Computer Vision – 67542

MATLAB Assignment 1 – Segmentation

Preliminary Instructions

General

- Submission deadline: 29.3.2016
- Late submission is allowed, but every day past the deadline will reduce 5 points from the assignment grade.
- The assignment is individual (i.e. you are not allowed to work in pairs or groups).
- Submit your solution through the course website. All of your files should be zipped in a folder named *matlab1_[ID].zip* ([ID] standing for your 9-digit ID).
- Your submission should include all .m files used during the assignment. Provide a brief description of each code file in a text file named *readme.txt*.
- Your code should be supplemented by a PDF file named *report.pdf*, which will include documentation of your results.
- The assignment consists of several mandatory parts and two bonus parts. The basic 100 points of the grade are distributed evenly between the mandatory parts. The bonus parts are worth 5 points each.
- **Good luck!**

Dataset

The dataset you will be working with throughout this assignment resides in the folder *dataset*. It consists of several color images on which we will apply our segmentation algorithms.
The images are taken from
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html , where they, along with many other images, are accompanied by human specified segmentations. These human annotations serve as the 'ground truth' in the evaluation of state-of-the-art segmentation algorithms.
In this assignment, you will develop segmentation engines which are rather simple, far from being state-of-the-art. Accordingly, you will not compare your results against the 'ground truth',

but rather evaluate them qualitatively. For your convenience, exemplar results are attached in the Appendix. Note that these do NOT represent a golden standard that you should aim for. Rather, they are meant to loosely illustrate the expected quality of your results.

Part 1

Implement the k-means clustering algorithm via the following function (note that you are NOT allowed to use MATLAB's built-in k-means function):

[C, labels] = KMeans(X, k, max_iter, change_thresh, num_runs)

Inputs

- **X**: *d*-by-*m* matrix with columns holding $x_1, x_2, ..., x_m \in \mathbb{R}^d$ - the input points to cluster.
- **k**: Integer ≥ 2 holding the required number of clusters.
- **max_iter**: Positive integer holding the maximum number of iterations allowed in a run.
- **change_thresh**: Positive scalar holding the minimal objective change allowed between two consecutive iterations in a run. If the objective change is smaller than this value, the run terminates.
- **num_runs**: Positive integer holding the number of runs to carry out.

Outputs

- **C**: *d*-by-*k* matrix with columns holding $c_1, c_2, ..., c_k \in \mathbb{R}^d$ - the cluster centroids.
- **labels**: *m*-element vector of integers between *1* and *k*. A values of *r* in entry *i* means that x_i belongs to cluster *r*.

Description

The function applies the k-means clustering algorithm on the given data points (in *X*) *num_runs* times, and returns the clustering which produced the best (lowest) objective. Each k-means run is initialized by randomly choosing *k* different data points and setting them as centroids. A run terminates when any of the following two conditions is met:

1. *max_iter* iterations have been carried out.
2. The change in the objective between two consecutive iterations is less than *change_thresh*.

Part 2

In this part we will attempt to segment the images in our dataset by applying k-means clustering to their pixels.

Given a color image, we map its pixels into $\mathbb{R}^5$ by concatenating the color in L*a*b* space (this space is designed to capture perceptual difference through Euclidean distance, use MATLAB's *rgb2lab* function to reach it) with weighted image coordinates. Namely, we apply the following pixel-wise mapping:

$$T_\mu : p \mapsto \begin{bmatrix} L^* \text{ color band} \\ a^* \text{ color band} \\ b^* \text{ color band} \\ \mu \cdot (x \text{ coordinate}) \\ \mu \cdot (y \text{ coordinate}) \end{bmatrix}$$

where $\mu \geq 0$ is a parameter. After going through this mapping, the image pixels are fed into the k-means clustering engine. The produced labels provide the image segmentation.

Using the function *KMeans* which you have implemented in Part 1, apply the above procedure and segment each of the images in our dataset. When doing so, tune the parameters of *KMeans* (*k*, *max_iter*, *change_thresh*, *num_runs*) to reach results you find satisfactory. Add images of the segmentation results to your report (using MATLAB's *imagesc* function), along with the parameter values which produced them.
Repeat this process twice: once with $\mu = 0$ (i.e. without spatial information) and once while tuning μ to reach the best results you can. Explain the difference in the segmentation results between these two cases.

Bonus Part I

Repeat Part 2, while replacing T_μ with a pixel-wise mapping of your choice. Try to reach the best segmentation results you can (as stated above, we evaluate results subjectively in this assignment). Add the results to your report along with a description of the mapping you used.

Part 3

Implement the following spectral clustering function:

labels = SpectralCluster(W, k, algo)

Inputs

- **W**: *m*-by-*m* real non-negative symmetric matrix holding the affinities between the input points to cluster. In particular, entry (*r*,*s*) holds the affinity between input points x_r and x_s.
- **k**: Integer ≥ 2 holding the required number of clusters.

- **algo**: String specifying the spectral clustering algorithm to carry out:
    - 'rc' – Ratio-Cuts
    - 'nc' – Normalized-Cuts

Outputs

- **labels**: *m*-element vector of integers between *1* and *k*. A values of *r* in entry *i* means that x_i was assigned to cluster *r*.

Description

The function carries out Ratio-Cuts\Normalized-Cuts spectral clustering. Namely, it operates as follows:

1. Form $D = diag\left(W1\right)$
2. Define W' according to the algorithm specified in *algo*:
    a. Ratio-Cuts: $W' = W - D + I$
    b. Normalized-Cuts: $W' = D^{-1/2}WD^{-1/2}$
3. Solve:
$$V^* = \underset{V\in\mathbb{R}^{m,k}\ s.t.\ V^TV=I}{\arg\max}\ Tr\left(V^TW'V\right)$$
i.e. set the columns of V^* to hold the *k* leading orthonormal eigenvectors of W'.
4. Form the matrix Y by normalizing the rows of V^* to unit length:
$$Y_{ij} = V^*_{ij} \Big/ \sqrt{\sum_j V^{*2}_{ij}}$$
5. Apply k-means (using the function you have implemented in Part 1) to the rows of Y (corresponding to input points) to obtain the clustering. Assign *labels* output accordingly.

Part 4

In this part we will use the spectral clustering function implemented in Part 3 to segment the images of our dataset. We will apply the clustering to image pixels, without taking any measures to ensure sparsity of the affinity matrix. This approach is impractical even for moderately sized images, and we will therefore **rescale our images by a factor of 1/8 (in each axis) before segmenting them**. There are several methods to overcome this computational burden and apply spectral clustering to large images, one of which we will see in the following part of the assignment.

We define the affinity between pixels *r* and *s* in a color image to be:

$$W_{r,s} = \exp\left(-\frac{1}{2\sigma_c^{\ 2}}\left\|L^*a^*b^*\left(r\right) - L^*a^*b^*\left(s\right)\right\|_2^2 - \frac{1}{2\sigma_s^{\ 2}}\left\|XY\left(r\right) - XY\left(s\right)\right\|_2^2\right)$$

where:

- $L^*a^*b^*(r)$ and $L^*a^*b^*(s)$ stand for the L*a*b* color vectors of pixels *r* and *s* respectively.
- $XY(r)$ and $XY(s)$ stand for the spatial coordinate vectors of pixels *r* and *s* respectively.
- σ_c and σ_s are positive scalars parameterizing the affinity measure.

Construct an affinity matrix as above for each of the **rescaled** dataset images, and apply on it the spectral clustering function you have implemented in Part 3. Tune the parameters *k*, σ_c and σ_s to reach the best results you can. Add to your report the segmentation results and selected parameters for each rescaled image. Repeat this process for both Ratio-Cuts and Normalized-Cuts algorithms.

Part 5

The naïve spectral clustering approach we have undertaken in Part 4 is suitable only for very small images. In this part we will exploit an approximation technique named 'Nystrom Extension' to apply Normalized-Cuts segmentation to larger images.

In the root folder of the assignment you will find the file NystromNCuts.m, which implements the following function:

labels = NystromNCuts(A, B, k)

Inputs

- **A**: *n*-by-*n* real non-negative symmetric matrix holding the affinities between a selected sample of *n* input points (without loss of generality we denote these by $x_1, x_2, \ldots, x_n$).

- **B**: *n*-by-*(m-n)* real non-negative matrix holding the affinities between the selected sample ($x_1, x_2, \ldots, x_n$) and the rest of the input points ($x_{n+1}, x_{n+2}, \ldots, x_m$).

- **k**: Integer ≥ 2 holding the required number of clusters.

Outputs

- **labels**: *m*-element vector of integers between *1* and *k*, with the first *n* entries corresponding to the selected sample ($x_1, x_2, \ldots, x_n$) and the following *m-n* entries corresponding to the rest of the input points ($x_{n+1}, x_{n+2}, \ldots, x_m$). A value of *r* in entry *i* means that the corresponding input point was assigned to cluster *r*.

Description
The function uses the partial affinity information in the matrices *A* and *B* to apply approximate Normalized-Cuts clustering. Namely, it approximates (using the Nystrom method) the leading *k* eigenvectors of $D^{-1/2}WD^{-1/2}$, with *W* being the full affinity matrix. It then concatenates these approximate eigenvectors so that they form columns of a matrix, normalizes the rows of this matrix to unit length, and applies k-means clustering to the rows so that each input point is assigned to a cluster between *1* and *k*.
Details regarding the application of the Nystrom method to spectral clustering can be found in http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1262185&userType=inst.

Note:
For the application of k-means, *NystromNCuts* uses the k-means function you have implemented in Part 1. Without this function being visible, *NystromNCuts* cannot be run.

Use the function *NystromNCuts* to segment the (full-sized) images of our dataset. As before, the clustering will be applied to image pixels, with the same affinity measure as that used in Part 4. The *n*-sized pixel sample for which we compute all affinities (denoted $x_1, x_2, ..., x_n$ above) will be chosen randomly. Tune the parameters *k*, σ_c and σ_s to reach the best results you can. Add to your report the segmentation results and selected parameters for each image. Repeat this process with *n* being equal to 50, 100, 200 and 400. Is there a significant difference in terms of run-time or performance (segmentation quality)? Does the quality of your results depend on the chosen *n*-sized sample ($x_1, x_2, ..., x_n$)? Can you suggest alternative methods for choosing it?

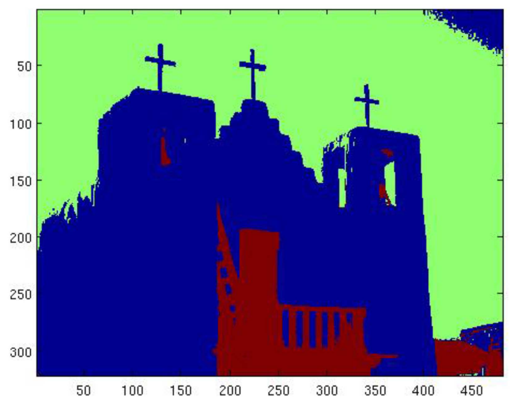
Bonus Part II

Repeat Part 5, while replacing the affinity measure with any measure of choice. Try to reach the best segmentation results you can. Add the results to your report along with a description of the affinity measure you used.

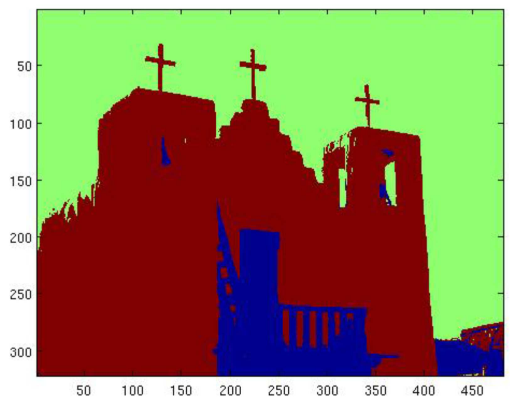
Appendix – Exemplar Results

As stated above, the following results do NOT represent a ‘correct solution’ that you should try to reach. Their sole purpose is to qualitatively illustrate the kind of results that can be reached in this assignment.

Part 2

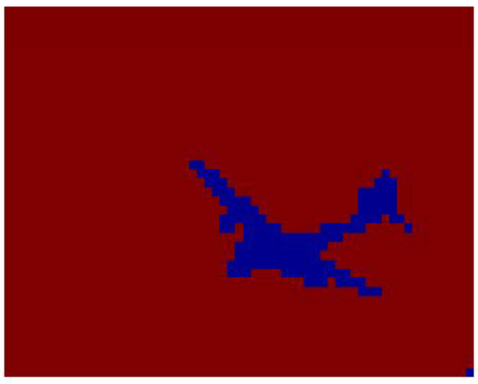


$k = 3, \mu = 0$

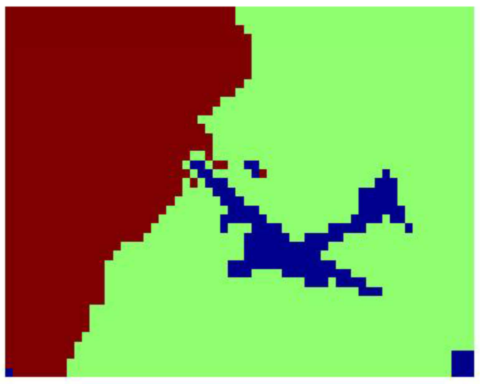


$k = 3, \mu > 0$

Part 4

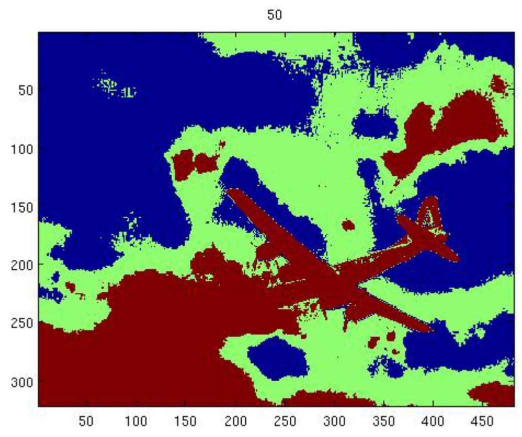


$k = 2$, Normalized-Cuts

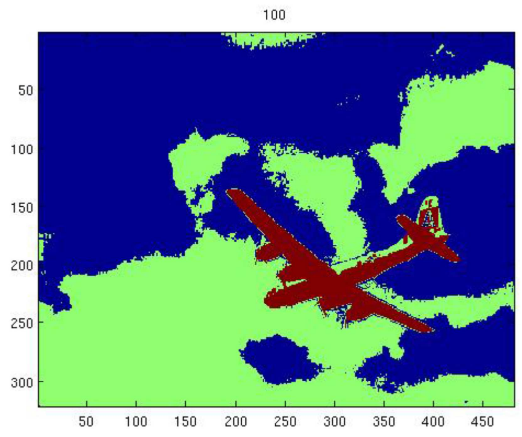


$k = 3$, Ratio-Cuts

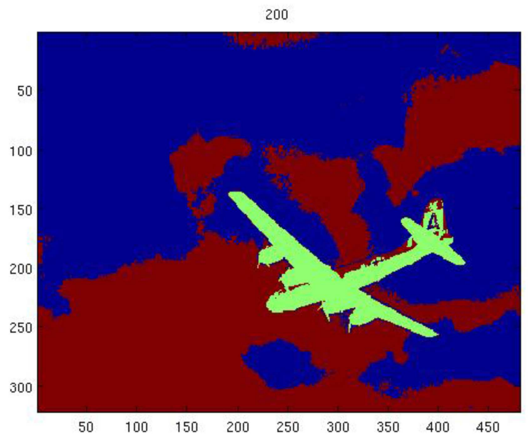
Part 5



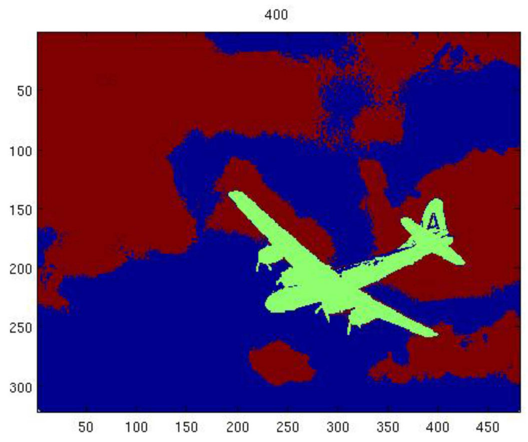
$k = 3, n = 50$



$k = 3, n = 100$



$k = 3, n = 200$



$k = 3, n = 400$