A PRACTICAL METHOD FOR SELF-ADAPTING GAUSSIAN EXPECTATION MAXIMIZATION

Nicola Greggio [⊘,‡], Alexandre Bernardino [‡] and José Santos-Victor [‡]
[⊘] *ARTS Lab - Scuola Superiore S. Anna, Polo S. Anna Valdera, Viale R. Piaggio, 34, 56025 Pontedera, Italy*
[‡] *Instituto de Sistemas e Robótica, Instituto Superior Técnico, 1049-001 Lisboa, Portugal*

Keywords: Unsupervised learning, Self-adapting gaussian mixture, Expectation maximization, Machine learning, Clustering.

Abstract: Split-and-merge techniques have been demonstrated to be effective in overtaking the convergence problems in classical EM. In this paper we follow a split-and-merge approach and we propose a new EM algorithm that makes use of a on-line variable number of mixture Gaussians components. We introduce a measure of the similarities to decide when to merge components. A set of adaptive thresholds keeps the number of mixture components close to optimal values. For sake of computational burden, our algorithm starts with a low initial number of Gaussians, adjusting it in runtime, if necessary. We show the effectivity of the method in a series of simulated experiments. Additionally, we illustrate the convergence rates of of the proposed algorithms with respect to the classical EM.

1 INTRODUCTION

UNSUPERVISED clustering classifies different data into classes based on redundancy contained within the data sample. The classes, also called *clusters*, are detected automatically, i.e. without any input by an external agent. This method finds applications in many fields, such as in image processing (e.g. segmentation of different object from the background), sound analysis (e.g. word perceptions), segmentation of multivariate medical images, and many others. The techniques for unsupervised learning range from Kohonen maps, Growing Neural gas, k-means, to Independent component analysis, Adaptive resonance theory, etc. Particularly interesting is the Expectation Maximization algorithm applied to Gaussian mixtures which allows to model complex probability distribution functions. Fitting a mixture model to the distribution of the data is equivalent, in some applications, to the identification of the clusters with the mixture components (McLachlan and Peel, 2000).

Expectation-Maximization (EM) algorithm is the standard approach for learning the parameters of the mixture model (Dempster et al., 1977). It is demonstrated that it always converges to a local optimum. However, it also presents some drawbacks. For instance, EM requires an *a-priori* selection of model order, namely, the number of components (M) to be incorporated into the model, and its results depend on initialization. The more Gaussians there are within the mixture, the higher will be the total log-likelihood, and more precise the estimation. Unfortunately, increasing the number of Gaussians will lead to overfitting the data and it increases the computational burden. Therefore, finding the best compromise between precision, generalization and speed is a must. A common approach to address this compromise is trying different configurations before determining the optimal solution, e.g. by applying the algorithm for a different number of components, and selecting the best model according to appropriate criteria.

1.1 Related Work

Different approaches can be used to select the best number of components. These can be divided into two main classes: *off-line* and *on-line* techniques.

The first ones evaluate the best model by executing independent runs of the EM algorithm for many different initializations, and evaluating each estimate with criteria that penalize complex models (e.g. the Akaike Information Criterion (AIC) (Sakimoto et al., 1986) and the Rissanen Minimum Description Length (MDL) (Rissanen, 1989)). These, in order to be effective, have to be evaluated for every possible number of models under comparison. Therefore, it is clear that,

Greggio N., Bernardino A. and Santos-Victor J. (2010).
A PRACTICAL METHOD FOR SELF-ADAPTING GAUSSIAN EXPECTATION MAXIMIZATION.
In *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics*, pages 36-44
DOI: 10.5220/0002894600360044

for having an sufficiently exhaustive search the complexity goes with the number of tested models, and the model parameters.

The second ones start with a fix set of models and sequentially adjust their configuration (including the number of components) based on different evaluation criteria. Figueiredo and Jain proposed a method that starts with a high number of mixture parameters, merging them step by step until convergence (Figueiredo and Jain, 2002). This method can be applied to any parametric mixture where the EM algorithm can be used. Pernkopf and Bouchaffra proposed a Genetic-Based EM Algorithm capable of learning Gaussian mixture models (Pernkopf and Bouchaffra, 2005). They first selected the number of components by means of the minimum description length (MDL) criterion. A combination of genetic algorithms with the EM has been explored.

Ueda *et Al.* proposed a split-and-merge EM algorithm to alleviate the problem of local convergence of the EM method (Ueda et al., 2000). Subsequently, Zhang *et Al.* introduced another split-and-merge technique (Zhang et al., 2003). The split-and-merge equations show that the merge operation is a well-posed problem, whereas the split operation is ill-posed. Two methods for solving this problem are developed through singular value decomposition and Cholesky decomposition and then a new modified EM algorithm is constructed. They demonstrated the validity of split-and-merge approach in model selection, in terms of convergence properties. Moreover, the merge an split criterion is efficient in reducing number of model hypothesis, and it is often more efficient than exhaustive, random or genetic algorithm approaches.

1.2 Our Contribution

In this paper, we propose an algorithm for computing the number of components as well as the parameters of the mixture model. Similarly to other split-and-merge methods, our technique uses a local parameter search, that reuses the information acquired on previous steps, being suitable to problems with slowly changing distributions or to adapt the parameters when new samples are added or removed. The algorithm starts with a fixed number of Gaussians, and automatically decides whether increasing or reducing it. The key feature of our technique is the decision of when add or merge a Gaussian. In order to accomplish this at best we introduce a new concept, the dissimilarity index between two Gaussian distributions. Moreover, in order to evade local optimal solutions we make use of self-adaptative thresholds for deciding when Gaussians are split or merged. Our algorithm starts with high thresholds levels, preventing large changes in the number of Gaussian components at the beginning. It also starts with a low initial number of Gaussians, which can be increased during computation, if necessary. The time evolution of the threshold values allows periods of stability in the number of components so that they can freely adapt to the input data. After this period of stability these thresholds become more sensitive, promoting the escape from local optimum solutions by perturbing the system configuration when necessary, until a stopping criterion has reached. This makes our algorithm results less sensitive to initialization. The algorithm is presented for Gaussian mixture models.

1.3 Outline

The paper is organized as follows. In sec. 2 we describe the notation and formulate the classical Expectation Maximization algorithm. In sec. 3 we introduce the proposed algorithm. Specifically, we describe the insertion of a new Gaussian in sec. 3.3, its merging in sec. 3.4, the initializations in sec. 3.2, and the decision thresholds update rules in sec. 3.5. Furthermore, in sec. 4 we describe our experimental set-up for testing the validity of our new technique and the results. Finally, in sec. 5 we conclude and propose directions for future work.

2 EXPECTATION MAXIMIZATION ALGORITHM

2.1 EM Algorithm: The Original Formulation

A common usage of the EM algorithm is to identify the *"incomplete, or unobserved data"* $\bar{y} = (\bar{y}^1, \bar{y}^2, \ldots, \bar{y}^k)$ given the couple $(\bar{x}, \bar{y})$ - with $\bar{x}$ defined as $\bar{x} = \{\bar{x}^1, \bar{x}^2, \ldots, \bar{x}^k\}$, also called *"complete data"*, which has a probability density (or joint distribution) $p(\bar{x}, \bar{y}|\bar{\vartheta}) = p_{\bar{\vartheta}}(\bar{x}, \bar{y})$ depending on the parameter $\bar{\vartheta}$. We define $E^{'}(\cdot)$ the expected value of a random variable, computed with respect to the density $p_{\bar{\vartheta}}(\bar{x}, \bar{y})$.

We define $Q(\bar{\vartheta}^{(n)}, \bar{\vartheta}^{(n-1)}) = E^{'} L(\bar{\vartheta})$, with $L(\bar{\vartheta})$ being the log-likelihood of the observed data:

$$L(\bar{\vartheta}) = \log p_{\bar{\vartheta}}(\bar{x}, \bar{y}) \tag{1}$$

The EM procedure repeats the two following steps until convergence, iteratively:

- E-step: It computes the expectation of the joint probability density:

$$Q(\bar{\vartheta}^{(n)}, \bar{\vartheta}^{(n-1)}) = E^{'}[\log p(\bar{x}, \bar{y}|\bar{\vartheta}^{(n-1)})] \quad (2)$$

- M-step: It evaluates the new parameters that maximize Q:

$$\bar{\vartheta}^{(n+1)} = arg\max_{\bar{\vartheta}} Q(\bar{\vartheta}^{n}, \bar{\vartheta}^{(n-1)}) \quad (3)$$

The convergence to a local maxima is guaranteed. However, the obtained parameter estimates, and therefore, the accuracy of the method greatly depend on the initial parameters $\hat{\vartheta}^0$.

2.2 EM Algorithm: Application to a Gaussians Mixture

When applied to a Gaussian mixture density we assume the following model:

$$\begin{aligned} p(\bar{x}) &= \sum_{i=1}^{nc} w_i \cdot p_i(\bar{x}) \\ p_c(\bar{x}) &= \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_c|^{\frac{1}{2}}} e^{-\frac{1}{2}(\bar{x}-\bar{\mu}_c)^T|\Sigma_c|^{-1}(\bar{x}-\bar{\mu}_c)} \end{aligned} \quad (4)$$

where $p_c(\mathcal{X})$ is the component prior distribution for the class c, and with d, $\bar{\mu}_c$ and Σ_c being the input dimension, the mean and covariance matrix of the Gaussians component c, and nc the total number of components, respectively.

Consider that we have nc classes C_{nc}, with $p(\bar{x}|C_c)$ and $P(C_c) = w_c$ being the density and the *a-priori* probability of the data of the class C_c, respectively. Then the E and M steps become, respectively:

E-step:

$$\begin{aligned} P(y_c^k = 1|x^k) &= P(C_c|x^k) = E^{'}(y_c|x^k) \\ &= \frac{p(x^k|C_c)\cdot P(C_c)}{p(x^k)} = \frac{w_c \cdot p_c(x^k)}{\sum_{c=1}^{nc} w_c \cdot p_c(x^k)} \end{aligned} \quad (5)$$

For simplicity of notation we refer to $E^{'}(y_c|x^k)$ as π_c^k. This is probability that the x^k belongs to the class C_c.

M-step:

$$\begin{aligned} \bar{\mu}_c^{(n+1)} &= \frac{\sum_{i=1}^{k} \pi_c^k \bar{x}^i}{\sum_{i=1}^{k} \pi_c^i} \\ \Sigma_c^{(n+1)} &= \frac{\sum_{i=1}^{k} \pi_c^k (\bar{x}^i - \bar{\mu}_c^{(n)})(\bar{x}^i - \bar{\mu}_c^{(n)})^T}{\sum_{i=1}^{k} \pi_c^i} \end{aligned} \quad (6)$$

Finally, *a-priori* probabilities of the classes, i.e. the probability that the data belongs to the class c, are reestimated as:

$$w_c^{(n+1)} = \frac{1}{K}\sum_{i=1}^{k} \pi_c^i, \quad with \;\; c = \{1, 2, \ldots, nc\} \quad (7)$$

3 SAGEM: SELF-ADAPTING GAUSSIAN EXPECTATION MAXIMIZATION ALGORITHM

The key issue of our algorithm is looking whether one or more Gaussians are not increasing their own likelihood during optimization. In other words, if they are not participating in the optimization process, they will be split into two new Gaussians or merged. We will introduce two new concepts related to the state of a Gaussian component:

- The *Area* of a Gaussian, that measures how much a Gaussian component covers the data space; This actually correspond to the covariance matrix determinant;
- Its age, that measures how long the component's own likelihood does not increase significantly (see sec. 3.1);

Then, the merge and split processes are controlled by the following adaptive thresholds:

- One adaptive threshold L_{TH} for determining a significant increase in likelihood (see sec. 3.5);
- One adaptive threshold A_{TH} for triggering the merge or split process based on the component's own age (see sec. 3.5);
- One adaptive threshold M_{TH} for deciding to merge two Gaussians based on their area and position (see sec. 3.4);
- One adaptive threshold S_{TH} for deciding to split a Gaussian based on its area (see sec. 3.3).

It is worth noticing that even though we consider four thresholds to tune, all of them are adaptive, and only require a coarse initialization. All these thresholds cooperate in keeping the number of components stable for a period after a split or merge operation (see sec. 3.5).

These parameters will be fully detailed within the next sections.

3.1 SAGEM Formulation

Our formulation can be summarized within four steps:

- Initializing the parameters;
- Adding a Gaussian;
- Merging two Gaussians;
- Updating decision thresholds.

Each Gaussian element i of the mixture is represented as follows:

$$\bar{\vartheta}_i = \rho(w_i, \bar{\mu}_i, \Sigma_i, \xi_i, \Lambda_{last(i)}, \Lambda_{curr(i)}, a_i) \quad (8)$$

where w_i is the *a-priori* probabilities of the class, $\bar{\mu}_i$ is its mean, Σ_i is its the covariance matrix, ξ_i its *area*, $\Lambda_{last(i)}$ and $\Lambda_{curr(i)}$ are its last and its current log-likelihood value, and a_i its *age*. Here, we define two new elements, the area and the age of the Gaussian, which will be described later.

During each iteration, the algorithm keeps memory of the previous likelihood. Once the re-estimation of the vector parameter $\bar{\vartheta}$ has been computed in the EM step, our algorithm evaluates the current log-likelihood of each single Gaussian as:

$$\Lambda_{curr(i)}(\vartheta) = \sum_{j=0}^{k-1} log(w_i \cdot p_i(\bar{x}^j)) \tag{9}$$

Then, for each Gaussian the difference between the current and last log-likelihood value is compared with a increment threshold, equal for all the Gaussians, L_{TH}. If the difference is smaller than the threshold L_{TH}, i.e. there is no significant increment, the algorithm increases the *age* a_i of the relative Gaussian. If a_i overcomes the age threshold A_{TH} (i.e. the Gaussian i does not increase its own likelihood for a predetermined number of times significally), the algorithm decides whether to split this Gaussian or merging it with existing ones.

Then, after a certain number of iterations the algorithm will stop. The whole algorithm pseudocode is shown in Fig. 3.1.

3.2 Split Threshold Initialization

At the beginning, S_{TH} will be automatically initialized to the area of the covariance of all the data set, relative to the total mean. The other decision thresholds will be initialized as follows:

$$\begin{aligned} M_{TH-INIT} &= 0 \\ L_{INIT} &= k_{LTH} \\ Age_{INIT} &= k_{ATH} \end{aligned} \tag{10}$$

with k_{LTH} and k_{ATH} (namely, the minimum amount of likelihood difference between two iterations and the number of iterations required for taking into account the lack of a likelihood consistent variation) relatively low (i.e. both in the order of 10, or 20). Of course, higher values for k_{LTH} and smaller for k_{ATH} give rise to a faster adaptation, however adding instabilities.

3.3 Adding a Gaussian

If the covariance matrix area of the examined Gaussian at each stage overcomes the maximum area threshold S_{TH}, then another Gaussian is added to the mixture.

Algorithm 3.1. Pseudocode.

```
- Parameter initialization;
while (stopping criterion is not met) do
    Λ_i evaluation, for i = 0,1,...,c;
    L(ϑ) evaluation (1);
    re-estimate priors w_i, for i = 0,1,...,c;
    recompute center μ̄_i^(n+1) and covariances Σ_i^(n+1), for
    i = 0,1,...,c (5);
    - Evaluation whether changing the Gaussian
    distribution structure -
    for (i = 0 to c) do
      if (a_i > A_TH) then
        if ((Λ_curr(i) − Λ_last(i)) < L_TH) then
          a_i+ = 1;
          - General condition for changing satisfied;
          now checking those for each Gaussians -
          if (Σ_i > S_TH) then
            if (c < maxNumGaussians) then
              add Gaussian → split;
              c+ = 1;
              reset S_TH to its initial value;
              reset L_TH to its initial value;
              return;
            end if
          end if
          for (j = 0 to c) do
            eval χ_i,j (14)
            if (χ_i,j < M_TH) then
              merge Gaussian → merge;
              c− = 1;
              reset M_TH to its initial value;
              reset L_TH to its initial value;
              return;
            end if
          end for
          S_TH = S_TH · (1 + α · ξ);
          M_TH = M_TH · (1 + γ · χ_A,B);
        end if
      end if
    end for
end while
```

More precisely, the original Gaussian with parameters $\bar{\vartheta}_{old}$ will be split within other two ones. The new means, A and B, will be:

$$\begin{aligned} &\bar{\mu}_A = \bar{\mu}_{old} + \frac{1}{2}(\Sigma_{i=j})^{\frac{1}{2}}; \quad \bar{\mu}_B = \bar{\mu}_{old} - \frac{1}{2}(\Sigma_{i=j})^{\frac{1}{2}} \\ &i,j = \{1,2,\ldots,d\} \end{aligned} \tag{11}$$

where d is the input dimension.

The covariance matrixes will be updated as:

$$\begin{aligned} &\Sigma_{A(i,j)} = \Sigma_{old(i,j)}; \quad \Sigma_{B(i,j)} = \Sigma_{old(i,j)} \\ &i,j = \{1,2,\ldots,d\} \end{aligned} \tag{12}$$

The *a-priori* probabilities will be

$$w_A = \frac{1}{2} w_{old}; \quad w_B = \frac{1}{2} w_{old} \tag{13}$$

Finally, their ages, a_A and a_B, will be reset to zero.

3.4 Merging Two Gaussians

For each Gaussian B other than the given one A, the algorithm evaluates a dissimilarity index. If that index is lower than the merge threshold M_{TH}, Gaussian B will be merged with Gaussian A. There are still no exact methods for computing the overlap between two Gaussians in arbitrary dimension. The existing methods use iterative procedures to approximate the degree of overlap between two components (Sun et al., 2007).

Here, we introduce a new dissimilarity index between Gaussian distributions, χ, that uses a closed form expression, evaluated as follows:

$$\begin{aligned} \chi_{i,j} &= \|\rho_{i,j} + \delta_{i,j}\|^2 \\ \rho_{i,j} &= \frac{\sum_{i,j} \|\Sigma_{Ai,j} - \mu_{Ai}\| \cdot \|\Sigma_{Bi,j} - \mu_{Bi}\|}{\xi_i \cdot \xi_j} \\ & with \;\; i,j = \{1,2,\ldots,d\} \end{aligned} \tag{14}$$

where d is the input dimension, $\chi_{i,j}$ is a new *dissimilarity index* for mixture components, $\delta_{i,j}$ is the Euclidean distance between the two Gaussians i and j. Another solution would have been to choose the Mahalanobis distance (Mahalanobis, 1936) instead of $\chi_{i,j}$. This takes into account both the mean an the covariance of the matrixes. Hoverer, the Mahalanobis distance gives rise to zero if the two Gaussians have the same mean, independently whether they cover the same space (or, whether they have the same area ξ), then rendering it not suitable for our purposes. Again, we to avoid singularities, we use the *area* instead of the covariance matrix determinant at the denominator of the correlation index $\rho_{i,j}$. Finally, we defined $\chi_{i,j}$ as the square of the norm because this heightens values of $\|\rho_{i,j} + \delta_{i,j}\| > 1$ while reducing those $\|\rho_{i,j} + \delta_{i,j}\| < 1$.

The new Gaussian (*new*) will have the following mean and covariance:

$$\begin{aligned} \mu_{new(i)} &= \frac{1}{2}(\mu_{iA} + \mu_{iB}); \quad \rho_{new(i,j)} = \frac{1}{2}(\rho_{Ai,j} + \rho_{Bi,j}) \\ & with \;\; i,j = \{1,2,\ldots,d\} \end{aligned} \tag{15}$$

where d is the input dimension. The *a-priori* probabilities for the remaining Gaussian will be updated as:

$$\begin{aligned} c_{new} &= c_{old} - 1; \quad w_i = w_i + w_{old}/c_{new} \\ & with \;\; i|_{i \neq old} = \{1,2,\ldots,c_{new}\} \end{aligned} \tag{16}$$

Finally, like when adding a Gaussian, the age a_{new} of the resultant Gaussian will be reset to zero.

3.5 Updating Decision Thresholds

The thresholds L_{TH}, S_{TH}, and ξ_{TH} vary with the following rules:

$$\begin{aligned} L_{TH} &= L_{TH} - \lambda \cdot L_{TH} = L_{TH} \cdot (1 - \lambda) \\ S_{TH} &= S_{TH} + \alpha \cdot (\xi - S_{TH}) = S_{TH} \cdot (1 + \alpha \cdot \xi) \\ M_{TH} &= M_{TH} + \gamma \cdot (\chi_{A,B} - M_{TH}) = M_{TH} \cdot (1 + \gamma \cdot \chi_{A,B}) \\ & with \;\; \xi - S_{TH} < 0, \;\; \chi_{A,B} - M_{TH} > 0 \end{aligned} \tag{17}$$

with λ, α, and γ chosen arbitrarily low (we used $\lambda = 0.04$, $\alpha = 0.04$, $\gamma = 0.001$). Following this rules L_{TH} will decrease step by step, approaching the current value of the global log-likelihood increment. This is the same for S_{TH}, which will become closer to the *area* of some Gaussian, and for M_{TH} that will increase. This will allow the system to avoid some local optima, by varying its configuration if a stationary situation occurs.

Finally, every time a Gaussian is added or merged, these thresholds will be reset to their initial value.

3.6 Computational Complexity Evaluation

Within this section we will use the following convention: *ng* is the number of the mixture Gaussian components, k is the number of input vectors, d is the number of input dimension, and *it* is the number of iterations.

The computational burden of the EM algorithm is, referring to the pseudocode in tab. 3.1 as follows:

- the original EM algorithm (steps 3 to 6) take $O(k \cdot d \cdot ng)$ for 3 and 6, while step 4 and step 5 take $O(1)$ and $O(k \cdot ng)$;
- our algorithm takes $O(ng)$ for evaluating all the Gaussians (step 8 to 36) and another $O(ng)$ in step 23 for evaluating the dissimilarity between each Gaussian (14);
- our merge and split (step 15 and 25) operations require $O(d)$ and $O(d \cdot ng)$, respectively.
- the others take $O(1)$.

Therefore, the original EM algorithm takes $O(k \cdot d \cdot ng)$, while our algorithm adds $O(d \cdot ng^2)$ on the whole, giving rise to $O(k \cdot d \cdot ng) + O(d \cdot ng^2) = O(k \cdot d \cdot ng + d \cdot ng^2) = (ng \cdot d \cdot (k + ng))$. Considering that usually $d << k$ and $ng << k$ this does not add a considerable burden, while giving an important improvement to the original computation in terms of self-adapting to the data input configuration at best.

4 EXPERIMENTAL VALIDATION

In order to evaluate the performance of our algorithm, we tested it by classifying different input data randomly generated by a known Gaussian mixture, and subsequently saved to a file. This is in order to use the same input data to our algorithm, SAGEM, and to the original EM. Our algorithm starts with a low initial number of Gaussians, while the original EM starts with the exact number of Gaussians, i.e. the mixture configuration we generated the input data points from. Moreover, in order to make a fair comparison, both the algorithms started with the same input Gaussian means (of course, since SAGEM has fewer Gaussian components than EM these are a subset of those of EM).

We refer to the EM with the exact number of Gaussians that generated the input data as the best EM algorithm applicable, i.e. the algorithm that uses the best compromise between number of Gaussian components and final likelihood. Therefore, we will use its results as comparison for our algorithm.

4.1 Experimental Set-up

We made different trials, with mixtures containing 4, 8, 12, 14, and 16 Gaussians. Each of them contains 2000 points in two dimensions. We choose to show the results for 2-dimensional input because they are easier to show than multidimensional ones (for instance, a 2-dimensional Gaussian is represented in 2D as an ellipse). As the ratio *(# Gaussians)/(# data points)* increases, it become harder to reach a good solution in the reasonable number of steps. Therefore, we are interested in evaluating how SAGEM behaves when the model complexity gradually increases.

We evaluated the performance of our algorithm compared with the original EM principally based on three main points:

- Final log-likelihood value;
- Final number of Gaussian components;
- Number of iterations needed for reaching stability.

Therefore, we used the following equations for evaluating the error on the final Log-likelihod, and the error on the predicted number of Gaussian components as follows, respectively (the required number of iterations can be extracted from the plots in Fig. 1, directly.

$$Log - Lik_{err} = \frac{Log - Lik_{(SAGEM)}}{Log - Lik_{(EM)}}$$
$$NumGauss_{err} = NumGauss_{(SAGEM)} - NumGauss_{(EM)} \quad (18)$$

In table I the output of the different computations are shown.

The results will be discussed within the next section.

4.2 Discussion

The output of the two algorithms is shown in Fig. 1. The input data (green points) with the generation mixture (blue) and the evaluated one (red) are represented in the same figure, while the resultant log-likelihood (blue for the EM algorithm and red for SAGEM) is shown. It is worth noticing that we also represented the iterations at which a split or a merge operation is performed, as vertical blue and red lines, respectively. Here, it is possible to see how just after a splitting or merging operation the final log-likelihood has some spikes. When a merge operation is performed the algorithm decreases the number of Gaussian components, therefore decreasing the log-likelihood momentarily abruptly. An example is at iteration 92 of the 12-Gaussian plot. Besides, when a new class is added by means of a splitting operation, it may happen that the final log-likelihood starts increasing smoothly due to the new component's contribute (e.g. see the 8-Gaussian plot at iteration 22, or the 14-Gaussian plot at iteration 50), or has a decreasing spike (e.g. see the 8-Gaussian plot at iteration 34 or 53) If the components are reorganized by the EM procedure in order to describe the data better, the log-likelihood will start to increase smoothly.

Finally, Fig. 2.6 shows the 3D histogram representation of a generated Gaussian mixture data and the estimated one. Due to space limitations, we choose to show only the one that gave rise to the worst log-likelihood estimation plot, i.e. the one with 8 Gaussians.

In table I the results of the different computations are shown. The table contains, for both SAGEM and the EM algorithm:

- The starting number of Gaussian components;
- The final (i.e. detected, in the case of the SAGEM approach) number of Gaussian components;
- The error on the final number of components.
- The final reached log-likelihood.
- The error on the final log-likelihood, as absolute value.

Due to the formulation in (18), if $Log - Lik_{err} < 1$ than SAGEM reached a final log-likelihood greater than EM (both are negative), and vice versa. Similarly, when $NumGauss_{err} < 0$ it means that SAGEM

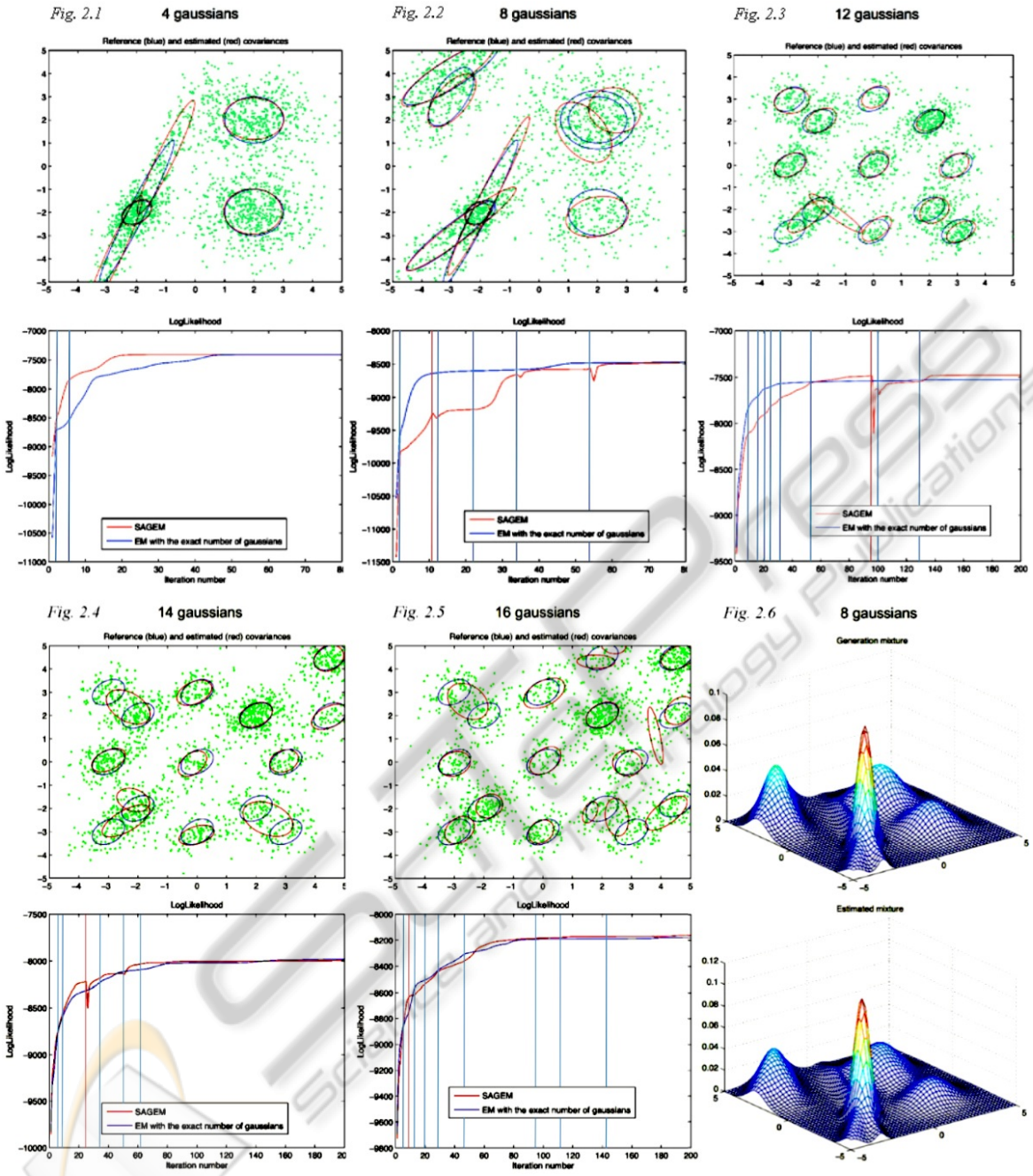

Figure 1: The 2D representation of the final SAGEM Gaussian mixture vs. the real one and the SAGEM and EM log-likelihood output as function of the iterations number, for different input mixtures of data (4, 8, 12, 14, and 16 Gaussian components). Moreover the 8-Gaussians case comparison between the generated and computed mixtures is shown.

used fewer Gaussians then real ones. These two comparisons are shown in plots 2(a), 2(b), and 2(c) respectively.

We can see that the final SAGEM log-likelihood is often better than the one obtained with the original EM with the exact number of Gaussians, except for the case of 8-Gaussians. There are three interesting points:

- Within the 14-Gaussians case SAGEM reached a higher log-likelihood than EM even with less Gaussians components than required.
- Increasing the number of components, the points representing each class will be fewer. This in-

Table 1: Experimental results.

Number of effective mixture Gaussians	Algorithm	Starting number of Gaussians	Arrival number of Gaussians	Error on the number of Gaussians	Final log-likelihood	Error on the final log-likelihood
4	SAGEM	2	4	0	-7402.82	-2.26
	EM	4	4	/	-7405.08	/
8	SAGEM	4	8	0	-8448.56	14.43
	EM	8	8	/	-8434.13	/
12	SAGEM	6	12	0	-7501.62	-28.36
	EM	12	12	/	-7526.74	/
14	SAGEM	8	12	-2	-7928.95	-53.71
	EM	14	14	/	-7982.66	/
16	SAGEM	10	17	+1	-8161.29	-17.13
	EM	16	16	/	-8178.42	/

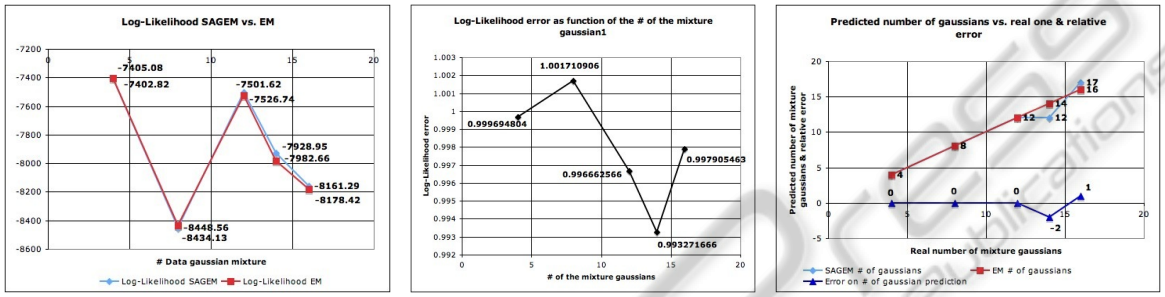

Figure 2: The final log-likelihood of both SAGEM and EM as function of the number of input data Gaussian components (a), the relative error (b), and the final reached number of Gaussians as function of the input ones (c).

creases the difficulty of clustering them into distinct classes. Nevertheless, SAGEM has been able to detect a number of required Gaussians close to the original one (12 vs. 14 and 17 vs. 16), and to have a better final log-likelihood than EM.

- Regarding to the plots of the log-likelihood we can see that both SAGEM and EM reach a similar likelihood within the first 60-80 iterations, even though EM is usually faster in increasing it. Some spikes in the SAGEM plots occur, due to the modification of the number of Gaussians. In fact, when merging two mixture components the log-likelihood momentarily decreases.
- Even though the 8-Gaussian case is the worst one in terms of final likelihood, the histogram of the final Gaussian mixture is very similar to the true one.

5 CONCLUSIONS AND FUTURE WORK

In this paper we proposed a new split-and-merge Expectation Maximization algorithm, called SAGEM. The algorithm is specific for Gaussian mixtures. SAGEM starts with a fixed number of Gaussians, and automatically decides whether increasing or reducing it. We introduced a new concept, a dissimilarity index between two Gaussian distributions. Moreover, in order to evade local optimal solutions we make use of self-adaptative thresholds for deciding when Gaussians are split or merged. We tested it with different Gaussian mixtures, comparing its results with those of the EM original algorithm set with the model complexity that matches those we generated the data from (ideal case). We showed that our algorithm is capable to evaluate a number of Gaussian components close to the true one, and to provide a final mixture description with a log-likelihood comparable with the original EM, sometimes even better. Finally, its convergence occurs with the same number of iterations than it does for the original EM (60-80 iterations).

5.1 Future Work

At the moment we tested our algorithm with synthetic data. As future work, we will test SAGEM for the purpose of image segmentation on real images captured in order to test it in real robotic applications, where the computational burden must be kept low. More specifically, being part of the RobotCub European Project, we will test it with the iCub robotics platform.

ACKNOWLEDGEMENTS

This work was supported by the European Commission, Project IST-004370 RobotCub and FP7-231640 Handle, and by the Portuguese Government - Fundação para a Ciência e Tecnologia (ISR/IST pluriannual funding) through the PIDDAC program funds and through project BIO-LOOK, PTDC / EEA-ACR / 71032 / 2006.

REFERENCES

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood estimation from incomplete data via the em algorithm. *J. Royal Statistic Soc.*, 30(B):1–38.

Figueiredo, A. and Jain, A. (2002). Unsupervised learning of finite mixture models. *IEEE Trans. Patt. Anal. Mach. Intell.*, 24(3).

Mahalanobis, P. C. (1936). On the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):39–45.

McLachlan, G. and Peel, D. (2000). Finite mixture models. *John Wiley and Sons*.

Pernkopf, F. and Bouchaffra, D. (2005). Genetic-based em algorithm for learning gaussian mixture models. *IEEE Trans. Patt. Anal. Mach. Intell.*, 27(8):1344–1348.

Rissanen, J. (1989). Stochastic complexity in statistical jnquiry. *Wold Scientific Publishing Co. USA*.

Sakimoto, Y., Iahiguro, M., and Kitagawa, G. (1986). Akaike information criterion statistics. *KTK Scientific Publisher, Tokio*.

Sun, H., Sun, M., and Wang, S. (19-22 August 2007). A measurement of overlap rate between gaussian components. *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong,.*

Ueda, N., Nakano, R., Ghahramani, Y., and Hiton, G. (2000). Smem algorithm for mixture models. *Neural Comput*, 12(10):2109–2128.

Zhang, Z., Chen, C., Sun, J., and Chan, K. (2003). Em algorithms for gaussian mixtures with split-and-merge operation. *Pattern Recognition*, 36:1973 – 1983.