LEARNING FINITE GAUSSIAN MIXTURES USING DIFFERENTIAL EVOLUTION

Wojciech Kwedlo[1]

[1]Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: In the paper the problem of parameter estimation of finite mixture of multivariate Gaussian distributions is considered. A new approach based on differential evolution (DE) algorithm is proposed. In order to avoid problems with infeasibility of chromosomes our version of DE uses a novel representation, in which covariance matrices are encoded using their Cholesky decomposition. Numerical experiments involved three version of DE differing by the method of selection of strategy parameters. The results of experiments, performed on two synthetic and one real dataset indicate, that our method is able to correctly identify the parameters of the mixture model. The method is also able to obtain better solutions than the classical EM algorithm.

Keywords: Gaussian mixtures, differential evolution, EM algorithm.

1. Introduction

Mixture models [13] are versatile tools used for modeling complex probability distributions. They are capable to model observations, which are produced by a random data source, randomly selected from many possible sources. Estimation of the parameters of these sources and identifying which source produced each observation leads to clustering of data [10]. Mixture models can be also used for feature selection [17] or representation of class-conditional probability density functions in discriminant analysis [9].

A finite mixture model $p(\mathbf{x}|\Theta)$ can be described by a weighted sum of $M > 1$ components $p(\mathbf{x}|\theta_m)$:

$$p(\mathbf{x}|\Theta) = \sum_{m=1}^{M} \alpha_m p(\mathbf{x}|\theta_m), \tag{1}$$

Zeszyty Naukowe Politechniki Białostockiej. Informatyka, vol. 5, pp. 19-33, 2010.

where $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ is the d-dimensional data vector, $\alpha_1, \alpha_2, \ldots, \alpha_M$ are mixing probabilities, which satisfy the following conditions:

$$\alpha_m > 0, \;\; m = 1, \ldots, M \text{ and } \sum_{m=1}^{M} \alpha_m = 1.$$

θ_m is the set of parameters defining the mth component and $\Theta = \{\theta_1, \theta_2, \ldots, \theta_M, \alpha_1, \alpha_2, \ldots, \alpha_M\}$ is the complete set of the parameters needed to define the mixture. In the paper we consider a class of finite mixture models called Gaussian mixtures in which each component $p(\mathbf{x}|\theta_m)$ follows multivariate normal (Gaussian) distribution:

$$p(\mathbf{x}|\theta_m) = \frac{1}{(2\pi)^{d/2}|\Sigma_m|^{1/2}} \exp(-\frac{1}{2}(\mathbf{x} - \mu_m)^T \Sigma_m^{-1}(\mathbf{x} - \mu_m)), \tag{2}$$

where μ_m and Σ_m denote the mean vector and covariance matrix, respectively, $|\cdot|$ denotes a determinant of a matrix. The set parameters of the mth component is $\theta_m = \{\mu_m, \Sigma_m\}$. The set of the parameters of the complete Gaussian mixture model can be defined as:

$$\Theta = \{\mu_1, \Sigma_1, \ldots, \mu_M, \Sigma_M, \alpha_1, \ldots, \alpha_M\}. \tag{3}$$

Estimation of the parameters of the mixture model is usually performed using the maximum likelihood (ML) approach. Given a set of independent and identically distributed samples $X = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$, called the training set, the log-likelihood corresponding to M-component mixture is given by:

$$\log p(X|\Theta) = \log \prod_{i=1}^{N} p(\mathbf{x}^i|\Theta) = \sum_{i=1}^{N} \log \sum_{m=1}^{M} \alpha_m p(\mathbf{x}^i|\theta_m). \tag{4}$$

The maximum likelihood estimate of the parameters

$$\Theta_{ML} = \operatorname*{argmin}_{\Theta} \{-\log p(X|\Theta)\}$$

cannot be found analytically (e.g. [3]). Therefore, the estimation must be performed using an optimization algorithm.

In the paper a novel application a global optimization algorithm called differential evolution (DE) to the problem of Gaussian mixture learning is proposed. The rest of this paper is organized as follows. Section 2 discusses the research related to our work. In Section 3 the details of the proposed application of DE to the problem of Gaussian mixture learning are described. Section 4 presents the results of computational experiments. The last section of this paper contains conclusions.

2. Related Work

The usual choice of obtaining ML parameters of a Gaussian mixture model is the EM algorithm [6]. It is iterative procedure for minimizing $-\log p(X|\Theta)$. The EM algorithm for Gaussian mixtures is easy to implement and computationally efficient [13]. However, it has an important drawback. Being a local search algorithm, it can be easily trapped in a local minimum of $-\log p(X|\Theta)$. Thus, the quality of the obtained solutions is highly dependent on the initialization of the EM algorithm. The most simple solutions include using multiple random starts and choosing the final estimate with the smallest $-\log p(X|\Theta)$ [13] and initialization by the clustering (e.g. *k*-means) algorithms [3,13].

Many more elaborate extensions of the EM approach have been suggested in order to tackle the problem of convergence to a local optimum. Verbeek et. al. [20] proposed a deterministic greedy method in which the mixture components are inserted into the mixture one after another. The method starts with the optimal one-component mixture, the parameters of which can be easily found. After each insertion of a new component the EM algorithm is applied to the new mixture. This approach is much lest sensitive to initialization than the original EM algorithm. Figueiredo and Jain [7] also used component-wise approach. Using minimum message length criterion they developed a method, which is robust with respect to the initialization and capable of discovering the number of the components. In the paper by Ueda et al. [19] the EM algorithm was extended by a split-and-merge technique in order to alleviate the problem of local optima.

Evolutionary algorithms (EAs) [14] are stochastic search techniques inspired by the concept of the Darwinian evolution. Unlike local optimization methods, e.g. the EM algorithm, they simultaneously process a population of problem solutions, which gives them the ability to escape from local optima of the fitness function. Recently, applications of EAs to the problem of ML estimation of parameters of a Gaussian mixture model have been proposed. Most of the researches used a hybrid scheme, which alternates between a step of EA consisting of selection and recombination and a step consisting of iterations of EM. This approach was used by Martinez and Vitrià [12], who employed selection and the mutation operators of evolution strategies [2]. In their method the value of the fitness function is obtained by running the EM algorithm (until it the converges) on mixture parameters encoded by a chromosome. Pernkopf and Bouchaffra [15] proposed a combination of an EA with EM, which by using fitness function based on the minimum description length principle, is able to estimate the number of the components in a mixture.

Differential evolution (DE) is an evolutionary algorithm proposed by Storn and Price [18], employing a representation based on real-valued vectors. It has been successfully applied to many optimization problems. DE is based on the usage of vector differences for perturbing the population elements. Many researches suggested extensions to the original DE. For an overview of recent developments and practical applications of DE the reader is referred to [5]. The version of DE with the self-adaptation of control parameters, which we employ in the paper, was proposed by Brest et al. [4]. According to our knowledge no application of DE to the problem of Gaussian mixture learning has been proposed so far.

3. DE algorithms for Gaussian mixtures

3.1 Differential evolution

Several versions of DE have been proposed. For the purpose of this study the most common variant is used, which, according to the classification proposed by [18] can be described as DE/rand/1/bin.

Like all EAs, DE maintains a population of S solutions of the optimization problem. At the start of the algorithm, members of the population are initialized randomly with the uniform distribution. Then DE performs multiple iterations in three consecutive steps: reproduction (creation of a temporary population), computing of the fitness function for all members of the temporary population, and selection.

Let $u_{i,G}$ denote the i-th member ($i = 1,\ldots,S$) of the population in the G-th iteration. Usually $u_{i,G}$ takes a form of a D-dimensional real-valued vector, i.e. $u_{i,G} \in \Re^D$.

Reproduction in DE creates a temporary population of trial vectors. For each solution $u_{i,G}$ a corresponding trial vector $y_{i,G}$ is obtained by mutation and crossover operators. The mutation operator generates a mutant vector $y'_{i,G}$ according to the equation:

$$y'_{i,G} = u_{a,G} + F * (u_{b,G} - u_{c,G}), \tag{5}$$

where $F \in [0,2]$ is a user-supplied parameter called amplification factor and $a,b,c \in 1,\ldots,S$ are randomly selected in such way that $a \neq b \neq c \neq i$.

The final trial vector $y_{i,G}$ is obtained by the crossover operator, which mixes the mutant vector $y'_{i,G}$ with the original vector $u_{i,G}$. Let us assume that $u_{i,G} = (u_{1i,G}, u_{2i,G}, \ldots, u_{Di,G})$. Each element $y_{ji,G}$ (where $j = 1,\ldots,D$) of the trial vector $y_{i,G}$ is generated as:

$$y_{ji,G} = \begin{cases} y'_{ji,G} & \text{if } rnd(j) < CR \text{ or } j = e \\ u_{ji,G} & \text{otherwise} \end{cases}. \tag{6}$$

where $CR \in [0,1]$ is another user-supplied parameter called crossover factor, $rnd(j)$ denotes a random number from the uniform distribution on $[0,1]$ which is generated independently for each j. $e \in 1,\ldots,S$ is a randomly chosen index which ensures that at least one element of the trial vector $y_{i,G}$ comes from the mutant vector $y'_{i,G}$.

The remaining two phases of DE generation are computation of the fitness for all members of the trial population and selection. Selection in differential evolution is very simple. The fitness of each trial solution $y_{i,G}$ is compared to the fitness of the corresponding original solution $u_{i,G}$. The trial vector $y_{i,G}$ survives into the next iteration becoming $u_{i,G+1}$ if its fitness is better. Otherwise $y_{i,G}$ is discarded and $u_{i,G+1}$ is set to $u_{i,G}$

3.2 Choice of control parameters F and CR

The parameters F and CR have a significant impact on convergence speed and robustness of the optimization process. The choice of their optimal values is an application-dependent task. In [18] the values $F = 0.5$ and $CR = 0.9$ were suggested. F and CR may be also selected using a trial-and-error approach, which requires many optimization runs and may be infeasible in many practical applications.

To alleviate the problem of parameter tuning, Brest et al. [4] proposed a self-adaptation method for F and CR. In this method, amplification and crossover factors evolve together with population members. Each member of the both trial and target populations is augmented with its own amplification factor and crossover factor. Let us denote by $F^u_{i,G}$ and $F^y_{i,G}$ the amplification factors associated with the vectors $u_{i,G}$ and $y_{i,G}$, respectively. Similarly, let us denote by $CR^u_{i,G}$ and $CR^y_{i,G}$ the crossover factors associated with the vectors $u_{i,G}$ and $y_{i,G}$, respectively.

Before the mutation $F^y_{i,G}$ is generated as:

$$F^y_{i,G} = \begin{cases} L + rnd_2 * U & \text{if } rnd_1 < \tau_1 \\ F^u_{i,G} & \text{otherwise} \end{cases}. \tag{7}$$

rnd_1 and rnd_2 are uniform random values from $[0,1]$, $\tau_1 \in [0,1]$ is the probability of choosing new random value of $F^y_{i,G}$, L and U are the parameters determining the range for $F^y_{i,G}$.

Similarly to $F^y_{i,G}$, $CR^y_{i,G}$ is generated before the mutation as:

$$CR^y_{i,G} = \begin{cases} rnd_3 & \text{if } rnd_4 < \tau_2 \\ CR^u_{i,G} & \text{otherwise} \end{cases}, \tag{8}$$

where $\tau_2 \in [0,1]$ is the probability of choosing new random value of $CR^y_{i,G}$.

$F^y_{i,G}$ obtained by (7) is used to generate the mutant vector according to (5). $CR^y_{i,G}$ obtained by (8) is used to generated the trial vector according to (6). The amplification and crossover factors undergo selection together with associated vectors. If in the selection process $u_{i,G+1}$ is set to $y_{i,G}$ then $F^u_{i,G+1}$ is set to $F^y_{i,G}$ and $CR^u_{i,G+1}$ is set to $CR^y_{i,G}$. Otherwise, $F^u_{i,G+1}$ is set to $F^u_{i,G}$ and $CR^u_{i,G+1}$ is set to $CR^u_{i,G}$.

It may seem that self-adaption of F and CR introduces another four parameters (L, U, τ_1, τ_2) which must be fine-tuned by the user by means of the trial-and-error method. However, Brest et al. [4] used fixed values of these parameters in all experiments and obtained promising results. Following their suggestion in our experiments we used $\tau_1 = \tau_2 = 0.1$. L and U were set to 0.05 and 0.35 respectively, which ensured that $F^y_{i,G} \in [0.05, 0.4]$.

The original method of Brest et. al. self-adapted both parameters F and CR. It was tested on benchmark numerical optimization problems, with dimension $D \leq 30$. However, our preliminary experiments indicated, that it yielded poor results, when applied to the much more difficult problem of Gaussian mixture learning. Therefore in the paper we propose to use a new approach, in which F is self-adapted in a manner described in [4], whereas CR is set to a constant value close to 0 according to the following experimentally developed formula: $CR = 2/D$, where D is the dimension of a population element.

3.3 The fitness function

The fitness function used by the DE is $-\log p(X|\Theta)$, where $\log p(X|\Theta)$ is defined by (4). Because of the minus sign, the algorithm is configured to minimize the fitness.

3.4 Encoding of the parameters of a Gaussian mixture model by chromosomes

In order to apply a DE algorithm to the problem of Gaussian mixture learning we have to work out a method for encoding mixture parameters (3) by real-valued vectors (chromosomes). The encoding of mixing probabilities $\alpha_1, \alpha_2, \ldots, \alpha_M$ and mean vectors $\mu_1, \mu_2, \ldots, \mu_M$ is very straightforward. The mixing probabilities (M real values) and all the elements of mean vectors (dM real values) are directly encoded in a chromosome using the floating-point representation.

Unfortunately, the elements of covariance matrices cannot be encoded in a similar way. A covariance matrix Σ_m, $m = 1, \ldots, M$ of each Gaussian component of a mixture (1) must be a symmetric positive-definite matrix, i.e. for each non-zero $\mathbf{x} \in \Re^d$ $\mathbf{x}^T \Sigma_m \mathbf{x} > 0$ [11]. Since, each Σ_m is symmetric, only its $d(d+1)/2$

diagonal and over-diagonal elements need to be encoded. However, if we encoded these elements directly, the purely random crossover and mutation operators of DE would not preserve positive-definiteness of the covariance matrices Σ_m. In almost all cases the matrices encoded in a trial vector $y_{i,G}$ would not be positive-definite and thus could not be interpreted as covariance matrices.

To overcome the above obstacle, the covariance matrices are encoded in a chromosome using their Cholesky decomposition [16]. Each covariance matrix Σ_m, $m = 1,\ldots,M$ can be expressed as a product of a lower triangular matrix L_m and its transpose:

$$\Sigma_m = L_m L_m^T. \tag{9}$$

The diagonal elements of the lower triangular matrix L_m must be strictly positive. This condition is much easier to satisfy during the DE evolution, than the positive-definiteness of the covariance matrix Σ_m. For that reason we have chosen to encode elements of L_m rather than Σ_m in a chromosome. To ensure the positive-definiteness of Σ_m we inspect the trial vector $y_{i,G}$ and check each diagonal element of L_m. If the value of the diagonal element is lower than zero, it is replaced in the trial vector $y_{i,G}$ by its absolute value.

4. Experiments

4.1 Experimental setup

In the experiments we used three versions of DE differing by the method of parameter control:

- A version using fixed values of the parameters (i.e. $F = 0.5, CR = 0.9$ as proposed in [18]) during the run of the algorithm.
- A version using self-adaptive F and CR [4] described in the subsection 3.2.
- A version using self-adaptive F only.

Additionally, the EM algorithm [13], which is the standard method for estimation of the parameters of the Gaussian Mixture Model was used in the comparison.

In all the experiments the population size S was set to 32. All the results reported in this section are the average over 30 independent runs using random starting conditions. The EM algorithm was initialized using a random method described in [7].

All the algorithms were implemented in the C++ language and compiled by the Intel C++ version 10.1 compiler using optimizing options (-O3 -ipo -xT -fno-alias). It was run on a system with two quad-core Intel Xeon 5355 (2.66 GHz) processors.

The comparison of the DE algorithms was performed on the equal CPU time basis: all three versions of DE were were allocated 200 seconds of CPU time, and the final result was the fitness of the best individual in the population obtained after 200 seconds of evolution.

4.2 Synthetic datasets

The first example uses 900 samples from 3-component bivariate mixture from [7]. For this example $\alpha_1 = \alpha_2 = \alpha_3 = 1/3$, $\mu_1 = [0,-2]^T$, $\mu_2 = [0,0]^T$, $\mu_3 = [0,2]^T$ and all three covariance matrices are equal: $\Sigma_1 = \Sigma_2 = \Sigma_3 = \left[\begin{smallmatrix} 2 & 0 \\ 0 & 0.2 \end{smallmatrix}\right]$.

Fig. 1a shows the true mixture and 900 samples of it, whereas Fig. 1b shows the mixture estimated by the DE with self-adaptive F (mixtures estimated by the other two versions were almost identical). It can be seen, that the DE algorithm has correctly identified the mixture parameters. Fig. 2 shows the convergence of three versions of DE. The curves represent the fitness of the best individual in the DE population plotted as a function of the number of fitness function calls performed by the algorithms. For that dataset all the algorithms converged towards global minimum, although the DE with self-adaptive F converged slower than the other two methods.

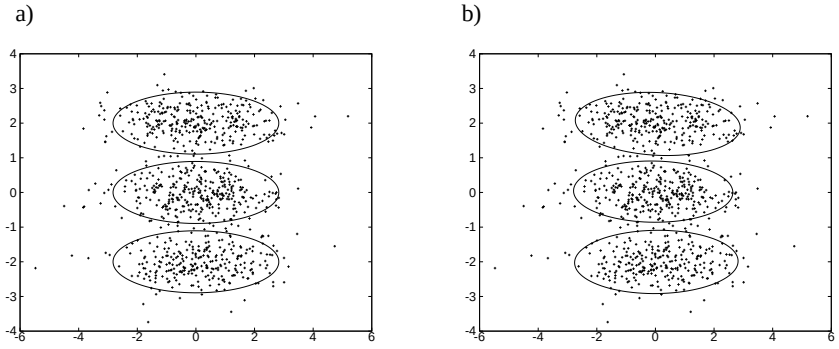

Fig. 1. Estimation of parameters of 3-component bivariate mixture. The solid ellipses are level-curves of each component: a) true mixture b) mixture estimated by DE with self-adaptive F.

In the second example, the mixture components overlap. Two of them share a common mean, but have different covariance matrices. The 1000 samples were taken from the mixture with following parameters: $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0.25$ and

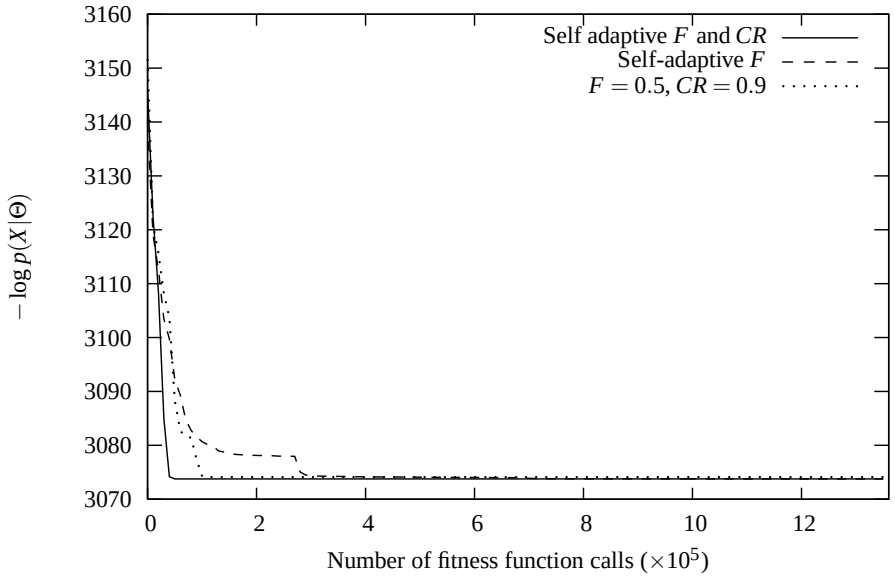

Fig. 2. Convergence of three versions of DE for the 3-component mixture. The curves are average over 30 independent runs.

$\mu_1 = \mu_2 = [-4,4]^T$, $\mu_3 = [2,2]^T$, $\mu_4 = [-1,-6]^T$ and $\Sigma_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 6 & -2 \\ 2 & 6 \end{bmatrix}$, $\Sigma_3 = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$, $\Sigma_4 = \begin{bmatrix} 0.125 & 0 \\ 0 & 0.125 \end{bmatrix}$. Fig. 3a shows the true mixture and 1000 samples of it, whereas Fig. 3b shows the mixture estimated by DE with self-adaptive F. Once again it can be seen, that the algorithm has correctly identified the mixture parameters. Fig. 4 shows the convergence of three variants of DE. This example was clearly too difficult for the DE with constant vales of F and CR as it was unable to find the global minimum of the log-likelihood. It can be also seen, that DE with self adaptive F and CR converged slightly faster than DE with self-adaptive F only.

4.3 Real-life dataset

In the last experiment, the well-known iris dataset (150 samples, $d = 4$) [8] describing iris flowers from the Gaspé peninsula was used. This dataset was obtained from the UCI machine learning repository [1]. The iris flowers described in this dataset are divided into three classes. For that reason we used $M = 3$. Of course, since it is a real-life, not synthetic dataset, the underlying probability density functions are not known and could not be visualized. Therefore Fig. 5 shows only the mixture estimated by

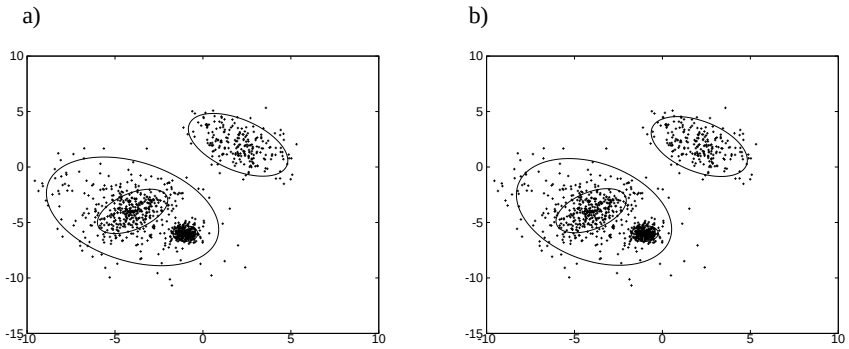

Fig. 3. Estimation of parameters of 4-component bivariate Gaussian mixture with overlapping components. The solid ellipses are level-curves of each component: a) true mixture b) mixture estimated by DE with self-adaptive F.

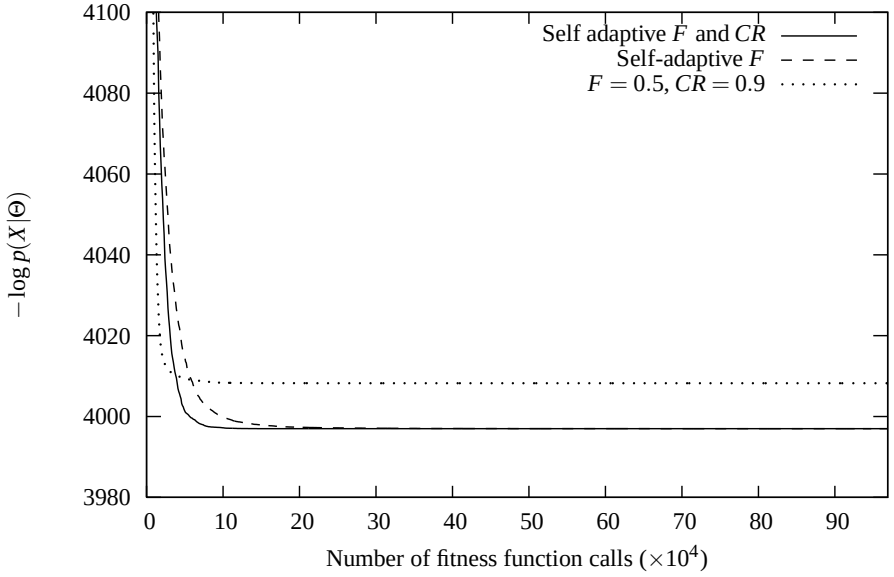

Fig. 4. Convergence of three versions of DE for the 4-component mixture with overlapping components. The curves are average over 30 independent runs.

the DE with self-adaptive F. Since iris is four-dimensional dataset, we used principal component analysis (PCA) [11] to project data and the estimated mixture on the first

two principal components. Fig. 6 shows the convergence of three versions of DE. This experiment clearly demonstrated that fast convergence in the initial phase of the evolution does not necessarily lead to best final solution, as the slowest algorithm (i.e. DE with self-adaptive F) found the solution with the lowest $-\log p(X|\Theta)$.

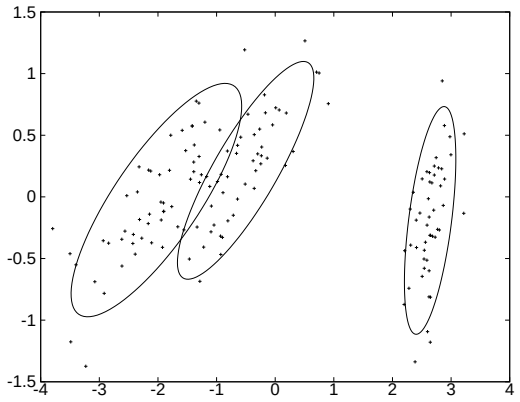

Fig. 5. iris data (projected on two principal components) and the mixture estimated by the DE with self-adaptive F. The solid ellipses are level-curves of each component.

4.4 Summary of the experiments and the comparison with the EM algorithm

The experiments are summarized by the Table 4.4, which shows the final $-\log p(X|\Theta)$ obtained by the DE algorithms after 200 seconds of evolution. Additionally, the last column of the table shows the result obtained by the EM algorithm [13]. The results suggest the following conclusions:

- All versions of DE outperform the local search algorithm (EM). However it should be noted that EM algorithm is much faster than DE, requiring much less than one second (on our hardware) to converge.
- DE with self-adaptive F only outperforms two other versions of DE.

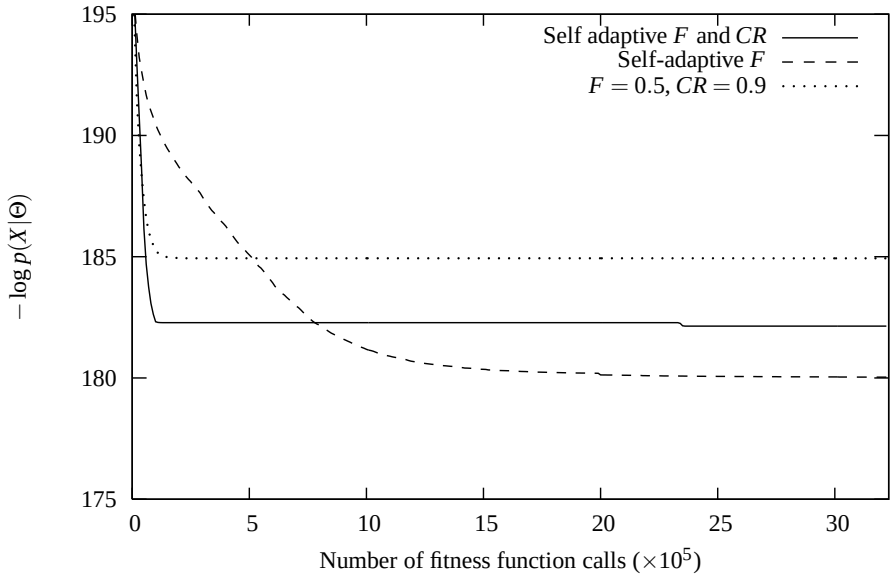

Fig. 6. Convergence of three versions of DE for the iris dataset. The curves are average over 30 independent runs.

5. Conclusions

In this paper an application of DE to the problem of Gaussian mixture learning was described. To avoid a problem with infeasibility of chromosomes we proposed a novel encoding method, in which covariance matrices were encoded using their Cholesky decomposition. In the experiments three versions of DE, differing by the method of parameter control, were examined and compared to the EM algorithm.

The results of our study allow us to recommend DE with self-adaptive F over the other two versions and the EM algorithm. Although it converges slowly in the initial phase of evolution, it is able to find solutions not worse (and sometimes better, if the problem is difficult enough as it was demonstrated by the experiment with the iris dataset) than other versions of DE.

There are several possible directions of future work. One of them is the development of a memetic algorithm combining DE with fast local search procedure (e.g. the EM algorithm). Such hybrid method would be able to retain the advantages of both approaches i.e. the fast convergence of EM and the ability find a global optimum of DE.

Table 1. The final $-\log p(X|\Theta)$ values obtained by the four tested methods. The results are average over 30 independent runs.

Dataset	$F = 0.5, CR = 0.9$	Self adaptive F and CR	Self adaptive F	EM
3-component mixture	3074.1	3073.8	3073.8	3148.8
4-component mixture	4008.2	3997	3997	4085.6
iris	184.93	182.14	180.03	187.58

Another direction of future research is related to the development of a parallel version of our approach. The most time-consuming step of the algorithm is computation of the fitness function since it requires the M passes over the training set. It would be quite straightforward to parallelize this step by concurrently computing the fitness of the different population members on different processors of a parallel system.

References

[1] A. Asuncion and D. J. Newman. UCI machine learning repository, 2007.

[2] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[4] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[5] U. K. Chakraborty, editor. *Advances in Differential Evolution*. Springer, 2008.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B*, 39(1):1–38, 1977.

[7] M.A.T. Figueiredo and A.K. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396, 2002.

[8] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7:179–188, 1936.

[9] T. Hastie and R. Tibshirani. Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):155–176, 1996.

[10] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[11] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 6th edition, 2007.

[12] A. M. Martinez and J. Vitria. Learning mixture models using a genetic version of the EM algorithm. *Pattern Recognition Letters*, 21(8):759–769, 2000.

[13] G. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley and Sons, 2000.

[14] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1996.

[15] F. Pernkopf and D. Bouchaffra. Genetic-based EM algorithm for learning Gaussian mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1344–1348, 2005.

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[17] P. Pudil, J. Novovicova, N. Choakjarernwanit, and J. Kittler. Feature selection based on the approximation of class densities by finite mixtures of special type. *Pattern Recognition*, 28(9):1389–1398, 1995.

[18] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[19] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. SMEM algorithm for mixture models. *Neural Computation*, 12(9):2109–2128, 2000.

[20] J. J Verbeek, N. Vlassis, and B. Kröse. Efficient greedy learning of Gaussian mixture models. *Neural Computation*, 15(2):469–485, 2003.

UCZENIE SKOŃCZONYCH MIESZANIN ROZKŁADÓW NORMALNYCH PRZY POMOCY ALGORYTMU EWOLUCJI RÓŻNICOWEJ

Streszczenie W artykule rozważono problem uczenia parametrów skończonej mieszaniny wielowymiarowych rozkładów normalnych. Zaproponowano nową metodę uczenia opartą na algorytmie ewolucji różnicowej. W celu uniknięcia problemów z niedopuszczalnością chromosomów algorytm ewolucji różnicowej wykorzystuje nową reprezentację, w której macierze kowariancji są reprezentowane przy pomocy dekompozycji Cholesky'ego. W eksperymentach wykorzystano trzy wersje algorytmu ewolucji różnicowej różniące się

metodą doboru parametrów. Wyniki eksperymentów, przeprowadzonych na dwóch syntetycznych i jednym rzeczywistym zbiorze danych, wskazują, że zaproponowana metoda jest w stanie poprawnie identyfikować parametry modelu. Metoda ta osiąga również lepsze wyniki niż klasyczyny algorytm EM.

Słowa kluczowe: mieszaniny rozkładów normalnych, ewolucja różnicowa, algorytm EM

Artykuł zrealizowano w ramach pracy statutowej S/WI/2/08