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 \(\theta\) 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 \(j\), \(\mu_j\) is the mean of component \(j\), and \(\Sigma_j\) is the covariance of component \(j\). Then, the probability of a point arising from the GMM \(\theta\) may be calculated as

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

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

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

Then, we may use Bayes’ rule to define

\[
\omega_{ij} := P(c_j|p_i, \theta) = \frac{a_{ij} \phi_j}{\sum_k a_{ik} \phi_k}.
\] (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 \( \omega_{ij} \) using the current parameters of the model \( \theta \) and the already-calculated \( a_{ij} \). Then, in the second step, we update the parameters of the model \( \theta \) according to the following rules:

\[
\phi_j \leftarrow \frac{1}{n} \sum_{i=0}^{n} \omega_{ij}, \\
\mu_j \leftarrow \frac{1}{\sum_{i=0}^{n} \omega_{ij}} \sum_{i=0}^{n} \omega_{ij} p_i, \\
\Sigma_j \leftarrow \frac{1}{\sum_{i=0}^{n} \omega_{ij}} \sum_{i=0}^{n} \omega_{ij} (p_i - \mu_j)(p_i - \mu_j)^T.
\]

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 \( S \in \mathbb{R}^{N \times D} \) 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 \( \mathcal{N} \) has a convex subset of \( \mathbb{R}^D \) 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 \( \mathcal{N}_i \) is denoted \( \mathcal{C}(\mathcal{N}_i) \) or \( \mathcal{C}_i \).
- The set of points held in a node \( \mathcal{N}_i \) is denoted \( \mathcal{P}(\mathcal{N}_i) \) or \( \mathcal{P}_i \).
The set of descendant nodes of a node \( N_i \), denoted \( D_n(N_i) \) or \( D_n^i \), is the set of nodes \( C(N_i) \cup C(C(N_i)) \cup \ldots \).  

The set of descendant points of a node \( N_i \), denoted \( D_p(N_i) \) or \( D_p^i \), is the set of points \( \{ p : p \in D_n(N_i) \cup D_p(N_i) \} \).

The parent of a node \( N_i \) is denoted \( \text{Par}(N_i) \).

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 \( N_i \), this is denoted \(|D_p^i|\), in accordance with the notation above.
- The empirical centroid of a node’s descendant points. This can be calculated recursively:

\[
\mu(N_i) = \frac{1}{|D_p^i|} \left( \sum_{p_j \in D_p^i} p_j + \sum_{N_c \in C_i} |D_p^c| \mu(N_c) \right). \tag{7}
\]

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

\[
C(N_i) = \frac{1}{|D_p^i|} \left( \sum_{p_j \in D_p^i} (p_j - \mu_i)(p_j - \mu_i)^T + \sum_{N_c \in C_i} |D_p^c| \left( C(N_c) + (\mu(N_c) - \mu(N_i)) (\mu(N_c) - \mu(N_i))^T \right) \right). \tag{8}
\]

Also, we introduce a few notions relating to distance:

**Definition 2.** The minimum distance between two nodes \( N_i \) and \( N_j \) is defined as

\[
d_{\min}(N_i, N_j) = \min \{ \|p_i - p_j\| : p_i \in D_p^i, p_j \in D_p^j \}. \]

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

---

1. By \( C(C(N_i)) \) we mean all the children of the children of node \( N_i \): \( C(C(N_i)) = \{ C(N_c) : N_c \in C(N_i) \} \).
2. The meaning of \( D_n(N_i) \) is similar to \( C(C(N_i)) \).
3. This is the fundamental concept behind the \textit{mixture} tree: see [6].
Definition 3. The maximum descendant distance of a node $N_i$ is defined as the maximum distance between the centroid $\mu(N_i)$ and points in $D_{p_i}$:

$$\lambda(N_i) = \max_{p \in D_{p_i}} \|C_i - p\|.$$ 

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 $\text{Score}()$ 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 $\text{BaseCase}()$ 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 $\text{BaseCase}()$ and $\text{Score}()$ functions. Thus, we may devote the rest of the paper to devising a suitable $\text{BaseCase}()$ and $\text{Score}()$ 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 $\omega_{ij}$ 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 $\omega_{ij}$ and simply approximate it as 0. Further, if we can place a bound such that $\zeta \leq \epsilon < \omega_{ij} < \zeta + \epsilon$, then we can simply approximate $\omega_{ij}$ as $\zeta$.

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

$$a_{ij}^{\text{max}} = (2\pi\|\Sigma_j\|)^{-1/2}e^{d_{\text{min}}^M(N_i, p_j, \Sigma_j^{-1})}.$$ 

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

$$d_M(p_i, p_j, \Sigma^{-1}) = (p_i - p_j)^T\Sigma^{-1}(p_i - p_j)$$ 

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

$$d_{\text{min}}^M(N_i, p_j, \Sigma^{-1}) = \min \{ (p_i - p_j)^T\Sigma^{-1}(p_i - p_j) \mid p_i \in D^p \}.$$ 

We again assume that we can quickly calculate a lower bound on $d_{\text{min}}^M(\cdot, \cdot, \cdot)$ without checking every descendant point in the tree node. Now, we may use this
Algorithm 1 GMM training BaseCase().

1: **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 \( (\omega'_0, \ldots, \omega'_{m-1}) \)

2: **Output:** updated partial model \( \theta' \)

3: {Some trees hold points in multiple places; ensure we don’t double-count.}

4: if point \( p_i \) already visited then return

5: {Calculate all \( a_{ij} \).}

6: for all \( j \) in \([0, m)\) do

7: \( a_{ij} \leftarrow (2\pi \|\Sigma_j\|)^{-1/2} e^{-1/2(p_i - \mu_j)^T \Sigma_j^{-1}(p_i - \mu_j)} \)

8: end for

9: \( a_{\text{sum}} \leftarrow \sum_k a_{ik} \phi_k \)

10: {Calculate all \( \omega_{ij} \) and update model.}

11: for all \( j \) in \([0, m)\) do

12: \( \omega_{ij} \leftarrow \frac{a_{ij} \phi_j}{a_{\text{sum}}} \)

13: \( \omega'_j \leftarrow \omega'_j + \omega_{ij} \)

14: \( \mu_j \leftarrow \mu_j + \omega_{ij} p_i \)

15: \( \Sigma_j \leftarrow \Sigma_j + \omega_{ij} (p_i p_i^T) \)

16: end for

17: return \( a_{ij} \)

lower bound to calculate the upper bound \( a_{j}^{\text{max}} \). We may similarly calculate a lower bound \( a_{j}^{\text{min}} \):

\[
a_{j}^{\text{min}} = (2\pi \|\Sigma_j\|)^{-1/2} e^{d_{j}^{\text{max}}(\cdot, \cdot, \cdot, \cdot)} \]

with \( d_{j}^{\text{max}}(\cdot, \cdot, \cdot, \cdot) \) defined similarly to \( a_{\text{min}}^{\text{max}}(\cdot, \cdot, \cdot, \cdot) \). Finally, we can use Bayes’ rule to produce the bounds \( \omega_{j}^{\text{min}} \) and \( \omega_{j}^{\text{max}} \) (see Equation 3):

\[
\omega_{j}^{\text{min}} = \frac{a_{j}^{\text{min}} \phi_j}{a_{j}^{\text{min}} \phi_j + \sum_{k \neq j} a_{k}^{\text{max}} \phi_k},
\]

\[
\omega_{j}^{\text{max}} = \frac{a_{j}^{\text{max}} \phi_j}{a_{j}^{\text{max}} \phi_j + \sum_{k \neq j} a_{k}^{\text{min}} \phi_k}.
\]

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 \( \omega_{j}^{\text{min}} \), we must take the bound \( a_{ik} \leq a_{k}^{\text{max}} \), except for when \( j = k \), where we can use the tighter \( a_{j}^{\text{min}} \). Symmetric reasoning applies for the case of \( \omega_{j}^{\text{max}} \).

Now, following the advice of Moore [8], we note that a decent pruning rule is to prune if, for all components \( j \), \( \omega_{j}^{\text{max}} - \omega_{j}^{\text{min}} < \tau \omega_j' \), where \( \omega_j' \) 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 \( \theta \) and a partial model \( \theta' \), which will hold unnormalized means and covariances of components. After the single-tree algorithm runs, we can normalize \( \theta' \) to produce the next model \( \theta \).

Algorithm 1 defines the BaseCase() function and Algorithm 2 defines the Score() function. At the beginning of the traversal, we initialize the weight estimates \( \omega_0, \ldots, \omega_m \) 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 \( j \in [0, m) \):

\[
\phi_j \leftarrow \frac{1}{n} \omega_j^t
\]

(15)

\[
\mu_j \leftarrow \frac{1}{\omega_j^t} \mu_j'
\]

(16)

\[
\Sigma_j \leftarrow \frac{1}{\omega_j^t} \Sigma_j'
\]

(17)

After this, the array of \( \phi_j \) values will need to be normalized to sum to 1; this is necessary because each \( \omega_j^t \) 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 \( \theta' \) 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 \( \omega_{ij} \) for each component, and then we may update \( \omega_j^t \) (our lower bound on the total weight of component \( j \)) and our partial model components \( \mu_j' \) and \( \Sigma_j' \). 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 \( \mu_j' \) equal to the result of Equation 5 and \( \Sigma_j' \) equal to the result of Equation 6. In addition, \( \omega_j^t \) would be an exact lower bound.

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

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

1: **Input:** model $\theta = \{(\phi_0, \mu_0, \Sigma_0), \ldots, (\phi_{m-1}, \mu_{m-1}, \Sigma_{m-1})\}$, node $\mathcal{N}_i$, weight estimates $(\omega^0_t, \ldots, \omega^{m-1}_t)$, pruning tolerance $\tau$

2: **Output:** $\infty$ if $\mathcal{N}_i$ can be pruned, score for recursion priority otherwise

3: {Calculate bounds on $a_{ij}$ for each component.}
4: for all $j$ in $[0, m)$ do
5:   $a^\min_j \leftarrow \left(2\pi \|\Sigma_j\|\right)^{-1/2} e^{-1/2(d^b_{\max}(\mathcal{N}_i, \mu_j, \Sigma_j^{-1}))}$
6:   $a^\max_j \leftarrow \left(2\pi \|\Sigma_j\|\right)^{-1/2} e^{-1/2(d^b_{\min}(\mathcal{N}_i, \mu_j, \Sigma_j^{-1}))}$
7: end for

8: {Calculate bounds on $\omega_{ij}$ for each component.}
9: for all $j$ in $[0, m)$ do
10:   $\omega^\min_j \leftarrow \frac{a^\min_j \phi_j + \sum_k \omega^\min_k \phi_k}{a^\max_j \phi_j + \sum_k \omega^\min_k \phi_k}$
11:   $\omega^\max_j \leftarrow \frac{a^\max_j \phi_j + \sum_k \omega^\min_k \phi_k}{a^\min_j \phi_j + \sum_k \omega^\min_k \phi_k}$
12: {Remove parent’s prediction for $\omega^t_j$ contribution from this node.}
13: if $\mathcal{N}_i$ is not the root then
14:   $\omega^p_j \leftarrow$ the value of $\omega^\min_j$ calculated by the parent
15: $\omega^t_j \leftarrow \omega^t_j - |\mathcal{D}^p(\mathcal{N}_i)| \omega^p_j$
16: end if
17: end for

18: {Determine if we can prune.}
19: if $\omega^\max_j - \omega^\min_j < \tau \omega^t_j$ for all $j \in [0, m)$ then
20:   {We can prune, so update $\mu_j$ and $\Sigma_j$.}
21:   for all $j$ in $[0, m)$ do
22:     $\omega^\text{avg} \leftarrow 1/2(\omega^\max_j + \omega^\min_j)$
23:     $\omega^t_j \leftarrow \omega^t_j + |\mathcal{D}^p(\mathcal{N}_i)| \omega^\text{avg}$
24:     $c_i \leftarrow$ centroid of $\mathcal{N}_i$
25:     $\mu_j \leftarrow \mu_j + \omega^\text{avg} c_i$
26:     $\Sigma_j \leftarrow \Sigma_j + \omega^\text{avg} c_i c_i^T$
27:   end for
28:   return $\infty$
29: end if
30: {Can’t prune; update $\omega^t_j$ and return.}
31: for all $j \in [0, m)$ do
32:   $\omega^t_j \leftarrow \omega^t_j + |\mathcal{D}^p(\mathcal{N}_i)| \omega^\min_j$
33: end for
34: return $1/\left(\max_{j \in [0, m)} \omega^\max_j\right)$

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 $\tau$ to control the approximation, but there is no derived relationship between $\tau$ 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 $\mu_j$ and $\Sigma_j$ 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 $a^{\min}_j$, $a^{\max}_j$, $\omega^{\min}_j$, and $\omega^{\max}_j$ for a node $N_i$ will have some geometric relation to those quantities as calculated by the parent of $N_i$. 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


