k-averages, a linear optimization alternative to kernel k-means

Mathias Rossignol Mathieu Lagrange

October 20, 2014

Abstract

We present an iterative flat clustering algorithm designed to operate on arbitrary similarity matrices, provided, with the only constraint that these matrices be symmetrical. Although functionally very close to kernel k-means, our proposal performs an maximization of average intra-class similarity, instead of a squared distance minimization, in order to remain closer to the semantics of similarities. We show that this approach allows relaxing the conditions on usable matrices, as well as opening better optimization possibilities. Systematic evaluation on a variety of data sets shows that the proposed approach equals or outperforms kernel k-means in most cases.

1 Introduction

parler de kmedoids et d'implem rapide pour leur similarite de critere kkmeans.

ML : Proposition de plan:

1. kkmeans (critere et algo)
2. kaverages (critere et algo)
3. differences qualitatives (critere)
4. difference quantitatives (algo)

2 Related Works

Kernel k-means (cite Girolami) has been for the past few years an algorithm of choice for flat data clustering with known number of clusters (cite salient uses of kkmeans). It makes use of a mathematical technique known as the "kernel trick" to extend the classical k-means clustering algorithm (cite original kmeans paper) to criteria beyond simple euclidean distance proximity. Since it constitutes the closest point of comparison with our own work, we will present it here in detail, before moving on to the description of our algorithm, before dedicating a section to the comparison of the two approaches.

In the case of kernel k-means, the kernel trick consists in considering that the k-means algorithm is operating in an unspecified, possibly very high-dimensional

Euclidian space; but instead of specifying the properties of that space, and the coordinates of objects in it, the equations governing the algorithm are modified so that everything can be computed knowing only the scalar products between points. The symmetrical matrix containing those scalar products is known as a kernel, noted $\mathcal{K}$.

2.1 Kernel k-means objective function

In this section and the following, we shall adopt the following convention: N is the number of objects to cluster and C the number of clusters; N_c is the number of objects in cluster c, and μ_c is the centroid of that cluster. z_{cn} is the membership function, whose value is 1 is object o_n is in class c, 0 otherwise.

Starting from the objective function minimized by the k-means algorithm, expressing the sum of squared distances of points to the centroids of their respective clusters:

$$S = \sum_{c=1}^{C} \sum_{n=1}^{N} z_{cn} \left(o_n - \mu_c\right) \left(o_n - \mu_c\right)^\top$$

And using the definition of centroids as:

$$\mu_c = \frac{1}{N_c} \sum_{n=1}^{N} z_{cn} o_n$$

S can be developed and rewritten in a way that does not explicitly refer to the centroid positions, since those cannot be computed:

$$S = \sum_{c=1}^{C} \sum_{n=1}^{N} z_{cn} Y_{cn}$$

where

$$\begin{aligned}
Y_{cn} &= \left(o_n - \mu_c\right) \left(o_n - \mu_c\right)^\top && (1)\\
&= o_n.o_n - 2o_n.\mu_c + \mu_c.\mu_c && (2)\\
&= o_n.o_n - 2o_n.\frac{1}{N_c} \sum_{i=1}^{N} z_{ci} o_i + \left(\frac{1}{N_c} \sum_{i=1}^{N} z_{ci} o_i\right) . \left(\frac{1}{N_c} \sum_{i=1}^{N} z_{ci} o_i\right) && (3)\\
&= o_n.o_n - \frac{2}{N_c} \sum_{i=1}^{N} z_{ci} o_n.o_i + \frac{1}{N_c^2} \sum_{i=1}^{N} \sum_{j=1}^{N} z_{ki} z_{kj} o_i.o_j && (4)\\
&= \mathcal{K}_{nn} - \frac{2}{N_c} \sum_{i=1}^{N} z_{ci} \mathcal{K}_{ni} + \frac{1}{N_c^2} \sum_{i=1}^{N} \sum_{j=1}^{N} z_{ki} z_{kj} \mathcal{K}_{ij} && (5)
\end{aligned}$$

Since the sum of K_{nn} over all points remains constant, and the sum of squared centroid norms (third, quadratic, term of Equation 5) is mostly bounded by the general geometry of the cloud of objects, we can see that minimizing this value implies maximizing the sum of the central terms, which are the average scalar products of points with other points belonging to the same class. Given a similarity matrix possessing the necessary properties to be considered as a kernel matrix (positive semidefinite), the kernel k-means algorithm can therefore

be used to create clusters that somehow maximize the average intra-cluster similarity.

2.2 Algorithm

Finding the configuration that asbolutely minimizes S (eq 2.1) is an NP-hard problem; however several approaches allow finding a good approximate result. We shall only focus here on the fastest and most popular, an iterative assignment / update procedure commonly referred to as the "k-means algorithm", or as a discrete version of Lloyd's algorithm, detailed in Algorithm 1.

Data: number of objects N, number of classes C, kernel matrix $\mathcal{K}$
Result: label vector L defining a partition of the objects into C classes

1 **Initialization:** fill L with random values in $[1..C]$;
2 **while** L *is modified* **do**
3 **for** $n \leftarrow 1$ *to* N **do**
4 **for** $c \leftarrow 1$ *to* C **do**
5 Compute Y_{cn} following eq 5 (note: $z_{cn} = (L_n == c)\,?\,1\ :\ 0$)
6 **end**
7 $L_n = \text{argmin}_c(Y_{cn})$;
8 **end**
9 **end**

Algorithm 1: K-means algorithm applied to kernel k-means.

As can be seen, the operation on line 5 is the most costly part of the algorithm: for each object n and class c, at each iteration, it is necessary to compute Y_{cn} from Equation 5—an $O(N^2)$ operation in itself, per object. The impossibility of simply computing the distances to a know centroid as is done in simple k-means, gives kernel k-means a much higher complexity, globally $O(N^3)$ per iteration, independently of how many objects are moved for that iteration. That operation also means that the whole kernel matrix must be loaded and accessed at each iteration, which can be a serious performance bottleneck in the case of large object collections.

3 Foundations of the k-averages algorithm

Our proposal with the k-averages algorithm is to adopt an alternative objective function that does not rely on a geometric interpretation, like kernel k-means, but on a more simple, direct understanding of a similarity matrix. The goal is to maximize the average intra-cluster similarity between points, a commonly used metric to evaluate clustering quality, and one whose computation is very simple—linear in time.

Due to its simplicity, however, that objective function cannot simply be "plugged into" the standard kernel k-means algorithm: it lacks the geometric requisites to ensure convergence. We must therefore propose an adapted algorithmic framework to exploit it: first, we show here that it is possible to easily compute the impact on the global objective function of moving a single point

from one class to another; we then introduce an algorithm intended to take advantage of that formula.

3.1 Conventions and possible objective functions

In addition to the notations already presented above, we index here the set of elements belonging to a given cluster c_k as $c_k = \{o_{k1}, \ldots, o_{kN_k}\}$. To simplify below, when we're simply considering one class, no matter which, we shall omit the first index and write $c = \{o_1, \ldots, o_{N_c}\}$.

The similarity between objects shall be written $s(o_i, o_j)$. Let us extend the notation s to the *similarity of an object to a class*, which we define as the average similarity of that object with all objects of the class. $s(o, c)$ accepts two definitions, depending on whether or not o is in c:

If $o \notin c$,

$$s(o, c) = \frac{1}{N_c} \sum_{i=1}^{n_c} s(o, o_i) \tag{6}$$

If $o \in c$, then necessarily $\exists i \mid o = o_i$

$$s(o, c) = s(o_i, c) = \frac{1}{N_c - 1} \sum_{j=1 \ldots n_c, j \neq i} s(o_i, o_j) \tag{7}$$

Let's call "quality" of a class the average intra-class object-to-object similarity, and write it $\mathcal{Q}$:

$$\mathcal{Q}(c) = \frac{1}{N_c} \sum_{i=1}^{n_c} s(o_i, c) \tag{8}$$

We do not, in our work, explicitly refer to class centroids, preferring to directly consider averages of similarity values between individuals within clusters. Indeed, we never refer to a geometrical interpretation of the data. However, it should be noted that since in k-means (and kernel k-means) the centroid of a class is defined as an average of all points in that class, $\mathcal{Q}$ is strictly equivalent to the average point to centroid similarity.

Using the notations above, we define our objective function as the average class quality, either normalized by class:

$$O_1 = \frac{1}{C} \sum_{i=1}^{C} \mathcal{Q}(c_i)$$

Or normalized by object:

$$O_2 = \frac{1}{N} \sum_{i=1}^{C} N_i \mathcal{Q}(c_i)$$

We will now compute, for both of those values, the impact on the global objective function of moving one object from one class to another. We will then show how, by using those formulas as a guide for the optimal reallocation of objects, and only moving objects that have a strictly positive impact on the function, we can guarantee the convergence of an iterative algorithm attempting to maximize those objective functions.

3.2 Impact of object reallocation on class quality

Considering a class c, let us develop the expression of $\mathcal{Q}(c)$ into a more useful form. Since all objects are in c, we use the formula in (7) to get:

$$\begin{aligned}\mathcal{Q}(c) &= \frac{1}{N_c}\sum_{i=1}^{N_c}\frac{1}{N_c-1}\sum_{\substack{j=1\ldots N_c\\ j\neq i}} s(o_i,o_j)\\ &= \frac{1}{N_c(N_c-1)}\sum_{i=1}^{N_c}\sum_{\substack{j=1\ldots N_c\\ j\neq i}} s(o_i,o_j)\end{aligned} \tag{9}$$

Using the assumption that the similarity matrix is symmetrical, we can reach (this is an indispensable transformation for future calculations):

$$\mathcal{Q}(c) = \frac{2}{N_c(N_c-1)}\sum_{i=2}^{N_c}\sum_{j=1}^{i-1} s(o_i,o_j) \tag{10}$$

For future use, let us define the notation:

$$\Sigma(c) = \sum_{i=2}^{N_c}\sum_{j=1}^{i-1} s(o_i,o_j) \tag{11}$$

Thus:

$$\mathcal{Q}(c) = \frac{2}{N_c(N_c-1)}\Sigma(c) \qquad \text{and} \qquad \Sigma(c) = \frac{N_c(N_c-1)\mathcal{Q}(c)}{2} \tag{12}$$

3.2.1 Removing an object from a class

Assuming that $o \in c$, necessarily $\exists i \mid o = o_i$. Since the numbering of objects is arbitrary, we can assume that $o = o_{N_c}$ then generalize from the result thus obtained.

$$\begin{aligned}\mathcal{Q}(c \smallsetminus o_{N_c}) &= \frac{2}{(N_c-1)(N_c-2)}\sum_{i=2}^{N_c-1}\sum_{j=1}^{i-1} s(o_i,o_j)\\ &= \frac{2}{(N_c-1)(N_c-2)}\left[\Sigma(c) - \sum_{j=1}^{N_c-1} s(o_{N_c},o_j)\right]\\ &= \frac{2}{(N_c-1)(N_c-2)}\left[\Sigma(c) - (N_c-1)s(o_{N_c},c)\right]\\ &= \frac{2N_c(N_c-1)\mathcal{Q}(c)}{2(N_c-1)(N_c-2)} - \frac{2(N_c-1)s(o_{N_c},c)}{(N_c-1)(N_c-2)}\\ &= \frac{N_c\mathcal{Q}(c) - 2s(o_{N_c},c)}{N_c-2}\end{aligned} \tag{13}$$

The quality of a class after removal of an object is thus:

$$\mathcal{Q}(c \smallsetminus o) = \frac{N_c\mathcal{Q}(c) - 2s(o,c)}{N_c-2} \tag{14}$$

And the change in quality from its previous value:

$$\begin{aligned}\mathcal{Q}\left(c \smallsetminus o\right) - \mathcal{Q}\left(c\right) &= \frac{N_c\mathcal{Q}(c) - (N_c - 2)\mathcal{Q}(c) - 2s\left(o, c\right)}{N_c - 2} \\ &= \frac{2\left(\mathcal{Q}(c) - s\left(o, c\right)\right)}{N_c - 2}\end{aligned} \tag{15}$$

3.2.2 Adding an object to a class

Assuming that $o \notin c$, we can similarly to what has been done previously (numbering is arbitrary) consider for the sake of simplicity that o becomes o_{N_c+1} in the modified class c. Following a path similar to above, we get:

$$\begin{aligned}\mathcal{Q}(c \cup o_{N_c+1}) &= \frac{2}{N_c(N_c+1)} \sum_{i=2}^{N_c+1} \sum_{j=1}^{i-1} s\left(o_i, o_j\right) \\ &= \frac{2}{N_c(N_c+1)} \left[\Sigma(c) + N_c s\left(o_{N_c+1}, c\right)\right] \\ &= \frac{(N_c - 1)\mathcal{Q}(c) + 2s\left(o_{N_c+1}, c\right)}{N_c + 1}\end{aligned} \tag{16}$$

The quality of a class c after adding an object o is thus:

$$\mathcal{Q}\left(c \cup o\right) = \frac{(N_c - 1)\mathcal{Q}(c) + 2s\left(o, c\right)}{N_c + 1} \tag{17}$$

And the change in quality from its previous value:

$$\begin{aligned}\mathcal{Q}\left(c \cup o\right) - \mathcal{Q}\left(c\right) &= \frac{(N_c - 1)\mathcal{Q}(c) - (N_c + 1)\mathcal{Q}(c) + 2s\left(o, c\right)}{N_c + 1} \\ &= \frac{2\left(s\left(o, c\right) - \mathcal{Q}(c)\right)}{N_c + 1}\end{aligned} \tag{18}$$

3.3 Impact of object reallocation on the global objective function

The influence these changes in class quality have on the global objective function depends upon what normalization we choose to adopt.

3.3.1 Class-normalized objective function

In that case, the calculation is direct: from (15) and (18), we can see that the impact on the objective function of moving an object c from class c_s ("source"), to whom it belongs, to a distinct class c_t ("target") is:

$$\delta_o(c_s, c_t) = \frac{2\left(s\left(o, c_t\right) - \mathcal{Q}(c_t)\right)}{N_t + 1} + \frac{2\left(\mathcal{Q}(c_s) - s\left(o, c_s\right)\right)}{N_s - 2} \tag{19}$$

3.3.2 Object-normalized objective function

This complicates the calculation a bit, but not much: when moving an object o from class c_s ("source"), to whom it belongs, to a distinct class c_t ("target"), $(N_s - 1)$ objects are affected by the variation in (15), and N_t) are affected by that in (18), in addition to the variation in similarity of o to the class it belongs to:

$$\delta_o(c_s, c_t) = \frac{2N_t \left(s\left(o, c_t\right) - \mathcal{Q}(c_t)\right)}{N_t + 1} + \frac{2(N_s - 1)\left(\mathcal{Q}(c_s) - s\left(o, c_s\right)\right)}{N_s - 2} + s(o, c_t) - s(o, c_s) \tag{20}$$

In both cases, computing that impact is a fixed-cost operation; we can therefore use those formulas as the basis for an efficient iterative algorithm.

4 K-averages algorithm

Our approach does not allow us to benefit, like kernel k-means, from the convergence guarantee brought by the geometric foundation of k-means. In consequence, we cannot apply a "batch" approach where at each iteration all elements are moved to their new class, and all distances (or similarities) are computed at once. Therefore, for each considered object, after finding its ideal new class, we must update the class properties for the two modified classes (source and destination), as well as recompute the average class-object similarities for them.

Although this seems at first like systematically updating everything at each object re-allocation should have a huge performance impact, our reliance on simple averages without any quadratic terms makes it possible to have very simple update formulas: new class qualities are given by Equations 14 and 17, and new object-class similarities can be computed by:

$$\begin{aligned} s(i, c_s(t+1)) &= \frac{N_s(t).s(i, c_s(t)) + s(i, n)}{N_s(t) + 1} \\ s(i, c_t(t+1)) &= \frac{N_t(t).s(i, c_s(t)) - s(i, n)}{N_t(t) - 1} \end{aligned} \tag{21}$$

where i is any object index, n is the recently reallocated object, c_s the "source" class that object i was removed from, and c_t the "target" class that object n was added to.

The full description of k-averages is given in Algorithm 2.

For this algorithm, the complexity of each iteration is

- $O(NC)$ corresponding to the best class search at line 7
- $O(NM)$ corresponding to the object-to-class similarity update at line 13, where M is the number of objects moved at a given iteration.

Even in the worst case scenario where $M = N$, the complexity for one iteration of the algorithm remains $O(n^2)$, but in practice, as can be seen on Figure 4, the number of objects moving from one class to another decreases sharply after the first iteration, meaning the the complexity of one iteration becomes quickly much lower than $O(n^2)$. That lowered computational costs is also accompanied by a diminution in memory access: as can be seen from Equation 21, in order

Data: number of objects N, number of classes C, similarity matrix $\mathcal{S}$
Result: label vector L defining a partition of the objects into C classes

Initialization: Fill L with random values in $[1..C]$;
Compute initial class qualities $\mathcal{Q}$ following eq 10;
Compute initial object-class similarities S following eq 7 or eq 6;
while L *is modified* **do**
  for $i \leftarrow 1$ *to* N **do**
    previousClass $\leftarrow L_i$;
    nextClass $\leftarrow \text{argmin}_k \, \delta_i(\text{previousClass}, k)$ (following the definition of δ in eq 19 or 20);
    if *nextClass* $\neq$ *previousClass* **then**
      $L_i \leftarrow$ nextClass;
      Update $\mathcal{Q}_{\text{previousClass}}$ following eq 14;
      Update $\mathcal{Q}_{\text{nextClass}}$ following eq 17;
      for $j \leftarrow 1$ *to* N **do**
        Update $S(j, nextClass)$ and $S(j, previousClass)$ following eq 21;
      end
    end
  end
end

Algorithm 2: K-averages algorithm.

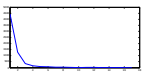

Figure 1: Number of moved objects per iteration.

to compute the new object-to-class similarities after moving an object n, only line n of the similarity matrix needs to be read. For the rest of the algorithm, only the (much smaller) object-to-class similarity matrix is used.

5 Comparison of approaches

(copy-pasted from the end of the former kkmeans section)

However, it should be noted that this is an indirect result of a squared distance minimization objective, and that the impact of the third term of Eq 5 on the the minimized value is hard to quantify. This raises question as to the actual semantics of algorithms based on that objective function, and concerning the exact connection between the initial intent (similarity-based clustering) an the obtained result.

Moreover, if a Euclidean geometric interpretation of the studied objects is

not trivial—which is likely to be the case, otherwise the use of a simple k-means could be considered—then similarity values gathered into a matrix may not necessarily make a positive semi definite matrix, and thus not a proper kernel. To solve that problem, (Dhillon-Kulis) suggest, following (Roth-Laub), to offset all diagonal elements of the matrix by a constant value. Since directly computing the necessary offset to make the matrix positive definite is not feasible for such large matrices, the proposed solution consists in iteratively increasing the diagonal offset until the matrix tests positive. It appears clearly in Equation refeq:yki that this only adds a constant factor to the minimized objective function, and thus doesn't affect the optimal solution; however, as acknowledged by (Dhillon-Kulis), by affecting the spread of points and the weight distribution in the matrix, it does affect the operation of algorithms looking for that optimum in unforeseeable ways. ML : too strong, for a well known problem, this impact can be studied in terms of convergence and quality of the solution and may be positive, so I guess that this unpredictability is only for new type of datasets. MR : toy example : point d'exclamation avec origin au bout du trait

Such is the price to pay to benefit from the solid geometric foundations and convergence guarantee of the k-means algorithm: a dissociation from immediate semantics, possibly made worse by the necessity of a clunky matrix conditioning procedure.

Our purpose with the work presented in this paper is to propose an alternative algorithm that operates in a way very similar to kernel k-means, but is explicitly designed to process similarities, remains as close as possible to their semantics, and directly attempts to maximize a meaningful quantity: the average intra-class similarity.

6 Experiments

6.1 Datasets

6.2 Evaluation Protocol

6.3 Results

6.4 From Progressive to Batch Updates

6.5 From Raw to Object Normalized Criterion

7 Discussion

8 Conclusion

			kMeans	kkMeans	kAverages object p	kAverages raw p	kAverages object b	kAverages raw b
id	nbClasses	nbSamplesPerClass						
7	2	28±1	48±12	**48±32**	**54±26**	**66±32**	**48±26**	**51±37**
12	2	100±47	12±2	14±3	**15±0**	10±5	**15±0**	10±5
13	2	442±0	0±0	2±2	**4±6**	**5±2**	**9±12**	**5±2**
17	2	100±0	0	0	0	**1**	0	**1**
20	2	548±1	**0±0**	**1±0**	**0±0**	**1±0**	**1±5**	**1±0**
21	2	60±18	**3±0**	1±0	1±0	**3±4**	1±0	**4±4**
25	2	636±69	30±0	**52±0**	41±0	36±1	41±0	36±1
28	2	310±54	55±21	**78±0**	72±1	21±0	72±1	21±0
29	2	490±161	**24**	21	22	15	22	15
34	2	581±0	0±0	7±0	7±0	**8±0**	7±0	**8±0**
42	2	3582±3988	**0**	0	0	0	0	0
43	2	1650±170	0±0	**0±0**	**0±0**	0±0	**0±0**	0±0

Table 1: nbIterations: 200, distance: dtw, normalizeData: 1, nbRuns: 20

			kMeans	kkMeans	kAverages object p	kAverages raw p	kAverages object b	kAverages raw b
id	nbClasses	nbSamplesPerClass						
3	5	12±0	**30±4**	29±3	**32±2**	**31±4**	29±8	20±4
4	3	310±0	36±1	**51±4**	**51±3**	42±12	**52±3**	44±12
5	3	1436±755	0±0	0±0	0±0	**0±0**	0±0	**1±1**
6	4	355±0	23±3	**44±8**	41±5	**46±0**	40±7	**45±4**
11	4	80±31	**83±3**	**81±10**	65±10	57±16	21±20	32±27
15	4	28±5	45±4	67±9	**72±10**	63±3	**73±7**	65±7
18	5	93±9	9±0	**9±1**	**10±1**	8±1	**9±2**	8±1
19	7	93±22	**5±1**	**5±1**	**5±0**	**5±1**	**5±0**	5±1
22	7	20±8	44±2	51±4	50±2	**54±1**	44±15	37±15
26	6	74±20	22±3	23±3	**25±2**	21±1	**24±2**	21±2
27	4	15±8	66±4	59±9	61±7	**72±3**	53±19	**70±11**
30	3	3079±2045	**60±0**	**61±4**	**60±0**	**61±0**	**60±0**	**61±0**
32	6	170±9	76±6	**79±4**	**80±1**	77±5	**77±18**	68±17
33	4	50±0	53±2	**58±7**	53±3	52±2	**56±6**	43±15
35	4	1250±43	2±0	**15±13**	11±8	**12±13**	**16±7**	**13±12**
37	7	50±0	31±2	**42±2**	**42±2**	40±2	**38±10**	17±8
38	6	100±0	79±3	84±4	**87±5**	**87±1**	84±5	65±26

Table 2: nbIterations: 200, distance: dtw, normalizeData: 1, nbRuns: 20

			kMeans	kkMeans	kAverages object p	kAverages raw p	kAverages object b	kAverages raw b
id	nbClasses	nbSamplesPerClass						
1	50	18±21	64±1	70±1	71±1	**72±1**	26±29	12±0
2	37	21±2	58±1	**62±1**	60±1	58±1	26±25	10±1
8	12	65±0	26±1	29±2	**31±1**	27±2	**29±7**	19±9
9	12	65±0	30±1	**35±2**	**35±1**	33±1	**35±1**	24±9
10	12	65±0	25±1	30±1	**31±1**	28±2	**30±7**	20±9
14	14	161±73	37±2	**77±3**	74±1	67±3	74±2	57±17
16	14	161±73	37±2	**77±3**	**77±2**	70±3	**72±17**	52±21
23	8	300±0	87±5	**88±5**	87±3	**90±4**	75±33	45±26
24	10	114±171	25±1	**32±1**	30±2	31±2	28±7	25±8
31	15	75±0	54±2	**66±3**	**67±2**	56±2	40±33	10±12
36	25	36±41	42±1	**51±1**	**51±1**	51±1	30±25	16±16
39	8	560±0	44±1	**46±1**	**46±0**	**46±1**	**46±1**	**45±5**
40	8	560±0	44±0	**45±0**	44±0	**45±1**	44±0	**44±5**
41	8	560±0	0±0	43±1	**44±1**	42±1	**44±0**	42±0

Table 3: nbIterations: 200, distance: dtw, normalizeData: 1, nbRuns: 20

dataSet	id
50words	1
Adiac	2
Beef	3
CBF	4
ChlorineConcentration	5
CinC_ECG_torso	6
Coffee	7
Cricket_X	8
Cricket_Y	9
Cricket_Z	10
DiatomSizeReduction	11
ECG200	12
ECGFiveDays	13
FaceAll	14
FaceFour	15
FacesUCR	16
Gun_Point	17
Haptics	18
InlineSkate	19
ItalyPowerDemand	20
Lighting2	21
Lighting7	22
MALLAT	23
MedicalImages	24
MoteStrain	25
OSULeaf	26
OliveOil	27
SonyAIBORobotSurface	28
SonyAIBORobotSurfaceII	29
StarLightCurves	30
SwedishLeaf	31
Symbols	32
Trace	33
TwoLeadECG	34
Two_Patterns	35
WordsSynonyms	36
fish	37
synthetic_control	38
uWaveGestureLibrary_X	39
uWaveGestureLibrary_Y	40
uWaveGestureLibrary_Z	41
wafer	42
yoga	43

Table 4: normalizeData: 1