Gradual Model Generator for Single-pass Clustering

Ismo Kärkkäinen and Pasi Fränti
Department of Computer Science, University of Joensuu, FIN-80101, Finland
ismo.karkkainen@cs.joensuu.fi, pasi.franti@cs.joensuu.fi

Abstract

We present an algorithm for generating a mixture model from data set by performing a single pass over the data. The method is applicable when the entire data is not available at the same time in the main memory. We use Gaussian mixture model but the algorithm can be adapted to other types of models, too. We also outline a post processing method, which can iteratively reduce the size of the model obtained by the single-pass algorithm. This will result in a model with fewer components, but with approximately the same representation accuracy than the result of the original model from the single-pass algorithm.

1. Introduction

Clustering algorithms are used to find structure in data. Majority of the existing algorithms store the whole data in memory, or make several passes over the data, which can restrict their usefulness in the case of large data sets that exceed the available memory resources. Algorithms designed specifically for very large data sets must be able to read the data as few times as possible, preferably only once. Another desired property is to have independence of the order in which the data is processed. Random access to the data may be costly, and therefore, algorithms requiring it can be too slow for practical use.

There are existing algorithms that generate a Gaussian mixture model (GMM) from a large data set. *Online EM* algorithm by Sato and Ishii is described in [1]. It starts with an initial solution and then updates the existing components, deletes them or adds new ones if necessary.

Scalable EM by Bradley et al [2] has been modified from the EM algorithm to operate with the compressed data along with the uncompressed data. The algorithm consists of the EM algorithm, *k*-means and agglomerative clustering algorithm run on the compressed data set.
Here the aim is to model the data vectors using a Gaussian mixture model. The goodness of the model is measured with the average *log-likelihood* of the data.

In this paper, we propose an algorithm that converts the data into a multivariate Gaussians by selecting small, compact subsets of the data and updating them with data that lies close to the generated Gaussians.

2. Overview of the algorithm

The proposed algorithm consists of two stages: single-pass model generator, and model simplification as a post-processing step, illustrated in Figure 1. First, a model is generated from the data by the single-pass algorithm. The model can then be post-processed to obtain a simpler representation. Post-processing can be performed without the original data. The result of the post-processing is a model that represents the same data but with fewer components. The benefit of the two stages is that the model of the first stage takes less time and memory to be processed than the entire data set would take.

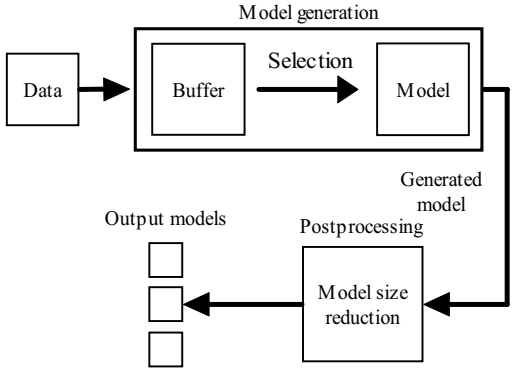

Figure 1. Outline of the single-pass clustering.

It is not a trivial task to restrict the size of the model during the single-pass of the data without affecting the quality of the model. We have decided to let the model grow and over-fit the data, and leave the reduction of the model size to the second stage. If the model grows too large, we can save it, start a new one from scratch, and continue to process the rest of the data. Saved models can later be joined together by adjusting the weights of the components in proportion to the number of points used to generate each model.

The goal of the second stage is to find the actual clusters. An ordinary clustering algorithm can be adapted to use the components of the model instead of points.

Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)
1550-4786/05 $20.00 © 2005 IEEE

3. Generating the model

The basic idea of generating the model is either to update the existing model with new points whenever the model describes the data sufficiently well, or to use a set of points to create a new component into region where the existing model does not describe the data well enough. For the purposes of storing the data that has not yet been included, we use a fixed-size buffer from which the points are selected to create new components.

```
GMG(data set, buffer size, points to select): GMM
Initialize data buffer and create empty model.
WHILE data points left and data buffer is not empty:
    Read next point.
    IF point fits into model THEN
        Update model
    ELSE
        Insert point into data buffer.
    IF data buffer is full or no data is left THEN
        Check if any points in buffer fit into model.
        IF some points fit into model THEN
            Update model
        ELSE
            Select points from buffer.
            Create a new component.
            Add it into the model
RETURN model.
```

Figure 2. Pseudocode of the algorithm

The pseudocode for the algorithm is shown in Figure 2. The model and data buffer are initially empty. The buffer will first be filled with new points, then the first component is created.

3.1. Creating new components

A new component is created by selecting a point and its *k*-neighborhood from the buffer so that the maximum distance from the point to the *k*th nearest neighbor is minimal. The goal is to find a group of points that lie on a small area, and are likely to belong to the same cluster. We must also select enough points to avoid numerical problems. The new component generated from selected points is added to the model. When all data points have been read, the remaining points in the buffer are processed and if necessary, more components are created.

In practice, we maintain the sum of the selected points along with the sum of their outer products, as proposed in [3] and [4] so that we can perform the updates.

The buffer size and the number of points required to generate a new component are the parameters of the algorithm. Larger buffer implies smaller *k*-neighborhood, producing more compact components. A larger number of selected points produces more widespread components. Having a small buffer and large *k* may produce components that join two clusters. Experiments show that on average only 3% of the points are used in creating new components.

3.2. Model update

Primarily we try to update existing components with new points. If the probability density at the location of a new point is higher than a given threshold determined by the algorithm, we update the model as in the EM algorithm. We compute the probability of the point with regard to every component, and update the weight, mean and covariance matrix of each component in proportion to it. Several updates are performed at the same time as in [4]. Experimentally we have found that on average 84% of the points are inserted into the model directly.

The acceptance threshold for direct model updates is updated during new component creation using the selected points. The new limit is the minimum of the old limit and the minimum probability density of the selected points with respect to the entire model.

When the data buffer fills up, we try to update the model with points in the buffer. If no points fit the model well enough, we generate a new component. Testing the points in the buffer is done because we may accept points for direct model update after the limit of acceptance or model has changed. Points in the buffer are tested for direct updates 34 times on average before they are used. On average, 13% of the points are stored in the buffer before they are used to update the model.

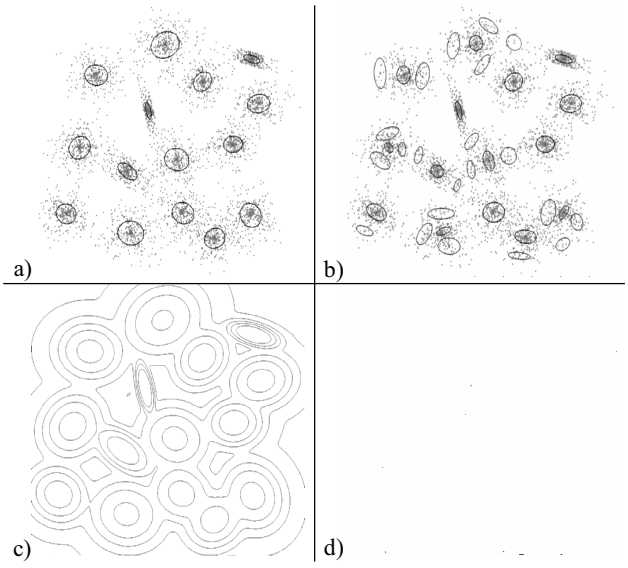

Figure 3. (a) the GMM generated by EM, (b) by the GMG, and (c and d) the corresponding contours of the probability density distributions

4. Post-processing the model

The model obtained using the GMG can have unnecessarily high number of components for cluster analysis as shown in Figure 3. It also shows that despite a higher number of components, the contour plots of the probability distributions do not differ significantly. The major modes are still in the same places. In order to study if the data is clustered, we can use adapted clustering algorithms for the generated model to try to find clusters. We used a modified k-means algorithm that is quite close to the algorithm presented in [5], except that Bhattacharyya-distance [6, 7] is used to measure the distance between components. Since the input model from GMG is considerably smaller than the original data, this procedure is much faster than using the original data.

5. Test results

We test the combination of GMG and adapted k-means (AKM), and compare it against Online EM, Scalable EM and EM [8, 9], which is used as a reference method. All of these algorithms, except EM, depend on the order in which the points in the data set are processed.

Two data sets are Corel Image features previously used in [10]. We selected the co-occurrence texture and color moments data sets. Third data set is *El Niño* used previously in the American Statistical Association Statistical Graphics and Computing Sections 1999 Data Exposition. Indices and points with missing attributes have been removed. We use three different orderings to study the order-dependency: the original order in which the data sets are available, data points sorted in an increasing order according to the first variable, and a random ordering. Online EM forgets past data as shown in [1], so it was only tested with the random ordering.

All data sets were obtained from [11] and are summarized in Table 1.

Table 1. Attributes of the real data sets.

Name	Dimensionality	Points
CM	9	68 040
CT	16	68 040
El Niño	7	93 935

5.1. Parameters used with different algorithms

The only parameters given to the EM-algorithm are the number of components and an ending threshold for the relative improvement of log-likelihood, set to 2^{-16}. Online EM required the number of components, and a factor for forgetting old data, which we set to 0.01.

Scalable EM requires the number of components, the number of points and components that are stored, variance limit for the compression stage, and the amount of how much the limit is increased if no compression appears to be possible. The number of points and components stored corresponds to the amount of memory available. This favors the algorithm since the components take up more space. We set it to 4000. The variance limit is estimated from the data, which corresponds to a good guess. Limit multiplier in case of failed compression is 1.5. The model size in the secondary compression phase was set to $n/2D$, where n is the size of the retained set and D is the data dimensionality. Hence the secondary compression was independent of the model size.

GMG was run once for each parameter/data set ordering combination. Buffer size was 4000. The number of points selected to create a component was 20 and 30.

5.2. Log-likelihoods

We show log-likelihood as a function of the number of components. For GMG, the plots show the log-likelihoods of models obtained after model size reduction performed using AKM. With other algorithms, we generated the models from scratch. Each curve represents an average log-likelihood of 20 runs.

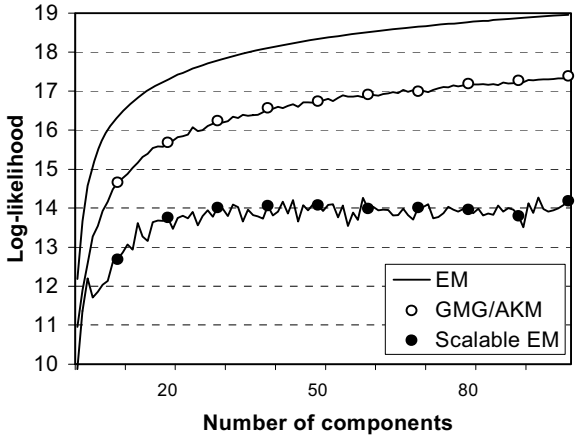

Figure 4. Results for *CT* in original order

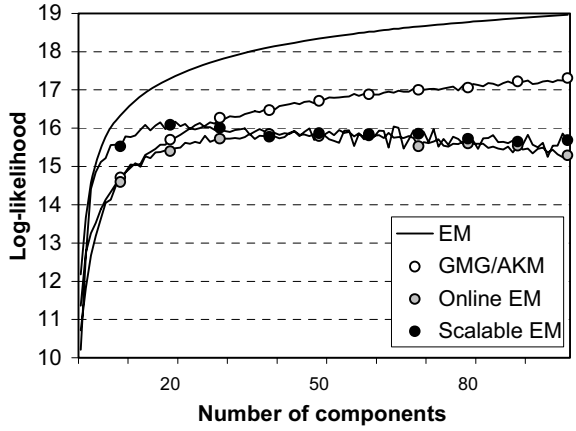

Figure 5. Results for *CT* in random order

Figure 4 shows average log-likelihoods for *CT* data set for original order and Figure 5 for random order. For GMG the number of points selected is 20, since this yielded GMMs of at least 100 components in all cases. Scalable EM performs better with smaller models, while still not reaching the performance of the proposed method for larger models. Figure 6 shows the results for sorted *El Niño* data set. The difference between orderings is much smaller for GMG/AKM than for Scalable EM.

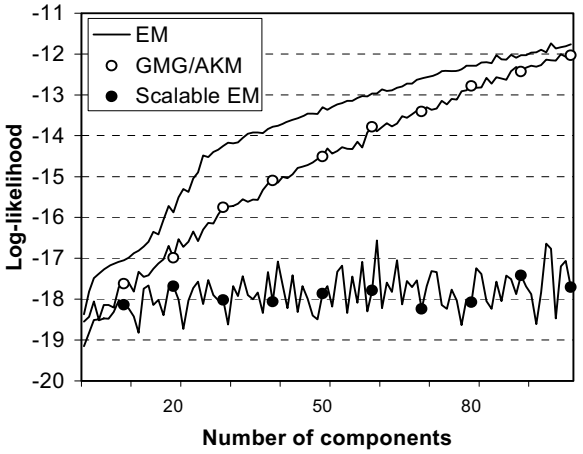

Figure 6. Results for *El Niño* in sorted order

5.3. Running times

The running times for EM, Online EM and Scalable EM show how much time, on average, it would take to generate one solution for each of model sizes of 2 – 100 components. For GMG/AKM the total includes the time spent by GMG in constructing the model and the time used by AKM to process the model. Table 2 shows the time for *CT* and Table 3 for *CM*. For the proposed method, the number of points selected to generate new component is shown in parenthesis.

Table 2: Total times for *CT* in seconds.

Algorithm	Data order		
	Original	Sorted	Random
EM	81 523	81 523	81 523
GMG/AKM (20)	7323	19 651	5613
GMG/AKM (30)	96	135	96
Scalable EM	15 043	50 977	13 137
Online EM	N/A	N/A	2257

Table 3: Total times for *CM* in seconds.

Algorithm	Data order		
	Original	Sorted	Random
EM	40 203	40 203	40 203
GMG/AKM (20)	279	421	205
GMG/AKM (30)	89	84	70
Scalable EM	14 445	93 906	7483
Online EM	N/A	N/A	1060

In Table 2, the proposed method is the fastest when 30 points are selected, but with 20 points selected the total time exceeds that of Online EM. The reason might be that we select 20 points from 16-dimensional data, which may produce lots of very small components. The number of points selected apparently should be close to twice the dimensionality of the data, or more. Running times of Online EM are quite similar between different runs.

6. Conclusions

In this paper, we present an algorithm that converts the input data into a GMM. The basic idea of the first stage is to find small, compact subsets of the data and turn these into multivariate Gaussians. The result can be post-processed in the second stage to generate a model of a desired size. We find out that the order of the data has only small effect on the performance of GMG. The time usage shows that the proposed method can efficiently generate a model from the original data and then a series of models with a smaller number of components.

References

[1] M. Sato, and S. Ishii, "On-line EM Algorithm for the Normalized Gaussian Network", *Neural Computation*, 12(2) (2000), pp. 407-432.

[2] P. Bradley, U. Fayyad, and C. Reina, "Clustering Very Large Databases Using EM Mixture Models", *Proc. of the 15th Int. Conf. on Pattern Recognition vol. 2*, 2000, pp. 76-80.

[3] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: A New Data Clustering Algorithm and Its Applications", *Data Mining and Knowledge Discovery*, 1(2) (1997) 141-182.

[4] C. Ordonez, and E. Omiecinski, "FREM: Fast and Robust EM Clustering for Large Data Sets", *Proc. of the 11th int. conf. on Information and Knowledge Management*, 2002, pp. 590-599.

[5] H. Jin, K.-S. Leung, M.-L. Wong, and Z.-B. Xu, "Scalable model-based cluster analysis using clustering features", *Pattern Recognition*, 38 (5), 2005, pp. 637-649.

[6] A. Bhattacharyya, "On a Measure of Divergence Between Two Statistical Populations Defined by Their Distributions", *Bull. Calcutta Math. Soc.*, 35, 1943, pp. 99-110.

[7] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, Boston, 1990.

[8] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm", *Journal of the Royal Statistical Society B*, 39, 1977, pp. 1-38.

[9] G. McLachlan, and T. Krishnan, *The EM Algorithm and Extensions*, John Wiley & Sons, New York, 1997.

[10] M. Ortega, Y. Rui, K. Chakrabati, K. Porkaew, S. Mehrotra, and T.S. Huang, "Supporting Ranked Boolean Similarity Queries in MARS", *IEEE Transactions on Knowledge and Data Engineering*, 10(6), 1998, pp. 905-925.

[11] S. Hettich, and S. Bay, The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science. 1999.