± .00
Rel KM w/ Sparsity	.99 ± .00	.76 ± .02	.56 ± .01	.67 ± .02	.40 ± .01
Sparse KM w/ C-SP	.99 ± .03	.77 ± .02	.70 ± .01	.68 ± .01	.42 ± .01
Sparse KM 1	.99 ± .02	.76 ± .01	.70 ± .01	.70 ± .02	.41 ± .02
Sparse KM 2	.99 ± .00	.76 ± .02	.70 ± .01	.69 ± .03	.42 ± .02

TABLE IX: Silhouette Index Results
Mean±Standard Deviation

	Convex	Iris	Digit	Glass	Wine
K-means	.79 ± .00	.55 ± .00	.17 ± .00	.51 ± .00	.56 ± .00
Rel KM w/ Sparsity	.79 ± .00	.54 ± .00	.12 ± .01	.50 ± .00	.56 ± .01
Sparse KM w/ C-SP	.79 ± .02	.54 ± .00	.17 ± .00	.51 ± .00	.57 ± .01
Sparse KM 1	.79 ± .02	.54 ± .00	.17 ± .00	.51 ± .00	.55 ± .02
Sparse KM 2	.79 ± .00	.54 ± .01	.17 ± .01	.51 ± .00	.55 ± .03

E. Discussion

In our experiments, the external quality indexes ARI (table VII) and NMI (table VIII) are on the same levels for the different versions of K-means tested. More specifically, that means our sparse K-means implementation yields as good results as vectorial K-means and the other relational method. In the same manner, the internal quality index Silhouette (table IX) shows that, aside from Digit dataset, the clusters created are reasonably structured, and again the results are comparable with vector K-means, and Rossi's sparsing algorithm.

Concerning processing cost, as predicted with the complexity values already mentioned, the relational variations of K-means can be more expensive time-wise than the vectorial version, although it is not always the case. Our methods (sparse KM1 and especially sparse KM2) achieve good performances in comparisons to the other relational algorithms. Rel KM w/ Sparcity, in particular, can be very slow, especially when the size of the dataset increase. One can note that the cost polynomially increases with the dimension of the dataset. That can be explained by the decision to select $dimension + 1$ support points to represent the prototypes.

As far as memory usage is concerned, we notice at first sight that the vectorial version is less demanding than the relational ones, which can be explained by the storage of pairwise distances matrix. For the relational methods, the less memory consuming is the Rel KM w/ Sparcity, but at the expense of time processing, as shown earlier. Among the remaining methods, Rel KM1 and even more Rel KM2 are clearly less memory consuming: we observe a noticeable memory gain when using common support points over cluster-dedicated representatives.

To summarize, regular sparsity implementation in relational K-means consume a minimal amount of memory, but the processing durations are extremely high, which render this option unusable. The sparse algorithm with distinct support points is time-effective, but the results showed that the computation time can be improved with the usage of fixed support points

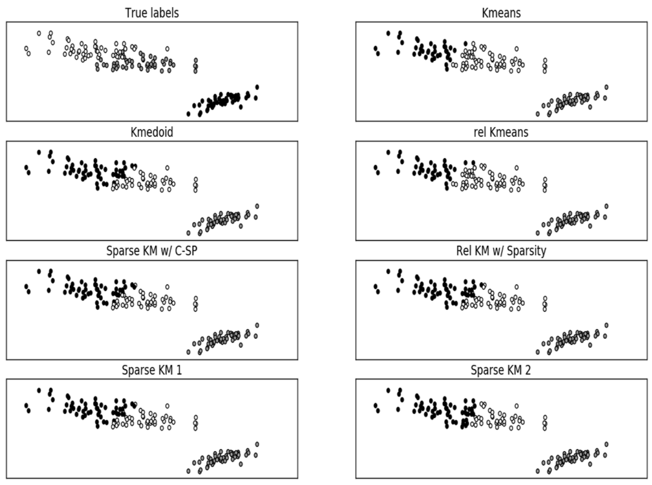


Fig. 1: Representation of clustering results for Iris Dataset (using MDS)

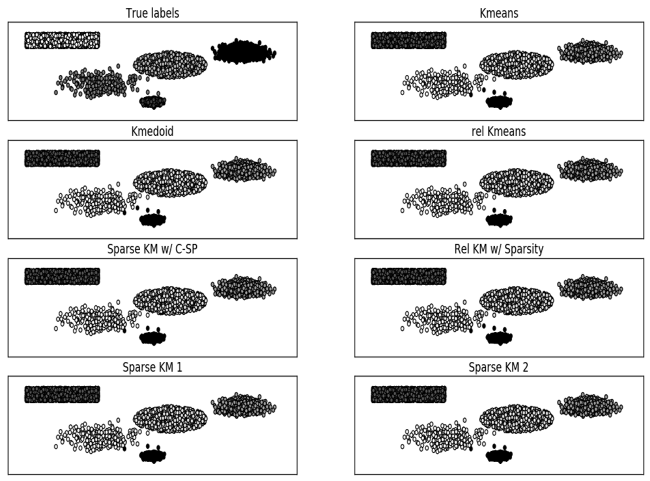


Fig. 2: Representation of clustering results for the artificial convex Dataset

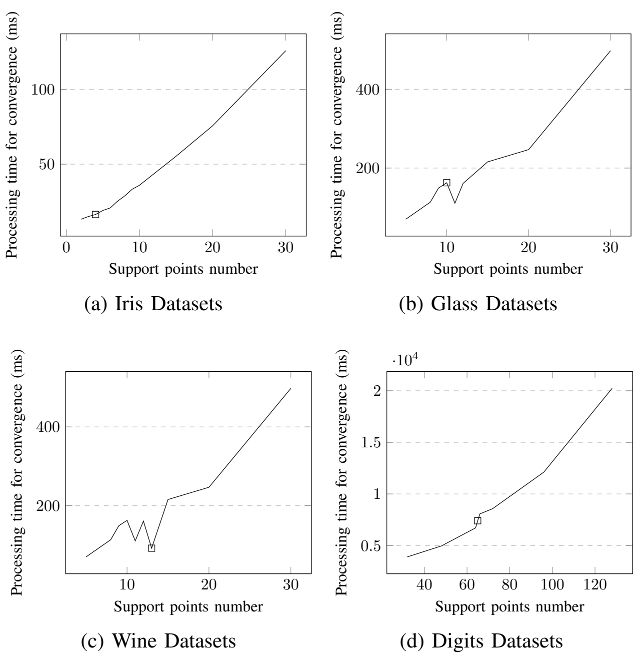
as we propose here. In addition, the former is more expensive on the memory side by a difference ranging from 4% (Iris dataset), to 33% (Glass dataset), comparing with the latter. The clustering quality is similar for both of them and with the vectorial K-means (see figures 1 and 2 for an example).

We conclude then that the sparse relational K-means with fixed representatives is a good candidate to optimize memory usage and processing time for relational data.

F. Impact of support points number

Our experience with real datasets show that with random $dimension + 1$ representatives initialization, the sparse K-means with shared support points converge in an average of 6,25 iterations. Meanwhile, an initial setting inspired from K-means++ of representatives converge in 5,05 iterations (mean over 20 runs).

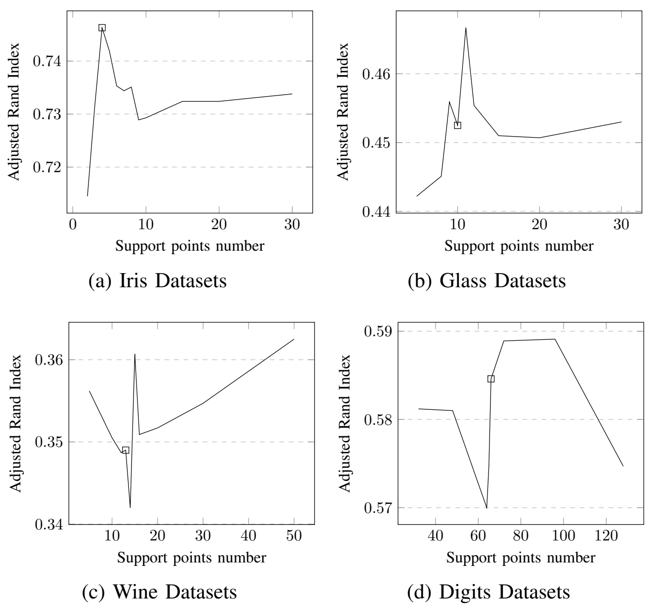
When changing the number of support points, we notice a small improvement in clustering quality before a decrease when the number is too high (figure 4).



(a) Iris Datasets (b) Glass Datasets

(c) Wine Datasets (d) Digits Datasets

Fig. 3: Effect of support points number on processing time



(a) Iris Datasets (b) Glass Datasets

(c) Wine Datasets (d) Digits Datasets

Fig. 4: Effect of support points number on Adjusted Rand Index values

There is also an increase in the time needed for convergence (figure 3). We notice that a good compromise between quality and time complexity is when the number of representatives is equal to $dimension + 1$ of each dataset (represented by a square in 4 and in 3).

VI. Conclusion

We have reviewed in this paper the main variants of K-means, for vector and relational data. We have shown that the variants differ more in terms of complexity than in terms of logic behind the algorithms.

Following F. Rossi [4], we have shown that different relational and sparse versions of K-means are identical in their clustering quality. Taking into account computational aspects and known experimental results, the proposed sparse relational K-means with common support points for all the clusters is a compelling compromise to optimize memory usage and processing duration, while resulting in a clustering quality on the same magnitude as the other K-means versions.

As for future work, we would like to optimize the construction of distance matrix of support points, such as to reduce the complexity even further.

However, the practical usefulness of fixed support points is increased when we take into account the application to dynamic data, changing asynchronously as new information become available. In that case, with predetermined support points, changing clusters would not be a deterrent to adapt prototype-based learning algorithms. This open the way to incremental and dynamic clustering of relational data streams.

References

[1] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651 – 666, 2010, award winning papers from the 19th International Conference on Pattern Recognition (ICPR)19th International Conference in Pattern Recognition (ICPR). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865509002323

[2] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Model-based clustering and visualization of navigation patterns on a web site," *Data Mining and Knowledge Discovery*, vol. 7, no. 4, pp. 399–424, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1024992613384

[3] R. J. Hathaway, J. W. Davenport, and J. C. Bezdek, "Relational duals of the c-means clustering algorithms," *Pattern Recognition*, vol. 22, no. 2, pp. 205–212, 1989. [Online]. Available: http://dx.doi.org/10.1016/0031-3203(89)90066-6

[4] F. Rossi, A. Hasenfuss, and B. Hammer, "Accelerating relational clustering algorithms with sparse prototype representation," in *Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM 07)*, Bielefeld (Germany), 9 2007.

[5] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theor.*, vol. 28, no. 2, pp. 129–137, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.1982.1056489

[6] L. Kaufman and P. Rousseeuw, *Clustering by means of medoids*. North-Holland, 1987.

[7] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. [Online]. Available: http://dl.acm.org/citation.cfm?id=1283383.1283494

[8] F. Rossi, "How many dissimilarity/kernel self organizing map variants do we need?" in *Advances in Self-Organizing Maps and Learning Vector Quantization (Proceedings of the 10th International Workshop on Self Organizing Maps, WSOM 2014)*, ser. Advances in Intelligent Systems and Computing, T. Villmann, F.-M. Schleif, M. Kaden, and M. Lange, Eds., vol. 295. Mittweida (Germany): Springer International Publishing, 7 2014, pp. 3–23.

[9] B. Hammer, A. Hasenfuss, F. Rossi, and M. Strickert, "Topographic Processing of Relational Data," in *Proceedings of 6th International Workshop on Self-Organizing Maps*, 2007.

[10] C. Bustos, G. Navarro, N. Reyes, and R. Paredes, *An Empirical Evaluation of Intrinsic Dimension Estimators*. Cham: Springer International Publishing, 2015, pp. 125–137. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25087-8_12

[11] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[12] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, no. 2, pp. 245–248, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10994-009-5103-0

[13] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, *The Planar k-Means Problem is NP-Hard*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 274–285. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00202-1_24

[14] M. Inaba, N. Katoh, and H. Imai, "Applications of weighted voronoi diagrams and randomization to variance-based k-clustering (extended abstract)," in *IN SCG 94: Proceedings of the Tenth Annual Symposium on Computational Geometry*. ACM Press, 1994, pp. 332–339.

[15] D. Coppersmith and S. Winograd, "Computational algebraic complexity editorial matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251 – 280, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0747717108800132

[16] A. J. Stothers, "On the complexity of matrix multiplication," 2010.

[17] F. L. Gall, "Powers of tensors and fast matrix multiplication," 2014.

[18] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356

[19] C. Studholme, D. L. G. Hill, and D. J. Hawkes, "An overlap invariant entropy measure of 3d medical image alignment," *Pattern Recognition (PR)*, vol. 32, no. 1, pp. 71–86, 1999. [Online]. Available: http://dx.doi.org/10.1016/S0031-3203(98)00091-0

[20] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988, attention: PDF is rather large (39MB). [Online]. Available: http://portal.acm.org/citation.cfm?id=46712