Segmentation: Clustering, Graph Cut and EM

Ying Wu
Electrical Engineering & Computer Science
Northwestern University
Evanston, IL 60208
yingwu@ece.northwestern.edu

Contents
1 Motivations and Applications 2
  1.1 Background Subtraction .................................................. 2
  1.2 Object Representation .................................................... 2
  1.3 Layer Representation .................................................... 3
2 Image Segmentation and Motion Segmentation 3
  2.1 Image Segmentation ....................................................... 3
  2.2 Motion Segmentation ..................................................... 3
3 Image Segmentation by Clustering 4
  3.1 K-Means Algorithm ....................................................... 4
  3.2 Self-Organizing Map ...................................................... 5
4 Image Segmentation by Graph Cut 6
  4.1 Basic Idea ................................................................. 6
  4.2 Block-diagonalize the Affinity Matrix ................................ 6
5 Segmentation by Expectation-Maximization 7
  5.1 Missing Data Problem .................................................... 7
  5.2 E-M iteration .............................................................. 9
  5.3 Questions Remained ..................................................... 10
1 Motivations and Applications

Segmentation is a very fundamental problem in vision. Intuitively, segmentation is to group up similar components such as image pixels, image regions or even video clips. However, this problem becomes very complicated when it is difficult to define the similarity measurements, e.g., define similarity in terms of intensity, color, texture or motion, and when people are ambitious to expect some semantics from segmentation, e.g., we want to segment a people from an image.

![Figure 1: Some examples of image segmentation](image)

1.1 Background Subtraction

In most video surveillance applications, cameras are somehow fixed, and the systems should be able to separate the foreground such as moving people and the background. Because the camera motion is quite small, the background images do not change much. As a result, we could maintain the background and foreground is obtained by subtracting the background from images. Usually, an adaptive scheme can be used:

\[ B_{n+1} = \frac{w_a F + \sum_i w_i B_{n-i}}{w_c} \]

by selecting \( w_a, w_i \) and \( w_c \). The background subtraction technique is widely used in video surveillance and smart environments applications. However, when the camera is not fixed, we can hardly apply such a simple segmentation scheme.

1.2 Object Representation

To achieve a content-based image analysis, we need to have a compact representation for the objects in images. For example, to fulfill object recognition, we need to know what the
object is located in images and get rid of other image regions which are not belong to the object of interests. Segmenting the object from others is the first step for recognition.

1.3 Layer Representation

In many scene modelling applications, a scene is decomposed into different layers, each of which consists of different objects, which in turn suggests the segmentation process. Obviously, given one single image, it is very difficult to obtain the layer representation. But if we can an image sequence, motion consistancy could be taken into account to separate different layers.

2 Image Segmentation and Motion Segmentation

We will discuss two segmentation problems in class, i.e., image segmentation and motion segmentation, although there are some other problems could be formulated as segmentation problems as well.

2.1 Image Segmentation

Image segmentation is to group up similar pixels together to form a set of coherent image regions, given a single image. The pixel similarity could be measured based on the consistancy of location, intensity, color, and texture of different pixels. Generally, we can compound these elements together to represent an image pixel, or use some of them. For example, we can only use color components or use both location and intensities. So, for each image pixel, we associate it with a feature vector $x$.

This lecture will mainly concentrate on several different approaches to this problem, including:

- segmentation by clustering
- segmentation by graph cut
- segmentation by EM algorithm

2.2 Motion Segmentation

Motion segmentation is to group up similar pixels based on the motion consistancy of different pixels, given say two images of the same scene but under motion. For example, the scene consists of two different cars and the environment, motion segmentation can output three segments to represent the background environment, and the two cars respectively, based on their different movements. We shall cover some of the issue of motion segmentation in our later lectures.
3 Image Segmentation by Clustering

Clustering basically means grouping similar data points into different clusters or groups. This section presents two related approaches for clustering: the K-means algorithms and the self-organizing map. The two most important issues in clustering include similarity measurement and the clustering procedure.

3.1 K-Means Algorithm

We assume the number of clusters, \( K \), is given. We use the center of each clusters \( C_i \) to represent each cluster. The center of each cluster is the mean of the data points which belong to such a cluster. How do we determine which cluster a data point belongs to? Basically, we can define a distance measurement, thus the similarity measurement, \( D(x,y) \), in any sense. For example, \( D(x,y) = ||x - y||_2 \). We can compare the distance of a data point to these cluster centers, and such a data point belongs to the nearest cluster:

\[
  l_k(x_k) = \arg\min_i D(x_k, C_i) = \arg\min_i ||x_k - C_i||^2
\]

where \( l_k \) is the label for the data point \( x_k \).

The K-means algorithm tries to find a set of such cluster centers such that the total distortion is minimized. Here, the distortion is defined by the total summation of the distances of data points from its cluster center:

\[
  \phi(\mathcal{X}, C) = \sum_{i \in C} \sum_{j \text{--th cluster}} ||x_j - C_i||^2
\]

To minimize \( \phi \), K-means algorithm iterates between two steps:

- **Labelling**: Assume the \( p \)-th iteration ends up with a set of cluster centers \( C_i^{(p)} \), \( i = 1, \ldots, K \). We label each data point based on such a set of cluster centers, i.e., \( \forall x_k \), find

\[
  l_k^{(p+1)}(x_k) = \min_i ||x_k - C_i^{(p)}||^2
\]

and group data points belong to the same cluster

\[
  \Omega_j = \{ x_k : l_k(x_k) = C_j \}
\]

- **Re-centering**: Re-calculating the centers for all the clusters

\[
  C_i^{(p+1)} = \frac{\sum_{x_k \in \Omega_i} x_k}{|\Omega_i|}
\]

The algorithm iterates between such labelling and re-centering steps, and is expected to converge to a local stationary status.
3.2 Self-Organizing Map

Self-organizing map (SOM) could be used for visualizing and interpreting large high-dimensional data set by mapping them to a low-dimensional space based on a competitive learning scheme. SOM consists of an input layer and an output layer. Figure 2 shows the structure of 1-D SOM. The number of nodes in the input layer is the same as the dimension of the input vector. The structure of the output layer can be 1-D or 2-D connected nodes (neuron) that are connected to each input node with some weights $W_k$. The dimensionality of $W_k$ is the same as the input vectors. For any given input $x$, all those neurons compete with each other and the winner is selected as the one whose weight is closest to the input vector, i.e.,

$$y_i^* = \arg \min_i D(x_i, W_i)$$

Through competition, the index of the winning neuron is taken as the output of SOM. An interesting thing is that we are not only adjusting the weight of the winning neuron, but also train those neurons that are close to the winner neuron, and counter-train those that are far from the winner neuron by using a window function $\Lambda(||y - y_k^*||)$ and the Hebbian learning rule:

$$W(t+1) = W(t) + \eta(t)\Lambda(||y - y_k^*||)(x_k - W_{y_k^*}(t))$$

The window function identifies neurons inside or outside the region. The Hebbian learning rule adjusts the weights of the winning neuron and its neighborhood neurons. Intuitively, the input data point will attract the neuron inside the window to its location, but push those neuron outside the window far away. But iterate this training step again and again, the SOM is expected to converge to a stationary status.

One good characteristics of SOM is its property of partial data density preservation if properly trained. SOM is highly related to vector quantization (VQ) and K-means clustering. The question we should ask here is that “what is the relationship between SOM and the K-means algorithm?”, and “what is the advantage of using a neighborhood function in SOM over K-means?” We have had a discussion in class, but I will leave the questions to you for further thinking.
4 Image Segmentation by Graph Cut

Segmentation by graph cut is a very interesting approach!

4.1 Basic Idea

The basic idea of this approach is the following:

- each image pixel is viewed as a vertex of a graph;
- the similarity between two pixels is viewed as the weight of the edge of these two vertices;
- segmentation is achieved by cutting edges in the graph to form a good set of connected components.

The weights of the within-component edges will be large compared to the across-component edges. As a result, image segmentation problem becomes a graph cut problem. In math, we can represent an image ($i = 1, \ldots, N$ pixels) by a $N \times N$ matrix, which is called Affinity matrix:

$$ A_{ij} = \exp \left( - \frac{||x_i - x_j||^2}{2\sigma^2} \right) $$

which means that a larger $A_{ij}$ indicates the $i$-th and the $j$-th pixels are more similar. Given such a matrix, the segmentation problem becomes “can we block-diagonalize this affinity matrix?”

4.2 Block-diagonalize the Affinity Matrix

As a solution, we can imagine that if such an affinity matrix is block-diagonalized, the summation of the weights of those off-diagonal block matrices is minimized, or the summation of those diagonal block matrices is maximized. We introduce a new weight variable for each cluster component $W_n$,

$$ W_n = \begin{bmatrix} w_{n1} \\ w_{n2} \\ \vdots \\ w_{nN} \end{bmatrix} $$

where $w_{ni}$ means the weight of the $i$-th pixel associated with the $n$-th cluster component, i.e., the larger the weight, the stronger the association. Usually, such a weight vector is normalized, i.e., we have:

$$ W_n^T W_n = 1 $$

By using this weight indication variable, the block-diagonalization problem can be formulated by a linear programming problem, i.e.,

$$ W_n^* = \max_{W_n} W_n^T A W_n \ s.t. \ W_n^T W_n = 1 $$
To solve this optimization problem, we look at the Lagrangian $W_n^T A W_n + \lambda (W_n^T W_n - 1)$. Amazingly, we end up with a very nice solution:

$$AW_n^* = \lambda W_n^*$$

which means that the solution $W_n^*$ is an eigenvector of the affinity matrix $A$.

So, this approach suggests a very interesting solution. First, we perform eigenvalue decomposition of the affinity matrix, and identify $C$ largest eigenvalue $\lambda_k, k = 1, \ldots, C$, and find the corresponding eigenvectors $e_k, k = 1, \ldots, C$. Here, $e_k$ reveal the identities of each pixel for the $k$-th cluster component, since if $e_{ki}$ is roughly zero, we can say the $i$-th pixel does not belong to the $k$-th cluster, otherwise, it does.

However, we may be confronted by a problem: those eigenvalues are very similar, such that we could end up with eigenvectors that do not split clusters, because any linear combination of eigenvectors with the same eigenvalue is also an eigenvector. So, what should we do? If interested, please refer to the Normalized Cut approach [1, 2].

5 Segmentation by Expectation-Maximization

The Expectation-Maximization (EM) algorithm is quite important and fundamental for statistical learning. This section presents the basic idea of EM algorithm to deal with the missing data problem, thus a solution to segmentation.

5.1 Missing Data Problem

Segmentation problem can be reformulated as a missing data problem. What is the missing part? Let’s have a look at the forward problem of producing a set of data points. We assume each image pixel is produced by a probability density associated with one of the $g$ image segments. To produce an image pixel, we first choose an image segment, and then generate the pixel from the density associated with the segment, i.e.,

$$p(x) = \sum_i p(x|\theta_i)\pi_i$$

where $\pi$ is the prior for the $i$-th image segment, and $\theta_i$ is the parameter for the density associated with the $i$-th segment. This equation basically means that once we know from which segment (or component) the pixel is drawn, we can write its probability. We also associate a label $l_k$ for each pixel $x_k$ to indicate the identity of such a pixel. Equation 1 is a generative model describing the generation of the data sets from a mixture model.

We can usually assume Gaussian for each component, i.e.,

$$p(x|\theta_i) \sim G(\mu_i, \Sigma_i)$$

So, our task is to do the inverse: given a set of data point (image pixels) $\mathcal{X} = \{x_k, k = 1, \ldots, N\}$, we need to estimate those parameters $\theta_i, \pi_i$, and estimating the labels for all the data points by:

$$l_j^* = \arg \max_k p(l_j = k|x_j, \Theta), \forall x_j$$
which gives the posteriori probability of \( x_j \). Generally, the parameters of the density can be estimated by maximizing the likelihood of the data set. The likelihood of the data set can be written by:

\[
p(\mathcal{X}|\Theta) = \prod_j \left( \sum_{i=1}^{g} p(x_j|\theta_i)\pi_i \right)
\]

Usually, we take log of such a term to get the log likelihood:

\[
\log p(\mathcal{X}|\Theta) = \sum_j \log \left( \sum_{i=1}^{g} p(x_j|\theta_i)\pi_i \right)
\]

which is very ugly, because we have to handle the summation inside the log! In this sense, it is even impossible to estimate those density parameters \( \theta_i \) and \( \pi_i \) by looking into the the likelihood of the data set.

To ease this problem, we introduce an indicator variable \( z \):

\[
z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_g \end{bmatrix}
\]

If a data point \( x \) is drawn from the \( k \)-th component, then \( z_k = 1 \), and all other \( z_{i \neq k} = 0 \). This indicator variable represents the identity of a data point, which is the missing part!

What is the good news brought by these indicator variables? Let’s form the complete data point by stacking the data point and its indicator, i.e.,

\[
y_k = \begin{bmatrix} x_k \\ z_k \end{bmatrix}
\]

And the complete data set is \( \mathcal{Y} = \{y_k, k = 1, \ldots, N\} \). We can write up the likelihood of the complete data point \( y_k \) and the whole data set \( \mathcal{Y} \):

\[
p(y_k|\Theta) = \sum_{i=1}^{g} z_{ki} p(x_k|\theta_i)
\]

Why it is nice? Because we can write:

\[
\log p(y_k|\Theta) = \sum_{i=1}^{g} z_{ki} \log p(x_k|\theta_i)
\]

So, for the whole data set, we have

\[
p(\mathcal{Y}|\Theta) = \prod_{k=1}^{N} \sum_{i=1}^{g} z_{ki} p(x_k|\theta_i)
\]
And thus:

$$\log p(\mathcal{Y}|\Theta) = \sum_{k=1}^{N} \log(\sum_{i=1}^{g} z_{ki} p(x_k|\theta_i)) = \sum_{k=1}^{N} \sum_{i=1}^{g} z_{ki} \log p(x_k|\theta_i)$$

Because we eliminate the summation terms inside log, the ML estimation becomes easier:

$$\Theta^* = \arg \max_{\Theta} \log p(\mathcal{Y}|\Theta)$$

However, the bad news is that the indicator variable $z_k$ make the ML difficult, since we do not know $z_k$.

### 5.2 E-M iteration

Fortunately, life won’t be too bad. We observe a quite interesting phenomenon:

- if we know such $z_k$, i.e., we know the identities for each data point, we can easily estimate the density parameters $\Theta$ based on ML, without any doubt.

- At the same time, if we know the density parameters, we can easily solve such indicator variables $z_k$ based on MAP.

This phenomenon suggest an iterative procedure, which is exactly the Expectation-Maximization iteration:

- **E-step**: computing an expected value of the complete data, here only $E[z_k]$;

- **M-step**: maximizing the the log likelihood of the complete data to estimate $\Theta$.

The EM algorithm iterates between these two steps and expect a convergence to a local maximum of the likelihood. We should notice here that the EM algorithm only converges to a local maximum. A strict proof was given by Dempster *et al.* in 1977.

So, let’s have a look at the EM approach for image segmentation:

- **E-step**:

  $$E[z_{ki}] = 1 \cdot p(\text{kth pixel comes from ith component}) + 0 \cdot p(\text{kth pixel doesn’t come from ith component}) = p(\text{kth pixel comes from ith component}) = \frac{\pi_i p(x_k|\theta_i)}{\sum_{j=1}^{g} \pi_j p(x_k|\theta_j)}$$
- **M-step**

\[
\pi_i = \frac{1}{r} \sum_{l=1}^{r} p(i|x_l, \Theta)
\]

\[
\mu_i = \frac{\sum_{l=1}^{r} x_l p(i|x_l, \Theta)}{\sum_{l=1}^{r} p(i|x_l, \Theta)}
\]

\[
\Sigma_i = \frac{\sum_{l=1}^{r} p(i|x_l, \Theta) [(x_l - \mu_i)(x_l - \mu_i)^T]}{\sum_{l=1}^{r} p(i|x_l, \Theta)}
\]

### 5.3 Questions Remained

Although EM has been quite successful in both theory and practice, we still have some questions to ask:

- We notice that the EM algorithm assume the number of components is given in advance. What if we do not the number of components? This problem is common also for clustering methods like K-means and SOM. There is not satisfied answer to this question. In theory, people can take the minimum description length (MDL) principle to alleviate it, but there is no specific algorithm associated with MDL. In practice, people take the **cross-validation** procedure to find the best structural parameters.

- Another problem we need to investigate is that when the dimensionality of \( x \) is very high, we have many density parameters to estimate, which requires a huge amount of data; otherwise, the estimation will be heavily biased or unable to be solved at all.

I shall address these issues again in my later lectures. If interested, please read [4, 5, 3].

### References


