

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/309153198>

NumPy / SciPy / NetworkX Recipes for Data Science: Spectral Clustering

Technical Report · October 2016

DOI: 10.13140/RG.2.2.18771.17446

CITATION

1

READS

6,753

1 author:



Christian Bauckhage

University of Bonn

456 PUBLICATIONS 7,729 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



lectures on game AI [View project](#)



reading group machine learning / [AI View project](#)

NumPy / SciPy / NetworkX Recipes for Data Science: Spectral Clustering

Christian Bauckhage
B-IT, University of Bonn, Germany
Fraunhofer IAIS, Sankt Augustin, Germany

Abstract—We revisit the idea of relational clustering and look at NumPy code for spectral clustering that allows us to cluster graphs or networks. In addition, our topic in this note provides us with the opportunity to study the use of NetworkX functions.

I. INTRODUCTION

In an earlier note, we studied k -medoids clustering and saw that the algorithm relies on a matrix of distances, i.e. relations, between given data points [1]. In this note, we revisit the idea of relational clustering but generalize our point of view. This is to say that we will discuss a method that applies to almost arbitrary relations and not only to distances.

Graphs are the natural data structure to represent relational data because we may think of any relation between data points to be represented in terms of a (possibly weighted) adjacency matrix of a graph. This, however, is to say that the problem of relational clustering can be understood as the problem of finding clusters in graph.

In this note, we therefore look at a very general and powerful method for graph clustering called *spectral clustering*. We first briefly summarize the underlying theory¹ and then discuss practical implementations.

As always, we assume that our readers are passably familiar with NumPy / SciPy [2] and Matplotlib [3]. Moreover, as a first-timer in this series of notes, our discussion will also involve functions from the NetworkX library [4]. For our code snippets to work, we therefore need the following imports

```
import numpy as np
import networkx as nx
import numpy.linalg as la
import scipy.cluster.vq as vq
import matplotlib.pyplot as plt
```

II. THEORY

Figure 1 shows an undirected graph $G(V, E)$ consisting of a set of vertices

$$V = \{v_1, v_2, \dots, v_n\} \quad (1)$$

and a set of edges

$$E \subseteq V \times V. \quad (2)$$

¹Readers who are interested in much deeper insights into the why and how of spectral clustering might want to watch the following video lectures <https://www.youtube.com/watch?v=rgxOXRWcSoE> <https://www.youtube.com/watch?v=xNZgjBVR62A>.

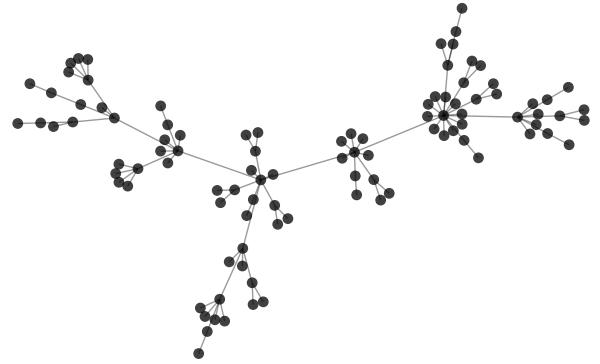


Fig. 1: A power-law graph G of $n = 100$ vertices.

Regardless of what their vertices may represent, graphs like this are often referred to as relational data structures because an edge $(v_i, v_j) \in E$ relates an item $v_i \in V$ to an item $v_j \in V$.

Looking at the figure, it appears that there are groups of vertices that cluster together. For small graphs like this, it is easy to detect such clusters through visual inspection. But what if the graph was much larger or would have a structure that cannot be plotted easily? Are there any clustering algorithms we could implement to automatically cluster graphs?

Yes, there are! In fact there are several approaches, however, the arguably most powerful one is called *spectral clustering*.

Given any graph G , spectral clustering considers spectral properties of the graph *Laplacian matrix*

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \quad (3)$$

where \mathbf{A} denotes the graph *adjacency matrix* whose entries are given by

$$A_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

and \mathbf{D} is the *degree matrix* whose entries amount to

$$D_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Looking at the definition in (5), we recall that the *degree* $\deg(v_i)$ of a vertex v_i is nothing else but the number of edges

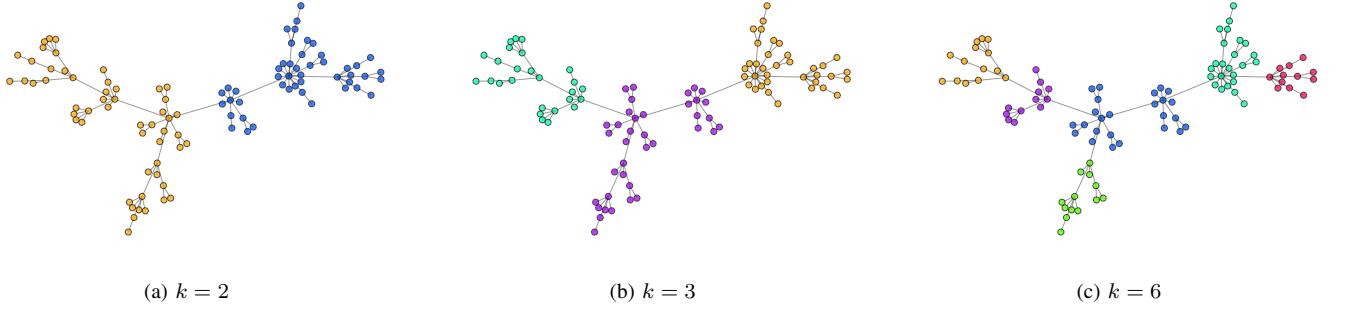


Fig. 2: Graph clusters obtained from spectral clustering. The result in (a) was obtained considering the signs of the entries of the Fiedler vector \mathbf{f} . The results in (b) and (c) were obtained from running k -means on the rows of matrix \mathbf{U}_k in (10).

incident to it. We therefore have that the diagonal elements of \mathbf{D} can be expressed as

$$D_{ii} = \deg(v_i) = \sum_{j=1}^n A_{ij} \quad (6)$$

which is to say that to be able to compute \mathbf{L} we only need to know \mathbf{A} . In other words, if \mathbf{A} is given, we can compute \mathbf{L} .

To say that spectral clustering is concerned with spectral properties of \mathbf{L} is to say that it requires us to compute the spectral decomposition

$$\mathbf{L} = \mathbf{U} \Lambda \mathbf{U}^T \quad (7)$$

where Λ is a diagonal matrix of eigenvalues λ_i of \mathbf{L} and \mathbf{U} is an orthonormal matrix whose columns \mathbf{u}_i correspond to the eigenvectors of \mathbf{L} .

Before we discuss how the decomposition in (7) may inform clustering, we note that, if G has n vertices, then \mathbf{A} , \mathbf{D} , and \mathbf{L} will all be $n \times n$ matrices.

Moreover, if G is undirected (as we are currently assuming), its adjacency matrix \mathbf{A} will be symmetric and, since \mathbf{D} is but a diagonal matrix, the Laplacian \mathbf{L} will be symmetric, too.

We therefore know that all eigenvalues and eigenvectors of \mathbf{L} will be real valued (recall our discussion in [5]). However, there is even more we can say about the eigenvalue spectrum

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{n-1} \geq \lambda_n. \quad (8)$$

If we assume that all the entries of the (possibly weighted) adjacency matrix \mathbf{A} are non-negative, then \mathbf{L} will be **positive semi-definite**. For its eigenvectors \mathbf{u}_i we therefore have

$$\mathbf{u}_i^T \mathbf{L} \mathbf{u}_i = \lambda_i \mathbf{u}_i^T \mathbf{u}_i \geq 0 \quad (9)$$

which is to say that the eigenvalues λ_i must be non-negative.

Moreover, it is easy to verify that the vector of all ones will be an eigenvector of \mathbf{L} and that the corresponding eigenvalue will be zero. Since all the eigenvalues will be greater or equal to zero, we therefore know that $\lambda_n = 0$ and $\mathbf{u}_n = \mathbf{1}$.

The eigenvector \mathbf{u}_{n-1} belonging to the second smallest eigenvalue λ_{n-1} is more interesting. It is called the Fiedler vector \mathbf{f} and can be used to partition G . This is because its entries f_j are either greater or less than zero so that we can

consider their sign in order to group the vertices v_j of G into two clusters (see Fig. 2(a)).

If we would want to cluster G into $k > 2$ clusters, we might think of two strategies. First of all, we could compute $\mathbf{f}(G)$ to partition G into sub-graphs G_1 and G_2 and then proceed recursively, i.e. compute $\mathbf{f}(G_1)$ and $\mathbf{f}(G_2)$ in order to obtain G_{11} , G_{12} , G_{21} and G_{22} and so on and so forth.

Or, we could consider information in the next couple of smallest eigenvectors. Here, the typical strategy is to apply k -means clustering to the n row vectors of the $n \times k$ matrix

$$\mathbf{U}_k = [\mathbf{u}_{n-1}, \dots, \mathbf{u}_{n-k}] \quad (10)$$

to obtain a cluster label for each of the n vertices v_1, v_2, \dots, v_n of the given graph G .

III. PRACTICE

In order to have a simple example for our practical discussion, let us create a power-law graph such as in Fig. 1. Using *NetworkX*, this can be accomplished by means of

```
| G = nx.powerlaw_cluster_graph(100, 1, 0.0)
```

which produces a graph according to the following preferential attachment process: iteratively create $n = 100$ vertices; each time a vertex is created, randomly link it to 1 of the existing vertices and close a triangle of edges with probability 0.0.

In order to visualize our graph G in a planar plot, we need to compute 2D coordinates for its vertices. To this end, we may use

```
| coord = nx.spring_layout(G, iterations=1000)
```

This then allows us to plot the graph as follows

```
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')
ax.axis('off')
nx.draw_networkx_edges(G, coord)
nx.draw_networkx_nodes(G, coord,
                      node_size=45,
                      node_color='k')
plt.show()
```

which should produce an output similar to the one in Fig. 1.

Because spectral clustering considers spectral properties of the Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$ of G , we next need to determine

matrices A , D , and L . Since we created G as a *NetworkX* data structure, we immediately obtain its adjacency matrix as a *NumPy* array

```
| A = nx.adjacency_matrix(G)
```

Once A is available, computing D and L is a triviality

```
| D = np.diag(np.ravel(np.sum(A, axis=1)))
L = D - A
```

Given L , we can now compute its spectral decomposition. Since L is symmetric, we recall from an earlier note [5] that this should be done as follows

```
| l, U = la.eigh(C)
```

At this point, it actually comes in handy that *NumPy*'s *la.eigh* function returns eigenvalues and eigenvectors in ascending order, because spectral clustering is all about eigenvectors belonging to small eigenvalues.

For instance, the Fiedler vector f of L can easily be read off of U as

```
| f = U[:,1]
```

To use it for graph clustering, we let

```
| labels = np.ravel(np.sign(f))
```

and modify the function for node plotting in our above plotting routine as follows

```
| fig = plt.figure()
nx.draw_networkx_nodes(G, coord,
                       node_size=45,
                       node_color=labels)
```

This should then produce a result similar to what we can see in Fig. 2(a).

In order to cluster G into $k > 2$ sub-graphs, we next apply k -means clustering to the rows of the matrix U_k of the k smallest eigenvectors (where we begin counting from the second smallest one). That is, we simply execute

```
| k = 3
means, labels = vq.kmeans2(U[:,1:k], k)
```

and then use our previous plotting routine to produce a result similar to the one in Fig. 2(b).

IV. NOTES AND REFERENCES

The literature on spectral clustering and its applications is vast. A popular and openly accessible tutorial that discusses technical details, algorithmic variants, and best practices is the text by von Luxburg [6].

In the computer science literature, the idea of computing spectral decompositions of the graph Laplacian L is usually

derived from arguments as to graph cuts (cf. e.g. [7]). While there is nothing wrong with this, these derivations cannot provide an intuitive interpretation of what the entries eigenvectors of L really signify. There is, however, also a physical view on spectral clustering that has to do with dynamic processes on networks. This view reveals the eigenvectors to indicate which vertices behave similar under certain dynamics. This is best seen, once it is understood that the graph Laplacian L is the discrete analog of the continuous Laplace operator Δ in multivariate calculus. For details, we once again refer to the following video lectures

<https://www.youtube.com/watch?v=rgxOXRWcSoE>

<https://www.youtube.com/watch?v=xNZgjBVR62A>

The simple techniques we discussed in this note also apply to directed graphs with possibly asymmetric adjacency matrices. However, for such matrices there are better methods out there. Examples would be DEDICOM or DESICOM [8]–[10].

Finally, the Fiedler vector is named in honor of Miroslav Fiedler, who was one of the pioneers of algebraic graph theory [11]. Sometimes, the Laplacian matrix is also referred to as the Kirchhoff matrix named after the physicist Gustav Kirchhoff who was the first to rigorously study the flow of currents in electrical circuits.

REFERENCES

- [1] C. Bauckhage, “NumPy / SciPy Recipes for Data Science: k-Medoids Clustering,” researchgate.net, Feb. 2015, <https://dx.doi.org/10.13140/2.1.4453.2009>.
- [2] T. Oliphant, “Python for Scientific Computing,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.
- [3] J. Hunter, “Matplotlib: A 2D Graphics Environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [4] A. Hagberg, D. Schult, and P. Swart, “Exploring Network Structure, Dynamics, and Function Using NetworkX,” in *Proc. Python in Science Conf. (SciPy)*, 2008.
- [5] C. Bauckhage, “NumPy / SciPy Recipes for Data Science: Eigenvalues / Eigenvectors of Covariance Matrices,” researchgate.net, Sep. 2015, <https://dx.doi.org/10.13140/RG.2.1.2307.5046>.
- [6] U. von Luxburg, “A Tutorial on Spectral Clustering,” *arXiv*, 2007.
- [7] J. Shi and J. Malik, “Normalized Cuts and Image Segmentation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.
- [8] R. Harshman, “Models for Analysis of Asymmetrical Relationships among N Objects or Stimuli,” in *Proc. Joint Meeting of the Psychometric Society and the Society for Mathematical Psychology*, 1978.
- [9] H. Kiers, “DESICOM: Decomposition of Asymmetric Relationships Data into Simple Components,” *Behaviormetrika*, vol. 24, no. 2, pp. 203–217, 1997.
- [10] C. Bauckhage, R. Sifa, A. Drachen, C. Thurau, and F. Hadji, “Beyond Heatmaps: Spatio-Temporal Clustering using Behavior-Based Partitioning of Game Levels,” in *Proc. Conf. on Computational Intelligence and Games*. IEEE, 2014.
- [11] M. Fiedler, “A Property of Eigenvectors of Nonnegative Symmetric Matrices and its Application to Graph Theory,” *Czechoslovak Mathematical J.*, vol. 25, no. 4, pp. 619–633, 1975.