ESTIMATION OF PARAMETERS OF GAUSSIAN MIXTURE MODELS BY A HYBRID METHOD COMBINING A SELF-ADAPTIVE DIFFERENTIAL EVOLUTION WITH THE EM ALGORITHM

Wojciech Kwedlo

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: In the paper the problem of learning of Gaussian mixture models (GMMs) is considered. A new approach based on hybridization of a self-adaptive version of differential evolution (DE) with the classical EM algorithm is described. In this approach, called DE-EM, the EM algorithm is run until convergence to fine-tune each solution obtained by the mutation and crossover operators of DE. To avoid the problem with parameter representation and infeasible solutions we use a method in which the covariance matrices are encoded using their Cholesky factorizations. In a simulation study GMMs were used to cluster synthetic datasets differing by a degree of separation between clusters. The results of experiments indicate that DE-EM outperforms the standard multiple restart expectation-maximization algorithm (MREM). For datasets with high number of features it also outperforms the state-of-the-art random swap EM (RSEM).

Keywords: Gaussian mixture models, differential evolution, expectation maximization, model-based clustering

1. Introduction

Gaussian mixture models (GMMs) [18] are one of the most versatile probability density models, which are used commonly in machine learning and pattern recognition. They are capable of approximating any multimodal distribution. Applications of GMMs include clustering [7] discriminant analysis [9], speaker recognition [22] and texture segmentation [19].

The standard method for maximum likelihood estimation (MLE) of parameters of GMMs is the expectation-maximization (EM) algorithm [21]. It starts from an initial set of mixture parameters and generates a sequence of mixture parameters with

Advances in Computer Science Research, vol. 11, pp. 109-123, 2014.

increasing log likelihood. However, the application of the EM algorithm to GMM parameter learning has several issues. The most important of these is ease of getting trapped in a local maxima of the log likelihood. Consequently the quality of the final solution is strongly dependent on the initial guess of the mixture parameters.

The most common approach proposed to overcome the above problem is to run the EM algorithm many times, starting each run from different random initial conditions, and return the solution with the highest log likelihood. We call this method multiple restart EM (MREM). However this approach lacks effective utilization of available CPU time, as multiple independent EM procedures are likely to exploit similar local maxima.

In [3], an extension of MREM called *emEM* was proposed. The idea of *emEM* involves performing several short EM runs using different random starting points and a lax convergence criterion. The mixture parameters obtained by the best (in the sense of the highest $\log p(X|\Theta)$) short run are used as a starting point for a long EM run. This strategy can be improved by repeating it many times until the available CPU time is exhausted. A variant of *emEM* called *rndEM* [16] reduces the short EM phase to the evaluation of $\log p(X|\Theta)$ of the random starting position.

Researchers investigating the problem of local maxima of the log likelihood have increasingly started to apply population based global optimization algorithms such as genetic algorithms [1,17,20], particle swarm optimization (PSO) [2] or differential evolution [12]. However, a random nature of of search operators employed by these algorithms makes it difficult to represent covariance matrices, because a random modification of individual elements of covariance matrix usually results in a matrix that is not valid (i.e. symmetric and positive definite). Consequently many applications of global optimization algorithms to problem of GMM learning use diagonal (or even spherical) covariance structure.

To avoid the above restriction on covariance structure, the encoding of covariance matrices in candidate solutions must allow for independent modification of individual parameters [2]. Two such encodings have been proposed so far. In [2] covariance matrix of d-dimensional Gaussian distribution was encoded using d eigenvalues and $d(d-1)/2$ Givens rotation angles. In our previous work [12], the covariance matrix was represented by its Cholesky factorization.

The main contribution of this paper, in comparison with our previous works [12], is the inclusion of the EM algorithm into the process of differential evolution (DE). We show, that DE augmented in such way is able to compete with state-of-the-art GMM parameter estimation methods such as random swap EM algorithm (RSEM) [24].

The rest of the paper is organized as follows. Section 2 presents the problem of GMMs parameter estimation. Section 3 describes the EM algorithm, which is the standard method for GMMs learning. Section 4 presents differential evolution algorithm. Section 5 describes a self-adaptation scheme for two key DE parameters. Section 6 presents the application of a hybrid self-adaptive DE to the problem of GMM parameter estimation. Section 7 presents the results of simulation study in which GMMs were used for data clustering. The last section concludes the paper.

2. Background on GMMs

A finite mixture model $p(\mathbf{x},\Theta)$ is defined by a weighted sum of K components:

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

where α_m is m-th mixing proportion and p_m is the probability density function of the m-th component. In (1) θ_m is the set of parameters defining the m-th component and $\Theta = \{\theta_1,\theta_2,\ldots,\theta_K,\alpha_1,\alpha_2,\ldots,\alpha_K\}$ is the complete set of the parameters needed to define the mixture. The mixing proportions $\alpha_m \in (0,1)$ are constrained to sum up to 1. In this work we assume, that the number of components K is known a priori.

In GMMs m-th component follows a multivariate Gaussian distributions with mean vector μ_m and covariance matrix Σ_m. Its probability density function is given by:

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

where $|\cdot|$ denotes a determinant of a matrix, ^{T} denotes transposition of a matrix, and d is the dimension of the feature space. Thus, for a GMM Θ is defined by: $\Theta = \{\mu_1,\Sigma_1,\ldots,\mu_K,\Sigma_K,\alpha_1,\ldots,\alpha_K\}$.

A standard method for learning the parameters of GMMs is the maximum likelihood estimation (MLE). Given a training set of independent and identically distributed feature vectors $X = \{\mathbf{x}^1,\mathbf{x}^2,\ldots,\mathbf{x}^N\}$, where $\mathbf{x}^i = [x_1^i,x_2^i,\ldots,x_d^i] \in R^d$, the log likelihood corresponding to the K-component GMM is given by:

$$\log p(X|\Theta) = \log\prod_{i=1}^{N} p(\mathbf{x}^i|\Theta) = \sum_{i=1}^{N}\log\sum_{m=1}^{K}\alpha_m p_m(\mathbf{x}^i|\theta_m). \tag{3}$$

The maximum likelihood estimate of the parameters is given by:

$$\Theta_{ML} = \underset{\Theta}{\operatorname{argmax}}\{\log p(X|\Theta)\}. \tag{4}$$

It is a well known fact, that a solution of this maximization problem cannot be obtained in a closed form [4]. For that reason a numerical optimization algorithm must be employed to find it.

Model-based clustering [7] is an important application of GMMs. The aim of clustering is to group similar feature vectors together. In this application of GMMs each feature vector is assumed to originate from one K mixture components. We also assume that mixture components are well-separated. The goal of the model-based clustering is to identify, for each feature vector, the mixture component from which it was generated. If we are able to estimate mixture parameters Θ, we can achieve this by allocating e feature vector $\mathbf{x}^i$ to a cluster (mixture component) the highest posterior probability. Using the Bayes theorem this probability for mixture component m can be expressed as:

$$h_m(\mathbf{x}^i) = \frac{\alpha_m p_m(\mathbf{x}^i|\theta_m)}{p(\mathbf{x}^i|\Theta)}. \tag{5}$$

Maximization of (5) is equivalent to finding the mixture index m with the highest value $\alpha_m p_m(\mathbf{x}|\theta_m)$.

3. EM algorithm for GMM learning

The standard method for maximizing (3) is the EM algorithm. It is an iterative algorithm, which, starting from initial guess of a parameters $\Theta^{(0)}$, generates a sequence of estimations $\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta^{(j)}, \ldots$, with increasing log likelihood (i.e., $\log p(X|\Theta^{(j)}) > \log p(X|\Theta^{(j-1)})$. Each iteration j of the algorithm consists of two steps called expectation step (E-step) and maximization step (M-step) followed by a convergence check. For the GMMs these steps are defined as follows [21]:

1. E-step: Given the set of mixture parameters $\Theta^{(j-1)}$ from the previous iteration, for each $m = 1, \ldots, K$ and $i = 1, \ldots, N$, the posterior probability that a feature vector $\mathbf{x}^i$ was generated from mth component is computed as:

$$h_m^{(j)}(\mathbf{x}^i) = \frac{\alpha_m^{(j)} p_m(\mathbf{x}^i|\theta_m^{(j-1)})}{\sum_{k=1}^{K} \alpha_k^{(j)} p_k(\mathbf{x}^i|\theta_k^{(j-1)})}, \tag{6}$$

   where $\theta_m^{(j-1)}$ and $\theta_k^{(j-1)}$ denote parameters of components m and k, in the iteration $j-1$, respectively.
2. M-step: Given the posterior probabilities $h_m^{(j)}(\mathbf{x}^i)$ obtained in the E-step the set of parameters $\Theta^{(j)}$ is calculated as:

$$\alpha_m^{(j)} = \frac{1}{N} \sum_{i=1}^{N} h_m^{(j)}(\mathbf{x}^i) \tag{7}$$

$$\mu_m^{(j)} = \frac{\sum_{i=1}^{N} h_m^{(j)}(\mathbf{x}^i) * \mathbf{x}^i}{\sum_{i=1}^{N} h_m^{(j)}(\mathbf{x}^i)} \tag{8}$$

$$\Sigma_m^{(j)} = \frac{\sum_{i=1}^{N} h_m^{(j)}(\mathbf{x}^i)(\mathbf{x}^i - \mu_m^{(j)})(\mathbf{x}^i - \mu_m^{(j)})^T}{\sum_{i=1}^{N} h_m^{(j)}(\mathbf{x}^i)} \tag{9}$$

3. Convergence check: The log likelihood $\log p(X|\Theta^{(j)})$ is computed according (3). The algorithm is terminated if the following convergence criterion is met.

$$\frac{\log p(X|\Theta^{(j)}) - \log p(X|\Theta^{(j-1)})}{\log p(X|\Theta^{(j)})} < \varepsilon, \tag{10}$$

where $\varepsilon \ll 1$ is a user defined termination threshold. If the convergence criterion is not met algorithm proceeds to Step 1.

The above algorithm is easy to implement. However it has one important drawback. It is highly sensitive to initialization and easily gets trapped in a local maximum of the log likelihood function. For that reason the quality of the final solution is strongly dependent on the initial guess of the mixture parameters $\Theta^{(0)}$. The problem can be to some degree alleviated by performing multiple runs of the algorithm, each of them starting from different random initial conditions, and returning the result with the highest $\log p(X|\Theta)$. We call this approach multiple restart EM (MREM).

4. Differential evolution

Differential evolution, proposed in [23], is an evolutionary algorithm, which in each generation maintains a population of S solutions to optimization problem. In this section the most common variant with rand/1/ mutation and binomial crossover is described.

Let $u_{i,G}$ denote the i-th member ($i = 1, \ldots, S$) of the population in the G-th iteration. It is assumed that $u_{i,G}$ Is a D-dimensional real-valued vectors (i.e., $u_{i,G} \in \Re^D$).

At the start of the algorithm all population members are initialized randomly. Each generation G consists of three steps. Two of them are mutation and crossover, which for each population element $u_{i,G}$ create a trial solution $y_{i,G}$. The mutation and crossover are followed by a selection, in which fitness of population member $u_{i,G}$ is compared to fitness of the trial solution $y_{i,G}$. The solution with the better (i.e., higher in our application) fitness survives into the next generation:

$$y_{i,G+1} = \begin{cases} y_{i,G} & \text{if} f(y_{i,G}) > f(y_{i,G}) \\ u_{i,G} & \text{otherwise} \end{cases}, \tag{11}$$

where $f : \Re^D \rightarrow \Re$ is the fitness function.

The mutation operator of DE generates a creates a mutant vector $v'_{i,G}$ according to the equation:

$$v'_{i,G} = u_{a,G} + F * (u_{b,G} - u_{c,G}), \tag{12}$$

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 $v_{i,G}$ 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:

$$y_{ji,G} = \begin{cases} v_{ji,G} & \text{if } rnd(j) < CR \text{ or } j = e \\ u_{ji,G} & \text{otherwise} \end{cases}. \tag{13}$$

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 $y'_{i,G}$.

5. Self adaptation of DE control parameters

Experimental studies have shown, that the choice of control parameters F and CR has a significant impact on the performance of DE. In the first experiments with DE [23] the parameters were fixed during the run of the algorithm. Later, some methods for parameter control [6], which change the parameters during the run, were developed. Among these, the approach called self-adaptive parameter control attracted many researchers. In this approach the parameters are encoded into individuals and undergo evolution. The better values of the parameters result in better individuals which are more likely to reproduce and produce offspring and, thus, disseminate these better parameters.

In our DE-EM method, a self-adaptation scheme proposed by Brest et al. [5] was used. It works as follows. Each population element and each trial vector 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:

$$F^y_{i,G} = \begin{cases} L + rnd_2 * U & \text{if } rnd_1 < \tau_1 \\ F^u_{i,G} & \text{otherwise} \end{cases}. \tag{14}$$

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:

$$CR^y_{i,G} = \begin{cases} rnd_3 & \text{if } rnd_4 < \tau_2 \\ CR^u_{i,G} & \text{otherwise} \end{cases}, \tag{15}$$

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

It may seem that self-adaptation of F and CR introduces another four parameters (L, U, τ_1, τ_2) which require a costly fine-tuning using the trial-and-error approach. However, Brest et al. [5] used fixed values of these parameters obtaining very good results for a very diverse range of benchmark numerical optimization problems. Following their advice in our experiments we set $\tau_1 = \tau_2 = 0.1$. L and U were set to 0.05 and 0.35 respectively, which ensured that $F^y_{i,G} \in [0.05, 0.4]$.

6. Application of hybrid self-adaptive DE to the problem of GMM learning

6.1 Representation of GMM parameters

Since DE represents the problem solutions as real-valued vectors the encoding of mixing proportions and mean vectors is very straightforward: they simply are stored in solution vectors using the floating point representation. Unfortunately, this method cannot be use in case of covariance matrices. A covariance matrix of Σ of d-dimensional Gaussian distribution is symmetric, and thus has $d(d+1)/2$ free parameters. However, if a distribution is non-degenerate, this matrix must be positive definite i.e., for each non-zero $\mathbf{x} \in \Re^d$ $\mathbf{x}^T \Sigma \mathbf{x} > 0$ [11]. For that reason it is impossible to store these parameters directly, because matrices obtained by a random operators of crossover and mutation would violate the positive-definiteness constraint [2].

To overcome this obstacle DE-EM uses the representation of covariance matrices, first proposed in [12], based on their Cholesky factorization. Each positive-definite matrix Σ can be decomposed as a product of a lower triangular matrix L with positive diagonal elements and its transpose [8]:

$$\Sigma = LL^T. \tag{16}$$

The Cholesky factorization of a positive-definite matrix is unique [8]. The matrix L is called the Cholesky factor of Σ or the square root of Σ.

In the DE-EM method the covariance matrices of a GMM are represented in solution vectors of DE by their Cholesky factors. The constraints on Cholesky factors (the diagonal elements must be positive) are easily handled by DE, because each constraint on solution handled independently from the other constraints.

6.2 Fitness function

The fitness function used by DE-EM is $\log p(X|\Theta)$. The selection method is configured to maximize the fitness.

6.3 Hybridization with the EM algorithm

Before the selection step of DE, each candidate solution is fine-tuned by the EM algorithm. First, the solution is used to initialize the EM. Next, the EM algorithm is run until the convergence criterion (10) is met. Then, the solution obtained by the EM algorithm is used in the selection step.

Similar fine-tuning by the EM algorithm is performed on random initial solutions in generation 0.

7. Experimental results

In this section the results of the computational experiments on synthetic datasets, in which the GMMs were used for model-based clustering, are reported. We compared our DE-EM method to two other approaches: the standard multiple restart EM (MREM) and recently proposed [24] random swap EM (RSEM), which is state-of-the-art method for GMM parameters estimation, capable of escaping from local maxima of log likelihood.

The algorithms were implemented in C++ language and compiled by the Intel C++ compiler version 14.0.1 using optimizing options (-O3 -ipo -march=core2 -fno-alias). The compiled programs were run on a Dell Poweredge 1950 server with two quad-core Intel Xeon 5355 (2.66 GHz) processors and 16 GB of RAM, running Ubuntu Linux 12.04. The implementation of EM was parallelized [14] using

OpenMP standard for shared memory computers, taking advantage of all eight cores of the system.

In the experiments we used a generator proposed by [15], which generates randomly Gaussian clusters according to the user-defined overlap characteristic. The overlap ω_{ij} between two clusters i and j is defined as the sum of two misclassification probabilities $\omega_{j|i}$ and $\omega_{i|j}$, where: $\omega_{j|i} = Pr[\alpha_i p(\mathbf{x}|\mu_i,\Sigma_i) < \alpha_j p(\mathbf{x}|\mu_j,\Sigma_j)|\mathbf{x} \sim \mathcal{N}(\mu_i,\Sigma_i)]$, and similarly $\omega_{i|j} = Pr[\alpha_j p(\mathbf{x}|\mu_j,\Sigma_j) < \alpha_i p(\mathbf{x}|\mu_i,\Sigma_i)|\mathbf{x} \sim \mathcal{N}(\mu_j,\Sigma_j)]$.

The overlap characteristic of the generator [15] was controlled by one parameter ω expressing the average pairwise overlap between clusters. In our experiments the number of components K was fixed at 20. Figure 1 shows example two-dimensional training sets simulated from mixtures obtained from the generator for different values of ω. It can be seen that by using different values of ω we can control the separation of clusters.

In our experiments we generated mixtures with dimension $d \in \{5,10,25\}$. For each dimension we used $\omega \in \{0.0001, 0.0002, 0.0005, 0.001, 0.0025, 0.005, 0.01, 0.0250, 0.05, 0.1\}$. We used the adjusted Rand index (ARI) [10] to measure the degree of agreement between partitions of data discovered by the clustering algorithms and the original partitions (we knew them because we used synthetic datasets drawn by a random generator, which allowed us to track the source of each feature vector). The ARI is bounded between -1 and 1. The expected value of ARI in case of randomly generated partitions is 0. A higher value of ARI indicates a higher similarity between partitions; a maximum value of 1 means, that two partitions are identical. A similar experimental setting was used for comparison of different EM initialization methods in [13].

The feature vectors were clustered according to the MAP rule, as described in Section 2. Since in this experiment the original (ground truth) mixture parameters were available, we also performed clustering using them.

The experimental protocol was as follows. For every combination of d and ω 50 different random mixtures were generated. For each mixture a single dataset was realized. For $d = 5$ and $d = 10$ the number of feature vectors in dataset was set to 6000. For $d = 25$ we had to increase this number to 30000 to avoid issues with the singularity of covariance matrices. To assure a fair comparison, each of three algorithms was allocated equal CPU time.

Figure 2a shows the obtained values of ARI, (averaged over 50 different mixtures) when clustering was performed on the basis of the ground truth parameters. As expected, whereas for clusters with very small overlap ARI close to 1 (indicating very good agreement between original partitions and clustering results) could be obtained, an increase of overlap between clusters led to lower values of ARI.

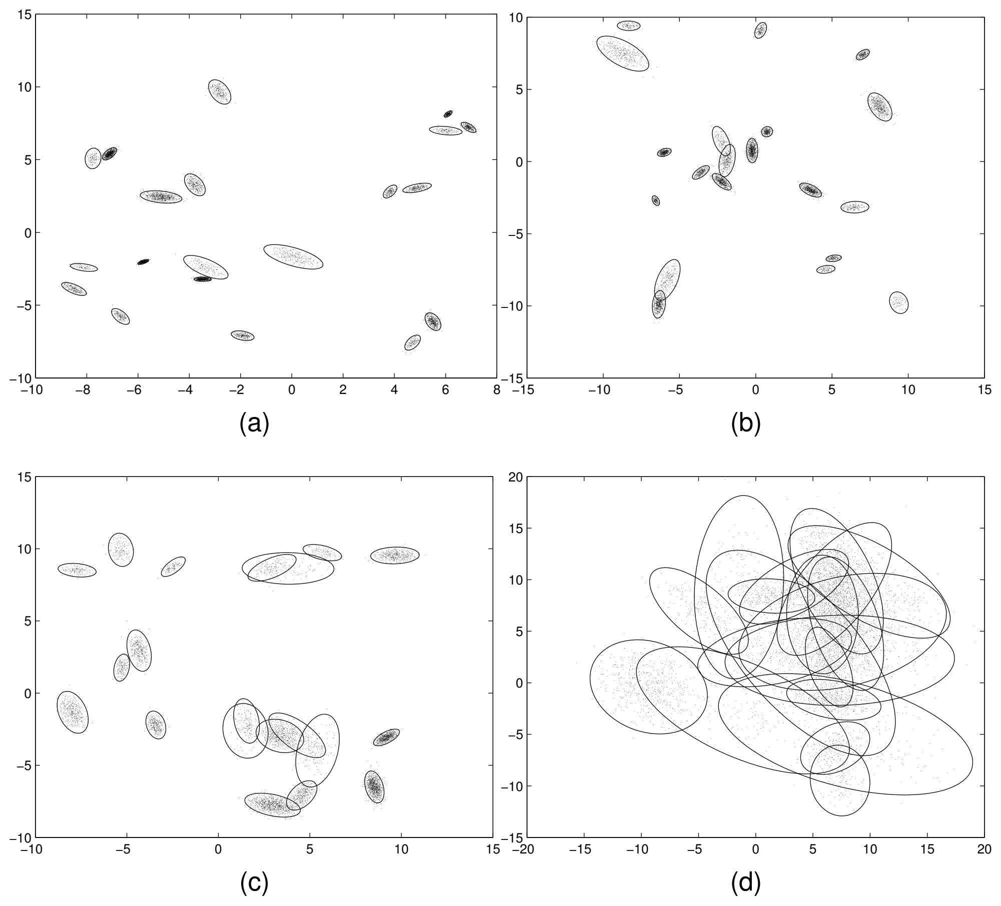

Fig. 1. Two-dimensional training sets simulated from 20–component mixtures with (a) $\omega = 0.0001$, (b) $\omega = 0.001$, (c) $\omega = 0.01$, (d) $\omega = 0.1$. The ellipses are centered around component means and represent 95% confidence regions.

The average ARI values obtained for ground truth mixture parameters were used as the baseline for comparison of three GMM parameter estimation methods. The results concerning these methods are shown on Figures 2b, 2c, 2d. The result of each method is shown as a % error relative ground truth mixture parameters. The % error of the method A is computed as $(\mathrm{ARI}_T - \mathrm{ARI}_A)/\mathrm{ARI}_\mathrm{T} * 100$, where ARI_T is the average (over 50 different mixtures) ARI obtained using the ground truth mixture parameters and ARI_A is average ARI obtained using mixture parameters estimated by the method A. A lower value of % error indicates a better performance, values close to 0 indicate that clustering using a given GMM parameter estimation method achieves similar results as clustering using the ground truth parameters. The results

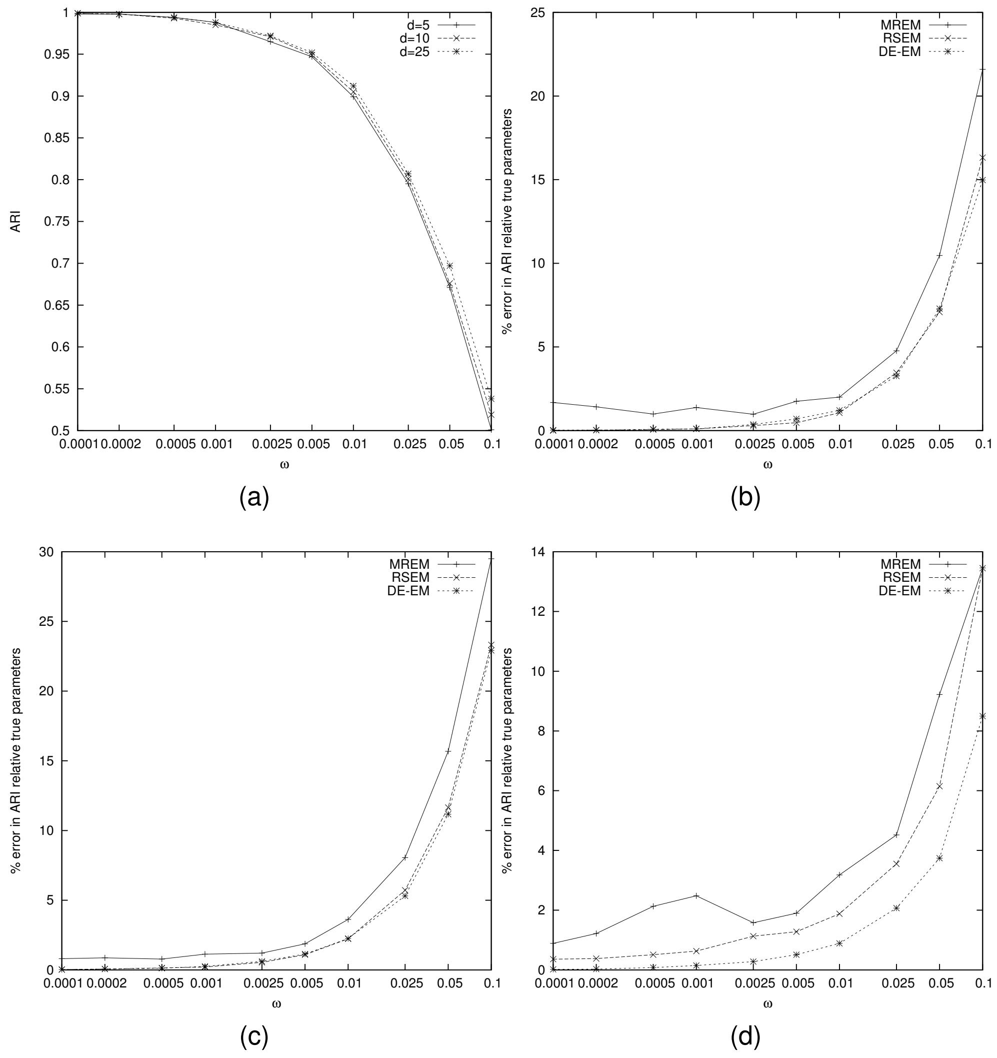

Fig. 2. (a) The average ARI values obtained for clustering using true mixture parameters. % Error in ARI relative true mixture parameters for (b) $d = 5$, (c) $d = 10$, (d) $d = 25$.

from Figures 2b – 2d are summarized by Table 1, which shows the results averaged over 10 different values of ω separately for each dimension d. The last row of the table shows the total average result for each of compared methods.

Table 1. The average error in ARI relative known mixture parameters

d	MREM	RSEM	DE-EM
5	4.70	2.89	2.81
10	6.35	4.50	4.39
25	4.06	2.93	1.63
Total	5.04	3.44	2.94

The results achieved by the three algorithms indicate that:

- The difference between the results obtained by clustering using each of three estimation methods and clustering using the ground truth parameters widens as the average overlap between clusters is increased.
- MREM is the worst estimation method irrespectively from the dimension d of feature space.
- For $d = 5$ and $d = 10$ the results of the DE-EM method are on par with the RSEM approach. However, our method clearly outperforms RSEM for $d = 25$.

8. Conclusions

In this paper a new method for GMMs learning which combines the self-adaptive differential evolution with the EM algorithm was proposed. To avoid the problem with infeasibility of solutions we used an representation, in which covariance matrices were encoded using their Cholesky factorization.

The results of our study allow us to recommend DE-EM method over the MREM and RSEM algorithms in application of GMMs to clustering problems. Although there was little difference between DEEM and RSEM in experiments where $d = 5$ and $d = 10$ our method was clear winner for more difficult problems where $d = 25$.

In future works we are going to compare the performance of the DE-EM method to other well established hybrid evolutionary algorithms, for instance the GA-EM algorithm [20]. We also plan to use other encodings of covariance matrices, especially based on Givens angles [2]. Finally, we are going to to test the performance of DE-EM method in other applications of GMMs, for instance in discriminant analysis [9].

Acknowledgments

This work was supported by the grant S/WI/2/2013 from Bialystok University of Technology.

References

[1] J. L. Andrews and P. D. McNicholas. Using evolutionary algorithms for model-based clustering. *Pattern Recognit. Lett.*, 34(9):987–992, 2013.

[2] C. Ari, S. Aksoy, and O. Arikan. Maximum likelihood estimation of Gaussian mixture models using stochastic search. *Pattern Recognit.*, 45(7):2804–2816, 2012.

[3] Christophe Biernacki, Gilles Celeux, and Gérard Govaert. Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models. *Comput. Stat. Data Anal.*, 41(3):561–575, 2003.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[5] 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.

[6] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. Evol. Comput.*, 3(2):124–141, 1999.

[7] C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis, and density estimation. *J. Am. Stat. Assoc.*, 97(458):611–631, 2002.

[8] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins, Baltimore, MD, 1996.

[9] T. Hastie and R. Tibshirani. Discriminant analysis by Gaussian mixtures. *J. Royal Stat. Soc. Ser. B*, 58(1):155–176, 1996.

[10] L. Hubert and P. Arabie. Comparing partitions. *J. Classif.*, 2(1):193–218, 1985.

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

[12] W. Kwedlo. Learning finite Gaussian mixtures using differential evolution. *Zeszyty Naukowe Politechniki Białostockiej. Informatyka*, 5:19–33, 2010.

[13] W. Kwedlo. A new method for random initialization of the EM algorithm for multivariate Gaussian mixture learning. In *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*, pages 81–90. Springer, 2013.

[14] W. Kwedlo. A parallel EM algorithm for Gaussian mixture models implemented on a NUMA system using OpenMP. In *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing PDP 2014*, pages 292–298. IEEE CPS, 2014.

[15] R. Maitra and V. Melnykov. Simulating data to study performance of finite mixture modeling and clustering algorithms. *J. Comput. Graph. Stat.*, 19(2):354–376, 2010.

[16] Ranjan Maitra. Initializing partition-optimization algorithms. *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, 6(1):144–157, 2009.

[17] 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.

[18] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York, 2000.

[19] H. Permuter, J. Francos, and I. Jermyn. A study of Gaussian mixture models of color and texture features for image classification and segmentation. *Pattern Recognit.*, 39(4):695–706, 2006.

[20] F. Pernkopf and D. Bouchaffra. Genetic-based EM algorithm for learning Gaussian mixture models. *IEEE Trans. Pattern Analysis Mach. Intell.*, 27(8):1344–1348, 2005.

[21] R. A. Redner and H. F. Walker. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Rev.*, 26(2):195–239, 1984.

[22] D.A. Reynolds, T.F. Quatieri, and R.B. Dunn. Speaker verification using adapted Gaussian mixture models. *Digit. Signal Process.*, 10(1):19–41, 2000.

[23] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.*, 11(4):341–359, 1997.

[24] Q. Zhao, V. Hautamäki, I. Kärkkäinen, and P. Fränti. Random swap EM algorithm for Gaussian mixture models. *Pattern Recognit. Lett.*, 33(16):2120–2126, 2012.

ESTYMACJA PARAMETRÓW MODELI MIESZANIN ROZKŁADÓW NORMALNYCH PRZY POMOCY METODY HYBRYDOWEJ ŁĄCZĄCEJ SAMOADPTACYJNĄ EWOLUCJĘ RÓŻNICOWĄ Z ALGORYTMEM EM

Streszczenie: W pracy poruszono problem uczenia modeli mieszanin rozkładów normalnych. Zaproponowano nowe podejście, nazwane DE-EM, oparte na hybrydyzacji samodaptacyjnego algorytmu ewolucji różnicowej i klasycznego algorytmu EM. W nowej metodzie rozwiązanie otrzymane jako wynik operatorów mutacji i krzyżowania jest poddawane optymalizacji lokalnej, prowadzonej aż do momentu uzyskania zbieżności, przez algorytm EM. Aby uniknąć problemu z reprezentacją macierzy kowariancji i niedopuszczalnością rozwiązań użyto metody, w której macierze kowariancji są kodowane przy pomocy dekompozycji Cholesky'ego. W badaniach symulacyjnych modele mieszanin rozkładów normalnych zastosowano do grupowania danych syntetycznych. Wyniki eksperymentów wskazują, że metoda DE-EM osiąga lepsze wyniki niż standardowa technika wielokrotnego startu algorytmu EM. Dla zbiorów danych z dużą liczbą cech, metoda osiąga lepsze wyniki niż technika losowej wymiany rozwiązań połączona z algorytmem EM.

Słowa kluczowe: Mieszniny rozkładów normalnych, ewolucja różnicowa, algorytm EM, grupowanie danych

Artykuł zrealizowano w ramach pracy badawczej S/WI/2/2013.