ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Signal and Information
Processing Laboratory
Institut für Signal- und
Informationsverarbeitung

Fachpraktikum Signalverarbeitung

SV7: K-means and Spectral Clustering

1 Introduction

Clustering has been a prominent, successful and challenging topic for years. The task is to group similar objects together. Clustering is a method from a larger family of methods called unsupervised learning algorithms. The paradigm beneath these methods assumes that no labeled training data are available. As an example, imagine a group of persons of which a lot of personal details are given, but not their favorite music genre. How would one assign these persons to different party cliques (based on the assumption that a common taste of music is the most important criteria). Typically, we might expect that persons with similar characteristics (e.g. age, gender and fashion style) like the same kind of music. Therefore, identifying such groups, leads us to desired conclusion. Obviously, unsupervised algorithms and clustering in particular, have a plethora of applications in various fields (e.g. vector quantization, grouping proteins, density estimation).

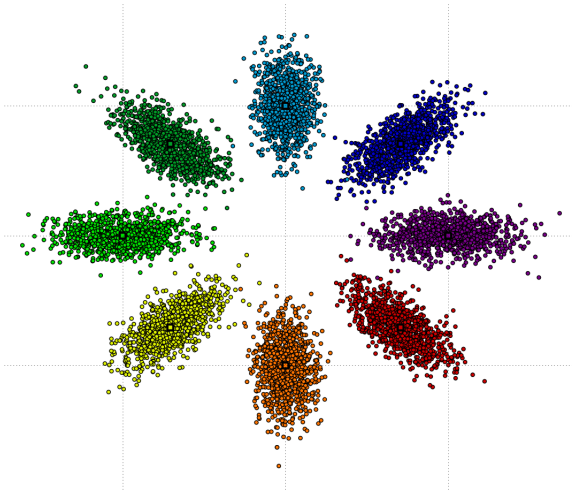

Figure 1: Gaussian flower

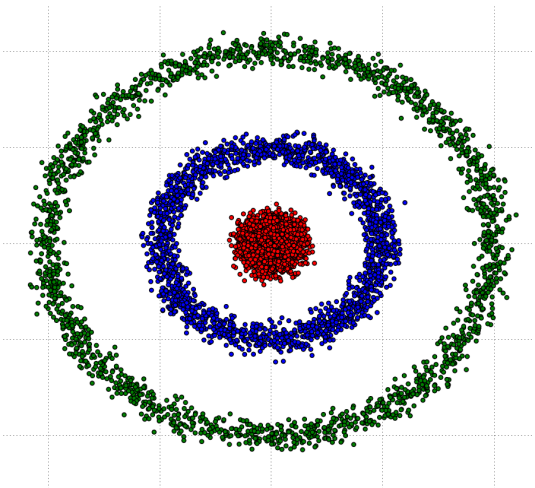

Figure 2: Three Rings

Given a fixed number K of clusters and n objects $x_1, \ldots, x_n$, we want to partition the n objects in K clusters. The general rule for clustering is that similar objects should belong to the same cluster. For instance, looking at the Gaussian flower in Figure 1, we expect that the clustering of the points for $K = 8$ will approximately assign all points of one

cloud (or petal) to the same cluster. Similarly, for the three rings in Figure 2, we would like to assign all points of one ring to the same cluster.

Depending on the objects available and more specifically on the object space, different approaches can be developed. In this work, we will be concerned with two different representations of the objects. The first case is when the objects $x_1, \ldots, x_n$ are vectors of $\mathbb{R}^N$. This space being Euclidean, it is naturally equipped with a norm and an inner product, which can be used to measure distances between vectors. In this representation, we can directly manipulate the objects. Very often, x_i corresponds to a vector of observations and clustering those vectors may be seen as grouping observation vectors generated by a same noisy process. The second case is when the objects $x_1, \ldots, x_n$ are not known directly but instead we have access to a pairwise measure between x_i and x_j. This measure can be either a similarity or a dissimilarity measure between the objects. This second representation is linked with graph clustering by the following analogy: an object x_i is a node i and the measure $\mu_{i,j}$ between objects x_i and x_j is a weighted edge between nodes i and j. The goal is to group similar nodes into a same cluster. In Figure 3, we depict such a graph with a possible partitioning.

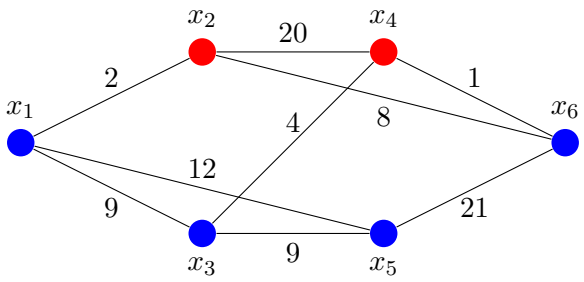

Figure 3: Partitioning a weighted graph

In this work, we first study the popular K-means clustering used to groups vectors of $\mathbb{R}^N$. We then apply it for clustering data points generated by normal distributions and for quantifying colored pixels in images. In the second part, we derive and implement a spectral clustering algorithm to group areas of research from an Electrical Engineering department. Clustering algorithms and simulations are implemented in Python[1]. Additionally, a useful library called Numpy is used to implement image processing and linear algebra.

2 K-means Clustering

2.1 Problem Setup

In this setting, n objects $x_1, \ldots, x_n$ which are vectors of $\mathbb{R}^N$ are given. It is assumed that the data can be well separated into a fixed number of K clusters. The problem setup of K-means clustering is then: Find K prototype vectors $y_1, \ldots, y_K$ (also called centroids) and K associated clusters (i.e. sets of data objects) $C_1, \ldots, C_K$, such that the following cost

$$R^{K\text{-}means}(y_1, \ldots, y_K, C_1, \ldots, C_K) = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - y_k\|^2 \tag{1}$$

[1] `http://www.python.org/`

is minimal. Each cluster can be either identified by the set C_k or its prototype vector y_k. In fact, y_k should be the best explanation for the objects of cluster k: $x_i \in C_k$.
Clearly, a direct joint optimization over the $y_k \in \mathbb{R}^N$ and the sets C_k is not feasible due to a high complexity. Indeed, clustering n objects into K clusters has in the order of K^n possible configurations, which is exponential in the number n of objects. The exact solution of this problem even belongs to the class of NP-hard problems, which are usually computationally not feasible.

2.2 K-means Algorithm

Also known as the Lloyd's algorithm, the K-means algorithm intends to find an approximate (suboptimal) solution of the K-means clustering problem. We will briefly see how this algorithm can be derived. The trick is to use a cyclic minimization. Instead of minimizing jointly $R^{K\text{-}means}$ over the variables $y_1, \ldots, y_K$ on one hand and the cluster assignments $C_1, \ldots, C_K$ on the other hand, we decompose the minimization in two steps where we first minimize $R^{K\text{-}means}$ over the C_k letting all the y_k fixed and then minimize over the y_k letting all the C_k fixed. We repeat these two steps until convergence or a certain time ends, hoping that $R^{K\text{-}means}$ is minimized. This algorithm makes sense only if the two minimization steps can be done easily. Let us have a look at these two steps.

First fix all the prototypes y_k and search for the optimal clusters C_k. Looking at the cost function, it can be seen that only a single term of the sum is modified if a particular x_ℓ is moved to a cluster. Thus we can conclude that the quadratic terms $\|x_\ell - y_k\|^2$ for each ℓ can be minimized separately. Since the optimization runs over the clusters C_k, each sub-problem corresponds to assigning x_ℓ to the closest (in an Euclidean sense) cluster set C_k with prototype y_k. On the other hand, the minimum value with fixed prototypes y_k are then clusters C_k, composed of all objects x_ℓ that are closest to y_k.
In the second step, clusters C_k are held fixed and we are looking for the optimum centroids y_k. The cost function is a continuous function in y_k. Hence, the minimum value can be found by setting the first derivatives with respect to all y_k to zero. Deriving the cost function in Equation 1 with respect to y_p, leads to:

$$\begin{aligned}
\frac{\partial R^{K\text{-}means}}{\partial y_p} &= \sum_{k=1}^{K} \sum_{x_i \in C_k} \frac{\partial \|x_i - y_k\|^2}{\partial y_p} \\
&= \sum_{x_i \in C_p} \frac{\partial \|x_i - y_p\|^2}{\partial y_p} \\
&= \sum_{x_i \in C_p} 2y_p - 2x_i \\
&= 2|C_p| \left(y_p - \frac{1}{|C_p|} \sum_{x_i \in C_p} x_i \right)
\end{aligned}$$

Where $|C_p|$ is the cardinality of C_p (the number of elements). The condition of optimality follows readily:

$$y_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i \ , \forall k \ . \tag{2}$$

We can now formulate the algorithm:

Algorithm 1 K-means Algorithm

Initialization: $y_1, \ldots, y_K$

repeat

- Assign each data points x_i to the closest prototype vector y_k. C_k is then the cluster containing the objects x_i which are closest to y_k.
- Update the new centroids according to: $y_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i \quad , \forall k \in [\![1, K]\!]$.

until convergence or allocated time ends

It can be shown that the cost function $R^{K\text{-}means}$ is monotonically non-increasing. As this cost is non negative, it is therefore ensured that the algorithm converges[2]. However, as this algorithm only approximately solves the K-means problem, it is only guaranteed to converge to a local minimum of $R^{K\text{-}means}$ but not to the global minimum. Getting stuck in a local optimum is actually the main drawback of this algorithm. To overcome this issue, careful initialization (e.g., choosing starting centroids as far as possible) may help but increases the computational complexity. Another method to avoid this effect, is to randomly choose K different prototype vectors $y_1, \ldots, y_K$ among the data vectors $x_1, \ldots, x_n$.

Question 1. Before implementing K-means algorithm, try to apply it manually to Figure 4 in order to group the 8 points into $K = 2$ clusters. First, initialize the prototypes y_1 and y_2 with one point from the blue square and one from the red one. In Figure 4, draw the prototype vectors and the clusters at each iteration. After convergence (only two iterations are needed), evaluate the final cost $R^{K\text{-}means}$. Now, start again the algorithm but initialize the two prototype vectors with the most left points (draw also the solution). After convergence, compute the final cost function and compare it to the previous one. What can you observe?

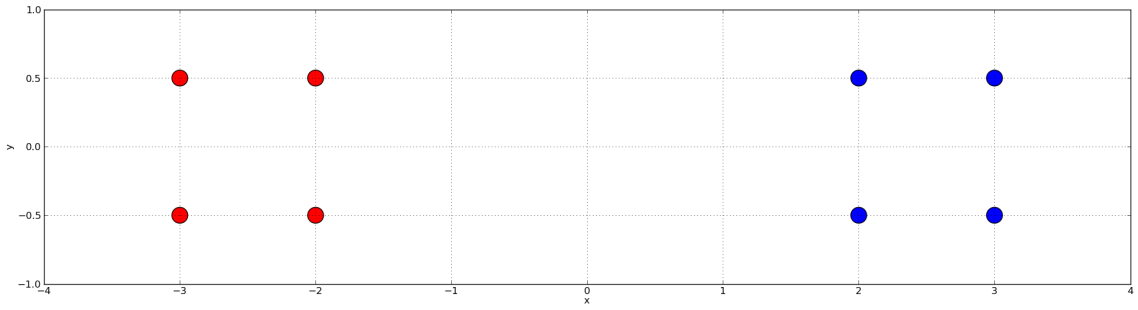

Figure 4: Two Squares

Literature The standard K-means clustering algorithm was first proposed by Lloyd (who was working in Bell Labs) in 1957 and published only in 1982 in [1]. In the meantime, Forgy published the same algorithm in 1965 in [2]. To solve the K-means problem, other similar algorithms can be derived from the original one. A popular and slightly more efficient algorithm was proposed by Hartigan et al. in [3].

[2]Furthermore it can be proven that the time complexity is polynomial in the number n of data vectors.

2.3 Implementation

2.3.1 Preliminaries

A short tutorial on Python and the library Numpy is available in the Appendix. Please spend 5 minutes of your time to familiarize yourself with this easy programming language. Open a terminal, go to your home directory and type:
/home/isistaff/glf/fachprak_isi/SV7/copySV7.sh
Use a text editor (e.g Bluefish) in order to open the Python scripts *.py. Ignore Gtk-Warnings if any.

2.3.2 K-means function

Question 2. Open func.py and look at the first function Kmean. It takes n column vectors (put into a matrix V), a number of cluster K and a maximum number of iterations *maxit* as arguments. The outputs are the assignment vector *classi* containing n elements taking values between 0 and $K-1$ and the prototype column vectors $y_1, \ldots, y_K$ put into the matrix *proto*. In line 70, write the update rule of the prototype vectors.

2.3.3 Clustering Samples from Normal Distributions

In this first simulation, we use K-means algorithm to cluster $2D$-points generated by Gaussian distributions. Initially, we have points from only two Gaussian distributions of means $\mu_0 = \begin{bmatrix} -1 & -1 \end{bmatrix}^\top$, $\mu_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top$ and covariance matrices $\Sigma_0 = \sigma_0^2 I_2$, $\Sigma_1 = \sigma_1^2 I_2$. With $\sigma_0 = 0.5$ and $\sigma_1 = 0.3$, a typical data set is plot in Figure 5.

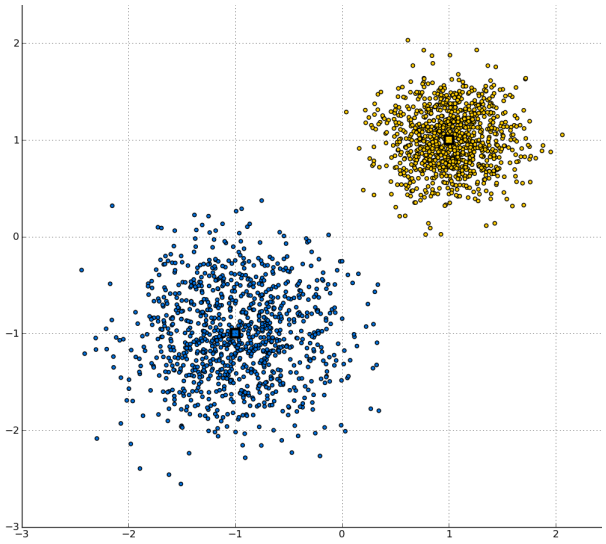

Figure 5: Points generated by two Gaussian distributions

Without knowing the parameters (means and covariance matrices) of the Gaussian distributions, the goal is to group together data points generated by the same distribution. Open plotdata.py with a text editor.

Question 3. In line 29, call the function Kmean from the module func.py and with the right parameters. Save the file. Then, using a terminal, go to your directory containing the files and type: ./plotdata.py. Observe the results. Are they good? Can you observe any clustering errors?

Question 4. Now set σ_0 to 1.2, save the file and run the python script. Observe the results and repeat this procedure for $\sigma_0 = 3$ and $\sigma_0 = 5$. Comment the results.

Question 5. Set σ_0 to 0.5 and modify $K = 3$. Run the Python script several times and observe the plots. Is there something remarkable about the clustering result? By repeating this simulation many times, do you expect a result in which a cluster contains points from both the two original clouds? Why? What can you conclude on the way to select the parameter K?

Question 6. Adapt the code in the file plotdata.py to generate a third cloud of data points from a $2D$ normal distribution with mean $\mu_2 = \begin{bmatrix} -1 & 1 \end{bmatrix}^\top$ and covariance matrix $\Sigma_2 = \sigma_2^2 I_2$.

2.3.4 Quantizing Pixel Values

In this second simulation, we use K-means algorithm to quantize colored pixels of an image. A raw image (e.g opened with Python) can be seen as a matrix of size $m \times n$ where each element corresponds to a pixel. In the RGB format (red, green, blue), one pixel has three components ($3D$ vector) and each component indicates the level of each basic color (R, G and B). The values are coded with 1 byte, that is to say a range from 0 to 255. A colored pixel is then defined by a $3D$ vector such as [128 0 26]. Using this convention, there are $(2^8)^3$ possible colors. Some basic colors are: [255 255 255] white, [0 0 0] black, [255 0 0] red, [0 255 0] green and [0 0 255] blue.

Question 7. Knowing that the color yellow is composed of red and green, what is the RGB vector of the color yellow?

Using K-means, the goal of this experiment is to smartly (re-)quantize the colored pixel with only K possible vectors instead of $(2^8)^3$. In Figure 6, we can observe an original image with 3 bytes per pixel (1 byte per basic color). In Figure 7, we show the resulting quantized image where we have applied K-means quantization with $K = 8$. It means that the algorithm has selected 8 colors among the $(2^8)^3$ and has assigned one color (among the 8 selected colors) to each pixel. In Figure 8, we also observe the result of a uniform quantization with $K = 8$ colors. The space $[\![0, 255]\!]^3$ is partitioned into $K = 8$ cubes of the same length and each original pixel belonging to a cube is assigned to the cube center.

Figure 6: Original Image with $(2^8)^3$ colors (3 bytes)

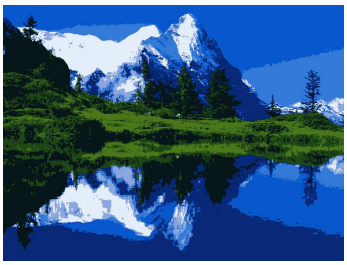

Figure 7: K-means Quantization with $K = 8$ colors

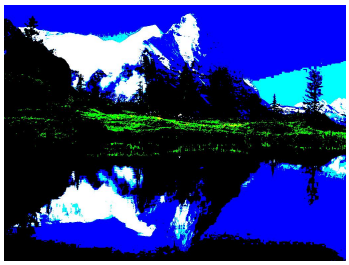

Figure 8: Uniform Quantization with $K = 8$ colors

Open image_seg.py with a text editor. We use the library matplotlib.image for image handling.

Question 8. Observe the code. There is nothing to add in this script. Notice how the image is handled. We first rearrange the pixels by concatenating each rows of the image into a unique row (using np.reshape). Then we cluster the $3D$ pixels using K-means. In

the terminal, run: ./image_seg.py
You may need to wait a little bit (less than one minute, depending on your computer). Can you still recognize the main object in the picture? Note that only $K = 8$ pixel values are used. Which colors have completely disappeared? Why?

Question 9. Assume we apply a uniform quantization to this same image where the 8 possible colors are defined by the $3D$ vectors whose components take either the value 0 or 255. First, find the common names of these 8 colors. Then, can you predict the result of the uniform quantization algorithm? Which colors are likely to disappear?

Question 10. One can say that uniform quantization is a special case of K-means quantization. How can it be? What is the main advantage and the main drawback of K-means quantization compared to uniform quantization?

Question 11. Change the number of clusters to $K = 4$ and then $K = 2$. Try also with other images test2.jpg and test3.jpg. You can also try out with images of your own, just copy them into your working folder and be careful about the resolution (the higher it is the longer it takes).

Question 12 (Optional)**.** If your are fast enough, implement the uniform quantization algorithm and compare the results with K-means quantization. Some useful lines of code may be helpful if you uncomment the lines 25 to 31.

2.4 Limits of K-means

The two applications we used show how useful K-means algorithm can be. Especially for colored-pixel quantization, the results are quite impressive compared to a uniform quantization algorithm. However this algorithm has several drawbacks and restrictions:

a) We have already seen that Algorithm 1 may get stuck into a local minimum (e.g. Figure 4).

b) In the introduction, we have seen that the general rule for clustering is to group similar objects together. Nevertheless, in general, we do not have a clear definition of similarity. In K-means clustering, the dissimilarity measure we use is implicitly the squared distance between vectors: $\mu_{i,j} = \|x_i - x_j\|^2$. We then group vectors that are close to each other. Unfortunately, the distance between vectors may not be the most suitable metric. Indeed, looking at Figure 9, a human observer can easily distinguish three different groups of points. Applying K-means algorithm to these data will never succeed in finding the grouping solution we are looking for because the distance between vectors is not the right dissimilarity measure to use.

   Question 13. In Figure 9, try to predict the result of K-means algorithm for $K = 3$ (draw the clusters) and check your prediction by running the script rings.py. Do you think K-means algorithm can still give the result we expect if we do some pre-processing on the data (e.g. feature extraction)? If yes, explain how to do it.

c) K-means clustering can be used only if the objects x_i are vectors of an Euclidean space. Therefore, there are many situations in which we can simply not apply this algorithm. For example, imagine we want to cluster words in a text. The idea can be to regroup all declinations and forms of a same word into one cluster. In this situation, we cannot use vectors any more because words do not have the same length and no operations are defined. In order to circumvent this problem, the idea is not to care about the objects (the words) in themselves but rather to know how

similar the objects are between each other. This new approach is developed in the next part. Notice that K-means is a special case where the dissimilarity measure corresponds to the distance between vectors.

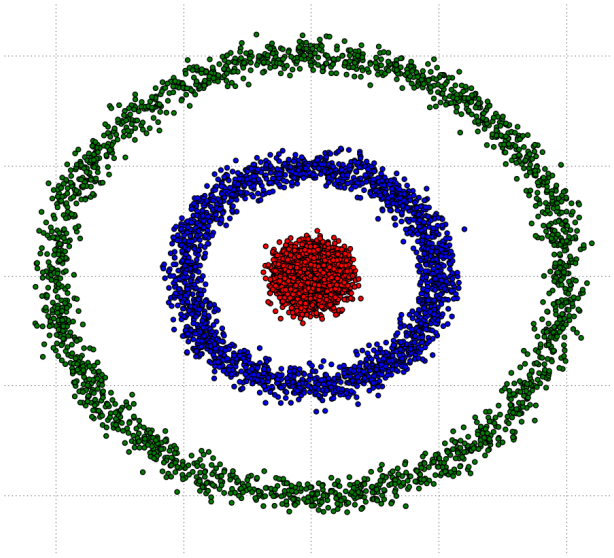

Figure 9: Three Rings

3 Spectral Clustering

3.1 Introduction

Suppose we have n objects $x_1, \ldots, x_n$ that we want to cluster into K groups. We have access to a similarity measure $S_{i,j}$, $(i,j) \in [\![1,n]\!]^2$ where $S_{i,j}$ is a positive real value indicating how similar objects i and j are. For simplicity, we assume $S_{i,j} = S_{j,i}$. The problem can be represented using an undirected weighted graph where a node is an object and an edge between nodes i and j is weighted by the value $S_{i,j}$. In Figure 10, we give an example of such a graph. If there is no edge between two objects, it simply means that their similarity is zero. The graphical structure allows us to use and apply well-known operators and algorithms on graphs, as we will see in the following. The second way of representing our problem is to use the matrix S containing the pairwise similarities $S_{i,j}$ between objects x_i and x_j. Define an adjacency matrix by writing similarities $S_{i,j}$ in matrix-form. The matrix representation corresponding to the example in Figure 10 is displayed in Figure 11. It is a symmetric matrix with zero diagonal elements.[3]

Analogously to clustering in $\mathbb{R}^N$, which was demonstrated before, the goal of graph clustering is to find groups of vertices on a graph that are similar to each other. As a practical example, consider the problem setup shown at the end of this chapter. In this problem, the goal is to identify large and comprehensive research areas. The data consists of interest data from a group of professors. Each professor had to grade a list of research areas according to their interest. Suppose that professors will not choose randomly very different research areas, but areas that are related to each other. Thus considering two

[3]One can argue that the similarity $S_{i,i}$ between an object x_i and itself should be as big as possible (or even $+\infty$). In fact S should be seen as an adjacency matrix where a strictly positive coefficient indicates both the presence of an edge and a strictly positive weight. Then the zero coefficients indicate the absence of an edge. As the nodes are not linked with themselves, the S matrix has a zero diagonal.

topics, the more professors that are interested in both topics at the same time, the more similar the research areas ought to be. Towards the goal of finding common research themes, one could now represent each research area in the questionnaire as a vertice with edges based on the similarity and apply a graph clustering method to this graph.

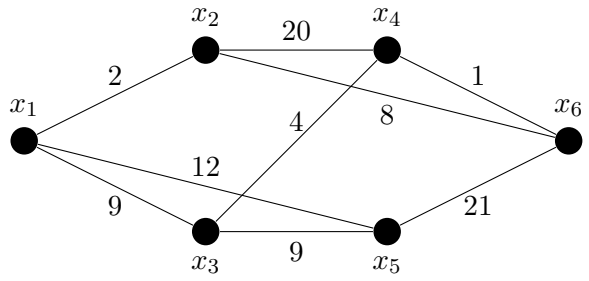

Figure 10: Undirected weighted graph

$$S = \begin{bmatrix} 0 & 2 & 9 & 0 & 12 & 0 \\ 2 & 0 & 0 & 20 & 0 & 8 \\ 9 & 0 & 0 & 4 & 9 & 0 \\ 0 & 20 & 4 & 0 & 0 & 1 \\ 12 & 0 & 9 & 0 & 0 & 21 \\ 0 & 8 & 0 & 1 & 21 & 0 \end{bmatrix}$$

Figure 11: Matrix representation of undirected weighted graph

Remarks

- In some applications, instead of having a similarity measure we only have access to a dissimilarity D. Using a suitable transformation, we can change dissimilarities into similarities (e.g. $S_{i,j} = \exp(-D_{i,j}/\delta)$ or $S_{i,j} = \max_{k,l} D_{k,l} - D_{i,j}$).
- Depicting our problem into a graphical representation is natural but may not seem useful in our situation. Be aware that in practical applications it is not always clear what the nodes, edges and similarities are. We can even define several meaningful graphs for the same clustering problem.
- The graphical representation we used is quite simple because we restrict ourselves to undirected weighted graphs with positive weights. Graph clustering theory is a huge topic and can be applied to various types of graph: weighted and unweighted, directed and undirected...

3.2 Spectral Clustering Theory

In this section, we briefly describe the theory behind spectral clustering. All graphs are considered undirected and weighted with positive values.

3.2.1 Graph Clustering

Graph clustering is the task of partitioning a graph $(\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ is the set of nodes and for simplicity, we assume they are labeled from 1 to n such that $\mathcal{V} = \{1, \ldots, n\}$. $\mathcal{E}$ is the set of edges and $\mathcal{E}_{i,j}$ is the edge between nodes i and j with the weight $w_{i,j}$. We also define K clusters $\mathcal{G}_\nu$, $\nu = 1, \ldots, K$, which represent sets of nodes. By definition of clustering, the sets $\mathcal{G}_\nu$ are disjoint. Analogous to the presentation of clustering in the previous chapter, we first introduce cost functions that measure the quality of a clustering.
To obtain a measure of a clustering on a graph, some notation is introduced. Let A and B be two subsets of the nodes $\mathcal{V}$. We can define two useful quantities:

$$\text{cut}(A, B) = \sum_{i \in A,\ j \in B} w_{i,j} \tag{3}$$

$$\text{assoc}(A) = \text{cut}(A, \mathcal{V}) = \sum_{i \in A,\ j \in \mathcal{V}} w_{i,j} \tag{4}$$

$\text{cut}(A, B)$ measures the interaction between the two sets A and B of nodes. $\text{assoc}(A)$ measures the interaction of the nodes in set A with all the nodes of the graph, even those from A. These two quantities are helpful to define cost functions for graph partitioning. In this work, we present the cut cost function and the normalized-cut cost function introduced by Shi and Malik in [4]. The cut cost function is:

$$R^C(\mathcal{G}_1, \ldots, \mathcal{G}_K) = \sum_{\nu=1}^{K} \text{cut}(\mathcal{G}_\nu, \mathcal{V} \backslash \mathcal{G}_\nu) \tag{5}$$

where $\mathcal{V} \backslash \mathcal{G}_\nu$ is the set of all nodes of the graph except those belonging to cluster $\mathcal{G}_\nu$. Observe that the expression sums the cut cost of all K clusters. In the simplest case $K = 2$, minimization of this cost is known as the min-cut problem (i.e., to find a cut of a weighted graph into two parts with minimum cost).

The normalized-cut cost expression is:

$$R^{NC}(\mathcal{G}_1, \ldots, \mathcal{G}_K) = \sum_{\nu=1}^{K} \frac{\text{cut}(\mathcal{G}_\nu, \mathcal{V} \backslash \mathcal{G}_\nu)}{\text{assoc}(\mathcal{G}_\nu)} \tag{6}$$

Compared to Equation 5, the normalized-cut cost function weighs each cut cost term by the total interaction $\text{assoc}(\mathcal{G}_\nu)$. The term $\frac{\text{cut}(\mathcal{G}_\nu, \mathcal{V} \backslash \mathcal{G}_\nu)}{\text{assoc}(\mathcal{G}_\nu)}$ measures how strongly $\mathcal{G}_\nu$ interacts with the complete graph $\mathcal{V}$. Conversely, a small value indicates that $\mathcal{G}_\nu$ can be separated well from the graph. Minimizing this cost thus leads to a partition of the graph and edge cuts such that the sum of the average interaction of each cluster with the rest of the nodes is minimal: we want clusters to be as independent as possible. Of course, other cost functions can be used but the normalized-cut has a very nice property as we will see in the next part.

3.2.2 Spectral Clustering

To obtain a clustering algorithm that (approximately) minimizes the given cost functions, we look at spectral clustering methods. The general idea of spectral clustering is to define a symmetric matrix L on a graph using the adjacency matrix S and/or other matrices such as the degree matrix. Then an eigenvalue decomposition is performed on the matrix L: $L = U \Lambda U^\top$. Depending on the application, a dimensionality reduction procedure may be carried on by removing some eigenvalues. Finally, the remaining row vectors contained in U (or sometimes in $U\tilde{\Sigma}$) are clustered using K-means algorithm. As a result, as each row vector corresponds to one node, we have obtained our partitioning. In the following, we will call the matrix L the graph Laplacian and detail spectral clustering algorithms that find an approximate minimum clustering for the cost functions given in Equation 5 and Equation 6.

Spectral Clustering Algorithm The graph Laplacians we need are defined by:

$$L^C = D - S \ , \tag{7}$$

and

$$L^{NC} = I - D^{-1/2} S D^{-1/2} \ , \tag{8}$$

with I the identity matrix and D the diagonal matrix with diagonal elements:

$$D_{i,i} = \sum_{j=1}^{n} S_{i,j} \tag{9}$$

Recall the similarity matrix S (or adjacency matrix) defined in the last subsection. It can be shown that the graph Laplacian L^C and L^{NC} are linked to the cost functions R^C respectively R^{NC} [4]. This means that performing spectral clustering on one of the latter graph Laplacians corresponds to graph clustering with respect to the cost functions R^C and R^{NC}.

Given a graph Laplacian, spectral clustering proceeds to compute the eigenvalue decomposition of L: $L = U\Sigma U^\top$. Then we choose the K smallest eigenvalues and extract the matrix $\tilde{U}$ which contains the K columns of U corresponding to theses values. $\tilde{U}$ is of dimension $n \times K$. Finally, we apply K-means algorithm to cluster the n row vectors of $\tilde{U}$. Node i is assigned to the cluster of the i^{th} row vector of $\tilde{U}$.

Algorithm 2 Spectral Clustering

- Compute the graph Laplacian L (Equation 7 or 8)
- Perform an eigenvalue decomposition: $L = U\Sigma U^\top$
- Extract $\tilde{U}$ by taking the K columns of U corresponding to the K smallest eigenvalues
- Cluster the row vectors of $\tilde{U}$ using K-means

The spectral clustering algorithm described above is quite simple but nonetheless powerful.

Summary The term Spectral Clustering is used when the task of minimizing a cost function (defined on the initial graph) reduces to define an operator T (in a simple case it is a matrix) with which the minimization problem becomes an eigenvalue decomposition with some dimensionality reduction and a K-means clustering applied to a submatrix of the eigenvectors. Mathematically, we embed the initial objects into an Euclidean space where the initial cost function R^{NC} changes to $R^{K\text{-}means}$ in this new space.

3.3 Implementation

3.3.1 From the similarity to vectors

Question 14. Open func.py. Write the function sim_to_vect which takes a similarity matrix S of size $n \times n$ and a number K of clusters as arguments and returns a $K \times n$ matrix Data containing n K-dimensional vectors in columns. Use Algorithm 2 with the graph Laplacian L^C from Equation 7.

3.3.2 Clustering research areas

A survey has been carried out in an Electrical Engineering department. A list of research areas has been provided and professors had to grade each areas on a scale from 0 to 3 according to their interests. The data can be seen by opening the file topis.csv with a text editor. The goal is to cluster the research areas, assuming that the professors' interests indicate how strongly related two different research areas are.

Pre-processing From the raw data contained in topis.csv, we first need to define the similarity measure between the research areas to compute a similarity matix S. As it is not the main point of this work, we will simply explain how we transform the data. The data can be seen as a matrix of size $m \times n$, where m is the number of professors and n the number of research areas. Each coefficient is an integer from 0 to 3. We first normalize this matrix such that each row average is zero (average grades given by one professor becomes zero). Then we define the n column vectors $c_1, \ldots, c_n$ of this matrix. One vector

corresponds to all the normalized grades of one research area. Finally we use a Gaussian kernel to measure the similarity between areas:

$$S_{i,j} = \exp\left(-\frac{\|c_i - c_j\|^2}{2\sigma^2}\right), \tag{10}$$

with $\sigma = 5$.

Question 15. Open spectral_clu.py. Check the code for the pre-processing step (from line 21 to 38). Is there any mistake compared to what we have described? If yes, correct it.

Question 16. In a terminal run: ./spectral_clu.py. For the clustering, we have chosen $K = 3$. Observe the results (displayed in the terminal). Do you know what is plotted in the figure? Explain the meaning of this plot.

Question 17. Launch this simulation several times (./spectral_clu.py). Do you always observe the same results? Why?

In performing multiple clustering runs, you will have noticed that frequently very small clusters appear.

Question 18. Explain why small clusters are not desirable in this task. Then to overcome this issue, modify sim_to_vect such that it performs spectral clustering based on the graph Laplacian L^{NC} related to normalized-cut cost function.

Question 19. Again perform several simulation runs of ./spectral_clu.py. Do you still observe very small clusters? Compare the graph Laplacians and explain the difference observed in the clustering results?

Congratulations, you have completed this lab exercise. You can now proceed with optional question 12 or test the behavior of the simulations by changing parameter values.

3.4 Beyond the Presented Spectral Clustering Algorithms

Another cost function In the literature, a lot of cost functions on graph can be found. However the associated minimization problem may be quite complex. Another example of cost function that can be solved using spectral clustering is pairwise data clustering, see [5] and also [6] for further information. The main advantage of pairwise data clustering is that small and big clusters are not penalized by the cost function, which can be useful if we are confronted to groups of very different sizes.

Choosing K The number of cluster K has to be set in advance. We have seen that if K is not chosen correctly, the clustering result is not meaningful. There are many ways of selecting K such as using a penalty term depending on K or using some stability measure.

References

[1] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[2] Edward W Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[3] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[4] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[5] Volker Roth, Julian Laub, Motoaki Kawanabe, and Joachim M Buhmann. Optimal cluster preserving embedding of nonmetric proximity data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(12):1540–1551, 2003.

[6] Joachim Buhmann. Data clustering and learning. *The Handbook of Brain Theory and Neural Networks*, pages 278–281, 1995.

Appendix: Short Tutorial on Python and Numpy

In this appendix we introduce some basics on Python. We assume that you know some basics on Matlab.

3.5 Generalities on Python

Python is a very simple programming language and it is included in many linux distributions. Python can be used in the same manner as Matlab, that is to say as a scripting language or directly in a shell. A main file main.py has to be created with a text editor. In order to run the script, you can open a terminal and just run the command line ./main.py. If there are some errors, the first execution error will be displayed in your terminal.

3.5.1 Structuring your main file

Importing Libraries Libraries contain constants, objects and functions that are useful. Depending on your needs (plotting, array handling, optimizations...), you need to import the additional libraries before using them. The first lines of your main file should contain all the libraries you want to import. For instance, if you need some mathematical functions such as sin or cos or some constant such as π, you need to import the math library by writing on a line: import math. Then press enter to go to the next line. Be aware that if you want to use *pi* in your script, you will need to add the prefix math.*pi* to indicate that the constant π has to be found in the library math. In case the name of the library is too long or complex you can rename it. For example, we will need the library numpy to handle arrays and linear algebra operations. We can write: import numpy as np. Then all functions and constants from numpy should be prefixed with np. instead of numpy.
Of course you can write libraries of your own: just create a file mylibrary.py and in your main file, write the following line import mylibrary. Proceeding this way, you can use all constants, objects and functions created in mylibrary.py directly in your main file.

Basic Structures and Examples Python is a language where the punctuation is strict. Specifically, the indentation, the colons and the return to line is meaningful. To end a line no need to put semicolons as in Matlab, just return to a new line.

- Open a terminal and type the following commands

  ```
  >ipython
  ```

- A Python shell has been opened and we can try out some simple examples. To define variables (an integer or a float):

  ```
  a=1
  b=1.
  ```

- To print the value of variable

  ```
  print a,b
  ```

- Multiple assignments at the same time

  ```
  a,b,c = 3. , 2, 9
  ```

- Basic operations

```
a+b
a**b
```

- The For Loop starts by writing this line.

  ```
  for k in range(10):
  ```

  The colons indicate that a block is starting so when you return to a new line an indentation is needed (the Python shell and some text editors do it automatically). Now enter this new lines:

  ```
     print 'Inside Loop'
     print 'k=',k
  ```

  Then press Enter, remove the indentation and press Enter again. The loop is executed. Note that k will take values between 0 and 9 and not 1 and 10 ! The block ends when a line has no indentation.

- The While and If statements are similar to the For statement.

  ```
  k=0
  while (k<10):
     k+=1
     if (k==5):
        print 'k=',k
     else:
        print 'k is not equal to 5'
     print 'End Iteration ',k
  ```

To define a function A simple example:

```
def hello(i)
   if (i==0):
      print 'Hello Madam'
   elif (i==1):
      print 'Hello Sir'
   else:
      print 'Hello'
   return(i+1)
```

To test it, just type hello(0).

3.6 Generalities on Numpy

Numpy is used for handling arrays and linear algebra. Compared to Matlab, the main difference is that indexes start at 0 ! In your Python shell, type

```
import numpy as np
```

Familiarize yourself with some easy commands:

```
n=5
m=3
v=np.zeros(n)  #n-D vector of 0's
A=np.zeros((m,n)) #Matrix
B=np.array([[1,5,6],[1,9,12]])
C=np.zeros((m,n))
v[0]=5
v[2:4]=4
A[1,:]=np.arange(n)
v[2:3]  #Be careful, index 3 is not included !
B[:,0:2]
B[0,1::]  #From column 1 to the end
condi=(v==4)  #Boolean vector condi[k]=1 if and only if v[k]=4.
A[0,condi]  #Boolean extraction
```

Some useful functions:

```
np.dot(B,A)  #Matrix multiplication
C*A  #Element-wise multiplication
np.arange(0,10)  #Integer values from 0 to 9
np.sum(B)  #Sum of the elements of M.
np.sum(B,axis=0)  #Sum of each column
np.amin(B)  #Minimum of M (other possibilities: argmin, amax, argmax)
np.amin(B,axis=0)  #Minimum values of each column
```

To know more about the Numpy library, go to: `http://docs.scipy.org/doc/numpy/reference/`. The Quick Search section `http://docs.scipy.org/doc/numpy/search.html` is quite useful.