Segments as Gaussian Distributions

1. Introduction

One way to approach segmentation is to think of each image segment as a set of pixels drawn from some common distribution. When we think of it this way, then we have two problems. The first is to describe the distribution for each segment. The second is to assign each pixels to a distribution, corresponding to a segment. This naturally leads to iterative algorithms, in which we fix the distribution, determine the assignment, then fix the assignment and determine the distribution, and then keep repeating this process. Today, we will take this approach, considering each distribution to be Gaussian. Doing this, we wind up with the E-M algorithm.

2. E-M

The basic idea of E-M is simple. We want to group together some primitives. If we knew which items belonged to a group, we could come up with a good description of the group, such as its position, intensity, or other descriptions of the group. On the other hand, if we knew a good description of a group, we could figure out which primitives belong to it. This is a chicken and egg problem. There is a standard, iterative solution to such problems. We somehow guess one side of the answer, then figure out the other side, then refigure out the first side, and keep going till we converge.

In normalized cut we did clustering based on pairwise affinities. In E-M, we will instead assume that our clusters have some parametric description. That is, a group can be described by a few parameters that can be derived from data. One clean example of this is to cluster colors into Gaussian distributions. Another is to cluster points into simple geometric groups, such as straight lines. A third is to cluster points based on their motion, assuming that large sets of points share a common, simple motion such as an affine transformation.

EM, by the way, is one of the most basic techniques in machine learning. It can be applied to all sorts of non-image clustering problems. This kind of clustering is called unsupervised learning, and is one of the main problems in machine learning. For example, suppose I have information about many ill patients and I want to cluster them into diseases.

I'm going to explain EM in stages. First I'll talk about K-means. This has many of the elements of EM. It has this back and forth iteration. But it is missing a key idea of soft assignment, and a probabilistic interpretation. So we'll then add that, in discussing EM.

2.1 K-Means

We'll consider a model problem. We have a bunch of points in the plane. We want to group these into clusters of nearby points. Let's define the following problem. We have

K clusters. Each cluster has a center, c_j. Every point is assigned to one cluster. We want to minimize the sum of squares difference between each point and its cluster center.

To be more precise, we have clusters $A, \ldots A_K$. We want to minimize:

$$\min_{A,c} \sum_{j=1}^{K} \sum_{x_i \in A_j} \left(c_j - x_i\right)^2$$

To solve this we 1) guess centers; 2) assign each point to the nearest center; 3) then recompute each center as the average of the points assigned to it, returning to step 2.

We can see that this iteration always reduces the error measure. This is because reassigning a point to the nearest center reduces error (step 2). And the center that minimizes sum of squared error is the average (step 3).

(To prove this for step 3, suppose we have points $x_1 \ldots x_n$, and we want to pick m to minimize

$$\sum_{i=1}^{n} \left(m - x_i\right)^2$$

If we take the derivative and set it to zero we have:

$$\sum_{i=1}^{n} \left(m - x_i\right)^2$$

$$m = \sum x_i \Big/ n \;)$$

We can also see that this must converge in a finite number of steps, because there are only a finite number of possible assignments.

Finally, we can see that this produces local minima that need not be global minima. For example, suppose we want to cluster 2, 6, 12 into two clusters. The global optima has centers at 4 and 12. But if we start at 0 and 6, say, we converge to 2 and 9, which is a local minima.

The key problems with this are: how do we initialize, and how many clusters do we look for? The first problem can be solved heuristically. Pick random points near the points. A good heuristic is to try many initializations and pick the one that leads to the best answer. Determining the number of clusters is always a problem, however this is work on this. Note that this is similar to the standard problem in learning of choosing the right number of parameters so you do not overfit your data.

2.2 E-M

K-means is very simple, but E-M tends to work better. The key idea is to make soft assignments, so that instead of a point going to one cluster, it is partially assigned to all of them. This seems to be a very good idea, and is key in algebraic multigrid as well. To do this in a principled way, we use a probabilistic formulation. Each cluster is a probability distribution over possible primitives, so we can assign a probability that each primitive belongs to each cluster.

One way to formalize this is to assume that our data points came from a mixture of Gaussian distributions, and that our goal is to find the mixture of Gaussians that is most likely to have generated the data.

First, let's define a mixture of Gaussians. We will write $g(x; m,\sigma)$ to indicate the probability of a point x based on a Gaussian distribution with mean m and variance σ. Note that we will be thinking of x and m as points in a high-dimensional space while σ is a scalar since the Gaussian is symmetric (ie it has a covariance matrix that is σI where I is the identity matrix).

A mixture of K Gaussians is a distribution in which we generate a point by randomly selecting one of K Gaussians, and then randomly draw a point from that distribution. If we select Gaussian k with a probability of p_k then we can write:

$$p(x;m,\sigma) = \sum_{k=1}^{K} p_k g(x;m_k,\sigma_k)$$

where $g(x; m_k,\sigma_\kappa)$ indicates the k'th Gaussian distribution, and m and σ indicate all k parameters. Our goal, then, is to find the parameters p_k, m_k, σ_κ that maximize the probability of our data points. Implicitly, this gives a soft assignment to each data point, since it gives the probability of that point coming from each distribution. Note that the negative log likelihood of the data will be closely related to the sum of square distance between each point and the center.

Now we can define an iterative algorithm. In the E step, we determine $p(k|n)$, the probability that each data point comes from each distribution. We denote this using a superscript i to indicate the number of the iteration, as:

- E Step:

$$p^{(i)}(k\,|\,n) = \frac{p_k^{(i)}\, g(\mathbf{x}_n;\, \mathbf{m}_k^{(i)}, \sigma_k^{(i)})}{\sum_{m=1}^{K} p_k^{(i)}\, g(\mathbf{x}_n;\, \mathbf{m}_k^{(i)}, \sigma_k^{(i)})}$$

That is, we compute the probability that point n is generated by distribution k, and then normalize this by the probability of all distributions generating it. We can think of this as softly assigning a point to each distribution, with these probabilities.

Next, in the M step we recompute the parameters of all these distributions, using these soft assignments.

- M Step:

$$\begin{aligned}
\mathbf{m}_k^{(i+1)} &= \frac{\sum_{n=1}^{N} p^{(i)}(k \mid n)\, \mathbf{x}_n}{\sum_{n=1}^{N} p^{(i)}(k \mid n)} \\
\sigma_k^{(i+1)} &= \sqrt{\frac{1}{D} \frac{\sum_{n=1}^{N} p^{(i)}(k \mid n)\, \|\mathbf{x}_n - \mathbf{m}_k^{(i+1)}\|^2}{\sum_{n=1}^{N} p^{(i)}(k \mid n)}} \\
p_k^{(i+1)} &= \frac{1}{N} \sum_{n=1}^{N} p^{(i)}(k \mid n) \; .
\end{aligned}$$

We can also prove that this converges to a locally optimal solution by showing that each step increases the probability of the points given the distributions.

E-M also can get stuck in local optima easily. However, it does seem to get stuck less than K-means. This is illustrated in a simple example with 1D points at 0 20 32, and beginning with centers at 10 and 32. This would be a local minima for k-means. But with E-M, the point at 20 is almost evenly shared between the two centers, at first. So the center at 32 gets smaller, and as it moves closer to 20 it takes over more of it. As the first center loses 20, it shifts to the left.

2.3 Extensions and Variations

This can be applied to many other problems. We can cluster colors or textures, by treating them as points in a three or high dimensional space. We can also add in position as a feature, and form clusters localized in position and texture/color. We can also use this to find lines, fitting a line to the points, and allowing for Gaussian noise in the normal direction.

3. Kernel Density Estimation

A basic tool for estimating a probability distribution from discrete samples is Kernel Density Estimation (KDE). The idea is that for every sample, you act as if you have sensed a continuous kernel, centered at that point. We'll only consider Gaussian kernels, which means that our distribution is a sum of identical Gaussians, centered at all the sample points:

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} K_{\mathbf{H}} (\mathbf{x} - \mathbf{x}_i)$$

Notice that this is equivalent to smoothing our sample points with a Gaussian. This makes sense if we assume that there is Gaussian noise in detecting sample points.

4. Mean Shift

Mean shift is an iterative algorithm for finding the modes of a probability distribution formed by KDE. By dividing the distribution up according to its modes, we are in effect segmenting our sample points. This is very closely related to EM, but has some nice advantages. First, we don't have to decide a priori how many clusters we will have (instead we have to choose the variance of a Gaussian kernel, but this may be more intuitive). Second, we don't have to initialize the iteration.

Mean shift requires us to define a Kernel. There is a theory of what kernels, *K*, are admissible, but we will only consider the case of zero mean, symmetric Gaussians, which is the most common. Such a kernel has a single parameter, the variance of the distribution, which is also called the *bandwidth.*

Then, given a set of data points, we think of there as being a distribution which is the result of convolving the points with that kernel. (ie, KDE). Mean shift is a method for finding the modes of this distribution. Given a starting point, we shift to the mean of the points, weighted by the kernel centered at that point. It can be proven that this converges to a mode of the distribution. We can also show that every step is in the direction of the gradient of the distribution.

We can apply mean-shift to color segmentation of an image, but considering pixels to be points in a 5D space of (*x*,*y*,*R*,*G*,*B*).

$$K_{h_s,h_r}(\mathbf{x}) = \frac{C}{h_s^2 h_r^p} k\left(\left\|\frac{\mathbf{x}^s}{h_s}\right\|^2\right) k\left(\left\|\frac{\mathbf{x}^r}{h_r}\right\|^2\right)$$

Here, we simply take the product of two Gaussian kernels for a color component and a spatial component, using different bandwidths for each.

Mean shift can then be used to segment a color image. Initialize mean shift at every pixel; pixel that converge to the same mode of the distribution belong to the same segment. We may also want to cluster some nearby modes together.