TP1: Spectral Clustering

pierre (dot) perrault (at) inria.fr
omar (dot) darwiche-domingues (at) inria.fr

October 15, 2019

Abstract

In this practical session we will cover fundamental graph building techniques, and apply them to the Spectral Clustering problem. The session will be evaluated on a short written report and a final image segmentation implementation. During the TD we will implement all the necessary tools to do the segmentation and answer the report questions. The report and the code are due in 2 weeks (deadline 23:59 29/10/2019). You will find instructions on how to submit the report on piazza, as well as the policies for scoring and late submissions. All the code related to the TD must be submitted, to provide background for the image segmentation code evaluation.

1 Graph construction

The file `generate_data.py` contains functions that will generate artificial data for the experiments, as described below. You can run the script to visualize the data.

> **N-Blob**: Sample random points in $\mathbb{R}^2$ according to N Gaussian distribution with mean $\mu_i = [\cos(2\pi i/N), \sin(2\pi i/N)]$ and variance $\text{Diag}(\sigma^2)$. As an example, 3-Blob is a dataset generated using this distribution and three clusters.
>
> **Two Moons**: Sample random points shaped as two intertwined moons.
>
> **Point and Circle**: Sample random points from a concentrated Gaussian point in the middle and a wide circle around it.

A prerequisite to build a similarity graph is to define a similarity function to score the distance between nodes in the graph. For the rest of the session we will use an inverse exponential function as the similarity measure, controlled by the euclidean distance.

$$d(x_i, x_j) = \exp\left\{-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}\right\}$$

The variance σ^2 of the Gaussian will control the bandwidth of the similarity.

In the file `build_similarity_graph.py`, do the following:

- Write the code to build an ε graph and an (OR) k-nn graph. More details on these terms are contained in the source code of the file.
- Use the function `plot_similarity_graph` to visualize the graph for some generated data. The function `plot_graph_matrix`, in the file `utils.py` might also be useful.
- Complete the function `how_to_choose_epsilon`. You may use the function `min_span_tree` in `utils.py`.

Answer the following questions:

1.1. What is the purpose of the option parameter in `worst_case_blob` (in the file `generate_data.py`)?

1.2. What happens when you change the generating parameter of `worst_case_blob` in `how_to_choose_espilon` and run the function? What if the parameter is very large?

1.3. Using `plot_similarity_graph` and one of the datasets, compare k-nn to ε graphs. When is it easier to build a connected graph using k-nn? When using ε graphs?

2 Spectral Clustering

In the file `spectral_clustering.py`, do the following:

- Complete the function `build_laplacian`.
- Complete the function `spectral_clustering`.

2.1. Build a graph starting from the data generated in `two_blobs_clustering`, and remember to keep the graph connected. Motivate your choice on which eigenvectors to use and how you computed the clustering assignments from the eigenvectors. Now compute a similar clustering using the built-in k-means and compare the results.

2.2. Build a graph starting from the data generated in `two_blobs_clustering`, but this time make it so that the two components are separate. How do you choose which eigenvectors to use in this case? Motivate your answer.

So far we only considered clustering with $c = 2$, and we did not establish a systematic rule on how to select eigenvalues. Do the following:

- Complete the function `spectral_clustering_adaptive`, which is almost identical to `spectral_clustering`, except for the eigenvectors selection, which must use the function `choose_eig_function`.

2.3. Look at the function `find_the_bend`. Generate a dataset with 4 blobs and $\sigma^2 = 0.03$. Build a graph out of it and plot the first 15 eigenvalues of the Laplacian. Complete the function `choose_eig_function` to automatically choose the number of eigenvectors to include. The decision rule must adapt to the actual eigenvalues of the problem.

2.4. Now increase the variance of the Blobs to $\sigma^2 = 0.20$ as you keep plotting the eigenvalues. Use the function `choose_eig_function`. Do you see any difference?

2.5. When you built the cluster assignment, did you use thresholding, k-means or both? Do you have any opinion on when to use each?

2.6. What is another use that looking at the distribution of the eigenvalues can have during clustering, beside choosing which eigenvectors to include?

We will now consider more complex structures than Gaussian blobs.

- Complete the function `two_moons_clustering`
- Complete the function `point_and_circle_clustering`.

2.7. Plot your results using spectral clustering and k-means in `two_moons_clustering` and compare the results. Do you notice any difference? Taking into consideration the graph structure, can you explain them?

2.8. In the function `point_and_circle_clustering`, compare spectral clustering using the normal laplacian L and the random-walk regularized Laplacian L_{rw}. Do you notice any difference? Taking into consideration the graph structure, can you explain them?

How did you choose the parameters when building the graph? We will compare clustering solutions while we change one of the parameters. Evaluating clustering is not straightforward, since the final labeling is arbitrary and not related to the original labels (in the rare cases when talking about labels make sense). In particular, a labeling obtained with a clustering algorithm might coincide with the true labels, but with the label indices arbitrarily reshuffled (e.g. points in class i labeled as j and points in class j labeled as i). To evaluate the assignment we will use the Adjusted Rand Index score [1]. The Rand Index takes as input two labelings of a dataset and outputs a value between 0 (unrelated) and 1 (equal) comparing how much these labeling overlap. The Adjusted RI gives a better estimation for large number of clusters at the expense of some (possibly unsatisfied) statistical assumption.

2.9. Complete the function `parameter_sensitivity`, and generate a plot of the ARI index while varying one of the parameters in the graph construction ε or k. Comment on the stability of spectral clustering.

2.10. If we did not have access to *true*[1] labels how could we evaluate the clustering result (or what should we not use as evaluation)?

[1] Definitions of true may vary.

3 Image Segmentation

Your final task, that you will complete after the class, is implementing image segmentation using spectral clustering. As a simple example we will segment images based on colors. In order to do this you must complete the file `image_segmentation.py`. Here are some pointers (these are only suggestion)

- The images provided are 50×50 pixels RGB images in the data folder. You need to specify them as input to the function `image_segmentation`. Inside the script, the images is loaded as a $50 \times 50 \times 3$ matrix, and converted into an 2500×3 matrix X where each pixel is a $\mathbb{R}^3$ vector in RGB space.
- The image was chosen so that it could be easily segmented using colors. The easiest way is to build a graph based on color distance between individual pixels and cluster them.

The function `image_segmentation` will plot your segmentation. Because in clustering the label are arbitrary the colors might be wrong. This does not matter as long as the separation is correct.

Final questions:

3.1. The first documentation is the code. If your code is well written and well commented, that is enough. If you think you need to better express some of the choices you made in the implementation, you can use this question in the report. Remember to include all the related code in the submission.

3.2. A full graph built between the pixels of a 50×50 image corresponds to 50^2 nodes. Solving the full eigenvalue problem in this case would scale in the order of 2^{34}. Even on weak hardware (i.e. iPhone) this takes only seconds to minutes. Segmenting a Full HD picture of 1920×1080 would scale in the order of 2^{64} (about a month on a decent machine i.e. not an iPhone). Beyond that, the large picture would require to store in memory a graph over millions of nodes. A full graph on that scale requires about 1TB of memory. Can you think two simple techniques to reduce the computational and occupational cost of Spectral Clustering?

3.3. Did you use `eig` or `eigs` to extract the final eigenvectors? Shortly, what is the difference between the two? How do they scale to large graphs (order of complexity)?

References

[1] Silke Wagner and Dorothea Wagner. *Comparing clusterings: an overview.* Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007.