Probabilistic Clustering by Random Swap Algorithm

Pasi Fränti, Olli Virmajoki and Ville Hautamäki
Speech and Image processing Unit
Department of Computer Science, University of Joensuu, Finland
{franti, ovirma, villeh}@cs.joensuu.fi

Abstract

We formulate probabilistic clustering method based on a sequence of random swaps of cluster centroids. We show that the algorithm has linear dependency on the number of data vectors, quadratic on the number of clusters, and inverse dependency on the dimensionality. Each halving of the probability of failure (e.g. from 1% to 0.5%) is achieved at the cost of only linear increase in the processing time.

1. Introduction

The main challenge in clustering is to find the correct global allocation of the clusters. The exact location of the partition boundaries is less relevant since the exact partition boundaries can be locally fine-tuned by *k-means* algorithm.

According to the above definition, the solution shown in Fig. 1 (current solution) is not the correct clustering. Most of the centroids at the top are correctly allocated, except that there are two centroids in one cluster. In the middle, one centroid is also missing. K-means is not able to fix the problem as the two regions are spatially separated from each other, and gradual changes cannot therefore happen. This problem is typical for data that contains well-separated clusters.

The clustering can be found by a sequence of *centroid swaps* and by fine-tuning their exact location by k-means. In Fig. 1, only one swap is enough to fix the problem. Important observation is that it is not necessary to remove one of the redundant centroids, and to relocate the centroid within the missing cluster. Instead, it is sufficient that the swap is made in the neighborhood of the problematic regions.

Swap-based or closely related methods have been considered in [1] but the main drawback is the computational complexity. Much simpler but effective approach is to select the swap randomly in a trial-and-error manner. This approach was first formulated as *tabu search* [2], and then simplified to *randomized local search* [3]. The main observation was that the same quality is reached independent on the initialization. The same conclusion was later confirmed by other authors [4].

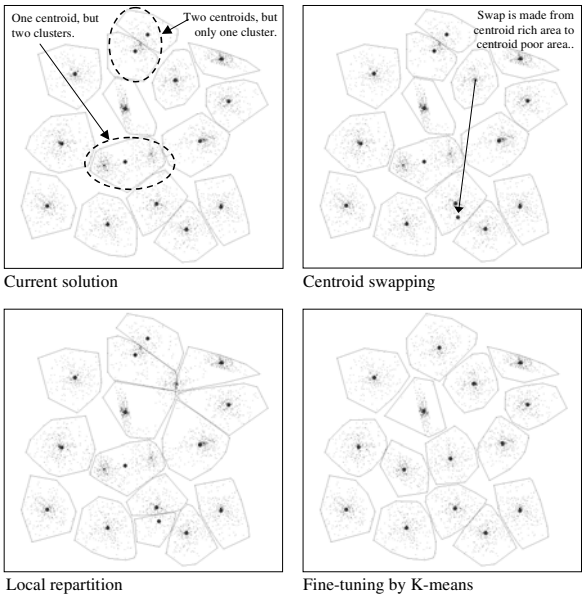

Figure 1. Demonstration of the centroid swap

We formulate this *random swap* (RS) technique as a probabilistic clustering algorithm. We show by theoretical analysis and experimental results that the algorithm outperforms k-means clustering both in time and quality, and it reaches competitive quality with agglomerative clustering but much faster. The algorithm is also extremely simple to implement, and therefore, useful for practitioners.

978-1-4244-2175-6/08/$25.00 ©2008 IEEE

```
RandomSwap(X) → C, P

    C ← SelectRandomRepresentatives(X);
    P ← OptimalPartition(X, C);
    REPEAT T times
        (C^new,j) ← RandomSwap(X, C);
        P^new ← LocalRepartition(X, C^new, P, j);
        C^new, P^new ← Kmeans(X, C^new, P^new);
        IF f(C^new, P^new) < f(C, P) THEN
            (C, P) ← C^new, P^new;
    RETURN (C, P);
```

Figure 2. Structure of the *Random Swap* (RS) method. For details see [3], and pseudocode: http://cs.joensuu.fi/pages/franti/research/rls.txt

2. Number of iterations

The probability to fix one incorrectly allocated cluster by centroid swap, we need to find favorable centroid for removal, and find favorable location for the new centroid. For adding new cluster, there are N distinct locations in total (one per each data vector) but only M of them is uniquely distinct (one per each cluster). However, the exact location within the cluster is not important since k-means can relocate them within just a few iterations.

At first sight, the probability for a successful swap seems to be at order of $(1/M)^2$. However, the capability of k-means to perform local fine-tuning is not limited within cluster. It can also move centroids between neighbor clusters by gradual movements if their distance is smaller than the deviation within the clusters. It is therefore not necessary to find exactly the correct clusters for removal and insertion, but selecting a neighbor cluster is sufficient.

By following the assumption that any neighbor of the desired cluster is good enough for the removal and for addition, the probability can be approximated as:

$$p(\text{good swap}) = (\alpha/M)\cdot(\alpha/M) = (\alpha/M)^2$$

where α is the number of neighbors, on average. It can be estimated from *Voronoi partition* of the vector space according to the given set of centroids.

The probability becomes lower when the number of clusters (M) increases, and higher when the dimensionality (d) increases. Interesting observation is that the probability is independent on the number of data vectors (N). Dependency on dimensionality is less obvious but results from literature imply that α increases exponentially with the dimensionality [5].

We define the probability to find the clustering in T iterations as p, and the probability for failure as q=1-p. Assuming that only one swap is needed, the probability of failure equals to the probability of selecting T unfavorable swap in a row: $q = 1-(\alpha/M)^2$

We can estimate the number of iterations needed to find the correct clustering with probability q as:

$$\log q = T \cdot \log\left(1 - \frac{\alpha^2}{M^2}\right)$$

$$\Leftrightarrow T = \frac{\log q}{\log\left(1 - \frac{\alpha^2}{M^2}\right)} \qquad (1)$$

The random swap method can now be formulated as a probabilistic algorithm as follows. For a given confidence level (probability q), iterate the algorithm by the number of times given Eq. (1). It can be shown that the number of iterations has tight bounds as:

$$T = \Theta\left(-\ln q \cdot \frac{M^2}{\alpha^2}\right) \qquad (2)$$

Proof is omitted here and will be presented in the journal version (in preparation).

In Fig. 1, we can visually estimate that the clusters have about 4 neighbors, on average. This gives an estimate for the probability of a favorable swap as $(\alpha/M)^2 = (4/15)^2 \approx 7\%$. According to (1), 41 iterations would be enough to find the correct clustering with 95% probability, and 95 iterations with 99.9% probability. The dependency between p and T is further demonstrated in Fig. 3.

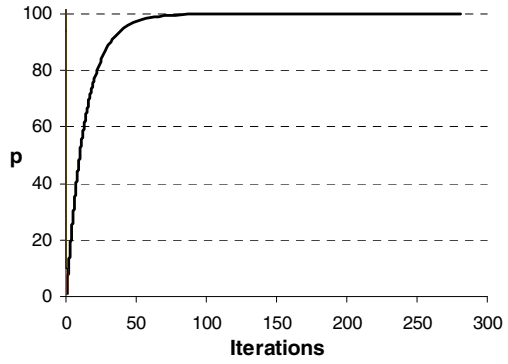

Figure 3. Probability of success (p) by iterations

The above analyses were made for the case when at most one cluster was incorrectly located. In the case of S_1-S_4 data sets (see Section 4), this is the case for 60% of any random initialization. In 38% times, there are two incorrectly located clusters, and only rarely (<2%) three or more swaps would be needed.

3. Efficiency of the overall algorithm

The efficiency of the method (see Fig. 2) depends on how many iterations (swaps) are needed, and how much time each iteration takes. Time complexity of a single iteration depends on the implementation of the following steps:

1. Swap of centroid.
2. Remove old cluster.
3. Create new cluster.
4. Update neighbor centroids.
5. Perform two k-means iterations.

The time complexities of these steps have been summarized in Table 1. Steps 1 and 4 have only marginal effect on the total processing time. The steps 2 and 3 (removal and addition) are efficient and require only O(N) time, whereas the k-means iterations are the bottleneck. Although we perform only 2 iterations, and use the fast reduced search variant [6], the complexity of k-means sums up to $4\alpha N = O(\alpha N)$.

Fig. 4 shows the distribution of the processing time between the local repartition (steps 1-4) and k-means (step 5) for a data set with N=4096 vectors and M=256 clusters. Selected numbers have been collected in Table 1.

Table 1. Time complexities and the observed number of steps over 100 and 500 iterations.

Step:	Time	100	500
Centroid swap	3	3	3
Cluster removal	$2N$	8448	10137
Cluster addition	$2N$	8192	8192
Update centroid	$4N/M + 2\alpha + 1$	61	60
K-means iter.	$\leq 4\alpha N$	285555	197327
Total	$O(\alpha N)$	302259	215719

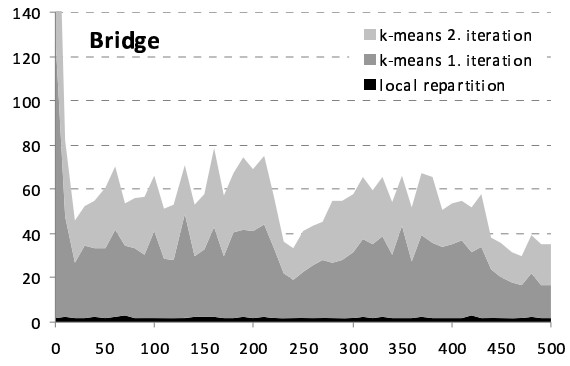

Figure 4. Number of calculations (per data vector) required by the algorithm during the iterations.

The total processing time is the time required per single iteration multiplied by the number of iterations. Based on the results of Sections 2, it is estimated as:

$$T(N,M) \leq -\ln q \cdot \frac{M^2}{\alpha^2} \cdot \alpha N = O\left(\frac{-\ln(q) \cdot NM^2}{\alpha}\right) \quad (3)$$

From (3), we can make the following observations about the time complexity of the algorithm.

- Logarithmic dependency on q.
- Linear dependency on N.
- Quadratic dependency on M.
- Inverse dependency on α.

The main advantages are that the time complexity increases only linearly with the size of data, and that the probability of failure (q) has only minor effect on the processing time. Furthermore, since α increases with the dimensionality, the algorithm has inverse dependency on dimensionality. Thus, the higher the dimensionality, the faster is the algorithm.

4. Experiments

We cluster the data sets from [7] including four generated data sets, and four image data sets (http://cs.joensuu.fi/sipu/datasets/).

Table 2. Average number of neighbors (α).

Set	d	α	Set	d	α
Bridge	16	15.3	S_1	2	3.9
House	3	7.2	S_2	2	3.3
Miss America	16	38.7	S_3	2	3.6
Europe	2	5.3	S_4	2	3.7

Table 2 reports the observed number of neighbors. Using these α-values, we calculated the estimated number of iterations required to obtain the correct clustering with three different failure probabilities (10%, 1% and 0.1%), see Table 1. The number of iterations is relatively small for the higher dimensional data sets *Bridge* and *Miss America*: 589 and 89. Significantly higher number of iterations is needed for the 2- and 3-dimensional sets (*Europe*, *House*).

Three other clustering methods were implemented: *k-means* (KM), *repeated k-means* (RKM) and *agglomerative clustering* (AC). K-means results are obtained by the fast exact variant [6] [8]. The repeated k-means is the best result after 10 repeats. Agglomerative clustering refers to Ward's method as implemented in [9].

The results are summarized in Table 3. The proposed RS algorithm reaches the result of k-means (KM and RKM) within a few iterations, and bypasses

AC by using less iterations than indicated by the q=1%. The only exception is the Europe data set, which has relatively large number of clusters and low dimensionality. Nevertheless, the result of AC is outperformed after 30226 iterations.

Time-distortion comparison of the proposed method and RKM are illustrated in Fig. 5. With the higher dimensional data sets, the method works nearly as fast as k-means, but it is able to provide significantly better result when iterated further. Table 4 summarizes comparative results for error probability of q=1%, which shows the competence against AC as well.

Table 3. Number of iterations estimated (left), and required (right) to bypass the other methods

Data set	Iterations required according to (1)			Iterations needed to bypass		
	q= 0.1	q= 0.01	q= 0.001	KM	RKM	AC
Bridge	644	1287	1931	11	22	749
House	2910	5820	8730	7	53	148
Miss America	100	200	299	13	52	2742
Europe	5371	10742	16113	1476	1477	30226
$BIRCH_1$	1136	2272	3408	38	54	1606
$BIRCH_2$	5221	10441	15661	8	23	902
$BIRCH_3$	1513	3026	4539	15	28	123
S_1	33	66	99	1	17	41
S_2	47	93	140	2	35	59
S_3	39	78	117	9	27	45
S_4	37	74	111	9	14	14

Table 4. Summary of the time and quality comparison of RS (q=0.01), RKM and AC

Data set	Processing time (seconds)			Clustering quality (mse)		
	RS	RKM	AC	RS	RKM	AC
Bridge	85	17	40	167.6	178.4	168.9
House	531	45	509	5.94	6.41	6.27
Miss America	21	34	116	5.77	5.87	5.36
Europe	46709	9142	10477	2.83	3.38	2.62
$BIRCH_1$	1824	428	3432	4.64	5.27	4.73
$BIRCH_2$	2009	108	3464	2.28	6.46	2.28
$BIRCH_3$	2324	409	3466	1.86	2.11	1.96
S_1	0.7	0.7	7	8.92	13.51	8.93
S_2	1.0	1.0	7	13.28	16.82	13.44
S_3	1.0	1.5	7	16.89	18.60	17.70
S_4	1.0	2.0	8	15.75	16.64	17.52

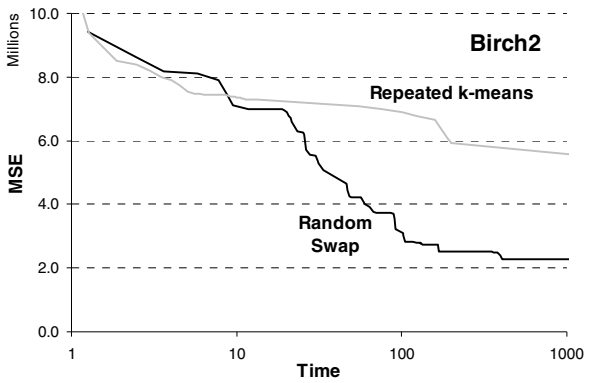

Figure 5. Time-distortion efficiency

5. Conclusion

We have formulated a simple swap-based method as a probabilistic clustering algorithm. It is very simple to implement, and it outperforms competitive methods in efficiency. We conclude that the method has potential to be widely used algorithm for practitioners.

References

1. A. Likas, N. Vlassis and J.J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition* 36:451-461, 2003.
2. P. Fränti, J. Kivijärvi and O. Nevalainen. Tabu search algorithm for codebook generation in VQ. *Pattern Recognition*, 31(8):1139-1148, August 1998.
3. P. Fränti and J. Kivijärvi. Randomised local search algorithm for the clustering problem. *Pattern Analysis and Applications*, 3(4):358-369, 2000.
4. T. Kanungo, D.M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A.Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(1):89-112, May 2004.
5. V. Pestov, "On the geometry of similarity search: dimensionality curse and concentration of measure", *Information Processing Letters*, vol. 73, p. 4751, 2000.
6. T. Kaukoranta, P. Fränti and O. Nevalainen. A fast exact GLA based on code vector activity detection. *IEEE Trans. Image Proc.*, 9(8):1337-1342, 2000.
7. P. Fränti and O. Virmajoki. Iterative shrinking method for clustering problems. *Pattern Recognition,* 39(5):761-765, May 2006.
8. K.-L. Chung and J.-S. Lin. Faster and more robust point symmetry-based k-means algorithm. *Pattern Recognition*, 40(2):410-422, February 2007.
9. P. Fränti, T. Kaukoranta, D.-F. Shen and K.-S. Chang. Fast and memory efficient implementation of the exact PNN. *IEEE Trans. Image Proc.*, 9(5):773-777, 2000.