Single-tree GMM training

Ryan R. Curtin

May 27, 2015

1 Introduction

In this short document, we derive a tree-independent single-tree algorithm for Gaussian mixture model training, based on a technique proposed by Moore [8]. Here, the solution we provide is tree-independent and thus will work with any type of tree and any type of traversal; this is more general than Moore's original formulation, which was limited to *mrkd*-trees. This allows us to develop a flexible, generic implementation for GMM training of the type commonly found in the **mlpack** machine learning library [3].

A better introduction to Gaussian mixture models, their uses, and their training is given by both [9] and [2]; readers unfamiliar with GMMs should consult those references, as this minor discussion is intended as more of a refresher and also for terminology establishment.

Before describing the single-tree algorithm, assume that we are given a dataset $S = \{p_0, p_1, \ldots, p_n\}$, and we wish to fit a Gaussian mixture model with m components to this data. Each component in our Gaussian mixture model θ is described as $c_j = (\phi_j, \mu_j, \Sigma_j)$ for $j \in [0, m)$, where $\phi_j = P(c_j|\theta)$ is the mixture weight of component j, μ_j is the mean of component j, and Σ_j is the covariance of component j. Then, the probability of a point arising from the GMM θ may be calculated as

$$P(p_i|\theta) = \sum_{j=1}^{m} \omega_j (2\pi \|\Sigma_j\|)^{-1/2} e^{-\frac{1}{2}(p_i - \mu_j)^T \Sigma_j^{-1} (p_i - \mu_j)}. \tag{1}$$

We may also define the probability of a point p_i arising from a particular component in the mixture as

$$a_{ij} := P(p_i|c_j, \theta) = \omega_j (2\pi \|\Sigma_j\|)^{-1/2} e^{-\frac{1}{2}(p_i - \mu_j)^T \Sigma_j^{-1} (p_i - \mu_j)}. \tag{2}$$

Then, we may use Bayes' rule to define

$$\omega_{ij} := P(c_j|p_i, \theta) = \frac{a_{ij}\phi_j}{\sum_k a_{ik}\phi_k}. \tag{3}$$

Often, GMMs are trained using an iterative procedure known as the EM (expectation maximization) algorithm, which proceeds in two steps. In the first

step, we will compute the probability of each point $p_i \in S$ arising from each component (so, we calculate $a_{ij} = P(p_i|c_j, \theta)$ for all $p_i \in S$ and $c_j \in M$). We can then calculate ω_{ij} using the current parameters of the model θ and the already-calculated a_{ij}. Then, in the second step, we update the parameters of the model θ according to the following rules:

$$\phi_j \leftarrow \frac{1}{n} \sum_{i=0}^{n} \omega_{ij}, \tag{4}$$

$$\mu_j \leftarrow \frac{1}{\sum_{i=0}^{n} \omega_{ij}} \sum_{i=0}^{n} \omega_{ij} p_i, \tag{5}$$

$$\Sigma_j \leftarrow \frac{1}{\sum_{i=0}^{n} \omega_{ij}} \sum_{i=0}^{n} \omega_{ij} (p_i - \mu_j)(p_i - \mu_j)^T. \tag{6}$$

Implemented naively and exactly, we must calculate a_{ij} for each p_i and c_j, giving $O(nm)$ operations per iteration. We can do better with trees, although we will introduce some level of approximation.

2 Trees and single-tree algorithms

In scribing this algorithm, we do not want to restrict our description to a single type of tree; for instance, Moore's original formulation is restricted to the *mrkd*-tree [8]. So, instead, we will (re-)introduce a host of definitions that will allow us to describe our algorithm without considering the type of tree, or how it is traversed. These definitions are taken from Curtin et. al. [5].

Definition 1. *A **space tree** on a dataset $S \in \Re^{N \times D}$ is an undirected, connected, acyclic, rooted simple graph with the following properties:*

- *Each node (or vertex), holds a number of points (possibly zero) and is connected to one parent node and a number of child nodes (possibly zero).*
- *There is one node in every space tree with no parent; this is the root node of the tree.*
- *Each point in S is contained in at least one node.*
- *Each node $\mathscr{N}$ has a convex subset of $\Re^D$ containing each point in that node and also the convex subsets represented by each child of the node.*

We will use the same notation for trees:

- The set of child nodes of a node $\mathscr{N}_i$ is denoted $\mathscr{C}(\mathscr{N}_i)$ or $\mathscr{C}_i$.
- The set of points held in a node $\mathscr{N}_i$ is denoted $\mathscr{P}(\mathscr{N}_i)$ or $\mathscr{P}_i$.

- The set of descendant nodes of a node $\mathscr{N}_i$, denoted $\mathscr{D}^n(\mathscr{N}_i)$ or $\mathscr{D}^n_i$, is the set of nodes $\mathscr{C}(\mathscr{N}_i) \cup \mathscr{C}(\mathscr{C}(\mathscr{N}_i)) \cup \ldots$ [1].
- The set of descendant points of a node $\mathscr{N}_i$, denoted $\mathscr{D}^p(\mathscr{N}_i)$ or $\mathscr{D}^p_i$, is the set of points $\{\, p : p \in \mathscr{P}(\mathscr{D}^n(\mathscr{N}_i)) \cup \mathscr{P}(\mathscr{N}_i) \,\}$[2].
- The parent of a node $\mathscr{N}_i$ is denoted $\mathrm{Par}(\mathscr{N}_i)$.

It is often useful to cache particular information in each node of the tree[3]. For the task of Gaussian mixture model training, we will cache the following quantities:

- The number of descendant points of a node. For a node $\mathscr{N}_i$, this is denoted $|\mathscr{D}^p_i|$, in accordance with the notation above.
- The empirical centroid of a node's descendant points. This can be calculated recursively:

$$\mu(\mathscr{N}_i) = \frac{1}{|\mathscr{D}^p_i|}\left(\sum_{p_j \in \mathscr{P}_i} p_j + \sum_{\mathscr{N}_c \in \mathscr{C}_i} |\mathscr{D}^p_c| \mu(\mathscr{N}_c)\right). \tag{7}$$

- The empirical covariance of a node's descendant points. This can also be calculated recursively:

$$C(\mathscr{N}_i) = \frac{1}{|\mathscr{D}^p_i|}\left(\sum_{p_j \in \mathscr{P}_i} (p_j - \mu_i)(p_j - \mu_i)^T + \sum_{\mathscr{N}_c \in \mathscr{C}_i} |\mathscr{D}^p_c| \left(C(\mathscr{N}_c) + \big(\mu(\mathscr{N}_c) - \mu(\mathscr{N}_i)\big)\big(\mu(\mathscr{N}_c) - \mu(\mathscr{N}_i)\big)^T\right)\right). \tag{8}$$

Also, we introduce a few notions relating to distance:

Definition 2. *The **minimum distance** between two nodes $\mathscr{N}_i$ and $\mathscr{N}_j$ is defined as*

$$d_{min}(\mathscr{N}_i, \mathscr{N}_j) = \min\left\{\, \|p_i - p_j\| \,,\; p_i \in \mathscr{D}^p_i, p_j \in \mathscr{D}^p_j \,\right\}.$$

We assume that we can calculate a lower bound on $d_{\min}(\cdot,\cdot)$ quickly (i.e. without checking every descendant point of $\mathscr{N}_i$).

[1] By $\mathscr{C}(\mathscr{C}(\mathscr{N}_i))$ we mean all the children of the children of node $\mathscr{N}_i$: $\mathscr{C}(\mathscr{C}(\mathscr{N}_i)) = \{\mathscr{C}(\mathscr{N}_c) : \mathscr{N}_c \in \mathscr{C}(\mathscr{N}_q)\}$.

[2] The meaning of $\mathscr{P}(\mathscr{D}^n(\mathscr{N}_i))$ is similar to $\mathscr{C}(\mathscr{C}(\mathscr{N}_i))$.

[3] This is the fundamental concept behind the *mrkd*-tree: see [6].

Definition 3. *The **maximum descendant distance** of a node $\mathscr{N}_i$ is defined as the maximum distance between the centroid $\mu(\mathscr{N}_i)$ and points in $\mathscr{D}_i^p$:*

$$\lambda(\mathscr{N}_i) = \max_{p \in \mathscr{D}_i^p} \|C_i - p\|.$$

Next, we must describe the way the tree we build will be traversed. A specification of the below definition might be a depth-first or a breadth-first traversal (or some combination of the two).

Definition 4. *A **pruning single-tree traversal** is a process that, given a space tree, will visit nodes in the tree and perform a computation to assign a score to that node (call this the `Score()` function). If the score is above some bound, the node is "pruned" and none of its descendants will be visited; otherwise, a computation is performed on any points contained within that node (call this the `BaseCase()` function). If no nodes are pruned, then the traversal will visit each node in the tree once.*

Now, we may describe a single-tree algorithm simply by supplying a type of tree, a pruning single-tree traversal, and `BaseCase()` and `Score()` functions. Thus, we may devote the rest of the paper to devising a suitable `BaseCase()` and `Score()` function, and proving its correctness.

3 The single-tree algorithm

Note that for any $p_i \in S$, there is likely to be some component (or many components) c_j such that $P(p_i|c_j, \theta)$ (and therefore $P(c_j|p_i, \theta)$) is quite small. Because $P(c_j|p_i, \theta)$ never decays to 0 for finite $\|p_i - \mu_j\|$, we may not avoid any calculations of ω_{ij} if we want to perform the exact EM algorithm.

However, if we allow some amount of approximation, and can determine (for instance) that $\omega_{ij} < \epsilon$, then we can avoid empirically calculating ω_{ij} and simply approximate it as 0. Further, if we can place a bound such that $\zeta - \epsilon < \omega_{ij} < \zeta + \epsilon$, then we can simply approximate ω_{ij} as ζ.

Now, note that for some node $\mathscr{N}_i$, we may calculate $a_j^{\max}$ for some component j, which is an upper bound on the value of a_{ij} for any point $p_i \in \mathscr{D}^p(\mathscr{N}_i)$:

$$a_j^{\max} = (2\pi\|\Sigma_j\|)^{-1/2} e^{d_{\min}^M(\mathscr{N}_i, \mu_j, \Sigma_j^{-1})} \tag{9}$$

In the equation above, $d^M(\cdot, \cdot, \Sigma^{-1})$ is the Mahalanobis distance:

$$d^M(p_i, p_j, \Sigma^{-1}) = (p_i - p_j)^T \Sigma^{-1} (p_i - p_j) \tag{10}$$

and $d_{\min}^M(\cdot, \cdot, \Sigma^{-1})$ is a generalization of $d_{\min}(\cdot, \cdot)$ to the Mahalanobis distance:

$$d_{\min}^M(\mathscr{N}_i, p_j, \Sigma^{-1}) = \min \left\{ (p_i - p_j)^T \Sigma^{-1} (p_i - p_j) \ , \ p_i \in \mathscr{D}_i^p \right\}. \tag{11}$$

We again assume that we can quickly calculate a lower bound on $d_{\min}^M(\cdot, \cdot, \cdot)$ without checking every descendant point in the tree node. Now, we may use this

Algorithm 1 GMM training `BaseCase()`.

Input: model $\theta = \{(\phi_0, \mu_0, \Sigma_0), \ldots, (\phi_{m-1}, \mu_{m-1}, \Sigma_{m-1})\}$, point p_i, partial model $\theta' = \{(\mu'_0, \Sigma'_0), \ldots, (\mu'_{m-1}, \Sigma'_{m-1})\}$, weight estimates $(\omega_0^t, \ldots, \omega_{m-1}^t)$
Output: updated partial model θ'

{Some trees hold points in multiple places; ensure we don't double-count.}
if point p_i already visited **then return**

{Calculate all a_{ij}.}
for all j in $[0, m)$ **do**
    $a_{ij} \leftarrow (2\pi\|\Sigma_j\|)^{-1/2} e^{-1/2(p_i - \mu_j)^T \Sigma_j^{-1}(p_i - \mu_j)}$
end for
$a_{\text{sum}} \leftarrow \sum_k a_{ik}\phi_k$

{Calculate all ω_{ij} and update model.}
for all j in $[0, m)$ **do**
    $\omega_{ij} \leftarrow \frac{a_{ij}\phi_i}{a_{\text{sum}}}$
    $\omega_j^t \leftarrow \omega_j^t + \omega_{ij}$
    $\mu_j \leftarrow \mu_j + \omega_{ij} p_i$
    $\Sigma_j \leftarrow \Sigma_j + \omega_{ij}(p_i p_i^T)$
end for

return a_{ij}

lower bound to calculate the upper bound $a_j^{\max}$. We may similarly calculate a lower bound $a_j^{\min}$:

$$a_j^{\min} = (2\pi\|\Sigma_j\|)^{-1/2} e^{d_{\max}^M(\mathcal{N}_i, \mu_j, \Sigma_j^{-1})} \tag{12}$$

with $d_{\max}^M(\cdot,\cdot,\cdot)$ defined similarly to $d_{\min}^M(\cdot,\cdot,\cdot)$. Finally, we can use Bayes' rule to produce the bounds $\omega_j^{\min}$ and $\omega_j^{\max}$ (see Equation 3):

$$\omega_j^{\min} = \frac{a_j^{\min}\phi_j}{a_j^{\min}\phi_j + \sum_{k \neq j} a_k^{\max}\phi_k}, \tag{13}$$

$$\omega_j^{\max} = \frac{a_j^{\max}\phi_j}{a_j^{\max}\phi_j + \sum_{k \neq j} a_k^{\min}\phi_k}. \tag{14}$$

Note that in each of these, we must approximate the term $\sum_k a_{ik}\phi_k$, but we do not know the exact values a_{ik}. Thus, for $\omega_j^{\min}$, we must take the bound $a_{ik} \leq a_k^{\max}$, except for when $j = k$, where we can use the tighter $a_j^{\min}$. Symmetric reasoning applies for the case of $\omega_j^{\max}$.

Now, following the advice of Moore [8], we note that a decent pruning rule is to prune if, for all components j, $\omega_j^{\max} - \omega_j^{\min} < \tau\omega_j^t$, where ω_j^t is a lower bound on the total weight that component j has.

Using that intuition, let us define the `BaseCase()` and `Score()` functions that will define our single-tree algorithm. During our single-tree algorithm, we will have the current model θ and a partial model θ', which will hold unnormalized means and covariances of components. After the single-tree algorithm runs, we can normalize θ' to produce the next model θ.

Algorithm 1 defines the `BaseCase()` function and Algorithm 2 defines the `Score()` function. At the beginning of the traversal, we initialize the weight estimates $\omega_0^t, \ldots, \omega_m^t$ all to 0 and the partial model $\theta' = \{(\mu'_0, \Sigma'_0), \ldots, (\mu'_m, \Sigma'_m)\}$ to 0. At the end of the traversal, we will generate our new model as follows, for each component $j \in [0, m)$:

$$\phi_j \leftarrow \frac{1}{n}\omega_j^t \tag{15}$$

$$\mu_j \leftarrow \frac{1}{\omega_j^t}\mu'_j \tag{16}$$

$$\Sigma_j \leftarrow \frac{1}{\omega_j^t}\Sigma'_j \tag{17}$$

After this, the array of ϕ_j values will need to be normalized to sum to 1; this is necessary because each ω_j^t may be approximate.

To better understand the algorithm, let us first consider the `BaseCase()` function. Given some point p_i, our goal is to update the partial model θ' with the contribution of p_i. Therefore, we first calculate a_{ij} for every component $(\phi_j, \mu_j, \Sigma_j)$. This allows us to then calculate ω_{ij} for each component, and then we may update ω_j^t (our lower bound on the total weight of component j) and our partial model components μ'_j and Σ'_j. Note that in the `BaseCase()` function there is no approximation; if we were to call `BaseCase()` with every point in the dataset, we would end up with μ'_j equal to the result of Equation 5 and Σ'_j equal to the result of Equation 6. In addition, ω_j^t would be an exact lower bound.

Now, let us consider `Score()`, which is where the approximation happens. When we visit a node $\mathscr{N}_i$, our goal is to determine whether or not we can approximate the contribution of all of the descendant points of $\mathscr{N}_i$ at once. As stated earlier, we prune if $\omega_j^{\max} - \omega_j^{\min} < \tau\omega_j^t$ for all components j. Thus, the `Score()` function must calculate $\omega_j^{\max}$ and $\omega_j^{\min}$ (lines 4–11) and make sure ω_j^t is updated.

Keeping ω_j^t correct requires a bit of book-keeping. Remember that ω_j^t is a lower bound on $\sum_i \omega_{ij}$; we maintain this bound by using the lower bound $\omega_j^{\min}$ for each descendant point of a particular node. Therefore, when we visit some node $\mathscr{N}_i$, we must remove the parent's lower bound before adding the lower bound produced with the $\omega_j^{\min}$ value for $\mathscr{N}_i$ (lines 13–16).

Because we have defined our single-tree algorithm as only a `BaseCase()` and `Score()` function, we are left with a generic algorithm. We may use any tree and any traversal (so long as it satisfies the definitions given earlier).

Algorithm 2 GMM training `Score()`.

Input: model $\theta = \{(\phi_0, \mu_0, \Sigma_0), \ldots, (\phi_{m-1}, \mu_{m-1}, \Sigma_{m-1})\}$, node $\mathscr{N}_i$, weight estimates $(\omega_0^t, \ldots, \omega_{m-1}^t)$, pruning tolerance τ
Output: ∞ if $\mathscr{N}_i$ can be pruned, score for recursion priority otherwise

{Calculate bounds on a_{ij} for each component.}
for all j in $[0, m)$ **do**
  $a_j^{\min} \leftarrow (2\pi\|\Sigma_j\|)^{-1/2} e^{-1/2(d^M_{\max}(\mathscr{N}_i, \mu_j, \Sigma_j^{-1}))}$
  $a_j^{\max} \leftarrow (2\pi\|\Sigma_j\|)^{-1/2} e^{-1/2(d^M_{\min}(\mathscr{N}_i, \mu_j, \Sigma_j^{-1}))}$
end for

{Calculate bounds on ω_{ij} for each component.}
for all j in $[0, m)$ **do**
  $\omega_j^{\min} \leftarrow \frac{a_j^{\min}\phi_j}{a_j^{\min}\phi_j + \sum_{k\neq j} a_k^{\max}\phi_k}$
  $\omega_j^{\max} \leftarrow \frac{a_j^{\max}\phi_j}{a_j^{\max}\phi_j + \sum_{k\neq j} a_k^{\min}\phi_k}$
  {Remove parent's prediction for ω_j^t contribution from this node.}
  if $\mathscr{N}_i$ is not the root **then**
    $\omega_j^p \leftarrow$ the value of $\omega_j^{\min}$ calculated by the parent
    $\omega_j^t \leftarrow \omega_j^t - |\mathscr{D}^p(\mathscr{N}_i)|\omega_j^p$
  end if
end for

{Determine if we can prune.}
if $\omega_j^{\max} - \omega_j^{\min} < \tau\omega_j^t$ **for all** $j \in [0, m)$ **then**
  {We can prune, so update μ_j and Σ_j.}
  for all j in $[0, m)$ **do**
    $\omega_j^{\text{avg}} \leftarrow 1/2(\omega_j^{\max} + \omega_j^{\min})$
    $\omega_j^t \leftarrow \omega_j^t + |\mathscr{D}^p(\mathscr{N}_i)|\omega_j^{\text{avg}}$
    $c_i \leftarrow$ centroid of $\mathscr{N}_i$
    $\mu_j \leftarrow \mu_j + \omega_j^{\text{avg}} c_i$
    $\Sigma_j \leftarrow \Sigma_j + \omega_j^{\text{avg}} c_i c_i^T$
  end for
  return ∞
end if

{Can't prune; update ω_j^t and return.}
for all $j \in [0, m)$ **do**
  $\omega_j^t \leftarrow \omega_j^t + |\mathscr{D}^p(\mathscr{N}_i)|\omega_j^{\min}$
end for
return $1/(\max_{j\in[0,m)} \omega_j^{\max})$

4 Conclusion

This document has demonstrated how GMM training can be performed approximately with trees. This may be used as a black-box replacement to a single

iteration of the EM algorithm. The algorithm, as given, is generic and can be used with any type of tree. Despite this, there are still several extensions and improvements that may be performed but are not detailed here:

- A better type of approximation. We are only performing relative approximation using the same heuristic as introduced by Moore [8]. But other types of approximation exist: absolute-value approximation [5], or budgeting [7].
- Provable approximation bounds. In this algorithm, the user selects τ to control the approximation, but there is no derived relationship between τ and the quality of the results. A better user-tunable parameter might be something directly related to the quality of the results; for instance, the user might place a bound on the total mean squared error allowed in μ_j and Σ_j for each j.
- Provable worst-case runtime bounds. Using cover trees, a relationship between the properties of the dataset and the runtime may be derived, similar to other tree-based algorithms which use the cover tree [1, 4].
- Caching during the traversal. During the traversal, quantities such as $a_j^{\min}$, $a_j^{\max}$, $\omega_j^{\min}$, and $\omega_j^{\max}$ for a node $\mathscr{N}_i$ will have some geometric relation to those quantities as calculated by the parent of $\mathscr{N}_i$. These relations could potentially be exploited in order to prune a node without evaluating those quantities. This type of strategy is already in use for nearest neighbor search and max-kernel search in **mlpack**.

References

[1] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, pages 97–104, 2006.

[2] J.A. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, Department of Electrical Engineering and Computer Science, University of California, Berekeley.

[3] R.R. Curtin, J.R. Cline, N.P. Slagle, W.B. March, P. Ram, N.A. Mehta, and A.G. Gray. mlpack: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013.

[4] R.R. Curtin, D. Lee, W.B. March, and P. Ram. Plug-and-play dual-tree algorithm runtime analysis. *Journal of Machine Learning Research*, 2015.

[5] R.R. Curtin, W.B. March, P. Ram, D.V. Anderson, A.G. Gray, and C.L. Isbell Jr. Tree-independent dual-tree algorithms. In *Proceedings of The 30th International Conference on Machine Learning (ICML '13)*, pages 1435–1443, 2013.

[6] K. Deng and A.W. Moore. Multiresolution instance-based learning. In *Proceedings of the 1995 International Joint Conference on AI (IJCAI-95)*, volume 95, pages 1233–1239, 1995.

[7] A.G. Gray and A.W. Moore. Nonparametric density estimation: Toward computational tractability. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SDM'03)*, pages 203–211, 2003.

[8] A.W. Moore. Very fast EM-based mixture model clustering using multiresolution *kd*-trees. In *Advances in Neural Information Processing Systems 11 (NIPS 1998)*, pages 543–549. 1999.

[9] D. Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, pages 659–663. 2009.