Computer Vision 1: Homework 11

Deadline 24.01. 12:15

Important: Submit your programming solutions through Moodle. The deadline for submitting your work is always on Thursday, at 12.15, the week after handing out the homework.
For other, non-programming homework, bring your solution with you to the exercise class. For each homework problem, one student will be chosen at random to present their solution.

Programming tasks.

This homework will be about K-means clustering, a building block of many segmentation algorithms in computer vision. There are no images in this homework. In fact, even our old friend `scikit-image` is not with us! Instead, we will be using `scikit-learn`, the big brother of `scikit-image`, for the first time in this course. As the name suggests, `scikit-learn` is about machine learning rather than image processing. This is an indication that our course has progressed farther away from the low-level image processing end and closer to the machine learning end of a spectrum.

For starter, please install `scikit-learn` on your computer following the instruction at `https://scikit-learn.org/stable/install.html`.

The dataset in this exercise is a synthetic dataset called `Aggregation`, it can be downloaded from `http://cs.joensuu.fi/sipu/datasets/` under the section `Shape sets`. The data is contained in a `txt` text file. It consists of 788 lines, each corresponds to a data point. Each data point has 2 attributes specified in the first and second columns. The last column specifies the label of the cluster to which the data point belongs. The data points belong to one of seven clusters. Figure 1 illustrates the ground truth clusters.

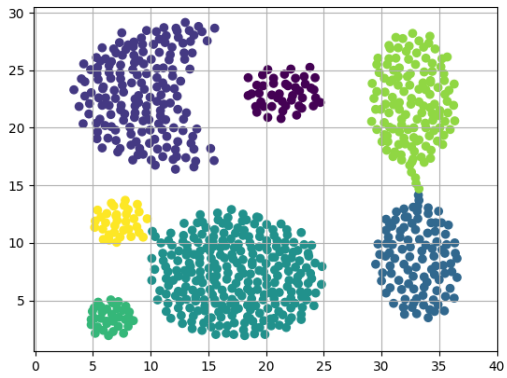

Figure 1: Visualization of the clusters in the Aggregation dataset. Different colors indicate different clusters.

- Using `numpy.loadtxt`, load the dataset into a numpy array. Remove elements of the third column in the array.
- Using `sklearn.cluster.KMeans`, cluster the data with different pre-defined number of clusters from the set {4, 8, 12}. To get ubiquitous result, use the same `random_state = 2019`.
- Using `matplotlib`, visualize the clustering results. Intuitively, which pre-defined number of clusters results in the best and the worst clustering?

- Set the pre-defined number of clusters to 788 (the number of data points). The random state is still 2019. Are there any empty clusters? If yes, print out the coordinates of these empty clusters. In any cases, explain briefly why there are / there are not empty clusters. Will the answer change if a different random state is used? Write your explanation and answer in a Python comment block at the end of your submission.

Note: it is sometimes not absolutely clear to tell if one clustering is better than the others, and to date there is no one single consensus measurement for determining what a good clustering is. Research papers often report performances on a variety of measurements, some of them are available at `https://scikit-learn.org/stable/modules/classes.html#clustering-metrics`.

Other tasks.

1. Consider the following image where the matrix entries represent intensity values:

$$I = \begin{bmatrix} 10 & 22 & 15 & 9 \\ 5 & 26 & 17 & 17 \\ 4 & 18 & 14 & 12 \\ 1 & 15 & 14 & 10 \end{bmatrix}.$$

   We perform one step of the k-means algorithm given two initial values for cluster centers, $c_1 = 8$ and $c_2 = 12$.

   (a) Write down two matrices D_i, $i = 1, 2$ where the entries in D_i correspond to distance of the corresponding element in I from c_i.

   (b) Given the matrices D_i, find the cluster assignment for each element in the image. Write the result as a matrix the same size as I, with entries equal either to 1 or 2 to indicate the cluster assignment of each element.

   (c) Using the cluster assignments, calculate the updated cluster centers $c_i^{(\text{new})}$.

   (d) Did the algorithm converge after this iteration?

2. We have developed a segmentation algorithm that has partitioned an image into segments S_i, $i = 1, \ldots, n$ as described on the lecture slides. For the same image, we have a *ground truth segmentation* G_j, $j = 1, \ldots, m$, which specifies how the image ideally should be partitioned.

   Intuitively, the *undersegmentation error* for an individual ground truth segment G_j measures the amount of "bleeding" that a segmentation exhibits beyond G_j. The undersegmentation error for a ground truth segment G_j is defined as

$$\text{UE}(G_j) = \frac{\left[\sum_{\{S_i | S_i \cap G_j \neq \emptyset\}} \text{Area}(S_i)\right] - \text{Area}(G_j)}{\text{Area}(G_j)}$$

   where the summation is over all segments S_i which have any overlap (non-empty intersection) with G_j, and the function Area returns the area, i.e., the number of pixels, of the corresponding segment.

   Our image has 12 pixels in it, and our segmentation algorithm produces the segments S_i shown in Figure 2. Calculate the undersegmentation errors $\text{UE}(G_1)$ and $\text{UE}(G_2)$ of the ground truth segments G_1 and G_2 shown in Figure 3.

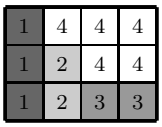

Figure 2: A segmentation S_i, $i = 1, 2, 3, 4$, with i indicated on each pixel.

Figure 3: Two ground truth segments G_1 (left) and G_2 (right) highlighted in gray.