A Fast Convergence Clustering Algorithm Merging MCMC and EM Methods

David Sergio Matusevich
University of Houston
Houston, TX 77204, USA

Carlos Ordonez
University of Houston
Houston, TX 77204, USA

Veerabhadran Baladandayuthapani
MD Anderson Cancer Center
University of Texas
Houston, TX 77030, USA

ABSTRACT

Clustering is a fundamental problem in statistics and machine learning, whose solution is commonly computed by the Expectation-Maximization (EM) method, which finds a locally optimal solution for an objective function called log-likelihood. Since the surface of the log-likelihood function is non convex, a stochastic search with Markov Chain Monte Carlo (MCMC) methods can help escaping locally optimal solutions. In this article, we tackle two fundamental conflicting goals: Finding higher quality solutions and achieving faster convergence. With that motivation in mind, we introduce an efficient algorithm that combines elements of the EM and MCMC methods to find clustering solutions that are qualitatively better than those found by the standard EM method. Moreover, our hybrid algorithm allows tuning model parameters and understanding the uncertainty in their estimation. The main issue with MCMC methods is that they generally require a very large number of iterations to explore the posterior of each model parameter. Convergence is accelerated by several algorithmic improvements which include sufficient statistics, simplified model parameter priors, fixing covariance matrices and iterative sampling from small blocks of the data set. A brief experimental evaluation shows promising results.

1. INTRODUCTION

Clustering is a fundamental data mining technique that is frequently used as a building block for the treatment of more complex problems such as Class Decomposition and Bayesian Classifiers. Maximization algorithms' weakness is that, if the problem to be explored has more than one possible extrema, they don't return the best possible solution, but only the first maximum encountered. This has been extensively studied in the literature. Considerable work has gone into improving the EM algorithm and into accelerating its convergence. In [11], the authors present a survey of several approaches from the Machine Learning community, with mathematical justifications for them.

In this work we present a modification to the classic EM algorithm that searches for better solutions (e.g. lower log-likelihood) based on the Monte Carlo (MC) approach. MC techniques use random walks to explore the solution space. In general it is found that, even though MC algorithms take a long time to converge, they reach very good solutions avoiding been trapped in local extrema. While in recent years the problem of accelerating the EM algorithm has been studied, ([6], [10] among others) not much has been done to speed-up the convergence of Gibbs Samplers for mixtures of Gaussians. In this paper we describe the FAMCEM algorithm, the hybridization of EM with MC for faster, more accurate results.

2. DEFINITIONS AND PRELIMINARIES

The Expectation-Maximization Algorithm (EM) is used to estimate the parameters of a mixture of k normal distributions. The *multivariate normal distribution* for a d dimensional vector x for cluster $j \in \{1, \dots, k\}$ with mean C_j and covariance matrix R_j is:

$$P(x; C_j, R_j) = \left((2\pi)^d |R_j|\right)^{-\frac{1}{2}} e^{-\frac{1}{2}(x-C_j)^T R_j^{-1}(x-C_j)} \quad (1)$$

and $P(y; C, R, W) = \sum_{i=1}^{k} W_j P(x; C_j, R_j)$ is the probability of the mixture. W is a matrix that contains the weights of each of the clusters. Note that R_j is a $d \times d$ matrix, however since it is diagonal, we can just keep the non-zero elements as a $d \times 1$ vector

The *Mahalanobis Distance* of a multivariate vector x from a group of variables with mean C_j and covariance R_j is defined as

$$\delta(x, C_j, R_j) = \delta_{ij} = (x - C_j)^T R_j^{-1} (x - C_j) \quad (2)$$

This distance is used in EM instead of the regular Euclidean distance [3] since it provides an understanding, not only of how far a certain point is from the center of mass of the cluster, but also how dense the cluster is, by scaling the distance by the covariance matrix. EM takes as an input the data-set ($X = \{x_1, \dots x_n\}$) and gives as an output the matrices C, R and W. The quality of the solution is measured by the *Log-Likelihood*, defined as:

$$L(\Theta) = \frac{1}{n} \sum_{i=1}^{n} \log(P(y_i; \Theta)) \quad (3)$$

In general EM algorithms stop when the change in log-likelihood is less than a predetermined precision.

According to [12] a Markov process is a stochastic process such that the conditional probability of value x_i at time

t_i is uniquely determined by the value x_{i-1} at time t_{i-1} and not by any knowledge of the values at earlier times. In a Monte Carlo algorithm, we generate a Markov chain in which each successive state is generated by its predecessor and accepted (or rejected) according to an acceptance ratio determined by the probabilities of each state. This algorithm randomly attempts to sample the solution space, sometimes accepting the move and sometimes remaining in place. Gibbs Samplers are Markov Chain Monte Carlo algorithms for obtaining sequences of observations of the joint probability distribution of a set of random variables, when direct sampling is difficult. Gibbs Samplers are commonly used as means of Bayesian Inference instead of deterministic algorithms such as EM. In order to illustrate the computational power of Monte Carlo techniques, it is customary to use examples borrowed from Physics. A common problem in the area, is to determine the minimum of the potential energy in order to find the equilibrium state of a system. Determining the global extrema of this potential function, and the state with the lowest energy is not trivial since systems with more than 3 bodies are not solvable in a closed manner.

Monte Carlo algorithms (in particular the Metropolis Hastings model) were developed to solve this kind of problems [4]. Deterministic schemes, such as the Gauss- Newton Algorithm and its variants (or for that matter EM) find progressively better solutions in an iterative manner, with the result that, if there are several possible minima, the algorithm can stop at a local one, without finding the global result we are looking for. In contrast Monte Carlo iterations allow the solution to worsen, climbing out of the shallow valleys in order to find the deep ones. In other words, Monte Carlo techniques effectively sample phase space until the best possible solution is found.

3. MERGING EM WITH MONTE CARLO

Markov Chain Monte Carlo (MCMC) methods are a set of algorithms for sampling probability distributions by constructing a Markov Chain whose equilibrium distribution is the desired one. Since it is difficult to determine a priori the number of steps (iterations) that will result in a stable distribution, MCMC models tend to overestimate the number of iterations needed, therefore requiring longer computation time. This issue is particularly important when the size of the problem is significantly large, since each step of the computation will be repeated a very large number of times. In this section we describe our efforts to improve EM by incorporating ideas from Monte Carlos style algorithms. To begin we will briefly describe the FREM algorithm from [8], since we used it as our starting point. We will also analyze the time complexity of the complete algorithm.

3.1 Sufficient Statistics

Sufficient statistics are multidimensional functions that summarize the properties of clusters of points. One of the interesting properties of these summaries is that they are independent, that is, statistics from one cluster do not depend on the values of the data points from other clusters. In this case we introduce three statistics per cluster D_j: $N_j = |D_j|$, $L_j = \sum_{i=1}^{N_j} x_i$ and $Q_j = \sum_{i=1}^{N_j} x_i x_i^T$

N stores the number of points per cluster and is a $k \times 1$ matrix while L stores the sum of the data points and Q stores the sum of the squares. L and Q are $k \times d$ matrices. FREM uses these sufficient statistics to reduce the number of reads of the dataset, allowing the periodic estimation of the parameters without resorting to extra I/O steps, significantly reducing the running time of the algorithm.

3.2 FREM's Improvements over EM

A significant improvement described in [8] was the introduction of the parameter λ to circumvent the weakness of EM when variances approach zero, that is, when there is very little variation in one of the dimensions. This parameter is a small positive constant that, multiplied by the global standard deviation, is added to the variances during its update in the maximization step. It is chosen small enough to avoid the real variance values, but large enough to avoid zeros that could make the results undefined.

The formulae used in FREM (and in FAMCEM) is

$$C_j = \frac{L_j}{N_j} \tag{4}$$

The d-dimensional cluster average

$$W_j = \frac{N_j}{\sum_{j'=1}^{k} N_{j'}} \tag{5}$$

The weight of the cluster

$$R_j = \frac{Q_j}{N_j} - \frac{L_j L_j^T}{N_j^2} + \lambda \Sigma \tag{6}$$

The d-dimensional variance, with the λ factor

FREM also makes use of the fact that, for sufficiently large data sets, a portion of the data-set is representative of the whole. This "fractal" behavior of data is illustrated in Figure 1. In it, we plot two dimensional synthetic data in the (i, j) plane. The complete data-set consists of 5500 points distributed in three clusters. In the second part of the graph we show a random sampling of 10% of the data points and it presents the same clustering behavior (albeit with a lower density). Going back to FREM, the algorithm can be accelerated by including Maximization steps before the Expectation state has finished processing the complete data-set. FREM uses a parameter ψ, that the authors fixed experimentally to $\psi = \lfloor \sqrt{n} \rfloor$. Since the M step is performed roughly n/ψ times during a single iteration of the algorithm, the convergence is accelerated substantially.

3.3 The FAMCEM Algorithm

FAMCEM introduces two significant changes to FREM. The first one is a modification to the Expectation step: In FREM, as in the regular EM algorithm, the datapoint is assigned a series of probabilities of belonging to each of the clusters. In FAMCEM we use these probabilities to find a unique cluster to which the data point belongs, this produces a decoupling between the clusters, simplifying (as we will see) the calculation of the Log-Likelihood. However, instead of assigning the data point to the cluster with highest probability, we use the Alias Method [5] and [2]. This is the main Monte Carlo change, since it allows the algorithm to *explore* the parameter space, by allowing the possibility that the data point belongs to a lower probability cluster. By letting the algorithm *swim upstream*, climbing over local maxima, we allow it to find possible better energetic minima.

To illustrate this step we can imagine a distribution of points in two-dimensional space: Some points are relatively easy to place within a cluster, that is to say, the proba-

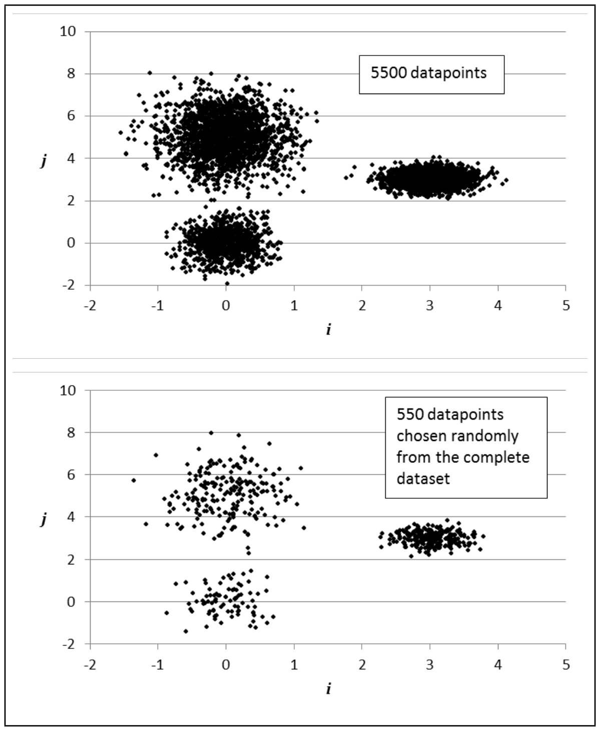

Figure 1: The Fractal behavior of data.

bilities are skewed in favor of only one cluster. However, in general this is not the case. In fact any point between two centroids could belong to either one, particularly if the standard deviations (radii) are large enough. While at some point during the procedure, the probability might favor one cluster in detriment of others it is possible that this is just an artifact of the current distribution. Allowing a certain opportunity to be included in other clusters, while making the current iteration slightly worse, could lead to an overall improvement of the solution.

The FAMCEM divides the iterations into two main phases. In the first, burn-in phase, the algorithm uses a regular FREM Maximization step in order to calculate the initial standard deviations and variances. After the burn-in period is finished, we start a Sampling phase where, instead of using Maximization steps we use Sampling steps.

The Sampling step is the second main difference with FREM. The initial update of the parameters is performed as in equations 4 and 5, however we add an extra step: After we have an estimate for C_j we use a normal distribution to calculate C'_j, using $|R_j|$ as the standard deviation. This allows for further exploration of the solution space, improving our chances of finding the global extrema. It is important to emphasize that the Sampling phase only explores the priors of C and W but not R. R uses a non-informative prior and remains fixed during Sampling, to reduce the complexity of the problem [7].

The reason for these hybrid approach is to use the first part to estimate the covariance matrices, and use them in the sampling phase. Sampling requires knowledge of the initial distribution of the priors [7] and we resort to FREM to calculate them. It is important to mention that the posterior probability for C arises naturally as a result of our calculations.

Table 1: Experimental Results.

k	FAMCEM		EM	
	$L(\Theta)$	time [s]	$L(\Theta)$	time [s]
9	-102.797	88	-92.706	34
12	-167.203	126	-174.418	44
15	-110.838	152	-134.374	61
18	-168.189	189	-188.437	76

As we mentioned previously, choosing a unique cluster for each data point, coupled with the fact that the covariance matrices are diagonal, has the added benefit of simplifying the calculation of the log-likelihood. This quantity can be calculated using the sufficient statistics matrices, in a single step, without resorting to reading the data-set a second time, or making extra calculation steps in the Expectation subroutine.

$$L(\Theta) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^{k}\left[\frac{1}{n}\left(Q_i R_i^{-1} - 2C_i^T Li R_i^{-1}\right) + W_i\left(C_i^T R_i^{-1} C_i - |R_i|\right)\right] \quad (7)$$

3.4 Time Complexity

The time complexity of this algorithm is dominated by the number of iterations of the main loop and the number of data points in the data-set. In the worst case scenario the time complexity is $O(ndk)$ per iteration, as was calculated for FREM in [8]. However, while FREM reaches equilibrium in few passes, per force Monte Carlo style algorithms need a much larger number of iterations, since it much more difficult to achieve stability and convergence.

4. EXPERIMENTAL VALIDATION

We evaluated out proposed method by using a real data-set from the UCI Machine Learning Repository [1]. The data-set, represented measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Different electrical quantities and some sub-metering values were available. The data-set has 9 dimensions that include date of the measurement, time and 4 electrical measurements and 3 dimensions that represented discrete characteristics of the households.

We implemented the algorithm in C++. The data set was stored in a plain text file and the experiments were run on a machine using an Intel Core i5 CPU, running at 1.8 Ghz, with 4Gb of main memory. The computer was running a 64 bit version of Windows 8 operating system, and the program was developed using Windows Visual Studio 2012 Pro. The algorithm reads the data-set into memory once and operates in main memory.

We show the results in Table1. It shows the calculation of the Log-Likelihood for the data-set, when we varied the number of clusters considered for a fixed $n = 100000$ and $d = 5$. For comparison purposes we show the results from EM runs of exactly the same data. When comparing running times, FAMCEM is 2.49 times slower than a classic EM algorithm: for 18 clusters EM took 76.11 sec, versus 189.45

Algorithm 1 The FAMCEM Clustering Algorithm

```
Input: $X = \{x_1, x_2, \ldots, x_n\}$ and k
Output: $\Theta = \{C, R, W\}$ and $L(\Theta)$
  /* Parameters (defined in the text) */
  $\psi \leftarrow \lfloor\sqrt{n}\rfloor$, $\alpha \leftarrow \frac{1}{d\times k}$, $\lambda \leftarrow 0.01$
  for $j = 1$ to $k$ do
    $C_j \leftarrow \mu \pm \alpha \times r \times \text{diag}|\sigma|$, $R_j \leftarrow \Sigma_j$, $W_j \leftarrow 1/k$
  end for
  $I = 0$
  while $(|L(\Theta)_I - L(\Theta)_{I-1}| > \epsilon$ and $I \leq$ MAXITER$)$ do
    /* Initialize the sufficient statistic matrices */
    for $j = 0$ to $k$ do
      $L_j \leftarrow \overrightarrow{0}$, $Q_j = \overrightarrow{0}$, $N_j = 0$
    end for
    for $i = 1$ to $n$ do
      /* Expectation Step: Choose to which cluster
      the data point $x_i$ belongs according to the posterior
      probabilities. */
      Find the maximum probability cluster $m_0$
      Choose $m$ randomly, according to the
      probabilities $p_{ij}$.
      $a \leftarrow$ Random number $a \in [0, 1)$
      if $p(m)/p(m_0) > a$ then
        Use $m$
      else
        Use $m \leftarrow m_0$
      end if
      if $(i \mod \psi = 0$ or $i = n)$ then
        if $(i <$ BURNIN$)$ then
          /* Maximizing Step: During the burnin pe-
          riod, we maximize, to calculate the standard de-
          viations */
          Update Equations 4 through 6
        else
          /* Sampling Step: After the burn-in period,
          we sample instead of maximizing */
          Update Equations 4 and 5
          /* We purposely de-tune the centroids, in order
          to avoid local minima */
          $C_j \leftarrow \textbf{rnorm}(C_j, R_j)$
        end if
      end if
    end for
    Calculate the Log-Likelihood using Equation 7
    $\epsilon \leftarrow (1 - L(\Theta_{I-1})/L(\Theta_I))$
  end while
```

sec for FAMCEM due to running about 4 times as many iterations.

5. CONCLUSIONS

Although more experimentation is required, FAMCEM shows an improvement over FREM in the quality of solutions, however this is counterbalanced by the considerable increase in execution time. FAMCEM could be considered as an alternative to FREM when the accurate results are more important than time efficiency.

Future work will include optimizations for storage and time efficiency, as well as the translation of the code into C#, in order to enable SQL calls and, ultimately, the creation of an UDF that can be called directly from a DBMS, as well as the possible integration with MapReduce to enable Big Data Analytics. Further development on this problem will involve programming the algorithm directly in SQL as in [9].

Acknowledgements

This research work was partially supported by National Science Foundation grants IIS 0914861 and IIS 0915196.

6. REFERENCES

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] L. Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation*, pages 260–265. ACM, 1986.

[3] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. J. Wiley and Sons, New York, 1973.

[4] D. B. Hitchcock. A history of the Metropolis–Hastings algorithm. *The American Statistician*, 57(4):254–257, 2003.

[5] R. A. Kronmal and A. V. Peterson Jr. On the Alias Method for generating random variables from a discrete distribution. *The American Statistician*, 33(4):214–218, 1979.

[6] N. Kumar, S. Satoor, and I. Buck. Fast parallel expectation maximization for gaussian mixture models on GPUs using CUDA. In *High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference on*, pages 103–109. IEEE, 2009.

[7] L. Liang. On simulation methods for two component normal mixture models under Bayesian approach. *Uppsala Universitet, Project Report*, 2009.

[8] C. Ordonez and E. Omiecinski. Accelerating EM clustering to find high-quality solutions. *Knowledge and Information Systems (KAIS)*, 7(2):135–157, 2005.

[9] C. Ordonez and S. Pitchaimalai. Bayesian classifiers programmed in SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(1):139–144, 2010.

[10] B. Thiesson, C. Meek, and D. Heckerman. Accelerating EM for large databases. *Machine Learning*, 45(3):279–299, 2001.

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

[12] N. G. Van Kampen. *Stochastic processes in physics and chemistry.* North Holland, 1992.