Journal of Machine Learning Research (2004) Submitted May 2004; Published

Application of the Cross-Entropy Method to Clustering and Vector Quantization

Dirk P. Kroese KROESE@MATHS.UQ.EDU.AU
Department of Mathematics
The University of Queensland
Brisbane 4072, Australia

Reuven Y. Rubinstein IERRR01@IE.TECHNION.AC.IL
Faculty of Industrial Engineering and Management
Israel Istitute of Technology (Technion)
Haifa, Israel

Thomas Taimre TTAIMRE@MATHS.UQ.EDU.AU
Department of Mathematics
The University of Queensland
Brisbane 4072, Australia

Editor:

Abstract

We apply the cross-entropy (CE) method to problems in clustering and vector quantization. The CE algorithm involves the following iterative steps: (a) the generation of clusters according to a certain parametric probability distribution, (b) updating the parameters of this distribution according to the Kullback-Leibler cross-entropy. Through various numerical experiments we demonstrate the high accuracy of the CE algorithm and show that it can generate near-optimal clusters for fairly large data sets. We compare the CE method with well-known clustering and vector quantization methods such as K-means, fuzzy K-means and linear vector quantization, and apply each method to benchmark and image analysis data.

Keywords: Cross-Entropy, Clustering, Vector Quantization, Simulation.

1. Introduction

Clustering and vector quantization are concerned with the grouping of unlabeled "feature" vectors into clusters such that samples within a cluster are more similar to each other than samples belonging to different clusters. Usually, it is assumed that the number of clusters is known in advance, but otherwise no prior information is given about the data. Applications of clustering and vector quantization can be found in the areas of communication, data compression and storage, database searching, pattern matching, and object recognition (Ahalt et al., 1990; Salomon, 2000; Webb, 1999). A good reference for clustering and vector quantization is Duda et al. (2001).

In mathematical terms, the clustering problem reads as follows: Given a dataset $\mathcal{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ of points in some d-dimensional Euclidean space, partition the data into K clusters $R_1, \ldots, R_K$ (with $R_i \cap R_j = \emptyset$, for $i \neq j$, and $\cup_j R_j = \mathcal{Z}$), such that some empirical

loss function (performance measure) is minimized. A typical loss function is

$$\sum_{j=1}^{K} \sum_{\mathbf{z} \in R_j} ||\mathbf{z} - \mathbf{c}_j||^2, \tag{1}$$

where $\mathbf{c}_j$ represents the cluster center or *centroid* of cluster R_j. The objective is to find a vector of centroids $(\mathbf{c}_1, \ldots, \mathbf{c}_K)$ and the corresponding partition $\{R_1, \ldots, R_K\}$ that minimizes (1). This definition also combines both the encoding and decoding steps in *vector quantization* (Webb, 1999). Namely, we wish to "quantize" or "encode" the vectors in $\mathcal{Z}$ in such a way that each vector is represented by one of K *source vectors* $\mathbf{c}_1, \ldots, \mathbf{c}_K$, such that the loss (1) of this representation is minimized.

Most well-known clustering and vector quantization methods update the vector of centroids, starting from some initial choice and using iterative (typically gradient-based) procedures. It is important to realize that in this case (1) is seen as a function of the centroids, where each point $\mathbf{z}$ is assigned to the nearest centroid, thus determining the clusters. It is well known that this type of problem — optimization with respect to the centroids — is highly multi-extremal and, depending on the initial clusters, the gradient-based procedures converge to a *local minimum* rather than a global minimum. A standard heuristic is the *K-means* (KM) algorithm (Webb, 1999); a useful modification is the *fuzzy K-means* (FKM) algorithm (Bezdek, 1981). Another well-known method is the *linear vector quantization* (LVQ) algorithm. A detailed description of various types of clustering methods may be found in Duda et al. (2001) and the accompanying Stork and Yom-Tov (2004). For convenience we have summarized the KM, FKM and LVQ algorithms in the appendix.

An alternative approach to optimizing (1) is to view the loss function as a function of the clusters, rather than the centroids. More precisely, denoting $\mathbf{x} = (x_1, \ldots, x_n)$ the *cluster vector*, with $x_i = j$ if $\mathbf{z}_i \in R_j$, and letting $\mathbf{z}_{ij} = I_{\{x_i = j\}}\, \mathbf{z}_i$ (here I denotes the indicator function), we can write (1) as

$$\sum_{j=1}^{K} \sum_{i=1}^{n} I_{\{x_i = j\}}\, ||\mathbf{z}_{ij} - \mathbf{c}_j||^2 \,, \tag{2}$$

where the centroids are determined as

$$\mathbf{c}_j = \frac{1}{n_j} \sum_{i=1}^{n} \mathbf{z}_{ij}, \tag{3}$$

with $n_j = \sum_{i=1}^{n} I_{\{x_i = j\}}$ being the number of points in the j-th cluster.

In this paper we introduce the CE method (Rubinstein and Kroese, 2004)—see de Boer et al. (2004) for a tutorial— as an alternative to KM, FKM and LVQ, for solving the clustering problem. In correspondence with the two scenarios discussed above we present two different settings of the main CE Algorithm, which is given in Section A (appendix). Our first setting is based on reducing the clustering problem to a combinatorial partition problem with n nodes and K partitions, while in the second setting we view (1) as a continuous multi-extremal optimization problem. That is, in the first setting the decision variable is the cluster vector $\mathbf{x}$ and in the second setting it is the vector of centroids $(\mathbf{c}_1, \ldots, \mathbf{c}_K)$. Both settings are treated in (Rubinstein and Kroese, 2004).

For more references on CE for solving combinatorial and continuous multi-extremal problems see Keith and Kroese (2002) and Rubinstein (1999). Alternative well-known heuristics, capable of handling the clustering problem, are tabu search (Glover and Laguna, 1993), genetic algorithms (Goldberg, 1989), nested partitioning (Shi and Olafsson, 2000) and the Ant Colony Optimization (ACO) meta-heuristic (Dorigo and Caro, 1999). A fundamentally different approach to clustering analysis is to assume that the data comes from a mixture of (usually Gaussian) distributions; and the objective is to estimate the parameters of this mixture by maximizing the likelihood function. In Botev and Kroese (2004) a CE approach to global likelihood optimization for such mixture models is given, with good results when compared with the EM algorithm (McLachlan and Krishnan, 1997).

The rest of the paper is organized as follows. In Sections 2 and 3 we present the clustering problem as a combinatorial partition and a continuous multi-extremal problem, respectively. In Section 4 numerical results are given for several benchmark problems and for a real problem concerning texture images. Here we also compare the efficiency of the CE method with the well-known KM, FKM and LVQ algorithms, and show how CE outperforms these standard methods. In Section 5 we present the conclusions and give directions for future research. In the appendix we recapitulate, from Rubinstein and Kroese (2004), the main CE Algorithm for solving combinatorial and continuous multi-extremal optimization problems, and briefly describe the KM, FKM and LVQ algorithms.

2. The Clustering Problem as a Partition Problem

In this section we view the optimization of (1) as a combinatorial partition problem with K partitions. The idea is to partition the points into K disjoint clusters such that the cost of this partition (the loss function) is minimized. In particular, the clusters $R_1, \ldots, R_K$ are represented through a cluster vector $\mathbf{x} \in \mathcal{X} = \{1, \ldots, n\}^K$ as in (2). Thus, $x_i = j$ means that point $\mathbf{z}_i$ belongs to the j-th cluster. In this case, the "trajectory generation" of the CE Algorithm A.1 consists of drawing random cluster vectors $\mathbf{X} \in \mathcal{X}$ according to an n-dimensional discrete distribution with independent marginals, such that $\mathbb{P}(X_i = j) = p_{ij}$, $i = 1, \ldots, n,\ j = 1, \ldots, K$. Thus, in Algorithm A.1, the parameter vector $\mathbf{v}$ consists of the probabilities $\{p_{ij}\}$. For $K = 2$ we may, alternatively, let $\mathbf{X}$ be a binary vector with independent components, the i-th component being 1 with probability p_i and 0 with probability $1 - p_i$.

With the performance $S(\mathbf{x})$ given in (2), and the centroids defined via (3), the updating rule for p_{ij} is

$$\widehat{p}_{t,ij} = \frac{\sum_{k=1}^{N} I_{\{S(\mathbf{X}_k) \leqslant \widehat{\gamma}_t\}} I_{\{X_{ki} = j\}}}{\sum_{k=1}^{N} I_{\{S(\mathbf{X}_k) \leqslant \widehat{\gamma}_t\}}} \,. \tag{4}$$

This has a very simple interpretation: we update the probability that $X_i = j$ by taking the fraction of times that $X_i = j$ for the elite samples.

Example 1 Let $n = 5$ and $K = 2$. To generate a feasible cluster we draw $\mathbf{X}$ from a 5-dimensional Bernoulli distribution $\mathsf{Ber}(\mathbf{p})$ with independent marginals. Assume that a particular outcome of $\mathbf{X}$ is $\mathbf{x} = (1, 0, 0, 1, 0)$. The associated partition is, clearly, $\{R_1, R_2\} = \{\{1, 4\}, \{2, 3, 5\}\}$. In this case the loss and the centroids can be explicitly calculated, pro-

vided the points $\boldsymbol{z}_1, \boldsymbol{z}_2,\ \boldsymbol{z}_3,\ \boldsymbol{z}_4,\ \boldsymbol{z}_5$ are given. Namely, the loss is

$$||\boldsymbol{z}_1 - \mathbf{c}_1||^2 + ||\boldsymbol{z}_4 - \mathbf{c}_1||^2 + \\ ||\boldsymbol{z}_2 - \mathbf{c}_2||^2 + ||\boldsymbol{z}_3 - \mathbf{c}_2||^2 + ||\boldsymbol{z}_5 - \mathbf{c}_2||^2,$$

and the centroids are $\mathbf{c}_1 = \frac{1}{2}(\boldsymbol{z}_1 + \boldsymbol{z}_4)$ and $\mathbf{c}_2 = \frac{1}{3}(\boldsymbol{z}_2 + \boldsymbol{z}_3 + \boldsymbol{z}_5)$, respectively.

3. The Clustering Problem as a Continuous Multi-Extremal Problem

We next present a CE approach for solving the clustering problem by viewing it as a continuous multi-extremal optimization problem where, as in the KM method, the centroids $\mathbf{c}_1, \ldots, \mathbf{c}_K$ are the decision variables. In short, we consider the program

$$\min_{\mathbf{c}_1,\ldots,\mathbf{c}_K} S(\mathbf{c}_1, \ldots, \mathbf{c}_K) = \min_{\mathbf{c}_1,\ldots,\mathbf{c}_K} \sum_{j=1}^{K} \sum_{\mathbf{z}\in R_j} ||\mathbf{z} - \mathbf{c}_j||^2, \tag{5}$$

where $R_j = \{\mathbf{z} : ||\mathbf{z} - \mathbf{c}_j|| < ||\mathbf{z} - \mathbf{c}_k||,\ k \neq j\}$. That is, R_j is the set of data points that are closer to $\mathbf{c}_j$ than to any other centroid.

For better insight and easy reference we consider the program (5) with $K = 2$ clusters and present the main steps of the CE method while using Gaussian pdfs for updating the centroids $\mathbf{c}_j,\ j = 1, 2$ and assuming that each $\mathbf{z}_i \in \mathbb{R}^2$. Associated with the program (5) are two 2-dimensional normal distributions $\mathsf{N}(\boldsymbol{\mu}_1, \Sigma_1)$ and $\mathsf{N}(\boldsymbol{\mu}_2, \Sigma_2)$, where $\boldsymbol{\mu}_i$ and Σ_i, $i = 1, 2$ are the corresponding mean vectors and covariance matrices, respectively. As in a typical CE application for continuous multi-extremal optimization, we set the initial matrices Σ_1, Σ_2 to be diagonal (with quite large variances on the diagonals) and then we proceed according to Algorithm A.1:

1. Choose, deterministically or randomly, the initial mean vectors and covariance matrices $\boldsymbol{\mu}_i, \Sigma_i$, $i = 1, 2$.

2. Generate $K = 2$ sequences of centroids (for cluster 1 and 2, respectively)

   $$\mathbf{Y}_{11}, \ldots, \mathbf{Y}_{1N} \quad \text{and} \quad \mathbf{Y}_{21}, \ldots, \mathbf{Y}_{2N},$$

   according to $\mathbf{Y}_{jk} \sim \mathsf{N}(\boldsymbol{\mu}_j, \Sigma_j)$, $j = 1, 2$, independently. For each $k = 1, \ldots, N$ calculate the objective function as in (5), with $\mathbf{c}_j$ replaced by $\mathbf{Y}_{jk},\ j = 1, 2$.

3. Complete Steps 2,3 and 4 of Algorithm A.1. In particular, update the parameters $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)$ and (Σ_1, Σ_2), according to the Kullback-Leibler cross-entropy (see Remark 2).

4. If the stopping criterion (see Remark 1) is met, then stop and accept the resulting parameter vector $(\boldsymbol{\mu}_{1T}, \boldsymbol{\mu}_{2T})$ (at the final T-th iteration) as the estimate of the true optimal solution $(\mathbf{c}_1^*, \mathbf{c}_2^*)$ to the program (5); otherwise reiterate from Step 2.

Remark 1 (Stopping Criterion) There are many possible variations to the the standard stopping criterion in Step 5 of Algorithm A.1. In particular, for continuous optimization, criterion (13) may not be appropriate, as the $\widehat{\gamma}_t$ may, sometimes, never be equal, thus preventing the algorithm from stopping. An alternative, for the present problem, is to stop when the maximum of the diagonal elements in Σ_{1T} and Σ_{2T} is less than some η, say 10^{-4}.

Remark 2 (Parameter Updating) Using Remark 6 we see that the means and variances for each centroid are updated simply as the corresponding sample mean and sample variance of the $N^{\text{elite}} = \lceil \varrho N \rceil$ elite samples. Specifically, if $\mathbf{X}_1, \ldots, \mathbf{X}_{N^{\text{elite}}}$ are the elite samples corresponding to a specific centroid (for cluster 1 or 2), then the related $\boldsymbol{\mu}$ and Σ are updated as

$$\widehat{\boldsymbol{\mu}} = \frac{1}{N^{\text{elite}}} \sum_{i=1}^{N^{\text{elite}}} \mathbf{X}_i$$

and

$$\widehat{\Sigma} = \frac{1}{N^{\text{elite}}} \sum_{i=1}^{N^{\text{elite}}} (\mathbf{X}_i - \widehat{\boldsymbol{\mu}})(\mathbf{X}_i - \widehat{\boldsymbol{\mu}})^T \,.$$

Note that we have not assumed independent components for each centroid distribution $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$. Therefore, in the 2-dimensional case we need to update 5 parameters for each centroid. For a d-dimensional normal distribution the number of parameters is $d+(d+1)d/2$. However, if we use *independent* components for each $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$ centroid distribution, then the number of distributional parameters is $2\,d$, because only the means and variances need to be updated; the off-diagonal elements of Σ are equal 0. It follows that for K clusters the total number of decision variables is $2\,d\,K$, when using independent components. Henceforth we will only consider the case with independent components.

Remark 3 (Starting Positions) One advantage of the CE method is that, as a global optimization method, it is very robust with respect to the initial positions of the centroids. Provided that the initial standard deviations are chosen large enough, the initial means have little or no effect on the accuracy and convergence speed of the algorithm. In general we choose the initial means and standard deviations such that the initial sampling distribution is fairly "uniform" over the smallest rectangle that contains the data points. Practically, this means that the initial standard deviations should not be too small, say equal to the width or height of this "bounding box."

For the KM method, however, a correct choice of starting positions is essential. A well-known data-dependent initialization method is to generate the starting positions independently, drawing each centroid from a d-dimensional Gaussian distribution $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu}$ is the sample mean of the data and Σ the sample covariance matrix of the data.

Remark 4 (Modifications) Various modifications to the standard CE algorithm can be found in Rubinstein and Kroese (2004). In our numerical experiments the following two modifications proved useful.

1. **Injection.** The idea behind the injection modification, first proposed in Botev and Kroese (2004), is to inject extra variance into the sampling distribution in order to avoid premature "shrinkage" of the distribution. More precisely, let S_t^* denote the best performance found at the t-th iteration, and σ_t^* denote the largest standard deviation at the t-th iteration. If σ_t^* is sufficiently small, and $|S_t^* - S_{t-1}^*|$ is also small, then add $B = c\,|S_t^* - S_{t-1}^*|$ to each standard deviation, for some fixed c. Appropriate values for c and δ vary; however, for the texture image data in Section 4, $c = 50$ and

$\delta = 0.05$ were used. When using CE with injection, a possible stopping criterion (see also Remark 1) is to stop once a certain number of injections, $\text{Inj}_{\max}$ say, is reached.

Note that if B is very small ($\leqslant \delta$, say) we can just add some constant value, say δ, to avoid having injections that are too small.

2. **Componentwise updating.** The idea behind componentwise updating is, as the name suggests, to update the parameter vectors component–by–component. In particular, only a *single* component of one of the mean vectors is updated at each iteration. The order in which the components are updated can be either according to some fixed permutation, such as updating μ_{11} first, then μ_{12}, and so on, or according to some other (possibly random) permutation. In our numerical experiments we chose the second option. Componentwise updating is often used in simulated annealing; see e.g., Geman and Geman (1984). However, in the context of CE this is, to our knowledge, the first application of its kind.

4. Numerical Experiments

In this section we present numerical experiments using the CE Algorithm A.1 as well as the three well-known clustering heuristics KM, FKM and LVQ. The matlab code for these last three algorithms was taken from the matlab *classification toolbox* (Stork and Yom-Tov, 2004); see also Duda et al. (2001). We apply the methods first to two benchmark examples and then to a practical application in image analysis. We found that, in the examples, the continuous optimization approach of Section 3 is more accurate than the discrete optimization approach of Section 2. Hence we only present our numerical results for the former.

4.1 Benchmark Problems

Two well-known 2-dimensional data sets were used from Stork and Yom-Tov (2004):

(a) Banana data: Points are scattered around a segment of a circle.

(b) 3-Gaussian mixture data: Points are generated from a mixture of three 2-dimensional Gaussian distributions.

Generation of these data sets is straightforward. For convenience a banana data generation algorithm is included in Section A.

Tables 1–3 present a comparative study of CE Algorithm A.1 and the traditional clustering ones for $n = 200$ and various cluster sizes K, on the data sets (a) and (b). In all experiments, we use $\alpha = 0.7$ and $\varrho = 0.025$. In all cases, a sample size of $N = 800$ is taken, so that the number of elite samples is $N^{\text{elite}} = \varrho\, N = 20$. All initial standard deviations are 14 for the banana data and 6 for the 3-Gaussian data, corresponding to the width/height of the bounding box for the data. The initial means are chosen uniformly over this bounding box. The starting positions for the other algorithms are chosen according to the standard initialization procedure for the KM algorithm discussed in Remark 3. We stop the CE algorithm when the performance no longer changes in two decimal places for 10 iterations, or if the largest standard deviation is less than $\eta = 10^{-4}$.

Each method was repeated 10 times for both data sets (a) and (b). We use the following notations in the tables: T denotes the average total number of iterations; $\bar{\gamma}_T$ denotes the averaged solution over 10 runs; γ^* is the best known solution; $\bar{\varepsilon}$ denotes the average relative experimental error (based on 10 runs) with respect to the best known solution γ^*. That is,

$$\bar{\varepsilon} = \frac{\bar{\gamma}_T - \gamma^*}{\gamma^*}. \tag{6}$$

Similarly, ε_* and ε^* denote the largest and smallest relative experimental errors. Finally, CPU denotes the average CPU time in seconds on a 1.6GHz PC.

In order to accurately identify the global minimum, we repeated our experiments many times, using different ϱ, N and smoothing parameter α (see Remark 5). The smallest value found from these experiments is given by γ^* for each case. It was found that the smallest CE performance of the 10 runs gives a reliable estimate of the true global minimum.

We note that CE is quite robust with respect to the parameters N, ϱ and α. That is, similar good results were obtained for parameter choices in the ranges $N = 400 - 4000$, $\alpha = 0.2 - 0.8$ and $\varrho = 0.01 - 0.2$.

Table 1: Performance of the four different methods for the data sets (a) and (b), with $n = 200$, $K = 5$, $N = 800$, $N^{\text{elite}} = 20$, $\alpha = 0.7$.

Approach	T	$\bar{\gamma}_T$	γ^*	$\bar{\varepsilon}$	ε_*	ε^*	CPU
(a) - Banana data set							
CE	49.6	288.49	288.11	0.00	0.00	0.01	26.67
KM	9.3	294.31	288.11	0.02	0.01	0.04	0.09
FKM	80.6	290.19	288.11	0.01	0.01	0.01	0.14
LVQ	17.7	302.81	288.11	0.05	0.01	0.19	0.07
(b) - 3 Gaussian mixture							
CE	44.2	69.15	69.11	0.00	0.00	0.00	28.53
KM	7.8	81.68	69.11	0.18	0.02	0.96	0.11
FKM	43.9	69.92	69.11	0.01	0.01	0.01	0.09
LVQ	6.4	83.75	69.11	0.21	0.06	0.96	0.03

Table 2: Performance of the four different methods for the data sets (a) and (b), with $n = 200$, $K = 10$, $N = 800$, $N^{\text{elite}} = 20$, $\alpha = 0.7$.

Approach	T	$\bar{\gamma}_T$	γ^*	$\bar{\varepsilon}$	ε_*	ε^*	CPU
(a) - Banana data set							
CE	75.8	197.22	195.87	0.01	0.00	0.02	64.25
KM	9	221.49	195.87	0.13	0.03	0.18	0.20
FKM	86.1	199.36	195.87	0.02	0.01	0.03	0.20
LVQ	14	210.85	195.87	0.08	0.01	0.20	0.08
(b) - 3 Gaussian mixture							
CE	82.4	49.15	48.16	0.02	0.00	0.04	94.93
KM	10.9	58.38	48.16	0.21	0.07	0.44	0.42
FKM	63.5	49.04	48.16	0.02	0.00	0.05	0.15
LVQ	8.4	54.54	48.16	0.13	0.05	0.24	0.05

Table 3: Performance of the four different methods for the data sets (a) and (b), with $n = 200$, $K = 20$, $N = 800$, $N^{\text{elite}} = 20$, $\alpha = 0.7$.

Approach	T	$\bar{\gamma}_T$	γ^*	$\bar{\varepsilon}$	ε_*	ε^*	CPU
(a) - Banana data set							
CE	142.1	138.06	135.80	0.02	0.00	0.03	261.92
KM	10.1	169.03	135.80	0.24	0.12	0.31	1.20
FKM	385.2	141.26	135.80	0.04	0.02	0.07	1.32
LVQ	13.1	160.84	135.80	0.18	0.12	0.32	0.13
(b) - 3 Gaussian mixture							
CE	159.8	31.88	31.29	0.02	0.01	0.03	284.98
KM	10.9	45.32	31.29	0.45	0.26	0.66	2.15
FKM	108.8	32.94	31.29	0.05	0.02	0.08	0.38
LVQ	8.3	42.73	31.29	0.37	0.26	0.58	0.07

We see that the CE algorithm, although significantly slower, is more accurate and consistent than the other algorithms. Amongst the faster algorithms, FKM is by far the best for these data. Observe also from Tables 1-3 that, as K increases, the efficiency (in terms of $\bar{\varepsilon}$, ε_*, ε^*) of CE increases relative to its counterparts for KM, FKM and LVQ. In general, we found this to be the case. This can explained by arguing as follows:

1. The number of minima of the objective function in (5) increases with K.
2. The CE method, which presents a global optimization method typically avoids the local minima, and as a result settles down at the global minimum.
3. The alternatives, KM, FKM and LVQ, which present local optimization methods are typically trapped by these local minima.

It is clear that the "classical" KM method with an average relative experimental error of 10% -100% compares poorly to the CE method, which is slower but yields a vastly superior relative error of less than 1%.

Figures 1 and 2 illustrate the difference in the placement of the centroids for CE (circles) and KM (crosses) for the banana and 3-Gaussian data, respectively. Note that for the 3-Gaussian data the KM algorithm has (wrongly) placed *two* centroids in the lower left-hand cluster.

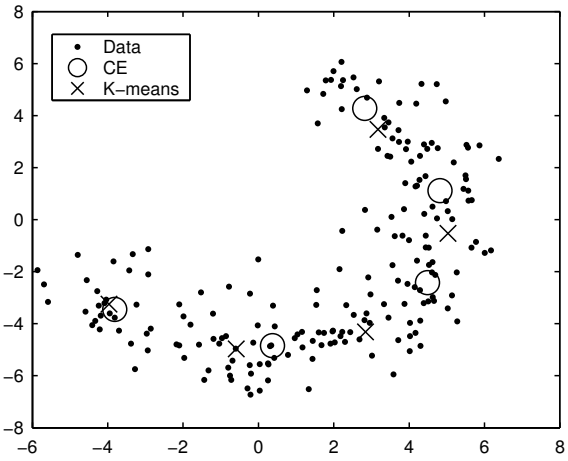

Figure 1: The CE results for vector quantization of the 2-D *banana* data set. Circles designate the final cluster centers (centroids) produced by CE. Crosses are cluster centers produced by the KM algorithm.

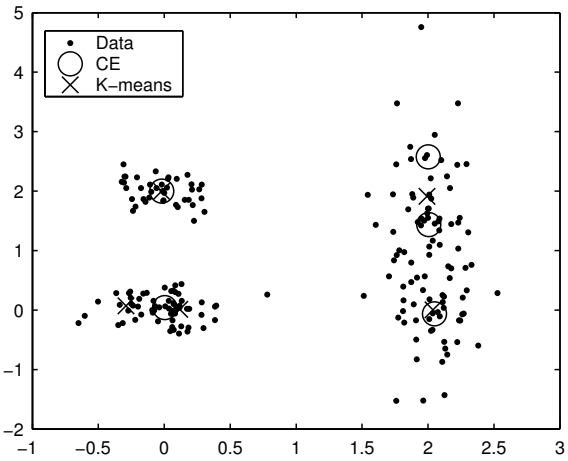

Figure 2: The CE results for vector quantization of the 2-D 3-Gaussian data set. Circles designate the final cluster centers (centroids) produced by CE. Crosses are cluster centers produced by the KM algorithm.

Finally, Table 4 presents the evolution of Algorithm A.1 for the banana problem with $n = 200$, and $K = 5$. Here S_t^* denotes the best (smallest) performance of the elite samples, $\widehat{\gamma}_t$ the worst performance of the elite samples, and σ_t^* the largest standard deviation (all at iteration t).

We found that the CE method for clustering works well even if the data set is noisy. This is in contrast to most clustering methods that often require stochastic approximation procedures, which are very slow.

Table 4: Evolution of Algorithm A.1 for the banana problem with $n = 200$, $d = 2$, $K = 5, N = 800, N^{\text{elite}} = 20$ and $\alpha = 0.7$, with $\sigma_1^* = 3$.

t	$\widehat{\gamma}_t$	S_t^*	σ_t^*
1	452.18	377.37	3.00000
2	434.09	395.01	3.20577
3	420.91	366.87	3.38746
4	403.82	356.78	3.16696
5	374.37	336.55	3.30599
6	364.62	333.48	2.94838
7	344.82	325.18	2.53459
8	333.42	313.58	2.22936
9	317.22	302.15	1.58735
10	305.09	295.90	1.15077
11	296.10	292.34	0.65408
12	291.75	290.31	0.45115
13	289.66	288.55	0.26106
14	288.80	288.50	0.18901
15	288.46	288.27	0.11668
16	288.26	288.18	0.07314
17	288.18	288.14	0.05173
18	288.14	288.13	0.03390
19	288.12	288.12	0.02360
20	288.11	288.11	0.01730
21	288.11	288.11	0.01013
22	288.11	288.11	0.00705

A fairer comparison

Because CE is much slower than the other three algorithms, one could object and say that the previous results are not fair on the other algorithms. Namely, one could run the other algorithms multiple times for each run of the CE algorithm. To assess if CE still outperforms the other algorithms in accuracy when we allow multiple runs, we conducted similar experiments as before, but now using the same amount of time for each algorithm.

All results that follow use CE with the following parameters: $N = 800$, $N^{\text{elite}} = 20$, $\alpha = 0.7$. The stopping criterion used for these experiments was the same as that used in the previous numerical experiments. As before, the initial cluster means are random on the spread of the data in each dimension, and the initial standard deviations in each dimension are equal to the maximum spread of the data over all dimensions. In the tables below, min, max and mean are the minimum, maximum and mean of the replications (10 in the case of CE; many more for the other algorithms). CPU gives the *total* CPU time in seconds. The last column lists the average number of iterations required for each replication. The 3-Gaussian and Banana data sets are the same as before. We have added a 5-Gaussian

data set. The 5-Gaussian data set was generated by drawing 300 points from a mixture of 5 Gaussian distributions with the following weights, means and covariances:

cluster	mean vector	covariance matrix	weight
1	$\begin{pmatrix} 0.6 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.3 \\ 0.3 & 1 \end{pmatrix}$	0.1
2	$\begin{pmatrix} 10 \\ -10 \end{pmatrix}$	$\begin{pmatrix} 1 & -0.2 \\ -0.2 & 1 \end{pmatrix}$	0.2
3	$\begin{pmatrix} 3 \\ -1 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$	0.1
4	$\begin{pmatrix} 0 \\ 10 \end{pmatrix}$	$\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$	0.3
5	$\begin{pmatrix} 10 \\ -30 \end{pmatrix}$	$\begin{pmatrix} 1 & 0.7 \\ 0.7 & 1 \end{pmatrix}$	0.3

As noted before, the 3-Gaussian and banana data sets were taken from the *Classification Toolbox*.

Table 5: 5-Gaussian Data Set, with $K = 5$ clusters.

	min	max	mean	trials	CPU	av its
CE	451.45	527.08	467.36	10	304.16	42
KM	452.24	2042.71	561.79	27974	304.25	8.23
FKM	453.11	742.19	497.88	12976	304.32	30.34
LVQ	452.21	2042.89	550.54	14617	304.34	9.63

Table 6: 5-Gaussian Data Set, with $K = 10$ clusters.

	min	max	mean	trials	CPU	av its
CE	326.71	354.36	339.82	10	1135.95	83.9
M	330.91	785.80	416.41	48798	1136.26	10.69
FKM	328.89	468.04	347.10	11415	1136.82	82.85
LVQ	331.07	786.14	419.78	26657	1136.17	12.55

Table 7: 5-Gaussian Data Set, with $K = 20$ clusters.

	min	max	mean	trials	CPU	av its
CE	235.58	269.89	246.74	10	4037.99	129.9
KM	263.14	547.95	346.79	87992	4038.82	11.39
FKM	234.58	297.52	249.41	12638	4039.68	130.68
LVQ	262.05	523.59	342.67	48548	4038.50	14.95

Table 8: 3-Gaussian Data Set, with $K = 5$ clusters.

	min	max	mean	trials	CPU	av its
CE	69.11	69.11	69.11	10	210.25	39.3
KM	70.44	220.70	79.05	23894	210.30	7.62
FKM	69.92	78.64	71.98	7537	210.36	50.85
LVQ	69.84	153.65	78.44	16604	210.39	6.51

Table 9: 3-Gaussian Data Set, with $K = 10$ clusters.

	min	max	mean	trials	CPU	av its
CE	48.40	50.26	49.58	10	636.00	62.7
KM	48.69	129.60	56.63	32390	636.19	10.94
FKM	48.20	58.37	49.25	8136	636.58	89.32
LVQ	48.68	124.65	56.04	24270	636.18	8.39

Table 10: 3-Gaussian Data Set, with $K = 20$ clusters.

	min	max	mean	trials	CPU	av its
CE	31.20	32.69	31.97	10	2235.28	100.5
KM	33.84	69.43	44.23	61189	2235.54	10.71
FKM	31.30	36.74	32.86	6728	2238.25	192.58
LVQ	33.11	63.23	43.16	45340	2235.55	9.49

Table 11: Banana Data Set, with $K = 5$ clusters.

	min	max	mean	trials	CPU	av its
CE	288.11	288.50	288.15	10	211.50	39.3
KM	289.29	549.54	302.16	18803	211.59	10.90
FKM	290.19	290.19	290.19	5253	211.69	75.70
LVQ	289.26	413.96	300.60	7754	211.68	14.96

Table 12: Banana Data Set, with $K = 10$ clusters.

	min	max	mean	trials	CPU	av its
CE	195.87	199.29	196.94	10	629.52	62.3
KM	197.33	290.44	218.77	31746	629.69	10.6
FKM	198.22	207.94	198.78	6967	630.01	102.74
LVQ	197.27	277.19	216.07	14170	629.89	14.05

Table 13: Banana Data Set, with $K = 20$ clusters.

	min	max	mean	trials	CPU	av its
CE	136.36	141.82	138.26	10	2151.91	97.4
KM	143.52	220.93	168.61	63773	2152.57	9.50
FKM	137.29	151.56	141.89	5377	2155.01	240.10
LVQ	142.89	201.64	162.56	29306	2152.49	13.30

4.2 Image Texture Data

In this section, we apply the clustering procedure to texture images. Texture data usually exhibits a complex spatial–frequency pattern, which is hard to describe. Some examples of real-life textures are depicted in Figure 3, where a number of different texture patterns are easily distinguished by human eye. However, such a task is not trivial using clustering algorithms. There are well-known techniques and algorithm for analyzing and classifying texture patterns. Some of them are based on statistical autocorrelation properties (Brown, 1992), wavelet transform (Chen and Kundu, 1995) and Markov random fields (Betke and Makris, 1995).

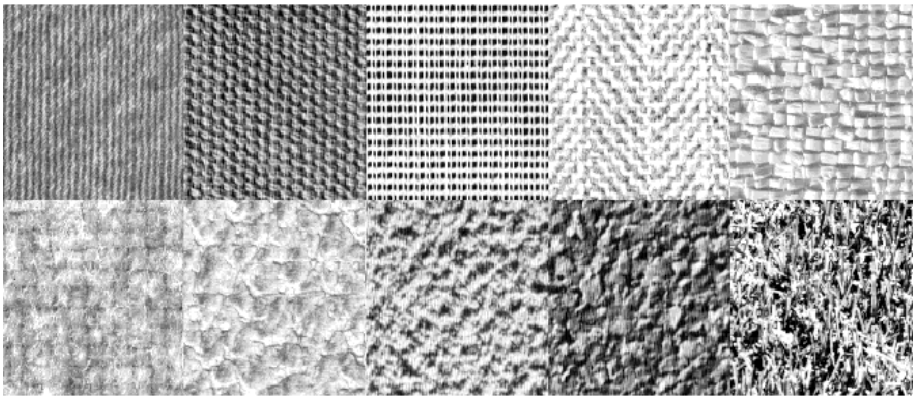

Figure 3: Different Image Texture Patterns.

To proceed, note that the gray scale value of each pixel in a texture image is meaningless without its neighbors. A common way to characterize a texture is to examine the entire neighborhood of each pixel. In our examples we use 5×5 pixel neighborhood as a feature patch. It is equivalent to a 25-dimensional data vector $\mathbf{z}_i$, where i corresponds to a specific pixel location in the image. The entire texture image produces a large data set of texture vectors $\{\mathbf{z}_i\}$. The high self-similarity of these texture patterns results in a large number of vectors clustered around the centroids $\{\mathbf{c}_j\}$. We assume that $K = 5$ clusters and a total of $n = 256$ points are sufficient to describe the textures of Figure 3 with low distortion. Table 14 presents the comparative study of the CE, KM, FKM and LVQ algorithms for a "raffia" test image. Similar to the other tables, each approach was run (repeated) 10 times. Figure 4 shows the original raffia texture image, taken from the *USC-SIPI Image Database* (usc).

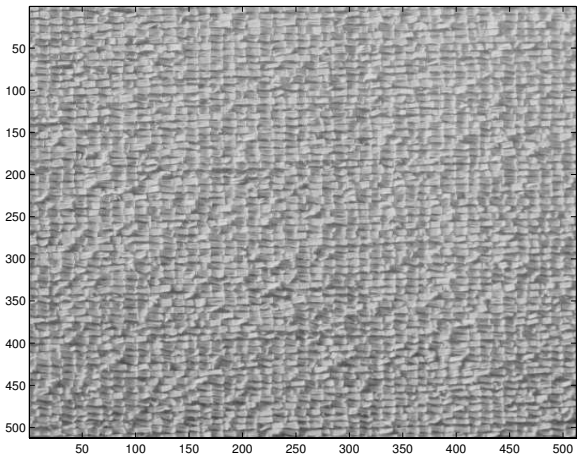

Figure 4: Raffia Image.

Figure 5 depicts our actual test image, which is a 20×20 subimage of the original raffia texture, chosen to represent the whole image.

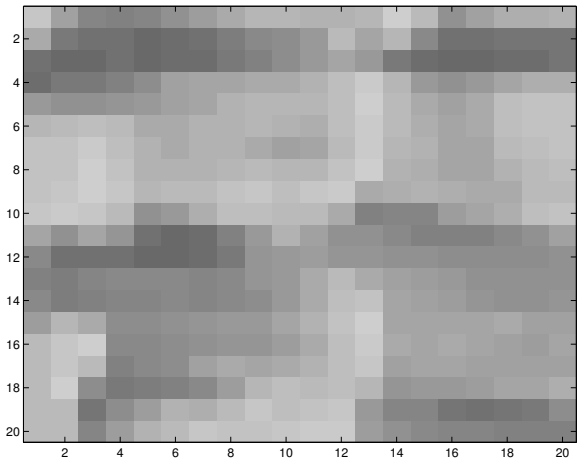

Figure 5: Raffia test image.

Figure 6 depicts all of the 256 subimages of size 5×5 generated from the 20×20 raffia test image.

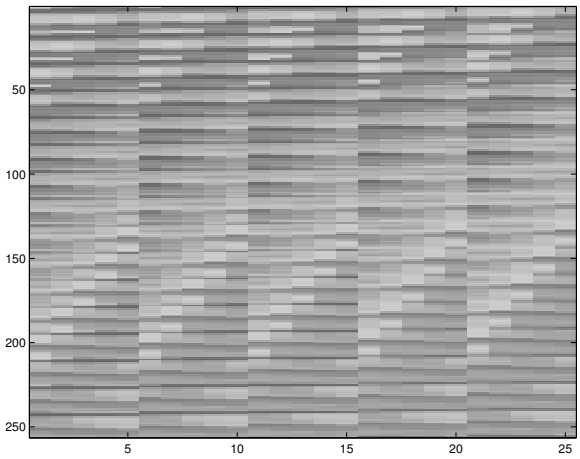

Figure 6: Collection of 256 subimages of size 5×5 of the raffia test image.

It is important to note the following:

- In contrast to the banana and Gaussian cluster test cases before, the KM and LVQ algorithms perform significantly better than FKM here.
- When applying the standard CE algorithm (without modifications), our results (not presented here) indicate that both KM and LVQ perform better, both in accuracy and speed, than CE.
- However, when applying the componentwise updating and injection modifications discussed in Section 3, CE outperforms all other methods.

All results that follow use CE with the componentwise updating, and injection, with the following parameters: $N = 100$, $N^{\text{elite}} = 10$, $\alpha = 0.9$, $\text{Inj}_{\max}$=100. The initial means are random on the spread of the data in each dimension, and the initial standard deviations in each dimension are equal to the maximum spread of the data over all dimensions. We stop when the number of injections that have occurred so far is equal to the maximum number of injections allowed (100 here). In all figures, the image grey levels take values between 0 and 1.

Table 14: Raffia Image Data Set, with $K = 5$ clusters

	min	mean	max	$\bar{\varepsilon}$	ε_*	ε^*	trials	CPU	av its
CE	83.67	84.21	85.45	0.0065	0	0.0213	10	4057.25	19.2
KM	83.81	84.61	95.80	0.0112	0.0017	0.1450	70363	4057.62	11.67
FKM	91.93	91.93	91.93	0.0987	0.0987	0.0987	13806	4058.70	174.03
LVQ	83.78	84.76	95.81	0.0130	0.0013	0.1451	61847	4057.73	10.12

Figure 7 depicts the images corresponding to the optimal five cluster centers (5 vectors of length 25) found by CE.

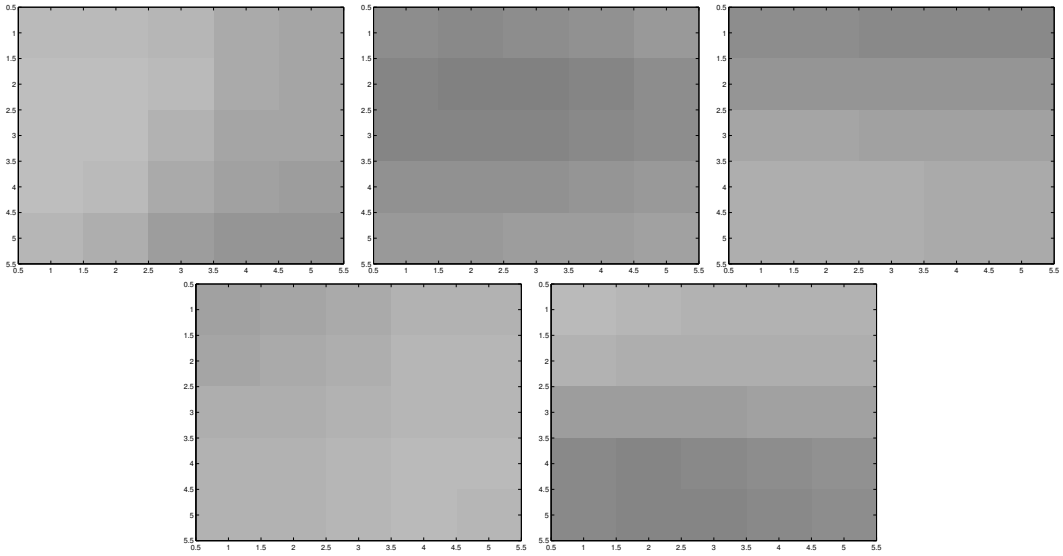

Figure 7: The optimal five cluster centers found by CE.

5. Conclusions and Directions for Future Research

This paper presents an application of the cross-entropy method to the clustering and vector quantization problems.

The proposed algorithm involves the generation of random clusters using either independent k-point distributions (Section 2) or independent Gaussian distributions (Section 3), followed by an updating of the parameters of the associated distributions using cross-entropy minimization. Our numerical studies suggest that the proposed algorithm is fast and reliable, in the sense that in approximately 99% of the cases the relative error ε does not exceed 1%. Our main conclusion is that the CE algorithm may be considered as an alternative to the standard clustering methods like KM, FKM and LVQ method.

Further topics for investigation include (a) establishing convergence of Algorithm A.1 for finite sampling (i.e., $N < \infty$) with emphasis on the complexity and the speed of convergence under the suggested stopping rules; (b) establishing confidence intervals (regions) for the optimal solution; and (c) application of parallel optimization techniques to the proposed methodology;

We note that in addition to its simplicity, the advantage of using the CE method is that it does not require direct estimation of the gradients, as many other algorithms do (for example, the stochastic approximation, steepest ascent or conjugate gradient method). Moreover, as a global optimization procedure the CE method is quite robust with respect to starting conditions and sampling errors, in contrast to some other heuristics, such as simulated annealing or guided local search.

Although in the present examples the discrete (partition) CE approach did not compete well with the continuous CE approach, the former may be useful when the performance is a complicated function of the data. For example, the data could represent a collection of proteins, each of which has a list of characteristics. In this case the performance function

is not merely the sum of the Euclidean distances but some complicated function of these characteristics.

Acknowledgments

We are most grateful to Uri Dubin for his contributions to an early version of this paper. This work was partially supported by the Australian Research Council (ARC) Centre of Excellence for Mathematics and Statistics of Complex Systems (MASCOS).

Appendix A.

The CE Method

The main idea of the CE method for optimization can be stated as follows: Suppose we wish to maximize some performance function $S(\mathbf{x})$ over all states $\mathbf{x}$ in some set $\mathcal{X}$. Let us denote the maximum by γ^*, thus

$$\gamma^* = \max_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x}) \; . \tag{7}$$

First, we randomize our deterministic problem by defining a family of pdfs $\{f(\cdot;\mathbf{v}), \mathbf{v} \in \mathcal{V}\}$ on the set $\mathcal{X}$. We assume that this family includes the degenerate density at $\mathbf{x}^*$, say $f(\cdot;\mathbf{v}^*)$. Next, we associate with (7) estimation problems of the form

$$\ell(\gamma) = \mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geqslant \gamma) = \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X})\geqslant\gamma\}}. \tag{8}$$

Here, $\mathbf{X}$ is a random vector with pdf $f(\cdot;\mathbf{u})$, for some $\mathbf{u} \in \mathcal{V}$ and $\gamma \in \mathbb{R}$. This is called the *associated stochastic problem* (ASP).

Having defined an ASP, the goal of the CE algorithm is to generate a sequence of tuples $\{(\gamma_t, \mathbf{v}_t)\}$, that converge quickly to a small neighborhood of the optimal tuple $(\gamma^*, \mathbf{v}^*)$. More specifically, we choose some initial $\mathbf{v}_0$ and a not too small ϱ, say $\varrho = 10^{-2}$, and proceed as follows:

1. **Adaptive updating of γ_t.** For a fixed $\mathbf{v}_{t-1}$, let γ_t be the $(1-\varrho)$-quantile of $S(\mathbf{X})$ under $\mathbf{v}_{t-1}$. A simple estimator $\widehat{\gamma}_t$ of γ_t can be obtained by drawing a random sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from $f(\cdot;\mathbf{v}_{t-1})$ and evaluating the sample $(1-\varrho)$-quantile of the performances:
$$\widehat{\gamma}_t = S_{(\lceil (1-\varrho)N \rceil)}, \tag{9}$$
where $S_{(k)}$ denotes the k-th order statistic of $\{S(\mathbf{X}_i)\}$.

2. **Adaptive updating of $\mathbf{v}_t$.** For fixed γ_t and $\mathbf{v}_{t-1}$, derive $\mathbf{v}_t$ from the solution of the program
$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{v}_{t-1}} I_{\{S(\mathbf{X})\geqslant\gamma_t\}} \ln f(\mathbf{X};\mathbf{v}) \; . \tag{10}$$
The stochastic counterpart of (10) is as follows: for fixed $\widehat{\gamma}_t$ and $\widehat{\mathbf{v}}_{t-1}$, derive $\widehat{\mathbf{v}}_t$ from the following program
$$\max_{\mathbf{v}} \widehat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i)\geqslant\widehat{\gamma}_t\}} \ln f(\mathbf{X}_i;\mathbf{v}) \; . \tag{11}$$

Remark 5 (Smoothed Updating) Instead of updating the parameter vector $\mathbf{v}$ directly via the solution of (11) we use the following *smoothed* version

$$\widehat{\mathbf{v}}_t = \alpha \tilde{\mathbf{v}}_t + (1-\alpha)\widehat{\mathbf{v}}_{t-1}, \tag{12}$$

where $\tilde{\mathbf{v}}_t$ is the parameter vector obtained from the solution of (11), and α is called the *smoothing parameter*, with (typically) $0.7 < \alpha \leq 1$. Clearly, for $\alpha = 1$ we have our original updating rule. The reason for using the smoothed (12) instead of the original updating rule is twofold: (a) to smooth out the values of $\widehat{\mathbf{v}}_t$, (b) to reduce the probability that some component $\widehat{v}_{t,i}$ of $\widehat{\mathbf{v}}_t$ will be zero or one at the first few iterations. This is particularly important when $\widehat{\mathbf{v}}_t$ is a vector or matrix of *probabilities*. Note that for $0 < \alpha < 1$ we always have that $\widehat{v}_{t,i} > 0$, while for $\alpha = 1$ one might have (even at the first iterations) that either $\widehat{v}_{t,i} = 0$ or $\widehat{v}_{t,i} = 1$ for some indices i. As result, the algorithm could converge to a wrong solution.

Thus, the main CE optimization algorithm, which includes smoothed updating of parameter vector $\mathbf{v}$ can be summarized as follows:

Algorithm A.1 (Main CE Algorithm for Optimization)

1. *Choose some* $\widehat{\mathbf{v}}_0$. *Set* $t = 1$ *(level counter).*

2. *Generate a sample* $\mathbf{X}_1, \ldots, \mathbf{X}_N$ *from the density* $f(\cdot; \mathbf{v}_{t-1})$ *and compute the sample* $(1-\varrho)$*-quantile* $\widehat{\gamma}_t$ *of the performances according to* (9).

3. *Use the* **same** *sample* $\mathbf{X}_1, \ldots, \mathbf{X}_N$ *and solve the stochastic program* (11). *Denote the solution by* $\widetilde{\mathbf{v}}_t$.

4. *Apply* (12) *to smooth out the vector* $\widetilde{\mathbf{v}}_t$.

5. *If for some* $t \geq d$, *say* $d = 5$,

$$\widehat{\gamma}_t = \widehat{\gamma}_{t-1} = \cdots = \widehat{\gamma}_{t-d}, \tag{13}$$

   then **stop** *(let* T *denote the final iteration); otherwise set* $t = t+1$ *and reiterate from step 2.*

Note that the parameter $\mathbf{v}$ is updated in (11) only on the basis of the $(1-\varrho)\%$ best samples. These are called the *elite samples*. The main ingredients of any CE algorithm are as follows.

- **Trajectory Generation:** Generate random samples in $\mathcal{X}$ according to $f(\cdot; \mathbf{v}_{t-1})$.
- **Parameter Updating:** Update $\mathbf{v}$ on the basis of the elite samples, in order to produce better performing samples in the next iteration.

Remark 6 (Maximum Likelihood Estimate) The updating rule for $\mathbf{v}$ follows from cross-entropy minimization and often has a simple form. In particular, it is given by the *maximum likelihood* estimate of $\mathbf{v}$ based on the elite samples.

Remark 7 (Minimization) Note that for a minimization program we take the ϱ-quantile of the best (smallest) performances. Also, the $\geqslant$ sign in (10) and (11) is replaced by a $\leqslant$ sign.

K-Means (KM)

The KM algorithm consists of the following steps:

1. Initialize by assigning (randomly or deterministically) to each point in $\mathcal{Z}$ a cluster number in $\{1, \ldots, K\}$.

2. Calculate the centroids $\mathbf{c}_1, \ldots, \mathbf{c}_K$ of the clusters.

3. (Re)assign each point to the nearest centroid.

4. Repeat steps 2 and 3 until convergence is reached, for example if the clusters no longer change.

Fuzzy K-means (FKM)

The FKM algorithm consists of the following steps (see Bezdek and Hathaway (1988) for a generalization):

1. Initialize by assigning (randomly or deterministically) to each point $\mathbf{z}_j$ in $\mathcal{Z}$ a "membership" weight u_{ij} for each cluster $i \in \{1, \ldots, K\}$, such that $u_{ij} \geqslant 0$ for all i, j, and $\sum_{i=1}^{K} u_{ij} = 1$ for each i.

2. Calculate the centroids of the clusters in the following way:

$$\mathbf{c}_i = \frac{\sum_{j=1}^{n} u_{ij}^2 \, \mathbf{z}_j}{\sum_{j=1}^{n} u_{ij}^2}$$

   for each $i = 1, \ldots, K$.

3. Recalculate the membership weight of each point $\mathbf{z}_j$ to every cluster i as follows:

$$u_{ij} = \frac{b_j}{\|\mathbf{z}_j - \mathbf{c}_i\|^2},$$

   where b_j is a normalization constant.

4. Repeat steps 2 and 3 until convergence is reached, for example if all of the membership weights hardly change.

Linear Vector Quantization (LVQ)

The LVQ algorithm comes in various versions. The one that is appropriate for our clustering problems is the (Type One) LVQ algorithm, when all of the data belong to only one "class" (Kohonen, 1990). It comprises the following steps:

1. Initialize (randomly or deterministically) the K centroids $\mathbf{c}_1, \ldots, \mathbf{c}_K$.

2. (Re)assign each point to the nearest centroid.

3. Calculate the centroids of the current point assignment, $\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_K$.

4. Set the new cluster centroids

$$\mathbf{c}_i = \beta_t \, \tilde{\mathbf{c}}_i + (1 - \beta_t) \, \mathbf{c}_i, \quad i = 1, \ldots, K,$$

   where β_t is some (typically small) value between 0 and 1, which could depend on the iteration counter t.

5. Repeat steps 2, 3, and 4 until convergence is reached, for example if the cluster centroids no longer change.

Remark 8 (LVQ and KM) Note that the LVQ algorithm above behaves like a "smoothed updating" version of the KM algorithm.

Generating Banana Data

Banana shaped data sets occur frequently in clustering and classification test problems. Below is a simple matlab function that generates such data.

```
function b = banana(n,sigma,radius,theta1,theta2)
% Generate a Banana-shaped data set
% n      - number of data points (default: 200)
% sigma  - move points according to a normal dist. with this
%          standard deviation in both x and y directions
%          (default: 1)
% radius - the radius of the circle (of which the "banana" is
%          an arc (default: 5)
% theta1 - starting angle of the arc (default: 9*pi/8)
% theta2 - ending angle of the arc (default: 19*pi/8)
if nargin<4,theta1=pi + pi/8;end
if nargin<5,theta2=(5*pi/2 - pi/8) - theta1;end
if nargin<3,radius=5;end
if nargin<2,sigma=1;end
if nargin<1,n=200;end

randn('seed', 123456789); %optional
rand('seed', 987654321);  %optional
angles=theta1 + rand(n,1)*theta2; % angles between pi/8 and 11*pi/8
b=radius.*[cos(angles),sin(angles)]; % transform these to random points
                                     % on an arc of a circle
b=b+sigma*randn(n,2); % shift these points off the arc ind.
                      % normally in x and y directions
```

References

The USC-SIPI Image Database. http://sipi.usc.edu/services/database/Database.html.

S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton. Competitive learning algorithms for vector quantization. *Neural Networks*, 3:277–290, 1990.

M. Betke and N. Makris. Fast object recognition in noisy images using simulated annealing. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 523–530, 1995.

J. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

J. C. Bezdek and R. J. Hathaway. Recent convergence results for the fuzzy c-means clustering algorithms. *Journal of Classification*, 5:237–247, 1988.

Z. Botev and D. P. Kroese. Global likelihood optimization via the cross-entropy method, with an application to mixture models. In R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, editors, *Proceedings of the 2004 Winter Simulation Conference*, Washington, DC, December 2004.

L. Brown. A survey of image registration techniques. Technical report, Department of Computer Science, Columbia University, 1992.

J. Chen and A. Kundu. Unsupervised texture segmentation using multichannel decomposition and hidden Markov models. *IEEE Trans. on Image Processing*, 4:603–619, 1995.

P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 2004. To appear.

M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2001.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. *IEEE Trans. on PAMI*, 6:721–741, 1984.

F. Glover and M. L. Laguna. *Modern Heuristic Techniques for Combinatorial Optimization*, chapter Chapter 3: Tabu search. Blackwell Scientific Publications, 1993.

D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

J. Keith and D. P. Kroese. Sequence alignment by rare event simulation. In *Proceedings of the 2002 Winter Simulation Conference*, pages 320–327, San Diego, 2002.

T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley & Sons, 1997.

R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 2:127–190, 1999.

R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer-Verlag, New York, 2004.

D. Salomon. *Data Compression: The Complete Reference*. Springer-Verlag, 2000.

L. Shi and S. Olafsson. Nested partitioning method for global optimization. *Operations Research*, 48(3):390–407, 2000.

D. G. Stork and E. Yom-Tov. *Computer Manual to Accompany Pattern Classification*. Wiley, 2004.

A. Webb. *Statistical Pattern Recognition*. Arnold, London, 1999.