Problem Set 4
Assigned: 04/22/2004
Due: 05/04/2004

Please submit an electronic copy of your writeup and code for each problem to `6.891-submit@ai.mit.edu`.

Problem 1 *Image Segmentation*

In this problem you will compare the performance of two different clustering algorithms for image segmentation: k-means and mean-shift. The k-means algorithm is discussed in Chapter 14 of Forsyth and Ponce. The mean-shift image segmentation algorithm is discussed in the paper, D. Comaniciu and P. Meer: "Robust Analysis of Feature Spaces: Color Image Segmentation", provided as additional reading.

This problem is structured into two parts. The first part focuses on the implementation of the k-means and mean-shift algorithms as general nD-data point clustering algorithms. The second part applies these algorithms to perform image segmentation.

k-Means and Mean-Shift Clustering

(a) Implement the k-means clustering algorithm described as Algorithm 14.5 in Forsyth and Ponce. Your function should have the following syntax:

$$\texttt{function}\ [\texttt{labels}, \texttt{means}] = \texttt{kmeans}(\texttt{data}, \texttt{k})$$

where `data` is a matrix storing the n-dimensional data as its column vectors and `k` is the number of desired clusters you wish the algorithm to form. The cluster labels are returned as `labels`, a vector that has an entry for each data column of `data` storing for each data point an associated cluster label. `means` is a matrix storing the centers of each cluster as its columns.

The MATLAB data file `pts.mat` stores a set of 3D points belonging to two 3D Gaussians. Test and debug your algorithm using this data set with $k = 2$. Plot the resulting clusters using the provided function `plot3dclusters`.

(b) The mean-shift algorithm is discussed in the paper by Comaniciu and Meer assigned as additional reading. Below we provide the details of an implementation of the mean-shift algorithm.

As discussed in the paper, the mean-shift algorithm clusters an n-dimensional data set by associating each point to a peak of the data set's probability density. For each point, mean-shift computes its associated peak by first defining a spherical window at the data point of radius r and computing the mean of the points that lie within the window. The algorithm then shifts the window to the mean and repeats until convergence, i.e. the shift is under some threshold ϵ (we found $\epsilon = 0.01$ to work well). With each iteration the window will shift to a more densely populated portion of the data set until a peak is reached, where the data is equally distributed in the window. Implement the peak searching processes as the function

$$\texttt{function peak} = \texttt{findpeak}(\texttt{data}, \texttt{idx}, \texttt{r})$$

where `data` is the n-dimensional data set as before, `idx` is the column index of the data point for which we wish to compute its associated density peak and `r` is the search window radius. The algorithm's dependence on r will become apparent from the experiments performed below.

Implement the mean-shift function, which calls `findpeak` for each point and then assign a label to each point according to its peak. This function should have the syntax

$$\texttt{function } [\texttt{labels}, \texttt{peaks}] = \texttt{meanshift}(\texttt{data}, \texttt{r})$$

where `labels` are the same as for part (a) and `peaks` is a matrix storing the density peaks found using meanshift as its columns. Note the mean-shift algorithm requires that peaks are compared after each call to `findpeak` and for similar peaks to be merged. For our implementation of meanshift, we will consider two peaks to be the same if the distance between them is $\leq r/2$. Also, if the peak of a data point is found to already exist in `peaks` then for simplicity its computed peak is discarded and it is given the label of the associated peak in `peaks`.

Debug your algorithm using the data set from part (a) with $r = 2$ (this should give two clusters). Plot your result using the `plot3dclusters` function.

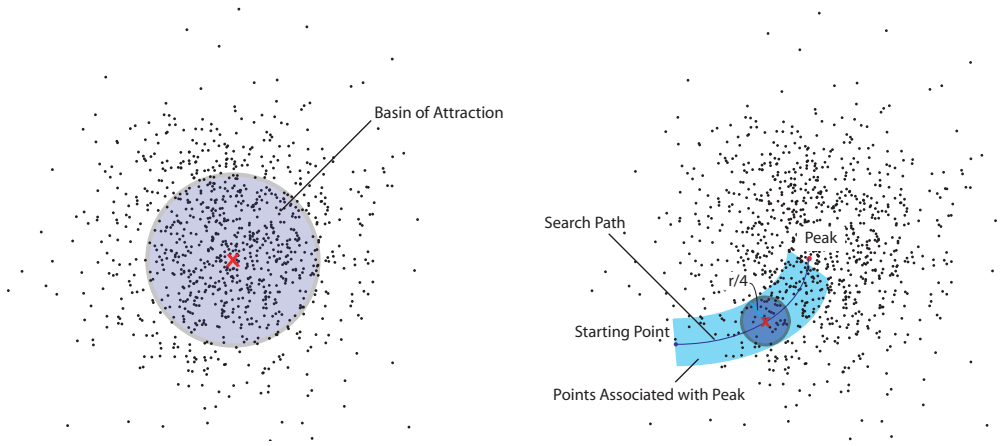

Figure 1: Speedups incorporated into the mean-shift algorithm: (a) basin of attraction, (b) points along the search path are associated with the converged peak.

(c) Using the data set stored by `pts.mat` run k-means with $k = 1, 2, 4, 8$ and mean-shift with $r = 1, 2, 4, 5, 10$. Plot the results of k-means and mean-shift in two separate subplots labelling each plot with its corresponding k or r value. Observe the results and discuss the differences between each of the algorithms. How is varying the search window radius r in mean-shift different from varying the k parameter of k-means? How are they the same?

(d) As implemented, the mean-shift algorithm of part (b) is too slow to be realized for image segmentation. We will therefore incorporate the following two speedups into our implementation. Upon finding a peak, the first speedup will be to associate each data point that is at a distance $\leq r$ from the peak with the cluster defined by that peak. This speedup is known as *basin of attraction* and is based on the intuition that points that are within one window size distance from the peak will with high probability converge to that peak (see Figure 1(a)). The second speedup is based on a similar principle, where points that are within a distance of r/c of the search path are associated with the converged peak, where c is some constant value (see Figure 1(b)). We will choose $c = 4$ for this problem.

Incorporate the above speedups into your mean-shift implementation by modifying your implementation from part (b). The resulting modified function should have syntax

$$\texttt{function}\ [\texttt{labels}, \texttt{peaks}] = \texttt{meanshift\_opt}(\texttt{data}, \texttt{r}).$$

To realize the second speedup you will need to modify `findpeak` as follows:

$$\texttt{function}\ [\texttt{peak}, \texttt{cpts}] = \texttt{findpeak\_opt}(\texttt{data}, \texttt{idx}, \texttt{r})$$

where `cpts` is a vector storing a 1 for each point that is a distance of $r/4$ from the path, 0 otherwise.

Image Segmentation

(a) In this section you will build upon your implementations of k-means and mean-shift to perform image segmentation. To do so, implement the function

$$\texttt{function segmIm} = \texttt{imSegment}(\texttt{im}, \texttt{p}, \texttt{alg})$$

where `im` is a color input image, p is the parameter associated with the k-means and mean-shift algorithms (i.e. p represents either k or r depending on which algorithm is run) and `alg`={'kmeans','meanshift'} specifies which clustering algorithm to use. In summary, this function is constructed by reshaping the image into RGB vectors and then clusters the resulting color data using either k-means or mean-shift depending on the value of `alg`. The segmented image is then constructed using the cluster labels and mean or peak values.

Note k-means and mean-shift both cluster using Euclidean distance metrics. As we saw earlier in the class when we studied the MacAdam ellipses of the CIE xy chromaticity diagram, Euclidean distance in RGB space does not correlate well to the perceived change in color. For example, in the green portion of the spectrum large distances are perceived as the same color, whereas in the blue part of the spectrum a small distance may represent a large change in perceived color (see Figures 6.13 and 6.14 of Forsyth and Ponce). For this reason we will use the

non-linear *Luv* color space. In this space Euclidean distance better models the perceived change in color. In `imSegment` cluster the image data in the *Luv* color space by first converting the *RGB* color vectors to *Luv* using the provided MATLAB function `rgb2luv`. Then convert the resulting cluster centers back to *RGB* using the function `luv2rgb`.

(b) Segment the images `sunset.bmp` and `terrain.bmp` using both k-means and mean-shift with $k = 3, 5, 7, 10$ and $r = 5, 10$. For each segmentation method, display the results of each segmentation in a subplot, labelling each image with its corresponding k or r value. What effect does varying k and r have on the resulting segmentations? Compare the regions formed by k-means for different values of k. How are these regions different than those formed by mean-shift for the different values of r? Explain. Note, although more efficient implementations exist, the above simplified implementation of mean-shift may take several minutes to run for the provided input images.

Problem 2 *Line fitting through Segmentation*

(a) Write a MATLAB function to generate noisy line segment data with outliers, similar to that shown in Figures 15.9 and 15.10 of Forsyth and Ponce. You should presume Gaussian distributed noise (in y) and uniformly distributed outlier noise. Your function should have the following syntax:

`function` $[$`X`, `Y`$]$ = `create_line`(`a`, `b`, `x1`, `x2`, `var`, `perc_out`, `range_out`)

The output parameters `[X,Y]` are samples of the line $y = ax+b$ between $x =$ `x1` and $x =$ `x2`. The input parameter `var` is the variance of the Gaussian distributed noise, `perc_out` is the percentage of outliers and `range_out` is the outlier range.

For each implementation (b), (c), (d) and (e), below,

(i) Plot cases where your implementation succeeds;

(ii) Plot cases where your implementation fails and discuss why it fails;

(iii) Discuss the effect of each input parameter on the algorithm;

(iv) Submit your files and print a copy of your code.

(b) Implement a simple least-squares fit of the line. Your function should have the following syntax:

$$\texttt{function}\ [\texttt{a}, \texttt{b}] = \texttt{fitline}(\texttt{X}, \texttt{Y})$$

(c) Implement a Hough transform accumulator array, and a simple maximum peak finder. Test this for different granularities of parameter sampling on single-line datasets with varying amounts of normal and outlier noise. Your function should have the following syntax:

$$\texttt{function}\ [\texttt{a}, \texttt{b}, \texttt{M}] = \texttt{hough}(\texttt{X}, \texttt{Y}, \texttt{r\_min}, \texttt{r\_max}, \texttt{r\_step}, \texttt{theta\_step})$$

where `[r_min, r_max]` is the range of the line distance from the origin, `r_step` is the incremental step between radius samples and `theta_step` is the incremental step between angle samples. The mask vector `M` is set to 1 when the point is an inlier and 0 when an outlier.

(d) Implement a RANSAC approach (Algorithm 15.4) using the least squares distance metric described in section 15.2.1. Test it on single-line data sets with varying amounts of normal and outlier noise. Your function should have the following syntax:

$$\texttt{function}\ [\texttt{a}, \texttt{b}, \texttt{M}] = \texttt{ransac}(\texttt{X}, \texttt{Y}, \texttt{n}, \texttt{max\_it}, \texttt{thresh}, \texttt{min\_inliers})$$

where `n` is the number of points sub-sampled to fit a line, `max_it` is the maximum number of iterations, `thresh` is the threshold used to identify inliers and `min_inliers` is the minimum number of inliers.

(e) Implement an EM approach to estimate a fixed number of noisy lines (Algorithm 16.4) and add an outlier model (section 16.4.2). The simplest way to do this is to consider a mixture model where the last mixture component is not a line, but a process that generates outlier points with uniform probability. Your function should have the following syntax:

$$\texttt{function}\ [\texttt{a}, \texttt{b}, \texttt{W}] = \texttt{em}(\texttt{X}, \texttt{Y}, \texttt{perc\_out}, \texttt{range\_out}, \texttt{W\_initial})$$

where `perc_out` models frequency with which outliers occur, `range_out` is the range of the uniform outlier process and the vector `W_initial` is

the initial weight values. The output parameter `W` is the final weight vector. Dont forget to define a convergence criterion.

Test your implementation on single- and multiple-line data sets with varying amounts of normal and outlier noise.