Learning Filaments

Geoffrey J. Gordon GGORDON@CS.CMU.EDU
Andrew Moore AWM@CS.CMU.EDU
Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Abstract

This paper is about new statistics and new efficient algorithms for a form of mixture model that learns filamentary structures. Such models are important in several areas of scientific data analysis, but in this paper our main example is identification of large-scale structure among galaxies. We describe software which can extract the positions of spherical and line-shaped clusters from data about the locations of objects such as galaxies. We do so by fitting a particular type of Gaussian mixture model to the galaxy locations. The most interesting feature of our model is that it directly represents line segments in the distribution, unlike standard Gaussian mixture models which can only handle ellipses. Because we fit the line segments directly, we do not need to do any post-processing to extract their locations. We use a modification of the k-means algorithm to find model parameters. Since our software needs to deal with large data sets, it is important to accelerate model-fitting as much as possible. So, we store the galaxy locations in a multi-resolution kd-tree, and we introduce new pruning algorithms that allow us to skip over large parts of the tree in each k-means step. We provide evaluations on both synthetic and real data sets.

1. Introduction

Sky surveys are an important part of modern astronomy. As telescopes improve, each new survey collects more data: the Las Campanas Redshift Survey (Shectman et al., 1996), the source of the galaxy data in this paper, measured about 25,000 galaxy positions, while the in-progress Sloan Digital Sky Survey will soon be collecting 5,000,000 galaxy positions per month.

One of the answers that astronomers hope to determine from sky survey data is which of several physical models are capable of explaining the qualitative features of the distribution of galaxies in the universe. It is easy to see by examination (see Figure 3) that this distribution is not uniform. Instead, galaxies tend to clump together, sometimes in roughly spherical groups and sometimes along lines or sheets. Different physical models predict different types of clumpiness in the distribution of galaxies. So, it is important to measure statistics such as the number, size, and shape of clumps in the actual distribution of galaxies, since these statistics may help us discriminate among the competing physical models.

We also believe that these same statistics will be useful in domains other than astronomy. For example, we have considered fitting filaments to the distribution of protein locations produced in a 2D gel electrophoresis (which separates proteins according to their mass and charge), and to the distribution of mitochondria in the cytoplasm of cells exposed to various drugs.

In this paper we describe software which can extract the positions of groups and lines from data about the locations of galaxies (we consider only two-dimensional slices through the three-dimensional universe, so we do not need to look for sheets). We do so by fitting a particular type of Gaussian mixture model, described below, to the galaxy locations. The most interesting feature of our model is that it directly represents line segments in the distribution, unlike standard Gaussian mixture models which can only handle ellipses. Because we fit the line segments directly, we do not need to do any post-processing to extract their locations from the model parameters.

We use a modification of the k-means algorithm to find model parameters. The modifications are to account for clusters whose centers are line segments instead of points. The original k-means algorithm alternates between assigning the data points to their closest cluster center (the E-step) and moving each cluster center to the mean of its assigned data points (the M-step). In our modified k-means algorithm, we still assign each data point to its closest cluster center; but, since some of the cluster centers are line segments rather than points, we need to change the M-step. The new M-step still tries to minimize the sum of squared distances between the data points and their assigned cluster centers, but since line segments may share endpoints, and since the locations of points in the middle of a line segment are a function of the locations of the endpoints, we end up with a sparse system of linear constraints we must respect during the minimization. (See Section 2 for more detail.)

Since our software needs to deal with large data sets, it is important to accelerate the model-fitting process as much as possible. To accelerate parameter fitting, we store the galaxy locations in a data structure called a multi-resolution kd-tree. The tree, which we review in Section 4, allows us to skip over large portions of the data during each iteration of the k-means algorithm while still preserving the accuracy of the resulting parameter estimates. For more detail about how we decide which portions of the tree to skip, see Section 5.

In Section 6 we evaluate our algorithm with experiments on data from the Las Campanas Redshift Survey. Finally, in Section 7, we discuss future research.

2. The Filament Model

The standard k-means model assumes that each data point is generated in two steps: first, we select one of k cluster centers at random with probabilities p_j $(1 \leq j \leq k)$. Then, the sample is equal to the location y_j of cluster center j plus a zero-mean displacement, which we will assume is normally distributed with fixed covariance $\sigma^2 I$.

The problem with this model is that it doesn't capture the kind of structure we're interested in, which is a network of long, thin filaments. While it might be possible to reconstruct such a structure from the output of the k-means algorithm, it is not clear how to do so, and in any case the extra step would introduce unnecessary errors. So instead, we will fit a model which represents the filaments more directly.

In the filament model, there are k^v point-like cluster centers, just like the centers in the standard k-means model. We will call these centers vertices. In addition, any two of the vertices may be connected by an edge, say k^e edges in all. Points may be generated either from edges or from vertices. To generate a new sample, first we pick a vertex or edge at random, with probabilities p^v_j and p^e_r (with $\sum_{j=1}^{k^v} p^v_j + \sum_{r=1}^{k^e} p^e_r = 1$). Then, if we picked an edge, we choose a point on the edge uniformly at random. We will call the chosen point (a vertex or a point along an edge) the generating point for this sample. Finally, to the generating point we add a normally distributed displacement.

The bottom-right panel of Figure 2 shows a filaments model along with a sample of data points generated from it. In this model, $k^v = 5$ and $k^e = 5$; of course, in the actual galaxy data, k^v and k^e will be larger.

For most of this paper, we will assume that the structure of the model (the number of vertices and edges and their connectivity) is given in advance. So, the problem will be to fit the model parameters σ, p^v_j, p^e_r, and y_j. At the end of the paper, we will return briefly to how one might fit the structure as well.

Since we have altered the standard k-means model, we will no longer be able to use the standard k-means algorithm to fit its parameters. Fortunately, it will turn out that we can fit the new model by using an algorithm that is almost as simple. The next section describes this algorithm.

3. Parameter Fitting

3.1 The Original k-means Algorithm

The standard k-means algorithm alternates between two steps, called the E-step and the M-step. In the E-step, we assign each data point to its closest cluster center, breaking ties arbitrarily; in the M-step, we move each cluster center to the mean of its assigned data points. Both the E-step and the M-step are guaranteed not to increase the sum of squared distances between data points and their assigned cluster centers. Since there are finitely many possible assignments of points to clusters, this means that eventually the cluster assignments and center locations will converge (except that a point may jump back and forth between two centers at exactly the same distance).

The E- and M-steps of the k-means algorithm can be viewed as a limiting case of the Expectation-Maximization algorithm for fitting a mixture of Gaussian distributions. In this more general EM algorithm (which is also called fuzzy k-means), we start by pretending that we know which clusters generated which points. Define indicator variables z_{ij} which are 1 if point i belongs to cluster j and 0 if it does not; then the pdf of the k-means model is

$$P(x_i | y_j, z_{ij}, \sigma)$$
$$= \prod_{i=1}^{N} \left(\sqrt{2\pi}\sigma\right)^{-d} \exp\left(-\frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{ij}\|x_i - y_j\|^2\right)$$

Here we have written N for the number of samples, x_i $(1 \leq i \leq N)$ for the samples themselves, and d

for the number of dimensions (that is, the number of coordinates in each vector x_i). Recall y_j is the location of vertex j.

Since we don't actually know the values of the indicator variables z_{ij}, we can get the true pdf by marginalizing z_{ij} out of $P(x_i|y_j, z_{ij}, \sigma)$. That is, we can compute

$$P(x_i|y_j,\sigma) = \sum P(z_{ij}|y_j,\sigma)P(x_i|y_j,z_{ij},\sigma)$$

where the summation is over all possible assignments to z_{ij}. We would then try to find y_j and σ to maximize $P(x_i|y_j,\sigma)$. Unfortunately, finding the maximizing y_j and σ directly is intractable.

However, the EM theorem (Dempster et al., 1977) tells us how to find y_j and σ iteratively. It says that we should alternate between two steps: the E- or expectation-step, in which we compute the expected log likelihood

$$l(y_j,\sigma) = \sum P(z_{ij}|y_j,\sigma)\ln P(x_i|y_j,z_{ij},\sigma)$$

and the M- or maximization-step, in which we find the values of y_j and σ that maximize l. Notice that $l(y_j,\sigma)$ is similar in form to $P(x_i|y_j,\sigma)$; the only difference is that we are taking the expected value of $\ln P(x_i|y_j,z_{ij},\sigma)$ instead of $P(x_i|y_j,z_{ij},\sigma)$. This small difference makes all the difference: since $\ln P$ is linear in z_{ij}, we can move the expectation operator inwards and find the expected log likelihood just by computing the expected value of each z_{ij} given the current values of y_j and σ. This expected value is the posterior probability that point i was generated from cluster j.

The end result is that, to perform one iteration of EM, we compute the expected value of each z_{ij}. Then for each cluster j we find the weighted average of all of the points, with weights equal to $E(z_{ij})$, and set y_j equal to this average. Finally, we can either hold σ^2 and p_j fixed at some a priori values, or we can set $\sigma = \frac{1}{N}\sum_{ij} E(z_{ij})\|x_i - y_j\|^2$ and p_j to the expected fraction of points assigned to center j.

To go from fuzzy k-means to hard k-means, we can take the limit as $\sigma \to 0$. In this limit, $E(z_{ij})$ will always (w.p.1) be either zero or one, so the E-step will assign each point to its closest cluster center. After running EM to convergence, we can update σ and p_j.

3.2 An Extension to k-means

In the next subsection we will describe a version of k-means which works on our filament model. As a stepping-off point, in this subsection we will consider what happens if we add the constraint that one of the centers (say center 2) has to be exactly halfway between two others (say 1 and 3). (Any linear function of the locations of other centers would work as well, so in addition to introducing k-means for filaments, this sort of constraint may be useful in its own right.)

First, suppose that we know z_{ij} for all i and j. Also, for the purposes of this subsection, assume σ is fixed at 1. Then the conditional negative log likelihood is

$$\frac{Nd}{2}\ln(2\pi) + \frac{1}{2}\sum_{\{i|z_{i,1}=1\}} \|x_i - y_1\|^2 + \\ \frac{1}{2}\sum_{\{i|z_{i,2}=1\}} \|x_i - (y_1+y_3)/2\|^2 + \\ \frac{1}{2}\sum_{\{i|z_{i,3}=1\}} \|x_i - y_3\|^2 + \dots$$

where the ellipsis hides terms corresponding to vertices other than 1, 2, or 3. Since y_2 is no longer an independent parameter, we have replaced it by $\frac{1}{2}(y_1+y_3)$. Taking the derivative of this log-likelihood with respect to y_1 yields

$$\sum_{\{i|z_{i,1}=1\}} (y_1 - x_i) + \sum_{\{i|z_{i,2}=1\}} ((y_1+y_3)/2 - x_i)/2$$

As one might expect, the points assigned to center 2 affect the location of center 1, but not as much as the points assigned to center 1 do.

Setting this derivative to zero gives

$$n_1 y_1 + n_2(y_1+y_3)/4 \quad = \quad \sum_{\{i|z_{i,1}=1\}} x_i + \frac{1}{2}\sum_{\{i|z_{i,2}=1\}} x_i$$

and similarly for other y_js, where we have written n_j for the number of points assigned to center j.

We can write these equations more compactly by redefining $z_{i,1}$ to be 1 if point i belongs to vertex 1 and $\frac{1}{2}$ if point i belongs to vertex 2, and similarly for $z_{i,3}$. (To keep notation simple, define $z_{i,2} = 0$ for all i.) With this redefinition, the negative log likelihood is

$$\sum_i \frac{1}{2}\left\| x_i - \sum_j z_{ij} y_j \right\|^2 + \frac{d}{2}\ln(2\pi)$$

Taking the derivative with respect to y_1 yields

$$\sum_i z_{i,1} \sum_j z_{ij} y_j - \sum_i z_{i,1} x_i$$

So, taking derivatives with respect to all y_j and setting to zero yields the likelihood equations

$$Z^{\mathrm{T}} Z Y = Z^{\mathrm{T}} X$$

where we have written X for the $N \times d$ matrix whose ith row is x_i, Y for the $k \times d$ matrix whose jth row is y_j, and Z for the $N \times k$ matrix whose entries are z_{ij}.

If there were no constraints on the locations of any centers, the matrix Z would have all elements 0 or 1, with exactly one 1 in each row. So, $Z^{\mathrm{T}}Z$ would be diagonal, and its j, jth entry would be the number of data points assigned to vertex j. Since the jth row of $Z^{\mathrm{T}}X$ would be the the sum of all data points assigned to center j, in this case the likelihood equations would reduce to the ordinary k-means update. More generally, the r, sth element of $Z^{\mathrm{T}}Z$ is the sum over all data points i of $z_{ir}z_{is}$. So, the only off-diagonal elements of $Z^{\mathrm{T}}Z$ that can be nonzero are $1, 3$ and $3, 1$, both of which are equal to $\frac{1}{4}$ of the number of data points assigned to center 2.

Just as in the ordinary k-means algorithm, if we don't know Z ahead of time, the EM theorem tells us to replace expressions involving Z by their expected values. This time, though, we need more than just $E(Z)$: we also need $E(Z^{\mathrm{T}}Z)$. Fortunately, $E(Z^{\mathrm{T}}Z)$ is easy to calculate. Terms $E(z_{ir}z_{is})$ will be zero unless $r = s$ or $r, s \in \{1, 3\}$. Since $z_{i,1}z_{i,3} = \frac{1}{4}$ if point i belongs to vertex 2 and zero otherwise, $E(z_{i,1}z_{i,3}) = \frac{1}{4}P(z_{i,2} = 1)$. Because $z_{ir}^2 = z_{ir}$ for $p \notin \{1, 3\}$, in those cases $E(z_{ir}^2)$ is just $E(z_{ir})$. Finally, $E(z_{i,1}^2)$ is the probability that point i belongs to vertex 1 plus a quarter of the probability that it belongs to vertex 2. (All expectations in this paragraph are conditional on the values of y_j and σ at the end of the previous M-step.)

3.3 Fitting Filaments

The EM algorithm for networks of filaments works the same way as the EM algorithm for constrained vertices. The only difference is that, if datum i's generating point was 27% of the way from vertex r to vertex s, we will set $z_{ir} = .73$ and $z_{is} = .27$. The intuition for this definition is that an edge from vertex r to vertex s can be conceptually broken down into a sequence of constrained vertices, one at a fraction ϵ of the distance between r and s, another at 2ϵ, and so forth.

The remainder of the derivation proceeds as before. The conditional pdf is

$$\begin{aligned} &P(X|Y, Z, \sigma) \\ &= \left(\sqrt{2\pi}\sigma\right)^{-Nd} \exp\left(-\frac{1}{2\sigma^2}\|X - ZY\|_F^2\right) \end{aligned}$$

where we have defined $\|A\|_F^2$ to be the sum of the squares of the elements of the matrix A. The negative log likelihood is

$$Nd \log\left(\sqrt{2\pi}\sigma\right) + \frac{1}{2\sigma^2}\|X - ZY\|_F^2$$

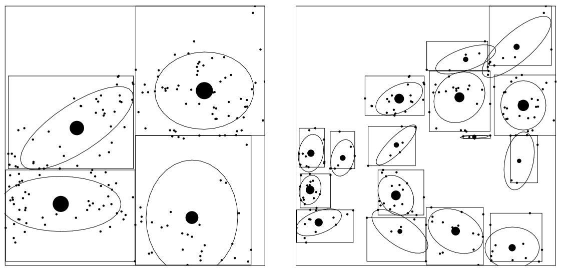

Figure 1. A kd-tree. Left: nodes at level 3. Right: nodes at level 5. The dots are the individual data points. The sizes and positions of the disks show the node counts and centroids. The ellipses and rectangles show the covariances and bounding boxes.

which means that the derivative with respect to Y is

$$\frac{1}{\sigma^2}\left(Z^{\mathrm{T}}ZY - Z^{\mathrm{T}}X\right)$$

and with respect to σ is

$$\frac{Nd}{\sigma} - \sigma^{-3}\|ZY - X\|_F^2$$

leaving us with the equations

$$E(Z^{\mathrm{T}}Z)Y = E(Z)^{\mathrm{T}}X$$

$$Nd\sigma^2 = E\left(\|ZY - X\|_F^2\right)$$

to solve at each step of the EM algorithm. (As before, expectations are conditional on the values of Y and σ at the end of the previous M-step.) These equations will usually be sparse, since the r, sth entry in $E(Z^{\mathrm{T}}Z)$ can only be nonzero if either $r = s$ or vertices r and s are connected by an edge.

4. The Multi-Resolution kd-tree

A MRKD-tree records a d-dimensional data set containing N records. It is a conventional kd-tree (Friedman et al., 1977) decorated, at each node, with extra statistics about the data within the node. Each node represents a set of data points. It records their count, centroid, covariance, and bounding box. A node containing fewer than $R_{\min}$ points is a leaf. Non-leaf nodes have two children, obtained by splitting the widest dimension of the parent's bounding box. Figure 1 shows an example.

MRKD-trees can be built quickly, in time $O(N + (dN/R_{\min})\log N + d^2N/R_{\min})$. Although we have not needed to do so, they can modified to become disk-resident for data sets with billions of records, and they can be efficiently updated incrementally. It is well known that kd-trees can accelerate proximity and range queries. But with the cached statistics, different kinds of tree traversal algorithms can accelerate (by several orders of magnitude) kernel methods, locally weighted regression, Gaussian mixture models, and simple k-means on large data sets (Deng & Moore, 1995; Moore et al., 1997; Moore, 1999; Pelleg & Moore, 2000). The essence of these traversals is that instead of iterating over all the data points, we can iterate over the tree nodes. Then, a combination of geometric and statistical bounds can often permit local proofs that approximating the contents of a node with its cached distribution will have negligible effect on the computation.

5. Accelerating the M-step

5.1 Overview

Once we have built the MRKD-tree, we can use it to help us sum up the contribution of each data point to the likelihood equations. The reason that the MRKD-tree helps us is that we can use the information stored in any node of the tree to compute bounds on $E(Z)$ and $E(Z^{\mathrm{T}}Z)$. If these bounds are tight enough, we can then approximate the contribution of the data points stored below the current kd-tree node without examining the node's children.

In this paper we will only describe the computations for the hard k-means algorithm. The fuzzy k-means algorithm requires more complicated computations, and we leave it for another paper.

First consider computing $E(z_{ij})$ for a data point i under the current kd-tree node and some vertex j. We can decompose $E(z_{ij})$ into the sum of several pieces: write q^v_{ij} and q^e_{ir} for the probability that point i was generated from vertex j or edge r. Write $\text{start}(r)$ and $\text{end}(r)$ for the vertices at each end of edge r. Write α_{ir} for the value that $z_{i,\text{start}(r)}$ would take if i were generated from r; note that $z_{i,\text{end}(r)} = 1 - z_{i,\text{start}(r)}$ is the fraction of the way from $\text{start}(r)$ to $\text{end}(r)$ that i's generating point would lie. Then, if $A = \{r|\text{start}(r) = j\}$ and $B = \{r|\text{end}(r) = j\}$,

$$E(z_{ij}) = q^v_{ij} + \sum_{r\in A} q^e_{ir}\alpha_{ir} + \sum_{r\in B} q^e_{ir}(1-\alpha_{ir})$$

Next, consider computing $E(z^2_{ij})$. Because the terms of $E(z_{ij})$ are mutually exclusive, the possible contributions to $E(z^2_{ij})$ are exactly the same except that the contributed values are squared. So, we have

$$E(z^2_{ij}) = q^v_{ij} + \sum_{r\in A} q^e_{ir}\alpha^2_{ir} + \sum_{r\in B} q^e_{ir}(1-\alpha_{ir})^2$$

Finally, consider computing $E(z_{i,\text{start}(r)}z_{i,\text{end}(r)})$ for a data point i under the current node and some edge r. (All products of z_{ij}s other than z^2_{ij} and $z_{i,\text{start}(r)}z_{i,\text{end}(r)}$ are identically zero.) If point i belongs to a vertex there is no contribution to this expectation, while if point i belongs to edge r there is a contribution $\alpha_{ir}(1-\alpha_{ir})$. So we have[1]

$$E(z_{i,\text{start}(r)}z_{i,\text{end}(r)}) = \sum_{r\in A\cup B} q^e_{ir}\alpha_{ir}(1-\alpha_{ir})$$

5.2 Distance Bounds

We can bound $E(z_{ij})$, $E(z^2_{ij})$, and $E(z_{i,\text{start}(r)}z_{i,\text{end}(r)})$ by bounding the q^vs, q^es, and αs. In this subsection we will consider bounding the q^vs and q^es.

Define ϵ^v_{ij} to be the squared Euclidean distance from point i to vertex j and ϵ^e_{ir} to be the squared Euclidean distance from point i to edge r. Whichever q corresponds to the minimal ϵ will be 1 and the rest will be 0; if several ϵs are equal, their corresponding qs will be equal and sum to 1. (The case of equal ϵs does have nonzero probability. For example, a point in the far upper-right region of the last panel of Figure 2 would project onto a corner of the filaments network and so would be equidistant from two of the edges and one of the vertices.)

To bound the distances we will use the bounding box stored in the current node of the kd-tree. Since the bounding boxes are axis-parallel, the upper and lower bounds on ϵ^v_{ij} can be found coordinate by coordinate, in time linear in the number of dimensions d.

To find the lower bound on ϵ^e_{ir} we can solve a simple convex quadratic program: the variables are x and α, the location of a point within the bounding box and the coordinate of a point on edge r. The constraints are that x must remain within the bounding box and that α must be in $[0,1]$. The objective is to minimize the squared distance between x and $y(\alpha) = \alpha y_{\text{end}(r)} + (1-\alpha)y_{\text{start}(r)}$.

To find the upper bound on ϵ^e_{ir}, we divide edge r into at most $d+1$ pieces by cutting along the midpoint of the bounding box in each dimension. As we vary α from 0 to 1, the farthest corner of the bounding box from $y(\alpha)$ changes only at piece boundaries. So, within each piece we compute the smallest distance from $y(\alpha)$ to the corresponding corner of the bounding box; then

[1]The equations for $E(z^2_{ij})$ and $E(z_{i,\text{start}(r)}z_{i,\text{end}(r)})$ are where fuzzy k-means differs from hard k-means. In hard k-means, α_{ir} is always equal to the coordinate of the projection of point i onto edge r, while in fuzzy k-means α_{ir} is a random variable that follows a truncated normal distribution.

we take the minimum over all pieces to find the distance from the segment to the farthest corner of the bounding box. This upper bound is not tight: we are computing $\min_\alpha \max_x ||y(\alpha) - x||^2$, while the actual maximum distance is $\max_x \min_\alpha ||y(\alpha) - x||^2$. The latter expression, while guaranteed to be no larger, seems harder to compute.

5.3 Bounds on α_{ir}

In order to complete the derivation of our bounds on $E(Z)$ and $E(Z^{\mathrm{T}}Z)$ we now need to bound α_{ir} and α_{ir}^2 for each edge r. For any r there is a linear function $a \cdot x + b$ so that $\alpha_{ir} = \max(0, \min(1, a \cdot x_i + b))$. We can use the bounding box for the current kd-tree node to compute upper and lower bounds on $a \cdot x_i + b$ for any point i under the current kd-tree node. If the upper bound is ≤ 0 or the lower bound is ≥ 1, we are done: since α_{ir} is constant, we can use its constant value along with our bounds on q_{ir}^e to bound the contributions to $E(Z)$ and $E(Z^{\mathrm{T}}Z)$ from the points under the current kd-tree node.

If the bounds are both between 0 and 1 inclusive, we will use an approximation: we will pretend that q_{ir} and α_{ir} vary independently as we vary i. While this assumption is false, the error introduced is usually small by the time we are far enough down in the kd-tree to consider pruning. So for example, we replace

$$\sum_{i \in K} E(q_{ir})\alpha_{ir}^2 \rightarrow \left(\sum_{i \in K} E(q_{ir})\right)\left(\frac{1}{|K|}\sum_{i \in K} \alpha_{ir}^2\right)$$

where K is the set of data points under the current kd-tree node. Since we know $0 \leq \alpha_{ir} \leq 1$ for $i \in K$, α_{ir} is a linear function of x_i for $i \in K$. So, we can compute $\sum_{i \in K} \alpha_{ir}^2$ from the stored mean and covariance in the current kd-tree node.

Finally, if the bounds on $a \cdot x_i + b$ include either 0 or 1, we can use Chebyshev's inequality, along with the recorded covariance matrix of the points under the current node, to bound the error we would introduce by changing some of the $a \cdot x_i + b$ values to 0 or 1.

5.4 When to Prune

By pruning we mean deciding to replace a group of terms in our estimates of $E(Z)$ and $E(Z^{\mathrm{T}}Z)$ by the best approximation we can compute without examining the portion of the kd-tree below the current node. For this approximation, we generally choose the average of the upper and lower bounds we have computed.

"A group of terms" means all of those terms that come from comparing a vertex j or an edge r to each of the examples i that fall within the bounding box of the current kd-tree node. This means that we can prune any part of our network of filaments while continuing to examine the children of the current kd-tree node for the benefit of the remaining part of the network. This usually happens when we have proven that some vertex or edge has its corresponding q identically zero for all examples under the current kd-tree node, but it can also happen if we prove (for example) that all of the qs are between $\frac{1}{3}$ and $\frac{1}{2}$, and there are few enough examples left in this branch of the kd-tree that this range is insignificant.

This ability to prune any part of the network, instead of waiting until we are ready to prune the entire network, can be a source of significant savings: while there may be hundreds of vertices and edges in the network, once the bounding boxes get small enough only a few of them are likely to have nonzero probability.

In order to decide when to prune, we introduce a parameter $\tau \in [0, 1]$. Whenever we compute new bounds on q_{ij}^v, we consider both how tight the bounds are and what is the sum of previously computed q_{sj}^v for other points s. For example, if there are n points under the current node with $a \leq q_{ij}^v \leq b$, and if we have previously computed q_{sj}^v values for other samples s that sum to S, we will prune if $n(b - a) \leq S + na$. That is, we will prune only if it will introduce a relative error less than τ. (Of course, we may prune several terms which add up to a relative error of more than τ.)

Whenever we compute new bounds on q_{ir}^e, we do a two-stage test. First we compare how tight the bounds are to the sum of previous terms for this edge, just as we did for vertices. Then we compute the bounds on α_{ir} and compare them to the same cutoff. Only if both tests predict an acceptable level of error do we prune.

The scheme described here is only an approximation to what we might like to do, which is to compute the relative magnitude of the error we are considering introducing into the final solution $E(Z^{\mathrm{T}}Z)^{-1}E(Z)^{\mathrm{T}}X$. We believe that this latter expression is too complicated and too expensive to compute in our tree-traversal code.

6. Experimental Results

Our first experiment, shown in Figure 2, demonstrates that we can recover a simple filaments model from data. We began by drawing a sample of size 10,000 from an artificial model with five vertices and five edges. Then we ran EM from the starting point shown, which has the correct structure but moderately incorrect parameters. The top left panel of the figure shows that we achieved significant pruning; the remaining

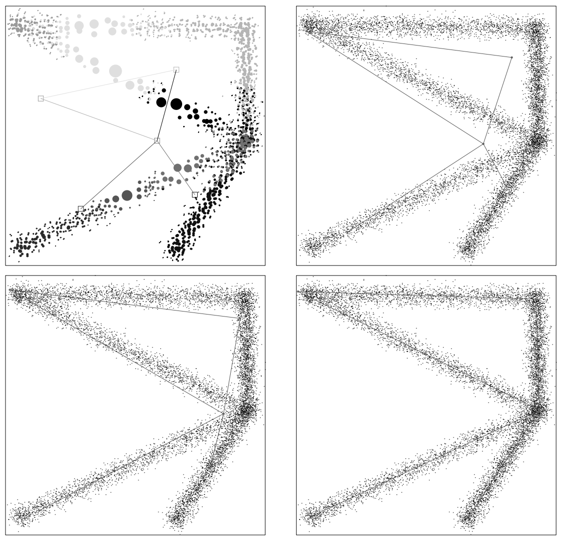

Figure 2. Fitting a simple filaments model. Top left shows initial position of model and pruning of kd-tree for first M-step. Area of each disk indicates number of samples pruned ($\tau = .01$), while gray level indidicates which center they were assigned to. Remaining three pictures show position of model after 1, 2, and 10 steps of EM.

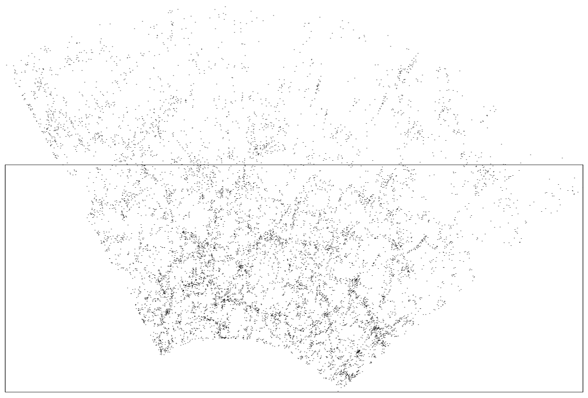

Figure 3. Galaxy data from the northern fan of the LCRS. Locations of 10,004 galaxies between 30 and 100 megaparsecs from Earth.

panels show the filaments model superimposed on the (unpruned) training data.

Our second experiment shows that we can achieve significant speedup over the version of k-means that doesn't use kd-trees. We ran our EM code to fit a filaments model to the locations of 20,731 galaxies from the Las Campanas Redshift Survey. The LCRS contains data on the position of about 25,000 galaxies located in six fan-shaped two-dimensional slices of the sky, three stacked on top of one another in the northern hemisphere and three stacked on top of one another in the southern hemisphere. For each galaxy, the LCRS indicates which fan it is in, its angular position within the fan, and its approximate distance from Earth as computed from its redshift. We selected the galaxies between 30 and 100 megaparsecs from Earth. Then we merged each fan with the others from the same hemisphere to produce two composite fans, one north and one south. See Figure 3 for a plot of the northern composite fan.

Since we have not yet implemented structure fitting, we started the EM iteration with a hand-drawn model that has 240 vertices and 282 edges. This model, not shown, has a reasonable guess at the structure in the northern fan of galaxies, but is deliberately incorrect in the southern fan.

Table 1 shows the time required to build the likelihood equations. Building the likelihood equations is the most expensive part of each iteration of the EM algorithm. The numbers are seconds of wall-clock time on a 600MHz DEC Alpha 21164 with 4GB of main memory, averaged over 100 repetitions, plus or minus the sample standard deviation. All differences are statistically significant at better than the $p = .001$ level.

Times are given for the k-means algorithm with and without the accelerations described in Section 5; for the accelerated k-means algorithm, we tried three different levels of the relative accuracy parameter τ. The table shows up to a two-fold speedup for the kd-tree-based algorithm; we expect that this ratio will increase as we move from the LCRS to larger data sets with millions instead of thousands of entries.

The different values of τ resulted in differing levels of error. The table shows the root mean square error both in the coefficients $E(Z)$ and $E(Z^{\mathrm{T}}Z)$ of the likelihood equation ("Error") and in the number of data points assigned to each center ("Count Err"). By way of comparison, the 2-norm of the vector containing all computed likelihood coefficients and counts is about 2,420. All values are averages over 100 replications. Count errors were generally larger than coefficient errors, since it is difficult to tell whether a sample was generated from a vertex or from one of the edges connected to it. But, count errors are also less harmful, since they only affect the p_j^e and p_j^v parameters, which are ignored until the last step of hard k-means.

7. Discussion

There are other possible approaches to finding filament-like structure. One group of approaches includes methods from computer vision that match parametric or semiparametric models to images. These

Table 1. Performance of k-means on LCRS data.

Method	Time $\pm\sigma$	Error	Count Err
Plain	15.51 ± 1.02	—	—
$\tau = 0.01$	10.11 ± 0.14	12.97	106.85
$\tau = 0.1$	9.33 ± 0.12	111.19	162.62
$\tau = 0.5$	7.27 ± 0.09	370.02	389.08

methods are usually designed for raster data, but could be modified to accept point sets like our galaxy locations. An example of such a method is Beveridge and Riseman (1997), which describes a local search algorithm to find matches between a set of line segments and an image. The search minimizes an error criterion similar to the one we use here, but unlike our filaments algorithm, it considers only affine transformations and not deformations of the model. Another example is Kass et al. (1988), which describes *snakes*, curves or rings that optimize their positions according to criteria designed to detect smooth contours. A second group of approaches includes statistical density estimation techniques such as nonlinear PCA (*e.g.*, (Bishop, 1995)) and principal curves (Hastie & Stuetzle, 1989). To our knowledge, no other methods, including the above, deal directly with fitting networks subject to the constraint that edges must meet up at vertices.

Eventually we would like to map filament structure automatically. A natural step in this direction is structure search: finding the best topology for the network. This has two challenges. First, defining a model selection criterion is tricky, both because it is more computationally expensive to compute likelihoods than it is to update parameters, and because hard k-means operates at the limit of zero variance and therefore infinite likelihood. Second, in the almost certain absence of a direct algorithm for finding an optimal structure we need to define a good set of incremental structure change operations for a structure search, perhaps using fast structure-change scoring methods similar to those in Pelleg and Moore (2000).

Finally, we would like to generalize this model to include sheets as well as filaments. Although implementation will be complicated, the underlying theory presented here is expected to generalize straightforwardly.

Acknowledgements

Thanks to Larry Wasserman, Bob Nichol, Dan Pelleg, Yanxi Liu, and the anonymous reviewers. Supported by NSF KDI award number DMS–9873442 to Andrew Moore.

References

Beveridge, J. R., & Riseman, E. M. (1997). *How easy is matching 2D line models using local search?* (Technical Report CS-96-117). Computer Science Department, Colorado State University, Fort Collins, Colorado.

Bishop, C. (1995). *Neural networks for pattern recognition.* Oxford: Oxford University Press.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*, 1–37.

Deng, K., & Moore, A. W. (1995). Multiresolution instance-based learning. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (pp. 1233–1239). San Francisco: Morgan Kaufmann.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, *3*, 209–226.

Hastie, T., & Stuetzle, W. (1989). Principal curves. *Journal of the American Statistical Association*, *84*, 502–516.

Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, *1*, 321–331.

Moore, A. W. (1999). Very fast mixture-model-based clustering using multiresolution kd-trees. *Advances in Neural Information Processing Systems 10* (pp. 543–549). San Francisco: Morgan Kaufmann.

Moore, A. W., Schneider, J., & Deng, K. (1997). Efficient locally weighted polynomial regression predictions. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 196–204). San Francisco: Morgan Kaufmann.

Pelleg, D., & Moore, A. W. (2000). X-means: Extending K-means with efficient estimation of the number of clusters. *Proceedings of the Seventeenth International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Shectman, S. A., Landy, S. D., Oemler, A., Tucker, D. L., Lin, H., Kirshner, R. P., & Schechter, P. L. (1996). The Las Campanas Redshift Survey. *The Astrophysical Journal*, *470*, 172. http://manaslu.astro.utoronto.ca/~lin/lcrs.html.

This research was sponsored in part by National Science Foundation (NSF) grant no. CCR-0122581.