Segmentation by Clustering

Clustering

One natural view of segmentation is that we are attempting to determine which components of a data set naturally "belong together". This is a problem known as clustering.

Generally, we can cluster in two ways:

i) **Partitioning**: Here we have a large data set, and carve it up according to some notion of the association between items inside the set. We would like to decompose it into pieces that are "good" according to our model.
   For example, we might:

   - Decompose an image into regions which have coherent colour and texture inside them;
   - Take a video sequence and decompose it into shots — segments of video showing about the same stuff from about the same viewpoint;
   - Decompose a video sequence into motion blobs, consisting of regions that have coherent colour, texture and motion.

ii) **Grouping**: Here we have a set of distinct data items, and wish to collect sets of data items that "make sense" together according to our model. Effects like occlusion mean that image components that belong to the same object are often separated.

Examples of grouping include:

– collecting together tokens that, taken together, forming an interesting object

– collecting together tokens that seem to be moving together.

Factors that postdate the main Gestalt movement:

• Proximity: tokens that are nearby tend to be grouped.
• Similarity: similar tokens tend to be grouped together.
• Common fate: tokens that have coherent motion tend to be grouped together.
• Common region: tokens that lie inside the same closed region tend to be grouped together.
• Parallelism: parallel curves or tokens tend to be grouped together.
• Closure: tokens or curves that tend to lead to closed curves tend to be grouped together.
• Symmetry: curves that lead to symmetric groups are grouped together.

• Continuity: tokens that lead to “continuous” — as in “joining up nicely”, rather than in the formal sense — curves tend to be grouped.
• Familiar Configuration: tokens that, when grouped, lead to a familiar object, tend to be grouped together

Shot Boundary Detection

- Long sequences of video are composed of shots — much shorter subsequences that show largely the same objects.
- These shots are typically the product of the editing process.
- It is helpful to represent a video as a collection of shots; each shot can then be represented with a key frame. This representation can be used to search for videos or to encapsulate their content for a user to browse a video or a set of videos.
- Finding the boundaries of these shots automatically — shot boundary detection — is an important practical application of simple segmentation algorithms.
- A shot boundary detection algorithm must find frames in the video that are “significantly” different from the previous frame. Our test of significance must take account of the fact that within a given shot both objects and the background can move around in the field of view. Typically, this test takes the form of a distance; if the distance is larger than a threshold, a shot boundary is declared

Algorithm:

```
For each frame in an image sequence

    Compute a distance between this frame and the
    previous frame

    If the distance is larger than some threshold,

      classify the frame as a shot boundary.

end
```

There are a variety of standard techniques for computing a distance:
• Frame differencing algorithms take pixel-by-pixel differences between each two frames in a sequence, and sum the squares of the differences. These algorithms are unpopular, because they are slow — there are many differences — and because they tend to find many shots when the camera is shaking.

• Histogram based algorithms compute colour histograms for each frame, and compute a distance between the histograms. A difference in colour histograms is a sensible measure to use, because it is insensitive to the spatial arrangement of colours in the frame — for example, small camera jitters will not affect the histogram.
• Block comparison algorithms compare frames by cutting them into a grid of boxes, and comparing the boxes. This is to avoid the difficulty with colour histograms, where (for example) a red object disappearing off-screen in the bottom left corner is equivalent to a red object appearing on screen from the top edge. Typically, these block comparison algorithms compute an interframe distance that is a composite — taking the maximum is one natural strategy — of inter-block distances, computed using the methods above.
• Edge differencing algorithms compute edge maps for each frame, and then compare these edge maps. Typically, the comparison is obtained by counting the number of potentially corresponding edges (nearby, similar orientation, etc.) in the next frame. If there are few potentially corresponding edges, there is a shot boundary. A distance can be obtained by transforming the number of corresponding edges.

Background Subtraction algorithm

- In many applications, objects appear on a background which is very largely stable. The standard example is detecting parts on a conveyor belt. Another example is counting motor cars in an overhead view of a road.

- In these kinds of applications, a useful segmentation can often be obtained by subtracting an estimate of the appearance of the background from the image, and looking for large absolute values in the result. The main issue is obtaining a good estimate of the background. One method is simply to take a picture. This approach works rather poorly, because the background typically changes slowly over time. For example, the road may get more shiny as it rains and less when the weather dries up; people may move books and furniture around in the room, etc.

- An alternative which usually works quite well is to estimate the value of background pixels using a moving average. In this approach, we estimate the value of a particular background pixel as a weighted average of the previous values. Typically, pixels in the very distant past should be weighted at zero, and the weights increase smoothly. Ideally, the

moving average should track the changes in the background, meaning that if the weather changes very quickly relatively few pixels should have non-zero weights, and if changes are slow, the number of past pixels with non-zero weights should increase.

- This yields algorithm, in which filter that smooths a function of time, and we would like it to suppress frequencies that are larger than the typical frequency of change in the background and pass those that are at or below that frequency.

Clustering and Segmentation by K-means

A natural objective function can be obtained by assuming that we know there are k clusters, where k is known. Each cluster is assumed to have a center; we write the center of the i'th cluster as c_i. The j^{th} element to be clustered is described by a feature vector x_j . For example, if we were segmenting scattered points, then x would be the coordinates of the points; if we were segmenting an intensity image, x might be the intensity at a pixel.

We now assume that elements are close to the center of their cluster, yielding the objective function

$$\Phi(\text{clusters}, \text{data}) = \sum_{i \in \text{clusters}} \left\{ \sum_{j \in i\text{'th cluster}} (\boldsymbol{x}_j - \boldsymbol{c}_i)^T (\boldsymbol{x}_j - \boldsymbol{c}_i) \right\}$$

- If the allocation of points to clusters is known, it is easy to compute the best center for each cluster. However, there are far too many possible allocations of points to clusters to search this space for a minimum. Instead, we define an algorithm which iterates through two activities:
- Assume the cluster centers are known, and allocate each point to the closest cluster center.
- Assume the allocation is known, and choose a new set of cluster centers. Each center is the mean of the points allocated to that cluster. We then choose a start point by randomly choosing cluster centers, and then iterate these stages alternately. This process will eventually converge to a local minimum of the objective function.
- It is not guaranteed to converge to the global minimum of the objective function, however. It is also not guaranteed to produce k clusters, unless we modify the allocation phase to ensure that each cluster has some non-zero number of points.
- This algorithm is usually referred to as k-means. It is possible to search for an appropriate number of clusters by applying k-means for different values of k, and comparing the results; One difficulty with using this approach for segmenting images is that segments are not connected and can be scattered very widely.

This effect can be reduced by using pixel coordinates as features, an approach that tends to result in large regions being broken up.

```
Choose k data points to act as cluster centers

Until the cluster centers are unchanged

    Allocate each data point to cluster whose center is nearest

    Now ensure that every cluster has at least
    one data point; possible techniques for doing this include .
    supplying empty clusters with a point chosen at random from
    points far from their cluster center.

    Replace the cluster centers with the mean of the elements
    in their clusters.

end
```

Algorithm 15.5: *Clustering by K-Means*