Parallelized Kernel Patch Clustering

Stefan Faußer and Friedhelm Schwenker

Institute of Neural Information Processing, University of Ulm, 89069 Ulm, Germany
{stefan.fausser,friedhelm.Schwenker}@uni-ulm.de

Abstract. Kernel based clustering methods allow to unsupervised partition samples in feature space but have a quadratic computation time $O(n^2)$ where n are the number of samples. Therefore these methods are generally ineligible for large datasets. In this paper we propose a meta-algorithm that performs parallelized clusterings of subsets of the samples and merges them repeatedly. The algorithm is able to use many Kernel based clustering methods where we mainly emphasize on Kernel Fuzzy C-Means and Relational Neural Gas. We show that the computation time of this algorithm is basicly linear, i.e. $O(n)$. Further we statistically evaluate the performance of this meta-algorithm on a real-life dataset, namely the Enron Emails.

1 Introduction

Having a large dataset of possibly 100,000 samples or more imposes several problems on an unsupervised clustering algorithm. While prototype-based clustering algorithms acting in input space, especially k-means, have a linear computation time to cluster the samples they still need to pass multiple times over the samples - one pass per iteration to adapt the prototypes until they converge. This even gets more computationally expensive for Kernel based clustering methods. Kernel based clustering methods allow to unsupervised partition samples in feature space. Often those samples are nonlinear distributed in input space and are easier to partition in a certain feature space which makes the Kernel methods more useful. However Kernel based clustering methods have the main penalties that they have a quadratic computation time and cannot express the prototypes directly but with a convex combination of existing samples. Therefore multiple passes over the dataset gets even more costly for the Kernel based clustering methods.

To reduce the passes over the data, Fahim et al. [1] have observed that certain samples near to their cluster centre stay in their cluster for some iterations and therefore the distance calculation and assignment steps can be ommited for those samples. However these observations only apply to standard k-means, i.e. needs convex cluster shapes and hard sample to cluster assignments. A slightly older but still popular attempt to cluster large datasets is the CLARANS algorithm [2] which approximates a k-medoid method that minimizes the search in the dataset by heuristics and two parameters. Other efforts have been made to parallelize the k-median algorithm (Guha et al. 2000 - 2003) [3] by independently clustering

F. Schwenker and N. El Gayar (Eds.): ANNPR 2010, LNAI 5998, pp. 131–140, 2010.

data streams (or Patches) to gain lk weighted cluster (which are represented by samples), where l is the number of parallelized clusterings, and then cluster again the lk samples to finally gain k cluster. First attempts to parallel cluster samples in subsets with k-means and exchange their statistics have been done as well [4] and speeded up the standard k-means algorithm by $O(k/2)$. Later on, Guhas popular method [3] has been extended for other prototype-based clustering algorithms, namely k-means and batch neural gas [5]. Speed ups for the Kernel based methods to cluster large datasets have been done in Kernel K-Means [6] by block-wise calculating and processing the Kernel matrix which represents the similarities between the samples. In a recent scientific paper, Hasenfuss et al [7] have extended Guhas method furthermode with the Relational Neural Gas method (called Patch Relational Neural Gas) but restricted it for the time being to a sequential clustering procedure.

Our contribution in this paper is a generalization of the Patch Relational Neural Gas algorithm [7] to integrate multiple Kernel based clustering methods and call this algorithm simple the Kernel Patch Clustering (KPC) method. We additionally show the integration of Kernel Fuzzy C-Means and Kernel K-Means within the KPC method and the new assignment update formulas. This expands the algorithm generally for soft memberships. Furthermore we describe a meta-algorithm that performs parallelized clusterings of subsets of the samples using the Kernel Patch Clustering method and merges them repeatedly. It can be shown that the computational time is linear regarding the amount of samples. Lastly we statistically evaluate the performance of this meta-algorithm on a real-life dataset, namely the Enron Emails and show that Kernel Fuzzy C-Means with its soft memberships integrated in KPC perform better than Relational Neural Gas with its hard memberships.

2 Kernel Based Methods for Clustering

Typically most divisive clustering methods aim to minimize a common quantization error. Assume that we want to partition $x_1, x_2, \ldots, x_N$ samples in K disjoint sets or cluster and each cluster has a representing prototype c_k then the actual quantization error or intra-cluster variance (ICV) can be written as:

$$E = \sum_{k=1}^{K} \sum_{i=1}^{N} f_k(i) d(c_k, x_i) \tag{1}$$

where $f_k(i)$ is a hard or bounded soft assignment of sample i to cluster k and $d(c_k, x_i)$ is the distance between sample x_i and cluster prototype c_k. If $d(c_k, x_i)$ is measured by the euclidean distance and $f_k(i)$ is a hard assignment then E is the exact quantization error that k-means minimizes. This happens by repeatedly 1. updating $f_k(i)$: assign samples to clusters based on their distance to their nearest prototypes and 2. updating c_k: move prototypes to their cluster centres until the prototypes converges locally. For the euclidean distance the general function to calculate the current prototypes is as follows:

$$c_k = \frac{\sum_{i=1}^{N} f_k(i) x_i}{\sum_{i=1}^{N} f_k(i)}$$

Now suppose that we would transform all samples $x_i \in X$ and prototypes $c_k \in X$ to a (higher dimensional) feature space using the mapping function $\phi : X \to \mathbb{F}$ that maps X from input space to a possible high-dimensional feature space $\mathbb{F}$. This would allow us to calculate the prototypes in feature space. Unfortunately such mapping functions are costly and often unknown. Still we can calculate the distance between such (theoretically) transformed samples using a positive-definite and symmetric kernel $\kappa(x_i, x_j)$ and applying the kernel trick, i.e. define the prototypes as linear combinations of existing transformed samples. Now further assume that we want to weight each sample i with a weight $w(i)$. We can then set up the new weighted distance function $d_{weighted}(\phi(c_k), \phi(x_i))$ in feature space. The distance function $d_{weighted}(\phi(c_k), \phi(x_i))$ can be written as:

$$\begin{aligned} d_{weighted}(\phi(c_k), \phi(x_i)) &= ||\phi(x_i) - \frac{\sum_{j=1}^{N} f_k^{\phi}(j) w(j) \phi(x_j)}{\sum_{j=1}^{N} f_k^{\phi}(j) w(j)}||^2 \qquad (2) \\ &= \langle \phi(x_i), \phi(x_i) \rangle - 2 \left\langle \phi(x_i), \frac{\sum_{j=1}^{N} f_k^{\phi}(j) w(j) \phi(x_j)}{\sum_{j=1}^{N} f_k^{\phi}(j) w(j)} \right\rangle + \\ &\quad \left\langle \frac{\sum_{j=1}^{N} f_k^{\phi}(j) w(j) \phi(x_j)}{\sum_{j=1}^{N} f_k^{\phi}(j) w(j)}, \frac{\sum_{j=1}^{N} f_k^{\phi}(j) w(j) \phi(x_j)}{\sum_{j=1}^{N} f_k^{\phi}(j) w(j)} \right\rangle \\ &= \kappa(x_i, x_i) - \frac{2 \sum_{j=1}^{N} f_k^{\phi}(j) w(j) \kappa(x_i, x_j)}{\sum_{j=1}^{N} f_k^{\phi}(j) w(j)} + \\ &\quad \frac{\sum_{j=1}^{N} \sum_{l=1}^{N} f_k^{\phi}(j) f_k^{\phi}(l) w(j) w(l) \kappa(x_j, x_l)}{[\sum_{j=1}^{N} f_k^{\phi}(j) w(j)]^2} \end{aligned}$$

Note that there are many repeatings in the above formula that have to be calculated in the right order to avoid wasting computational time. Contrary to standard k-means where we iteratively update the prototypes we can here only repeatedly update the assignments by calculating and comparing the distances. For Weighted Kernel K-Means the assignment update step is:

$$f_k^{\phi}(i) = \begin{cases} 1, \text{ if} & d_{weighted}(\phi(c_k), \phi(x_i)) < d_{weighted}(\phi(c_l), \phi(x_i)), \\ & l = 1, \ldots, K, l \neq k \\ 0, \text{ else} \end{cases} \qquad (3)$$

For Weighted Kernel Batch Neural Gas or Weighted Relational Neural Gas [9] the assignment update step is:

$$f_k^{\phi}(i) = exp\left(\frac{-rank(\phi(c_k), \phi(x_i))}{\lambda}\right) \qquad (4)$$

where $rank(\phi(c_k), \phi(x_i)) = |\{\phi(c_l) \mid d_{weighted}(\phi(c_l), \phi(x_i)) < d_{weighted}(\phi(c_k), \phi(x_i)), l = 1, \ldots, K, l \neq k\}| \in \{0, \ldots, K-1\}$. Lastly the assignment update steps for Weighted Kernel Fuzzy C-Means [8] gets:

$$f_k^{\phi}(i) = \frac{1}{\sum_{l=1}^{K} [\frac{d_{weighted}(\phi(c_k),\phi(x_i))}{d_{weighted}(\phi(c_l),\phi(x_i))}]^{\frac{2}{m-1}}} \tag{5}$$

The output of such a Kernel based clustering method is then the assignments f that determines the hard (Relational Neural Gas, Kernel K-Means) or soft (Kernel Fuzzy C-Means) assignments of the given samples to K cluster. Comparing the three algorithms, Kernel Fuzzy C-Means and Relational Neural Gas are more insensitive to initializations as both algorithms update their indirectly defined prototypes not only by their greedy winner samples but also by other samples determined through neighborhood size λ (Relational Neural Gas) or fuzzifier m (Kernel Fuzzy C-Means). The Kernel Fuzzy C-Means algorithm however has the plus that it uses soft assignments, i.e. gives possibly more information on the data. On the other hand if fuzzifier $m \to 1$ then this algorithms behaves exactly like Kernel K-Means. All three Kernel methods have the same drawback that they have a quadratic computation time $O(N^2)$ which renders them unusable for vast datasets.

3 Kernel Patch Clustering

Assume that we have N samples that we want to separate in K sets but we only can cluster up to M samples with one of the Kernel based clustering methods described above due to the high required computational time. Now suppose that we select instead the first M samples out of N, i.e. set up a Patch, cluster them into K cluster and choose the best k samples per cluster as approximative cluster prototypes. This results in exactly $k \cdot K$ samples that represent K cluster. Now they can be weighted by the number of samples they represent and can be itself clustered with the next M samples. This can be iterated until every N samples have been processed in Patches with a maximum size of M and $k \cdot K$ samples representing the final K prototypes are left. Let us now formulate these thoughts in the following *Kernel Patch Clustering* algorithm:

- **Input:**
  - kernel function κ, samples $x \in X^N$
  - number of patches C, whereas $C = \frac{N}{M}$, M = maximum samples to cluster per patch
  - number of cluster K
  - number of samples per cluster k
  - choose one Kernel clustering method (Kernel K-Means, Kernel Fuzzy C-Means, Relational Neural Gas)
- **Initialize:**
  - I = arbitrary permutation of $\{1, \ldots, N\}$
  - $J_0 = \{\emptyset\}$
  - $w_0 = \{\emptyset\}$

- **for** $t = 1$ to C
  - Construct Patch $P_t \in \mathbb{R}^{|S_t| \times |S_t|}$ with sample indices $S_t = \{J_{(t-1)}, I_{(t-1)\frac{N}{C}}, \ldots, I_{t\frac{N}{C}}\}$ using kernel funktion κ and samples x. This Patch P_t has now the similarities between all current choosen samples $\{I_{(t-1)\frac{N}{C}}, \ldots, I_{t\frac{N}{C}}\}$ plus the last k-approximation of the prototypes $J_{(t-1)}$. Append the vector $\{1\}^{\frac{N}{C}}$ to the weights w_{t-1} so that the whole weight vector w_t has a length of $k \cdot K + \frac{N}{C}$ (or a length of $\frac{N}{C}$ for the first time)
  - Arbitrary initialize cluster assignments $f_{old} \in \mathbb{R}^{K \times |S_t|}$ and perform selected Kernel clustering method using weights w_{t-1} and Patch P_t with K cluster to get new cluster assignments $f_{new} \in \mathbb{R}^{K \times |S_t|}$
  - Select the k best samples for each cluster K out of S_t using the k-approximation (6), the assignments f_{new} and the distance function for the selected clustering method. Fill the sample indices in $J_t \in \mathbb{N}^{K \cdot k}$
  - Calculate the new weights $w_{tl}^{new} = \{\frac{m_l}{k}\} \in \mathbb{R}^k$, for $l = 1, \ldots, K$, where m_l is the number of samples belonging to cluster l. Form the weight vector $w_t = \{w_{t1}^{new}, \ldots, w_{tK}^{new}\}$
- **Output:**
  - k-approximation of final prototypes J_C
  - final cluster assignments f_{new}

Note that the number of Patches C has to be choosen such that it is still computationally possible to cluster $\frac{N}{C} + k \cdot K$ samples per Patch. For the k-approximation we simply determine the k samples that are nearest to their own cluster centre (prototype) and do that for each cluster l (as introduced in [7] for Relational Neural Gas):

$$argmin_{i|g_l^{\phi}(i)=1}(d(\phi(x_i), \phi(c_l))), l = 1, \ldots, K \tag{6}$$

The above equation has to be repeated k-times to get the k-best approximations of each cluster prototype l, each time removing the winner sample x_i so that it cannot be selected once more. As this requires hard cluster assignments $g_l^{\phi}(i)$, the soft assignments of Kernel Fuzzy C-Means have to be first converted by defining that a sample i belongs to the cluster l with the highest fuzzy membership $\text{argmax}_l(f_l^{\phi}(i))$. For the most settings the algorithm will be done after having calculated the distances and assignments of the rest of the $(N - k \cdot K)$ samples using the k-approximation of final prototypes J_C and the final cluster assignments f_{new}.

3.1 Parallelized Kernel Patch Clustering

While the Kernel Patch Clustering approach described above clusters the samples in a single pass over the dataset, i.e. if we consider the k-approximations of the prototypes as new samples, it is still a serial process. Easily this can be parallelized by distributing the clustering of the Patches $P_1, \ldots, P_C$ and the calculating of the k-approximations of the resulting prototypes to multiple systems connected by a network or multiple threads on one system possibly handled by multiple processors. This can be formulated in the meta-algorithm to perform parallelized Kernel Patch Clusterings:

- **Input:**
  - maximum samples to cluster per patch M
  - number of maximum parallelizations μ
- **while** not all samples are processed
  1. Construct up to μ patches $P_1, \ldots, P_\mu$, each patch P_i having the next M samples and the last weighted k-approximations of the prototypes J_{last}
  2. Distribute the μ patches to μ systems or threads and cluster the patches parallelized
  3. Calculate the k-approximations of the resulting prototypes J_i and their weights w_i parallelized
  4. Collect all μ k-approximations $J_1, \ldots, J_\mu$, remove any duplicated samples and save it in J_{last}. Divide their corresponding weights by $\frac{1}{\mu}$ as they will be distributed later on in μ parallel clusterings
  5. (optional): Cluster the μ k-approximations in $J_{last} \in \mathbb{N}^{\mu \cdot k \cdot K}$ to get one k-approximation $J_{last} \in \mathbb{N}^{\cdot k \cdot K}$
- **Output:**
  - μ k-approximations of final prototypes J_{last}

Step 2 and 3 can be done completely parallelized, i.e. cluster $M \cdot \mu$ samples parallelized. In step 4, the results of those μ clusterings are collected resulting in $\mu \cdot k \cdot K$ weighted samples, representing k-approximations of K prototypes. As those samples are reinserted in step 1, it is possible that the result of the clusterings in step 4 produces duplicated samples that have to be removed. In step 5 we optionally cluster the final $\mu \cdot k \cdot K$ weighted samples once more to gain exactly $k \cdot K$ k-approximations of K prototypes and to be conform with the output of the serial Kernel Patch Clustering algorithm. This step is recommended for high values of parallelizations μ as without the (re-) clustering of the prototypes this might result in too much samples to cluster in step 1.

Complexity: The complexity of one clustering operation is $O((M + k \cdot K)^2)$ where M is the maximum number of samples per patch, K the number of cluster and k the number of samples per cluster (being the k-approximation of K prototypes). As the size of $M + k \cdot K$ however is bounded and generally independent of N and we have to perform $\frac{N}{M}$ such clustering operations, the summed clustering complexity results in $O((M + k \cdot K)^2 \frac{N}{M}) \sim O(M \cdot N) \sim O(N)$ in terms of N. Moreover this clustering complexity can even be reduced linear through parallelizations by a factor of μ. However the tradeoff is that the complexity of one clustering operation rises to $O((M + \mu \cdot k \cdot K)^2)$ which is bearable for small values of μ, k and K. For higher values of μ, k and K like $\mu \cdot k \cdot K \geq M$, it is advisable to perform an additional clustering step (step 5 in the algorithm above) to get only one k-approximation of the cluster.

4 Experiments and Results

To evaluate the Kernel Patch Clustering method we have conducted experiments with one synthetic (five two-dimensional gaussian distributed cluster), one widely

known (Wisconsin Diagnostic Breast Cancer) and one real-life (Enron Emails) dataset. We have compared the performances of Relational Neural Gas (RNG), Kernel Fuzzy C-Means (KFCMEANS) and integrations of those two methods into the Kernel Patch Clustering (KPC). Furthermore we have compared these methods to the basic approach to arbitrarily choose some samples out of all samples and cluster them using KFCMEANS. For all experiments we have set the neighborhood range λ for RNG to be exponentially falling from $\frac{N}{2}$ to 0.01 which are stable standard values (see [9]). The fuzzifier for KFCMEANS had been set to 2.0 for the synthetic dataset and 1.25 for both other datasets to reach more hard than soft memberhips. We have done time measurements of these algorithms for the Enron Emails dataset.

4.1 Five Two-Dimensional Gaussian Distributed Cluster

This synthetic dataset had been created by five multivariate (two-dimensional) gaussian distributions with the parameters $\mu_1 = \{0, 0\}, \mu_2 = \{1, 0\}, \mu_3 = \{0, 1\}$, $\mu_4 = \{1, 1\}, \mu_5 = \{0.5, 0.5\}$ and a variance of $\sigma^2 = 0.01$. As it can be seen in figure 1, the amount of samples drawn from the distributions are unequal which highly complicates the clustering problem. We have assigned each cluster a class label and have 50 samples in class 1, 100 in class 2, 100 in class 3, 500 in class 4 and 500 in class 5. We have choosen a linear kernel to calculate the similarities between the samples:

$$\kappa(x_i, x_j) = (x_i \cdot x_j)$$

For all clustering methods the aim was to partition the samples in $K = 5$ cluster. We have performed 50 test runs with each algorithm and then have calculated the intra-cluster variance (ICV), i.e. quantization error with hard assignments (same conditions for all methods) and the class prediction score by comparing

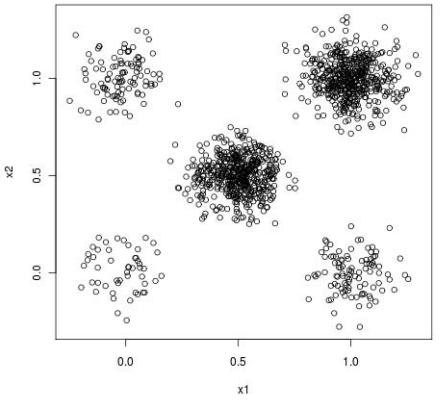

Fig. 1. Five two-dimensional gaussian distributed cluster, $\mu_1 = \{0, 0\}, \mu_2 = \{1, 0\}, \mu_3 = \{0, 1\}, \mu_4 = \{1, 1\}, \mu_5 = \{0.5, 0.5\}$, variance $\sigma = 0.01$, 1250 samples $(50, 100, 100, 500, 500)$

the cluster sets with the class labels. The results can be seen in table 1. For the last method we arbitrarily choose the same size of samples as in one Patch (100). We tried to experiment with the Patch size but have observed that any size between 50 to 500 had not made any statistical difference. Same applies for different number of parallelizations: $1 \leq \mu \leq 6$. However for Patch sizes below 50, the KPC - RNG algorithm had often problems finding $K = 5$ cluster while the KPC - KFCMEANS algorithm still converged. In general the KFCMEANS method performs way better than the RNG method for this dataset, probably because of the soft relaxations of the assignments of samples to cluster. Integrated into the KPC algorithm, both methods still perform nearly as well while being (theoretically) much faster.

Table 1. Cluster validations on five two-dimensional gaussian distributed cluster. All methods had to partition the sample into $K = 5$ cluster. For Kernel Patch Clustering (KPC), 100 samples were processed per Patch, k = 3-approximation, $\mu = 1, \ldots, 6$ parallelizations. The values were averaged over 50 runs.

	KFCMEANS	**RNG**	**KPC - KFCMEANS**	**KPC - RNG**	**arbitrary choosen KFCMEANS**
quantization error	23.43	43.29	23.89	43.77	31.85
class prediction	100%	88.82%	100%	88.8%	97.1%

4.2 Wisconsin Diagnostic Breast Cancer

The Wisconsin Diagnostic Breast Cancer database is a widely known dataset that has been contributed to the UCI machine learning repository [11] in 1995. It consists of 569 samples describing each characteristics of the cell nuclei taken from a Breast mass with 30 real valued features. The task is to classify the cell nuclei either as malignant or benign. We have standardized the samples to zero mean and unit variance and have calculated the similarities between them by a RBF-Kernel with parameter $\sigma = 5$:

$$\kappa(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$$

Same as for the synthetic dataset we have calculated the quantization error and the class prediction score. The results can be seen in table 2. All algorithms had about the same performance in terms of those two validation criteria.

4.3 Bag of Words – Enron Emails

The Bag of Words dataset at the UCI machine learning repository [11] donated in 2008 consists of five different text collections that each delivers plenty word to document assignments. For our experiments we have choosen the Enron Emails text collection which consists of $39,861$ documents, $28,102$ different words and approximately $1,900,000$ document to word assignments. To calculate

Table 2. Cluster validations on Wisconsin Diagnostic Breast Cancer database. All methods had to partition the sample into $K = 10$ cluster. For Kernel Patch Clustering (KPC), 50 samples were processed per Patch, $k = 3$-approximation. The values were averaged over 50 runs.

	KFCMEANS	**RNG**	**KPC - KFCMEANS**	**KPC - RNG**
quantization error	221	223	224	222
class prediction	90.5%	90.1%	89.8%	90%

the similarities between the documents we have choosen the Bhattacharyya kernel [10] with a multinomial distribution, setting parameter $X = \frac{1}{8}$:

$$\kappa(p, p') = [\sum_{i=1}^{M} (p_i p'_i)^{\frac{1}{2}}]^X$$

where $p_i = \frac{\text{number of words i in document a}}{\sum_j \text{number of words j in document a}}$
and $p'_i = \frac{\text{number of words i in document b}}{\sum_j \text{number of words j in document b}}$.

As the complexity of the co-occurence matrix necessary to calculate the kernel rises with the number of words, we could only handle about 200 samples per clustering and had parallelized the clusterings on a four core machine (parameter $\mu = 4$). Furthermore we have performed a serialized clustering on the same machine (parameter $\mu = 1$). As no class labels are known for these documents we can therefore solely evaluate the quantization error. The results with 5 and 10 cluster can be seen in table 3. This time the difference between the simple approach to choose arbitrarily 200 samples and cluster them and the appliance of the KPC algorithm with either KFCMEANS or RNG is clearly visible. Also it can be seen that KPC-KFCMEANS produces slightly better cluster results than KPC-RNG.

Table 3. Cluster validations on the Enron Emails. For Kernel Patch Clustering (KPC), 200 samples were processed per Patch, $k = 3$-approximation. The values were averaged over 20 runs.

	K	μ	**quant. error**	**time[s]**
KPC-RNC	5	1	17226	357
KPC-KFCMEANS	5	1	17141	369
KPC-RNC	5	4	17328	126
KPC-KFCMEANS	5	4	17272	131
arbitrary choosen, KFCMEANS	5	-	17596	3
KPC-RNC	10	1	16211	438
KPC-KFCMEANS	10	1	16234	493
KPC-RNC	10	4	16384	155
KPC-KFCMEANS	10	4	16372	203
arbitrary choosen, KFCMEANS	10	-	16780	5

5 Conclusion

We have described a meta-algorithm that performs parallelized clusterings with Kernel based methods and merges the results iteratively. The necessary extensions to the distance calculations and the assignment steps of Kernel Fuzzy C-Means, Kernel K-Means and Relational Neural Gas to include sample weightings have been shown. Experimentally we have observed that the loss of accuracy is rather low and the parallelized KPC algorithm can be used for vast real-life datasets that otherwise could not be clustered. As such a real-life dataset we have choosen the Enron Emails which have approximately $1,900,000$ total words and have shown that the parallelized KPC method performs far better than a simple randomized approach. By the integration of the Kernel Fuzzy C-Means algorithm this parallelized method can determine soft memberships of samples to cluster. The real usefulness of those soft memberships can be better determined by datasets that are naturally fuzzy which might be done in another contribution.

References

1. Fahim, A.M., Salem, A.M., Torkey, F.A., Ramadan, M.A.: An efficient enhanced k-means clustering algorithm. Journal of Zhejiang University SCIENCE A, 1626–1633 (2006) ISSN 1009-3095
2. Ng, R.T., Han, J.: Efficient and Effective Clustering Methods for Spatial Data Mining. In: Proceedings of the 20th VLDB Conference, pp. 286–296. Morgan Kaufmann Publishers, San Francisco (1994)
3. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering Data Streams: Theory and Practice. Proceedings of IEEE Transactions on Knowledge and Data Engineering 15(3), 515–528 (2003)
4. Kantabutra, S., Couch, A.L.: Parallel K-means Clustering Algorithm on NOWs. NECTEC Technical Journal 1(6) (2000)
5. Alex, N., Hammer, B.: Parallelizing single patch pass clustering. In: ESANN 2008 (2008) ISBN 2-930307-08-0
6. Zhang, R., Rudnicky, A.I.: A Large Scale Clustering Scheme for Kernel K-Means. In: ICPR 2002, 16th International Conference on Pattern Recognition, vol. 4, p. 40289 (2002)
7. Hasenfuss, A., Hammer, B., Rossi, F.: Patch Relational Neural Gas Clustering of Huge Dissimilarity Datasets. In: Prevost, L., Marinai, S., Schwenker, F. (eds.) ANNPR 2008. LNCS (LNAI), vol. 5064, pp. 1–12. Springer, Heidelberg (2008)
8. Zhang, D.Q., Chen, S.C.: Fuzzy clustering using kernel methods. In: International Conference of Control and Automatation (ICCA 2002), Xiamen, China, pp. 123–128 (2002)
9. Hammer, B., Hasenfuss, A.: Relational Neural Gas. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS (LNAI), vol. 4667, pp. 190–204. Springer, Heidelberg (2007)
10. Jebara, T., Kondor, R., Howard, A.: Probability Product Kernels. Journal of Machine Learning Research 5, 819–844 (2004)
11. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository (2009), `http://www.ics.uci.edu/~mlearn/MLRepository.html`