Gaussian Mixture Model with MapReduce

Andrea Mannocci, Giuseppe Portaluri, Lucia Vadicamo

Suppose we have a dataset of observations $X = \{x_1, \dots, x_n\}$, $x_j \in \mathbb{R}^d$, and we wish to model this data using a mixture of Gaussians. We can estimate the parameters of a Gaussian Mixture Model (GMM) using the Expectation-Maximization (EM) algorithm [1] to optimize a Maximum Likelihood (ML) criterion. We denote the parameters of a k-component GMM by $\Theta = \{w_i \in \mathbb{R}, \mu_i \in \mathbb{R}^d, \Sigma_i \in \mathbb{R}^{d\times d};\, i = 1, \dots, k\}$, where w_i, μ_i, Σ_i are respectively the mixture weight, mean vector and covariance matrix of Gaussian i. The Gaussian mixture distribution can be written as:

$$p_\Theta(x) = \sum_{i=1}^{k} w_i p_i(x), \tag{1}$$

where

$$p_i(x) = \frac{\exp\{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\}}{(2\pi)^{\frac{d}{2}} |\Sigma_i|^{\frac{1}{2}}} \tag{2}$$

and

$$\sum_{i=1}^{k} w_i = 1, \quad w_k \geq 0 \qquad \forall\, i = 1, \dots, k. \tag{3}$$

If we assume that the data points are drawn independently from the distribution, then we can express the log of the likelihood function as:

$$\ln p(X|\Theta) = \sum_{s=1}^{n} \ln \left\{ \sum_{i=1}^{k} w_i p_i(x_t|\Theta) \right\}. \tag{4}$$

1 EM for Gaussian mixtures

Given a dataset X and a Gaussian Mixture Model, the goal is to maximize the likelihood function with respect to the parameters (comprising the means and covariances of the components and the mixing weight).

1. Initialize the means μ_i, covariances Σ_i and mixing coefficient w_i. K-means algorithm can be used to find a suitable initialization.

2. **Expectation step (E-step)** Use the current values of the current parameters to calculate the posterior probabilities $\gamma_s(i)$

$$\gamma_s(i) := p(i|x_s, \Theta) = \frac{w_i p_i(x_s|\Theta)}{\sum_{j=1}^{k} w_j p_j(x_s|\Theta)} \qquad \forall\, i = 1, \dots, k. \tag{5}$$

3. **Maximization step (M-step)** Use the posterior probabilities $\gamma_s(i)$ to re-estimate the means, variances, and mixing coefficient:

$$w_i^{\text{new}} = \frac{n_i}{n} \tag{6}$$

$$\mu_i^{\text{new}} = \frac{1}{n_i}\sum_{s=1}^{n}\gamma_s(i)x_s \tag{7}$$

$$\Sigma_i^{\text{new}} = \frac{1}{n_i}\sum_{s=1}^{n}\gamma_s(i)(x_s-\mu_i^{\text{new}})(x_s-\mu_i^{\text{new}})^T \tag{8}$$

   where $n_i = \sum_{s=1}^{n}\gamma_s(i)$, for all $i = 1,\dots,k$.

4. Check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

Each update to the parameters resulting from an E-step followed by an M-step is guaranteed to increase the likelihood function. The algorithm is deemed to have converged when the change in the likelihood function, or alternatively in the parameters, falls below some threshold.

In our experiments we are interested to fit a GMM on a dataset of image local features, such as SIFTs [2]. We assume diagonal covariances matrices and we denote $\sigma_i^2 \in \mathbb{R}^d$ the variance vectors, i.e. the diagonal of Σ_i. This is a suitable assumption for most computer vision applications, where estimating a full covariance matrix would be prohibitive due to the relative high dimensionality of the data. For example, when clustering SIFT features, each data vector has dimension 128, and each full covariance matrix would contain more than 8k distinct parameters.

Note also that the equations 6-8 can be computed in terms of the following 0-order, 1st-order and 2nd-order statistics:

$$S_0(i) = \sum_{s=1}^{n}\gamma_s(i), \quad S_1(i) = \sum_{s=1}^{n}\gamma_s(i)x_s, \quad S_2(i) = \sum_{s=1}^{n}\gamma_s(i)x_s^2, \qquad i = 1,\dots k$$

where $S_0(i) \in \mathbb{R}$, $S_1(i) \in \mathbb{R}^d$ and $S_2(i) \in \mathbb{R}^d$. The square of a vector must be intended as a term-by-term operation. We have:

$$w_i^{\text{new}} = \frac{S_0(i)}{n}, \qquad \mu_i^{\text{new}} = \frac{S_1(i)}{S_0(i)}, \qquad (\sigma_i^2)^{\text{new}} = \frac{S_2(i)}{S_0(i)} - \frac{S_1(i)^2}{S_0(i)^2}. \tag{9}$$

2 MapReduce implementation

In our implementation of the EM algorithm for this project, each iteration is performed by a MapReduce job described as follows.

Input: a data set of observations $X = \{x_1,\dots,x_n\}$, such as the local descriptors of several images.

Map *Key in*: -, *Value in*: $x_s \in \mathbb{R}^d$

- read current Gaussian parameters $\Theta = \{w_i \in \mathbb{R}, \mu_i \in \mathbb{R}^d, \sigma_i^2 \in \mathbb{R}^d;\ i = 1,\dots,k\}$ from HDFS.

- E-step: compute $\gamma_s(i)$, $\forall\, i = 1, \ldots, k$.
- precompute statistics:
$$S_0(i) = \gamma_s(i), \qquad S_1(i) = \gamma_s(i) x_s, \qquad S_2(i) = \gamma_s(i) x_s^2$$

for$(i = 1, \ldots, k)\{$ *Key out*: i, *Value out*: $(S_0(i), S_1(i), S_2(i), 1)$ $\}$

Combine *Key in*: i , *Value in*: List($S_0(i)_j, S_1(i)_j, S_2(i)_j, m_j \,|\, j = 1, \ldots t$)

- compute statistics: $S_h(i) = \sum_{j=1}^{t} S_h(i)_j$, for $h = 0, 1, 2$.
- compute $m = \sum_{j=1}^{t} m_j$

Key out: i, *Value out*: $(S_0(i), S_1(i), S_2(i), m)$

Reduce *Key in*: i, *Value in*: List($S_0(i)_j, S_1(i)_j, S_2(i)_j, m_j \,|\, j = 1, \ldots t$)

- compute statistics: $S_h(i) = \sum_{j=1}^{t} (S_h(i))_j$, for $h = 0, 1, 2$.
- compute $n = \sum_{j=1}^{t} m_j$
- M-step: compute new GMM parameters
$$w_i^{\text{new}} = \frac{S_0(i)}{n}, \quad \mu_i^{\text{new}} = \frac{S_1(i)}{S_0(i)}, \quad (\sigma_i^2)^{\text{new}} = \frac{S_2(i)}{S_0(i)} - (\mu_i^{\text{new}})^2.$$

Key out: i, *Value out*: $(w_i^{\text{new}}, \mu_i^{\text{new}}, (\sigma_i^2)^{\text{new}})$

At the end of a Map-Reduce phase, the algorithm tests the convergence of μ parameters
$$\|\mu_i^{\text{new}} - \mu_i^{\text{old}}\|_2 < \epsilon$$
and it stops when the above condition is satisfied or the maximum number of iterations is reached.

2.1 Implementation notes: computation in the log domain

When the data points are high-dimensional (as in the case of low-level descriptors of images), the likelihood values $p_i(x)$ and the posterior probabilities $\gamma_i(x)$ can be extremely small. Hence for a stable implementation we perform all the computation in the log domain, as suggested in [3]. In the log domain we have:

$$\log p_i(x_s|\Theta) = -\frac{1}{2} \sum_{l=1}^{d} \left[\log(2\pi) + \log((\sigma_i^2)_l) + \frac{((x_s)_l - (\mu_i)_l)^2}{(\sigma_i^2)_l} \right] \tag{10}$$

$$\log \gamma_s(i) = \log\left(w_i p_i(x_s|\Theta)\right) - \log\left(\sum_{j=1}^{k} w_j p_j(x_s|\Theta) \right). \tag{11}$$

Given $\log\left(w_i p_i(x_s|\Theta)\right)$ for all $i = 1, \ldots, k$, we can compute $\log\left(\sum_{j=1}^{k} w_j p_j(x_s|\Theta)\right)$ using the fact that $\log(a+b) = log(a) + \log\left(1 + \exp(\log b - \log a)\right)$.

3 Tests

We tested our implementation in two scenarios:

Synthetic data. We used MATLAB to generate 2000 points distributed with a mixture of two Gaussians with parameters $w_1 = 0.5$, $\mu_1 = [1, 2]$, $\Sigma_1 = [2, 0; 0, 0.5]$, $w_2 = 0.5$, $\mu_2 = [-3, -5]$, $\Sigma_2 = [1, 0; 0, 1]$ and then we used Hadoop to estimate a Gaussian Mixture Model with $k = 2$. The results obtained are $w_1^* = 0.5$, $\mu_1^* = [0.95, 2.03]$, $\Sigma_1^* = [1.99, 0; 0, 0.5]$, $w_2^* = 0.5$, $\mu_2^* = [-2.96, -4.97]$, $\Sigma_2^* = [1.01, 0; 0, 0.99]$ (see Figure 1).

INRIA Holidays dataset. INRIA Holidays [1] is a collection of 1 491 personal holiday photos. We extracted SIFT local features from images and then we selected two subsets of 1K and 10K local features each. The starting parameters used in EM were computed by using K-means with $k = 64$. Then a mixture of 64 Gaussians was estimated. The obtained GMM could be used, for example, to compute Fisher Vector [3] image representation for image retrieval and classification task.

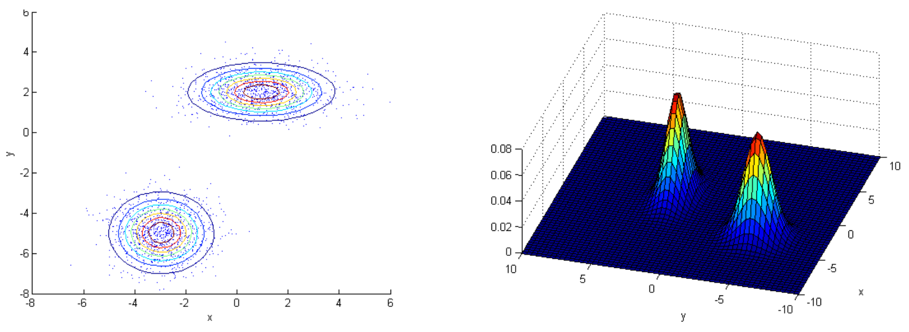

Figure 1: Example of GMM with $k = 2$, estimated on 2000 points of dimension $d = 2$.

References

[1] C. M. Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer, 2006.

[2] D. Lowe, Object recognition from local scale-invariant features, in: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, Vol. 2, 1999, pp. 1150–1157 vol.2.

[3] J. Sànchez, F. Perronnin, T. Mensink, J. Verbeek, Image classification with the fisher vector: Theory and practice, International Journal of Computer Vision 105 (3) (2013) 222–245.

[1] http://lear.inrialpes.fr/ jegou/data.php