Approximate Gaussian Mixtures for Large Scale Vocabularies

Yannis Avrithis and Yannis Kalantidis

National Technical University of Athens
{iavr,ykalant}@image.ntua.gr

Abstract. We introduce a clustering method that combines the flexibility of Gaussian mixtures with the scaling properties needed to construct visual vocabularies for image retrieval. It is a variant of expectation-maximization that can converge rapidly while dynamically estimating the number of components. We employ approximate nearest neighbor search to speed-up the E-step and exploit its iterative nature to make search incremental, boosting both speed and precision. We achieve superior performance in large scale retrieval, being as fast as the best known approximate k-means.

Keywords: Gaussian mixtures, expectation-maximization, visual vocabularies, large scale clustering, approximate nearest neighbor search.

1 Introduction

The *bag-of-words* (BoW) model is ubiquitous in a number of problems of computer vision, including classification, detection, recognition, and retrieval. The *k-means* algorithm is one of the most popular in the construction of *visual vocabularies*, or *codebooks*. The investigation of alternative methods has evolved into an active research area for small to medium vocabularies up to 10^4 visual words. For problems like image retrieval using local features and descriptors, finer vocabularies are needed, *e.g.* 10^6 visual words or more. Clustering options are more limited at this scale, with the most popular still being variants of k-means like *approximate k-means* (AKM) [1] and *hierarchical k-means* (HKM) [2].

The *Gaussian mixture model* (GMM), along with *expectation-maximization* (EM) [3] learning, is a generalization of k-means that has been applied to vocabulary construction for class-level recognition [4]. In addition to position, it models cluster population and shape, but assumes pairwise 'interaction' of all points with all clusters and is slower to converge. The complexity per iteration is $O(NK)$ where N and K is the number of points and clusters, respectively, so it is not practical for large K. On the other hand, a point is assigned to the nearest cluster via *approximate nearest neighbor* (ANN) search in [1], bringing complexity down to $O(N \log K)$, but keeping only one neighbor per point.

Robust approximate k-means (RAKM) [5] is an extension of AKM where the nearest neighbor in one iteration is re-used in the next, with less effort being spent for new neighbor search. This approach yields further speed-up, since

A. Fitzgibbon et al. (Eds.): ECCV 2012, Part III, LNCS 7574, pp. 15–28, 2012.

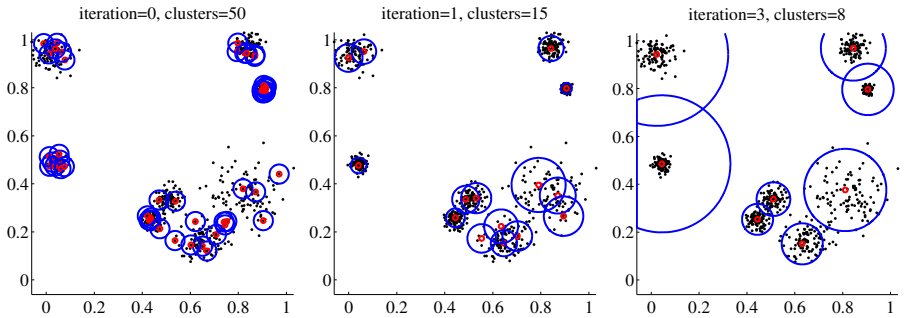

Fig. 1. Estimating the number, population, position and extent of clusters on an 8-mode 2d Gaussian mixture sampled at 800 points, *in just* 3 *iterations* (iteration 2 not shown). Red circles: cluster centers; blue: two standard deviations. Clusters are initialized on 50 data points sampled at random, with initial $\sigma = 0.02$. Observe the 'space-filling' behavior of the two clusters on the left.

the dimensionality D of the underlying space is high, *e.g.* 64 or 128, and the cost per iteration is dominated by the number of vector operations spent for distance computation to potential neighbors. The above motivate us to keep a larger, fixed number m of nearest neighbors across iterations. This not only improves ANN search in the sense of [5], but makes available enough information for an *approximate Gaussian mixture* (AGM) model, whereby each data point 'interacts' only with the m nearest clusters.

Experimenting with overlapping clusters, we have come up with two modifications that alter the EM behavior to the extent that it appears to be an entirely new clustering algorithm. An example is shown in Figure 1. Upon applying EM with relatively large K, the first modification is to compute the overlap of neighboring clusters and *purge* the ones that appear redundant, after each EM iteration. Now, clusters neighboring to the ones being purged should fill in the resulting space, so the second modification is to *expand* them as much as possible. The algorithm shares properties with other methods like agglomerative or density-based mode seeking. In particular, it can *dynamically estimate the number of clusters* by starting with large K and purging clusters along the way. On the other hand, expanding towards empty space apart from underlying data points *boosts convergence rate*.

This algorithm, *expanding Gaussian mixtures* (EGM), is quite generic and can be used in a variety of applications. Focusing on spherical Gaussians, we apply its approximate version, AGM, to large scale visual vocabulary learning for image retrieval. In this setting, contrary to typical GMM usage, descriptors of indexed images are assigned only to their nearest visual word to keep the index sparse enough. This is the first implementation of Gaussian mixtures at this scale, and it appears to enhance retrieval performance at no additional cost compared to approximate k-means.

2 Related Work

One known pitfall of *k-means* is that visual words cluster around the densest few regions of the descriptor space, while the sparse ones may be more informative [6]. In this direction, radius-based clustering is used in [7], while [8], [9] combine the partitional properties of k-means with *agglomerative clustering*. Our approximate approach is efficient enough to initialize from *all* data points, preserving even sparse regions of the descriptor space, and to *dynamically purge* as appropriate in dense ones.

Precise visual word region *shape* has been represented by *Gaussian mixture models* (GMM) for universal [10] and class-adapted codebooks [4], and even by *one-class SVM* classifiers [11]. GMMs are extended with spatial information in [12] to represent images by *hyperfeatures*. None of the above can scale up as needed for image retrieval. On the other extreme, *training-free* approaches like fixed quantization on a *regular lattice* [13] or *hashing* with random histograms [14] are remarkably efficient for recognition tasks but often fail when applied to retrieval [15]. The flat vocabulary of AKM [1] is the most popular in this sense, accelerating the process by the use of *randomized k-d trees* [16] and outperforming the hierarchical vocabulary of HKM [2], but RAKM [5] is even faster. Although constrained to spherical Gaussian components, we achieve *superior flexibility* due to GMM, yet converging as fast as [5].

Assigning an input descriptor to a visual word is also achieved by nearest neighbor search, and randomized k-d trees [16] are the most popular, in particular the FLANN implementation [17]. We use FLANN but we exploit the iterative nature of EM to make the search process *incremental*, boosting both speed and precision. Any ANN search method would apply equally, *e.g. product quantization* [18], as long as it returns multiple neighbors and its training is fast enough to repeat within each EM iteration.

One attempt to compensate for the information loss due to quantization is *soft assignment* [15]. This is seen as *kernel density estimation* and applied to small vocabularies for recognition in [19], while different *pooling* strategies are explored in [20]. Soft assignment is expensive in general, particularly for recognition, but may apply to the query only in retrieval [21], which is our choice as well. This functionality is moved inside the codebook in [22] using a blurring operation, while visual word *synonyms* are learned from an extremely fine vocabulary in [23], using geometrically verified *feature tracks*. Towards the other extreme, *Hamming embedding* [24] encodes approximate position in Voronoi cells of a coarser vocabulary, but this also needs more index space. Our method is complementary to learning synonyms or embedding more information.

Estimating the number of components is often handled by multiple EM runs and *model selection* or *split and merge* operations [25], which is impractical in our problem. A dynamic approach in a single EM run is more appropriate, *e.g. component annihilation* based on competition [26]; we rather estimate spatial overlap directly and purge components long before competition takes place. Approximations include *e.g.* space partitioning with k-d trees and group-wise parameter learning [27], but ANN in the E-step yields lower complexity.

3 Expanding Gaussian Mixtures

We begin with an overview of Gaussian mixture models and parameter learning via EM, mainly following [3], in section 3.1. We then develop our purging and expanding schemes in sections 3.2 and 3.3, respectively. Finally, we discuss initializing and terminating in the specific context of visual vocabularies.

3.1 Parameter Learning

The density $p(\mathbf{x})$ of a Gaussian mixture distribution is a convex combination of K D-dimensional normal densities or *components*,

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1}$$

for $\mathbf{x} \in \mathbb{R}^D$, where $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ are the *mixing coefficient*, *mean* and *covariance matrix* respectively of the k-th component. Interpreting π_k as the prior probability $p(k)$ of component k, and given observation $\mathbf{x}$, quantity

$$\gamma_k(\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{2}$$

for $\mathbf{x} \in \mathbb{R}^D, k = 1, \ldots, K$, expresses the posterior probability $p(k|\mathbf{x})$; we say that $\gamma_k(\mathbf{x})$ is the responsibility that component k takes for 'explaining' observation $\mathbf{x}$, or just the *responsibility of k for* $\mathbf{x}$. Now, given a set of i.i.d. observations or data points $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the *maximum likelihood* (ML) estimate for the parameters of each component $k = 1, \ldots, K$ is [3]

$$\pi_k = \frac{N_k}{N} \tag{3}$$

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} \mathbf{x}_n \tag{4}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}, \tag{5}$$

where $\gamma_{nk} = \gamma_k(\mathbf{x}_n)$ for $n = 1, \ldots, N$, and $N_k = \sum_{n=1}^{N} \gamma_{nk}$, to be interpreted as the *effective number* of points assigned to component k. The *expectation-maximization* (EM) algorithm involves an iterative learning process: given an initial set of parameters, compute responsibilities γ_{nk} according to (2) (E-step); then, re-estimate parameters according to (3)-(5), keeping responsibilities fixed (M-step). Initialization and termination are discussed in section 3.4.

For the remaining of this work, we shall be focusing on the particular case of *spherical* (*isotropic*) Gaussians, with covariance matrix $\boldsymbol{\Sigma}_k = \sigma_k^2 \mathbf{I}$. In this case, update equation (5) reduces to

$$\sigma_k^2 = \frac{1}{DN_k} \sum_{n=1}^{N} \gamma_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2. \tag{6}$$

Comparing to standard k-means, the model is still more flexible in terms of mixing coefficients π_k and standard deviations σ_k, expressing the population and extent of clusters. Yet, the model is efficient because σ_k is a scalar, whereas the representation of full covariance matrices $\mathbf{\Sigma}_k$ would be quadratic in D.

3.2 Purging

Determining the number of components is more critical in Gaussian mixtures than in k-means due to the possibility of overlapping components. Although one solution is a *variational approach* [3], the problem has motivated us to devise a novel method of *purging* components according to an *overlap* measure. Purging is *dynamic* in the sense that we initialize the model with as many components as possible and purge them as necessary during the parameter learning process. In effect, this idea introduces a *P-step* in EM, that is to be applied right after the E- and M-steps in every iteration.

Let p_k be the function representing the contribution of component k to the Gaussian mixture distribution of (1), with

$$p_k(\mathbf{x}) = \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \tag{7}$$

for $\mathbf{x} \in \mathbb{R}^D$. Then p_k can the treated as a representation of component k itself. Note that p_k is not a normalized distribution unless $\pi_k = 1$. Also, let

$$\langle p, q \rangle = \int p(\mathbf{x}) q(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{8}$$

be the L^2 *inner product* of real-valued, square-integrable functions p, q—again not necessarily normalized—where the integral is assumed to be over $\mathbb{R}^D$. The corresponding L^2 *norm* of function p is given by $\|p\| = \sqrt{\langle p, p \rangle}$. When p, q are normal distributions, the integral in (8) can be evaluated in *closed form*.

Theorem 1. *Let* $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{a}, \mathbf{A})$ *and* $q(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{B})$ *for* $\mathbf{x} \in \mathbb{R}^D$. *Then*

$$\langle p, q \rangle = \mathcal{N}(\mathbf{a}|\mathbf{b}, \mathbf{A} + \mathbf{B}). \tag{9}$$

Hence, given components represented by p_i, p_k, their *overlap* in space, as measured by inner product $\langle p_i, p_k \rangle$, is

$$\langle p_i, p_k \rangle = \pi_i \pi_k \mathcal{N}(\boldsymbol{\mu}_i|\boldsymbol{\mu}_k, (\sigma_i^2 + \sigma_k^2)\mathbf{I}) \tag{10}$$

under the spherical Gaussian model. It can be computed in $O(D)$, requiring only one D-dimensional vector operation $\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_k\|^2$, while squared norm $\|p_i\|^2 = \langle p_i, p_i \rangle = \pi_i^2 (4\pi\sigma_i^2)^{-D/2}$ is $O(1)$. Now, if function q represents any component or cluster, (10) motivates generalizing (2) to define quantity

$$\hat{\gamma}_k(q) = \frac{\langle q, p_k \rangle}{\sum_{j=1}^{K} \langle q, p_j \rangle}, \tag{11}$$

Algorithm 1: Component purging (P-step)

input : set of components $\mathcal{C} \subseteq \{1, \dots, K\}$ at current iteration
output: updated set of components $\mathcal{C}' \subseteq \mathcal{C}$, after purging

$\mathcal{K} \leftarrow \emptyset$ // set of components to keep
SORT $\mathcal{C}$ such that $i < k \rightarrow \pi_i \geq \pi_k$ for $i, k \in \mathcal{C}$ // re-order components $i \dots$
foreach $i \in \mathcal{C}$ **do** // $\dots$ in descending order of π_i
  if $\rho_{i,\mathcal{K}} \geq \tau$ **then** // compute $\rho_{i,\mathcal{K}}$ by (12)
    $\mathcal{K} \leftarrow \mathcal{K} \cup i$ // keep i if it does not overlap with $\mathcal{K}$
$\mathcal{C}' \leftarrow \mathcal{K}$ // updated components

so that $\hat{\gamma}_{ik} = \hat{\gamma}_k(p_i) \in [0, 1]$ is the *generalized responsibility* of component k for *component* i. Function p_i is treated here as a *generalized data point*, centered at μ_i, weighted by coefficient π_i and having spatial extent σ_i to represent the underlying actual data points. Observe that (11) reduces to (2) when q collapses to a Dirac delta function, effectively *sampling* component functions p_k.

According to our definitions, $\hat{\gamma}_{ii}$ is the responsibility of component i for *itself*. More generally, given a set $\mathcal{K}$ of components and one component $i \notin \mathcal{K}$, let

$$\rho_{i,\mathcal{K}} = \frac{\hat{\gamma}_{ii}}{\hat{\gamma}_{ii} + \sum_{j \in \mathcal{K}} \hat{\gamma}_{ij}} = \frac{\|p_i\|^2}{\|p_i\|^2 + \sum_{j \in \mathcal{K}} \langle p_i, p_j \rangle}. \tag{12}$$

Quantity $\rho_{i,\mathcal{K}} \in [0, 1]$ is the responsibility of component i for itself *relative to* $\mathcal{K}$. If $\rho_{i,\mathcal{K}}$ is large, component i can 'explain' itself better than set $\mathcal{K}$ *as a whole*; otherwise i appears to be redundant. So, if $\mathcal{K}$ represents the components we have decided to *keep* so far, it makes sense to purge component i if $\rho_{i,\mathcal{K}}$ drops below *overlap threshold* $\tau \in [0, 1]$, in which case we say that i *overlaps* with $\mathcal{K}$.

We choose to process components in descending order of mixing coefficients, starting from the most populated cluster, which we always keep. We then *keep* a component unless it overlaps with the ones we have already kept; otherwise we *purge* it. Note that $\tau \geq \frac{1}{2}$ ensures we cannot 'keep' two identical components. This *P-step* is outlined in Algorithm 1, which assumes that $\mathcal{C} \subseteq \{1, \dots, K\}$ holds the components of the current iteration. Now K refers to the initial number of components; the *effective number*, $C = |\mathcal{C}| \leq K$, decreases in each iteration. This behavior resembles agglomerative clustering—in fact, inner product (8) is analogous to the *average linkage* or *group average* criterion [28], which however measures dissimilarity and is discrete.

We perform the P-step right after E- and M-steps in each EM iteration. We also modify the M-step to re-estimate π_k, $\boldsymbol{\mu}_k$, σ_k according to (3), (4), (6) only for $k \in \mathcal{C}$; similarly, in the E-step, compute $\gamma_{nk} = \gamma_k(\mathbf{x}_n)$ for all $n = 1, \dots, N$ but only $k \in \mathcal{C}$, with

$$\gamma_k(\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j \in \mathcal{C}} \pi_j \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}, \tag{13}$$

where the sum in the denominator of (2) has been constrained to $\mathcal{C}$.

3.3 Expanding

When a component, say i, is purged, data points that were better 'explained' by i prior to purging will have to be assigned to neighboring components that remain. These components will then have to *expand* and cover the space populated by such points. Towards this goal, we modify (6) to *overestimate* the extent of each component as much as this does not overlap with its neighboring components. Components will then tend to fill in as much empty space as possible, and this determines the convergence rate of the entire algorithm.

More specifically, given component k in the current set of components $\mathcal{C}$, we use the constrained definition of $\gamma_k(\mathbf{x})$ in (13) to partition the data set $\mathcal{P} = \{1, \ldots, N\}$ into the set of its *inner points*

$$\underline{\mathcal{P}}_k = \{n \in \mathcal{P} : \gamma_{nk} = \max_{j \in \mathcal{C}} \gamma_{nj}\}, \tag{14}$$

contained in its *Voronoi cell*, and *outer points*, $\overline{\mathcal{P}}_k = \mathcal{P} \setminus \underline{\mathcal{P}}_k$. We now observe that re-estimation equation (6) can be decomposed into

$$D\sigma_k^2 = \frac{\underline{N}_k}{N_k}\underline{\Sigma}_k + \frac{\overline{N}_k}{N_k}\overline{\Sigma}_k, \tag{15}$$

where $\underline{N}_k = \sum_{n \in \underline{\mathcal{P}}_k} \gamma_{nk}$, $\overline{N}_k = \sum_{n \in \overline{\mathcal{P}}_k} \gamma_{nk}$, and

$$\underline{\Sigma}_k = \frac{1}{\underline{N}_k} \sum_{n \in \underline{\mathcal{P}}_k} \gamma_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2, \quad \overline{\Sigma}_k = \frac{1}{\overline{N}_k} \sum_{n \in \overline{\mathcal{P}}_k} \gamma_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2. \tag{16}$$

Since $\underline{N}_k + \overline{N}_k = N_k$, the weights in the linear combination of (15) sum to one. The *inner sum* $\underline{\Sigma}_k$ expresses a weighted average distance from $\boldsymbol{\mu}_k$ of data points that are better 'explained' by component k, hence fits the underlying data of the corresponding cluster. On the other hand, the *outer sum* $\overline{\Sigma}_k$ plays a similar role for points 'explained' by the remaining components. Thus typically $\underline{\Sigma}_k < \overline{\Sigma}_k$ though this is not always true, especially in cases of excessive component overlap. Now, the trick is to *bias* the weighted sum towards $\overline{\Sigma}_k$ in (15) as follows,

$$D\sigma_k^2 = w_k\underline{\Sigma}_k + (1 - w_k)\overline{\Sigma}_k, \tag{17}$$

where $w_k = \frac{\underline{N}_k}{N_k}(1 - \lambda)$ and $\lambda \in [0, 1]$ is an *expansion factor*, with $\lambda = 0$ reducing (17) back to (6) and $\lambda = 1$ using only the outer sum $\overline{\Sigma}_k$.

Because of the exponential form of the normal distribution, the outer sum $\overline{\Sigma}_k$ is dominated by the outer points in $\overline{\mathcal{P}}_k$ that are nearest to component k, hence it provides an estimate for the maximum expansion of k before overlaps with neighboring components begin. Figure 2 illustrates expansion for the example of Figure 1. The outer green circles typically pass through the nearest clusters, while the inner ones tightly fit the data points of each cluster.

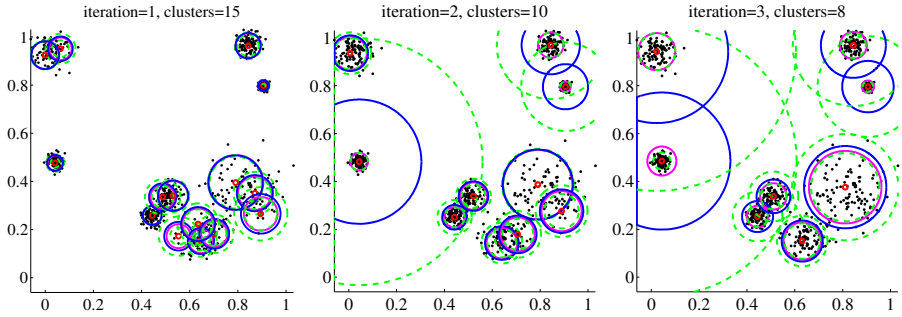

Fig. 2. Component expansion for iterations 1, 2 and 3 of the example of Figure 1. Blue circles: two standard deviations with expansion (17) and $\lambda = 0.25$, as in Figure 1; magenta: without expansion (6); dashed green: inner and outer sum contributions.

3.4 Initializing and Terminating

It has been argued [7][6] that sparsely populated regions of the descriptor space are often the most informative ones, so random sampling is usually a bad choice. We therefore initialize with *all* data points as cluster centers, that is, $K = N$. Using approximate nearest neighbors, this choice is not as inefficient as it sounds. In fact, only the first iteration is affected because by the second, the ratio of clusters that survive is typically in the order of 10%. *Mixing coefficients* are uniform initially. *Standard deviations* are initialized to the distance of the nearest neighbor, again found approximately.

Convergence in EM is typically detected by monitoring the *likelihood function*. This makes sense after the number of components has stabilized and no purging takes place. However, experiments on large scale vocabularies take hours or even days of processing, so convergence is never reached in practice. What is important is to measure the performance of the resulting vocabulary in a particular task—retrieval in our case—versus processing required.

4 Approximate Gaussian Mixtures

Counting D-dimensional vector operations and ignoring iterations, the complexity of the algorithm presented so far is $O(NK)$. In particular, the complexity of the E-step (13) and M-step (3), (4), (16)-(17) of each iteration is $O(NC)$, where $C = |\mathcal{C}| \leq K \leq N$ is the current number of components, and the complexity of the P-step (Algorithm 1) is $O(C^2)$. This is clearly not practical for large C, especially when K is in the order of N.

Similarly to [1], the approximate version of our Gaussian mixtures clustering algorithm involves indexing the entire set of clusters $\mathcal{C}$ according to their center $\boldsymbol{\mu}_k$ and performing an approximate nearest neighbor query for each data point $\mathbf{x}_n$, prior to the E-step of each iteration. The former step is $O(C\alpha(C))$ and the

Algorithm 2: Incremental m-nearest neighbors (N-step)

input : m best neighbors $\mathcal{B}(\mathbf{x}_n)$ found so far for $n = 1, \ldots, N$
output: updated m best neighbors $\mathcal{B}'(\mathbf{x}_n)$ for $n = 1, \ldots, N$

for $n = 1, \ldots, N$ **do** // for all data points
  $\mathcal{B}(\mathbf{x}_n) \leftarrow \mathcal{C} \cap \mathcal{B}(\mathbf{x}_n)$ // ignore purged neighbors
  $(\mathcal{R}, d) \leftarrow \mathrm{NN}_m(\mathbf{x}_n)$ // $\mathcal{R}$: m-NN of $\mathbf{x}_n$; d: distances to $\mathbf{x}_n$
  // (such that $d_k^2 = \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$ for $k \in \mathcal{R}$)
  for $k \in \mathcal{B}(\mathbf{x}_n) \setminus \mathcal{R}$ **do** // for all previous neighbors...
    $d_k^2 \leftarrow \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$ // ...find distance after $\boldsymbol{\mu}_k$ update (M-step)
  $\mathcal{A} \leftarrow \mathcal{B}(\mathbf{x}_n) \cup \mathcal{R}$ // for all previous and new neighbors...
  for $k \in \mathcal{A}$ **do** // ...compute unnormalized...
    $g_k \leftarrow (\pi_k / \sigma_k^D) \exp\{-d_k^2/(2\sigma_k^2)\}$ // ...responsibility of k for $\mathbf{x}_n$
  Sort $\mathcal{A}$ such that $i < k \rightarrow g_i \geq g_k$ for $i, k \in \mathcal{A}$ // keep the top-ranking...
  $\mathcal{B}'(\mathbf{x}_n) \leftarrow \mathcal{A}[1, \ldots, m]$ // ...m neighbors

latter $O(N\alpha(C))$, where α expresses the complexity of a query as a function of the indexed set size; *e.g.* $\alpha(C) = \log C$ for typical tree-based methods. Responsibilities γ_{nk} are thus obtained according to (13) for $n = 1, \ldots, N$, $k \in \mathcal{C}$, but with distances to cluster centers effectively replaced by metric

$$d_m^2(\mathbf{x}, \boldsymbol{\mu}_k) = \begin{cases} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2, & \text{if } k \in \mathrm{NN}_m(\mathbf{x}) \\ 0, & \text{otherwise,} \end{cases} \tag{18}$$

where $\mathrm{NN}_m(\mathbf{x}) \subseteq \mathcal{C}$ denotes the approximate m-nearest neighborhood of query point $\mathbf{x} \in \mathbb{R}^D$. Each component k found as a nearest neighbor of data point $\mathbf{x}_n$ is subsequently updated by computing the contributions γ_{nk}, $\gamma_{nk}\mathbf{x}_n$, $\gamma_{nk}\|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$ to N_k (hence π_k in (3)), $\boldsymbol{\mu}_k$ in (4), σ_k^2 in (16)-(17), respectively, that are due to $\mathbf{x}_n$. The M-step thus brings no additional complexity. Similarly, the P-step involves a query for each cluster center $\boldsymbol{\mu}_k$, with complexity $O(C\alpha(C))$. It follows that the overall complexity per iteration is $O(N\alpha(C))$. In the case of FLANN, $\alpha(C)$ is constant and equal to the number of *leaf checks* per query, since we are only counting vector operations and each split node operation is scalar.

Now, similarly to [5], we not only use approximate search to speed up clustering, but we also exploit the iterative nature of the clustering algorithm to enhance the search process itself. To this end, we maintain a list of the m *best neighbors* $\mathcal{B}(\mathbf{x}_n)$ found so far for each data point $\mathbf{x}_n$, and re-use it across iterations. The distance of each new nearest neighbor is readily available as a by-product of the query at each iteration, while distances of previous neighbors have to be re-computed after update of cluster centers (4) at the M-step. The list of best neighbors is updated as outlined in Algorithm 2.

This *incremental m-nearest neighbors* algorithm is a generalization of [5], which restricts to $m = 1$ and k-means only. It may be considered an *N-step* in the overall approximate clustering algorithm, to be performed prior to the E-step—in fact, providing responsibilities as a by-product. The additional cost

is m vector operations for distance computation at each iteration, but this is compensated for by reducing the requested precision for each query. As in [5], the rationale is that by keeping track of the best neighbors found so far, we may significantly reduce the effort spent in searching for new ones.

5 Experiments

5.1 Setup

We explore here the behavior of AGM under different settings and parameters, and compare its speed and performance against AKM [1] and RAKM [5] on large scale vocabulary construction for image retrieval.

Datasets. We use two publicly available datasets, namely *Oxford buildings*[1] [1] and *world cities*[2] [29]. The former consists of $5,062$ images and annotation (ground truth) for 11 different landmarks in Oxford, with 5 queries for each. The annotated part of the latter is referred to as the *Barcelona* dataset and consists of 927 images grouped into 17 landmarks in Barcelona, again with 5 queries for each. The distractor set of *world cities* consists of 2M images from different cities; we use the first one million as distractors in our experiments.

Vocabularies. We extract SURF [30] features and descriptors from each image. There are in total 10M descriptors in *Oxford buildings* and 550K in *Barcelona*. The descriptors, of dimensionality $D = 64$, are the data points we use for training. We construct *specific* vocabularies with all descriptors of the smaller *Barcelona* dataset, which we then evaluate on the same dataset for parameter tuning and speed comparisons. We also build *generic* vocabularies with all 6.5M descriptors of an independent set of 15K images depicting urban scenes, and compare on the larger *Oxford buildings* in the presence of distractors.

Evaluation protocol. Given each vocabulary built, we use FLANN to assign descriptors, keeping only one NN under Euclidean distance. We apply *soft assignment* on the query side only as in [21], keeping the first 1, 3, or 5 NNs. These are found as in [15] with $\sigma = 0.01$. We rank by dot product on ℓ^2-normalized BoW histograms and *tf-idf* weighting. We measure training time in D-dimensional *vector operations* (vop) per data point and retrieval performance in *mean average precision* (mAP) over all queries. We use our own C++ implementations for AKM and RAKM.

5.2 Tuning

We choose *Barcelona* for parameter tuning because, combined with descriptor assignment, it would be prohibitive to use a larger dataset. It turns out however, that AGM outperforms the state of the art at much larger scale with the same parameters. The behavior of EGM is controlled by *expansion factor* λ and *overlap threshold* τ; AGM is also controlled by *memory* m that determines ANN precision and cost, considered in section 5.3.

[1] `http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/`
[2] `http://image.ntua.gr/iva/datasets/wc/`

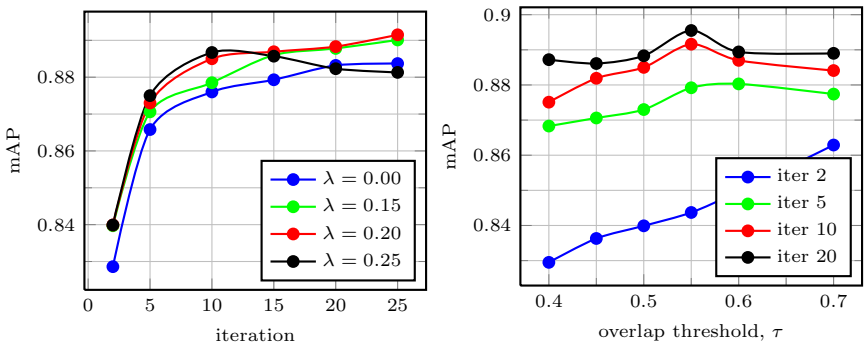

Fig. 3. *Barcelona*-specific parameter tuning. (left) mAP performance versus iteration during learning, for varying *expansion factor* λ and fixed $\tau = 0.5$. (right) mAP versus *overlap threshold* τ for different iterations, with fixed $\lambda = 0.2$.

Figure 3 (left) shows mAP against number of iterations for different values of λ. Compared to $\lambda = 0$, it is apparent that expansion *boosts convergence rate*. However, the effect is only temporary above $\lambda = 0.2$, and performance eventually drops, apparently due to over-expansion. Now, τ controls purging, and we expect $\tau \geq 0.5$ as pointed out in section 3.2. Since $\rho_{i,\mathcal{K}}$ is normalized, $\tau = 0.5$ appears to be a good choice, as component k 'explains' itself at least as much as $\mathcal{K}$ in (12). Figure 3 (right) shows that the higher τ the better initially, but eventually $\tau = 0.55$ is clearly optimal, being somewhat stricter than expected. We choose $\lambda = 0.2$ and $\tau = 0.55$ for the remaining experiments.

5.3 Comparisons

We use FLANN [17] for all methods, with 15 trees and precision controlled by *checks*, *i.e.* the number of leaves checked per ANN query. We use 1,000 checks during assignment, and less during learning. Approximation in AGM is controlled by *memory* m, and we need m FLANN checks plus at most m more distance computations in Algorithm 2, for a total of $2m$ vector operations per iteration. We therefore use $2m$ checks for AKM and RAKM in comparisons. Figure 4 (left) compares all methods on convergence for $m = 50$ and $m = 100$, where AKM/RAKM are trained for 80K vocabularies, found to be optimal.

AGM not only converges as fast as AKM and RAKM, but also outperforms them for the same learning time. AKM and RAKM are only better in the first few iterations, because recall that AGM initializes from *all* points and needs a few iterations to reach a reasonable vocabulary size. The relative performance of RAKM versus AKM is not quite as expected by [5]; this may be partly due to *random initialization*, a standing issue for k-means that AGM is free of, and partly because performance in [5] is measured in *distortion* rather than mAP in retrieval. It is also interesting that, unlike the other methods, AGM appears to improve in mAP for lower m.

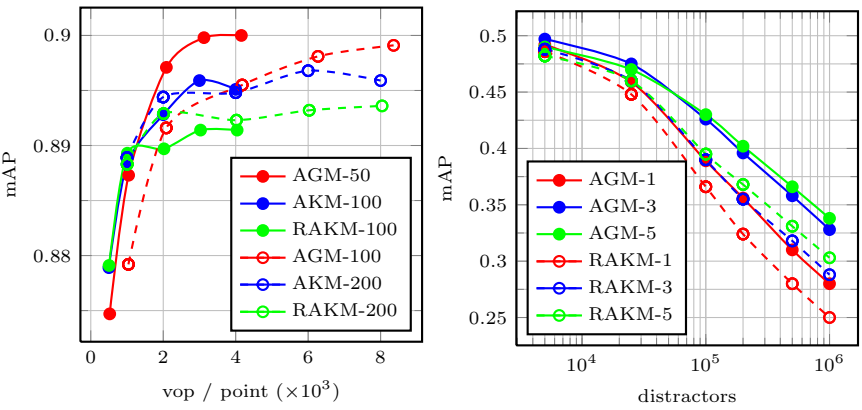

Fig. 4. (Left) *Barcelona*-specific mAP versus learning time, measured in *vector operations* (vop) per data point, for AGM, AKM and RAKM under varying approximation levels, measured in FLANN *checks*. There are 5 measurements on each curve, corresponding, from left to right, to iteration 5, 10, 20, 30 and 40. (Right) *Oxford buildings* generic mAP in the presence of up to 1 million *distractor* images for AGM and RAKM, using query-side *soft assignment* with 1, 3 and 5 NNs.

Table 1. mAP comparisons for *generic* vocabularies of different sizes on *Oxford Buildings* with a varying number of distractors, using 100/200/200 FLANN checks for AGM/AKM/RAKM respecively, 40 iterations for AKM/RAKM, and 15 for AGM.

Method	RAKM							AKM	AGM
Vocabulary	100K	200K	350K	500K	550K	600K	700K	550K	857K
No distractors	0.430	0.464	0.471	0.479	**0.486**	0.485	0.476	0.485	**0.492**
20K distractors	0.412	0.427	0.439	0.440	**0.448**	0.441	0.437	0.447	**0.459**

For *large scale image retrieval*, we train *generic* vocabularies on an independent dataset of 15K images and 6.5M descriptors, and evaluate on *Oxford buildings* in the presence of up to one million distractors from *world cities*. Because experiments are very expensive, we first choose the best competing method for up to 20K distractors as shown in Table 1. We use $m = 100$ in this experiment, with 40 iterations for AKM/RAKM, and only 15 for AGM. This is because we are now focusing on *mAP performance* rather than learning speed, and our choices clearly favor AKM/RAKM. It appears that 550K is the best size for RAKM, and AKM is more or less equivalent as expected, since their difference is in speed. AGM is slightly better with its vocabulary size $C = 857$K being *automatically inferred*, keeping $\lambda = 0.2$, $\tau = 0.55$ as explained in section 5.2.

Keeping exactly the same settings, we extend the experiment up to 1M *distractors* for the RAKM 550K and AGM 857K vocabularies under query-side *soft-assignment*, as depicted in Figure 4 (right). Without any further tuning

compared to that carried out on *Barcelona*, AGM maintains a clear overhead of up to **3.5**% in mAP, *e.g.* 0.303 (0.338) for RAKM (AGM) at 1M distractors.

The query time of RAKM (AGM) is 354ms (342ms) on average for the full 1M distractor index, so there appears to be no issue with the balance of cluster population. Spatial verification and re-ranking further increases mAP performance to 0.387 (0.411) for RAKM (AGM) at an additional cost of 323ms (328ms) using *Hough pyramid matching* [29] on a shortlist of 200 images at 1M distractors. Variants of Hamming embedding [24], advanced scoring or query expansion are complementary, hence expected to further boost performance.

6 Discussion

Grouping 10^7 points into 10^6 clusters in a space of 10^2 dimensions is certainly a difficult problem. Assigning one or more visual words to 10^3 descriptors from each of 10^6 images and repeating for a number of different settings and competing methods on a single machine has proved even more challenging. Nevertheless, we have managed to tune the algorithm on one dataset and get competitive performance on a different one with a set of parameters that work even on our very first two-dimensional example. In most alternatives one needs to tune at least the vocabulary size. Even with spherical components, the added flexibility of Gaussian mixtures appears to boost discriminative power as measured by retrieval performance. Yet, learning is as fast as approximate k-means, both in terms of EM iterations and underlying vector operations for NN search.

Our solution appears to avoid both singularities and overlapping components that are inherent in ML estimation of GMMs. Still, it would be interesting to consider a *variational approach* [3] and study the behavior of our purging and expanding schemes with different priors. Seeking visual synonyms towards *arbitrarily-shaped* clusters without expensive sets of parameters is another direction, either with or without training data as in [23]. More challenging would be to investigate similar ideas in a *supervised* setting, *e.g.* for classification. More details, including software, can be found at our project home page[3].

References

1. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: CVPR (2007)
2. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: CVPR (2006)
3. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
4. Perronnin, F.: Universal and adapted vocabularies for generic visual categorization. PAMI 30(7), 1243–1256 (2008)
5. Li, D., Yang, L., Hua, X.S., Zhang, H.J.: Large-scale robust visual codebook construction. ACM Multimedia (2010)
6. Boiman, O., Shechtman, E., Irani, M.: In defense of nearest-neighbor based image classification. In: CVPR (2008)

[3] `http://image.ntua.gr/iva/research/agm/`

7. Jurie, F., Triggs, B.: Creating efficient codebooks for visual recognition. In: ICCV (2005)
8. Leibe, B., Mikolajczyk, K., Schiele, B.: Efficient clustering and matching for object class recognition. In: BMVC (2006)
9. Fulkerson, B., Vedaldi, A., Soatto, S.: Localizing Objects with Smart Dictionaries. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part I. LNCS, vol. 5302, pp. 179–192. Springer, Heidelberg (2008)
10. Winn, J., Criminisi, A., Minka, T.: Object categorization by learned universal visual dictionary. In: ICCV (2005)
11. Wu, J., Rehg, J.M.: Beyond the euclidean distance: Creating effective visual codebooks using the histogram intersection kernel. In: ICCV (2009)
12. Agarwal, A., Triggs, B.: Hyperfeatures – Multilevel Local Coding for Visual Recognition. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006, Part I. LNCS, vol. 3951, pp. 30–43. Springer, Heidelberg (2006)
13. Tuytelaars, T., Schmid, C.: Vector quantizing feature space with a regular lattice. In: ICCV (October 2007)
14. Dong, W., Wang, Z., Charikar, M., Li, K.: Efficiently matching sets of features with random histograms. ACM Multimedia (2008)
15. Philbin, J., Chum, O., Sivic, J., Isard, M., Zisserman, A.: Lost in quantization: Improving particular object retrieval in large scale image databases. In: CVPR (2008)
16. Silpa-Anan, C., Hartley, R.: Optimised KD-trees for fast image descriptor matching. In: CVPR (2008)
17. Muja, M., Lowe, D.: Fast approximate nearest neighbors with automatic algorithm configuration. In: ICCV (2009)
18. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. PAMI 33(1), 117–128 (2011)
19. van Gemert, J., Veenman, C., Smeulders, A., Geusebroek, J.: Visual word ambiguity. PAMI 32(7), 1271–1283 (2010)
20. Liu, L., Wang, L., Liu, X.: In defense of soft-assignment coding. In: ICCV (2011)
21. Jegou, H., Douze, M., Schmid, C.: Improving bag-of-features for large scale image search. IJCV 87(3), 316–336 (2010)
22. Lehmann, A., Leibe, B., van Gool, L.: PRISM: Principled implicit shape model. In: BMVC (2009)
23. Mikulík, A., Perdoch, M., Chum, O., Matas, J.: Learning a Fine Vocabulary. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part III. LNCS, vol. 6313, pp. 1–14. Springer, Heidelberg (2010)
24. Jegou, H., Douze, M., Schmid, C.: Hamming Embedding and Weak Geometric Consistency for Large Scale Image Search. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part I. LNCS, vol. 5302, pp. 304–317. Springer, Heidelberg (2008)
25. Ueda, N., Nakano, R., Ghahramani, Z., Hinton, G.: SMEM algorithm for mixture models. Neural Computation 12(9), 2109–2128 (2000)
26. Figueiredo, M., Jain, A.: Unsupervised learning of finite mixture models. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(3), 381–396 (2002)
27. Verbeek, J., Nunnink, J., Vlassis, N.: Accelerated EM-based clustering of large data sets. Data Mining and Knowledge Discovery 13(3), 291–307 (2006)
28. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer (2009)
29. Tolias, G., Avrithis, Y.: Speeded-up, relaxed spatial matching. In: ICCV (2011)
30. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded Up Robust Features. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 404–417. Springer, Heidelberg (2006)