Semi-supervised clustering in graphs

David Chatel

Ecole doctorale ED072 “Sciences pour l’Ingénieur”

Semi-supervised clustering in graphs

Thèse préparée par
David Chatel

Pour l’obtention du grade de
Docteur de l’Université de Lille
Domaine: **Informatique**

soutenue le 27 Novembre 2017

Centre de Recherche en Informatique, Signal et Automatique de Lille
(CRIStAL - UMR CNRS 9189)

Abstract

Nowadays, decision processes in various areas (marketing, biology, etc) require the processing of increasing amounts of more and more complex data. Because of this, there is a growing interest in machine learning techniques. Among these techniques, there is clustering. Clustering is the task of finding a partition of items, such that items in the same cluster are more similar than items in different clusters. This is a data-driven technique. Data come from different sources and take different forms. One challenge consists in designing a system capable of taking benefit of the different sources of data, even when they come in different forms. Among the different forms a piece of data can take, the description of an object can take the form of a feature vector: a list of attributes that takes a value. Objects can also be described by a graph which captures the relationships objects have with each others. In addition to this, some constraints can be known about the data. It can be known that an object is of a certain type or that two objects share the same type or are of different types. It can also be known that on a global scale, the different types of objects appear with a known frequency. In this thesis, we focus on clustering with three different types of constraints: label constraints, pairwise constraints and power-law constraint. A label constraint specifies in which cluster an object belong. Pairwise constraints specify that pairs of object should or should not share the same cluster. Finally, the power-law constraint is a cluster-level constraint that specifies that the distribution of cluster sizes are subject to a power-law. We want to show that introducing semi-supervision to clustering algorithms can alter and improve the solutions returned by unsupervised clustering algorithms. We contribute to this question by proposing algorithms for each type of constraints. Our experiments on UCI data sets and natural language processing data sets show the good performance of our algorithms and give hints towards promising future works.

Contents

Notation

$\mathbb{N}$	Natural number
$\mathbb{R}$	Real number
$\mathrm{diag}(\boldsymbol{A})$	Diagonal matrix with entries $\{\boldsymbol{A}_{ii}\}_{i=1}^{k}$
$\boldsymbol{A}^{+}$	Moore-Penrose pseudo-inverse of $\boldsymbol{A}$
$\boldsymbol{D}$	Degree matrix
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{L}$	Combinatorial Laplacian matrix
$\boldsymbol{L_{rw}}$	Random walk Laplacian matrix
$\boldsymbol{L_{sym}}$	Normalized symmetric Laplacian matrix
$\boldsymbol{W}$	Similarity matrix
$\partial\mathcal{A}$	Boundary of $\mathcal{A}$
$\mathrm{ctd}(v_i, v_j)$	Commute-time distance between nodes v_i and v_j
$\mathrm{vol}(\mathcal{A})$	Volume of $\mathcal{A}$
$\mathcal{E}$	Set of edges
$\mathcal{V}$	Set of nodes
$\mathbf{0}$	zero vector or matrix
$\mathbf{1}$	Constant vector
$\boldsymbol{d}$	Degree vector
$\boldsymbol{e}_i$	ith standard basis vector

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Nowadays, more and more actors need to process increasing amounts of data about various items (customers of marketing companies, health information of individuals, plain natural language textual data are only few examples). Big data, as it is called, is growing exponentially. Being able to process these data is important as data take part in the decision and production process. For example, recommendation systems, like Netflix' recommendation system, take an important place into a decision process used to increase revenues. Another example is biological data, where the amount of data have already reached petabyte and even exabyte [43], and these data are used, for example, to get a diagnosis or take better decisions during treatment.

In order to efficiently process this data, cleaning, classification, summaries are required, so that prediction and analysis can be done accurately. These tasks are done using approaches like data reduction, classification, clustering. Data reduction is a task which consists in finding a representation of data that is devoided of noise or features unrelated to the task the data will be used for. Classification and clustering consist in assigning chunks of data into meaningful groups respectively called classes and clusters.

Unlike classification, clustering does not imply you know the groups or the number of groups. Clustering consists in partitioning data into groups called clusters, such that data in the same clusters are similar, while data in different clusters are dissimilar. This is a data-driven machine learning approach. Clustering can be used to discover more general information within data. For example, clustering is used to find topics in collections of documents [26]. Clustering is often used as a step of a larger task. For example, clustering can be used to clean and generalize models obtained using reinforcement learning [53].

The data come from different sources and take different forms, for example, in bioinformatics and cross-market customer relationship management there are data from multiple sources jointly describing the same sets of objects. Moreover, there are cases where knowledge from the domain is also available and should be used in clustering.

Among the forms data can take, there are feature vectors, and graphs. Feature vectors describe objects using a list of attributes. Graphs describe objects using the relationship between each objects. In both cases, obtaining the similarity between objects is not trivial. A classic example is the clustering of the characters in the T.V. show "The

Simpsons". How are two characters similar or dissimilar ? Are we interested in males vs. females or clustering by activity (student, employee at the nuclear plant, etc) ? Similarity, as well as clustering, is subjective. There is no such thing as a universal representation of a dataset. The representation of a dataset ultimately depends on the task it will be used in. Therefore, finding an appropriate representation of the data is an important step of any learning algorithm. This subjectivity gives us freedom in how we can describe the reality. The relationship between objects used to describe them using a graph is represented by a non-negative score, the similarity. Similarity between two objects is a human-readable way to describe them. Are those items similar or dissimilar ? This kind of representation is particularly suitable when we do not know what makes two objects similar or not, but we know how much they are similar or not.

It is not always possible to obtain a "good" representation of the data. Any representation can be noisy, incomplete or incorrect. It is also possible that the studied phenomenon can only be captured indirectly by studying another phenomenon that is only partially correlated. With these data sets, clustering assumptions (smoothness, cluster and manifold assumptions) are not always satisfied. That is, nodes which are close to each other are more likely to be in the same cluster, clusters should be dense and well-separated, and the data lie approximatively on a manifold whose dimension is much lower than the input space. When clustering assumptions are not satisfied, we can give a hint on the properties of the desired partition: it is possible to introduce some partial supervision. This supervision can be used to learn a new representation of the data which will then be used by standard clustering algorithms or can be used to inspire new algorithms that will directly return a solution satisfying all or most of the supervision. Partial supervision can take different forms. In this thesis, we are interested in the following types of partial supervision:

Label constraints consist in specifying for a subset of the items, the cluster they belong to. This kind of supervision is well suited when a lot of information is available about the data. For instance, when the clusters are known in advance and unlabeled data have to be assigned to one of these clusters.

Pairwise constraints specify for a subset of pairs of items if they are must-linked (they should belong to the same cluster) or cannot-linked (they should belong to different clusters). Pairwise constraints are meant to be used whenever the clusters are unknown, but information about fellowship of some individuals is known.

Power-law distributed cluster size constraints specify that the cluster sizes should follow a power-law distribution. With that kind of supervision, neither the clusters nor their number is known.

Notice that these constraints are weaker and weaker. You can always trivially obtain pairwise constraints from label constraints: nodes assigned to the same label create a must-link constraint while nodes assigned to different labels create a cannot-link constraint. On the other hand, obtaining label constraints from pairwise constraints is trivial only in certain cases. Label and pairwise constraints give hints about specific nodes, whereas power-law distributed cluster size constraint is a cluster-level constraint. That is, the formers locally constrain the nodes of the graph and these constraints are then propagated using the structure of the graph, while the latter constrains the structure of the partition the algorithm is looking for without giving any local hints.

In this thesis, we want to show that introducing semi-supervision to clustering algorithms can alter and improve the solutions returned by unsupervised clustering algorithms. We mainly work on weighted undirected graphs. These graphs can be given or are eventually built from feature vector data sets. Despite being a practical representation of the data, these graphs are not always very suited for the task they will be used in. The method that was used to build the graph is always prone to errors. Graphs that are given may be the result of an unstable process. For example social graphs are always changing and this instability can result in representations with intrinsic contradictions that will have to be solved in order to obtain a good clustering. For example a group of friends can break up for several reasons, and if the graph represents a snapshot of the situation during that break up, some individuals still may be linked to other people who broke up. In order to solve those contradictions, decisions will have to be made. These decisions can be modeled by introducing constraints. If it is known that two individuals share most of their value or live together, the likelihood that they break up in favor of a third individual with whom they share very little is very low. In this thesis, we contribute to this topic by introducing label, pairwise or power-law constraints to clustering algorithms.

Our first contribution is an extension of Mavroeidis [54] to handle clustering with label constraints for more than two clusters. The original method [54] only works for two clusters. The author had a proposition for more than two clusters, but we empirically show that our contribution yields better results. We also extend this to handle pairwise constraints. A second contribution consists in a novel algorithm to handle clustering with pairwise constraints. It can take a graph or a feature-vector based dataset and learns a new feature-vector based representation of the data. Then this new representation can be used in other applications, including feature-vector based clustering algorithms, like k-means. Li and Liu [44] is the clustering algorithm with pairwise constraints that is most closely related to that second contribution. Some other clustering algorithms with pairwise constraints take different approaches, such as adapting the similarity matrix directly [39, 48] or taking benefit of some interesting properties of the clustering problems, for example its submodularity [74] or the interpretation of the objective function of the standard normalized cut problem [18]. A third contribution concerns clustering when the sizes of clusters are known to follow a power-law distribution. While this kind of constraint is particularly hard to satisfy, as shown by our critique of Zhou *et al.* [90], we show some promising results in natural language processing.

We conduct experiments using UCI data sets, but we are also inspired by a natural language processing task: the coreference resolution [73]. In this thesis, we have the mentions of the text. Following steps consist in building an undirected weighted graph from these mentions, using the knowledge contained in the text, then applying a clustering algorithm suited for the particularities of this task.

Structure of this document

In Chapter 2, we will discuss unsupervised clustering. We will introduce useful notions such as clustering evaluation measures in Section 2.5 and graphs in Section 2.1. In Section 2.3, we will explain how graph cuts problems can be used to formulate the clustering problem in graphs. And in Section 2.4, we will introduce the spectral embedding, a projection of the graph in a vector space where distances between nodes correspond to the average time to travel back and forth from one node to another.

In Chapter 3, we will discuss clustering with label constraints. We will first see in Section 3.1 how propagating labels through the graph can be used for this task. We will then discuss, in Section 3.2, a method that introduces these constraints by modifying directly the similarity matrix of the graph. Then we will present a first contribution, Normalized One-against-all Supervised Spectral Clustering (or NOA-SSC), in Section 3.3, which extends the previous method.

Pairwise constraints will be the topic of Chapter 4. We will explain the early Spectral Learning method in Section 4.1, then in the same section we will propose an extension to NOA-SSC which attempts to handle pairwise constraints by turning them into label constraints. In Section 4.2, we will review two methods that introduce pairwise constraints by tuning the similarity matrix, which is pretty similar to the method presented in Section 3.2. Then we will see in Section 4.3 how directly transforming the spectral embedding can satisfy the pairwise constraints. First we will review a method that uses semi-definite programming to do so, then we will present a third contribution which is more scalable.

Finally, in Chapter 5, we will study a cluster-level constraint that we call the "power-law constraint". It specifies that cluster sizes should follow a power-law distribution. We will quickly review in Section 5.1 what a power-law is and the inherent difficulties we encounter while working with such constraints. Then we will review in Section 5.2 a method based on k-means that attempts to satisfy this constraint by introducing penalties to the distances that depend on the structure of the partition at the current iteration of the algorithm. We will see that this method has its limits. Finally we will present in Section 5.3 a fourth contribution, which attempts to satify the power-law constraint by modeling clustering as an instance of a stochastic process (called Pitman-Yor) that produces power-laws.

Chapter 2

Preliminaries: Unsupervised Graph Clustering

Contents

This section is dedicated to explaining unsupervised spectral clustering, that is: clustering the nodes of a graph according to their similarities. Representing a set of objects with a graph is very convenient. Instead of trying to find a good representation for all the objects at once, it is possible to determine for each pair of objects whether they are similar or dissimilar. Then, it is possible to get a feature-based k-dimensional representation of the data from the graph using eigenvalue/eigenvector analysis. This k-dimensional representation of a graph is called a spectral embedding.

In this section, we will briefly explain what are graphs in Section 2.1 and how to derive a kernel from them in Section 2.2. Then, in Section 2.3, we will explain how to obtain a spectral embedding and what to do with it. We will also explain, in Section 2.4, how it links to the commute-time distance, which is related to the time needed to travel back and forth between two nodes. Finally, in Section 2.5, we will explain evaluation of the results and the inherent difficulties with evaluating clustering.

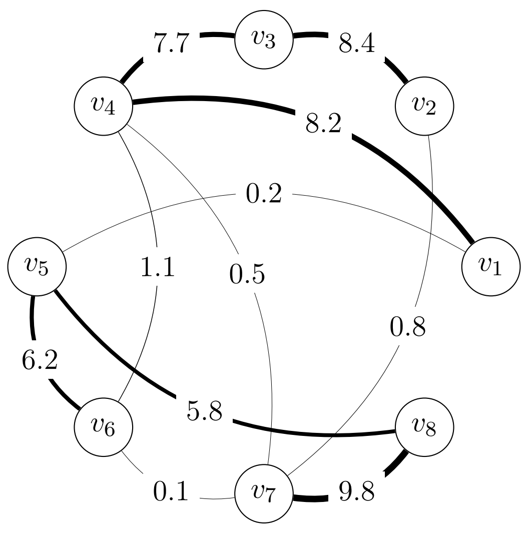


Figure 2.1: An undirected weighted graph consists of a set of nodes and a set of edges connecting pairs of nodes and labeled by some positive real number. In this example, the set of nodes is $\mathcal{V} = \{v_1, \ldots, v_8\}$ and the set of edges is represented by labeled arcs between pairs of nodes. In this representation, the thickness of the arcs is proportional to the weight of the edges.

2.1 Similarity graphs

In graph theory, a graph is a representation of a certain relationship between a set of objects. An object is represented by a mathematical abstraction called “vertex” or “node”. A node is the fundamental unit of which graphs are formed. The relationships between objects are represented by pairs of nodes, called “edges”. There are different types of graphs, mainly undirected graphs and directed graphs. The edges of an undirected graph consist of a set of unordered pairs of nodes, whereas the edges of a directed graph consist of a set of ordered pairs of nodes. Graphs can also be weighted; edges of weighted graphs are labeled by some numerical, usually non-negative, real value.

In this thesis, we study similarity graphs (see figure 2.1 for an example) whose undirected and weighted edges model a symmetric, homophilic and local relationship between objects associated to nodes. This relationship is usually called “similarity”.

2.1.1 Graph notation

Formally, an undirected weighted graph $G = (\mathcal{V}, \mathcal{E}, w : \mathcal{E} \to \mathbb{R})$ is an ordered tuple composed of a set of nodes $\mathcal{V}$ usually denoted $v_1, \ldots, v_n$, a set of unordered pairs of nodes, called edges, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and a function w mapping edges of the graph to, usually non-negative, real numbers. The function w is usually represented by a symmetric matrix, called the similarity matrix, denoted $\boldsymbol{W}$. The weight on the edge connecting nodes v_i and v_j is $\boldsymbol{W}_{ij} = w(v_i, v_j)$. As an example, the similarity matrix of the graph depicted in figure 2.1 is a 8×8 matrix with non-negative entries:

$$\boldsymbol{W} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 0.0 & 0.0 & 0.0 & 8.2 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 8.4 & 0.0 & 0.0 & 0.0 & 0.8 & 0.0 \\ 0.0 & 8.4 & 0.0 & 7.7 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.2 & 0.0 & 7.7 & 0.0 & 0.0 & 1.1 & 0.5 & 0.0 \\ 0.2 & 0.0 & 0.0 & 0.0 & 0.0 & 6.2 & 0.0 & 5.8 \\ 0.0 & 0.0 & 0.0 & 1.1 & 6.2 & 0.0 & 0.1 & 0.0 \\ 0.0 & 0.8 & 0.0 & 0.5 & 0.0 & 0.1 & 0.0 & 9.8 \\ 0.0 & 0.0 & 0.0 & 0.0 & 5.8 & 0.0 & 9.8 & 0.0 \end{pmatrix} \end{array} \tag{2.1.1}$$

Zero entries of the similarity matrix $\boldsymbol{W}$ correspond to pairs of nodes not connected by an edge. For example, there is no edge connecting v_2 and v_4 in this example.

The degree of a node v_i is defined as $\boldsymbol{d}_i \triangleq \sum_{j=1}^{n} \boldsymbol{W}_{ij}$ and the degree matrix of a graph is defined as the following diagonal matrix

$$\boldsymbol{D} \triangleq \sum_{i=1}^{n} \boldsymbol{d}_i \boldsymbol{e}_i \boldsymbol{e}_i^\top, \tag{2.1.2}$$

where $\boldsymbol{e}_i$ is the ith standard basis vector. The volume of a subset $\mathcal{S} \subseteq \mathcal{E}$ of the edges of G is

$$\mathrm{vol}(\mathcal{S}) = \sum_{(v_i,v_j)\in\mathcal{S}} w(v_i, v_j), \tag{2.1.3}$$

and the volume of the graph G is

$$\mathrm{vol}(G) = \mathrm{vol}(\mathcal{E}) = \sum_{i=1}^{n} \boldsymbol{d}_i. \tag{2.1.4}$$

2.1.2 Obtaining a graph

In applications where graphs do not arise naturally, unlike social networking applications for example, the data set may be composed of a set of data points $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, a set of pairwise similarities $w(v_i, v_j)$ or a set a pairwise distances $d(v_i, v_j)$. The construction used to build the graph from these data sets depends on the clustering task. In this section, I will present few popular methods used to build graphs from a set of data points.

ϵ-neighborhood graphs (ϵ-graph)

In an ϵ-graph, an undirected edge (v_i, v_j) is created whenever the distance $d(v_i, v_j)$ is at most ϵ. This kind of graph is usually unweighted. However, a weight on the edges of such a graph can provide additional information. It is also possible to obtain a sparse graph from a weighted graph using this method. Sparse graphs are a way of storing a graph in memory: only edges whose weight is greated than zero are stored in memory. In addition to being more memory efficient, it also enable the use of certain algorithms that takes benefit of this structure.

k-nearest neighbor graphs (k-nn graph)

In a k-nn graph, each node v_i is connected to each node in the set of the k-nearest neighbors of v_i according to $d(v_i, v_j)$. Notice that if v_j is in the k-nearest neighborhood of v_i, the converse is not necessarily true. Hence, the k-nearest neighborhood relationship is not symmetric and this definition leads to directed graphs. There are two ways of making such graph undirected. Firstly, the directions of the edges can be ignored: an undirected edge (v_i, v_j) is created whenever v_i is in the k-nearest neighborhood of v_j or v_j is in the k-nearest neighborhood of v_i. The resulting graph is usually called *the k-nearest neighborhood graph.* Secondly, an undirected edge (v_i, v_j) is created whenever both v_i and v_j are in each others k-nearest neighborhood. In this case, the graph is usually called *the mutual k-nearest neighborhood graph.* In both cases, the edges are weighted by the similarity of their endpoints.

Fully connected graphs

In fully connected graphs, all the nodes are connected and the edges are weighted by the similarity of their endpoints. In the following, I present some popular similarity functions, that model local neighborhoods.

- **Radial basis function kernel (or RBF kernel)** is a popular similarity function used in various learning algorithms taking advantage of the kernel method. The RBF kernel on two vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is defined as

$$\operatorname{rbf}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right), \tag{2.1.5}$$

  where $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ is the Euclidean distance between the two vectors and σ is a free parameter. The value of the RBF kernel decreases exponentially with the distance and ranges between 0 (in the limit) and 1 (when $\boldsymbol{x}_i = \boldsymbol{x}_j$). It can model local neighborhoods and can be interpreted as a similarity measure. The feature space of this kernel has an infinite number of dimensions, making it useful for handling data sets with non-linear cluster bounds. To obtain a graph, the weight on the edges are equal to $\boldsymbol{W}_{ij} = \operatorname{rbf}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. As the RBF kernel already models local neighborhoods, no further sparsification is necessary.

- **Cosine similarity** between vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is the cosine of the angle between these vectors. It can be obtained by using the Euclidean dot product

$$\cos(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\boldsymbol{x}_i \cdot \boldsymbol{x}_j}{\|\boldsymbol{x}_i\| \, \|\boldsymbol{x}_j\|}. \tag{2.1.6}$$

In general, the result ranges from -1 and 1. This similarity is typically used in some natural language processing tasks, such as document clustering; each vector $\boldsymbol{x}_i$ is comprised of m components which represent the occurence frequency of some words in the ith document. As word frequencies are non-negative, the result ranges from 0 and 1 in that case. However, the feature space of this kernel does not have more dimensions than the space described by vectors $\boldsymbol{x}_{i=1,\dots,n}$. Moreover, this kernel does not model local neighborhoods. An ϵ-graph or a k-nn graph is typically done over this similarity to obtain a proper graph.

- **Logistic regression** can also be used to produce graphs when each pair of nodes (v_i, v_j) is characterized by a feature vector $\boldsymbol{\omega}_{ij}$. A logit model is applied on this vector to obtain the similarity between nodes:

$$\text{logit}(\boldsymbol{\omega}_{ij}) = \frac{1}{1 + \exp\left(-(\beta + \boldsymbol{\theta}^\top \boldsymbol{\omega}_{ij})\right)}, \tag{2.1.7}$$

where β and $\boldsymbol{\theta}$ are the parameters of the model. The logit model smoothly varies from 0 (for dissimilar nodes) to 1 (for similar nodes).

2.2 The Laplacian matrix of the graph

Laplacian matrices of graphs are the main tool in spectral clustering and are extensively studied in spectral graph theory, see Chung [15]. In this section, we define graph Laplacian matrices, and present some interesting properties of these matrices.

Let $G = (\mathcal{V}, \mathcal{E}, w : \mathcal{E} \mapsto \mathbb{R})$ be a similarity graph (see section 2.1). The combinatorial Laplacian, or unnormalized Laplacian, of G is

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}. \tag{2.2.1}$$

The combinatorial Laplacian is also equal to $\boldsymbol{L} = \boldsymbol{G}^\top \boldsymbol{G}$, where $\boldsymbol{G}$ is the incidence matrix of the graph G. The incidence matrix is a $|\mathcal{E}| \times |\mathcal{V}|$ matrix defined such that for each edge (v_i, v_j), there is a row in $\boldsymbol{G}$ equal to $\sqrt{w(v_i, v_j)}\boldsymbol{e}_i^\top - \sqrt{w(v_i, v_j)}\boldsymbol{e}_j^\top$. Thus, the incidence matrix of G is

$$\boldsymbol{G} = \sum_{k=1}^{|\mathcal{E}|} \sqrt{w(i,j)}\boldsymbol{e}_k \boldsymbol{e}_i^\top - \sqrt{w(i,j)}\boldsymbol{e}_k \boldsymbol{e}_j^\top, \tag{2.2.2}$$

where $w(i, j)$ is the weight associated with some edge indexed k. For example, the

Laplacian matrix of the graph depicted in figure 2.1 is

$$\boldsymbol{L} = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 8.4 & 0.0 & 0.0 & -8.2 & -0.2 & 0.0 & 0.0 & 0.0 \\ 0.0 & 9.2 & -8.4 & 0.0 & 0.0 & 0.0 & -0.8 & 0.0 \\ 0.0 & -8.4 & 16.1 & -7.7 & 0.0 & 0.0 & 0.0 & 0.0 \\ -8.2 & 0.0 & -7.7 & 17.5 & 0.0 & -1.1 & -0.5 & 0.0 \\ -0.2 & 0.0 & 0.0 & 0.0 & 12.2 & -6.2 & 0.0 & -5.8 \\ 0.0 & 0.0 & 0.0 & -1.1 & -6.2 & 7.4 & -0.1 & 0.0 \\ 0.0 & -0.8 & 0.0 & -0.5 & 0.0 & -0.1 & 11.2 & -9.8 \\ 0.0 & 0.0 & 0.0 & 0.0 & -5.8 & 0.0 & -9.8 & 15.6 \end{pmatrix} \end{array} \tag{2.2.3}$$

The following proposition summarizes some interesting properties of the combinatorial Laplacian. For a more complete discussion on the properties of Laplacian matrices, see [58, 59]. The proofs comes from Luxburg [52].

Proposition 1. *The combinatorial Laplacian matrix satisfies the following properties:*

1. *$\boldsymbol{L}$ is symmetric ($\boldsymbol{L} = \boldsymbol{L}^\top$).*
2. *$\forall \boldsymbol{v} \in \mathbb{R}^n : \boldsymbol{v}^\top \boldsymbol{L}\boldsymbol{v} = \frac{1}{2}\sum_{i,j=1}^n \boldsymbol{W}_{ij}\,(\boldsymbol{v}_i - \boldsymbol{v}_j)^2$.*
3. *$\boldsymbol{L}$ is positive semi-definite ($\boldsymbol{L} \succeq 0$).*
4. *$\boldsymbol{L}$ is doubly-centered ($\boldsymbol{L}\mathbf{1} = \mathbf{0}$ and $\mathbf{1}^\top\boldsymbol{L} = \mathbf{0}^\top$).*
5. *The multiplicity of the eigenvalue 0 of $\boldsymbol{L}$ equals the number of connected components in the graph and the null space of $\boldsymbol{L}$ is spanned by the indicator vectors $\mathbf{1}_{\mathcal{A}_1}, \ldots, \mathbf{1}_{\mathcal{A}_k}$, where the components of $\mathbf{1}_{\mathcal{A}_i}$ equal 1 for the nodes inside cluster $\mathcal{A}_i$ and 0 for the others.*

Proof. Given that $\boldsymbol{W}$ is symmetric with positive entries:

1. The matrices $\boldsymbol{D}$ and $\boldsymbol{W}$ are symmetric by definition, hence $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$ is also symmetric.
2. First observe that $\boldsymbol{v}^\top\boldsymbol{L}\boldsymbol{v} = \boldsymbol{v}^\top\boldsymbol{D}\boldsymbol{v} - \boldsymbol{v}^\top\boldsymbol{W}\boldsymbol{v} = \sum_{i=1}^n \boldsymbol{d}_i\boldsymbol{v}_i^2 - \sum_{i,j=1}^n \boldsymbol{W}_{ij}\boldsymbol{v}_i\boldsymbol{v}_j$, then the rightmost part of the equation is half the sum of weighted perfect squares:

$$\boldsymbol{v}^\top\boldsymbol{L}\boldsymbol{v} = \sum_{i=1}^n \boldsymbol{d}_i\boldsymbol{v}_i^2 - \sum_{i,j=1}^n \boldsymbol{W}_{ij}\boldsymbol{v}_i\boldsymbol{v}_j = \sum_{i,j=1}^n \boldsymbol{W}_{ij}(\boldsymbol{v}_i^2 - \boldsymbol{v}_i\boldsymbol{v}_j) \tag{2.2.4}$$

$$= \frac{1}{2}\left(\sum_{i,j=1}^n \boldsymbol{W}_{ij}(\boldsymbol{v}_i^2 + \boldsymbol{v}_j^2 - 2\boldsymbol{v}_i\boldsymbol{v}_j)\right) = \frac{1}{2}\sum_{i,j=1}^n \boldsymbol{W}_{ij}(\boldsymbol{v}_i - \boldsymbol{v}_j)^2. \tag{2.2.5}$$

3. This is a direct consequence of property 2 and $\boldsymbol{W}_{ij} \geq 0$.

4. As $\boldsymbol{D1} = \boldsymbol{d}$ and $\boldsymbol{W1} = \boldsymbol{d}$, we have: $\boldsymbol{L1} = \boldsymbol{D1} - \boldsymbol{W1} = \boldsymbol{d} - \boldsymbol{d} = \boldsymbol{0}$. Moreover, $\boldsymbol{L}$ is symmetric, thus, we also have: $\boldsymbol{0}^\top = (\boldsymbol{L1})^\top = \boldsymbol{1}^\top \boldsymbol{L}^\top = \boldsymbol{1}^\top \boldsymbol{L}$.

5. if $\boldsymbol{v}$ is an eigenvector of $\boldsymbol{L}$ with eigenvalue 0, then $\boldsymbol{v}^\top \boldsymbol{L v} = \sum_{i,j=1}^n \boldsymbol{W}_{ij}(\boldsymbol{v}_i - \boldsymbol{v}_j)^2 = 0$, however as the weights $\boldsymbol{W}_{ij}$ are non-negative, this sum only vanishes when all terms $\boldsymbol{W}_{ij}(\boldsymbol{v}_i - \boldsymbol{v}_j)^2$ vanish. If two nodes v_i and v_j are connected, that is $\boldsymbol{W}_{ij} > 0$, then $\boldsymbol{W}_{ij}(\boldsymbol{v}_i - \boldsymbol{v}_j)^2 = 0$ only when $\boldsymbol{v}_i = \boldsymbol{v}_j$. Thus, by transitivity, $\boldsymbol{v}$ has to be constant for every node connected by a path in the graph. Also, if there exists a non-zero vector $\boldsymbol{v} \in \ker(\boldsymbol{L})$ such that there are two sets of nodes $\mathcal{A}, \mathcal{B}$ that satisfy $\boldsymbol{v}_{\mathcal{A}} \neq \boldsymbol{v}_{\mathcal{B}}$ then nodes $v_{\mathcal{A}}$ and $v_{\mathcal{B}}$ are not connected by any path in the graph. If there is exactly k non-zero vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \ker(\boldsymbol{L})$ and for all i, j we have $\boldsymbol{v}_i \perp \boldsymbol{v}_j$, then the multiplicity of eigenvalue 0 of $\boldsymbol{L}$ is k and $\boldsymbol{v}_i^\top \boldsymbol{v}_j = \sum_{k=1}^n v_i(k) v_j(k) = 0$. Therefore there is at least k connected components in the graph. Moreover, if there were one more connected component in the graph, we could find another non zero vector $\boldsymbol{v}_{k+1} \in \ker(\boldsymbol{L})$ that is not a linear combination of the first k vectors.

□

Also called normalized Laplacian matrices, the random walk Laplacian matrix, denoted $\boldsymbol{L_{rw}}$, and the symmetric normalized Laplacian matrix, denoted $\boldsymbol{L_{sym}}$, are both defined from the Laplacian matrix:

$$\boldsymbol{L_{rw}} = \boldsymbol{D}^{-1}\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{W} \tag{2.2.6}$$

$$\boldsymbol{L_{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}. \tag{2.2.7}$$

Proposition 2. *The normalized Laplacian matrices satisfy the following properties:*

1. $\forall \boldsymbol{v} \in \mathbb{R}^n : \boldsymbol{v}^\top \boldsymbol{L_{sym}} \boldsymbol{v} = \frac{1}{2}\boldsymbol{W}_{ij}\left(\frac{\boldsymbol{v}_i}{\sqrt{d_i}} - \frac{\boldsymbol{v}_j}{\sqrt{d_j}}\right)^2$.

2. $\boldsymbol{L_{rw}}$ *and* $\boldsymbol{L_{sym}}$ *are similar matrices.*

3. *The eigenpairs of* $\boldsymbol{L_{rw}}$ *are solutions of the generalized eigenvalue problem* $\boldsymbol{Lu} = \lambda \boldsymbol{Du}$.

4. $\boldsymbol{1} \in \ker(\boldsymbol{L_{rw}})$.

5. $\boldsymbol{L_{rw}}$ *and* $\boldsymbol{L_{sym}}$ *are positive semi-definite.*

Proof. 1. $\boldsymbol{v}^\top \boldsymbol{L_{sym}} \boldsymbol{v} = (\boldsymbol{D}^{-1/2}\boldsymbol{v})^\top \boldsymbol{L}(\boldsymbol{D}^{-1/2}\boldsymbol{v}) = \frac{1}{2}\boldsymbol{W}_{ij}\left(\frac{\boldsymbol{v}_i}{\sqrt{d_i}} - \frac{\boldsymbol{v}_j}{\sqrt{d_j}}\right)^2$.

2. $\boldsymbol{L_{rw}} = \boldsymbol{D}^{-1}\boldsymbol{L}$ and $\boldsymbol{L_{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$, thus $\boldsymbol{L_{rw}} = \boldsymbol{D}^{-1/2}\boldsymbol{L_{sym}}\boldsymbol{D}^{1/2}$.

3. As the matrix $\boldsymbol{D}$ is inversible and $\boldsymbol{L_{rw}} = \boldsymbol{D}^{-1}\boldsymbol{L}$, if $\boldsymbol{L_{rw}}\boldsymbol{u} = \lambda \boldsymbol{u}$ then $\boldsymbol{Lu} = \lambda \boldsymbol{Du}$.

4. Obvious as $\boldsymbol{L_{rw}}\boldsymbol{1} = \boldsymbol{0}$.

5. $\boldsymbol{L_{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} = \boldsymbol{D}^{-1/2}\boldsymbol{G}^{\top}\boldsymbol{G}\boldsymbol{D}^{-1/2}$, where $\boldsymbol{G}$ is the incidence matrix. Moreover, as $\boldsymbol{L_{rw}}$ and $\boldsymbol{L_{sym}}$ are similar matrices, they share the same eigenvalue. Thus $\boldsymbol{L_{sym}} \succeq 0$ and $\boldsymbol{L_{rw}} \succeq 0$.

□

2.3 Graph cuts

Clustering in graphs can be seen as finding a partition of nodes, such that edges whose ends are inside the same cluster (or intra-cluster edges) have a high weight and edges whose ends are in different clusters (or inter-cluster edges) have a low weight.

But what is exactly a "high weight" comparatively to a "low weight" ? Consider for example figure 2.2, the clusters we would like to recover are colored red and blue, however we can see that there exists at least two blue data points b_1, b_2 and one red data point r such that the distance $\|b_1 - r\|$ is smaller than $\|b_1 - b_2\|$.

Using any of the graph construction methods, among the simplest ones[1] we have seen in section 2.1.2, will produce a graph such that $w(b_1, b_2) < w(b_1, r)$. This shows some intra-cluster edge weight can have a lower value than some inter-cluster edge. Thus, we need to find a global measure to represent the similarity between two sets of nodes. Then, clustering in graphs would consist in finding a partition of the nodes that minimizes this measure.

Among the simplest measures, "cut" only ensures that inter-cluster edges have a low weight. For two clusters, it is a relatively simple problem that can be computed efficiently [79], by minimizing the following objective function:

$$\text{cut}\left(\{\mathcal{A}_i\}_{i=1}^{k}\right) = \sum_{i=1}^{k} vol(\partial\mathcal{A}_i), \tag{2.3.1}$$

where $\partial\mathcal{A}_i = \{(v_i, v_j) : v_i \in \mathcal{A}_i, v_j \notin \mathcal{A}_i\}$ is the edge boundary of $\mathcal{A}_i$. However, in practice, the solutions given by mincut often simply separate one node from the rest of the graph, and this may not be satisfactory.

Among other measures, RatioCut [30] and Ncut [77] are two popular objective functions that tackle this problem by balancing the size or the volume of the clusters. These objective functions are defined as follows:

$$\text{RatioCut}(\{\mathcal{A}_i\}_{i=1}^{k}) = \sum_{i=1}^{k} \frac{\text{vol}(\partial\mathcal{A}_i)}{|\mathcal{A}_i|} = \sum_{i=1}^{k} \frac{\text{cut}(\mathcal{A}_i, \overline{\mathcal{A}_i})}{|\mathcal{A}_i|} \tag{2.3.2}$$

$$\text{Ncut}(\{\mathcal{A}_i\}_{i=1}^{k}) = \sum_{i=1}^{k} \frac{\text{vol}(\partial\mathcal{A}_i)}{\text{vol}(\mathcal{A}_i)} = \sum_{i=1}^{k} \frac{\text{cut}(\mathcal{A}_i, \overline{\mathcal{A}_i})}{\text{vol}(\mathcal{A}_i)} \tag{2.3.3}$$

These functions take a smaller value as the clusters A_i become larger. In particular, given two different partitions, whose volume of the boundaries are equal, the minimum

[1]graph constructions methods that do not require to train a model, unlike logistic regression

of these functions is achieved when either all $|\mathcal{A}_i|$ or $\mathrm{vol}(\mathcal{A}_i)$, depending on the objective, are equal. The introduction of any of these normalization factors makes these problems NP- hard [82]. However, relaxed versions of these problems can be solved in polynomial time. In particular, "spectral clustering" may refer to two different algorithms: spectral bisection and k-way spectral clustering. In the following sections, I will explain how to approximate the more popular Ncut for 2 and more than 2 clusters. Similar proofs for RatioCut can be found in Luxburg [52, section 5.1 and 5.2].

2.3.1 Approximation of Ncut for 2 clusters

The goal is to optimize the following NP-hard problem

$$\min_{\mathcal{A}\subset\mathcal{V}} \mathrm{Ncut}(\{\mathcal{A},\overline{\mathcal{A}}\}). \tag{2.3.4}$$

Proposition 3. *Equation (2.3.4) is equivalent to*

$$\min_{\boldsymbol{f}} \boldsymbol{f}^\top \boldsymbol{L}\boldsymbol{f} : \boldsymbol{D}\boldsymbol{f} \perp \mathbf{1}, \boldsymbol{f}^\top \boldsymbol{D}\boldsymbol{f} = \mathrm{vol}(G), \tag{2.3.5}$$

where $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$ and for some vector $\boldsymbol{f}$ such that

$$\boldsymbol{f}_i = \begin{cases} \sqrt{\mathrm{vol}(\overline{\mathcal{A}})/\mathrm{vol}(\mathcal{A})} & \text{if } v_i \in \mathcal{A} \\ -\sqrt{\mathrm{vol}(\mathcal{A})/\mathrm{vol}(\overline{\mathcal{A}})} & \text{if } v_i \in \overline{\mathcal{A}}. \end{cases} \tag{2.3.6}$$

Proof. We begin by rewriting equation (2.3.4) in matrix form.

$$\mathrm{Ncut}(\{\mathcal{A},\overline{\mathcal{A}}\}) = \mathrm{vol}(\partial\mathcal{A})\left(\frac{1}{\mathrm{vol}(\mathcal{A})} + \frac{1}{\mathrm{vol}(\overline{\mathcal{A}})}\right) \tag{2.3.7}$$

$$\mathrm{vol}(G)\mathrm{Ncut}(\{\mathcal{A},\overline{\mathcal{A}})\}) = \mathrm{vol}(\partial\mathcal{A})\left(\frac{\mathrm{vol}(\mathcal{A})+\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\mathcal{A})} + \frac{\mathrm{vol}(\mathcal{A})+\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\overline{\mathcal{A}})}\right) \tag{2.3.8}$$

$$= \mathrm{vol}(\partial\mathcal{A})\left(\frac{\mathrm{vol}(\mathcal{A})}{\mathrm{vol}(\overline{\mathcal{A}})} + \frac{\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\mathcal{A})} + 2\right) \tag{2.3.9}$$

$$= \frac{1}{2}\sum_{v_i\in\mathcal{A}, v_j\in\overline{\mathcal{A}}} w(v_i,v_j)\left(\sqrt{\frac{\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\mathcal{A})}} + \sqrt{\frac{\mathrm{vol}(\mathcal{A})}{\mathrm{vol}(\overline{\mathcal{A}})}}\right)^2 \tag{2.3.10}$$

$$+ \frac{1}{2}\sum_{v_i\in\overline{\mathcal{A}}, v_j\in\mathcal{A}} w(v_i,v_j)\left(-\sqrt{\frac{\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\mathcal{A})}} - \sqrt{\frac{\mathrm{vol}(\mathcal{A})}{\mathrm{vol}(\overline{\mathcal{A}})}}\right)^2 \tag{2.3.11}$$

$$\mathrm{vol}(G)\mathrm{Ncut}(\{\mathcal{A},\overline{\mathcal{A}})\}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{W}_{ij}\left(\boldsymbol{f}_i - \boldsymbol{f}_j\right)^2, \tag{2.3.12}$$

Then, proposition 1 shows that $\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{W}_{ij}\left(\boldsymbol{f}_i - \boldsymbol{f}_j\right)^2$ can be written in matrix form:

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{W}_{ij}\left(\boldsymbol{f}_i-\boldsymbol{f}_j\right)^2=\boldsymbol{f}^\top\boldsymbol{L}\boldsymbol{f} \tag{2.3.13}$$

The vector $\boldsymbol{D}\boldsymbol{f}$ is orthogonal to the constant vector $\boldsymbol{1}$:

$$(\boldsymbol{D}\boldsymbol{f})^\top\boldsymbol{1}=\sum_{i=1}^{n}\boldsymbol{d}_i\boldsymbol{f}_i=\sum_{v_i\in\mathcal{A}}\boldsymbol{d}_i\sqrt{\frac{\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\mathcal{A})}}-\sum_{v_i\in\overline{\mathcal{A}}}d_i\sqrt{\frac{\mathrm{vol}(\mathcal{A})}{\mathrm{vol}(\overline{\mathcal{A}})}} \tag{2.3.14}$$

$$=\mathrm{vol}(\mathcal{A})\sqrt{\frac{\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\mathcal{A})}}-\mathrm{vol}(\overline{\mathcal{A}})\sqrt{\frac{\mathrm{vol}(\mathcal{A})}{\mathrm{vol}(\overline{\mathcal{A}})}}=0. \tag{2.3.15}$$

We also have that $\boldsymbol{f}^\top\boldsymbol{D}\boldsymbol{f}=\mathrm{vol}(G)$:

$$\boldsymbol{f}^\top\boldsymbol{D}\boldsymbol{f}=\sum_{i=1}^{n}\boldsymbol{d}_i\boldsymbol{f}_i^2=\mathrm{vol}(\mathcal{A})\frac{\mathrm{vol}(\overline{\mathcal{A}})}{\mathrm{vol}(\mathcal{A})}+\mathrm{vol}(\overline{\mathcal{A}})\frac{\mathrm{vol}(\mathcal{A})}{\mathrm{vol}(\overline{\mathcal{A}})}=\mathrm{vol}(G). \tag{2.3.16}$$

□

Rewriting equation (2.3.4) in matrix form does not change the fact it is NP-hard. However, we can now relax the conditions on

$$\min_{\boldsymbol{f}}\boldsymbol{f}^\top\boldsymbol{L}\boldsymbol{f}:\boldsymbol{D}\boldsymbol{f}\perp\boldsymbol{1},\boldsymbol{f}^\top\boldsymbol{D}\boldsymbol{f}=\mathrm{vol}(G),\boldsymbol{f}\text{ as in equation (2.3.6)}, \tag{2.3.17}$$

by letting the components of $\boldsymbol{f}$ take arbitrary values in $\mathbb{R}$. This leads to the following relaxed problem:

$$\min_{\boldsymbol{f}}\boldsymbol{f}^\top\boldsymbol{L}\boldsymbol{f}:\boldsymbol{D}\boldsymbol{f}\perp\boldsymbol{1},\boldsymbol{f}^\top\boldsymbol{D}\boldsymbol{f}=\mathrm{vol}(G),\boldsymbol{f}\in\mathbb{R}^n. \tag{2.3.18}$$

Equivalently, this problem consists in finding the critical points of $\mathcal{L}(\boldsymbol{f})=\boldsymbol{f}^\top\boldsymbol{L}\boldsymbol{f}-\lambda\left(\boldsymbol{f}^\top\boldsymbol{D}\boldsymbol{f}-\mathrm{vol}(G)\right)$ subject to $\boldsymbol{D}\boldsymbol{f}\perp\boldsymbol{1}$, where λ is a Lagrange multiplier. The critical points of $\mathcal{L}(\boldsymbol{f})$ occur at

$$\frac{d\mathcal{L}(\boldsymbol{f})}{d\boldsymbol{f}}=\boldsymbol{L}\boldsymbol{f}-\lambda\boldsymbol{D}\boldsymbol{f}=0. \tag{2.3.19}$$

Therefore, the eigenvectors $\boldsymbol{u}_1,\dots,\boldsymbol{u}_n$ associated with eigenvalues $\lambda_1\leq\cdots\leq\lambda_n$ of the generalized eigenvalue problem $\boldsymbol{L}\boldsymbol{v}_i=\lambda_i\boldsymbol{D}\boldsymbol{v}_i$ are the critical points of $\mathcal{L}(\boldsymbol{f})$. Moreover, as $\boldsymbol{D}$ is invertible, $\boldsymbol{u}_1,\dots,\boldsymbol{u}_n$ are the eigenvectors of the random walk Laplacian matrix of the graph $\boldsymbol{D}^{-1}\boldsymbol{L}$, denoted by $\boldsymbol{L}_{\boldsymbol{rw}}$. The first eigenvector $\boldsymbol{u}_1=\boldsymbol{1}$ is not a solution to equation (2.3.18), because the vector $\boldsymbol{D}\boldsymbol{1}$ is not orthogonal to $\boldsymbol{1}$. Hence, the solution is given by the second eigenvector $\boldsymbol{u}_2$. Another way to find this solution consists in substituting $\boldsymbol{g}\triangleq\boldsymbol{D}^{1/2}\boldsymbol{f}$ in equation (2.3.18), then the problem becomes:

$$\min_{\boldsymbol{g}} \boldsymbol{g}^\top \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}\boldsymbol{g} : \boldsymbol{g} \perp \boldsymbol{D}^{1/2}\mathbf{1}, \|\boldsymbol{g}\| = \mathrm{vol}(G), \boldsymbol{g} \in \mathbb{R}^n. \tag{2.3.20}$$

The matrix $\boldsymbol{L_{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$ is the normalized symmetric Laplacian matrix of the graph. This problem is in the form of the standard Rayleigh-Ritz theorem and its solution is given by the second eigenvector of $\boldsymbol{L_{sym}}$. The matrices $\boldsymbol{L_{sym}}$ and $\boldsymbol{L_{rw}}$ are similar, so they represent the same linear operator under different bases. The matrix $\boldsymbol{D}^{-1/2}$ is the change of basis matrix from $\boldsymbol{L_{rw}}$ to $\boldsymbol{L_{sym}}$. Hence, if $\boldsymbol{u}_2'$ is the second eigenvector for $\boldsymbol{L_{sym}}$, $\boldsymbol{u}_2 = \boldsymbol{D}^{-1/2}\boldsymbol{u}_2'$ is the second eigenvector of $\boldsymbol{L_{rw}}$.

The second eigenvector of $\boldsymbol{L_{rw}}$ and $\boldsymbol{L_{sym}}$, here denoted by $\boldsymbol{f}$, are approximate minimizers of equation (2.3.5). In order to obtain a partition, we need to transform the approximate minimizers $\boldsymbol{u}_2$ into a discrete indicator vector, as in equation (2.3.6).

The easiest way to recover a partition from $\boldsymbol{u}_2$ is to use the sign of its coordinates as a cluster indicator: let $\boldsymbol{u}_2(i)$ be the ith component of $\boldsymbol{u}_2$ and v_i the corresponding node of the graph, then $v_i \in \mathcal{A}$ if $\boldsymbol{u}_2(i) \geq 0$, otherwise $v_i \in \overline{\mathcal{A}}$.

Another more complex way consists in solving the following problem:

$$\arg\min_{c} \mathrm{Ncut}(\mathcal{A}, \overline{\mathcal{A}}) : \mathcal{A} = \{v_i : \boldsymbol{u}_2(i) \geq c\}, \tag{2.3.21}$$

where $c \in \mathbb{R}$. This problem can be solved in linear time by ordering the nodes of the graph such that $\boldsymbol{u}_2(1) \leq \cdots \leq \boldsymbol{u}_2(n)$ and then by choosing an index i for which $\mathcal{A} = \{v_j : j \geq i\}$ minimizes equation (2.3.21). This algorithm is called a sweep cut or "Cheeger sweep" in the literature.

This kind of cut yields a bound that relates the isoperimetric number h_G of a graph and the value of the second eigenvalue λ_2 by

$$\frac{\lambda_2}{2} \leq h_G \leq \sqrt{2\lambda_2}. \tag{2.3.22}$$

Notice there is also a higher-order Cheeger inequality, proved in Lee *et al.* [42]:

$$\frac{\lambda_k}{2} \leq \rho(k) \leq \mathcal{O}(k^2)\sqrt{\lambda_k}. \tag{2.3.23}$$

All of this can be used to cut the graph in two parts, but this is already enough to perform hierarchical cuts of the graph, as shown in the recursive algorithm 1.

2.3.2 Approximation of Ncut for more than 2 clusters

The goal is now to optimize the following NP-hard problem

$$\min_{\{\mathcal{A}_i\}_{i=1}^k} \mathrm{Ncut}\left(\{\mathcal{A}_i\}_{i=1}^k\right) \tag{2.3.24}$$

Similarly to section 2.3.1, we rewrite this problem into matrix form, then a relaxation is applied.

Algorithm 1: Hierarchical bisection (H. Bisection)

```
Input: Similarity matrix $\boldsymbol{W} \in \mathcal{M}_{n\times n}(\mathbb{R}_{\geq 0})$, stopping criterion $\alpha$
1 begin
2     $\boldsymbol{z} \leftarrow 0^n$
3     Let $\mathcal{A}, \mathcal{B}$ be the result of a cheeger sweep as in Equation 2.3.21
4     if Ncut$(\mathcal{A}, \mathcal{B}) < \alpha$ then
5         Let $\boldsymbol{W}_{\mathcal{A}}$ $(\boldsymbol{W}_{\mathcal{B}})$ be the submatrix of $\boldsymbol{W}$ corresponding to nodes in $\mathcal{A}$ $(\mathcal{B})$
6         $\boldsymbol{z}(\mathcal{A}) \leftarrow 2 \cdot$ H. Bisection$(\boldsymbol{W}_{\mathcal{A}}, \alpha)$
7         $\boldsymbol{z}(\mathcal{B}) \leftarrow 1 + 2 \cdot$ H. Bisection$(\boldsymbol{W}_{\mathcal{B}}, \alpha)$
8     return $\boldsymbol{z}$
Output: Node labels $\boldsymbol{z} \in \mathbb{N}^n$
```

Algorithm 2: Normalized cut according to [77].

```
Input: Similarity matrix $\boldsymbol{W}$, number of clusters $k$.
1 begin
2     Compute the first $k$ eigenvectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_k$ of the generalized eigenvalue
        problem $\boldsymbol{Lu} = \lambda \boldsymbol{Du}$.
3     Let $\boldsymbol{U} \in \mathbb{R}^{n\times k}$ be the matrix containing $\boldsymbol{u}_1, \dots, \boldsymbol{u}_k$ as columns.
4     Let $x_i \in \mathbb{R}^k$ be the vector corresponding to the $i$th row of $\boldsymbol{U}$.
5     Cluster the $n$ points $\{x_i\}_{i=1}^k$ with $k$-means into $k$ clusters $\{\mathcal{A}_i\}_{i=1}^k$.
Output: Partition $\{\mathcal{A}_i\}_{i=1}^k$.
```

Proposition 4. *Let $\boldsymbol{H}_{ic}$ be the cluster indicator matrix for the partition $\{\mathcal{A}_i\}_{i=1}^k$, such that*

$$\boldsymbol{H}_{ic} = \begin{cases} \frac{1}{\sqrt{\mathrm{vol}(\mathcal{A}_c)}} & \textit{if } v_i \in \mathcal{A}_c, \\ 0 & \textit{otherwise.} \end{cases} \tag{2.3.25}$$

Equation (2.3.24) *is equivalent to*

$$\min_{\{\mathcal{A}_i\}_{i=1}^k} \operatorname{trace}\left(\boldsymbol{H}^\top \boldsymbol{L} \boldsymbol{H}\right) : \boldsymbol{H}^\top \boldsymbol{D} \boldsymbol{H} = \boldsymbol{I}, \boldsymbol{H} \textit{ as in equation } (2.3.25). \tag{2.3.26}$$

By substituting $\boldsymbol{T} = \boldsymbol{D}^{1/2}\boldsymbol{H}$ and relaxing the discreteness condition, we obtain

$$\min_{\boldsymbol{T}\in\mathbb{R}^{n\times k}} \operatorname{trace}\left(\boldsymbol{T}^\top \boldsymbol{L_{sym}} \boldsymbol{T}\right) : \boldsymbol{T}^\top \boldsymbol{T} = \boldsymbol{I}. \tag{2.3.27}$$

This standard trace minimization problem is solved by the matrix $\boldsymbol{T}$ which contains the first k eigenvectors of $\boldsymbol{L_{sym}}$ in columns. Equivalently, the solution is given by the first k eigenvectors of the generalized eigenvalue problem $\boldsymbol{Lu} = \lambda \boldsymbol{Du}$. Once, the first k eigenvectors are obtained, nodes are partitioned using k-means. [77, 65] proposed two different algorithms to approximate normalized cuts (see algorithms 2 and 3).

Algorithm 3: Normalized cut according to [65].

```
Input: Similarity matrix $\boldsymbol{W}$, number of clusters $k$.
1 begin
2     Compute the first $k$ eigenvectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_k$ of $\boldsymbol{L_{sym}}$.
3     Let $\boldsymbol{U} \in \mathbb{R}^{n \times k}$ be the matrix containing $\boldsymbol{u}_1, \dots, \boldsymbol{u}_k$ as columns.
4     Let $\hat{\boldsymbol{U}}$ be the matrix obtained by normalizing the rows of $\boldsymbol{U}$, that is
      $\hat{\boldsymbol{U}}_{ij} = \boldsymbol{U}_{ij} / \sqrt{\sum_k \boldsymbol{U}_{ik}^2}$.
5     Let $x_i \in \mathbb{R}^k$ be the vector corresponding to the $i$th row of $\hat{\boldsymbol{U}}$.
6     Cluster the $n$ points $\{x_i\}_{i=1}^k$ with $k$-MEANS into $k$ clusters $\{\mathcal{A}_i\}_{i=1}^k$.
Output: Partition $\{\mathcal{A}_i\}_{i=1}^k$.
```

Notice that algorithm 3 has an extra normalization step on line 4. Reasons for this are discussed in section 2.2.

Recently, research on higher order Cheeger's inequalities [42, 67] have yielded further bounds for spectral clustering with more than 2 clusters.

Our implementation of k-MEANS, provided in algorithm 4, is based on [4, 66] and has some nice guarantees over standard implementations of k-MEANS. First, at each iteration t, we include the current cluster centroid $\boldsymbol{\mu}_c^{(t)}$ into its update $\boldsymbol{\mu}_c^{(t+1)}$. This guarantees that no cluster becomes empty, without having any other impact on the final result. Second, the algorithm is initialized at random in such a way that the minimal distance between each initial cluster centroid is maximized. That is, the first cluster centroid is selected at random among the data points. Subsequent cluster centroids are selected at random with probability proportional to the maximal distance to the closest previously selected cluster centroids. This ensures that the algorithm is $\Theta(\log k)$-competitive with the optimal clustering.

2.4 The spectral embedding

A justification for using k-MEANS on the first k eigenvectors of the random walk Laplacian comes from random walks on the similarity graph. A random walk is a stochastic process that jumps from node to node with probability proportional to the weight of the edge between these nodes. In this section, we will see that spectral clustering can be seen as finding a partition such that a random walker stays in the same cluster with high probability and migrates to other clusters with lower proability. We will also see that random walks on a graph induce an embedding of the graph that maps its nodes v_i on points $\boldsymbol{v}_i \in \mathbb{R}^n$ such that the Euclidean distance between the points corresponds to the expected time to go back and forth from the corresponding nodes using a random walk. Thus, justifying the use of center-based clustering methods, like k-MEANS.

Algorithm 4: k-MEANS

```
   Input: Data points $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$, number of clusters $k$
1  begin
2      Choose an initial center $\boldsymbol{\mu}_1$ uniformly at random from $\mathcal{X}$
3      repeat
4          Choose the next center $\boldsymbol{\mu}_i$ among $\mathcal{X}$ with probability
               $$\frac{\min_{j<i} \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2}{\sum_{\boldsymbol{x}\in\mathcal{X}} \min_{j<i} \|\boldsymbol{\mu}_i - \boldsymbol{x}\|^2}$$
5      until $k$ centers have been chosen
6      repeat
7          For all $\boldsymbol{x} \in \mathcal{X}$, assign $\boldsymbol{x}$ into the cluster that minimizes $\|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2$
8          foreach centroid $\boldsymbol{\mu}_i : i \in 1, \dots, k$ do
9              $$\boldsymbol{\mu}_i \leftarrow \frac{\sum_{\boldsymbol{x}\in\mathcal{A}_i} \boldsymbol{x}}{|\mathcal{A}_i| + 1} + \frac{\boldsymbol{\mu}_i}{|\mathcal{A}_i| + 1}$$
10     until $\sum_{i=1}^k \sum_{\boldsymbol{x}\in\mathcal{A}} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2$ is stable
   Output: Partition $\{\mathcal{A}_i\}_{i=1}^k$.
```

2.4.1 Normalized cut from random walks

A random walker starting at node v_i has a probability $p(v_i, v_j)$ to jump to v_j proportional to the weight on the edge (v_i, v_j), it is given by $p(v_i, v_j) = w(v_i, v_j)/\boldsymbol{d}_i$. The transition matrix of the random walk is then defined by

$$\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{W}. \tag{2.4.1}$$

It is clear there is a strong connection between the transition matrix $\boldsymbol{P}$ and the random walk Laplacian matrix $\boldsymbol{L_{rw}}$, as $\boldsymbol{L_{rw}} = \boldsymbol{I} - \boldsymbol{P}$. In particular, $\boldsymbol{u}$ is an eigenvector of $\boldsymbol{P}$ with eigenvalue λ if and only if $\boldsymbol{u}$ is an eigenvector of $\boldsymbol{L_{rw}}$ with eigenvalue $1 - \lambda$. If the graph is connected and non-bipartite, then the random walk possesses a unique stationary ditribution $\boldsymbol{\pi}$ given by the largest left eigenvector of $\boldsymbol{P}$, that is $\boldsymbol{P}^\top \boldsymbol{\pi} = \boldsymbol{\pi} = \boldsymbol{d}/\text{vol}(G)$.

Proof. Recall that the multiplicity of the smallest eigenvalue $\lambda_1(\boldsymbol{L_{rw}}) = 0$ is one and its associated eigenvector is $\boldsymbol{1}$. Since $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{L_{rw}}$, $\boldsymbol{1}$ is an eigenvector of $\boldsymbol{P}$ with eigenvalue $\lambda_n(\boldsymbol{P}) = 1$. Then, matrices $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{W}$ and $\boldsymbol{W}\boldsymbol{D}^{-1}$ are similar, thus is $\boldsymbol{u}$ is an eigenvector of $\boldsymbol{D}^{-1}\boldsymbol{W}$ with eigenvalue λ, then $\boldsymbol{D}\boldsymbol{u}$ is an eigenvector of $\boldsymbol{W}\boldsymbol{D}^{-1}$ with the same eigenvalue:

$$\boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{u} = \lambda\boldsymbol{u} = \boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{D}\boldsymbol{u} \iff \boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{D}\boldsymbol{u} = \lambda\boldsymbol{D}\boldsymbol{u}. \tag{2.4.2}$$

Therefore, $\boldsymbol{D}\boldsymbol{1} = \boldsymbol{d}$ is the largest right eigenvector of $\boldsymbol{D}^{-1}\boldsymbol{W}$, associated with eigenvalue 1. Thus, the stationary distribution $\boldsymbol{\pi} = \boldsymbol{d}/\text{vol}(G)$, where $\text{vol}(G) = \sum_{i=1}^{n} \boldsymbol{d}_i$ is a normalization factor. $\square$

Now, we are interested in the probability to jump from one cluster to another. Formally, as we run a random walk $(x_t)_{t\in\mathbb{N}}$, starting from node x_0 in the stationary distribution $\boldsymbol{\pi}$, we are interested in the probability $p(x_1 \in \mathcal{B} | x_0 \in \mathcal{A})$, where $\{\mathcal{A}, \mathcal{B}\}$ is a partition of the graph. First observe that

$$p(x_0 \in \mathcal{A}, x_1 \in \mathcal{B}) = \sum_{v_i \in \mathcal{A}, v_j \in \mathcal{B}} p(x_0 = i, x_1 = j) = \sum_{v_i \in \mathcal{A}, v_j \in \mathcal{B}} \boldsymbol{\pi}_i p(v_i, v_j) \tag{2.4.3}$$

$$= \sum_{v_i \in \mathcal{A}, v_j \in \mathcal{B}} \frac{\boldsymbol{d}_i}{\text{vol}(G)} \frac{\boldsymbol{W}_{ij}}{\boldsymbol{d}_i} = \frac{\sum_{v_i \in \mathcal{A}, v_j \in \mathcal{B}} \boldsymbol{W}_{ij}}{\text{vol}(G)}. \tag{2.4.4}$$

Then, we can compute the probability and see it is equivalent to definition (2.3.3).

$$p(x_1 \in \mathcal{B} | x_0 \in \mathcal{A}) = \frac{p(x_0 \in \mathcal{A}, x_1 \in \mathcal{B})}{p(x_0 \in \mathcal{A})} = \frac{\sum_{v_i \in \mathcal{A}, v_j \in \mathcal{B}} \boldsymbol{W}_{ij}}{\text{vol}(G)} \left(\frac{\text{vol}(A)}{\text{vol}(G)} \right)^{-1} \tag{2.4.5}$$

$$= \frac{\sum_{v_i \in \mathcal{A}, v_j \in \mathcal{B}} \boldsymbol{W}_{ij}}{\text{vol}(A)}. \tag{2.4.6}$$

Thus, minimizing equation (2.3.3) is equivalent to finding a partition such that random walk seldom transitions between different clusters.

2.4.2 The commute-time distance

The commute-time distance [87] between nodes v_i and v_j is the expected number of steps a random walker needs to go back and forth from node v_i to v_j. In this section, we will see that the commute-time distance is an actual Euclidean distance, and thus spectral clustering algorithms are equivalent to center-based clustering algorithms in a kernel space. Since a random walk seldom transitions between different clusters, we would expect that an Euclidean embedding preserving the commute-time distance would also map nodes such that clusters are dense and well separated.

Now, we define the expected first hitting time, that is the expected number of steps a random walker first hits node v_j, starting from v_i. Let $\boldsymbol{H}_{ij}$ be the expected first hitting time from v_i to v_j. The random walker can get from v_i to v_j either directly in one step with probability $\boldsymbol{P}_{ij}$, or it first goes from v_i to v_k, where $k \neq j$, in one step with probability $\boldsymbol{P}_{ik}$ and then from v_k to v_j in the expected first hitting time $\boldsymbol{H}_{kj}$. Formally, $\boldsymbol{H}_{ii} = 0$ for all i and for $i \neq j$ we have:

$$\boldsymbol{H}_{ij} = \boldsymbol{P}_{ij} + \sum_{k \neq j} \boldsymbol{P}_{ik} (1 + \boldsymbol{H}_{kj}) = \sum_{k=1}^{n} \boldsymbol{P}_{ik} + \sum_{k \neq j} \boldsymbol{P}_{ik} \boldsymbol{H}_{kj} = 1 + \sum_{k=1}^{n} \boldsymbol{P}_{ik} \boldsymbol{H}_{kj} - \boldsymbol{P}_{ij} \boldsymbol{H}_{jj}. \tag{2.4.7}$$

If we dump the zero condition on the diagonal of $\boldsymbol{H}$, we get the following matrix:

$$\boldsymbol{M} = \mathbf{1}\mathbf{1}^\top + \boldsymbol{P}(\boldsymbol{M} - \boldsymbol{M}_d) \tag{2.4.8}$$

where $\boldsymbol{M}_d$ is the diagonal matrix with entries $\boldsymbol{M}_{11}, \dots, \boldsymbol{M}_{nn}$. Then $\boldsymbol{H} = \boldsymbol{M} - \boldsymbol{M}_d$. From there, we first want to figure out $\boldsymbol{M}_d$. To that end, recall $\boldsymbol{P}^\top \boldsymbol{d} = \boldsymbol{d}$. Then, we get

$$\begin{aligned} \boldsymbol{d}^\top \boldsymbol{M} &= \boldsymbol{d}^\top \mathbf{1}\mathbf{1}^\top + \boldsymbol{d}^\top \boldsymbol{P}(\boldsymbol{M} - \boldsymbol{M}_d) && (2.4.9)\\ &= \boldsymbol{d}^\top \mathbf{1}\mathbf{1}^\top + \boldsymbol{d}^\top \boldsymbol{M} - \boldsymbol{d}^\top \boldsymbol{M}_d. && (2.4.10) \end{aligned}$$

Thus, $\boldsymbol{M}_d = \boldsymbol{d}^\top \mathbf{1}\mathbf{1}^\top = \left(\sum_{i=1}^n \boldsymbol{d}_i\right) \mathbf{1}^\top$, equivalently $\boldsymbol{d}_i \boldsymbol{M}_{ii} = \sum_{i=1}^n \boldsymbol{d}_i = \mathrm{vol}(G)$. Hence, $\boldsymbol{M}_{ii} = \mathrm{vol}(G)/\boldsymbol{d}_i$. Now, we assume that the graph is connected, then the matrix $\boldsymbol{P}$ is irreducible and the solution $\boldsymbol{M}$ to equation (2.4.8) is unique.

Proof. Assume $\boldsymbol{M}'$ is another solution: $\boldsymbol{M}' = \mathbf{1}\mathbf{1}^\top + \boldsymbol{P}(\boldsymbol{M}' - \boldsymbol{M}_d)$, then $\boldsymbol{M} - \boldsymbol{M}' = \boldsymbol{P}(\boldsymbol{M} - \boldsymbol{M}')$. The columns of $\boldsymbol{M} - \boldsymbol{M}'$ are the right eigenvectors of $\boldsymbol{P}$ with eigenvalue 1. But if $\boldsymbol{P}$ is irreducible, the only eigenvector with eigenvalue 1 is the all-one vector $\mathbf{1}$. Therefore $\boldsymbol{M} - \boldsymbol{M}' = \mathbf{1}\boldsymbol{u}^\top$ for some vector $\boldsymbol{u}$. However, $\mathrm{diag}(\boldsymbol{M}) = \mathrm{diag}(\boldsymbol{M}') = \boldsymbol{M}_d$, hence $\boldsymbol{u} = \mathbf{0}$ and $\boldsymbol{M} = \boldsymbol{M}'$. □

Now, recall that $\boldsymbol{H} = \boldsymbol{M} - \boldsymbol{M}_d$, therefore equation (2.4.8) is equivalent to $(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{H} = \mathbf{1}\mathbf{1}^\top - \boldsymbol{M}_d$. The matrix $\boldsymbol{I} - \boldsymbol{P}$ is not invertible, so we have to be a little more clever to figure out $\boldsymbol{H}$. The combinatorial graph Laplacian is defined as $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} = \boldsymbol{D}(\boldsymbol{I} - \boldsymbol{P})$, hence $\boldsymbol{L}\boldsymbol{H} = \boldsymbol{D}\mathbf{1}\mathbf{1}^\top - \boldsymbol{D}\boldsymbol{M}_d$. Let $\boldsymbol{L}^+$ be the Moore-Penrose pseudo-inverse of $\boldsymbol{L}$. If the graph is connected, the null space of $\boldsymbol{L}$ is of dimension one and equals the all-one vector $\mathbf{1}$. Thus we have

$$\begin{aligned} \boldsymbol{L}^+\boldsymbol{L} &= \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top && (2.4.11)\\ (\boldsymbol{L}^+\boldsymbol{L})\boldsymbol{H} &= \boldsymbol{L}^+\boldsymbol{D}\mathbf{1}\mathbf{1}^\top - \boldsymbol{L}^+\boldsymbol{D}\boldsymbol{M}_d && (2.4.12)\\ \boldsymbol{H} &= \boldsymbol{L}^+\boldsymbol{D}\mathbf{1}\mathbf{1}^\top - \boldsymbol{L}^+\boldsymbol{D}\boldsymbol{M}_d + \mathbf{1}\boldsymbol{u}^\top, && (2.4.13) \end{aligned}$$

for some vector $\boldsymbol{u}$. As $\boldsymbol{D}\boldsymbol{M}_d = \mathrm{vol}(G)\boldsymbol{I}$ and $\boldsymbol{H}_{ii} = 0$, we have

$$\begin{aligned} \boldsymbol{H}_{ij} &= \sum_{k=1}^n \boldsymbol{L}^+_{ik}\boldsymbol{d}_k - \mathrm{vol}(G)\boldsymbol{L}^+_{ij} + \boldsymbol{u}_j && (2.4.14)\\ \boldsymbol{u}_i &= -\sum_{k=1}^n \boldsymbol{L}^+_{ik}\boldsymbol{d}_k + \mathrm{vol}(G)\boldsymbol{L}^+_{ii}. && (2.4.15) \end{aligned}$$

Finally, from the last equations, we have the commute-time distance:

$$\mathrm{ctd}(v_i, v_j) = \boldsymbol{H}_{ij} + \boldsymbol{H}_{ji} = \mathrm{vol}(G)(\boldsymbol{L}^+_{ii} + \boldsymbol{L}^+_{jj} - 2\boldsymbol{L}^+_{ij}) = \mathrm{vol}(G)(\boldsymbol{e}_i - \boldsymbol{e}_j)^\top \boldsymbol{L}^+(\boldsymbol{e}_i - \boldsymbol{e}_j). \tag{2.4.16}$$

The matrix $\boldsymbol{L}^+$ is positive semi-definite. Moreover, if $\boldsymbol{u}$ is an eigenvector of $\boldsymbol{L}^+$ with eigenvalue $\lambda \neq 0$, then $\boldsymbol{u} \perp \mathbf{1}$, thus the coordinates of $\boldsymbol{u}$ are centered around 0. Hence the matrix $\boldsymbol{L}^+$ is a covariance matrix and equation (2.4.16) is expressing the commute-time distance as a Mahalanobis distance. Therefore, there exists a matrix $\boldsymbol{V} = \begin{pmatrix}\boldsymbol{v}_1 & \dots & \boldsymbol{v}_n\end{pmatrix}$ such that $\boldsymbol{L}^+ = \boldsymbol{V}^\top \boldsymbol{V}$. The data points $\{\boldsymbol{v}_i\}_{i=1}^n$ then represent the coordinates of the nodes of the graph in an Euclidean space, such that distances between data points are proportional to the commute-time distance in the graph:

$$\operatorname{ctd}(v_i, v_j) \propto (\boldsymbol{v}_i - \boldsymbol{v}_j)^\top (\boldsymbol{v}_i - \boldsymbol{v}_j). \tag{2.4.17}$$

The construction $\{\boldsymbol{v}_i\}_{i=1}^n$ is called the spectral embedding of the graph. For any node v_i, the data point $\boldsymbol{v}_i$ equals

$$\boldsymbol{v}_i = \begin{pmatrix}\frac{1}{\lambda_2}\boldsymbol{u}_2(i) & \dots & \frac{1}{\lambda_n}\boldsymbol{u}_n(i)\end{pmatrix}, \tag{2.4.18}$$

where λ_j and $\boldsymbol{u}_j$ are eigenvalues and eigenvectors of the combinatorial Laplacian matrix $\boldsymbol{L}$.

2.5 Evaluation of clustering

In the literature, authors use different approaches to evaluate the output of their algorithms.

A first approach, and most commonly used, consists in comparing the partition returned by your algorithm (or system partition) with some reference (or truth, or target) partition. This is the “classification” approach. We have different algorithms (or scorers) to compare two partitions (for example ARI, NMI, CoNLL score, etc). Given a reference partition, these algorithms produce a partial order of system partitions. This partial order is supposed to tell whether a partition is “better” than another for a given reference partition. Different scorers will not produce the same partial order for the same reference partition. What is the best scorer ? Is there anything like a best scorer ? Since different scorers reward for different things (large or small clusters can be favored), one first answer to this question is “it depends on what is important to us”.

A second approach consists in showing that your algorithm performs better results than other algorithms with regard to some objective function. Typically, your algorithm is designed to minimize this objective function. And since you are comparing this algorithm with algorithms which have not been designed with this objective function in mind, of course your algorithm usually performs better. However, what makes a good objective function ? Even if some intuition seems to make sense, this question is difficult to answer when confronted to the real world. Regardless of the performance of the algorithm, if the objective function it optimizes fails to capture some important (and sometimes hidden) particularities of the task, then it will fail when confronted to real world datasets. In the context of this thesis, we introduce constraints to guide clustering towards a solution that suits us better. We think this should help when faced with these cases where important particularities of the data were not captured. A possible difficulty, however, is that there are many different ways to satisfy a large portion of the constraints, in particular when there are few constraints. Thus, one should not necessarily think that a particular algorithm does not its job for a particular dataset because the system partition it produces reports a low score when compared to a reference partition. This may only mean that there are more than one solution to the problem.

Sometimes clustering is only a part of a larger system of prediction. In that case, the evaluation metric is often the measure in terms of performance of the whole process. In other words, the clustering subtask is evaluated indirectly. Usually clusters obtained in such systems are not required to make sense in themselves. What is important is their capacity to improve the performance of the global system. This could be the case with natural language processing, as clustering is only a step toward other tasks, there are works on the application of coreference resolvers in other tasks: automatic summary [78], automatic translation [57], etc. However it is usually evaluated directly using an approach using different scorers and combinations of scorers, see Luo [50]. In natural language processing, clusters are meaningful: they correspond to real entities (like persons, places, etc.) or events. There are numerous different tasks in natural language processing that would benefit from a good coreference resolver (Coreference resolution is the task of clustering parts of text (or mentions) according to the referred entities.). And natural language processing tasks are so complex that it is very hard to determine what is really evaluated when subparts of a long chain of processes is only evaluated at the end. As coreference resolvers require some preprocessing of the text (like mention detection, syntactic parsing, etc.), it is already hard to determine what is evaluated when evaluating the coreference resolver directly, because authors do not necessarily use the same algorithms for preprocessing, even when they try to reproduce results of their colleagues.

In the following, we describe which evaluation measures have been used in the experiments. In the coreference literature, there is a debate revolving around which evaluation measure is most informative. Through the years different evaluation measures have been proposed. Today, the consensus is to use the CoNLL score, which is the mean of three measures: MUC [80], B^3 [5] an Entity- based CEAF (or CEAF$_e$) [50], because these measures are supposed to capture different aspects of the comparison between the reference partition $\mathcal{P}$ and the system partition $\widehat{\mathcal{P}}$. Other measures are traditionally used in clustering tasks, such as Adjusted Rand Index or ARI [33], Normalized Mutual Information or NMI, and Variation of Information or VI [56]. In our experiments, we use all of them but we only report CoNLL score and ARI.

In the following, $\boldsymbol{p}_i$ (resp. $\widehat{\boldsymbol{p}}_i$) denotes the cluster indicator for reference (resp. system) cluster i. A cluster indicator is a vector $\boldsymbol{p}$ with binary components, $\boldsymbol{p}_i = 1$ if the ith item belong to that cluster, otherwise $\boldsymbol{p}_i = 0$.

MUC is computed by counting the number of common links between the reference and the system partition. The precision (resp. recall) is equal to that number divided by the number of links in the system (resp. reference) partition.

$$\text{(precision)} = \frac{\sum_{ij} \max(0, \widehat{\boldsymbol{p}}_i^\top \boldsymbol{p}_j - 1)}{\sum_i (\widehat{\boldsymbol{p}}_i^\top \widehat{\boldsymbol{p}}_i - 1)} \quad \text{(recall)} = \frac{\sum_{ij} \max(0, \widehat{\boldsymbol{p}}_i^\top \boldsymbol{p}_j - 1)}{\sum_i (\boldsymbol{p}_i^\top \boldsymbol{p}_i - 1)} \tag{2.5.1}$$

MUC does not reward the discovery of singletons, as it is a ratio of count of links, and there is zero link in singleton clusters. Moreover, it favors partitions with larger clusters. MUC will report a precision of 1 for reference partitions consisting of only singletons and a recall of 1 for reference partitions consisting of one single cluster, regardless of the system partition. These evaluations are, of course, incorrect.

B^3 looks at the presence or absence of entities relative to each other entities in the equivalence classes. Precision and recall are computed for each entity in the document (including singletons), which are then combined to produce the precision and recall for the document. Let $\boldsymbol{R} = \sum_i \boldsymbol{p}_i \boldsymbol{p}_i^\top$ and $\boldsymbol{S} = \sum_i \widehat{\boldsymbol{p}}_i \widehat{\boldsymbol{p}}_i^\top$.

$$\text{(precision)} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\boldsymbol{R}_{ij} \odot \boldsymbol{S}_{ij}}{\boldsymbol{S}_{ij}} \quad \text{(recall)} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\boldsymbol{R}_{ij} \odot \boldsymbol{S}_{ij}}{\boldsymbol{R}_{ij}} \tag{2.5.2}$$

where $\odot$ is the element-wise product. Unlike MUC, B^3 rewards the discovery of singletons. However, because B^3 uses the same entity "intersecting" procedure as MUC, it can give the same counter-intuitive results: sometimes reporting a precision or a recall of 1, despite clusters being incorrect or incomplete.

CEAF$_e$ introduces a similarity $\phi(\boldsymbol{p}, \widehat{\boldsymbol{p}}) = \frac{2\boldsymbol{p}^\top \widehat{\boldsymbol{p}}}{\sum_i \boldsymbol{p}_i + \sum_i \widehat{\boldsymbol{p}}_i}$ between a reference cluster $\boldsymbol{p}$ and a system cluster $\widehat{\boldsymbol{p}}$. Then it computes precision and recall as follows:

$$\text{(precision)} = \frac{\sum_i \phi(\boldsymbol{p}_i, \widehat{\boldsymbol{p}}_i)}{\sum_i \phi(\widehat{\boldsymbol{p}}_i, \widehat{\boldsymbol{p}}_i)} \quad \text{(recall)} = \frac{\sum_i \phi(\boldsymbol{p}_i, \widehat{\boldsymbol{p}}_i)}{\sum_i \phi(\boldsymbol{p}_i, \boldsymbol{p}_i)} \tag{2.5.3}$$

However, in order to perform this operation, reference and system clusters have to be aligned, which can be done by maximizing the trace of the resized $n \times n$ matrix $\boldsymbol{C}$, where n is the greatest between the number of clusters in reference partition and the number of clusters in system partiton:

$$\boldsymbol{C}_{ij} = \begin{cases} \phi(\boldsymbol{p}_i, \widehat{\boldsymbol{p}}_j) & \text{if } \boldsymbol{p}_i \text{ and } \widehat{\boldsymbol{p}}_j \text{ both exist} \\ 0 & \text{otherwise} \end{cases} \tag{2.5.4}$$

Various algorithms can be used to solve this problem in polynomial time exactly, for example Kuhn-Munkres Hungarian algorithm [38].

Adjusted Rand Index (or ARI) is the corrected-for-chance Rand Index. The latter compares two clusterings by counting correctly classified pairs of items. Let $\boldsymbol{r} = \text{vec}\big(\sum_i \boldsymbol{p}_i \boldsymbol{p}_i^\top\big)$ and $\boldsymbol{s} = \text{vec}\Big(\sum_i \widehat{\boldsymbol{p}}_i \widehat{\boldsymbol{p}}_i^\top\Big)$. The different counts of pairs of items are:

$$\begin{aligned}
a &= \frac{\boldsymbol{r}^\top \boldsymbol{s} - n}{2} && \text{(true positive)} \\
b &= \frac{(\mathbf{1} - \boldsymbol{r})^\top (\mathbf{1} - \boldsymbol{s})}{2} && \text{(true negative)} \\
c &= \frac{(\mathbf{1} - \boldsymbol{r})^\top \boldsymbol{s}}{2} && \text{(false positive)} \\
d &= \frac{\boldsymbol{r}^\top (\mathbf{1} - \boldsymbol{s})}{2} && \text{(false negative)}
\end{aligned} \tag{2.5.5}$$

The Rand Index is equal to

$$\mathrm{RI}(\mathcal{P}, \widehat{\mathcal{P}}) = \frac{a+b}{a+b+c+d} \tag{2.5.6}$$

Letting the confusion matrix $\left[\boldsymbol{C}_{ij}\right] = \boldsymbol{p}_i^\top \widehat{\boldsymbol{p}}_j$, and the vectors $\boldsymbol{a} = \boldsymbol{C}\mathbf{1}$ and $\boldsymbol{b} = \boldsymbol{C}^\top \mathbf{1}$, it is easy to verify that the Rand Index equals:

$$\mathrm{RI}(\mathcal{P}, \widehat{\mathcal{P}}) = \frac{\binom{n}{2} - \mathrm{dis}(\mathcal{P}, \widehat{\mathcal{P}})}{\binom{n}{2}}, \tag{2.5.7}$$

where $\mathrm{dis}(\mathcal{P}, \widehat{\mathcal{P}}) = \sum_i \binom{\boldsymbol{a}_i}{2} + \sum_i \binom{\boldsymbol{b}_i}{2} - 2\sum_{ij} \binom{\boldsymbol{C}_{ij}}{2}$ is the number of disagreements between reference and system partitions. The expected value of the Rand Index is not constant, it equals:

$$\mathbb{E}(\mathrm{RI}(\mathcal{P}, \widehat{\mathcal{P}})) = \binom{n}{2} - \left(\sum_i \binom{\boldsymbol{a}_i}{2} + \sum_i \binom{\boldsymbol{b}_i}{2} - 2\frac{\sum_i \binom{\boldsymbol{a}_i}{2} \sum_i \binom{\boldsymbol{b}_i}{2}}{\binom{n}{2}} \right) \tag{2.5.8}$$

Adjusted Rand Index is the corrected-for-chance Rand Index:

$$\text{Adjusted Rand Index} = \frac{\text{number of agreements} - \text{expected index}}{\text{max agreements} - \text{expected index}}$$

After some algebra, we get the following formula:

$$\mathrm{ARI}(\mathcal{P}, \widehat{\mathcal{P}}) = \frac{\sum_{ij} \binom{\boldsymbol{C}_{ij}}{2} - \frac{\sum_i \binom{\boldsymbol{a}_i}{2} \sum_i \binom{\boldsymbol{b}_i}{2}}{\binom{n}{2}}}{\frac{1}{2}\left(\sum_i \binom{\boldsymbol{a}_i}{2} + \sum_i \binom{\boldsymbol{b}_i}{2}\right) - \frac{\sum_i \binom{\boldsymbol{a}_i}{2} \sum_i \binom{\boldsymbol{b}_i}{2}}{\binom{n}{2}}} \tag{2.5.9}$$

The Adjusted Rand Index takes values between -1 and 1. Zero means the system partition is not better than a random partition, while 1 means the system and reference partitions are equal. Unlike CEAF$_e$, ARI does not require to solve any problem, therefore its computation is very fast.

Normalized Mutual Information is a measure of the mutual dependence between two variables (in our case, between reference and system partitions). More specifically, it quantifies the amount of information that can be recovered about one variable, through the other variable. It equals:

$$\mathrm{NMI}(\mathcal{P},\widehat{\mathcal{P}}) = \frac{I(\mathcal{P},\widehat{\mathcal{P}})}{H(\mathcal{P},\widehat{\mathcal{P}})} = \frac{H(\mathcal{P}) + H(\widehat{\mathcal{P}}) - H(\mathcal{P},\widehat{\mathcal{P}})}{H(\mathcal{P},\widehat{\mathcal{P}})}, \tag{2.5.10}$$

where $I(\mathcal{P},\widehat{\mathcal{P}})$ is the mutual information between $\mathcal{P}$ and $\widehat{\mathcal{P}}$, $H(\mathcal{P})$ is the entropy of $\mathcal{P}$ and $H(\mathcal{P},\widehat{\mathcal{P}})$ is the joint entropy of $\mathcal{P}$ and $\widehat{\mathcal{P}}$. Notice that $H(\mathcal{P}) = H(\mathcal{P},\mathcal{P})$ and the joint entropy of $\mathcal{P}$ and $\widehat{\mathcal{P}}$ is

$$H(\mathcal{P},\widehat{\mathcal{P}}) = -\sum_{ij:\boldsymbol{p}_i^\top \widehat{\boldsymbol{p}}_j \neq 0} \left(\frac{\boldsymbol{p}_i^\top \widehat{\boldsymbol{p}}_j}{n} \log \frac{\boldsymbol{p}_i^\top \widehat{\boldsymbol{p}}_j}{n} \right) \tag{2.5.11}$$

Variation of Information is a measure of the distance between two partitions. It equals to:

$$\mathrm{VI} = H(\mathcal{P}|\widehat{\mathcal{P}}) + H(\widehat{\mathcal{P}}|\mathcal{P}) \tag{2.5.12}$$

The variation of information is a distance.

There is a relation between ARI, NMI and VI. The relation between NMI and VI is quite obvious and illustrated in figure 2.3. For the relation between ARI and the two others, consider the function:

$$d(\mathcal{P},\widehat{\mathcal{P}},f) = \sum_{ij:\boldsymbol{p}_i^\top \widehat{\boldsymbol{p}}_j \neq 0} f(\boldsymbol{p}_i^\top \boldsymbol{p}_j) + f(\widehat{\boldsymbol{p}}_i^\top \widehat{\boldsymbol{p}}_j) - 2f(\boldsymbol{p}_i^\top \widehat{\boldsymbol{p}}_j) \tag{2.5.13}$$

The value of $d(\mathcal{P},\widehat{\mathcal{P}},f)$ is either the number of disagreements when $f(x) = \binom{x}{2}$, or the variation of information when $f(x) = \frac{x \log x}{n}$.

$$d\left(\mathcal{P},\widehat{\mathcal{P}},x \to \binom{x}{2}\right) = \mathrm{dis}(\mathcal{P},\widehat{\mathcal{P}}) \tag{2.5.14}$$

$$d\left(\mathcal{P},\widehat{\mathcal{P}},x \to \frac{x \log x}{n}\right) = \mathrm{VI}(\mathcal{P},\widehat{\mathcal{P}}) \tag{2.5.15}$$

x y t 0.2720039125760934 -0.3026690299695173 1 0.2973622194719073 -0.2916199584344616
1 0.3149367855811935 -0.308933966588549 1 0.3112453316589759 -0.2805130586143653
1 0.2823551169961314 -0.2834448019911391 1 0.2865230976654942 -0.2580654002755283
1 0.315822001499513 -0.2769932527717872 1 0.3428517674049123 -0.266866363219666l
1 0.3288868170383221 -0.2871061599261159 1 0.3513096267015724 -0.281269964643233
1 0.3226426849516429 -0.2770305167751901 1 0.3310134780685944 -0.2776683591434725
1 0.3525812779350509 -0.2558657485547249 1 0.3497536586160158 -0.213940327888477
1 0.3509228125738454 -0.2503168246391781 1 0.3463753137038432 -0.25480967844664
1 0.3921877451752307 -0.2126038813167247 1 0.3259544453064698 -0.2405951828325522
1 0.3457949438207012 -0.1992488830687905 1 0.3583632178526911 -0.2193840239758695
1 0.356203656705777 -0.2168365101815135 1 0.3700158638158145 -0.1713535348121177
1 0.4077799445645798 -0.1985633673558848 1 0.3766263786302073 -0.162999569272523
1 0.3652495878508265 -0.2066770043989561 1 0.4376707295405931 -0.1841394603657381
1 0.3786709232257706 -0.1624852621663322 1 0.3902784196869949 -0.1681761582912379
1 0.4083547646045773 -0.1518051344264901 1 0.3951955889322587 -0.1700410610035573
1 0.392803964556462 -0.1495100728337154 1 0.4066561740183158 -0.1328203962903574
1 0.4060634189703441 -0.1614331017978678 1 0.4389906545599266 -0.1386709226137353
1 0.437568729356864 -0.09246856780335555 1 0.4290273291336143 -0.1183794009865103
1 0.4401369350195206 -0.1304167370429258 1 0.4359980875374245 -0.1149887727774656
1 0.4140725515784223 -0.1096767309197517 1 0.4527942301167885 -0.07349460965326643
1 0.4501559683243113 -0.07010175117459766 1 0.4507756615294292 -0.07194361949073347
1 0.4286578107660167 -0.05120688636632909 1 0.4176299604793207 -0.04099542928151717
1 0.4349602227619861 -0.05461965755046849 1 0.4628119914887487 -0.08336939698737983
1 0.4503126493975212 -0.05067395507525674 1 0.4463996117452594 -0.05159456742215546
1 0.4411216980889442 -0.04909518066712515 1 0.4463309097205318 -0.06956153402502058
1 0.4531785682122035 0.01100082294139015 1 0.4490912856303364 -0.001803832742944489
1 0.4767451827279239 0.0187120164533785 1 0.4254314762394453 -0.004338586056270688
1 0.4500729190317102 0.0029267687856257 1 0.4695467964695957 0.03673049346805646
1 0.4564677459700683 0.04247912811261145 1 0.4534058460080149 -0.002922252609490417
1 0.4415809551339048 0.03070076330669569 1 0.4418304803142672 0.05076606042739062
1 0.4435966641785311 0.0170108554090086 1 0.4990445562144759 0.0521341571013072
1 0.4601221956309572 0.06528667323434989 1 0.4571446883541983 0.05426088590245448
1 0.4629051495119627 0.07937577232683757 1 0.4903102909782825 0.07852093834703699
1 0.4775279603774845 0.06647040441599605 1 0.4681981820970865 0.1248681434541908
1 0.4386526346009945 0.1032079858422614 1 0.4708021585576214 0.08613458667217577
1 0.4634039471972722 0.1123611552073249 1 0.4717693744126773 0.1311951502681948
1 0.4521549818641659 0.1679992993820828 1 0.4376395802923131 0.1395990510253482
1 0.4425698661041996 0.1566194788673025 1 0.4567329933805424 0.179062511252442
1 0.4703384001416243 0.1550277999707786 1 0.4374820947433483 0.136150073245618
1 0.407024673344386 0.1462412408028358 1 0.4431615557417879 0.1380240727759822
1 0.4368635954197376 0.2296775209838757 1 0.4517615288319545 0.1898888334071965
1 0.4536236291708012 0.2468015653440181 1 0.4127536729890849 0.2205185697175839
1 0.4193074137295507 0.202185833841709 1 0.3923782690213573 0.222203718558187
1 0.3975699405140011 0.2498761151959774 1 0.4265664554800378 0.2381375746792691
1 0.4065414682740651 0.2256434463997634 1 0.4284864940048753 0.2398635770539801
1 0.4483869128487504 0.2438069918091725 1 0.3887630018205074 0.265930373502593
1 0.4107478702483459 0.2677185802729084 1 0.3862351133021883 0.2298309627474814
1 0.3993785417687116 0.2738613918485734 1 0.3779483946334825 0.2942130597162463
1 0.381227976886625 0.2627335826324214 1 0.38175059224087 0.3314438849540879
1 0.4049084225209499 0.329528675124357 1 0.3851545860072717 0.3064270391259604
1 0.3773529006184126 0.2992551247548314 1 0.3672003137305871 0.2927136287250467
1 0.3929806615868593 0.3454271163180278 1 0.3940057390880888 0.3452846619517635
1 0.3619477569508601 0.3259880505918271 1 0.3767032027522357 0.3504439920020948
1 0.3673231102840899 0.3137850883349569 1 0.3484845142212207 0.3632941084762984
1 0.3425071135342116 0.3730614665810011 1 0.304020936925637 0.3521599500412196
1 0.3326742914004884 0.3513643868851059 1 0.3380801606472087 0.3770796404991194
1 0.3159350871848755 0.3634337503682931 1 0.3181927105997517 0.3668388694292252
1 0.3242026285208922 0.3861643812673197 1 0.329278535084166 0.3621069624553789
1 0.3466677284806143 0.3744027932386846 1 0.3358345243862991 0.3699089913118387
1 0.27818243522342 0.3611838096062738 1 0.3126098935161067 0.4094670318222446
1 0.312758592837158 0.4225274662609576 1 0.299871592047723 0.4199548228110029
1 0.26364486453032 0.4012405663740117 1 0.2671231148223756 0.3965132835715972

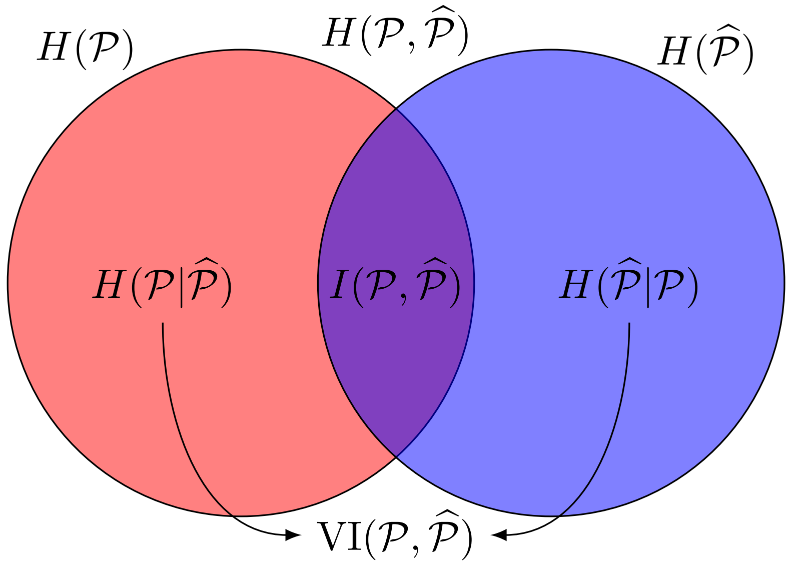

$H(\mathcal{P})$
$H(\mathcal{P},\widehat{\mathcal{P}})$
$H(\widehat{\mathcal{P}})$
$H(\mathcal{P}|\widehat{\mathcal{P}})$
$I(\mathcal{P},\widehat{\mathcal{P}})$
$H(\widehat{\mathcal{P}}|\mathcal{P})$
$\mathrm{VI}(\mathcal{P},\widehat{\mathcal{P}})$

Figure 2.3: Venn diagram for various information measures associated with partitions $\mathcal{P}$ and $\widehat{\mathcal{P}}$. The area contained by both circles is the joint entropy $H(\mathcal{P},\widehat{\mathcal{P}})$. The circle on the left (red and violet) is the individual entropy $H(\mathcal{P})$, with the red being the conditional entropy $H(\mathcal{P}|\widehat{\mathcal{P}})$. The circle on the right (blue and violet) is $H(\widehat{\mathcal{P}})$, with the blue being $H(\widehat{\mathcal{P}}|\mathcal{P})$. The violet is the mutual information $I(\mathcal{P},\widehat{\mathcal{P}})$.

Chapter 3

Label constraints

Contents

In this chapter, we study clustering when label constraints are available. That is we are given a graph in which few nodes are labeled with a cluster number and the task consists in using both the information from the graph and the labels to recover the labels of all the nodes of the graph. The labeling information may be costly to obtain, particularly when this knowledge comes from a domain where an expert is labeling the nodes, but we assume that only few nodes need to be supervised. Thus, we are in a context where labels of few nodes are known and labels of the vast majority of the remaining nodes are unknown. Moreover, as we are working on semi-supervised learning approaches, we would like to use the unlabeled nodes of the graph as a part of the learning process. To do so, we must assume that the structure of the graph already yields some valuable information. This is why we can make use of at least one of the following assumptions.

Smoothness assumption Nodes which are close to each other are more likely to be in the same cluster.

Cluster assumption Clusters tend to be dense and well-separated.

Manifold assumption The data lie approximatively on a manifold whose dimension is much lower than the input space.

We introduce label constraints to correct the input data when those assumptions are not completely satisfied. For example, the cluster assumption states that clusters tend to be dense and well-separated, however it is possible that a cluster is divided into two sub-clusters whose properties differ enough from each other so that they are separated in the input space. In such case, a label constraint for each sub-cluster can be used to reform the desired cluster. It is also possible that two clusters are not well- separated but interleaved in a such way that unsupervised algorithms will not be able to make the difference between them.

In Belkin *et al.* [9], these assumptions are used to provide a semi- supervised framework that incorporates labeled and unlabeled data in a general- purpose learner. Labeled data are fixed, so we are in a hard-constrained setup. The authors show that some graph learning algorithms and some other standard methods can be obtained as special cases. This includes support vector machines and regularized least squares. Another well known algorithm that uses the smoothness assumption and the manifold assumption in conjunction to hard label constraints is label propagation. We dedicate Section 3.1 to this algorithm and a review is also available in Zhu [91]. Bengio *et al.* [10] discuss a common framework for various label propagation, spreading and laplacian regularization. Label propagation has links with diffusion of information in a graph, and can be used to study how information propagate in a social network or a computer network. Hard label constraints have also been used to modify k-means. In Basu *et al.* [6], the authors use label constraints to compute approximations of cluster centroids.

When labeled nodes are not fixed, we are in soft-constrained setups. We may find that a labeled node has been mistakenly labeled and we might want to correct that error with a mode appropriate label. In Basu *et al.* [7] and Davidson and Ravi [19] ignoring some constraints is allowed if their satisfaction leads to a significant degradation of the objective function. In Zhou *et al.* [89], label constraints are used to design an iterative algorithm that diffuses labels through the graph. This is somehow similar to label propagation, however in this case the labels are not fixed. The authors also provide a closed-form for the solution obtained by their algorithm. In Section 3.2, we review Mavroeidis [54, 55], in which the author uses label constraints to accelerate computations required by spectral clustering. However, this method can also be used to integrate partial label information into the “classic” unsupervised spectral clustering.

In any case, clustering algorithms with label constraints often imply that we know the number of clusters beforehand and that each cluster has at least one supervised node. Among the different kinds of semi-supervision studied in this thesis, label constraints are the strongest and most restrictive constraints. Clustering algorithms with label constraints are also the most similar to classification algorithms. Particularly in the case of hard constraints, as all the constraints must be satisfied. Moreover, clustering with label constraints implies that nodes are constrained to a specific *cluster label* rather than a specific cluster. Cluster labels are often represented by numbers, letters or colors. But actually any representation is possible. Cluster labels are not necessarily meaningful. They are only a way to link together nodes belonging to the same cluster. Because of that, cluster labels are interchangeable. In classification, class labels are meaningful, thus not interchangeable. Clustering algorithms with label constraints require that specific cluster labels are assigned to some nodes, therefore cluster labels are not, in practice, interchangeable. To our knowledge, no clustering algorithm with label constraints can produce a number of clusters different than the number of labels used for supervision. Therefore, alike classification the number of clusters must be known in advance.

In the following Sections, we will see that despite their similarity with classification algorithms, clustering algorithms with label constraints in graphs uses the manifold of the whole graph to get a good representation of all the nodes. So, both labeled and unlabeled nodes are part of the learning process. This is the main difference between classification algorithms and clustering algorithms studied in this chapter. We will begin by reviewing label propagation in Section 3.1. This is a clustering algorithm with hard label constraints that attempts to find the smallest cut of a graph given that a subset of nodes *must* be assigned to specific clusters labels. In Section 3.2, we will review an algorithm [54] that uses soft label constraints. Even though this algorithm was originally designed to accelerate the computation of eigenvectors used in spectral clustering in the case of 2 clusters, it actually integrates label constraints into unsupervised clustering in a way that does not destroy some interesting properties of the graph. The same author proposed another algorithm for clustering with more than 2 clusters [55], however in this second algorithm properties of the graph are not maintained. In Section 3.3 we will present our contribution, extending Mavroeidis [54] to more than 2 clusters, while maintaining said properties of the graph. We will also explain its advantages over the original proposition Mavroeidis [55].

3.1 Label Propagation

Label propagation consists in the diffusion of the known label of some nodes of a graph to their neighbors whose labels are unknown. You can grasp this process by picturing the graph as a water supply network, whose tubes are of different sizes, and known labels as colored water valves attached to some nodes of the water supply network. If you have two different labels, then you have two different water valves that pour red and blue water indefinitely for example. As water valves continuously pour colored water into the water supply network, it quickly becomes filled with water whose color is a mixture of the different colors. Over time the color in the water supply network converges to a stabilization point. Then, the final color obtained is used to determine the label of the different nodes of the network.

In this section, we present this process, known as label propagation. Because working on disconnected graphs is actually the same as propagating labels in each connected component independently, we will only consider connected graphs, that is a graph where, for each pair of nodes, there exists a path connecting these nodes. An iterative algorithm is presented in [91] to solve this problem. The solution given by this algorithm has a closed form. In [1], a generalized formulation of the label propagation problem is discussed. This formulation leads to different algorithms, including the iterative algorithm presented in [91].

In the following, we denote by l the indices of labeled nodes and by u the indices of unlabeled nodes. We define a $|l| \times k$ indicator matrix $\boldsymbol{Y}_l$ for the labeled nodes, where $|l|$ is the number of labeled nodes and k is the number of clusters, such that

$$\boldsymbol{Y}_{ic} = \begin{cases} 1 & \text{if the node } i \text{ is assigned to cluster } c \\ 0 & \text{otherwise} \end{cases}$$

Unknown labels are obtained by pushing the label of neighboring nodes through the edges with a force proportional to the weight of the edges. To this end, we use the transition matrix $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{W}$ of the graph (see section 2.4.1). Recall that $\boldsymbol{P}_{ij}$ is the transition probability from node i to node j. At each iteration, the unknown labels take a value equal to the weighted mean of the label of neighboring nodes. Let $\boldsymbol{Y}(i)_t$ be the label for node i at iteration t.

$$\boldsymbol{Y}(i)_t = \sum_{j=1}^{n} \frac{\boldsymbol{W}_{ij}}{\boldsymbol{d}_i} \boldsymbol{Y}(j)_{t-1}$$

See algorithm 5 for the implementation. Line 6 ensures that labeled nodes remain the same during all the process.

Algorithm 5: Iterative Label Propagation

```
    Input: $\boldsymbol{W}$: Similarity matrix, $l$: set of labeled nodes, $\boldsymbol{Y}_l$: known labels.
    Output: $\boldsymbol{H}$: a labeling of the graph.
1   begin
2       Initialize a $n \times k$ matrix $\boldsymbol{H}$ randomly
3       $\boldsymbol{P} \leftarrow \boldsymbol{D}^{-1}\boldsymbol{W}$
4       repeat
5           $\boldsymbol{H} \leftarrow \boldsymbol{P}\boldsymbol{H}$
6           $\boldsymbol{H}_l \leftarrow \boldsymbol{Y}_l$
7       until convergence
8       return $\boldsymbol{H}$
```

Proposition 5. *Algorithm 5 converges to* $(\boldsymbol{I} - \boldsymbol{P}_{uu})^{-1}\boldsymbol{P}_{ul}\boldsymbol{Y}_l$.

proof. (See Zhu [91]). Regarding unknown labels, the algorithm 5 is equivalent to the following sequence

$$\boldsymbol{H}_u(t+1) = \boldsymbol{P}_{uu}\boldsymbol{H}_u(t) + \boldsymbol{P}_{ul}\boldsymbol{Y}_l \tag{3.1.1}$$

For $\boldsymbol{H}_u(1)$, $\boldsymbol{H}_u(2)$ and so forth we have:

$$\begin{aligned}
\boldsymbol{H}_u(1) &= \boldsymbol{P}_{uu}\boldsymbol{H}_u(0) + \boldsymbol{P}_{ul}\boldsymbol{Y}_l \\
\boldsymbol{H}_u(2) &= \boldsymbol{P}_{uu}(\boldsymbol{P}_{uu}\boldsymbol{H}_u(0) + \boldsymbol{P}_{ul}\boldsymbol{Y}_l) + \boldsymbol{P}_{ul}\boldsymbol{Y}_l \\
&= \boldsymbol{P}_{uu}^2\boldsymbol{H}_u(0) + (\boldsymbol{P}_{uu} + \boldsymbol{I})\boldsymbol{P}_{ul}\boldsymbol{Y}_l \\
\boldsymbol{H}_u(3) &= \boldsymbol{P}_{uu}(\boldsymbol{P}_{uu}^2\boldsymbol{H}_u(0) + (\boldsymbol{P}_{uu} + \boldsymbol{I})\boldsymbol{P}_{ul}\boldsymbol{Y}_l) + \boldsymbol{P}_{ul}\boldsymbol{Y}_l \\
&= \boldsymbol{P}_{uu}^3\boldsymbol{H}_u(0) + \left(\sum_{i=1}^{3}\boldsymbol{P}_{uu}^{(i-1)}\right)\boldsymbol{P}_{ul}\boldsymbol{Y}_l
\end{aligned}$$

When $t \to \infty$, we have this

$$\lim_{t\to\infty}\boldsymbol{H}_u(t) = \lim_{t\to\infty}(\boldsymbol{P}_{uu})^t\,\boldsymbol{H}_u(0) + \lim_{t\to\infty}\left(\sum_{i=1}^{t}(\boldsymbol{P}_{ul})^{(i-1)}\right)\boldsymbol{P}_{ul}\boldsymbol{Y}_l \tag{3.1.2}$$

The matrix $\boldsymbol{P}$ is row normalized and $\boldsymbol{P}_{uu}$ is a submatrix of $\boldsymbol{P}$, therefore the row sum of $\boldsymbol{P}_{uu}$ is lesser than 1 and $(\boldsymbol{P}_{uu})^t \to \boldsymbol{0}$ when $t \to \infty$. Moreover,

$$\lim_{t\to\infty}\left(\sum_{i=1}^{t}(\boldsymbol{P}_{uu})^{(i-1)}\right) = (\boldsymbol{I} - \boldsymbol{P}_{uu})^{-1} \tag{3.1.3}$$

Hence, algorithm 5 converges to the solution whose closed form is

$$\boldsymbol{H}_u = (\boldsymbol{I} - \boldsymbol{P}_{uu})^{-1}\boldsymbol{P}_{ul}\boldsymbol{Y}_l. \tag{3.1.4}$$

□

In [1], the problem is formulated as a minimization problem

$$\min_{\boldsymbol{H}} E(\boldsymbol{H}) = \operatorname{trace}\left(\boldsymbol{H}^\top\boldsymbol{L}\boldsymbol{H}\right) : \|\boldsymbol{H}_l - \boldsymbol{Y}_l\|^2 \leq \epsilon, \tag{3.1.5}$$

where $\epsilon \geq 0$ is a constant. The function E is harmonic, therefore predicted labels will satisfy $\boldsymbol{H} = \boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{H}$. Since the nodes are split into labeled and unlabeled nodes and labeled nodes are fixed, we have $\boldsymbol{H}_l = \boldsymbol{Y}_l$ and

$$\begin{pmatrix}\boldsymbol{D}_{ll} & 0 \\ 0 & \boldsymbol{D}_{uu}\end{pmatrix}\begin{pmatrix}\boldsymbol{H}_l \\ \boldsymbol{H}_u\end{pmatrix} = \begin{pmatrix}\boldsymbol{W}_{ll} & \boldsymbol{W}_{lu} \\ \boldsymbol{W}_{ul} & \boldsymbol{W}_{uu}\end{pmatrix}\begin{pmatrix}\boldsymbol{H}_l \\ \boldsymbol{H}_u\end{pmatrix}. \tag{3.1.6}$$

After simplification, we have $\boldsymbol{H}_u = (\boldsymbol{D}_{uu} - \boldsymbol{W}_{uu})^{-1}\boldsymbol{W}_{ul}\boldsymbol{Y}_l$, that is equivalent to equation 3.1.4.

At the end of Algorithm 5 or after the computation of the closed form, each node but the supervised nodes have a label equal to the mean of the labels of their neighbor weighted by the weight on the edges. This algorithm requires that each cluster has at least one constrained node. Also the propagation of labels cannot happen between disconnected components. So this algorithm either requires that the graph is connected or that each connected component is supervised.

3.1.1 Relation between label propagation and least squares

This algorithm can be interpreted as a classification algorithm rather than a clustering algorithm, for few reasons. Each cluster must be supervised, as a consequence this algorithm cannot discover new clusters. This is a hard constrained algorithm, that is all constraints must be satisfied. It implies that the algorithm cannot decide that part of the constraints were wrong. It cannot decide either that despite supervision a cluster does not exist.

Also, as first contribution on label constraints, we show that this algorithm has a close connection with least squares in the full spectral embedding of the space describing the commute-time distance between nodes of the graph, that is: the pseudo-inverse of the Laplacian.

Consider the following least square problem:

$$\begin{aligned} \boldsymbol{x}^\star &= \min_{\boldsymbol{x}} \|\boldsymbol{x}\|^2 \\ &\text{such that } \boldsymbol{A}_l\boldsymbol{x} - \boldsymbol{y}_l = 0 \end{aligned} \tag{3.1.7}$$

where $\boldsymbol{A}$ is a matrix whose rows describe the nodes; in our case, the matrix $\boldsymbol{A}$ corresponds to a space describing the commute-time distance between nodes of the graph. That is, if $\boldsymbol{V}$ is the matrix whose columns are the eigenvectors of the Laplacian matrix $\boldsymbol{L}$ of the graph and $\boldsymbol{\Lambda}$ is the diagonal matrix whose entries are the corresponding eigenvalues, then $\boldsymbol{V}(\boldsymbol{\Lambda}^+)^{1/2}$ is a matrix whose rows are the coordinates of the nodes in the space describing their commute-time distance. Likewise, in our case, we have $\boldsymbol{A} = \widehat{\boldsymbol{V}}\widehat{\boldsymbol{\Lambda}}^{-1/2}$ where $\widehat{\boldsymbol{V}} = \boldsymbol{V}$ and $\widehat{\boldsymbol{\Lambda}}$ contains the eigenvectors and eigenvalues of $\widehat{\boldsymbol{L}} = \boldsymbol{L} + \epsilon\mathbf{1}\mathbf{1}^\top$. This does not change the fact that $\boldsymbol{A}$ is also a space describing the commute-time distance between nodes of the graph, as we only increased the eigenvalue associated with the eigenvector $\mathbf{1}$, so that it is not equal to zero. When computing the commute-time distance, this dimension is irrelevant as all the nodes have the same value along this dimension. The matrix $\boldsymbol{A}$ is divided in two parts $\boldsymbol{A}_l$ and $\boldsymbol{A}_u$ for labelled nodes and unlabeled nodes. Likewise, the vector $\boldsymbol{y}$ is divided in two parts $\boldsymbol{y}_l$ and $\boldsymbol{y}_u$ corresponding to known labels and unknown labels that we must discover. We are going to show that unknown labels computed with Equation (3.1.4) are very close to labels computed by solving Equation (3.1.7) and letting $\boldsymbol{y}_u = \widehat{\boldsymbol{V}_u}\widehat{\boldsymbol{\Lambda}}^{-1/2}\boldsymbol{x}$.

Simple resolution of this problem shows that $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}$, that is:

$$\boldsymbol{x} = \widehat{\boldsymbol{\Lambda}}^{-1/2}\boldsymbol{V}_l^\top(\boldsymbol{V}_l\widehat{\boldsymbol{\Lambda}}^{-1}\boldsymbol{V}_l^\top)^{-1}\boldsymbol{y}_l$$

This gives a value for $\boldsymbol{y}_u$:

$$\begin{aligned} \boldsymbol{y}_u &= \boldsymbol{V}_u\widehat{\boldsymbol{\Lambda}}^{-1}\boldsymbol{V}_l^\top(\boldsymbol{V}_l\widehat{\boldsymbol{\Lambda}}^{-1}\boldsymbol{V}_l^\top)^{-1}\boldsymbol{y}_l \\ &= (\widehat{\boldsymbol{L}}^{-1})_{ul}(\widehat{\boldsymbol{L}}^{-1})_{ll}^{-1}\boldsymbol{y}_l \end{aligned}$$

Using the Schur complement, we know that $(\widehat{\boldsymbol{L}}^{-1})_{ul} = -\widehat{\boldsymbol{L}}_{uu}^{-1}\widehat{\boldsymbol{L}}_{ul}(\widehat{\boldsymbol{L}}_{ll} - \widehat{\boldsymbol{L}}_{lu}\widehat{\boldsymbol{L}}_{uu}^{-1}\widehat{\boldsymbol{L}}_{ul})^{-1}$ and $(\widehat{\boldsymbol{L}}^{-1})_{ll} = (\widehat{\boldsymbol{L}}_{ll} - \widehat{\boldsymbol{L}}_{lu}\widehat{\boldsymbol{L}}_{uu}^{-1}\widehat{\boldsymbol{L}}_{ul})^{-1}$. Therefore we have:

$$\boldsymbol{y}_u = -\widehat{\boldsymbol{L}}_{uu}^{-1}\widehat{\boldsymbol{L}}_{ul}\boldsymbol{y}_l \tag{3.1.8}$$

From there, if $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{W}$, then it is trivial to show that Equation (3.1.8) is equivalent to Equation (3.1.4). If $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$, then we have

$$\begin{aligned}
\boldsymbol{y}_u &= -(\boldsymbol{D} - \boldsymbol{W})_{uu}^{-1}(\boldsymbol{D}_{ul} - \boldsymbol{W}_{ul})\boldsymbol{y}_l \\
&= -(\boldsymbol{D}_{uu} - \boldsymbol{W}_{uu})^{-1}\boldsymbol{D}_{uu}\boldsymbol{D}_{uu}^{-1}(\boldsymbol{D}_{ul} - \boldsymbol{W}_{ul})\boldsymbol{y}_l \\
&= (\boldsymbol{D}_{uu}^{-1}(\boldsymbol{D}_{uu} - \boldsymbol{W}_{uu}))^{-1}\boldsymbol{D}_{uu}^{-1}\boldsymbol{W}_{ul}\boldsymbol{y}_l \\
&= (\boldsymbol{I} - \boldsymbol{P}_{uu})^{-1}\boldsymbol{P}_{ul}\boldsymbol{y}_l
\end{aligned}$$

Which is equivalent to the closed form of label propagation, as in Equation (3.1.4). Therefore, label propagation is equivalent to solving the least square problem in Equation (3.1.7). One can ask what would happen if the label constraints were not necessarily satisfied. This is what we are going to see in the next section, with two papers from Mavroedis [54, 55], in which the author's goal consists in accelerating the computation of the second eigenvector of the Laplacian, namely the Fielder vector, by introducing semi-supervision. While this goal is different from ours – giving hints to the algorithm about the desired output – we will see that this algorithm can be used to achieve our objective.

3.2 Accelerating spectral clustering with partial supervision

In this section, we discuss two clustering algorithms from Mavroedis [54, 55], in which partial supervision is introduced by adding soft label constraints. The goal of the author is to accelerate the computation of the required eigenvectors required by spectral clustering. However, his work can be used to improve the quality of clustering output, as label constraints act as a hint for the clustering algorithm. The method has the effect of increasing the spectral gap, that is the difference between the third and the second eigenvalue of the graph Laplacian. Then it uses the power method to compute the required eigenvectors. The computation time of the power method depends on the spectral gap. By increasing the spectral gap, the computation time for the power method to converge is decreased. There are other methods to compute eigenvectors which are more efficient than the power method. However, these algorithms also depend on the spectral gap in the same way. Thus, in the following, we will first quickly review the power method to understand where the acceleration comes from, then we will consider the two algorithms proposed by Mavroedis in subsections 3.2.2 and 3.2.3.

3.2.1 The power method

The power method [21] is an iterative algorithm for computing the largest eigenvector of a matrix. The computational cost of the power method depends on the cost of multiplying a matrix by a vector and the number of steps required for convergence is dependent on the ratio between the first two largest eigenvalues of that matrix.

Consider a matrix $\boldsymbol{M}$ whose ordered eigenvalues are $|\lambda_1| < \cdots < |\lambda_n|$ and corresponding eigenvectors are $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. That is:

$$\boldsymbol{M}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \text{ for all } i = 1, \ldots, n$$

The method consists in initializing a random vector $\boldsymbol{u}_0 = \sum_{i=1}^{n} \sigma_i \boldsymbol{v}_i$, then applying the following iteration until convergence:

$$\boldsymbol{u}_k = \frac{\boldsymbol{M}\boldsymbol{u}_{k-1}}{\|\boldsymbol{M}\boldsymbol{u}_{k-1}\|} \tag{3.2.1}$$

Indeed, after k iterations, we have:

$$\begin{aligned}
\boldsymbol{u}_k &= \frac{\boldsymbol{M}^k\boldsymbol{u}_0}{\left\|\boldsymbol{M}^k\boldsymbol{u}_0\right\|} \\
&= \frac{\sum_{i=1}^{n} \lambda_i^k \boldsymbol{v}_i \boldsymbol{v}_i^\top \boldsymbol{u}_0}{\left\|\sum_{i=1}^{n} \lambda_i^k \boldsymbol{v}_i \boldsymbol{v}_i^\top \boldsymbol{u}_0\right\|} \\
&= \frac{\sum_{i=1}^{n} \sigma_i \lambda_i^k \boldsymbol{v}_i}{\left\|\sum_{i=1}^{n} \sigma_i \lambda_i^k \boldsymbol{v}_i\right\|} \\
&= \frac{\sigma_n \lambda_n^k \boldsymbol{v}_n + (\sum_{i=1}^{n-1} \sigma_i \lambda_i^k \boldsymbol{v}_i)}{\left\|\sigma_n \lambda_n^k \boldsymbol{v}_n + (\sum_{i=1}^{n-1} \sigma_i \lambda_i^k \boldsymbol{v}_i)\right\|}
\end{aligned}$$

As $\lambda_n > \lambda_i$ for all $i < n$, then we have:

$$\lim_{k\to\infty} \frac{\sum_{i=1}^{n-1} \sigma_i \lambda_i^k \boldsymbol{v}_i}{\left\|\sigma_n \lambda_n^k \boldsymbol{v}_n + (\sum_{i=1}^{n-1} \sigma_i \lambda_i^k \boldsymbol{v}_i)\right\|} = \boldsymbol{0}$$

That is, $\sum_{i=1}^{n-1} \sigma_i \lambda_i^k \boldsymbol{v}_i$ is negligible compared to $\sigma_n \lambda_n^k \boldsymbol{v}_n$. Therefore, we have:

$$\lim_{k\to\infty} \boldsymbol{u}_k = \boldsymbol{v}_n \tag{3.2.2}$$

The convergence rate of the power method depends on the difference between λ_n and λ_{n-1}, also called eigen gap. As the difference between the largest eigenvalue and any other eigenvalue increases, the part of the equation corresponding to smaller eigenvalues vanishes to zero faster. Thus, as this difference gets larger, the power method converges faster.

Also notice that we can compute other eigenvectors one after the other using the power method. Applying the power method to $\boldsymbol{M}$ will return the eigenvector $\boldsymbol{v}_n$ associated with the most dominant eigenvalue. The most dominant eigenvector of $\boldsymbol{M} - \lambda_n \boldsymbol{v}_n \boldsymbol{v}_n^\top = \sum_{i=1}^{n-1} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^\top$ is $\boldsymbol{v}_{n-1}$. Thus, it is possible to compute all the eigenvectors of $\boldsymbol{M}$ by removing iteratively all the eigenvectors. To get the ith eigenvector of $\boldsymbol{M}$ using the power method, we compute $\boldsymbol{v}_n, \dots, \boldsymbol{v}_{i+1}$ and then apply the power method to the matrix $\boldsymbol{M} - \sum_{j=i+1}^{n} \lambda_j \boldsymbol{v}_j \boldsymbol{v}_j^\top$.

In Mavroeidis [54, 55], the author proposes to compute the second eigenvector of the Laplacian, namely Fiedler vector, by applying the power method on a matrix obtained from a modification of the Laplacian so that the Fiedler vector (the second smallest eigenvector of the Laplacian) becomes the largest eigenvector of the new matrix. This new matrix receives a rank-1 update that is determined by semi-supervision on the graph, in the form of label constraints.

3.2.2 2-cluster case

In Mavroeidis [54], the author proposes a method to accelerate the computation of the vector required for spectral clustering with 2 clusters. He does so by introducing partial label supervision in both classes, which has the effect of increasing the spectral gap. Then he uses the power method to compute the required vector for clustering. Label supervision is integrated as a regularizer of the normalized cut objective function. That is, regularized normalized cut objective function is:

$$\begin{aligned} &\min_{\boldsymbol{f}} \sum_{i,j\in\mathcal{V}} \boldsymbol{W}_{ij}(\boldsymbol{f}_i - \boldsymbol{f}_j)^2 + \gamma \sum_{i,j\in\mathcal{A}^{in}} \boldsymbol{Y}_{ij}(\boldsymbol{f}_i - \boldsymbol{f}_j)^2 \\ &\text{such that } \boldsymbol{D}\boldsymbol{f} \perp \boldsymbol{1} \text{ and } \boldsymbol{f}^\top \boldsymbol{D}\boldsymbol{f} = \mathrm{vol}(\boldsymbol{D}), \end{aligned} \tag{3.2.3}$$

where $\mathcal{A}^{in}$ is the set of supervised nodes and $\boldsymbol{Y}_{ij}$ corresponds to the supervision for the pair of nodes i and j. If i or j is not supervised, then $\boldsymbol{Y}_{ij} = 0$. The authors impose that the original degrees of the nodes are not modified by the supervision. That is $\sum_j \boldsymbol{Y}_{ij} = 0$ for all i. The set of supervised nodes $\mathcal{A}^{in}$ can be divided in two parts: the set of nodes that belong to the first cluster $\mathcal{A}_1^{in}$ and the set of nodes that belong to the second cluster $\mathcal{A}_2^{in}$. That is: $\mathcal{A}^{in} = \mathcal{A}_1^{in} \cup \mathcal{A}_2^{in}$ and, of course, $\mathcal{A}_1^{in} \cap \mathcal{A}_2^{in} = \varnothing$. Notice that Equation (3.2.3) can be written in quadratic form:

$$\begin{aligned} &\min_{\boldsymbol{f}} \sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{W}_{ij}(\boldsymbol{f}_i - \boldsymbol{f}_j)^2 + \gamma \sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{Y}_{ij}(\boldsymbol{f}_i - \boldsymbol{f}_j)^2 \\ &\min_{\boldsymbol{f}} \sum_{i=1}^{n}\sum_{j=1}^{n} (\boldsymbol{W}_{ij} + \gamma \boldsymbol{Y}_{ij})(\boldsymbol{f}_i - \boldsymbol{f}_j)^2 \\ &\min_{\boldsymbol{f}} \sum_{i=1}^{n}\sum_{j=1}^{n} (\boldsymbol{W}_{ij} + \gamma \boldsymbol{Y}_{ij})(\boldsymbol{f}_i^2 + \boldsymbol{f}_j^2 - 2\boldsymbol{f}_i \boldsymbol{f}_j) \\ &\min_{\boldsymbol{f}} 2\boldsymbol{f}^\top (\boldsymbol{D} - \boldsymbol{W} + \gamma(\boldsymbol{D}_Y - \boldsymbol{Y}))\boldsymbol{f} \end{aligned}$$

where $\boldsymbol{D}_Y$ is the diagonal matrix with entries $\boldsymbol{d}_Y = \boldsymbol{Y}\boldsymbol{1}$. However, as the supervision does not change the degree of the original graph, we have $\boldsymbol{D}_Y = \boldsymbol{0}$. Moreover, we can drop the factor 2, since it applies to the whole equation. Thus, after substituting $\boldsymbol{g} = \boldsymbol{D}^{1/2}\boldsymbol{f}$, the quadratic form of Equation (3.2.3) is

$$\min_{\boldsymbol{g}} \boldsymbol{g}^\top \boldsymbol{D}^{-1/2}(\boldsymbol{D} - \boldsymbol{W} - \gamma\boldsymbol{Y})\boldsymbol{D}^{-1/2}\boldsymbol{g}$$
$$\text{such that } \boldsymbol{D}^{1/2}\boldsymbol{g} \perp \boldsymbol{1} \text{ and } \boldsymbol{g}^\top\boldsymbol{g} = \mathrm{vol}(\boldsymbol{D})$$

So, what is $\boldsymbol{Y}$ exactly ? We know that the row sums of $\boldsymbol{Y}$ equal zero, by definition. Additionally, we would like that $\boldsymbol{Y}$ acts as a hint for spectral clustering, increasing similarity between nodes in the same cluster and decreasing similarity between nodes in different clusters. For unsupervised nodes, we do not want to change the similarity. That is:

- $\boldsymbol{Y}_{ij} > 0$ if i and j are in the same cluster.
- $\boldsymbol{Y}_{ij} < 0$ if i and j are in different clusters.
- $\boldsymbol{Y}_{ij} = 0$ if i or j are unsupervised.

Finally, the goal of the author is to use power method to compute the eigenvector associated with the second smallest eigenvalue of that new regularized Laplacian matrix $\widehat{\boldsymbol{L}} = \boldsymbol{D}^{-1/2}(\boldsymbol{D}-\boldsymbol{W}-\gamma\boldsymbol{Y})\boldsymbol{D}^{-1/2}$. So, we have to change the matrix $\widehat{\boldsymbol{L}}$ so that the desired eigenvector becomes the most dominant eigenvector in absolute value. Let $\lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of $\widehat{\boldsymbol{L}}$. We want the eigenvector associated with λ_2 to become associated with the largest eigenvalue of some new matrix $\overline{\boldsymbol{L}}$. Consider the ordered eigenvalues of the following matrices:

- For $\widehat{\boldsymbol{L}}$, we have $0 = \lambda_1 \leq \cdots \leq \lambda_n \leq 2$.
- For $-\widehat{\boldsymbol{L}}$, we have $-2 \leq -\lambda_n \leq \cdots \leq -\lambda_1 = 0$.
- For $2\boldsymbol{I} - \widehat{\boldsymbol{L}}$, we have $0 \leq 2 - \lambda_n \leq \cdots \leq 2 - \lambda_1 = 2$.
- For $2\boldsymbol{I} - \widehat{\boldsymbol{L}} - 2\boldsymbol{u}\boldsymbol{u}^\top$, we have $0 = \lambda_1 \leq 2 - \lambda_n \leq \cdots \leq \lambda_2 \leq 2$.

where, $\boldsymbol{u}$ is the normalized eigenvector of $\widehat{\boldsymbol{L}}$ associated with λ_1. Notice that all these operations preserve the associated eigenvectors. For $2\boldsymbol{I}-\widehat{\boldsymbol{L}}-2\boldsymbol{u}\boldsymbol{u}^\top$, the second eigenvalue λ_2 becomes dominant.

This shows why not changing the original degree matrix of the graph is so important. If the degree matrix of the graph changed, we would not be able to remove the eigenvector $\boldsymbol{u}$ so easily. In order not to change the degree matrix of the graph, the row sums of the matrix $\boldsymbol{Y}$ must be equal to zero. Moreover, $\boldsymbol{Y}$ has to be symmetric. To obtain this, Mavroeidis proposes to compute $\boldsymbol{Y}$ as a rank-1 matrix:

$$\begin{aligned}
\boldsymbol{Y} &= \boldsymbol{D}^{1/2}\boldsymbol{v}_1\boldsymbol{v}_1^\top\boldsymbol{D}^{1/2} \\
\boldsymbol{v}_1(i) &= \begin{cases} \sqrt{\frac{\boldsymbol{d}_i}{\mathrm{vol}(\mathcal{A}^{in})}} f(i) & \text{if } i \in \mathcal{A}^{in} \\ 0 & \text{otherwise} \end{cases} \\
f(i) &= \begin{cases} \sqrt{\frac{\mathrm{vol}(\mathcal{A}_2^{in})}{\mathrm{vol}(\mathcal{A}_1^{in})}} & \text{if } i \in \mathcal{A}_1^{in} \\ -\sqrt{\frac{\mathrm{vol}(\mathcal{A}_1^{in})}{\mathrm{vol}(\mathcal{A}_2^{in})}} & \text{if } i \in \mathcal{A}_2^{in} \end{cases}
\end{aligned} \tag{3.2.4}$$

While this is not the only possibility to make $\boldsymbol{Y}$ symmetric and have its rows sum to zero, $f(i)$ is reminiscent of the form used by Luxburg [52] (See Chapter 2) for the discrete vector to optimize during 2-way spectral clustering. The form $\boldsymbol{d}_i/\mathrm{vol}(\mathcal{A}^{in})$ is just there for ensuring that $\boldsymbol{v}_1$ has unit-norm. Plugging the regularization and the trick to compute the second eigenvector with the power method altogether gives Algorithm 6 and the following objective function:

$$\begin{aligned}
&\max_{\boldsymbol{g}} \boldsymbol{g}^\top \left(\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2} + \gamma\boldsymbol{v}_1\boldsymbol{v}_1^\top - 2\boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{I}\right)\boldsymbol{g} \\
&\text{such that } \boldsymbol{D}^{1/2}\boldsymbol{g} \perp \boldsymbol{1} \text{ and } \boldsymbol{g}^\top\boldsymbol{g} = \mathrm{vol}(\boldsymbol{D})
\end{aligned} \tag{3.2.5}$$

Algorithm 6: MAVRO

```
Input: Similarity matrix $\boldsymbol{W}$, Semi-supervision $\mathcal{A}_1^{in}$ and $\mathcal{A}_2^{in}$, Regularization
       parameter $\gamma$.
Output: Cluster assignments for 2 clusters.
1 begin
2     Compute $\boldsymbol{Y}$ as in Equation (3.2.4)
3     Compute $\boldsymbol{u} = \frac{\boldsymbol{D}^{1/2}\boldsymbol{1}}{\|\boldsymbol{D}^{1/2}\boldsymbol{1}\|}$
4     Compute $\overline{\boldsymbol{L}} = \boldsymbol{D}^{-1/2}(\boldsymbol{W} + \gamma\boldsymbol{Y})\boldsymbol{D}^{-1/2} - 2\boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{I}$
5     Apply the power method to $\overline{\boldsymbol{L}}$, store the result in $\boldsymbol{f}$
6     Assign nodes with a positive value in $\boldsymbol{f}$ to one cluster and the nodes with
        negatives values to the other cluster
```

3.2.3 k-cluster case

In [55], Mavroeidis proposes an extension to handle more than 2 clusters. The method is also reminiscent of Luxburg [52]. The author considers as input a set of cluster labels $\mathcal{A}_i^{in} \subseteq \mathcal{A}_i$ for each cluster and defines a vector $\boldsymbol{v}_i$ as:

$$\boldsymbol{v}_j(i) = \sqrt{\frac{\boldsymbol{d}_i}{\mathrm{vol}(\mathcal{A}_j^{in})}} \text{ if } i \in \mathcal{A}_j^{in} \quad \text{and } \boldsymbol{v}_j(i) = 0 \text{ otherwise.} \tag{3.2.6}$$

Then, the problem becomes:

$$\begin{aligned} &\min_{\boldsymbol{H}} \boldsymbol{H}^\top \left(\boldsymbol{I} - \boldsymbol{D}^{-1/2}(\boldsymbol{W} + \gamma \boldsymbol{Y})\boldsymbol{D}^{-1/2}\right) \boldsymbol{H} \\ &\text{such that } \boldsymbol{Y} = \sum_{i=1}^{k} \boldsymbol{D}^{1/2}\boldsymbol{v}_i\boldsymbol{v}_i^\top \boldsymbol{D}^{1/2}, \boldsymbol{H}^\top \boldsymbol{H} = \boldsymbol{I} \end{aligned} \tag{3.2.7}$$

For the k-cluster case, the degree of the matrix $\boldsymbol{D}$ of the graph is obviously not preserved. Therefore, the trick used for 2 clusters in Section 3.2.2 cannot be used here to compute the required eigenvectors with the power method. That said, even though the power method can be used to compute multiple eigenvectors one after the other, there are other more efficient methods to compute more than one eigenvector. So, not being able to use the power method here is not an issue.

Considering how $\boldsymbol{v}_i$ is defined in this more than 2 clusters case, see Equation (3.2.6), it is obvious that no correction is added to the original Laplacian matrix for nodes that are not in the same cluster. That is when two nodes i and j are not in the same cluster, then

$$\left(\sum_{c=1}^{k} \boldsymbol{v}_c\boldsymbol{v}_c^\top\right)_{ij} = 0. \tag{3.2.8}$$

Consequently, in that case, when nodes belong to different clusters, the objective function does not get any incentive to assign these nodes to different clusters. However, when nodes belong to the same cluster, the objective function still gets an incentive to assign them to the same cluster.

3.3 Normalized One-against-All Supervised Spectral Clustering

The above method has some disadvantages. It will not preserve the degree matrix of the graph, consequently algorithms that require that the degrees of the graph are preserved cannot be used. This includes the power method to compute the semi-supervised Fiedler vector. Moreover, supervision will only occur for nodes that are in the same cluster. Therefore, for more than 2 clusters, edges between nodes that are assigned to different clusters will not receive any supervision. We think this is a limitation of this method. In the current section, our goal is to present a method that fixes these two problems. We present NOA-SSC (for normalized one-against-all supervised spectral clustering) which extends [54] to k clusters without the drawbacks of [55]. We will first present [54] very quickly. See Section 3.2.2 for more details. Then we will explain what we cannot do and what we can do to achieve our goal.

In [54], label constraints are introduced by applying a rank-1 update to the similarity matrix of the graph. That is, the supervised similarity matrix is equal to $\widehat{\boldsymbol{W}} = \boldsymbol{W} + \gamma \boldsymbol{D}^{1/2}\boldsymbol{v}_1\boldsymbol{v}_1^\top \boldsymbol{D}^{1/2}$, see Equation (3.2.4). Recall that if ℓ_1 and ℓ_2 correspond to sets of supervised nodes assigned to the cluster 1 and the cluster 2, and $\ell = \ell_1 \cup \ell_2$, the supervision vector $\boldsymbol{v}_1$ is equal to

$$\boldsymbol{v}_1(i) = \sqrt{\frac{\boldsymbol{d}_i}{\mathrm{vol}(\ell)}} y_i \qquad y_i = \begin{cases} \sqrt{\frac{\mathrm{vol}(\ell_2)}{\mathrm{vol}(\ell_1)}} & \text{if } i \in \ell_1 \\ -\sqrt{\frac{\mathrm{vol}(\ell_1)}{\mathrm{vol}(\ell_2)}} & \text{if } i \in \ell_2 \\ 0 & \text{otherwise} \end{cases}$$

When nodes i and j do not belong to the same cluster then the corresponding cell in the matrix $\boldsymbol{v}_1\boldsymbol{v}_1^\top$ is negative and it is positive when i and j have the same label. Moreover, rows and columns of $\boldsymbol{D}^{1/2}\boldsymbol{v}_1\boldsymbol{v}_1^\top\boldsymbol{D}^{1/2}$ sum to 0 so that the diagonal degree matrix of the graph is unchanged. Then, we must find the eigenvector of the supervised Laplacian corresponding to the Fiedler vector, that is the eigenvector corresponding to the second smallest eigenvalue of the unsupervised graph Laplacian. To do so, we can used the power method, as explained in [54] and in Section 3.2.2.

Now consider the case where more than $k > 2$ clusters are involved. Ideally we want to find k cluster indicators, that is k vectors $\boldsymbol{f}_c : 1 \leq c \leq k$ for which positive entries correspond to nodes assigned to cluster c, and negative entries correspond to nodes assigned to other clusters. This is already covered for $k = 2$ by [54]. Since adding $\boldsymbol{D}^{1/2}\boldsymbol{v}_1\boldsymbol{v}_1^\top\boldsymbol{D}^{1/2}$ to the similarity matrix of the graph does not change the degrees of the graph, we may think we can add the supervision $\boldsymbol{v}_1, ..., \boldsymbol{v}_k$ to the similarity matrix all at once:

$$\boldsymbol{Y} = \sum_{c=1}^{k} \boldsymbol{D}^{1/2}\boldsymbol{v}_c\boldsymbol{v}_c^\top\boldsymbol{D}^{1/2}$$

$$\boldsymbol{v}_c(i) = \begin{cases} \sqrt{\frac{\boldsymbol{d}_i}{\mathrm{vol}(\ell_c)}}\sqrt{\frac{\mathrm{vol}(\overline{\ell_c})}{\mathrm{vol}(\ell_c)}} & \text{if } i \in \ell_c \\ -\sqrt{\frac{\boldsymbol{d}_i}{\mathrm{vol}(\ell_c)}}\sqrt{\frac{\mathrm{vol}(\ell_c)}{\mathrm{vol}(\overline{\ell_c})}} & \text{if } i \in \overline{\ell_c} \\ 0 & \text{otherwise} \end{cases}$$

Where $\overline{\ell_c}$ denote the complement of ℓ_c in ℓ, that is: $\overline{\ell_c} = \ell - \ell_c$. The volume of a node set is $\mathrm{vol}(\ell_c) = \sum_{i \in \ell_c} \boldsymbol{d}_i$.

Then, in a similar way to [54] optimize the following problem:

$$\begin{aligned} &\max_{\boldsymbol{f}} \boldsymbol{f}^\top \left(\boldsymbol{D}^{-1/2}(\boldsymbol{W} + \gamma\boldsymbol{Y})\boldsymbol{D}^{-1/2} - 2\boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{I}\right)\boldsymbol{f} \\ &\text{where } \boldsymbol{f} \perp \boldsymbol{D}^{1/2}\boldsymbol{e}, \|\boldsymbol{f}\|^2 = \mathrm{vol}(\mathcal{V}) \end{aligned} \tag{3.3.1}$$

However, it can be shown that in a such case, if the volume of supervision in any cluster c is strictly two times larger than the volume of the remaining supervision, then supervised nodes assigned to other clusters will have a tendency to merge into a single cluster, regardless of the supervision, see Proposition 6.

Proposition 6. *For any cluster c, if $\mathrm{vol}(\ell_c) > 2\mathrm{vol}(\overline{\ell_c})$, then $\boldsymbol{Y}_{ij} > 0$ for every supervised node i and j, even if the supervision for i and j assigns them to different clusters.*

Proof of Proposition 6. Without loss of generality, suppose i belongs to ℓ_1 and j belongs to ℓ_2. Then consider the following update in the cell of the similarity graph corresponding to the edge (i, j):

$$\begin{aligned}
\boldsymbol{Y}_{ij} &= \sum_c \boldsymbol{v}_c(i)\boldsymbol{v}_c(j) \\
&= \boldsymbol{v}_1(i)\boldsymbol{v}_1(j) + \boldsymbol{v}_2(i)\boldsymbol{v}_2(j) + \sum_{c \notin \{1,2\}} \boldsymbol{v}_c(i)\boldsymbol{v}_c(j) \\
&= \frac{\sqrt{\boldsymbol{d}_i\boldsymbol{d}_j}}{\mathrm{vol}(\ell)} \left(-\sqrt{\frac{\mathrm{vol}(\overline{\ell_1})}{\mathrm{vol}(\ell_1)}}\sqrt{\frac{\mathrm{vol}(\ell_1)}{\mathrm{vol}(\overline{\ell_1})}} - \sqrt{\frac{\mathrm{vol}(\overline{\ell_2})}{\mathrm{vol}(\ell_2)}}\sqrt{\frac{\mathrm{vol}(\overline{\ell_2})}{\mathrm{vol}(\ell_2)}} + \sum_{c \neq 1,2} \frac{\mathrm{vol}(\ell_c)}{\mathrm{vol}(\overline{\ell_c})} \right) \\
&= \frac{\sqrt{\boldsymbol{d}_i\boldsymbol{d}_j}}{\mathrm{vol}(\ell)} \left(-2 + \sum_{c \neq 1,2} \frac{\mathrm{vol}(\ell_c)}{\mathrm{vol}(\overline{\ell_c})} \right)
\end{aligned}$$

If $\mathrm{vol}(\ell_c) > 2\mathrm{vol}(\overline{\ell_c})$ for any c, then the update $\boldsymbol{Y}_{ij}$ corresponding to nodes i and j in every cluster $\ell_{c'} \neq \ell_c$ will be positive.

$\square$

To overcome this issue, we decompose the problem in k different problems. We introduce vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ for supervised nodes in $\ell_1, \ldots, \ell_k$ defined as

$$\boldsymbol{v}_c(i) = \begin{cases} \sqrt{\frac{\boldsymbol{d}_i}{\mathrm{vol}(\ell_c)}}\sqrt{\frac{\mathrm{vol}(\overline{\ell_c})}{\mathrm{vol}(\ell_c)}} & \text{if } i \in \ell_c \\ -\sqrt{\frac{\boldsymbol{d}_i}{\mathrm{vol}(\ell_c)}}\sqrt{\frac{\mathrm{vol}(\ell_c)}{\mathrm{vol}(\overline{\ell_c})}} & \text{if } i \in \ell_{c'}, c' \neq c \\ 0 & \text{otherwise} \end{cases} \tag{3.3.2}$$

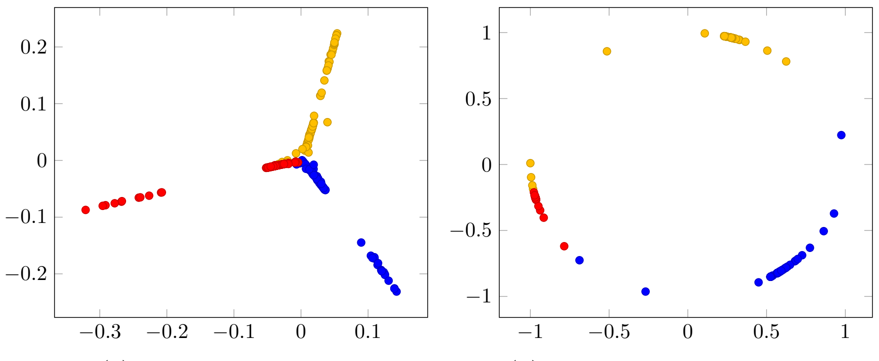
Then, for each cluster c, we optimize problem (3.2.5):

$$\begin{aligned}
&\boldsymbol{g}_c^\star = \arg\max_{\boldsymbol{g}} \boldsymbol{g}^\top \left(\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2} + \gamma\boldsymbol{v}_c\boldsymbol{v}_c^\top - 2\boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{I} \right) \boldsymbol{g} \\
&\text{such that } \boldsymbol{D}^{1/2}\boldsymbol{g} \perp \boldsymbol{1} \text{ and } \boldsymbol{g}^\top\boldsymbol{g} = \mathrm{vol}(\boldsymbol{D})
\end{aligned}$$

Then, we form the $n \times k$ matrix $\boldsymbol{S}$ by normalizing the rows of the matrix obtained by concatenating the vectors $\boldsymbol{g}_1^\star, \ldots, \boldsymbol{g}_k^\star$. Formally, we have

$$\boldsymbol{S}_{ic} = \frac{\boldsymbol{g}_c^\star(i)}{\sqrt{\sum_{c'=1}^{k} \boldsymbol{g}_{c'}^\star(i)^2}} \tag{3.3.3}$$

The introduction of semi-supervision has a potentially problematic effect. The eigenvector entries corresponding to supervised nodes become dominant compared to unsupervised entries. Although the amount of this dominance can be controlled by γ, the core idea behind this kind of algorithm is to pull supervised nodes far apart. Unsupervised nodes also get pulled but with less force. For each cluster, nodes are pulled in a particular direction. Supervised nodes are directly pulled in this direction, while unsupervised nodes are pulled in various directions depending on the structure of the graph. This results in a star shaped eigenspace, in which some clustering algorithm (like k-means) has to be run. However, running a clustering algorithm directly in that eigenspace would lead to bad results. Actually, if we had supervised k clusters, we would get $k+1$ clusters, because unsupervised nodes are more similar to each other than any supervised node. That's why we project data points of the eigenspace onto a unit sphere. That way, only the direction of the pull is preserved. Figure 3.1 illustrates this problem.



(a) No normalization (b) With row normalization

Figure 3.1: Illustration of the effect of normalization of $\boldsymbol{S}$. Only the two most dominant axis of the eigenspace of $\boldsymbol{S}$ are being shown. Without normalization (a), the supervised nodes are pulled far away in different directions, while the remaining (and unsupervised) nodes concentrate around the origin. Projection of the data points onto the unit sphere fix the issue.

Finally we obtain the clusters by running k-MEANS on the rows of $\boldsymbol{S}$. See algorithm 7.

3.4 Experiments with NOA-SSC

In this section, we evaluate NOA-SSC on several datasets. We want to show that adding semi-supervision can actually increase the performance of clustering. We compare NOA-SSC with label propagation (Section 3.1) proposed in [91] and Mavroeidis [54, 55] (Section 3.2). Mavroeidis [54] is used in the 2 clusters case, whereas Mavroeidis [55] is used for more than 2 clusters. We run clustering algorithms on the UCI datasets as well as on real networks. The predicted partitions are compared to the reference partitions using the Adjusted Rand Index (see Section 2.5 and Hubert and Arabie [33]), which yields a value between -1 and $+1$. It indicates how well a given partition conforms to the ground truth: 0 means the given partition is not better than a random assignment; 1 means the given partition matches the ground truth exactly. 10% label constraints are added uniformly at random.

Mavroeidis [55] and our proposition NOA-SSC use k-kmeans as a last step of the algorithm to retrieve a partition. Our implementation of k-means ensures that clusters

```
Algorithm 7: NOA-SSC
   Input: $\boldsymbol{W}$: $n \times n$ similarity matrix, $\boldsymbol{v}_1, ..., \boldsymbol{v}_k$: label constraints as defined in
          Eq. 3.3.2, $\gamma$: parameter controlling the amount of supervision
   Output: $\ell_1, ..., \ell_k$: final clusters
1  begin
2      foreach $c \in 1, ..., k$ do
3          Compute the vector $\boldsymbol{g}_c^\star$ such that
               $$\boldsymbol{g}_c^\star = \begin{array}{l}\arg\max_{\boldsymbol{g}} \boldsymbol{g}^\top \left(\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2} + \gamma \boldsymbol{v}_c\boldsymbol{v}_c^\top - 2\boldsymbol{u}\boldsymbol{u}^\top + \boldsymbol{I}\right)\boldsymbol{g} \\ \text{such that } \boldsymbol{D}^{1/2}\boldsymbol{g} \perp \boldsymbol{1} \text{ and } \boldsymbol{g}^\top\boldsymbol{g} = \mathrm{vol}(\boldsymbol{D})\end{array}$$
4      Form the matrix $\boldsymbol{S}$ such that
               $$\boldsymbol{S}_{ic} = \frac{\boldsymbol{g}_c^\star(i)}{\sqrt{\sum_{c'=1}^{k} \boldsymbol{g}_{c'}^\star(i)^2}}$$
       return $k$-MEANS$(\boldsymbol{S}, k)$
```

will never be empty and uses a smart initialization (See Algorithm 4).

We experimentally observed that the introduction of few labeling constraints can degrade clustering results. Indeed, for many algorithms, label constraints affect the graph Laplacian or the adjacency matrix and then distort the embedded space in which k-MEANS operates. This distortion can have important side effects, therefore a first baseline we consider is to ignore label constraints and run spectral clustering. Spectral clustering consists in running k-MEANS on data points spanned by the rows of the first k eigenvectors of the symmetric normalized Laplacian of the graph ([77]). As other methods presented here require to project the data points onto the unit-sphere before running k-means, we consider spherical spectral clustering (denoted by SPH. SC) which presents the same requirement (see Ng *et al.* [65]).

Table 3.1 reports the average of ARI over experiments repeated 10 times. We see that NOA-SSC clearly outperforms the other methods on most datasets and achieves similar results in the other cases. Mavroeidis [54] and NOA-SSC report the same exact partitions for experiments with only 2 clusters. This is expected as these two methods are equivalent in that case: after computing the same eigenvector $\boldsymbol{g}^\star$

- Mavroeidis [54] partition the graph such that nodes associated with a positive value in $\boldsymbol{g}^\star$ belong to one cluster and other nodes to the other cluster.
- NOA-SSC projects the coordinates of this vector onto the unit-sphere, thus positive coordinates are projected onto 1 and negative coordinates are projected onto -1. Then it applies k-means on this normalized vector.

We can see that these two procedures are equivalent. It is worth noting that in many cases Label Prop. performs worse than Sph. SC, despite the use of supervision. The variance of Label Prop. is 0.063 ± 0.078 on average whereas the variance of noa-ssc is 0.50 ± 0.037 on average. This is quite surprising given that noa-ssc is non-deterministic, because it uses k-means as a last step to produce a partition, which is not the case for Label Prop. where a deterministic vote is used. Thus the variance of Label Prop. exclusively originates in the random selection of supervised nodes. This suggests that noa-ssc might be more robust than Label Prop. to the randomness introduced by choosing the supervised nodes. Comparatively, the results obtained by Mavroeidis [55] have a variance comparable to that of noa-ssc, however the latter outperforms the former on many datasets.

Dataset	k	Sph. SC	Label Prop.	Mavro. 2010/11	noa-ssc
hepatitis	2	**0.13 ± 0.00**	0.06 ± 0.07	**0.12 ± 0.11**	**0.12 ± 0.11**
ionosphere	2	0.05 ± 0.00	0.06 ± 0.01	**0.26 ± 0.13**	**0.26 ± 0.13**
moons	2	0.07 ± 0.00	0.12 ± 0.13	**0.27 ± 0.03**	**0.27 ± 0.03**
promoters	2	0.25 ± 0.00	0.01 ± 0.00	**0.31 ± 0.09**	**0.31 ± 0.09**
spam	2	0.02 ± 0.00	0.05 ± 0.00	**0.98 ± 0.01**	**0.98 ± 0.01**
tic-tac-toe	2	-0.00 ± 0.00	0.06 ± 0.01	**0.22 ± 0.05**	**0.22 ± 0.05**
wdbc	2	0.71 ± 0.00	0.05 ± 0.01	**0.74 ± 0.02**	**0.74 ± 0.02**
xor	2	-0.00 ± 0.00	0.18 ± 0.31	**0.96 ± 0.11**	**0.96 ± 0.11**
hayes-roth	3	-0.01 ± 0.00	**0.03 ± 0.02**	-0.01 ± 0.00	**0.06 ± 0.05**
interlaced-circles	3	0.12 ± 0.01	0.06 ± 0.08	0.11 ± 0.01	**0.16 ± 0.02**
iris	3	**0.64 ± 0.01**	0.08 ± 0.17	0.54 ± 0.03	0.58 ± 0.03
wine	3	**0.91 ± 0.00**	0.06 ± 0.15	0.81 ± 0.02	0.68 ± 0.08
imdb	4	0.01 ± 0.01	0.14 ± 0.04	0.00 ± 0.00	**0.28 ± 0.02**
vehicles	4	0.05 ± 0.00	0.04 ± 0.04	0.04 ± 0.00	**0.12 ± 0.01**
phoneme	5	0.07 ± 0.00	0.01 ± 0.00	0.03 ± 0.00	**0.69 ± 0.01**
webkb-cornell	5	**0.30 ± 0.03**	0.06 ± 0.05	**0.32 ± 0.05**	**0.34 ± 0.07**
webkb-texas	5	**0.22 ± 0.04**	0.07 ± 0.04	**0.24 ± 0.04**	**0.27 ± 0.06**
webkb-wisconsin	5	0.38 ± 0.03	0.04 ± 0.01	0.30 ± 0.05	**0.46 ± 0.03**
breasttissue	6	0.33 ± 0.03	0.01 ± 0.03	0.21 ± 0.03	**0.40 ± 0.04**
glass	6	**0.18 ± 0.01**	0.02 ± 0.01	0.14 ± 0.02	**0.22 ± 0.06**
zoo	7	**0.61 ± 0.09**	0.10 ± 0.14	0.53 ± 0.08	**0.69 ± 0.13**

Table 3.1: noa-ssc results reported using Adjuted Rand Index (see Section 2.5). Boldfaces are used whenever an algorithm outperforms other algorithms. Variance is taken into account to determine the best performances. Label constraints represent 10% of the nodes, selected uniformly at random.

In addition to giving good performance, noa-ssc also appears to be robust to different parameter settings. As explained in Section 2.1.2, building an adapted graph representation from vectorial datasets relies on the choice of a distance, usually parametrized. We study the stability of clustering approaches with respect to the choice of these parameters. For a given dataset, we fix a percentage of supervision and we vary the number k of neighbors in the kNN graph construction step. We measure the ARI of spectral clustering (SPH. SC), semi- supervised node classification (LABEL PROP.), Mavroeidis [54, 55] and our algorithm NOA-SSC. Figure 3.2 gives the curves of the mean over 10 experiments over the selection of supervised nodes for the zoo dataset.

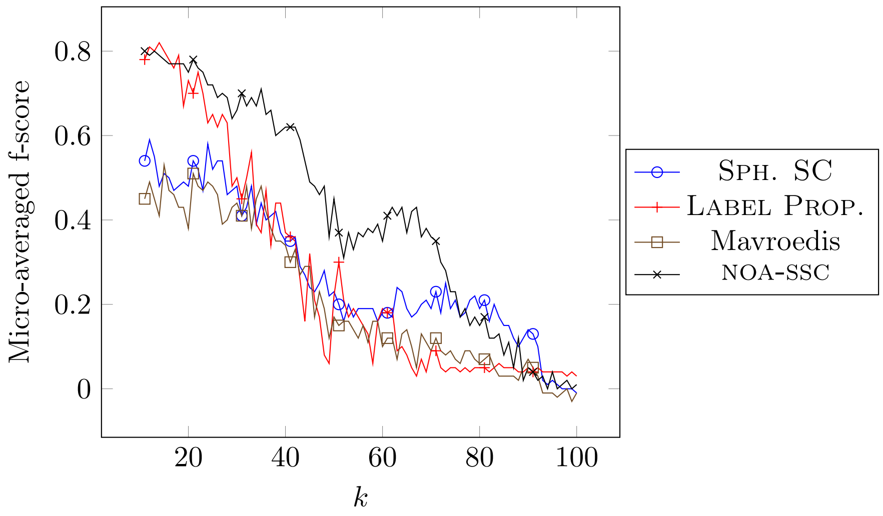


Figure 3.2: Evolution of ARI w.r.t. the parameter k in the k-nearest neighbor step used to build graphs for the dataset zoo. We can see that NOA-SSC is much less sensitive to parameter tuning than other algorithms. In the case of harmonic Laplacian, where a greater k degrades quickly the quality of the clustering.

In this experiment, both LABEL PROP. and NOA-SSC starts with a pretty good performance. Then, as the number k of nearest neighbors increases, the performances degrade. However, NOA-SSC degrades less quickly than other methods. An explanation for this is that the random walk implicitly performed by LABEL PROP. can "get lost" as the graph gets more and more connected [81]. Our approach NOA-SSC appears to be more stable with respect to the choice of k. Additionally, note that the hyperparameter γ in NOA-SSC was not tuned: we simply set it to 1.25, which is a reasonable value according to [54]. Tuning γ is likely to give better results.

Another parameter is the number of constraints. The Figure 3.3 shows the evolution of Adjusted Rand Index with respect to the number of label constraints. As this number increases, we can see that NOA-SSC improves the results, whereas it is not necessarily the case for other algorithms. Label propagation actually degrades the results when the number of constraints is low, but eventually reports better results than NOA-SSC when the number of constraints is unrealistically large. Increasing the number of constraints with Mavroeidis [55] can degrade the results on datasets for which unsupervised clustering already reports a good score, as shown with the 'zoo' dataset.

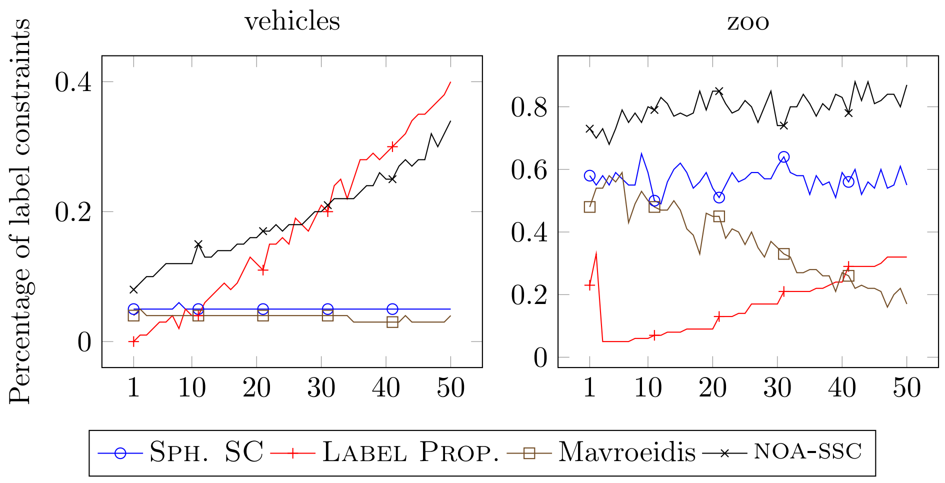


Figure 3.3: Evolution of ARI w.r.t. the number of constraints for the datasets vehicles and zoo. Vehicles is a dataset for which unsupervised clustering fails to report a good result, whereas unsupervised clustering on zoo reports a decent score. On both datasets the introduction of label constraints using NOA-SSC improves the scores. Label propagation degrades the results when the number of constraints is low. The proposal Mavroeidis [55] actually degrades the results as the number of constraints increases. The variation on Sph. SC is due to k-means.

3.5 Conclusion

In this chapter, we have reviewed and proposed methods for semi-supervised clustering that are not restricted to binary clustering problems and can handle label constraints. Experiments carried out on a large variety of datasets, including both vectorial and real network data, indicate that our algorithms outperform state of the art unsupervised and semi-supervised systems in most cases. We have shown that our algorithm is much less sensitive to parameter tuning than the semi-supervised harmonic Laplacian or Mavroeidis [55].

Chapter 4

Pairwise constraints

Contents

In this chapter, we consider the setting wherein semi-supervision takes the form of *pairwise constraints*, whereby two nodes are assigned to identical (*must-link*) or different clusters (*cannot-link*), regardless of the clusters labels. This setting is arguably more realistic when annotation is costly. Not all pairs of nodes are going to be supervised, but we would like to take advantage of both supervised and unsupervised nodes in the learning process. Again, we can make use of at least one of the semi-supervised learning assumptions:

Smoothness assumption Nodes which are close to each other are more likely to be in the same cluster.

Cluster assumption Clusters tend to be dense and well-separated.

Manifold assumption The data lie approximatively on a manifold whose dimension is much lower than the input space.

However, these assumptions are not always met. For example, clusters are not always dense and well-separated. They can be split in different parts. Nodes which are close are not necessarily in the same cluster either. The manifold assumption can also not hold when the data is particularly noisy. In all these cases, pairwise constraints can be introduced to correct the data set. Must-link constraints will be used to merge different parts of the data set together in the same cluster, while cannot-link will be used to separate parts of the data sets.

Pairwise constraints can always be obtained from label constraints. To do so, it suffices to create a must-link for all pairs of nodes that are assigned to the same label and a cannot-link for all pairs of nodes that are assigned to different labels.

The converse, retrieving label constraints from pairwise constraints requires that we make some assumptions. If all the must-link constraints are correct and uniformly distributed among the pairs of nodes that belong to the same cluster, then if, for every cluster i, at least $\binom{n_i}{2}(1+\epsilon)\log(n_i)/n_i$ pairs of nodes are constrained, where ϵ is an arbitrary positive value, n_i is the number of nodes in cluster i, then each cluster will be almost surely connected in the graph of must-link constraints [25]. This is a property of the Erdös-Rényi random graphs. In that case, a simple transitive closure on the must-links will almost surely return a graph with k cliques corresponding to k clusters. This is something to keep in mind when designing algorithms or experiments that handle pairwise constraints to avoid trivial problems. Indeed, we are not interested in experiments consisting of graphs fully partitionable using only a transitive closure of the must-link constraints, as this does not make use of semi-supervised learning assumptions: in that case, only supervised edges are used to recover a partition.

Also note that, in some problems, constraints may be inconsistent. This can happen when constraints are obtained automatically using some heuristics on the data set. To picture this, imagine that you have a set of objects and pairs of these objects have attributes. Then you observe empirically that when some attribute has a particular value, most of the time it corresponds to a must-link. It makes sense to produce must-links for these cases, as they will be mostly correct. However, there is a chance that some of them will be incorrect. In such cases, a transitive closure will return a bad partition, as the bad must-links will merge different clusters together.

Satisfying all the cannot-link constraints is NP-complete for $k > 2$. This uses a straightforward reduction from the graph k-coloring problem, see the appendix of Davidson and Ravi [19]. Unlike must-link constraints, cannot-link constraints are not transitive. When a node a cannot be linked with a node b, and b cannot be linked with c, what can we conclude about a and c ? If it is known that there is only 2 clusters, then a and c must link. However, if the number of clusters is unknown or larger than 2, then nothing can be said about the pair (a, c).

In Section 4.1, we will present a first contribution that extends NOA-SSC (See Section 3.3) to pairwise constraints. As the treatment of cannot-links in this approach is somehow similar to that of Kamvar *et al.* [37], we will begin this discussion with a short review of [37]. We will also discuss [39] that introduces pairwise constraints by modifying the similarity matrix of the graph. We will discuss the limitations of these approaches. In Section 4.2.2, we will present a first method [48] that handles some of the problems previously encountered. We will continue in Section 4.3 with Li and Liu [44] and our contribution on pairwise constraints: Chatel *et al.* [14]. Both of them are based on a nice geometrical interpretation of spectral clustering with pairwise constraints. In Section 4.4, we will finally present two methods that take a different approach: Rangapuram and Hein [74] and Cucuringu *et al.* [18].

4.1 First steps into pairwise constraints

In this section, we will discuss about some of the simplest approaches to introduce pairwise constraints into spectral clustering. Kamvar *et al.* [37], detailed in Section 4.1.1, is probably the simplest approach to spectral clustering with pairwise constraints. This approach consists in introducing or removing edges corresponding to the pairs involved in the pairwise constraints directly into the graph. We suspect that using this approach, constraints will not propagate well to the unsupervised nodes. A second approach, detailed in Section 4.1.2, is our first contribution on pairwise constraints. It takes advantage of the transitive property of the must-link constraints in order to propagate the must-link and cannot-link constraints to additional edges of the graph.

4.1.1 Spectral Learning

Spectral learning is an early method proposed by Kamvar *et al.* [37]. This is the simplest method to integrate pairwise constraints into spectral clustering. It relies on updating the similarity matrix $\boldsymbol{W}$ by setting $\boldsymbol{W}_{ij} = 1$ for all must-links and $\boldsymbol{W}_{ij} = 0$ for all cannot-links. Then, it consists in running k-MEANS or any other vector-based clustering algorithm on the top k largest eigenvectors of the following normalized Markov transition process:

$$\boldsymbol{N} = \frac{\boldsymbol{W} + \boldsymbol{d}_{\max}\boldsymbol{I} - \boldsymbol{D}}{\boldsymbol{d}_{\max}}, \tag{4.1.1}$$

where $\boldsymbol{d}_{\max}$ is the maximum degree of the graph. Notice that if $\boldsymbol{v}$ is an eigenvector for $\boldsymbol{N}$ with eigenvalue σ, then $\boldsymbol{v}$ is also an eigenvector for the Laplacian matrix of the graph $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$ with eigenvalue $\lambda = -\boldsymbol{d}_{\max}(\sigma - 1)$:

$$\begin{aligned}
\frac{\boldsymbol{W} + \boldsymbol{d}_{\max}\boldsymbol{I} - \boldsymbol{D}}{\boldsymbol{d}_{\max}}\boldsymbol{v} &= \sigma\boldsymbol{v} \\
\left(\boldsymbol{I} - \frac{\boldsymbol{D} - \boldsymbol{W}}{\boldsymbol{d}_{\max}}\right)\boldsymbol{v} &= \sigma\boldsymbol{v} \\
\boldsymbol{v} - \frac{1}{\boldsymbol{d}_{\max}}(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{v} &= \sigma\boldsymbol{v} \\
-\frac{1}{\boldsymbol{d}_{\max}}(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{v} &= (\sigma - 1)\boldsymbol{v} \\
\boldsymbol{L}\boldsymbol{v} &= -\boldsymbol{d}_{\max}(\sigma - 1)\boldsymbol{v}
\end{aligned}$$

So we can conclude that this method simply consists in running a graph cut algorithm after setting similarities corresponding to must-links and cannot-links to 1 and 0 respectively. Therefore, constraints do not propagate to unsupervised edges, thus the algorithm may require a large number of constraints to yield good performance.

The graph in Figure 4.1 shows why pairwise constraints propagation is essential to semi-supervised spectral clustering algorithms. Introducing a must-link between two regions of the graph that are sparsely connected will not have a sufficient effect to merge the two regions in a single cluster. Conversely, introducing a cannot-link into a dense region of the graph will not have a sufficient effect either to separate the region into different clusters, because there will remain a lot of paths connecting the cannot-linked nodes.

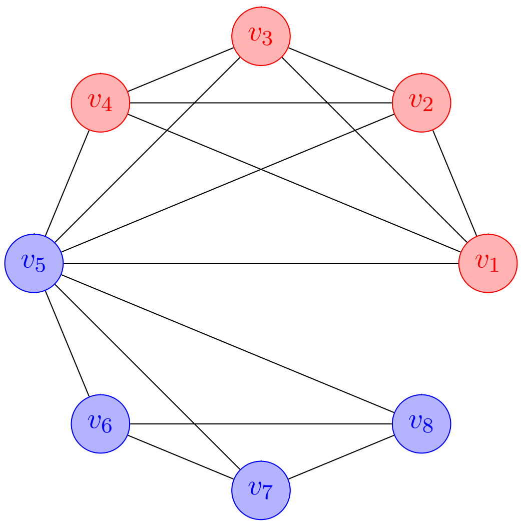


Figure 4.1: This toy example shows why the propagation of constraints to unconstraint edges is essential in semi-supervised spectral clustering algorithms. The node v_5 is in the blue class, however it is part of both cliques of the graph. Removing the edge (v_1, v_5) will have very little effect as nodes $v_2, \ldots, v_4$ will still strongly connect to the red cluster.

4.1.2 NOA-SSC: Extension to pairwise constraints

In this section, we develop an original method in order to find a good translation of must-link and cannot-link constraints into labeling constraints. Thereafter, we use NOA-SSC, a k-clustering algorithm with soft label constraints, in order to refine the labels found in the first step. As we assume that must-link constraints are correct, we can obtain label constraints for the subset of nodes supervised by the must-links using a simple transitive closure. This can be done in polynomial time. However, cannot-link constraints cannot be satisfied in polynomial time as soon as we are looking for more than 2 clusters. Therefore we will treat them separately by setting the corresponding similarities to zero. This approach for cannot-links have already been used in Kamvar *et al.* [37]. We have seen in Section 4.1.1, that this kind of approach for cannot-link constraints may require a lot of constraints to have an effect. In our present setup, cannot-link constraints are somewhat propagated to other pairs of nodes, through the transitive closure used on the must-link constraints.

Constraints are given by two $n \times n$ matrices $\boldsymbol{M}$ and $\boldsymbol{C}$:

$$\boldsymbol{M}_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ must link} \\ 0 & \text{otherwise} \end{cases} \qquad \boldsymbol{C}_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ cannot link} \\ 0 & \text{otherwise} \end{cases}$$

We first compute the transitive closure of must-link constraints. This can be done using the Warshall algorithm [83], which returns the transitive closure of a $n \times n$ boolean matrix in $O(n^3)$.

Computing the transitive closure of the must-link constraints leads to a partition of nodes into m components that can be represented by a $n \times m$ matrix $\boldsymbol{P}$:

$$\boldsymbol{P}_{if} = \begin{cases} 1 & \text{node } i \text{ belongs to component } f \\ 0 & \text{otherwise} \end{cases} \tag{4.1.2}$$

These connected components are in effect “super nodes” of the graph, gathering nodes that must be linked together. Their number m can be greater than the required number of clusters k. In this case we need to merge connected components until we obtain k clusters. Notice $(\boldsymbol{P}^\top \boldsymbol{W} \boldsymbol{P})_{ij}$ is the sum of the edge weights between the connected components i and j. And $(\boldsymbol{P}^\top \boldsymbol{C} \boldsymbol{P})_{ij}$ is the number of violated constraints when trying to merge the connected components i and j. Figure 4.2 shows an illustration of these computations.

We propose a first algorithm, Algorithm 8, that only handles must-link constraints. In order to merge connected components obtained by the transitive closure of must-link constraints, we approximate normalized cut on the graph of components represented by similarity matrix $\boldsymbol{W}^m = \boldsymbol{P}^\top \boldsymbol{W} \boldsymbol{P}$. We compute the $c \times k$ indicator matrix $\boldsymbol{Q}$ for components:

$$\boldsymbol{Q}_{cc'} = \begin{cases} 1 & \text{if component } c \text{ belongs to cluster } c' \\ 0 & \text{otherwise} \end{cases}$$

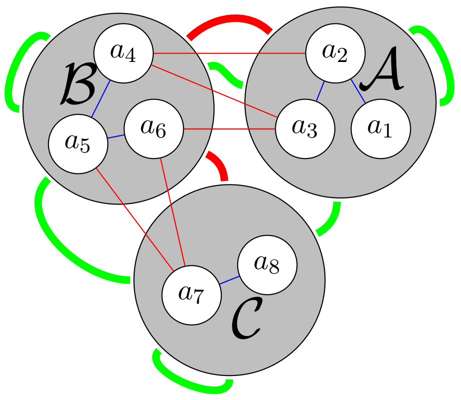


Figure 4.2: Example of a graph showing must-link constraints (blue edges) and cannot-link constraints (red edges). The connected components $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ are given by the matrix $\boldsymbol{P}$. We can see that only cannot-link edges remain between the connected components. Thick red edges weights are equal to the number of violated cannot-link constraints, this is given from the computation $\boldsymbol{P}^\top \boldsymbol{C}\boldsymbol{P}$. Thick green edges weights are equal to the volume of the edges, in the original graph, coming from one connected component to another (or remaining in the same, in case of self-loops), and are given from the computation $\boldsymbol{P}^\top \boldsymbol{W}\boldsymbol{P}$.

Algorithm 8: ML-NOA-SSC

```
  Input: $W$: $n \times n$ similarity matrix, $M$: $n \times n$ must-link constraint matrix
1 begin
2     Compute the transitive closure of $M$
3     foreach node $i$ and connected component $f$ do
4         $P_{if} = 1$ if $i$ belongs to component $f$ and 0 otherwise
5     $W^m = P^\top W P$
6     Run normalized cut on the similarity matrix $W^m$
7     foreach connected component $c$ and cluster $f$ do
8         $Q_{fc} = 1$ if component $c$ belongs to cluster $f$, 0 otherwise
9     return NOA-SSC on $W$ using $PQ$ as label constraints
```

such that $\boldsymbol{Q}$ optimizes normalized cut with similarity matrix $\boldsymbol{W}^m$. The $n \times k$ matrix $\boldsymbol{PQ}$ corresponds to the cluster indicator for the original nodes. We could simply return $\boldsymbol{PQ}$ as our final clustering, but this would amount to treating must-link constraints as hard constraints, which would lead to bad performance if supervision is noisy. In order to relax the constraints, we apply NOA-SSC on the initial graph but with labeled constraints obtained from the clustering of $\boldsymbol{W}^m$. We use these labels (over all nodes) to constraint the spectral clustering on $\boldsymbol{W}$ with the algorithm NOA-SSC.

We also propose Algorithm 9, a refinement of Algorithm 8 to handle cannot-link constraints. Inspired by Kamvar *et al.* [37], see Section 4.1.1, we simply set the entries in the matrix $\boldsymbol{W}^m$ corresponding to cannot-links to zero in order to obtain the labels. However, in our case, the cannot-link constraints are propagated through the must-link constraints.

Algorithm 9: MCL-NOA-SSC

```
Input: $\boldsymbol{W}$: $n \times n$ similarity matrix, $\boldsymbol{M}$: $n \times n$ must-link constraint matrix, $\boldsymbol{C}$:
       $n \times n$ cannot-link constraint matrix, $k$: number of clusters
Output: $\ell_1, ..., \ell_k$: final clusters.
1  begin
2      Compute the transitive closure of $\boldsymbol{M}$
3      foreach node $i$ and connected component $f$ do
4          $\boldsymbol{P}_{if} = 1$ if $i$ belongs to component $f$ and 0 otherwise
5      $\boldsymbol{W}^m = \boldsymbol{P}^\top \boldsymbol{W} \boldsymbol{P}$
6      foreach pair of connected component $(f, f')$ do
7          if any node in $f$ is cannot-linked with any node in $f'$ then
8              Set $\boldsymbol{W}^m_{ff'} = 0$
9      Run normalized cut on the similarity matrix $\boldsymbol{W}^m$
10     foreach connected component $c$ and cluster $f$ do
11         $\boldsymbol{Q}_{fc} = 1$ if component $c$ belongs to cluster $f$, 0 otherwise
12     return NOA-SSC on $\boldsymbol{W}$ using $\boldsymbol{PQ}$ as label constraints
```

4.1.3 First experiments on pairwise constraints

In this section, we compare Spectral Learning algorithm (Section 4.1.1, [37]) with MCL-NOA-SSC. We conjecture that the propagation of pairwise constraints to unsupervised nodes is important and is lacking in the Spectral Learning algorithm. We also suspect that the transitive closure can drastically improve the results, as well as retrieve the full clustering by itself in some cases. In particular, one of the results in the Erdös-Rényi's theory of random graphs is interesting here: graphs generated at random by adding an edge between each pair of nodes independently with probability $p = (1 + \epsilon) \log(n)/n$, where n is the number of nodes, is almost certainly a connected graph. A direct consequence for clustering with pairwise constraints is that for must-link constraints generated

Partition	Cluster sizes				
A	100	100	100	100	100
B	144	125	100	75	56
C	175	150	100	50	25
D	250	125	63	31	31

Table 4.1: Cluster sizes for partitions used in the experiment involving the transitive closure

uniformly and independently at random there is a threshold above which a simple transitive closure is very likely to solve the clustering problem.

A first experiment consists in generating partitions of different sizes (see Table 4.1), as well as must-link constraints and then apply a transitive closure on the constraints to obtain a new partition to compare with the original partition. In this experiment, the cluster sizes in the partition have been manually chosen in order to show increasing levels of imbalance. Dataset A is balanced. For datasets B and C some of the nodes in the right clusters have been moved to the left clusters. For dataset D, the leftmost cluster is twice the size of the next cluster, and so on, except for the last two clusters. The results are shown in Figure 4.3.

We can see that the scores reported in ARI for all the datasets get close to 1 around 1000 must-link constraints. This corresponds to theory, as the average number of nodes in any cluster in this experiment is 100 and the minimal number of edges to obtain a connected graph with 100 nodes is $\binom{100}{2}(1+\epsilon)\log(100)/100 \approxeq 228$. Then, we have 5 clusters, so we'd need about $228 \times 5 = 1140$ must-links generated uniformly at random from an oracle to be almost certain that the correct partition would be recovered. That being said, we can also see in the lower part of the figure, the scores reported as NMI. Unlike ARI, NMI does not adjust itself according to the expected score obtained from a random partition. While this absence of adjustment would make it harder to compare results from different algorithms, in this particular case, NMI allows us to have a better understanding of the experiment. In that part of Figure 4.3, we see that overall, dataset A performs better than B, while C and D report the worst results. The explanation lies in the fact that dataset A is comprised of balanced clusters, while subsequent datasets are comprised of more and more imbalanced clusters. In this setup, must-link constraints are generated uniformly and independently at random among all possible must-links of the graph, regardless of the cluster membership of nodes. Thus, with a cluster imbalanced dataset, the probability to supervise smaller clusters is smaller than the probability to supervise bigger clusters. The consequence is what we see in the lower part of Figure 4.3: imbalanced clusters are inherently harder to recover than balanced clusters.

That being said, obtaining uniformly distributed constraints is not always possible. The particular case of generating uniformly distributed constraints using an oracle is rather unrealistic in practice. As the result about the connectivity of a random graph applies only when edges are uniformly distributed, we may ask what happens when constraints are dependent on each other. Domain knowledge can be helpful to generate

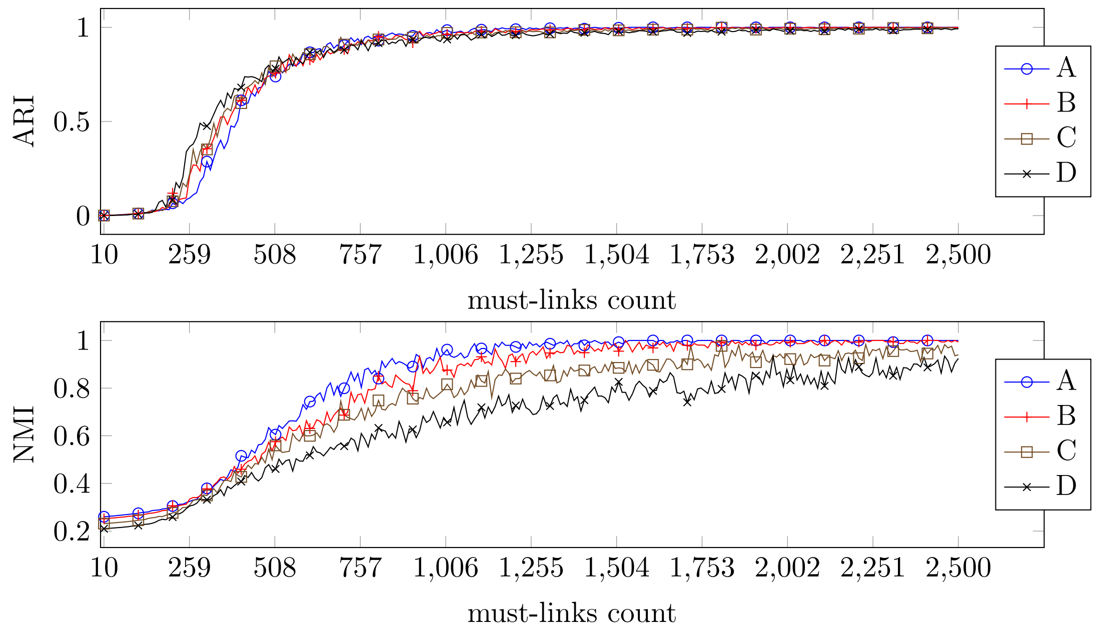


Figure 4.3: Evolution of ARI and NMI with respect to the number of must-link constraints, over 4 different partitions. Partition A has balanced cluster sizes, subsequent partitions have more and more imbalanced clusters. We can see that the number of must-link constraints required to recover the partition corresponds to the number given by Erdös-Rényi's theory of random graphs, as well as imbalanced partitions are harder to recover.

constraints automatically without the need for an oracle. Of course, generated constraints can be incorrect depending on the generating process. However, what does happen when all the must-link constraints are correct, but not uniformly distributed ? To answer this question, we design a new experiment around the same 4 partitions used in our first experiment. This time, we set the number of constraints to the threshold at which the different clusters in the different graphs would form connected components, according to the Erdös-Rényi's theory of random graph. Then, we generate must-links using the following process: the first must-link is chosen at random among possible correct must-links. Then, subsequent must-links are chosen among the remaining possible correct must-links with probability α, otherwise they are chosen among the remaining possible correct must-links that share a node with an already chosen must-link. In this setup, we call α "uniformity of must-links". Results are shown in Figure 4.4.

As previously, the cluster sizes of the dataset A are balanced, while subsequent datasets have increasingly imbalanced cluster sizes. We can see that the uniformity of the must-links greatly impacts the clustering results, this is particularly true for datasets whose cluster sizes are imbalanced. Non-uniformity degrades the clustering results, because uniformity in the distribution of must-links guarantees a certain level of diversity in the supervised nodes. The more uniformity decreases, the more the number of unsu-

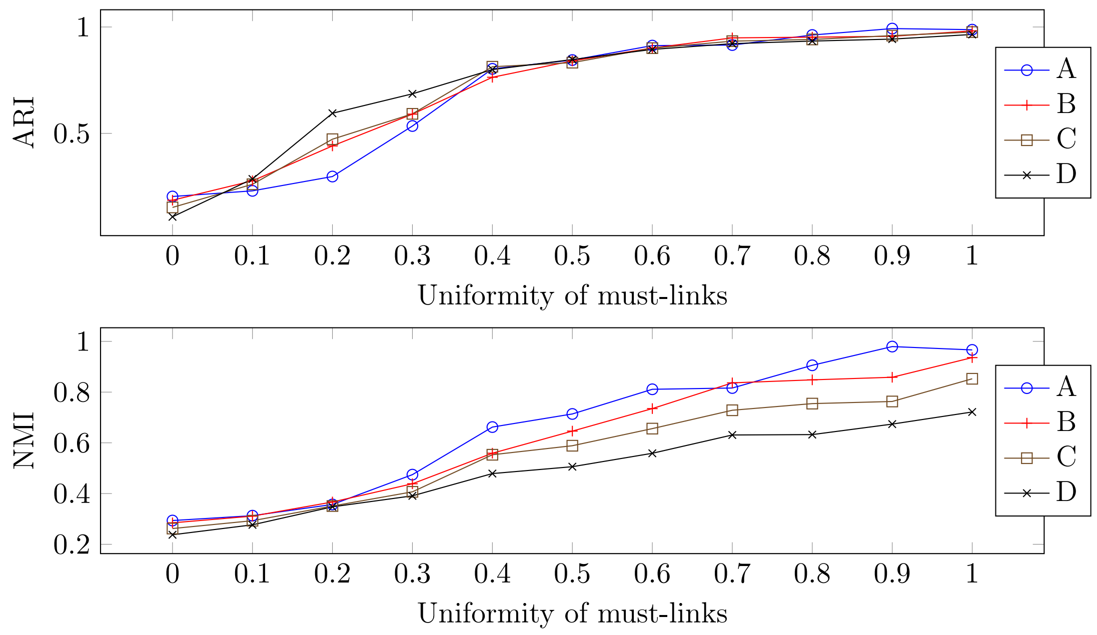


Figure 4.4: Evolution of ARI and NMI with respect to the uniformity of the distribution of must-link constraints, over 4 different partitions. Partition A has balanced cluster sizes, subsequent partitions have more and more imbalanced clusters. The x-axis represents the uniformity of the distribution of must-link constraints. Uniformity 0 means generated must-links are non-uniform, while 1 means generated must-links are completely uniform, so that the must-link graph corresponds to an Erdös-Rényi random graph.

pervised nodes is large. The same set of nodes gets involved in most of the must-link constraints.

Obtaining constraints from domain knowledge can be possible whenever something helpful is known about the nodes or the pairs of nodes. A must-link or a cannot-link constraint can be deduced. It is also possible to use some heuristics to obtain soft constraints: “should-probably-link” and “should-probably-not-link”. For convenience, we will specify when a constraint is soft or not, but we will still talk about must and cannot-link. Regarding the transitive closure, it is obvious that if a single must-link is incorrect, it will suffice to degrade the clustering results by an amount that depends on the size of the connected components involved with the incorrect constraint.

The second experiment consists in comparing results obtained with Spectral Learning algorithm (Section 4.1.1, [37]) and MCL-NOA-SSC. For this experiment, we reuse the datasets used in Section 3.4. We obtain 1% of the pairwise constraints from an oracle uniformly at random. Results are reported in Table 4.2.

As expected, Spectral learning algorithm does not really take advantage of semi-supervision when there are only few constraints. By contrast, MCL-NOA-SSC achieves better results, especially when the number of clusters increases. We can also see that in some cases MCL-NOA-SSC does not provide a significant improvement over spectral learn-

Dataset	k	Spectral Learning	MCL-NOA-SSC
hepatitis	2	0.01 ± 0.00	$\mathbf{0.21} \pm \mathbf{0.00}$
ionosphere	2	0.00 ± 0.00	0.00 ± 0.00
moons	2	0.06 ± 0.00	$\mathbf{0.16} \pm \mathbf{0.00}$
promoters	2	-0.00 ± 0.00	0.00 ± 0.00
spam	2	0.00 ± 0.00	0.00 ± 0.00
tic-tac-toe	2	0.03 ± 0.03	0.00 ± 0.00
wdbc	2	0.00 ± 0.00	$\mathbf{0.86} \pm \mathbf{0.00}$
xor	2	-0.00 ± 0.00	0.00 ± 0.00
wine	3	0.05 ± 0.14	$\mathbf{0.95} \pm \mathbf{0.00}$
hayes-roth	3	-0.01 ± 0.00	-0.01 ± 0.00
interlaced-circles	3	0.13 ± 0.01	0.18 ± 0.00
iris	3	0.57 ± 0.00	$\mathbf{0.72} \pm \mathbf{0.00}$
imdb	4	-0.00 ± 0.00	0.00 ± 0.00
vehicles	4	0.00 ± 0.00	0.06 ± 0.01
phoneme	5	-0.00 ± 0.00	$\mathbf{0.82} \pm \mathbf{0.00}$
webkb-cornell	5	0.01 ± 0.00	$\mathbf{0.53} \pm \mathbf{0.01}$
webkb-texas	5	-0.02 ± 0.00	$\mathbf{0.30} \pm \mathbf{0.00}$
webkb-wisconsin	5	0.01 ± 0.00	$\mathbf{0.39} \pm \mathbf{0.03}$
breasttissue	6	-0.01 ± 0.00	$\mathbf{0.25} \pm \mathbf{0.01}$
glass	6	0.01 ± 0.00	$\mathbf{0.18} \pm \mathbf{0.03}$
zoo	7	0.14 ± 0.03	$\mathbf{0.68} \pm \mathbf{0.00}$

Table 4.2: NOA-SSC results reported using Adjuted Rand Index (see Section 2.5). Boldfaces are used whenever an algorithm outperforms other algorithms. Variance is taken in account to determine the best performances. Pairwise constraints represent 1% of the edges, selected uniformly at random.

ing. This is the case for datasets spam, tic-tac-toe, interlaced-circles and vehicles. The quality of the clustering given by unsupervised normalized cut depends on the closeness between the span of the smallest eigenvectors of the unsupervised Laplacian of the graph and the span given by the reference cluster indicators, see Peng *et al.* [67]. When the smallest eigenvectors span a space which is too far from the span given by the reference cluster indicators, spectral learning and MCL-NOA-SSC does not seem capable to return the correct clustering, because these algorithms look for a solution which is very close to the unsupervised solution.

4.2 Tuning the similarity matrix

In this section, we discuss two approaches that modify the similarity matrix of the graph according to pairwise constraints in order to achieve better clusterings. The first one Kulis *et al.* [39] introduces pairwise constraints into weighted kernel k-means by modifying the kernel matrix. In the case of graphs, the kernel is given by the similarity matrix. This approach also introduces the notion of cluster-size weighted penalties: the penalty or reward for violating or satisfying a constraint is higher if the corresponding cluster is small. We hope that this approach can reduce the degradation effect occurring with datasets whose cluster sizes are imbalanced. We also present Lu and Carreira-Perpinán [48], which is an approach that explicitly propagates must- and cannot-link constraints to other edges of the graph using a Gaussian process.

4.2.1 Semi-supervised Graph Clustering: A Kernel Approach

In this section, we present Kulis *et al.* [39]. This method exploits a weighted kernel version of k-MEANS, denoted as W-K-k-MEANS in this thesis, used with a kernel composed of both the similarity matrix and a matrix of penalities for violating constraints. The main tool, W-K-k-MEANS, for this approach is a special implementation of k-MEANS, designed to operate with a kernel, instead of inner products, to compute distances between data points.

First, we explain W-K-k-MEANS. The objective function for k-MEANS is the following:

$$\sum_{c=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{A}_c} \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2 \quad \text{where } \boldsymbol{\mu}_c = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{A}_c} \boldsymbol{x}_i}{|\mathcal{A}_c|}, \tag{4.2.1}$$

where $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are the k clusters containing data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and μ_c is the centroid of the cluster $\mathcal{A}_c$. The objective function for W-K-k-MEANS is

$$\sum_{c=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{A}_c} \boldsymbol{\alpha}_i \|\phi(\boldsymbol{x}_i) - \mu_c\|^2 \quad \text{where } \boldsymbol{\mu}_c = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{A}_c} \boldsymbol{\alpha}_i \phi(\boldsymbol{x}_i)}{\sum_{\boldsymbol{x}_i \in \mathcal{A}_c} \boldsymbol{\alpha}_i}, \tag{4.2.2}$$

with $\boldsymbol{\alpha}_i$ a weight parameter for the data point $\boldsymbol{x}_i$ and ϕ is a function mapping data points in another space. Expanding the distance computation gives the following:

$$\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i) - 2\frac{\sum_{\boldsymbol{x}_j \in \mathcal{A}_c} \boldsymbol{\alpha}_j \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)}{\sum_{\boldsymbol{x}_j \in \mathcal{A}_c} \boldsymbol{\alpha}_j} + \frac{\sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in \mathcal{A}_c} \boldsymbol{\alpha}_j \boldsymbol{\alpha}_l \phi(\boldsymbol{x}_j)\phi(\boldsymbol{x}_l)}{\left(\sum_{\boldsymbol{x}_j \in \mathcal{A}_c} \boldsymbol{\alpha}_j\right)^2}. \tag{4.2.3}$$

Then we can substitute $\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)$ with any positive semidefinite matrix $\boldsymbol{K}$ and use this directly in W-K-k-MEANS algorithm (see Algorithm 10), without even knowing ϕ or $\boldsymbol{x}_i$.

Once this is set, the authors explain how different graph-based problems can be expressed and solved with W-K-k-MEANS, see Table 4.3. For more details, see Dhillon *et al.* [22].

The "kernels" used to solve these problems are not necessarily positive semidefinite. So, these matrices are not really kernels, since by definition kernels are positive semi-definite matrices. Positive semi-definiteness is very important in algorithms based on W-K-k-MEANS, as without this property, the algorithm is not guaranteed to converge. Indeed

Algorithm 10: W-K-k-MEANS

```
Input: Kernel matrix $\boldsymbol{K}$, cluster count $k$, weight vector $\boldsymbol{\alpha}$, optional initial
       clusters $\{\mathcal{A}_c^{(0)}\}_{c=1}^k$.
Output: Final partitioning $\{\mathcal{A}_c\}_{c=1}^k$
1 begin
2     Initialize $k$ clusters from $\{\mathcal{A}_c^{(0)}\}_{c=1}^k$ if provided, else randomly
3     repeat
4         For each point $\boldsymbol{x}_i$ and every cluster $c$, compute $d(\boldsymbol{x}_i, \boldsymbol{\mu}_c)$ using the kernel $\boldsymbol{K}$
5         Find $c^\star(\boldsymbol{x}_i) = \arg\min_c d(\boldsymbol{x}_i \boldsymbol{\mu}_c)$, resolving ties arbitrarily
6         Compute updated clusters as $\mathcal{A}_c^{(t+1)} = \{\boldsymbol{x}_i : c^\star(\boldsymbol{x}_i) = c\}$
7     until convergence or maximum iteration
```

Objective	Node weights	Kernel
Ratio-Association	$\mathbf{1}$	$\sigma\boldsymbol{I} + \boldsymbol{W}$
Ratio-Cut	$\mathbf{1}$	$\sigma\boldsymbol{I} - \boldsymbol{L}$
Normalized Cut	Degree of the nodes	$\sigma\boldsymbol{D}^{-1} + \boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{D}^{-1}$

Table 4.3: Kernels using by W-K-k-MEANS in Dhillon *et al.* [22] for solving different graph-based objectives.

non-positive semi-definiteness implies that at least one dimension of the space spanned by the eigenvectors of the kernel has a negative scale. Thus, proper distance cannot be computed with a non-positive semi-definite matrix. To enforce positive semi-definiteness, Dhillon *et al.* [22] proposes to add a diagonal shift controlled by an hyper-parameter σ to the kernel. However, this diagonal shift has an undesirable side effect: as σ increases, the nodes have a tendency to become closer to their own cluster. Because of this, more than often in our experiments, W-K-k-MEANS stopped at the very first iteration. That said, the problem does not come from W-K-k-MEANS, but from the choice of the kernels in Table 4.3. A better choice for normalized cut would be the Moore-Penrose pseudo-inverse of the graph Laplacian, as this is a proper positive semidefinite kernel for the commute-time distance in the graph. And the commute-time distance is closely related to normalized cut.

Then, the author presents how HMRF-k-MEANS [7] can be expressed using a kernel. HMRF-k-MEANS is a modified k-MEANS that integrates pairwise constraints whose objective function is the following:

$$\sum_{c=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{A}_c} \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2 - \sum_{\substack{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{M} \\ \mathcal{A}_i = \mathcal{A}_j}} w_{ij} + \sum_{\substack{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{C} \\ \mathcal{A}_i = \mathcal{A}_j}} w_{ij}, \tag{4.2.4}$$

where w_{ij} is a penalty for violating constraint (i, j). Authors of Kulis *et al.* [39] introduce the notion of cluster-size weighted penalties. That is, if two cannot-linked points are in the same cluster, they will penalize higher if the corresponding cluster is small. Similarly, if two must-linked nodes are in the same cluster, the reward will be higher if that cluster is small. Cluster-sized penalties may attenuate the problem identified in Section 4.1.3: in imbalanced data sets, more constraints are required to obtain enough supervision for smaller clusters. To obtain this behavior, the authors divide each weight w_{ij} by the size of the cluster that the points are in. Thus, adding cluster-sized penalties to the objective function yields the following:

$$\sum_{c=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{A}_c} \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2 - \sum_{\substack{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{M} \\ \mathcal{A}_i = \mathcal{A}_j}} \frac{w_{ij}}{|\mathcal{A}_i|} + \sum_{\substack{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{C} \\ \mathcal{A}_i = \mathcal{A}_j}} \frac{w_{ij}}{|\mathcal{A}_i|} \tag{4.2.5}$$

Then, the authors are using the following result from Duda *et al.* [23]:

$$\sum_{c=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{A}_c} 2\,\|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2 = \sum_{c=1}^{k} \sum_{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{A}_c} \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{|\mathcal{A}_c|}$$

After some algebra, and letting $\boldsymbol{E}_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$, the author rewrites this objective function in matrix form:

$$\operatorname{trace}\left(\boldsymbol{Z}^\top (\boldsymbol{E} - 2\boldsymbol{\Omega})\boldsymbol{Z}\right) \quad \text{where } \boldsymbol{Z}_{ic} = \begin{cases} |\mathcal{A}_c|^{-1/2} & \text{if } \boldsymbol{x}_i \in \mathcal{A}_c \\ 0 & \text{otherwise,} \end{cases} \tag{4.2.6}$$

where $\boldsymbol{\Omega}$ is the constraint penalty matrix. Finally, the author observes that the distance $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$ is equal to $\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle + \langle \boldsymbol{x}_j, \boldsymbol{x}_j\rangle - 2\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$. Letting the matrix $\boldsymbol{S}_{ij} = \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$ and the matrix $\widetilde{\boldsymbol{S}}_{ij} = \boldsymbol{S}_{ii} + \boldsymbol{S}_{jj}$, the distance matrix $\boldsymbol{E}$ becomes $\boldsymbol{E} = \widetilde{\boldsymbol{S}} - 2\boldsymbol{S}$. The objective function becomes

$$\operatorname{trace}\left(\boldsymbol{Z}^\top \left(\widetilde{\boldsymbol{S}} - 2\boldsymbol{S} - 2\boldsymbol{\Omega}\right) \boldsymbol{Z}\right). \tag{4.2.7}$$

Since $\operatorname{trace}\left(\boldsymbol{Z}^\top \widetilde{\boldsymbol{S}} \boldsymbol{Z}\right)$ is a constant, minimizing the previous equation is equivalent to maximizing $\operatorname{trace}\left(\boldsymbol{Z}^\top (\boldsymbol{S} + \boldsymbol{\Omega})\boldsymbol{Z}\right)$, which is equivalent to unweighted kernel k-means. The matrix $\boldsymbol{K} = \boldsymbol{S} + \boldsymbol{\Omega}$ may not be positive semidefinite. Indefinite matrices are not suitable kernel matrices. Indeed, weighted kernel k-means is not guaranteed to converge when using an indefinite matrix. More fundamentally, kernel matrices represent a collection of dimensions associated with a positive weight that can be interpreted as the width of the dimension. Let $\boldsymbol{K} = \sum_{i=1}^{n} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top$ be a $n \times n$ matrix with eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ and associated eigenvalues $\lambda_1, \ldots, \lambda_n$. If $\lambda_i \geq 0$ for all i, then the matrix $\boldsymbol{K}$ is positive semi-definite and can be used as a kernel. In that case, each dimension is represented by the pairs $(\lambda_i, \boldsymbol{u}_i)$, where $\boldsymbol{u}_i$ and λ_i are respectively the direction and the width of the dimension. However if the matrix $\boldsymbol{K}$ is indefinite, then at least one of its eigenvalues is negative and can no longer be interpreted as a width. In order to fix this issue, the authors use the same strategy used in [22]: they enforce positive semi-definiteness by adding a diagonal shift to the matrix $\boldsymbol{K}$. The objective function becomes:

$$\text{trace}\left(\boldsymbol{Z}^\top(\sigma\boldsymbol{I}+\boldsymbol{K})\boldsymbol{Z}\right) = \sigma k + \text{trace}\left(\boldsymbol{Z}^\top\boldsymbol{K}\boldsymbol{Z}\right), \tag{4.2.8}$$

where σ is a parameter to control the positive semi-definiteness of the matrix and k is the number of columns of $\boldsymbol{Z}$, which is equal to the number of clusters. Thus adding this diagonal shift adds a constant to the objective function, which guarantees that W-K-k-MEANS terminates and converges to a local optimum, while maintaining the global optimum. However, the authors also explain that adding a diagonal shift to the matrix $\boldsymbol{K}$ also greatly reduces the distance between each point and the current cluster where it belongs relatively to other data points. Consequently, any initialization of weighted kernel k-means is a local optimum. Thus adding a diagonal shift to the matrix $\boldsymbol{K}$ may prevent the execution of weighted kernel k-means. In their experiment section, the authors explore what is happening when adding a negative diagonal shift, instead of a positive one. In that case, the matrix $\sigma\boldsymbol{I}+\boldsymbol{K}$ with $\sigma \leq 0$ is not a suitable kernel, as it is indefinite. The authors also perform local search to improve the results of their algorithm.

For the authors, this approach can be used to perform graph cuts by setting the matrix $\boldsymbol{S}$ to one of the kernel matrices in Table 4.3. If the matrix $\boldsymbol{S}$ is equal to $\sigma\boldsymbol{I}+\boldsymbol{W}$ and the penalty for violating a must-link (i,j) is equal to $1-\boldsymbol{S}_{ij}$ and the penalty for violating a cannot-link (i,j) is $-\boldsymbol{S}_{ij}$, then this is equivalent to spectral learning (see Section 4.1.1). Such equivalences can also be obtained with other proposed kernels. Thus, similarly to spectral learning, we can expect that the must-link and cannot-link constraints do not propagate to unsupervised edges, and this approach may require a large number of constraints to yield good performance.

It is quite surprising that the authors have not used the pseudo-inverse of the Laplacian matrix, which is a suitable kernel that describes the commute-time distance between nodes of the graph (see Section 2.4.2).

4.2.2 Constrained Spectral Clustering through Affinity Propagation

In Lu and Carreira-Perpinán [48], soft pairwise constraints are diffused through the graph. The method has a Gaussian process interpretation and results in a closed-form expression for the new similarity matrix. The authors interpret the original similarity matrix $\boldsymbol{W} \succeq 0$ as the covariance matrix of a zero mean Gaussian process $\boldsymbol{f}$:

$$P(\boldsymbol{f}) = |2\pi\boldsymbol{W}|^{-n/2}e^{-1/2\boldsymbol{f}^\top\boldsymbol{W}\boldsymbol{f}}, \tag{4.2.9}$$

where $\boldsymbol{f} = \begin{pmatrix} f(\boldsymbol{x}_1) & \dots & f(\boldsymbol{x}_n)\end{pmatrix}^\top$ and $f(\boldsymbol{x}_i)$ is a continuous label of $\boldsymbol{x}_i$ ($f>0$: label 1, $f<0$: label 2). Pairwise constraints are treated as follows:

- If $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are must-linked, we assume that $f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) \sim \mathcal{N}(0,\epsilon_m^2)$
- If $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are cannot-linked, then $f(\boldsymbol{x}_i) + f(\boldsymbol{x}_j) \sim \mathcal{N}(0,\epsilon_c^2)$

where ϵ_m and ϵ_c soften the constraints. Let Ω be the observations described above and $\mathcal{M}$ and $\mathcal{C}$ the set of all must-links and cannot-links, respectively. The likelihood $P(\Omega|\boldsymbol{f})$ of Ω given $\boldsymbol{f}$ is proportional to:

$$\exp\left(-\sum_{i,j\in\mathcal{M}}\frac{(f(\boldsymbol{x}_i)-f(\boldsymbol{x}_j))^2}{2\epsilon_m^2}-\sum_{i,j\in\mathcal{C}}\frac{(f(\boldsymbol{x}_i)-f(\boldsymbol{x}_j))^2}{2\epsilon_c^2}\right). \tag{4.2.10}$$

From Bayes' rule, the posterior probability of f given Ω is:

$$\begin{aligned}P(\boldsymbol{f}|\Omega) \propto \exp&\left(-\frac{1}{2}\boldsymbol{f}^\top\boldsymbol{W}^{-1}\boldsymbol{f}\right)\times\\ &\exp\left(-\sum_{i,j\in\mathcal{M}}\frac{(f(\boldsymbol{x}_i)-f(\boldsymbol{x}_j))^2}{2\epsilon_m^2}-\sum_{i,j\in\mathcal{C}}\frac{(f(\boldsymbol{x}_i)-f(\boldsymbol{x}_j))^2}{2\epsilon_c^2}\right).\end{aligned} \tag{4.2.11}$$

The authors then propose to use $\overline{\boldsymbol{W}}_{ij} \equiv E\{f(\boldsymbol{x}_i)f(\boldsymbol{x}_j)|\Omega\}$ as the new affinity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Then he observes that $\boldsymbol{f}|\Omega$ is a Gaussian and $E\{\boldsymbol{f}|\Omega\}=0$. Thus, the following key result holds:

$$\begin{aligned}\overline{\boldsymbol{W}} &= (\boldsymbol{W}^{-1}+\boldsymbol{M})^{-1} = \boldsymbol{W}-\boldsymbol{W}(\boldsymbol{I}+\boldsymbol{M}\boldsymbol{W})^{-1}\boldsymbol{M}\boldsymbol{W}\\ \boldsymbol{M}_{ij} &= \begin{cases}\frac{m_i}{\epsilon_m^2}+\frac{c_i}{\epsilon_c^2} & \text{if } i=j,\\ -\frac{1}{\epsilon_m^2} & \text{if } (i,j)\in\mathcal{M},\\ \frac{1}{\epsilon_c^2} & \text{if } (i,j)\in\mathcal{C},\\ 0 & \text{otherwise.}\end{cases}\end{aligned} \tag{4.2.12}$$

When $\epsilon_m(\epsilon_c)\to 0$, we get hard must-links (cannot-links) and when $\epsilon_m(\epsilon_c)\to\infty$, we get no constraints. We need to inverse $\boldsymbol{I}+\boldsymbol{M}\boldsymbol{W}$. This can be computed in closed form using

$$\begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}^{-1} = \begin{bmatrix}\boldsymbol{A}^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix} \tag{4.2.13}$$

So, only the small $2L\times 2L$ matrix $\boldsymbol{A}$ needs to be inverted, where L is the number of must and cannot-links.

The authors observe that $\overline{\boldsymbol{W}}\succ 0$, however it may contain negative entries. Thus, the authors propose to apply normalized cuts on the matrix $[\boldsymbol{A}_{ij}]=\max\{0,\overline{\boldsymbol{W}}_{ij}\}$. The authors also discuss the limitations of this model. They say that $\overline{\boldsymbol{W}}$ is generally a sensible measure of affinity only when there are two classes. Suppose both $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are cannot-linked to $\boldsymbol{x}_k$, then we have that $f(\boldsymbol{x}_i)\approx -f(\boldsymbol{x}_k)\approx f(\boldsymbol{x}_j)$. This is equivalent to putting a must-link between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. While this is not a problem for two-classes partitionning, this is a huge issue when dealing with more than two classes. In order to overcome this difficulty, the authors propose to treat cannot-links separately. This leads to Algorithm 11 for two classes problems and Algorithm 12 for more than two classes problems.

Algorithm 11: CSCAP (for two classes)

```
   Input: Similarity matrix $\boldsymbol{W}$, must-link set $\mathcal{M}$, cannot-link set $\mathcal{C}$
1  begin
2      Compose the matrix $\boldsymbol{M}$ according to Equation (4.2.12) based on all constraints
3      Let $\overline{\boldsymbol{W}} = (\boldsymbol{W}^{-1} + \boldsymbol{M})^{-1}$
4      Let $\boldsymbol{A}_{ij} = \max\{0, \overline{\boldsymbol{W}}_{ij}\}$ for all $i, j$
5      Do normalized cut with $\boldsymbol{A}$ as the affinity matrix
```

Algorithm 12: CSCAP (more than two classes)

```
   Input: Similarity matrix $\boldsymbol{W}$, must-link set $\mathcal{M}$, cannot-link set $\mathcal{C}$, number of
          clusters $k$
1  begin
2      Compose the matrix $\boldsymbol{M}^m$ according to Equation (4.2.12) based on must-links
3      Let $\overline{\boldsymbol{W}}^m = (\boldsymbol{W}^{-1} + \boldsymbol{M}^m)^{-1}$
4      foreach $i \in \{1, \ldots, n_c\}$ where $n_c$ is the number of cannot-links do
5          Compose the matrix $\boldsymbol{M}^{c,i}$ according to Equation (4.2.12) based on the $i$th
           cannot-link
6          Let $\overline{\boldsymbol{W}}^{c,i} = ((\overline{\boldsymbol{W}}^m)^-1 + \boldsymbol{M}^{c,i})^-1$
7      Let $\boldsymbol{A}_{ij} = \max\{0, \min\{\overline{\boldsymbol{W}}_{ij}^{c,1}, \ldots, \overline{\boldsymbol{W}}_{ij}^{c,n_c}\}\}$ for all $i, j$
8      Do normalized cuts with $\boldsymbol{A}$ as the affinity matrix and $k$ clusters
```

4.3 Tuning the spectral embedding

Spectral clustering with pairwise semi-supervision can also be done within the spectral embedding instead of directly on the similarity matrix of the graph. In this section, we present Li *et al.* [45] and Chatel *et al.* [14]. Both of these approaches attempt to fix the spectral embedding by learning a transformation of this space such that constrained nodes behave in a certain way.

For Li *et al.* [45], the vectors describing constrained nodes should point in the same direction for must-linked nodes and in orthogonal directions for cannot-linked nodes. The authors cast the original non-convex problem into a semi-definite program. In addition to guaranteeing the optimality of the solution, this approach allows to solve the problem in a time independent of the size of the graph or the number of constraints. However, the semi-definite program works on multiple huge convex cones whose size grow quadratically with the number of analyzed dimensions of the spectral embedding, therefore, in practice, this approach is limited in that number.

The second approach we are presenting, Chatel *et al.* [14], is our main contribution on pairwise constraints. As in Li *et al.* [45], our approach consists in learning a linear transformation of the spectral embedding such that, in the transformed embedding, must-linked nodes are close to each others and cannot-linked nodes are far away from each others. And similarly to Lu and Carreira-Perpinán [48], we define a neighborhood using Gaussian functions for must-links and cannot-links that controls the distance at which these constraints will have an effect. The problem is then solved using a gradient descent on the objective function.

4.3.1 Constrained Clustering via Spectral Regularization

In this section, we present Constrained Clustering via Spectral Regularization, or CCSR. This clustering method from [45] relies on adapting the spectral embedding in order to best satisfy soft pairwise constraints which specifiy whether pairs of nodes should be in the same cluster or not. As these constraints are soft, it is allowed not to satisfy them all. Although the complexity of the semi-definite program used to solve the problem is independent of the number of nodes in the graph or the number of constraints, its size grows quadratically with the number of dimensions of the spectral embedding. Thus, even if this method is scalable to large-scale problems, the results, when compared to other algorithms, may be disappointing when the correlation between the reference partition and the spectral embedding is low.

In order to partition a set of data into k clusters, it is desired to learn k cluster indicators $\boldsymbol{F} = \begin{pmatrix}\boldsymbol{f}_1 & \dots & \boldsymbol{f}_k\end{pmatrix}$ such that $\boldsymbol{f}_i(j) = 1$ if the node i belong to cluster j and 0 otherwise. The authors observe that $(\boldsymbol{F}\boldsymbol{F}^\top)_{ij} = 1$ if and only if nodes i and j belong to the same cluster and 0 otherwise. From these observations, they propose to transform the spectral embedding such that most of the constraints are satisfied. It leads to the following optimization problem:

$$\min_{\boldsymbol{X}} \sum_{(i,j,t_{ij})\in\mathcal{S}} \left(\boldsymbol{u}_i^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{u}_j - t_{ij}\right)^2 \tag{4.3.1}$$

where $\boldsymbol{u}_i$ is the representation of the node i in a m-dimensional spectral embedding, $\mathcal{S}$ is the set of all constraints (i, j, t_{ij}) with node i, j and constraint $t_{ij} = 1$ if nodes i and j belong to the same cluster, 0 otherwise.

Notice that if there is no constraint between two nodes i and j, then there exists no tuple (i, j, t_{ij}) in $\mathcal{S}$. That is, unconstrained pairs of nodes are not part of this optimization problem.

The authors observe that Equation (4.3.1) is not convex with respect to $\boldsymbol{X}$ in general. However, it can be relaxed to a convex problem. Let $\boldsymbol{M} = \boldsymbol{X}\boldsymbol{X}^\top$. Then $\boldsymbol{M}$ is a positive semidefinite matrix, denoted as $\boldsymbol{M} \succeq 0$ and the objective function (4.3.1) becomes:

$$\sum_{(i,j,t_{ij})\in\mathcal{S}} \left(\boldsymbol{u}_i^\top \boldsymbol{M}\boldsymbol{u}_i - t_{ij}\right)^2 : \boldsymbol{M} \succeq 0. \tag{4.3.2}$$

Let $\boldsymbol{z} = \mathrm{vec}(\boldsymbol{M})$. We have

$$\boldsymbol{z}^\top \boldsymbol{B}\boldsymbol{z} + \boldsymbol{b}^\top \boldsymbol{z} + c, \tag{4.3.3}$$

where

Algorithm 13: CCSR

```
   Input: $W$: similarity matrix, $\mathcal{M}, \mathcal{C}$: pairwise constraints, $k$: number of clusters.
   Output: Cluster labels for all the nodes.
1  begin
2      Form the normalized graph Laplacian $L_{sym} = I - D^{-1/2} W D^{-1/2}$
3      Compute the $m$ eigenvectors $v_1, \ldots, v_m$ of $L_{sym}$ corresponding to the first $m$
        smallest eigenvalues
4      Let $F = \begin{pmatrix} v_1 & \ldots & v_m \end{pmatrix}$
5      Solve the SDP problem (4.3.4) for $M$.
6      Apply $k$-MEANS to the rows of $FM^{1/2}$ to form $k$ clusters.
```

$$\boldsymbol{B} = \textstyle\sum_{(i,j,t_{ij}) \in \mathcal{S}} \boldsymbol{s}_{ij} \boldsymbol{s}_{ij}^\top, \qquad \boldsymbol{b} = -2 \sum_{(i,j,t_{ij}) \in \mathcal{S}} t_{ij} \boldsymbol{s}_{ij},$$

$$c = \textstyle\sum_{(i,j,t_{ij}) \in \mathcal{S}} t_{ij}^2, \qquad \boldsymbol{s}_{ij} = \operatorname{vec}\left(\boldsymbol{u}_j \boldsymbol{u}_i^\top\right).$$

The matrix $\boldsymbol{B}$ is positive semidefinite. Let $\boldsymbol{B} = \boldsymbol{B}^{1/2} \boldsymbol{B}^{1/2}$ where $\boldsymbol{B}^{1/2}$ denotes the matrix square root of $\boldsymbol{B}$. Finally using Schur complement, the optimization problem (4.3.1) is equivalent to

$$\min_{\boldsymbol{z}, \nu} \nu + \boldsymbol{b}^\top \boldsymbol{z} : \boldsymbol{M} \succeq 0 \text{ and } \begin{pmatrix} \boldsymbol{I}_{m^2} & \boldsymbol{B}^{1/2} \boldsymbol{z} \\ \left(\boldsymbol{B}^{1/2} \boldsymbol{z}\right)^\top & \nu \end{pmatrix} \succeq 0, \tag{4.3.4}$$

where $\boldsymbol{I}_{m^2}$ is the identity matrix of size $m^2 \times m^2$ and ν is a dummy variable serving as an upper bound of $\boldsymbol{z}^\top \boldsymbol{B} \boldsymbol{z}$. The algorithm is summarized in Algorithm 13.

Although this semidefinite program is independent of the size of the graph or the number of constraints, it has $m(m+1)/2 + 1$ variables subject to two semidefinite cones constraints of size $m \times m$ and $(m^2 + 1) \times (m^2 + 1)$. Hence, its complexity grows quadratically with the number of dimensions of the spectral embedding. The authors say that for small m, for example $m = 15$, this problem can be solved under a minute using CSDP 6.0.1 in MATLAB 7.6.0 (R2008a) on a PC with 3.4 GHz CPU and 4 GB RAM. However, if the first m eigenvectors of the Laplacian matrix, used to obtain the spectral embedding, are not correlated with the reference partition, then this method will fail. Another remark about this method is that it will not give very good results on small clusters unless these clusters are heavily supervised. Indeed, in presence of many small clusters, there is no way to determine if different unsupervised groups of nodes should or should not connect and form a cluster. For example, if you consider 3 small groups of nodes a, b and c, and you know that a and b should not connect, because there is a cannot-link between them, and b and c should not connect for the same reason, you cannot say anything about a and c, unless there are only 2 clusters or unless there is some supervision between a and c. Now, consider the case where you have many small groups of nodes $a_1, \dots, a_c$, where c is the count of these small groups. Then unless every pair of small groups (a_i, a_j) has supervision, there is always an uncertainty about the clusters, because it suffices that one pair is not supervised to obtain a triangle, like the triangle described before (a, b, c-triangle).

4.3.2 Fast Gaussian Pairwise Constrained Clustering

In previous works, we have seen that integrating pairwise constraints is often a story about propagating the constraints to unconstrained nodes. In particular, we have seen that proposals that do not attempt to propagate the constraints [37, 18, 39] often fail to recover the correct partition. We can also observe that constraints propagation is often interpreted as a Gaussian function [48, 44].

In this section, we propose to exploit these observations by learning a linear transformation $\boldsymbol{X}$ of the spectral embedding of the graph with the partial supervision given by the constraints. Our algorithm also learns a similarity in order to find a partition such that similar nodes are in the same cluster, dissimilar nodes are in different clusters, and the maximum number of pairwise constraints are satisfied. When two nodes must link (respectively cannot link), their similarity is constrained to be close to 1 (respectively close to 0). In the learning step, the similarity is locally distorted around constrained nodes using a Gaussian function applied on the Euclidean distance in the feature space obtained by $\boldsymbol{X}$. In order to increase the gap between must-link and cannot-link constraints, we use two Gaussian functions of different variances. As illustrated in Figure 4.5, this technique ensures that the distance in the new feature space between nodes in cannot-link constraints is significantly larger than the distance between nodes that must link. From this modeling, we derive a non-convex optimization problem to learn the transformation $\boldsymbol{X}$. We solve this problem using a gradient descent approach with an initialization for $\boldsymbol{X}$ that coincides with the unconstrained solution of the problem.

In the following, we consider the d-dimensional spectral embedding, defined as

$$\boldsymbol{V}_d = \begin{pmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_d \end{pmatrix} = \begin{pmatrix} \boldsymbol{v}_1 \dots \boldsymbol{v}_n \end{pmatrix}^\top,$$

where $\boldsymbol{u}_1, \dots, \boldsymbol{u}_d$ correspond to the eigenvectors associated with the d smallest eigenvalues of the normalized graph Laplacian $\boldsymbol{L_{sym}} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$ and $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n$ correspond to the projections of the nodes $v_1, \dots, v_n$ of the graph into the d-dimensional spectral embedding.

Pairwise constraints are defined as follows. Let $\mathcal{M}, \mathcal{C} \subset \mathcal{V} \times \mathcal{V}$ be two sets of pairs of nodes, describing must-link and cannot-link constraints. Let K be the total number of constraints. If $(v_i, v_j) \in \mathcal{M}$, then v_i and v_j should be in the same cluster, and if $(v_i, v_j) \in \mathcal{C}$ then v_i and v_j should be in different clusters. We introduce the $K \times d$ matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and the K-dimensional vector $\boldsymbol{q}$:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{v}_{i_1} \\ \vdots \\ \boldsymbol{v}_{i_K} \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} \boldsymbol{v}_{j_1} \\ \vdots \\ \boldsymbol{v}_{j_K} \end{pmatrix} \qquad \boldsymbol{q}_k = \begin{cases} 1 & (v_{i_k}, v_{j_k}) \in \mathcal{M} \\ 0 & (v_{i_k}, v_{j_k}) \in \mathcal{C} \end{cases} \tag{4.3.5}$$

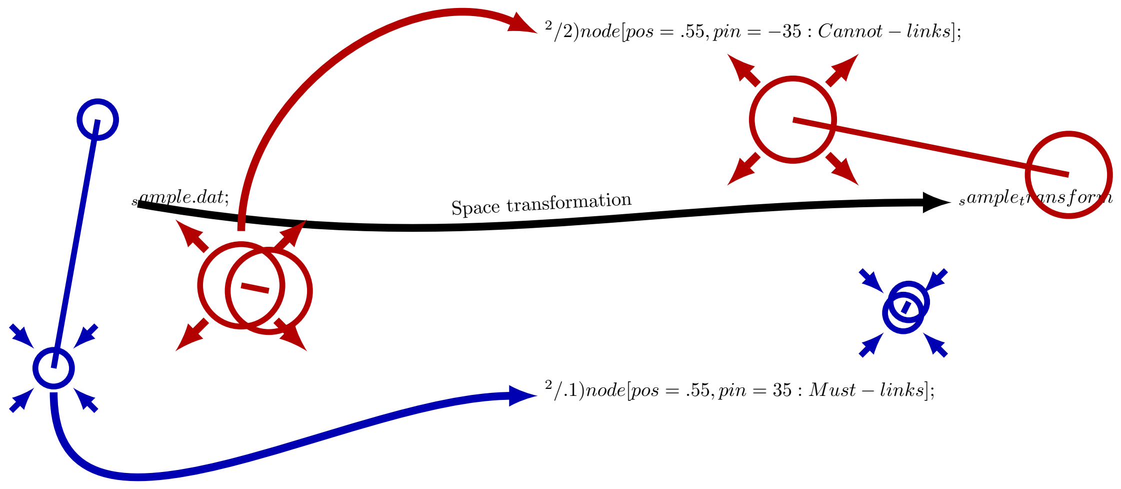

2/2)node[pos = .55, pin = −35 : Cannot − links];
sample.dat;
Space transformation
sample_transform
2/.1)node[pos = .55, pin = 35 : Must − links];

Figure 4.5: This figure shows intuitively the inner workings of behind FGPWC. From a spectral embedding of a graph, Gaussian functions distort the distance between constrained pairs of nodes such that it becomes smaller or larger depending depending on the quality (must-link or cannot-link) attributed to the constraint. Gaussian functions act as a new similarity for the pair of nodes and it should be close to 1 if the pair must link and close to 0 if the pair cannot link.

where $(\boldsymbol{v}_{i_k}, \boldsymbol{v}_{j_k})$ are vectors describing the kth pair of nodes (v_{i_k}, v_{j_k}) in $\mathcal{M} \cup \mathcal{C}$.

We propose to learn a linear transformation ϕ of the feature space $\boldsymbol{V}_d$ that best satisfies the constraints. Let $\phi(\boldsymbol{v}_i) = \boldsymbol{v}_i \boldsymbol{X}$ where $\boldsymbol{X}$ is a $d \times d$ matrix describing the transformation of the space. We want to find a projection of the feature space $\phi(\boldsymbol{v}_i)$ such that the clusters are dense and far away from each other. Ideally, if nodes $(v_i, v_j) \in \mathcal{M}$ then the distance between $\phi(\boldsymbol{v}_i)$ and $\phi(\boldsymbol{v}_j)$ should equal zero and if nodes $(v_i, v_j) \in \mathcal{C}$ then the distance between $\phi(\boldsymbol{v}_i)$ and $\phi(\boldsymbol{v}_j)$ should be very large. We introduce Gaussian functions with parameters σ_m and σ_c to locally distort the similarities for must-linked pairs and cannot-linked pairs. The similarity between two nodes v_i and v_j is:

$$\begin{array}{ll} \exp^{\frac{-\left\|\boldsymbol{v}_i - \boldsymbol{v}_j\right\|^2}{\sigma_m}} & \text{if } (v_i, v_j) \in \mathcal{M} \\ \exp^{\frac{-\left\|\boldsymbol{v}_i - \boldsymbol{v}_j\right\|^2}{\sigma_c}} & \text{if } (v_i, v_j) \in \mathcal{C} \end{array}$$

where $\|\cdot\|$ is the Frobenius norm. Therefore, we want to ensure that $\boldsymbol{X}$ is such that $\exp^{-\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2/\sigma_m}$ is close to 1 if $(v_i, v_j) \in \mathcal{M}$ and $\exp^{-\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2/\sigma_c}$ is close to 0 if $(v_i, v_j) \in \mathcal{C}$.

We now encode the set of all constraints in a matrix form. Let us first consider the

K-dimensional vector:

$$(\boldsymbol{\sigma}_i) = \begin{cases} \frac{1}{\sigma_m} & \text{if the } i\text{th constraint is a must-link,} \\ \frac{1}{\sigma_c} & \text{if the } i\text{th constraint is a cannot-link.} \end{cases} \tag{4.3.6}$$

Let $\boldsymbol{e}$ be the d-dimensional vector of all ones. Notice that $[(\boldsymbol{A}-\boldsymbol{B})\boldsymbol{X}]^2\boldsymbol{e}$, is the vector whose components are equal to the distance between pairs of constrained nodes in the transformed space. Let $\odot$ be the Hadamard product. Then $\exp^{-[(\boldsymbol{A}-\boldsymbol{B})\boldsymbol{X}]^2\boldsymbol{e}\odot\boldsymbol{\sigma}}$ is the vector whose components equal the corresponding must-link or cannot-link similarity depending on whether the associated pairs of nodes are in $\mathcal{M}$ or $\mathcal{C}$. The values in $\boldsymbol{X}$ are not bounded in this expression. So, we propose to add a regularization term on $\boldsymbol{X}$. This gives the optimization problem:

$$\min_{\boldsymbol{X}} F(\boldsymbol{X}) = \left\| \exp^{-[(\boldsymbol{A}-\boldsymbol{B})\boldsymbol{X}]^2\boldsymbol{e}\odot\boldsymbol{\sigma}} - \boldsymbol{q} \right\|^2 + \gamma \left\| \boldsymbol{X} \right\|^2 \tag{4.3.7}$$

The regularization makes the function convex and smooth far away from the global optimum.

The dimension d of the spectral embedding is related to the amount of contradiction between the graph and the constraints. Remember that eigenvectors of $\boldsymbol{L_{sym}}$ are functions which map nodes from the manifold of the graph to real lines and the associated eigenvalues provide us with an estimate of how far apart these functions map nearby points [8]. When the pairwise constraints do not contradict the manifold of the graph, i.e. must-link pairs are already close on the manifold and cannot-link pairs are already far apart, d does not need to be large, because the eigenvectors associated with smallest eigenvalues will provide eigenmaps which do not contradict the constraints. Hence, a solution can be found in the very first eigenvectors. However, when the pairwise constraints contradict the manifold of the graph: must-links that are initially far apart on the manifold or cannot-links that are close, we need to consider a larger number of eigenvectors d, because the eigenvectors providing the eigenmaps that will not contradict the constraints will be later dimensions of the embedded space, describing smaller details.

Our algorithm for learning the transformation $\boldsymbol{X}$ is presented in Algorithm 14. It takes as input a weighted adjacency matrix of a graph, and two matrices for must-link and cannot-link constraints. Parameters are the number k of clusters as usual in k-MEANS, but also the widths of the Gaussian functions σ_m and σ_c and the dimension d of $\boldsymbol{X}$. Our algorithm is a typical gradient descent and its initialization can be at random. However, we propose to initialize it close to unconstrained spectral clustering by setting

$$\boldsymbol{X}_0 = (\boldsymbol{V}_m^\top \boldsymbol{L_{sym}} \boldsymbol{V}_m)^{-1/2}. \tag{4.3.8}$$

We stop the descent after i_{max} iterations or when the Frobenius norm of the derivative $\nabla F(\boldsymbol{X})$ is less than ϵ.

4.4 Other approaches

In the following we briefly present two approaches that introduce pairwise constraints into the objective function of the graph cut problem in different ways. For Rangapuram

Algorithm 14: FGPWC

```
Input: $\boldsymbol{W} \in \mathbb{R}^{n\times n}, \boldsymbol{M} \in \mathbb{R}^{n\times n}, \boldsymbol{C} \in \mathbb{R}^{n\times n}, m, k, \sigma_m, \sigma_c$
Output: $\boldsymbol{X}^\star \in \mathbb{R}^{d\times d}, \mathcal{P}$ partition of $\mathcal{V}$
1 begin
2     $\boldsymbol{L_{sym}} \leftarrow \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
3     $\boldsymbol{V}_m \leftarrow$ first $d$ smallest eigenvectors of $\boldsymbol{L_{sym}}$
4     $\boldsymbol{X} \leftarrow (\boldsymbol{V}_m^\top \boldsymbol{L_{sym}} \boldsymbol{V}_m)^{-1/2}$
5     Descend the gradient $\nabla_{\boldsymbol{X}} F(\boldsymbol{X})$ until convergence
6     $\mathcal{P} \leftarrow k$-MEANS$(\boldsymbol{V}_m\boldsymbol{X}, k)$
7     return $\mathcal{P}$
```

and Hein [74] the problem consists in a tradeoff between the normalized graph cut and the number of violated constraints. Like the discrete graph cut problem, the discrete version of [74] is hard, but the authors cast the problem into a non-convex continuous optimization problem. The non-convexity is not very desirable but this approach guarantees that all constraints are satisfied as long as there are only 2 clusters. The second approach reinterprets the normalization factor in the normalized cut problem. The degrees of the nodes are considered as a repulsive force while the weights of the edges are considered as an attractive force. The authors use this interpretation to introduce must- and cannot-links in an elegant way.

4.4.1 Constrained 1-Spectral Clustering

Constrained 1-Spectral Clustering, or COSC, is a pairwise constrained spectral clustering method developed in Rangapuram and Hein [74]. The clustering problem is expressed as a trade-off between having a small normalized cut and a small number of violated constraints. The discrete problem is hard but it is cast into a non-convex continuous optimization problem. The convergence to the global optimum is not guaranteed, however it is shown that the algorithm either improves any given partition which satisfies all constraints or it stops after one iteration. The method is guaranteed to satisfy all the constraints for 2 clusters. This guarantee is lost for data sets with more than 2 clusters.

The authors start with a graph $G = (\mathcal{V}, \mathcal{E}, w, b)$ and the constraint matrices $\boldsymbol{Q}^m$ and $\boldsymbol{Q}^c$, where the element q_{ij}^m (respectively q_{ij}^c) $\in \{0,1\}$ specifies the must-link (respectively the cannot-link) constraint between i and j. They propose the set function $\hat{F}_\gamma : 2^{\mathcal{V}} \mapsto \mathbb{R}$:

$$\begin{aligned}\hat{F}_\gamma(\mathcal{C}) &= \frac{2\mathrm{cut}(\mathcal{C},\overline{\mathcal{C}}) + \gamma\left(\hat{M}(\mathcal{C}) + \hat{N}(\mathcal{C})\right)}{\mathrm{bal}(\mathcal{C})} \\ \text{where } \hat{M}(\mathcal{C}) &\triangleq 2\sum_{i\in\mathcal{C}, j\in\overline{\mathcal{C}}} q_{ij}^m, \\ \hat{N}(\mathcal{C}) &\triangleq \mathrm{vol}(\boldsymbol{Q}^c) - 2\sum_{i\in\mathcal{C}, j\in\overline{\mathcal{C}}} q_{ij}^c.\end{aligned} \tag{4.4.1}$$

We have $2\mathrm{cut}(\mathcal{C},\overline{\mathcal{C}})$ in this thesis, compared to just once in the original paper, because the author defines $\mathrm{cut}(\mathcal{C},\overline{\mathcal{C}})$ as $2\sum_{i\in\mathcal{C},j\in\overline{\mathcal{C}}} w_{ij}$, while in this thesis we have half of this value. Notice that $\hat{M}(\mathcal{C})$ — respectively $\hat{N}(\mathcal{C})$ — is equal to twice the number of violated must-link — respectively cannot-link — constraints. The denominator $\mathrm{bal}(\mathcal{C})$ is a normalization factor that turns the cut problem into normalized cut and is equal to

$$\mathrm{bal}(\mathcal{C}) = 2\frac{\mathrm{vol}(\mathcal{C})\mathrm{vol}(\overline{\mathcal{C}})}{\mathrm{vol}(\mathcal{V})}. \tag{4.4.2}$$

The objective function (4.4.1) depends on the hyper parameter γ, however its choice is constructive, as shown in lemma 1 and Theorem 1.

Lemma 1. *Let $(\mathcal{C},\overline{\mathcal{C}})$ be consistent with the given constraints and $\lambda = \mathrm{Ncut}(\mathcal{C},\overline{\mathcal{C}})$. If $\gamma \geq \frac{\mathrm{vol}(\mathcal{V})}{4(l+1)}\lambda$, then any minimizer of $\hat{F}_\gamma$ violates no more than l constraints.*

Theorem 1 is a direct consequence of Lemma 1.

Theorem 1. *Let $(\mathcal{C},\overline{\mathcal{C}})$ be consistent with the given constraints and $\lambda = \mathrm{Ncut}(\mathcal{C},\overline{\mathcal{C}})$. Then for $\gamma \geq \frac{\mathrm{vol}(\mathcal{V})}{4}\lambda$, it holds that*

$$\arg \min_{\substack{\mathcal{C}\subset\mathcal{V} \\ (\mathcal{C},\overline{\mathcal{C}})\ \text{consistent}}} \mathrm{Ncut}(\mathcal{C},\overline{\mathcal{C}}) = \arg\min_{\mathcal{C}\subset\mathcal{V}} \hat{F}_\gamma(\mathcal{C}) \tag{4.4.3}$$

and the optimum values of both problems are equal.

Minimizing $\hat{F}_\gamma$ is a hard combinatorial problem, however the authors propose a tight continuous relaxation of $\hat{F}_\gamma$. They define

$$M(\boldsymbol{f}) \triangleq \sum_{i,j=1}^{n} q_{ij}^{m}|\boldsymbol{f}_i - \boldsymbol{f}_j| \tag{4.4.4}$$

$$N(\boldsymbol{f}) \triangleq \mathrm{vol}(\boldsymbol{Q}^c)(\max(\boldsymbol{f}) - \min(\boldsymbol{f})) - \sum_{i,j=1}^{n} q_{ij}^{c}|\boldsymbol{f}_i - \boldsymbol{f}_j|, \tag{4.4.5}$$

where $\max(\boldsymbol{f})$ and $\min(\boldsymbol{f})$ are respectively the maximum and minimum elements of $\boldsymbol{f}$. Also let $\boldsymbol{B}$ denote the diagonal matrix with entries equal to the vertex weights $\boldsymbol{b}$ and

$$F_\gamma(\boldsymbol{f}) = \frac{\sum_{i,j=1}^{n} w_{ij}|\boldsymbol{f}_i - \boldsymbol{f}_j| + \gamma M(\boldsymbol{f}) + \gamma N(\boldsymbol{f})}{\|\boldsymbol{B}\left(\boldsymbol{f} - \mathrm{vol}(\mathcal{V})^{-1}\langle \boldsymbol{f}, \boldsymbol{b}\rangle \boldsymbol{1}\right)\|_1} \tag{4.4.6}$$

Then, Theorem 2 follows, as well as Theorem 3 as a direct consequence of Theorems 1 and 2.

Theorem 2. *For $\gamma \geq 0$,*

$$\min_{\mathcal{C}\subset\mathcal{V}} \hat{F}_\gamma(\mathcal{C}) = \min_{\substack{\boldsymbol{f}\in\mathbb{R}^n \\ \boldsymbol{f}\ \text{non-constant}}} F_\gamma(\boldsymbol{f}) \tag{4.4.7}$$

Moreover, a solution of the first problem can be obtained from the solution of the second problem.

Theorem 3. *Let $(\mathcal{C},\overline{\mathcal{C}})$ be consistent with the constraints and $\lambda = \mathrm{Ncut}(\mathcal{C},\overline{\mathcal{C}})$. Then for $\gamma = \frac{\mathrm{vol}(\mathcal{V})}{4}\lambda$,*

$$\min_{\substack{\mathcal{C}\subset\mathcal{V} \\ (\mathcal{C},\overline{\mathcal{C}}) \text{ consistent}}} \mathrm{Ncut}(\mathcal{C},\overline{\mathcal{C}}) = \min_{\substack{\boldsymbol{f}\in\mathbb{R}^n \\ \boldsymbol{f} \text{ non-constant}}} F_\gamma(\boldsymbol{f}) \tag{4.4.8}$$

Furthermore, an optimal partition of the constrained problem can be obtained from a minimizer of the right problem.

The problem is optimized using the method developed in Hein and Setzer [31].

4.4.2 Scalable Constrained Clustering: A Generalized Spectral Method

Scalable Constrained Clustering is a recent work from Cucuringu *et al.* [18], which, according to the authors, consists in a natural generalization of the basic spectral clustering algorithm. This work considers a graph cut with pairwise constraints. Must-link constraints are integrated directly into the similarity matrix $\boldsymbol{W}$, while cannot-link constraints receive a special treatment that is the main contribution of the article. The authors introduce a demand graph whose similarity matrix $\boldsymbol{K}_{ij} = \boldsymbol{d}_i\boldsymbol{d}_j/\mathrm{vol}(\mathcal{V})$ verifies

$$\mathrm{cut}_{\boldsymbol{K}}(\mathcal{A},\overline{\mathcal{A}}) = \mathrm{vol}(\mathcal{A})\mathrm{vol}(\overline{\mathcal{A}})/\mathrm{vol}(\mathcal{V}). \tag{4.4.9}$$

Then we have

$$\min_{\mathcal{A}\subset\mathcal{V}} \mathrm{Ncut}(\mathcal{A},\overline{\mathcal{A}}) = \min_{\mathcal{A}\subset\mathcal{V}} \frac{\mathrm{cut}_{\boldsymbol{W}}(\mathcal{A},\overline{\mathcal{A}})}{\mathrm{cut}_{\boldsymbol{K}}(\mathcal{A},\overline{\mathcal{A}})}. \tag{4.4.10}$$

Now, consider $\boldsymbol{W}$ as a graph of soft must-link constraints and $\boldsymbol{K}$ as a graph of soft cannot-link constraints. Cutting the graph $\boldsymbol{W}$ implies violating the constraints involved in the cut, while cutting $\boldsymbol{K}$ implies satisfying constraints involved in the cut. The degree in the matrix $\boldsymbol{W}$ of a node can be interpreted as an incentive to not be connected to any other node. Indeed, consider a graph with bounded edge weights. If a node v has a very high degree, it means it is connected to many nodes, but it also means that the probability to jump from v to any other node u is very low, since this probability is equal to $\boldsymbol{W}_{vu}/\boldsymbol{d}_v$ and $\boldsymbol{d}_v$ is very high compared to $\boldsymbol{W}_{vu}$ which is bounded. So, intuitively, it seems that the degree of a node has the same role as a soft cannot-link. The authors say the normalized cut objective can be seen as

$$\min_{\mathcal{A}\subset\mathcal{V}} \frac{\text{weight of the cut (violated) must-link constraints}}{\text{weight of the cut (satisfied) cannot-link constraints}}. \tag{4.4.11}$$

In the following, we assume that soft must-links and cannot-links are respectively encoded by the matrices $\boldsymbol{G}$ and $\boldsymbol{H}$. Then for each cluster, we have a measure of "badness" equal to

$$\phi_i(\boldsymbol{G},\boldsymbol{H}) = \frac{\mathrm{cut}_{\boldsymbol{G}}(\mathcal{C}_i,\overline{\mathcal{C}_i})}{\mathrm{cut}_{\boldsymbol{H}}(\mathcal{C}_i,\overline{\mathcal{C}_i})}. \tag{4.4.12}$$

Algorithm 15: FAST-GE

```
    Input: $\boldsymbol{G}$, $\boldsymbol{H}$, $\boldsymbol{d}$, $k$
    Output: embedding $\boldsymbol{U} \in \mathbb{R}^{n\times k}$
1   begin
2       $\boldsymbol{U} \leftarrow$ the first $k$ eigenvectors of $\boldsymbol{L_G x} = \lambda \boldsymbol{L_H x}$
3       $\boldsymbol{U} \leftarrow \boldsymbol{U} - \frac{\boldsymbol{1 d}^\top \boldsymbol{U}}{\boldsymbol{1}^\top \boldsymbol{d}}$
4       $\boldsymbol{U} \leftarrow \boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{L_H U})^{-1/2}$
5       $\boldsymbol{S} \leftarrow$ diagonal matrix with entries $\boldsymbol{S}_{ii} = \sum_{j=1}^n \boldsymbol{U}_{ij}^2$
6       $\boldsymbol{U} \leftarrow \boldsymbol{S}^{-1/2}\boldsymbol{U}$
7       return $\boldsymbol{U}$
```

The numerator equals the total weight of violated must-link constraints, while the denominator equals the total weight of satisfied cannot-link constraints. Equation (4.4.12) can be rewritten in terms of Laplacians:

$$\phi_i(\boldsymbol{G},\boldsymbol{H}) = \frac{\boldsymbol{x}_{\mathcal{C}_i}^\top \boldsymbol{L_G x}_{\mathcal{C}_i}}{\boldsymbol{x}_{\mathcal{C}_i}^\top \boldsymbol{L_H x}_{\mathcal{C}_i}}, \tag{4.4.13}$$

where $\boldsymbol{x}_{\mathcal{C}_i}$ is the indicator vector for cluster i. The problem may be not well defined, as there may be few cannot-link constraints. However this is not a problem because we are looking for a minimum and the optimization function avoids vectors that are in the null space of $\boldsymbol{L_H}$.

Relaxing the discrete condition on $\boldsymbol{x}_{\mathcal{C}_i}$, we are looking for the k smallest eigenvectors of the generalized eigenvalue problem $\boldsymbol{L_G x} = \lambda \boldsymbol{L_H x}$. Then, the authors normalize each of these vectors in various ways. First, they observe that the generalized eigenvalue problem can be viewed as a simple eigenvalue problem over a space endowed with a $\boldsymbol{H}$-inner product: $\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{\boldsymbol{H}} = \boldsymbol{x}^\top \boldsymbol{H y}$. So, they normalize the eigenvectors to a unit $\boldsymbol{H}$-norm. Given this normalization, the data points at this stage are expected to concentrate in k different directions. Hence, the authors project the data points onto the k-dimensional sphere. Finally, as $\boldsymbol{L_G}$ and $\boldsymbol{L_H}$ share the constant vector in their null spaces, if $\boldsymbol{x}$ is an eigenvector, then for all c the vector $\boldsymbol{x} + c\boldsymbol{1}^n$ is also an eigenvector with the same eigenvalue. Among all possible eigenvectors, the authors pick one such that $\boldsymbol{x} + c\boldsymbol{1}^n \perp \boldsymbol{d}$. This choice makes possible the analysis of a theoretical guarantee for a 2-way cut. Algorithm 15 summarizes this procedure.

Usually, the user provides a similarity graph $\boldsymbol{W}$, a set of must-link constraints $\mathcal{M}$ and a set of cannot-link constraints $\mathcal{C}$. Merging the matrix $\boldsymbol{W}$ and the two sets of constraints is a problem. The author constructs two weighted graphs $\boldsymbol{G}_{ML}$ and $\boldsymbol{G}_{CL}$ as follows: if (i,j) is a constraint, its weight in the corresponding graph equals $\boldsymbol{d}_i\boldsymbol{d}_j/(\min \boldsymbol{d} \max \boldsymbol{d})$. Then we let $\boldsymbol{G} = \boldsymbol{W} + \boldsymbol{G}_{ML}$ and $\boldsymbol{H} = \boldsymbol{K}/n + \boldsymbol{G}_{CL}$, where $\boldsymbol{K} = \boldsymbol{dd}^\top/\mathrm{vol}(\mathcal{V})$. The fact that high degree nodes have more influence in the graph justifies this choice.

In practice, this method requires a high number of constraints to work.

4.5 Experiments on pairwise constraints

In this section, we conduct two experiments on pairwise constraints. The first one uses standard UCI and network data sets, while the second one consists in the mention-to-entity resolution in the coreference task.

4.5.1 UCI and Network Data sets

In this first experiment, we want to show that adding pairwise constraints into clustering can result in better conformance with the expected result. First, we will explain our experimental setup, then we will present our results.

System settings

For each dataset, 10 different sets of constraints were selected at random. The number of constraints is chosen to avoid trivial solutions. Indeed, if the number of must-link constraints is high, a transitive closure quickly gives a perfect solution. So, the interesting cases are when only a few number of constraints is considered. Given a graph with n nodes, a set of pairs is added to the set of constraints with probability $1/n$. A pair forms a must- link constraint if the two nodes have the same class and a cannot-link constraint otherwise. We report the mean over the 10 runs corresponding to 10 sets of constraints of the Adjusted Rand Index computed against the ground truth. As an additional measure, we also report the number of violated constraints in the computed partition.

Algorithms FGPWC, SSC:KA, FAST-GE, CCSR and CSCAP rely on a k-means step which is non deterministic. So, we repeat 30 times each execution and select the partitions that violate a minimal number of constraints. The results evaluated on unconstrained pairs are averaged considering the 10 different sets of constraints.

The algorithm W-K-k-MEANS is not used directly because it's not specifically designed as a pairwise constrained clustering algorithm. Algorithm SSC:KA is the algorithm based on W-K-k-MEANS that is designed as a pairwise constrained clustering algorithm.

All experiments were conducted using a C++ implementation of our algorithm FG-PWC. We are using k-MEANS with smart initialization [4] and avoid empty clusters using [66]. We have found that initializing FGPWC so that it is close to unconstrained spectral clustering performs better than a random initialization. For CCSR and COSC, we use the code provided by the authors on their webpages.

Results and discussion

Results for the first set of experiments for 22 datasets are presented in Table 4.4. Empty cells correspond to the case where the algorithm did not terminate after 15 minutes.

We can see that FGPWC outperforms other algorithms on most datasets, either because it reports a higher Adjusted Rand Index score, or because it beats other algorithms in terms of computation time, as some algorithms did not terminate their computation within 15 minutes. Only on the dataset *wikipedia*, our algorithm is beaten by COSC, which works best with very sparse graphs and the *wikipedia* dataset is very sparse. There are also few cases where CSCAP beats FGPWC. But we can also see that CSCAP is not as scalable when the number of clusters is greater than 2.

The algorithm FAST-GE reports poor results. We have expected that this algorithm will require lots of constraints in order to work properly, and in this experiment the number of constraints is rather low.

The low number of constraints can also explain the following result, which can be a bit surprising at a first glance. We can see that there are a number of cases where the algorithm that reports the best Adjusted Rand Index score does not report the least amount of constraint violations. Intuitively, we could think that solutions that violate the least number of constraints would also report the best Adjusted Rand Index score. And here, this is not always the case. For instance, we can see that COSC is able to return partitions with 0 violated constraints when the number of clusters equals 2, however, the partitions are not necessarily close to the ground-truth partition. To a lesser extend, we can also observe this on other datasets where the number of clusters is greater than 2. This also happens with other algorithms (CCSR and CSCAP). An explanation of this phenomenon is that we are providing very few constraints to the different algorithms. Hence, there are many different ways to fullfill the constraints. Some of which are incorrect ways to satisfy the constraints. Another problem is that this is not always possible to determine which algorithm returns the correct result. For instance, despite COSC with 2 clusters often returning partitions whose graph cut score is smaller than that of FGPWC, the latter algorithm returns the correct partition more often than COSC. There are cases, where COSC will cut the graph with a sparser cut (hence a better cut) while satifying all constraints and still get it wrong compared to FGPWC whose graph cut will be less sparse and constraints partially violated. This shows that clustering algorithms with pairwise constraints can be very sensitive to constraints selection and to the structure of the graph.

The algorithm COSC is expecting a sparse graph as an input and satisfies all the constraints when the number of clusters is equal to 2. When the number of clusters is greater than two, COSC looses its guarantees. Moreover, when constraints are very sparse, there are many different ways to satisfy them, and the hierarchical 2-way clustering COSC performing for more than two clusters can achieve very poor results when the earliest cuts are wrong.

It is particularly interesting to compare FGPWC to CCSR, since the approaches developped in the two algorithms are both based on a change of representation of the spectral embedding. CCSR is competitive with FGPWC w.r.t. the ARI measure in many cases. However, we can see that CCSR becomes intractable as the size of the embedding d increases, while this is not a problem for FGPWC. This is also confirmed by the computational time. For algorithms that transform the spectral embedding, such as FGPWC and CCSR, small graphs can be harder if constraints contradict the similarity $\boldsymbol{W}$. The more contradictions between the constraints and the graph, the larger the size of the embedding d must be in order to find dimensions where the constraints can be satisfied. However, with small graphs, the number of dimensions available is also small. When the number of dimensions d increases, these algorithms will find a transformation that cancels all knowledge from the graph and retains only the constraints. So that constraints are indeed satisfied but nothing has been learnt. This is overfitting. This phenomenon is related to the degree of freedom in solving a system of K equations, where K is the fixed number of constraints, with more and more variables (as d increases).

Figure 4.6 shows that, as expected, algorithms SSC:KA and FAST-GE require a lot more constraints to be efficient. In that experiment, we have run FGPWC, SSC:KA and FAST-GE on the *circles* data set with a varying number of - accurate - constraints, selected uniformly at random. For each constraint set, the experiment is conducted 10 times and the average is reported in Figure 4.6. FGPWC was taking too long to compute cases with a number of constraints higher than 15000, but keep in mind these algorithms should not be used when a simple and fast transitive closure is enough to recover the correct partition.

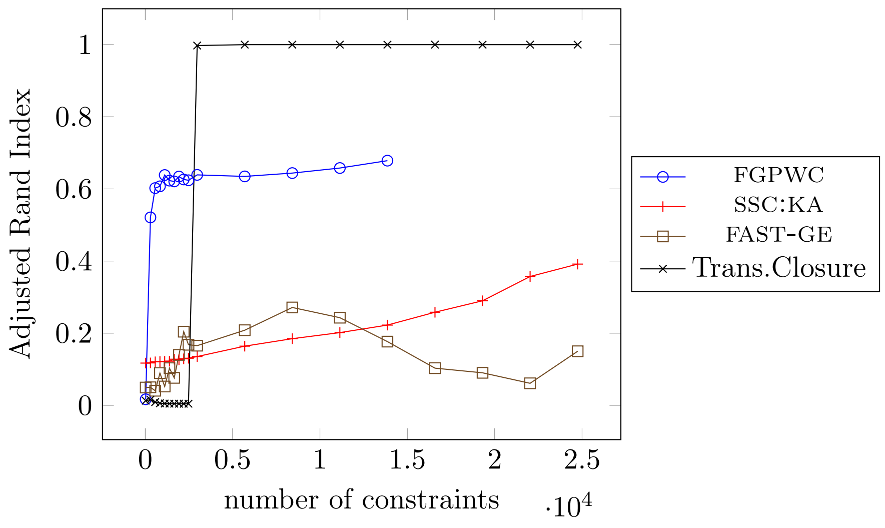


Figure 4.6: We can see that algorithms SSC:KA and FAST-GE require more constraints in order to achieve good results, as expected. FGPWC on the other hand performs better with fewer constraints. FGPWC may have some issues handling a very high number of constraints, but it is not supposed to be used in these cases, as its purpose is to handle data sets with a lower number of constraints. The transitive closure is given as a baseline to help interpret the high number of constraints.

4.5.2 Noun Phrase Coreference Resolution

System settings

Following the approach in [13], we first create for each document a fully connected similarity graph. Parameter estimation for this pairwise mention model was performed using Limited-memory BFGS implemented as part of the Megam package http://www.umiacs.umd.edu/~hal/megam/version0_3/. Default settings were used. Compared to the tasks on the UCI dataset, the main difficulties are the determination of the number

Dataset	k	FGPWC	viol.	SSC:KA	viol.	FAST-GE	viols.	CCSR	viol.	CSCAP	viol.	COSC	viol.
hepatitis	2	**0.1910**	10	0.0224	19	0.1299	6	−0.0127	17	0.1002	21	0.0184	**0**
ionosphere	2	**0.5041**	37	0.0045	86	0.1598	69	-	-	0.0982	70	0.4685	**0**
moons	2	0.9215	19	0.0647	227	−0.0003	176	0.6684	72	**0.9956**	**0**	-	-
promoters	2	**0.7182**	3	0.4274	7	−0.0006	29	0.5946	8	0.4563	**0**	0.3358	**0**
spam	2	0.9783	21	0.0000	1128	−0.0001	1931	0.9783	26	**0.9991**	**0**	-	-
tic-tac-toe	2	**1.0000**	**0**	−0.0010	216	0.0044	188	-	-	0.9916	**0**	-	-
wdbc	2	**0.8568**	**14**	0.0024	131	0.0051	222	0.7255	35	0.2594	92	-	-
xor	2	**1.0000**	**0**	−0.0011	248	0.1368	258	**1.0000**	**0**	**1.0000**	**0**	-	-
hayes-roth	3	0.2783	**3**	−0.0136	24	0.0012	30	0.0842	21	**0.4121**	22	0.0079	12
circles	3	**0.6458**	**53**	0.1164	180	−0.0009	265	-	-	-	-	0.0110	172
iris	3	**0.9410**	**1**	0.6465	9	0.2544	25	0.8485	4	0.3940	20	0.5685	10
wikipedia	3	0.6298	49	0.0009	238	0.6508	43	0.5409	76	-	-	**0.6960**	**33**
wine	3	**0.9649**	**0**	0.9134	2	0.4297	16	0.8566	10	0.5883	16	0.0091	41
imdb	4	**0.1385**	**93**	−0.0001	559	0.0111	477	-	-	-	-	0.0181	298
vehicles	4	**0.3175**	**55**	0.0820	209	0.0165	265	-	-	-	-	0.0038	116
phoneme	5	**0.7073**	**126**	0.0005	987	0.0011	1197	-	-	-	-	-	-
cornell	5	**0.4868**	**13**	0.3152	28	0.2009	14	0.3317	**13**	0.0290	52	0.0577	**13**
texas	5	**0.4705**	11	0.3102	29	0.0226	62	0.2848	25	-	-	0.0707	**9**
wisconsin	5	**0.6719**	**21**	0.4836	30	0.3345	27	0.3346	32	-	-	0.0226	23
breasttissue	6	0.3088	**3**	0.2358	12	0.0462	18	0.2104	5	**0.3720**	4	0.0695	9
glass	6	**0.2552**	**16**	0.1257	45	0.0039	56	0.1872	26	0.1733	46	0.0347	20
zoo	7	**0.9218**	**0**	0.7035	3	0.0833	16	0.7025	2	0.7416	2	0.1447	1

Table 4.4: This table shows results from the experiment we have conducted on the UCI and network datasets. Empty cells correspond to cases where the algorithm have not completed its computation within 15 minutes. The Adjusted Rand Index as well as the number of violated constraints are reported. Bold scores indicate the best results.

of clusters and the fact that we have to deal with many small graphs (documents contain between 1 and 300 mentions).

The same defaut values were used for the σ_m and σ_c parameters, as in the previous experiments (that is, 0.15 and 1.5, respectively). In our aglorithm we need to fix parameter d. We fix a value that is a tradeoff between the dimension of $\boldsymbol{L_{sym}}$ and the number of constraints. Indeed, we want to keep structural information coming from the graph through the eigendecomposition of $\boldsymbol{L_{sym}}$. Also, we reject the situations where d is much larger than the number of constraints because they can lead to solutions that are non satisfactory: when this happens, the optimization problem can be solved without any impact on non-constrained pairs and therefore without any generalization based on the given constraints. Because the multiplicity of eigenvalue 1 is large in this dataset, d is estimated by $d = |\{\lambda_i : \lambda_i \leq 0.99\}|$ where λ_i are the eigenvalues of $\boldsymbol{L_{sym}}$. The number of clusters n is estimated by $n = |\{\lambda_i : \lambda_i \geq 10^{-5}\}|$ where λ_i are the eigenvalues of $\boldsymbol{X}^\top \boldsymbol{X}$.

As for the inclusion of constraints, we experimented with two distinct settings. In the first setting, we automatically extracted based on domain knowledge (setting (c) in the results below). Must-link constraints were generated for pairs of mentions that have the same character string. For cannot-link constraints, we used number, gender, animacy, and named entity type dismatches (e.g., noun phrases with different values for gender cannot corefer). These constraints are similar to some of the deterministic rules used in [41] and overlap with the information already in the features. This first constraint extraction generates a lot of constraints (usually, more than 50% of all available constraints for a document), but it is also noisy. Some of the constraints extracted this way are incorrect as they are based on information that is not necessarily in the dataset (e.g., gender and number are predicted automatically). The precision of these constraints is usually higher than 95%. In a second simulated interactive setting, we extracted a smaller set of must-link and cannot-link constraints directly from the ground-truth partitions, by drawing coreferential and non-coreferential mention pairs at random according to a uniform law (setting (b) below). In turn, all of these constraints are correct. Each mention pair has a probability $1/n^2$ to be drawn, with n the mention count.

Results and discussion

We show that FGPWC works better on large graphs and larger clusters. We perform per-cluster evaluation, this is summarized in Figure 4.7. All plots represent the F1-score, averaged on runs on all documents per cluster size. Plot (a) reports results for the unconstrained spectral clustering approach of [13]. Their method uses a recursive 2-way spectral clustering algorithm. The parameter used to stop the recursion has been tuned on a development set. The other plots are obtained using (b) FGPWC with constraints generated uniformly at random from an oracle and (c) FGPWC with constraints derived automatically from text based on domain knowledge.

In the latter case (c) FGPWC has not been able to improve the results obtained by (a). We think that constraints extracted from text do not add new information but change the already optimized measure in the similarity graph. However, even adding fewer constraints at random from an oracle using a uniform distribution is more informative.

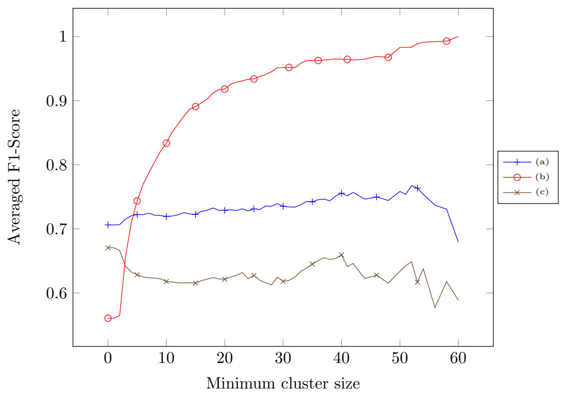


Figure 4.7: Averaged F1-score vs minimum cluster size for FGPWC with CoNLL 2012 data set: (a) method in [13], (b) FGPWC uniformly distributed from reference; (c) FGPWC All extracted must/cannot-links

When we are using constraints that do not come from the features used for the similarity construction step, we see that FGPWC outperforms other methods for clusters larger than 5. However, we can see that FGPWC can degrade smallest clusters. There are two explanations for this: we obtain better performance on larger clusters because of the way we select random constraints. Using a uniform distribution, there is more chance to add constraints for larger clusters. And moreover, clusters with few or no constraints, in our case: small clusters are usually scattered around the space, because FGPWC globally transforms the space to fit the constraints. We can also see that (b) outperforms (c) on small clusters. Probably because more constraints are being added for small clusters in (b). All of this supports the idea that constraints in this kind of task should be generated from another set of features applicable to all mentions, regardless of the size of the clusters they belong to.

Overall, we obtain a CoNLL score of 0.71 for [13], 0.56 using our method along with extracted constraints and 0.58 with a random subset of ground-truth constraints.

Algorithm	CoNLL	MUC	B^3	$CEAF_e$	ARI
[13]	0.71	0.80	0.75	0.57	0.48
FGPWC Extracted	0.56	0.76	0.57	0.36	0.31
FGPWC Ground-truth subset	0.58	0.67	0.58	0.49	0.40

That is, we see a clear drop of performance when using the constraints, be they noisy or not. Closer examination reveals that this decrease stems from poor performance on small clusters, while these clusters are the most representative in this task.

The F1-score is lower than for the state of the art. But interestingly, in presence of uniformly distributed pairwise constraints, our algorithm can significantly improve clustering results on clusters larger than 5, compared to the state of the art [13]. This suggests that active methods can lead to dramatic improvements and our algorithm easily supports that through the introduction of pairwise constraints. Moreover, our method can be used to detect larger clusters, and leave the smaller clusters to another method.

Chapter 5

Partition-level constraints

Contents

So far we have discussed strong forms of supervision. Label constraints are particularly strong and costly to obtain. Although pairwise constraints are weaker and cheaper than label constraints. Pairwise constraints are also a strong form of supervision and can be costly to obtain, because this kind of supervision constraints local parts of the graph to be in a particular state. In other words, a local pair of nodes should be (or not be) in the same cluster. An even weaker form of supervision exists and applies on the whole partition instead of applying to parts of it. In this chapter we study partition-level constraints, in particular those that constraint cluster sizes of a partition to follow a power-law distribution.

A power-law is a functional relationship between two quantities, where one quantity varies as a power of the other, independently of the initial size of those quantities. An example is the relation between the length and the area of the square. If the length is doubled, the area is multiplied by a factor of four. The area of the square varies proportionally to its length raised to the power 2. The power-law distribution appears in many different places. According to Li [46], power-law distributions happen in word usage in human languages, city populations, webpage visits and other internet traffic data, company sizes and other economic data, science citation and other bibliometric data, and many other natural and physical phenomenons.

In particular in this thesis, we are interested in processing human language (or natural language). In a text document, few words are used very often, while most words are rarely used. This is also true for topics of a document. In a document, there are usually many references to few topics and few references to many topics. This can be explained by the fact that when discussing about one topic, few references to many other topics have to be made. For example, in the novel ‘Twenty Thousand Leagues Under the Sea’, many references to the sea, to the Nautilus or the Captain Nemo are made. In order to discuss about these themes, Jules Verne introduces few references to many other themes, describing for example the position of Captain Nemo regarding the rest of the humanity. It would be very appealing to have an algorithm that could determine what the different parts of text refer to. In the previous example, such an algorithm could determine that Nemo is the Captain of the Nautilus. That he does not really like humanity and he decided to live in exile, under the sea.

In this chapter, we discuss clustering algorithms that attempt to do exactly that: partition a data set in such a way that cluster sizes follow a power-law distribution. In Section 5.1 and Section 5.1.3, we explain what a power-law distribution is and present how the Pitman-Yor Process can be used to generate partitions whose cluster sizes follow a power-law distribution. In the Section 5.2, we present the work from Zhou *et al.* [90], we present the underlying generative model for this proposal in Section 5.2.1, the algorithm in Section 5.2.2 and we present a first contribution about the limits of this approach in Section 5.2.3. Then, in Section 5.3, we present a second contribution on clustering power-law distributed data sets in one pass. In Section 5.3.2, we present a first proposal towards learning the parameters used in the model presented in Section 5.3.

5.1 Power-law distributions

As said in Newman [63], “many things that scientists measure have a typical size or ‘scale’ – a typical value around which individual measurements are centered”, and for which the variation is comparatively small. The author gives the example of heights of human beings and argues that “most adult human beings are about 180cm tall”, and “we never see people who are 10cm tall, or 500cm.” In machine learning many problems are formulated around this idea. Some examples are k-means, spectral clustering, support vector machines and logistic regression.

- In k-means individual observations belonging to the same cluster are centered around a typical value called centroid.
- Graph cuts can be formulated as running k-means algorithm in a kernel space, therefore this notion of typical value arises.

- Support vector machines and logistic regression attempts to find a linear separator that is representative of the typical frontier between two states of properties describing two families of objects.

This kind of data is said to follow a normal distribution. However not all things follow a normal distribution. Some vary over an enormous dynamic range, sometimes many orders of magnitude. Classic examples are the size of towns and cities, the frequency of words in human language, or the size of clusters in coreference tasks. Each of these examples have an histogram whose plot on a log-log scale is a straight line. The latter is illustrated in Figure 5.1.

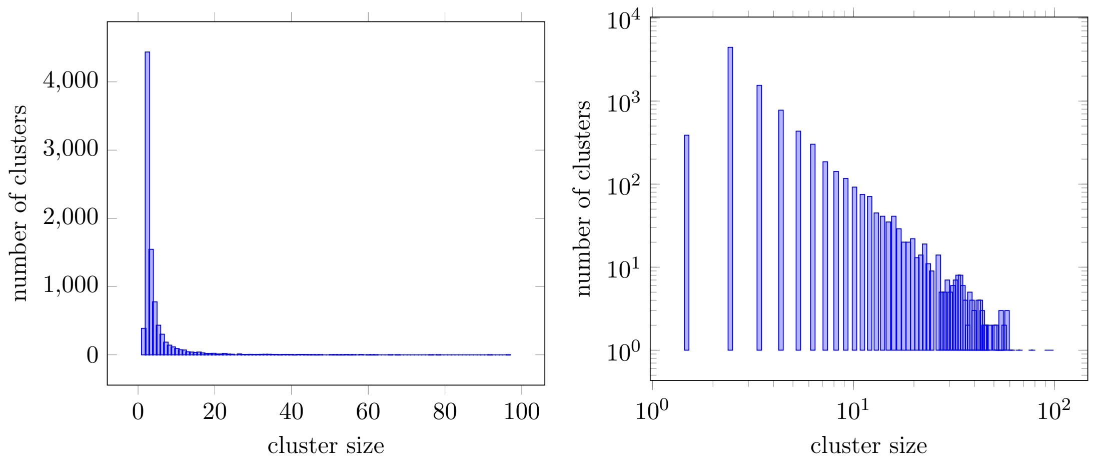


Figure 5.1: Histogram plots of the number of clusters in ConLL shared task 2012. On the left, the histogram on a linear scale. On the right, the histogram is on a loglog scale. This corresponds to the number of times an entity is referenced in the text.

In other words, if $p(x)$ follows a power-law distribution, then $\ln p(x) = -\alpha \ln x + c$ is a straight line, where α and c are constants. Or equivalently, we have:

$$p(x) \propto L(x)x^{-\alpha} \tag{5.1.1}$$

where $\alpha > 1$ and $L(x)$ is a slowly varying function, that is $L(x)$ satifies $\lim\limits_{x\to\infty} \frac{L(rx)}{L(x)} = 1$ for all $r > 0$. This condition on $L(x)$ follows from a requirement that $p(x)$ must be asymptotically scale invariant. The form of $L(x)$ only controls the shape and finite extent of the lower tail. For example, if $L(x)$ is a constant function, then we have $p(x) = Cx^{-\alpha}$, where the constant $C = \exp^c$. In such case the power-law holds for all values of x. Sometimes, it is useful to have a lower bound $x_{\min}$ from which the power-law holds. Then, the power-law has the form

$$p(x) = \frac{\alpha - 1}{x_{\min}} \left(\frac{x}{x_{\min}} \right)^{-\alpha} \tag{5.1.2}$$

where $\frac{\alpha-1}{x_{\min}}$ is the normalization constant. Power-law distributions happen in an extraordinary diverse range of phenomena: word frequency in human languages [92], size of earthquakes [29], moon craters [60], avalanches and solar flares [49], Internet traffic [17] and popularity of web pages [24], wars [75], culture and family names [88], scientific production [47] and citations [20], market in the world wide web [32], market share and distribution of bestsellers in book, popular music and almost every branded commodity [76, 16], the evolution of biological species [84].

5.1.1 Some properties of power-laws

First of all, power-laws are scale invariant. That is, given the relation $f(x) = Cx^{-\alpha}$, scaling the parameter x by a constant factor k only causes a proportional scaling of the function itself: $f(kx) = C(kx)^{-\alpha} = k^{-\alpha}f(x) \propto f(x)$.

Secondly, power-laws whose parameter α is less than 2 do not have a well defined mean:

$$\mathbb{E}(x) = \int_{x_{\min}}^{\infty} xp(x)\,\mathrm{d}x = C\int_{x_{\min}}^{\infty} x^{-\alpha+1}\,\mathrm{d}x = \frac{C}{2-\alpha}\left[x^{-\alpha+2}\right]_{x_{\min}}^{\infty}$$

It is obvious that for $\alpha < 2$, the average value of x diverges. The meaning of this is rather well explained in Newman [63] and is reported in the following:

> What does it mean to say that a distribution has an infinite mean? Surely we can take the data for real solar flares and calculate their average? Indeed we can and necessarily we will always get a finite number from the calculation, since each individual measurement x is itself a finite number and there are a finite number of them. Only if we had a truly infinite number of samples would we see the mean actually diverge. However, if we were to repeat our finite experiment many times and calculate the mean for each repetition, then the mean of those many means is itself also formally divergent, since it is simply equal to the mean we would calculate if all the repetitions were combined into one large experiment. This implies that, while the mean may take a relatively small value on any particular repetition of the experiment, it must occasionally take a huge value, in order that the overall mean diverge as the number of repetitions does. Thus there must be very large fluctuations in the value of the mean, and this is what the divergence [of the average value] really implies. In effect, our calculations are telling us that the mean is not a well defined quantity, because it can vary enormously from one measurement to the next, and indeed can become arbitrarily large. The formal divergence of [the average value] is a signal that, while we can quote a figure for the average of the samples we measure, that figure is not a reliable guide to the typical size of the samples in another instance of the same experiment.

The same can be done for higher moments, in particular there is no meaningful mean square for $\alpha < 3$, thus no well defined variance nor standard deviation:

$$\mathbb{E}(x^2) = \frac{C}{3-\alpha}\left[x^{-\alpha+3}\right]_{x_{\min}}^{\infty}$$

Most identified power-laws have exponents between 2 and 3. Thus, the mean of these power-laws is well-defined, but the variance is not. This implies that these power-laws can occasionally produce very large values, even though their mean is well-defined. This behavior is often referred as “black swan” behavior, 80-20 rule, Zipf’s law, 90-9-1 principle, etc.

Finally, Newman [63] shows that the average value of the largest value x can take is: $x_{\max} \sim n^{1/(\alpha-1)}$. Thus, the maximal value of x always increases as n becomes larger.

From these properties, we can see how different power-law distributions and normal distributions are. In the next section, we will see that determining if a distribution follows a power-law and estimating parameters of a power-law are both difficult.

5.1.2 Identifying and measuring power-law distributions

In the first part of this section, I will discuss the Kolmogorov-Smirnov type test, which is used to validate whether a particular probability distribution is identical to an hypothesized probability distribution. Then I will discuss estimators for the power-law distribution. Estimators based on the linear fit of the histogram plot of the distribution on loglog scale are widely used, because this plot on the loglog scale is a straight line. We will review this method and explain why using this kind of estimator is a mistake. Our explanation will also show how easy it is to mistake a lognormal distribution for a power-law distribution. In the last part of this section, I will present the maximum likelihood estimator which is unbiased.

Kolmogorov-Smirnov type test

The Kolmogorov-Smirnov type test (or KS test) is a nonparametric test of continuous, one-dimensional probability distributions that can be used to compare a sample probability distribution with an hypothesized probability distribution. Figure 5.2 illustrates the KS test. The Kolmogorov- Smirnov type test consists in computing the maximal distance between the sample cumulative probability distribution $P_{emp}(x)$ and the hypothesized cumulative probability distribution $P_\alpha(x)$. Formally:

$$D_\alpha = \max_x |P_{emp}(x) - P_\alpha(x)|$$

If D_α is small, then we can consider that samples from $P_{emp}(x)$ have been drawn from a distribution equal to $P_\alpha(x)$. Intuitively, two probability distributions are the same if their cumulative distributions do not differ too much.

Notice that it is always possible to estimate the parameters of a power-law distribution using the KS test. It suffices to minimize D_α over α.

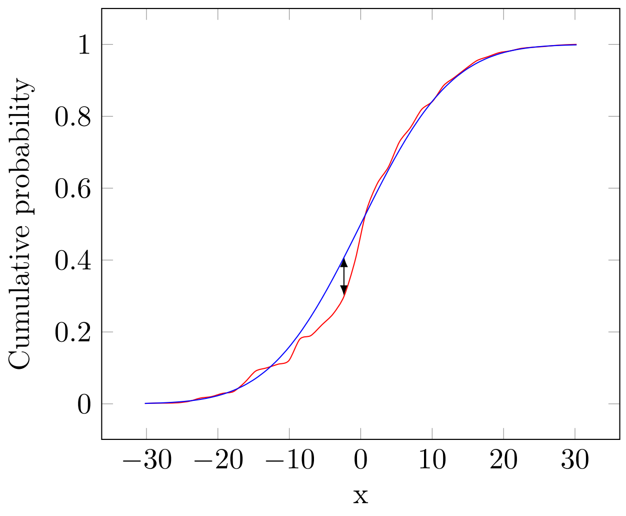


Figure 5.2: Illustration of the Kolmogorov-Smirnov type test. The red line is the sample cumulative distribution. The blue line is the hypothesized cumulative distribution. The black double-sided arrow is the value of the KS test.

Biased estimators

The plot on a log-log scale of the power-law probability density distribution function is a straight line. Because of that, many researchers [3, 2, 62, 61, 35] have estimated power-law distributions parameters using simple graphical method. Among them: direct linear fit of the log-log plot of the full raw histogram, fit of the first 5 points of the log-log plot of the raw histogram or linear fit to the logarithmically binned histograms. These methods seem justified by the straight line of the plots on a log-log scale. However, power-law distributions do not satisfy the assumptions made by linear regression models [72], in particular:

- measured variables are homoscedastic: the measured variables must have the same variance in their errors. In case of power-law distributed random variables, we have seen in Section 5.1.1 that there is no meaningful variance for power-laws whose exponent α is smaller than 3. This means that the variance of different measures can vary greatly, in other words random variables drawn from such power-law are heteroscedastic.

- measured variables must follow a normal conditional distribution, which is clearly not the case for power-law distributions: estimating the power-law parameters using a direct fit of histogram plots on a log-log scale implicitly makes the assumption that the random variables drawn from the power-law distribution actually follow a lognormal distribution.

In the following experiment, two ideas are highlighted. First, estimators based on a linear fit of the log-log plot introduce an error in the parameter estimation and therefore these estimators should not be used. Second, when the exponent parameter α is less than 2, the Kolmogorov-Smirnov Type test is not reliable. The experiment is as follows: draw n samples from a power- law distribution with exponent α, then use the different methods based on linear regression model to estimate α. Some results of this experiment are reported in Table 5.1.

Estimator	Exponent α	Estimated α	Error	KS-type test
Linear	1.75	4.69	168%	0.95
Linear 5-points		0.87	50%	0.94
Logarithmic bins		2.66	52%	0.89
MLE		1.7498	0.01%	0.31
Linear	2.5	13.39	436%	0.35
Linear 5-points		0.52	79%	0.21
Logarithmic bins		5.12	105%	0.29
MLE		2.5016	0.06%	0.01
Linear	3.5	15.03	329%	0.16
Linear 5-points		3.31	5%	0.02
Logarithmic bins		8.53	144%	0.13
MLE		3.4975	0.07%	4.5×10^{-4}

Table 5.1: Some results regarding the estimation of power-law parameters. We can see that estimators based on linear regression models report values with large errors, whereas maximum likelihood estimator (MLE) constantly reports values very close to the value used to generate the data. We can also see that the KS type test for MLE and $\alpha = 1.75$ is particularly large, unlike for values of α not lesser than 2. This is a consequence of the absence of a well defined mean for power-law distributions with exponent lesser than 2.

We can see that despite the bias in all these methods, the second method consisting in fitting a line on the first 5 points of the histogram reports the most accurate results among these models based on linear regression. The fact that most of the data is aggregated in the left-most part of the histogram plot explains this result. The right-most part of the histogram represents those very few “black swans” points: points whose value is suddenly very large compared to the majority of others points. Comparatively, log-normal distributions have a very low probability of producing very large values relatively to the others. Also notice reported values for the Kolmogorov-Smirnov type test. In Table 5.1, the KS-type test fails for every estimator and exponent, except for the maximum likelihood estimator when α is not less than 2. Although this is not surprising for linear regression based estimators, the KS-type test takes a particularly large value for maximum likelihood estimator and $\alpha = 1.75$. The average value produced by power-law distributions diverges when the exponent α is less than 2. One consequence of this is that the KS- type test becomes irrelevant, because in that case arbitrary large values can be drawn from the probability distribution.

Unbiased estimator and validation of power-law distributions

In Newman [63], maximum likelihood estimator (MLE) is proposed as an alternative, simple and reliable method to estimate the exponent α for identically and independently distributed data. For power-law distributions of the form described in Equation (5.1.2), the log likelihood function is:

$$\mathcal{L}(\alpha) = \log \prod_{i=1}^{n} \frac{\alpha - 1}{x_{\min}} \left(\frac{x_i}{x_{\min}} \right)^{-\alpha}$$

The maximum likelihood is then found by setting $\nabla_\alpha \mathcal{L}(\alpha) = 0$. This gives the maximum likelihood estimator for power-law distributions of the form depicted in Equation (5.1.2):

$$\widehat{\alpha} = 1 + n \left(\sum_{i=1}^{n} \ln \frac{x_i}{x_{\min}} \right)^{-1} \tag{5.1.3}$$

Then, we can perform a Kolmogorov-Smirnov type test in order to check the validity of the hypothesis that a set of observations follows a power-law distribution with the estimated exponent $\widehat{\alpha}$.

5.1.3 Pitman-Yor Process

Our goal consists in designing a clustering algorithm that outputs clusters whose sizes are power-law distributed. To that end, we use the Pitman-Yor Process, also called two parameters Poisson-Dirichlet process, which is a stochastic process. A random sample from this process follows a power-law distribution. The name “Pitman-Yor process” was introduced in Ishwaran and James [34] after the review Pitman and Yor [71] and was originally studied in Perman *et al.* [68], Perman [69]. The Pitman-Yor Process is also an extension of the Chinese Restaurant Process.

The Chinese Restaurant Process is as follows: customers enter a restaurant with an infinite number of tables (each table corresponds to a cluster). The first customer sits at the first table. Subsequent customers sit at occupied tables with probability proportional to the number of seated customers at that tables, and sit at a new table with probability α.

The Pitman-Yor Process introduces a new variable θ: the first customer sits at the first table. Subsequent customers sit at occupied tables with probability proportional to the number of occupants minus θ, where $\theta \in [0, 1)$, and sit at a new table with probability proportional to $k\theta + \alpha$, where k is the current number of occupied tables. As the number of tables k increases, there is a higher probability to sit at a new table, producing power-law distribution of cluster sizes.

Formally, we will denote the Pitman-Yor process with parameters α and θ as $\mathrm{PYCRP}(\alpha, \theta)$.

5.2 Power-Law Graph Cuts

In this section we discuss about the work of Zhou *et al.* [90], where an algorithm based on weighted kernel k-means is designed to perform cuts in a graph such that cluster sizes follow a power-law distribution. We present the generative model and its connection to the Pitman-Yor process, which justifies this approach in Section 5.2.1 and the algorithm used to solve this problem in Section 5.2.2. Then, we present the limitations of this approach in Section 5.2.3.

5.2.1 A generative model for power-law distributed clustering instance

In [90], the authors consider that data come from a generative model where cluster sizes are first determined by a Pitman-Yor process, then well-separated cluster centroids are chosen for each cluster. Finally data points are drawn from a normal distribution centered on each cluster centroid. They consider the following Bayesian non-parametric generative model:

$$\begin{aligned} \boldsymbol{Z} &\sim \mathrm{PYCRP}(\alpha, \theta) \\ \boldsymbol{x}_i &\sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{Z}_i}, \frac{\sigma}{2w_i}\boldsymbol{I}), \quad i = 1, \dots, n, \end{aligned} \tag{5.2.1}$$

where $\boldsymbol{Z}$ is an assignation of n objects to a number of clusters determined by a Pitman-Yor Chinese Restaurant Process with parameters α and θ (see Section 5.1.3), $\boldsymbol{x}_i$ is a vector of coordinates for the ith data point, $\boldsymbol{\mu}_{\boldsymbol{Z}_i}$ is the mean of the normal distribution for the cluster assigned to $\boldsymbol{x}_i$, and w_i is a weight associated with the ith data point. The set of all observations is $\mathcal{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$. To better understand this generative process, recall that the authors extend weighted kernel k-means algorithm, whose objective function is

$$\sum_{c=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{A}_c} w_i \left\|\phi(\boldsymbol{x}_i) - \boldsymbol{\mu}_c\right\|^2 \quad \text{where } \boldsymbol{\mu}_c = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{A}_c} w_i \phi(\boldsymbol{x}_i)}{\sum_{\boldsymbol{x}_i \in \mathcal{A}_c} w_i}$$

The weights w_i are given while the cluster centroids $\boldsymbol{\mu}_c$ are determined during the run of the algorithm. Also notice that in weighted kernel k-means, the data points come from a kernel, which is represented here by the function ϕ. For simplification purposes, in this presentation of power law graph cuts, there is no such kernel function. However, it can be easily integrated by letting $\boldsymbol{K}_{ij} = \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)$ and substituting $\left\|\phi(\boldsymbol{x}_i) - \boldsymbol{\mu}_c\right\|^2$ with

$$\left\|\phi(\boldsymbol{x}_i) - \boldsymbol{\mu}_c\right\|^2 = \boldsymbol{K}_{ii} - 2\frac{\sum_{\boldsymbol{x}_j \in \mathcal{A}_c} \boldsymbol{\alpha}_j \boldsymbol{K}_{ij}}{\sum_{\boldsymbol{x}_j \in \mathcal{A}_c} \boldsymbol{\alpha}_j} + \frac{\sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in \mathcal{A}_c} \boldsymbol{\alpha}_j \boldsymbol{\alpha}_l \boldsymbol{K}_{jl}}{\left(\sum_{\boldsymbol{x}_j \in \mathcal{A}_c} \boldsymbol{\alpha}_j\right)^2}.$$

Then we can use the kernel $\boldsymbol{K}$ conjointly with corresponding node weights w_i presented in Table 5.2.

Objective	Node weights	Kernel
Ratio-Association	$\mathbf{1}$	$\sigma \boldsymbol{I} + \boldsymbol{W}$
Ratio-Cut	$\mathbf{1}$	$\sigma \boldsymbol{I} - \boldsymbol{L}$
Normalized Cut	Degree of the nodes	$\sigma \boldsymbol{D}^{-1} + \boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{D}^{-1}$

Table 5.2: Kernels used by W-K-k-MEANS in Dhillon *et al.* [22] for solving different graph-based objectives.

The authors would like to find the partition $\boldsymbol{Z}$, along with the number of clusters k and the cluster centroids $\boldsymbol{\mu} = \{\boldsymbol{\mu}_{\boldsymbol{Z}_1}, \ldots, \boldsymbol{\mu}_{\boldsymbol{Z}_k}\}$ that maximize the join likelihood $p(\mathcal{X}, \boldsymbol{Z})$:

$$
\begin{aligned}
&\arg\max_{\boldsymbol{Z},k,\boldsymbol{\mu}} p(\mathcal{X}, \boldsymbol{Z}) \\
\equiv &\arg\min_{\boldsymbol{Z},k,\boldsymbol{\mu}} -\ln\left(\prod_{i=1}^{k} \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{Z}_i}, \frac{\sigma}{2w_i}\boldsymbol{I}\right) p(\boldsymbol{Z}|\alpha,\theta)\right) \\
\equiv &\arg\min_{\boldsymbol{Z},k,\boldsymbol{\mu}} \frac{1}{\sigma}\sum_{c=1}^{k}\sum_{\boldsymbol{x}_i\in\mathcal{A}_c} w_i \left\|\boldsymbol{x}_i - \boldsymbol{\mu}_c\right\|^2 - \ln p(\boldsymbol{Z}|\alpha,\theta)
\end{aligned}
$$

Then the problem becomes:

$$
\arg\min_{\boldsymbol{Z},k,\boldsymbol{\mu}} \sum_{c=1}^{k}\sum_{\boldsymbol{x}_i\in\mathcal{A}_c} w_i \left\|\boldsymbol{x}_i - \boldsymbol{\mu}_c\right\|^2 - \sigma \ln p(\boldsymbol{Z}|\alpha,\theta) \tag{5.2.2}
$$

The probability distribution of $p(\boldsymbol{Z}|\alpha,\theta)$ is somewhat cumbersome (See [70]):

$$
\begin{aligned}
p(\boldsymbol{Z}|\alpha,\theta) &= \frac{[\alpha+\theta]_{k-1,\theta}}{[\alpha+1]_{n-1}} \prod_{c=1}^{k} [1-\theta]_{n_c-1}, \\
\text{where } [x]_{m,a} &= \prod_{i=1}^{m} x + (i-1)a,
\end{aligned}
$$

where the size of the cluster c is denoted by n_c, and $[x]_m$ is defined as $[x]_{m,1}$. However its natural logarithm is more practicable:

$$
\log p\left(\boldsymbol{Z}|\alpha,\theta\right) = \sum_{i=1}^{k-1} \ln(\alpha + i\theta) - \sum_{i=1}^{n-1} \ln(\alpha + i) + \sum_{c=1}^{k}\sum_{i=1}^{n_c-1} \ln(i - \theta).
$$

In this method, the distance between a data point $\boldsymbol{x}_i$ and a cluster centroid $\boldsymbol{\mu}_c$ is regularized by the probability distribution of $p(\boldsymbol{Z}|\alpha,\theta)$.

5.2.2 Biasing weighted kernel k-means to fit the model

The authors propose to optimize Problem (5.2.2) with weighted kernel k-means. The part of the Problem (5.2.2) corresponding to the Pitman-Yor process depends on the partition, that cannot be computed unless the partition exists. The kernel describing the distance between data points does not provide such partition. Thus, the Pitman-Yor regularization cannot be integrated into this kernel. Instead, the authors propose to modify the weighted kernel k-means algorithm, so that it integrates the Pitman-Yor regularization. To do so, the authors propose the Algorithm 16, whose summary follows:

1. initialize weighted kernel k-means such that all data points belong to the same, single cluster.
2. Compute the score $d(\boldsymbol{x}_i, \boldsymbol{\mu}_c)$ for all i and c. How to compute this score will be explained in the following.
3. For all i, reassign $\boldsymbol{x}_i$ to the cluster whose score $d(\boldsymbol{x}_i, \boldsymbol{\mu}_c)$ is the smallest.
4. Recompute the cluster centroids $\boldsymbol{\mu}_c$.
5. Repeat from 2 convergence.

Moving a single point from a cluster to another (possibly new) cluster does not require to compute all the objective function. The regularization corrects the distance computation $w_i \left\|\boldsymbol{x}_i - \boldsymbol{\mu}_c\right\|^2$ for each data point. Let $d(\boldsymbol{x}, \boldsymbol{\mu}_c)$ be the corrected distance for data point $\boldsymbol{x}$ and cluster centroid $\boldsymbol{\mu}_c$. Suppose that at some iteration of weighted kernel k-means, $\boldsymbol{x}_i$ was assigned to the cluster c, then the distance between $\boldsymbol{x}_i$ and the centroid $\boldsymbol{\mu}_{c'}$ of another cluster c' depends on the following cases:

- If $c' = c$, the variation in the probability distribution $p(\boldsymbol{Z}|\alpha, \theta)$ equals zero, then

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_{c'}) = w_i \left\|\boldsymbol{x}_i - \boldsymbol{\mu}_{c'}\right\|^2. \tag{5.2.3}$$

- If $n_c > 1$ and c' is an existing cluster, the cluster count remains the same and the size of the cluster c is reduced by 1, while the size of the cluster c' is increased by 1. The variation in $\ln p(\boldsymbol{Z}|\alpha, \theta)$ is $\ln(n_{c'} - \theta) - \ln(n_c - 1 - \theta)$, therefore

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_{c'}) = w_i \left\|\boldsymbol{x}_i - \boldsymbol{\mu}_{c'}\right\|^2 + \sigma \ln \frac{n_c - 1 - \theta}{n_{c'} - \theta}. \tag{5.2.4}$$

- If $n_c = 1$ and c' is an existing cluster, the cluster count is reduced by 1 and the size of the cluster c' is increased by 1. The variation in $\ln p(\boldsymbol{Z}|\alpha, \theta)$ is $\ln(n_{c'} - \theta) - \ln(\alpha + (k-1)\theta)$, thus

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_{c'}) = w_i \left\|\boldsymbol{x}_i - \boldsymbol{\mu}_{c'}\right\|^2 + \sigma \ln \frac{\alpha + (k-1)\theta}{n_{c'} - \theta}. \tag{5.2.5}$$

- If $n_c > 1$ and c' is a new cluster, the cluster count is increased by 1 and the size of the cluster c is reduced by 1. Moreover, $\boldsymbol{x}_i$ becomes its own cluster centroid and the distance $\|\boldsymbol{x}_i - \boldsymbol{\mu}_{c'}\|^2$ vanishes. The variation in $\ln p(\boldsymbol{Z}|\alpha, \theta)$ is $\ln(\alpha + k\theta) - \ln(n_c - 1 - \theta)$, then

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_{c'}) = \sigma \ln \frac{n_c - 1 - \theta}{\alpha + k\theta}. \tag{5.2.6}$$

- Finally, if $n_c = 1$ and c' is a new cluster, it means we are moving one data point from a singleton to a new singleton. As this achieves nothing, this should not happen in the algorithm. Thus,

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_{c'}) = \infty. \tag{5.2.7}$$

From there, the authors observe that the distance to new clusters is inversely proportional to the number of clusters as well as the distance to an existing cluster and its size. Both these properties are analogous to the Pitman-Yor process and lead to power-law distributed cluster sizes: as more clusters are created, it is easier to create additional clusters, moreover data points are more likely to be assigned to the largest clusters. For normalized cut, the authors use the equivalence with weighted kernel k-means established in Dhillon *et al.* [22] and the corresponding kernel, see Table 4.3.

5.2.3 Limits of this approach

The parameter σ does not really correspond to the width of the Gaussians used to generate the data set. A simple example suffices to understand this: generate a data set using the generative model described in Equation (5.2.1) and position the cluster centroids such that they lie on a circle and the distance between them is large enough (for example 1). Then let $\sigma = 0.001$. Now, you have a data set with k very well separated clusters, whose size describe a power- law. From there, if you run Algorithm 16 with parameter $\sigma = 0.001$, then it happens that every data point becomes a singleton. This happens because if the parameter σ is small enough, then the value of Equation (5.2.6) becomes the smallest among Equations (5.2.3 to 5.2.7), except for Equation (5.2.3) when c is a singleton. Moreover, the value of Equation (5.2.7) is larger than the value of Equation (5.2.6), because in that case, the distance between any cluster centroid is much larger than the absolute value of the regularization.

A deeper analysis of Equation (5.2.6) shows that, except for the initial cluster, this algorithm may not be able to generate clusters larger than 1. Suppose that at Line 5 of Algorithm 16 the data point $\boldsymbol{x}_i$ belongs to a non-singleton cluster (call that cluster the “source cluster”). The data point may be assigned to three different “target clusters”:

1. The source cluster and target cluster are the same. The distance is computed as in Equation (5.2.3).

Algorithm 16: Power-law-means (vector case)

```
Input: Data points $\boldsymbol{x}_i$, weights $w_i$, trade-off parameter $\sigma$, Pitman-Yor
       parameters $\alpha, \theta$
Output: Number of clusters $k$, clustering $\{\mathcal{A}_i\}_{i=1}^k$.
1  begin
2      Init. $k \leftarrow 1$, $\mathcal{A}_1 \leftarrow \{\boldsymbol{x}_i : i = 1, \dots, n\}$, $\boldsymbol{\mu}_1 \leftarrow \frac{\sum_{i=1}^n w_i \boldsymbol{x}_i}{\sum_{i=1}^n w_i}$, $\forall i : \boldsymbol{Z}_i \leftarrow 1$
3      repeat
4          foreach data point $\boldsymbol{x}_i$ and each cluster $c'$ (including a new cluster) do
5              Compute the distance $d(\boldsymbol{x}_i, c')$ according to Equations (5.2.3) to (5.2.7)
6              Assign $\boldsymbol{x}_i$ to the cluster corresponding to the smallest
                regularizeddistance
7              $\boldsymbol{Z}_i \leftarrow \arg\min_{c'} d(\boldsymbol{x}_i, c')$
8              if $\boldsymbol{Z}_i$ corresponds to a new cluster then
9                  $k \leftarrow k + 1$, $\boldsymbol{Z}_i = k$, and $\boldsymbol{\mu}_k = \boldsymbol{x}_i$
10         foreach cluster $\mathcal{A}_c$ do
11             $$\mu_c \leftarrow \frac{\sum_{\boldsymbol{x} \in \mathcal{A}_c} w_i \boldsymbol{x}_i}{\sum_{\boldsymbol{x} \in \mathcal{A}_c} w_i}$$
12     until until convergence
```

2. The target cluster is another existing cluster. The distance is computed as in Equation (5.2.4).

3. The target cluster is new. The distance is computed as in Equation (5.2.6).

Equation (5.2.6) is smaller than Equations (5.2.3) and (5.2.4) when:

$$\begin{cases} (\alpha + k\theta) \exp \frac{w_i \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2}{\sigma} > n_c - 1 - \theta & \text{For Equation (5.2.3)} \\ (\alpha + k\theta) \exp \frac{w_i \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2}{\sigma} > n_c' - \theta & \text{For Equation (5.2.4)} \end{cases} \tag{5.2.8}$$

Therefore, to avoid systematic creation of singletons, the minimal size of the cluster c (or c') has to grow exponentially with the distance between its centroid and the data point processed on line 5 of Algorithm 16. Once the first data points moved out of the initial cluster, there is one big cluster and multiple singletons. Following data points can either remain in the big initial cluster, create a new singleton or join another singleton. As the number of clusters k increases, it also becomes harder to not satisfy the condition (5.2.8). In order to create a new cluster whose size is greater than 1, the algorithm first begins to move a data point to a singleton, forming a cluster of size 2. This happens when $(\alpha+k\theta) \exp \frac{w_i \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2}{\sigma} < 1-\theta$. The most favorable case happens right after the beginning of the algorithm, once a data point has moved out of the initial single cluster. In that case $k = 2$. Also suppose that $\boldsymbol{x}_i$ and $\boldsymbol{\mu}_c$ are at the same location, that is $\exp \frac{w_i \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2}{\sigma} = 1$. If these two points are not at the same location, then the exponential increases, worsening the situation. We have

$$(\alpha + 3\theta) < 1. \tag{5.2.9}$$

This shows the parameters α and θ must be chosen with great care and must be small enough, so that the condition (5.2.9) is satisfied. Otherwise, clusters of size 2, and larger, cannot happen. However, the expected number of clusters under a Pitman-Yor chinese restaurant process is $\mathcal{O}(\alpha n^{\theta})$. Therefore, either this number of clusters is expected to be very low, either the number of data points must be extremely large.

In Zhou *et al.* [90], the authors generate a synthetic power-law graph to experiment their algorithm. They first generate cluster assignments using a Pitman-Yor process with parameters $\alpha = 1$ and $\theta = 0.2$. Then they use a standard stochastic block model to obtain a random graph. The stochastic block matrix is sampled from two Gaussian distributions:

- $\mathcal{N}(0.3, 0.001)$ for diagonal block entries, and
- $\mathcal{N}(0.01, 0.001)$ for off-diagonal entries.

The author says that their algorithm report 0.866 NMI, while normalized cuts report 0.687 NMI. This is quite disturbing for two reasons. First, as said previously, in that experiment $\alpha = 1$, so the condition (5.2.8) is not satisfied and consequently their algorithm should not be able to produce anything but singletons. Secondly, with these parameters and this generative model, the similarity $\boldsymbol{W}_{ij}$ between two nodes i and j is almost certain to be greater than 0.295 for pairs of nodes in the same cluster and smaller than 0.015 for nodes in different clusters. Thus, this synthetic data set is trivial and normalized cuts should be able to find the correct partition.

5.3 Clustering Power-Law Distributed Data Sets in One Pass

In this section, we discuss a method improvement inspired by the co-reference resolution task. In our setup, mentions and their relations are represented as a similarity graph. As usual, similar mentions should refer to the same entity. However, what makes a good similarity for this task is a very complex question and the literature is prolific on the subject. Moreover, these datasets are typically small in size (the number of nodes n usually ranges between 2 and 300). These characteristics lead to graphs that are particularly hard to cluster. Nonetheless, few assumptions can help us and are the core of our inspiration:

Power-law Cluster sizes typically follow a power-law distribution. The number of clusters k is usually much smaller than the number of nodes n, and in any case $k \leq n$. Thus, evaluating if cluster sizes follow a power-law distribution or another distribution is a problem, especially for small data sets (like co-reference datasets).

Left context Recovering the correct clusters from left to right seems a reasonable hypothesis. That is, for any node v_i, we should be able to recover the cluster of the node v_i using only information from nodes $v_1, \dots, v_{i-1}$. The intuition behind this comes from the fact that human beings are capable of resolving co-reference sequentially.

Constraints Pairwise constraints can be automatically extracted from the data. However, these constraints may be noisy and filtering bad constraints may be required.

The first part of the problem consists in finding the sparsest cut of a graph, such that cluster sizes follow a power-law distribution. This problem has been studied in Fan *et al.* [27], Zhou *et al.* [90]. However, the methods proposed by these authors present some problems, that have been presented in [90] and at the end of section 5.2. These early works show that designing such an algorithm is a challenge.

That being said, if the structure of the graph naturally follows a power-law distribution, then H. Bisection (Algorithm 1) or even spectral k-means (Algorithm 2 or 3) suffice to recover the clusters and minimize the cut of the graph. This fact is obvious: these algorithms are guaranteed to recover the correct partition, as long as the graph is well structured enough (see Peng *et al.* [67]), regardless of the size of the clusters. In this section, we are interested in cases where the graph is not so easy to partition. In other words, in cases where H. Bisection or spectral k-means do not suffice.

In Zhou *et al.* [90], the authors use a Pitman-Yor process as a regularizer for w-k-k-means and plug it in w-k-k-means algorithm during the computation of the distance to each cluster. The first problem with this approach is that the resulted 'modified distance' can no longer be interpreted as a distance. The second issue is that Pitman-Yor process does not allow customers to change tables once they sat at a table, whereas in this algorithm, everyone is sitting at the same table at the beginning. Then, every customer, in the order decided by the indexes of the corresponding node in the graph, gets the chance to move to other tables, until some stability is achieved.

In our setup, we can take advantage of the left context property to obtain an efficient algorithm inspired by the Pitman-Yor process. A number of papers also takes benefit of the left context property in various ways, but to our best knowledge, using this property with the Pitman-Yor process is a novel idea. Nodes of the graphs are considered as customers and they come in the order of their index in the graph. The left context property ensures that we can decide the assignment of a node using only the previous nodes. Hence, clustering is done sequentially in a single pass. A partition is being constructed by assigning successive nodes to different clusters of the partition. The first node is assigned to the first cluster. Subsequent nodes are assigned to either an existing or a new cluster according to the state of the current partition and the score $\boldsymbol{W}_{ij}$ that represents the preference of node j to attach to node i. This preference can be thought as a probability that i and j are in the same cluster.

In the Pitman-Yor process, when a new node arrives, it is assigned to either an existing cluster c with probability proportional to $n_c - \theta$, either a new cluster with probability proportional to $\alpha + k\theta$, where n_c is the size of the cluster c, and n and k are the number of nodes processed so far and the number of clusters so far. It is an extension of the Chinese Restaurant process that assigns a new node to either an existing cluster c with probability proportional to n_c, either a new cluster with probability α. However, the Chinese Restaurant process can be formulated another way: a new node is either assigned to a new cluster with probability proportional to α, either it is linked to another processed node. Since the number of possible links for each cluster equals the size of the cluster, both formulations are equivalent. In our model, we assign nodes to existing clusters by linking them to already processed nodes. We observed empirically that linking nodes to other nodes instead of assigning nodes to a cluster improves the scores reported by the evaluation measures we use. That is we consider the model in which a node either initiates a new cluster, or links to a previously processed node, rather than a model where a node i is assigned to a cluster represented by some aggregation of the weights, for example a sum: $\sum_{j=1}^{i-1} \boldsymbol{W}_{ij}$.Thus, whenever a new node i is being processed, there are two possible outcomes:

- if there is no cannot-link between nodes i and j, the node is linked to the previous node $j < i$ with probability proportional to

$$f(i \sim j) = \boldsymbol{W}_{ij} \cdot \left(1 - \frac{\theta}{n_j}\right)$$

- otherwise, the node did not connect with any of the previous clusters, but the node is assigned to a new cluster with probability proportional to

$$f(i) = \left(\prod_{j=1}^{i-1} (1 - \boldsymbol{W}_{ij})\right) \cdot (\alpha + k\theta)$$

Algorithm 17 summarizes the procedure.

5.3.1 Experiments with one-pass power-law clustering

Conducting experiments with artificial data sets on one-pass power-law clustering can be difficult because one assumption of this algorithm is the *left context* property, that is: one can recover the correct cluster of some node using only information from preceding nodes. It is difficult to generate a synthetic data set that exhibits this property such that clustering does not become trivial. However, our supposition is that text documents have this property. We use the English dataset used for the CoNLL-2012 shared task [73].

The mention graphs are built from a model of pairwise similarity, which is trained on the training section of CoNLL-2012. The similarity function is learned using logistic

Algorithm 17: One-pass power-law clustering

```
Input: Similarity matrix $\boldsymbol{W} \in [0,1]^{n\times n}$, Cannot-link matrix $\boldsymbol{C}$, Pitman-Yor
       parameters $\alpha, \theta$
1 begin
2     Assign the first node to the first cluster.
3     foreach subsequent node $i$ do
4         $\widehat{j} \leftarrow \arg\max_j f(i \sim j)$ for all $j < i$ such that there is no cannot-link
          between $i$ and any node in the cluster containing $j$.
5         if $f\left(i \sim \widehat{j}\right) > f(i)$ then
6             Link node $i$ and $\widehat{j}$ in the solution.
```

regression, each pair of mentions being described by a set of features. We re-use features that are commonly used for mention pair classification (see e.g., [64],[11]), including grammatical type and subtypes, string and substring matches, apposition and copula, distance (number of separating mentions/sentences/words), gender and number match, synonymy/hypernymy and animacy (based on WordNet), family name (based on closed lists), named entity types, syntactic features and anaphoricity detection.

The systems' outputs are evaluated using the three standard coreference resolution metrics: MUC [80], B^3 [5], and Entity-based CEAF (or CEAF$_e$) [50]. Following the convention used in CoNLL-2012, we report a global F1-score (henceforth, CoNLL score), which corresponds to an unweighted average of the MUC, B^3 and CEAF$_e$ F1 scores. Micro-averaging is used throughout when reporting scores for the entire CoNLL-2012 test. Additionally, we are reporting the adjusted rand index.

We compare our algorithm one-pass power-law clustering (or OPPLC) to the algorithm proposed in Cai [13], which corresponds to hierarchical clustering (H. Bisection). Results from the approach presented in Section 5.2 are omited as they only consist in trivial solutions (a single cluster or only singletons). Parameters for both algorithms have been optimized using a grid search. The results are summarized in Table 5.3. Distribution of the Adjusted Rand Index and the CoNLL-score are reported in Figure 5.3. We can see that OPPLC reports better results than H. Bisection in this setting.

Algorithm	CoNLL-score	ARI
H. Bisection	0.747 ± 0.103	0.551 ± 0.232
OPPLC	0.821 ± 0.103	0.720 ± 0.233

Table 5.3: Summary of the results with ConLL-2012 shared task data set. The average of the CoNLL-score and Adjusted Rand Index over all the documents are reported, as well as the standard deviation.

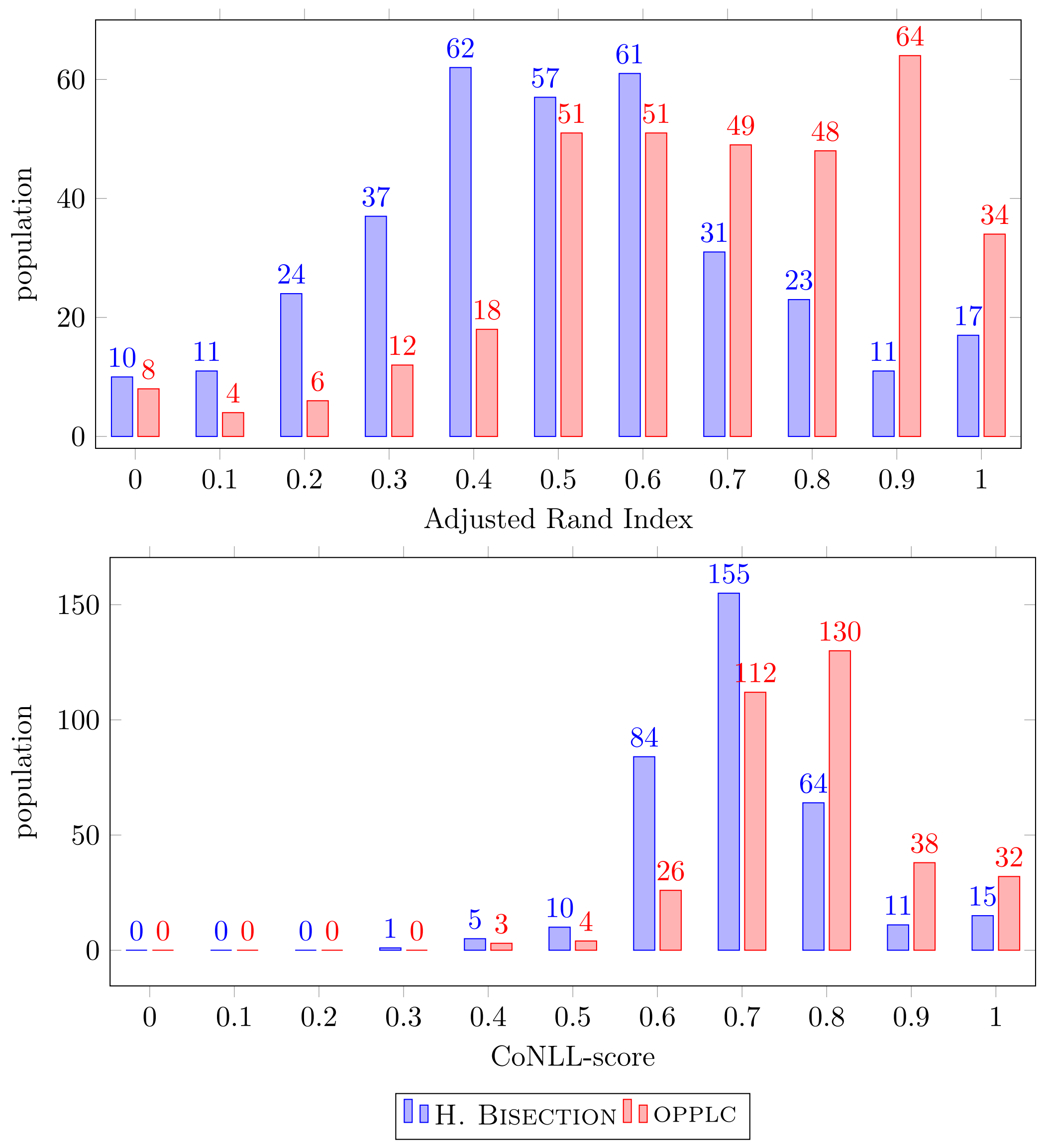


Figure 5.3: Distribution of the Adjusted Rand Index and CoNLL-score reported by H. Bisection and OPPLC across the CoNLL-2012 shared task corpus. We can see that OPPLC reports better results than H. Bisection.

5.3.2 Towards a structured prediction model

In the previous section, we presented an algorithm with hyperparameters α and θ corresponding to the parameters of the Pitman-Yor process. Now, based on Lassalle and Denis [40], we are discussing how to adapt Algorithm 17 to learn Pitman-Yor process parameters. We are also making a shift into how the preferential attachment between nodes is learned. In this new setup, each pair of nodes (i, j) is described by a feature vector $\boldsymbol{x}_{ij}$. The algorithm remains pretty much the same, except that the preferential attachment $\boldsymbol{W}_{ij}$ is given by taking the inner product $\boldsymbol{W}_{ij} = \boldsymbol{w}^\top \boldsymbol{x}_{ij}$, where $\boldsymbol{w}$ is a weight vector and that the criterion for creating new clusters $f(i)$ is no longer based on the preferential attachment of all the other nodes, but on a linear model: $f(i) = \boldsymbol{w}^\top \boldsymbol{x}_i$. After each assignment to a cluster, we check for the truth and update the weight $\boldsymbol{w}$ and the Pitman-Yor parameters α and θ, such that the correct assignment is made. That is, when processing node i, there are three possible cases: wrong assignment, bad cluster creation and bad assignment, that respectively correspond to cases where node i is linked to $\widehat{j}$, but should be linked to j (wrong assignemnt), node i formed a new cluster, but should be linked to j (bad cluster creation), and node i is linked to $\widehat{j}$, but should form a new cluster (bad assignment). On the basis of these three cases, we update the weights $\boldsymbol{w}$ and Pitman-Yor parameters α and θ according to the following rules:

In the following, the node $\widehat{j}$ is described by the feature vector $\widehat{\boldsymbol{x}_{ij}}$ and the node j by the vector $\boldsymbol{x}_{ij}$ and $\boldsymbol{w}_t$, α_t and θ_t are the weights, and the Pitman-Yor process parameters at iteration t.

wrong assignment (node i linked to $\widehat{j}$, but should be linked to j) The node $\widehat{j}$ has been chosen because, according to our algorithm, $f(i \sim \widehat{j})$ was greater than $f(i \sim j)$. That is:

$$f(i \sim \widehat{j}) = \boldsymbol{w}_t^\top \widehat{\boldsymbol{x}_{ij}} \cdot \left(1 - \frac{\theta_t}{n_{\widehat{j}}}\right) > \boldsymbol{w}_t^\top \boldsymbol{x}_{ij} \cdot \left(1 - \frac{\theta_t}{n_j}\right) = f(i \sim j)$$

Remember that $n_{\widehat{j}}$ and n_j correspond to the size of the clusters in which $\widehat{j}$ and j are.

We would like at this point to find weights $\boldsymbol{w}$ and a parameter θ, as close to their previous values as possible and such that $f(i \sim j) > f(i \sim \widehat{j})$. This gives the following problem:

$$\begin{aligned} \boldsymbol{w}_{t+1}, \theta_{t+1} = & \arg\min_{\boldsymbol{w},\theta} \|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + (\theta - \theta_t)^2 \\ & \text{such that}\, \boldsymbol{w}^\top \boldsymbol{x}_{ij} \left(1 - \frac{\theta}{n_j}\right) - \boldsymbol{w}^\top \widehat{\boldsymbol{x}_{ij}} \left(1 - \frac{\theta}{n_{\widehat{j}}}\right) \geq 1 \end{aligned} \tag{5.3.1}$$

bad cluster creation (node i formed a new cluster, but should be linked to j) Here, node i formed a new cluster because $f(i)$ was greater than $f(i \sim \widehat{j})$ for all

$\widehat{j}$, but in particular for $\widehat{j} = j$. Just like in the first case, we update weights and Pitman- Yor process parameters using this formula:

$$\begin{aligned} \boldsymbol{w}_{t+1}, \alpha_{t+1}, \theta_{t+1} = & \arg\min_{\boldsymbol{w},\alpha,\theta} \|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + (\alpha - \alpha_t)^2 + (\theta - \theta_t)^2 \\ & \text{such that } \boldsymbol{w}^\top \boldsymbol{x}_i (\alpha + k\theta) - \boldsymbol{w}^\top \boldsymbol{x}_{ij} \left(1 - \frac{\theta}{n_j}\right) \geq 1 \end{aligned} \tag{5.3.2}$$

bad assignment (node i linked to $\widehat{j}$, but should form a new cluster) The third case is much the opposite of the second case:

$$\begin{aligned} \boldsymbol{w}_{t+1}, \alpha_{t+1}, \theta_{t+1} = & \arg\min_{\boldsymbol{w},\alpha,\theta} \|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + (\alpha - \alpha_t)^2 + (\theta - \theta_t)^2 \\ & \text{such that } \boldsymbol{w}^\top \widehat{\boldsymbol{x}_{ij}} \left(1 - \frac{\theta}{n_{\widehat{j}}}\right) - \boldsymbol{w}^\top \boldsymbol{x}_i (\alpha + k\theta) \geq 1 \end{aligned} \tag{5.3.3}$$

In the following, we will see how to optimize Problems (5.3.1 to 5.3.3). Starting with Problem 5.3.1, the first thing we want to do is to rewrite the problem in matrix form. To that end, we introduce the vector $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{w} & \theta & \alpha \end{bmatrix}^\top$ and we let

$$\boldsymbol{P} = \begin{bmatrix} 0 & \frac{\boldsymbol{x}_{ij}}{n_j} - \frac{\widehat{\boldsymbol{x}_{ij}}}{n_{\widehat{j}}} & 0 \\ \left(\frac{\boldsymbol{x}_{ij}}{n_j} - \frac{\widehat{\boldsymbol{x}_{ij}}}{n_{\widehat{j}}}\right)^\top & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \text{ and } \boldsymbol{q} = \begin{bmatrix} \widehat{\boldsymbol{x}_{ij}} - \boldsymbol{x}_{ij} \\ 0 \\ 0 \end{bmatrix}$$

Then, Problem (5.3.1) becomes:

$$\begin{aligned} \boldsymbol{u}_{t+1} = & \arg\min_{\boldsymbol{u}} f_0(\boldsymbol{u}) = \|\boldsymbol{u} - \boldsymbol{u}_t\|^2 \\ & \text{such that } f_1(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{P} \boldsymbol{u} + \boldsymbol{u}^\top \boldsymbol{q} - 1 \leq 0 \end{aligned} \tag{5.3.4}$$

Now, we can observe that $f_0(\boldsymbol{u})$ is convex and $f_1(\boldsymbol{u})$ is neither convex nor concave, because the matrix $\boldsymbol{P}$ is not positive semi-definite. Indeed, this matrix has 2 eigenvectors whose values are $\pm \left\| \frac{\widehat{\boldsymbol{x}_{ij}}}{n_{\widehat{j}}} - \frac{\boldsymbol{x}_{ij}}{n_j} \right\|$. So, this might be a problem when optimizing Problem (5.3.1). However, strong duality holds for quadratically constrained quadratic problems with a single constraint, thus they can be solved exactly by solving the dual problem. There is a discussion about this in the appendix B of Boyd and Vandenberghe [12]. Problems (5.3.2 and 5.3.3) can also be reformulated in matrix form. They take the same form as Problem (5.3.4) with the following values for matrix $\boldsymbol{P}$ and vector $\boldsymbol{q}$:

Problem (5.3.2)

$$\boldsymbol{P} = -\begin{bmatrix} 0 & \frac{\boldsymbol{x}_{ij}}{n_j} + k\boldsymbol{x}_i & \boldsymbol{x}_i \\ \left(\frac{\boldsymbol{x}_{ij}}{n_j} + k\boldsymbol{x}_i\right)^\top & 0 & 0 \\ \boldsymbol{x}_i^\top & 0 & 0 \end{bmatrix}, \text{ and } \boldsymbol{q} = \begin{bmatrix} \boldsymbol{x}_i \\ 0 \\ 0 \end{bmatrix}$$

Problem (5.3.3)

$$\boldsymbol{P} = \begin{bmatrix} 0 & \frac{\widehat{\boldsymbol{x}_{ij}}}{n_{\widehat{j}}} + k\boldsymbol{x}_i & \boldsymbol{x}_i \\ \left(\frac{\widehat{\boldsymbol{x}_{ij}}}{n_{\widehat{j}}} + k\boldsymbol{x}_i\right)^\top & 0 & 0 \\ \boldsymbol{x}_i & 0 & 0 \end{bmatrix}, \text{ and } \boldsymbol{q} = -\begin{bmatrix} \widehat{\boldsymbol{x}_{ij}} \\ 0 \\ 0 \end{bmatrix}$$

These two problems correspond to quadratically constrained quadratic problems with a single non-convex constraint of the form:

$$\begin{aligned} &\min_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{P}_0 \boldsymbol{x} + 2\boldsymbol{x}^\top \boldsymbol{q}_0 + r_0 \\ &\text{such that } \boldsymbol{x}^\top \boldsymbol{P}_1 \boldsymbol{x} + 2\boldsymbol{x}^\top \boldsymbol{q}_1 + r_1 \leq 0 \end{aligned}$$

In the following section, we will see that we can optimize such problems, as long as the constraint is feasible.

5.3.3 Optimization of single non-convex contraint QCQP

In this section, we discuss the optimization of quadratically constrained quadratic programs with a single non-convex constraint. Letting $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$ be two square matrices that are not necessarily positive semi-definite, $\boldsymbol{q}_0$, $\boldsymbol{q}_1$ be two vectors and r_0 and r_1 be two scalars, we want to optimize the following problem:

$$\begin{aligned} &\min_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{P}_0 \boldsymbol{x} + 2\boldsymbol{x}^\top \boldsymbol{q}_0 + r_0 \\ &\text{such that } \boldsymbol{x}^\top \boldsymbol{P}_1 \boldsymbol{x} + 2\boldsymbol{x}^\top \boldsymbol{q}_1 + r_1 \leq 0 \end{aligned} \tag{5.3.5}$$

To do so, we introduce the Lagrangian $\mathcal{L}(\boldsymbol{x}, \lambda) = \boldsymbol{x}^\top(\boldsymbol{P}_0 + \lambda\boldsymbol{P}_1)\boldsymbol{x} + 2\boldsymbol{x}^\top(\boldsymbol{q}_0 + \lambda\boldsymbol{q}_1) + r_0 + \lambda r_1$ and now the problem consists in optimizing $\max_\lambda \min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \lambda)$. From there, we want that the derivative of the Lagrangian over all its variables vanishes. In particular, we want that $\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}, \lambda) = 0$. Let $\boldsymbol{P} = \boldsymbol{P}_0 + \lambda\boldsymbol{P}_1$, $\boldsymbol{q} = \boldsymbol{q}_0 + \lambda\boldsymbol{q}_1$ and $r = r_0 + \lambda r_1$. We have the following:

$$\begin{aligned} \nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}, \lambda) &= 0 \\ \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q} &= 0 \\ \boldsymbol{x} &= -\boldsymbol{P}^{-1}\boldsymbol{q} \end{aligned}$$

Here, we can see that a first condition to apply this method is that $\boldsymbol{q}$ must lie in the range space of $\boldsymbol{P}$, formally: $\boldsymbol{q} \in \ker(\boldsymbol{P})$. This means that if $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ are the eigenvectors of $\boldsymbol{P}$ then there exists some values $\sigma_1, \ldots, \sigma_n$ such that $\sum_{i=1}^n \sigma_i \boldsymbol{u}_i = \boldsymbol{q}$. Then, we can substitute $\boldsymbol{x}$ in the expression of the Lagrangian to form the dual function $g(\lambda)$:

$$\begin{aligned} g(\lambda) &= (-\boldsymbol{P}^{-1}\boldsymbol{q})^\top \boldsymbol{P}(-\boldsymbol{P}^{-1}\boldsymbol{q}) + 2(-\boldsymbol{P}^{-1}\boldsymbol{q})^\top \boldsymbol{q} + r \\ &= \boldsymbol{q}^\top \boldsymbol{P}^{-1}\boldsymbol{P}\boldsymbol{P}^{-1}\boldsymbol{q} - 2\boldsymbol{q}^\top \boldsymbol{P}^{-1}\boldsymbol{q} + r \\ &= r - \boldsymbol{q}^\top \boldsymbol{P}^{-1}\boldsymbol{q} \end{aligned}$$

Let γ be the optimal value of the Problem (5.3.5). Then we have:

$$\max_{\lambda,\gamma} g(\lambda) - \gamma \geq 0$$

From the Schur complement we have:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^\top & \boldsymbol{C} \end{bmatrix} \succeq 0 \iff \boldsymbol{A} \succeq 0, \boldsymbol{C} \succeq 0, \boldsymbol{C} - \boldsymbol{B}^\top \boldsymbol{A}^{-1}\boldsymbol{B} \succeq 0$$

Thus, we can cast Problem (5.3.5) into a semidefinite program:

$$\begin{aligned} &\max_{\lambda,\gamma} \gamma \\ &\text{such that} \begin{bmatrix} \boldsymbol{P}_0 & \boldsymbol{q}_0 \\ \boldsymbol{q}_0^\top & r_0 - \gamma \end{bmatrix} + \lambda \begin{bmatrix} \boldsymbol{P}_1 & \boldsymbol{q}_1 \\ \boldsymbol{q}_1^\top & r_1 \end{bmatrix} \succeq 0 \end{aligned} \tag{5.3.6}$$

This procedure is an application of the S-procedure to quadratically constrained quadratic programs. The S-procedure is a method first introduced in Lur'e and Postnikov [51]. But most of the results are due to Yakubovich [85, 28, 86]. Appendix B of Boyd and Vandenberghe [12] covers this topic, however more details about the S-procedure can be found in Jönsson [36].

5.3.4 Conclusion on one-pass power-law clustering

We have seen in the previous sections that it is possible to take benefit of the order of the nodes when this order is meaningful. For example in natural language processing, nodes correspond to parts of text and are ordered in the order of the text. Moreover, it is possible to take advantage of the distribution of the cluster sizes, at least when this distribution follows a power-law.

We have also proposed a method to learn the parameters of the Pitman-Yor process used to produce power-laws, as well as the parameters of a linear model used to determine the preferential attachment between nodes. However the results of this algorithm were a bit disappointing as they were not better than our baseline (H. Bisection). We have some intuitions that could explain those results. First of all, the best parameters for the Pitman-Yor process found by the grid were $\alpha = 0$ and $\theta = 0.1$. With these parameters, the Pitman- Yor process does not produce power-law distributions but exponential distributions. This may be an effect of the choice made by the designers of the ConLL-2012 shared task: singletons are removed from the analysis. So, the tail of the distribution followed by the cluster sizes is not very long, as it should be with power-law distributions. However, this fits exponential distribution better. Secondly, in our algorithm we update the linear model and the Pitman-Yor parameters after each decision. We update these parameters after processing each node for two reasons. The first one is that the Pitman- Yor process is a sequential process where the decision for each node is made after other decisions are made. So updating the parameters the same way seems relevant. The other reason is that the Problems we need to optimize in order to update the parameters are quadratically constrained quadratic programs with a single non-convex constraint. As we have seen in the previous section, this can be solved exactly only when there is a single constraint. We think that optimizing the parameters of our model after each document could help getting better results. In order to update the parameters of our model after each document, we would need to optimize a problem with several non-convex constraints, this cannot be done efficiently. A naive way would be to optimize each constraint individually until stabilization, however when there are many constraints, the updating process can move the parameters back and forth from different suboptimal regions of the search space. For example with two constraints, there will be two regions, in the first one, the first constraint is satisfied while the other is not and vice versa. In any case, if the problem is feasible, there will be a third region in which both constraints are satisfied, but this region can be far away from the initial point. Then, the closest point satisfying one constraint or the other may not be in this third region. When this happens, the parameters jump from one region to the other, without satisfying all the constraints. We could try to mix all the constraints into a single non-convex constraint, for example by taking the sum of all the constraints, but it does not make sense, as this is equivalent to updating the parameters of our models according to an “average decision” that is not representative of the individual decisions we have to take. Thus, how to update the parameters after each document remains an open question. Another track of improvement could be to update the parameters for each next decision. This remains to be studied as well.

Chapter 6

Conclusion

In this thesis, we focused on three different kinds of weaker and weaker supervisions for clustering algorithms: label constraints, pairwise constraints and cluster-level constraint.

Label constraints consist in labeling some nodes of the graph with a cluster indicator. This additional knowledge about the nodes can result in huge improvements compared to the unsupervised solutions, however obtaining label constraints can be costly. We proposed an algorithm NOA-SSC, based on Mavroeidis [54], to handle label constraints. For each cluster, it considers label constraints of this cluster against all the constraints from other clusters. This procedure returns a cloud of data points in a space that are then projected onto the unit sphere. Finally k-means is used to obtain a clustering. Experiments have shown that this approach is more effective than label propagation or the proposal of Mavroeidis [55] which also attempts to handle clustering with this kind of constraint for more than 2 clusters.

Pairwise constraints consist in labeling edges of the graph with a must-link or a cannot-link constraint. Must-link constraints are transitive, as long as they are all correct. However, as cannot-link constraints are not transitive, for a number of clusters greater than two, and acts as a repulsive force, introducing them into an optimization problem often implies working with non-convexity. Because of that, state of the art methods cannot guarantee to satisfy all pairwise constraints, unless we make the assumption that there are only two clusters in the solution. Moreover, in applications, like coreference, where the size of the clusters should follow a power-law distribution, working with graph cuts or pairwise constraints is harder because graph cuts tend to balance the size of the clusters and pairwise constraints work best when clusters are evenly supervised. That is smaller clusters have less chance to be supervised than bigger clusters and unsupervised clusters can deteriorate the solution. To handle pairwise constraints, we first proposed MCL-NOA-SSC, which is an extension of NOA-SSC. It consists in propagating the pairwise constraints through a transitive closure of the must-link constraints, which implies that must-links are assumed to be all correct. Then converting them into label constraints which are then softened with NOA-SSC to obtain a clustering. This first approach to handle pairwise constraints was not very convincing, because of the strong assumption on must-link constraints and the results were not very satisfying on some noisy data sets. Then we designed FGPWC, which is based on learning a linear transformation of the spectral embedding, so it could handle pairwise constraints without making strong assumptions on either must-links or cannot-links constraints. The only assumptions made are that, in the transformed embedding, must-linked nodes should be close and cannot-linked nodes should not be close. We also wanted that it could handle noise more efficiently. We implemented the notion of closeness by using Gaussian functions that can be defined either as global for must and cannot-links, separately for must-links and cannot-links or locally for each constraint. Learning the transformation is done using a gradient descent, which empirically proved to be faster than another method sharing a similar approach, Li and Liu [44], that use semi-definite programming. Experiments have shown that our algorithm, FGPWC, is more effective than the other algorithms on most data sets. Among the algorithms that report the best results, it is also the fastest algorithm. The experiments also showed that the selection of pairwise constraints can have a huge impact on the results. In particular, in some cases, the algorithm COSC reported solutions with fewer pairwise constraint violations than FGPWC, but with a much lower ARI score. This implies that multiple solutions can satisfy the pairwise constraints.

Sometimes the domain knowledge can be used to generate pairwise constraints automatically, but it does not guarantee that all the constraints will be correct, nor the constraints will be uniformly distributed. Even though we can use precise generators (with a precision greater than 90%), the few incorrect constraints can be enough to deteriorate solutions given by methods that satisfy them all to the point that simpler methods, like a hierarchical graph cut, can return better solutions. This problem is amplified by the fact pairwise constraints obtained using automatic generators may not be uniformly distributed. For example, in text applications, it often happens that when a pronoun gets a constraint, all the neighboring pronouns also get the constraint. It creates a case where pronouns are overly supervised, while other types of mentions are insufficiently supervised. We have seen earlier that non-uniformly distributed constraints make clustering with pairwise constraints more difficult.

Finally, cluster-level constraints consist in constraining the partitions obtained by clustering algorithms so that they follow specific properties. In this thesis, we have studied a constraint that tends to produce power-law distributed cluster-size clusters. A first attempt with an adaptation of k-means has shown that this constraint is difficult to handle. However, our inspiration for this constraint comes from natural language processing, where other properties of that kind of dataset can be used: word and mentions order are meaningful and pairwise constraints can be extracted automatically from domain knowledge. We took advantage of these properties to propose an algorithm, OPPLC, that builds a partition by processing mentions of a text sequentially, so that pairwise constraints are satisfied, while dispatching the mentions into clusters using an adaptation of the Pitman-Yor process. All this combined allowed us to build partitions whose cluster sizes follow a power-law distribution. When applied to coreference tasks, this method reported better scores than our baseline. However, this method is still very sensitive to errors in the supervision. Nevertheless, we think that this method is promising. Finding a way to make it less sensitive to supervision errors as well as finding a way to learn its parameters efficiently are good research directions.

We have shown that introducing semi-supervision to clustering algorithms can be done in various ways and lead to solutions more adequate with our expectations. It is even possible to combine different types of supervision to achieve our goals.

However, the selection of constraints can have a huge impact on the results. Automatic constraint generation can be prone to errors which then can drastically deteriorate the solution. Identifying these errors is a challenge. If it is not possible, designing an algorithm that attenuates the effect of these errors is a challenge in presence of small clusters. Another problem is that, sometimes, multiple solutions can satisfy a set of correct constraints. In the case where only one solution is correct, it is important to find a way to anticipate this problem. Finally, we have seen that different algorithms behave better with different datasets. Selecting which algorithm should be used and designing an algorithm that automatically selects this algorithm is an interesting problem.

Bibliography

[1] Charu C. Aggarwal and Chandan K. Reddy, editors. *Data clustering: algorithms and applications*. Chapman & Hall/CRC data mining and knowledge discovery series. Chapman and Hall/CRC, Boca Raton, 2014.

[2] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[3] R. Albert, H. Jeong, and A.-L. Barabási. Internet: Diameter of the world-wide web. 401:130–131, 1999.

[4] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

[5] Amit Bagga and Breck Baldwin. Algorithms for scoring coreference chains. In *In The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*, pages 563–566, 1998.

[6] Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Semi-supervised clustering by seeding. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 27–34, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[7] Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 59–68, 2004.

[8] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2002.

[9] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research*, 7:2399–2434, 2006.

[10] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. Label propagation and quadratic criterion. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-Supervised Learning*, pages 193–216. MIT Press, 2006.

[11] Eric Bengtson and Dan Roth. Understanding the value of features for coreference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 294–303, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[12] Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge University Press, Cambridge, UK ; New York, 2004.

[13] Jie Cai. *Coreference Resolution via Hypergraph Partitioning.* PhD thesis, Heidelberg University Library, 2012.

[14] David Chatel, Pascal Denis, and Marc Tommasi. Fast gaussian pairwise constrained spectral clustering. In *Machine Learning and Knowledge Discovery in Databases*, pages 242–257. Springer, 2014.

[15] F. R. K. Chung. *Spectral Graph Theory.* American Mathematical Society, 1997.

[16] Raymond A. K. Cox, James M. Felton, and Kee H. Chung. The concentration of commercial success in popular music: An analysis of the distribution of gold records. *Journal of Cultural Economics*, 19(4):333–340, 1995.

[17] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, 1997.

[18] Mihai Cucuringu, Ioannis Koutis, and Sanjay Chawla. Scalable constrained clustering: A generalized spectral method. *CoRR*, abs/1504.00653, 2015.

[19] Ian Davidson and S. S. Ravi. Clustering with constraints: Feasibility issues and the k-means algorithm. In *Proc. 5th SIAM Data Mining Conference*, 2005.

[20] Derek J. de Solla Price. Networks of scientific papers. *Science*, 149(3683):510–515, 1965.

[21] Gianna M. Del Corso. Estimating an eigenvector by the power method with a random start. *SIAM Journal on Matrix Analysis and Applications*, 18(4):913–937, 1997.

[22] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. *A unified view of kernel k-means, spectral clustering and graph cuts.* Citeseer, 2004.

[23] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification.* Wiley, New York, 2nd ed edition, 2001.

[24] David Easley and Jon Kleinberg. *Networks, crowds, and markets: reasoning about a highly connected world.* Cambridge University Press, New York, 2010.

[25] Paul Erdos and Alfred Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61, 1960.

[26] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. *Clustering and Information Retrieval*, 11:83–103, 2003.

[27] Xuhui Fan, Yiling Zeng, and Longbing Cao. Non-parametric power-law data clustering. *arXiv preprint arXiv:1306.3003*, 2013.

[28] A. L. Fradkov and V. A. Yakubovich. The s-procedure and the duality relation in convex quadratic programming problems. 1973.

[29] B. Gutenberg and C. F. Richter. Frequency of earthquakes in california. *Bulletin of the Seismological Society of America*, 34(4):185–188, 1944.

[30] Lars Hagen, Student Member, and Andrew B. Kahng. New spectral methods for ratio cut partition and clustering. *IEEE Trans. on Computer-Aided Design*, pages 1074–1085, 1992.

[31] Matthias Hein and Simon Setzer. Beyond spectral clustering - tight relaxations of balanced graph cuts. In *In In Advances in Neural Information Processing Systems (NIPS*, 2011.

[32] Bernardo A. Huberman and Lada A. Adamic. The nature of markets in the world wide web. Computing in Economics and Finance 1999 521, Society for Computational Economics, 1999.

[33] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[34] Hemant Ishwaran and Lancelot F James. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453):161–173, 2001.

[35] J. H. Jones and M. S. Handcock. An assessment of preferential attachment as a mechanism for human sexual network formation. *Proceedings of the Royal Society of London B: Biological Sciences*, 270(1520):1123–1128, 2003.

[36] Ulf T Jönsson. A lecture on the s-procedure. 2001.

[37] Sepandar D. Kamvar, Dan Klein, and Christopher D. Manning. Spectral learning. In *In IJCAI*, pages 561–566, 2003.

[38] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[39] B. Kulis, S. Basu, I. Dhillon, and Raymond J. Mooney. Semi-supervised graph clustering: A kernel approach. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 457–464, Bonn, Germany, 2005.

[40] Emmanuel Lassalle and Pascal Denis. Joint anaphoricity detection and coreference resolution with constrained latent structures. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2274–2280, 2015.

[41] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 28–34, 2011.

[42] James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multi-way spectral partitioning and higher-order cheeger inequalities. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1117–1130, New York, NY, USA, 2012. ACM.

[43] Yixue Li and Luonan Chen. Big biological data: Challenges and opportunities. *Genomics, Proteomics & Bioinformatics*, 12(5):187–189, 2014.

[44] Zhenguo Li and Jianzhuang Liu. Constrained clustering by spectral kernel learning. In *Computer vision, 2009 IEEE 12th international conference on*, pages 421–427. IEEE, 2009.

[45] Zhenguo Li, Jianzhuang Liu, and Xiaoou Tang. Constrained clustering via spectral regularization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 421–428, 2009.

[46] Wentian Li. Zipf's law everywhere. *Glottometrics*, 5:14–21, 2002.

[47] A. J. Lotka. The frequency distribution of scientific production. *Journal of Washingtonian Academy of Science*, 16:317–323, 1926.

[48] Zhengdong Lu and Miguel A. Carreira-Perpinán. Constrained spectral clustering through affinity propagation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[49] E. T. Lu and R. J. Hamilton. Avalanches and the distribution of solar flares. 380:L89–L92, 1991.

[50] Xiaoqiang Luo. On coreference resolution performance metrics. pages 25–32. Association for Computational Linguistics, 2005.

[51] A. I. Lur'e and V.N. Postnikov. On the theory of stability of control systems. *Applied mathematics and mechanics*, 1944.

[52] Ulrike von Luxburg. A tutorial on spectral clustering. *CoRR*, abs/0711.0189, 2007.

[53] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM, 2004.

[54] Dimitrios Mavroeidis. Accelerating spectral clustering with partial supervision. *Data Mining and Knowledge Discovery*, 21(2):241–258, 2010.

[55] Dimitrios Mavroeidis. Mind the eigen-gap, or how to accelerate semi-supervised spectral learning algorithms. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 2692, 2011.

[56] Marina Meilă. Comparing clusterings-an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.

[57] Lesly Miculicich Werlen and Andrei Popescu-Belis. Using coreference links to improve spanish-to-english machine translation. Idiap-RR Idiap-RR-07-2017, Idiap, 2017.

[58] Bojan Mohar, Y. Alavi, G. Chartrand, and O. R. Oellermann. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2:871–898, 1991.

[59] Bojan Mohar. *Some applications of Laplace eigenvalues of graphs.* Springer, 1997.

[60] G. Neukum and B. A. Ivanov. Crater size distributions and impact probabilities on earth from lunar, terrestrial-planet, and asteroid cratering data. In *Hazards Due to Comets and Asteroids*, page 359, 1994.

[61] M. E. Newman and J. Park. Why social networks are different from other types of networks. 68(3):036122, 2003.

[62] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[63] Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[64] Vincent Ng and Claire Cardie. Improving machine learning approaches to coreference resolution. page 104. Association for Computational Linguistics, 2001.

[65] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

[66] Malay K. Pakhira. A modified k-means algorithm to avoid empty clusters. *International Journal of Recent Trends in Engineering*, 1(1), 2009.

[67] Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! In *Proceedings of The 28th Conference on Learning Theory*, pages 1423–1455, 2015.

[68] Mihael Perman, Jim Pitman, and Marc Yor. Size-biased sampling of poisson point processes and excursions. *Probability Theory and Related Fields*, 92(1):21–39, 1992.

[69] Mihael. Perman. *Random discrete distributions derived from subordinators : dissertation.* [M. Perman], University of California at Berkeley, 1990.

[70] Jim Pitman et al. Combinatorial stochastic processes. 2002.

[71] Jim Pitman and Marc Yor. The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25(2):855–900, 1997.

[72] Michael A Poole and Patrick N O'Farrell. The assumptions of the linear regression model. *Transactions of the Institute of British Geographers*, pages 145–158, 1971.

[73] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, CoNLL '12, pages 1–40, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[74] Syama S. Rangapuram and Matthias Hein. Constrained 1-spectral clustering. In *International Conference on Artificial Intelligence and Statistics*, pages 1143–1151, 2012.

[75] David C Roberts and Donald L Turcotte. Fractality and self-organized criticality of wars. *Fractals*, 6(04):351–357, 1998.

[76] Raaj Sah and Rajeev Kohli. Market shares: Some power law results and observations. Working Papers 0401, Harris School of Public Policy Studies, University of Chicago, 2004.

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

[78] Josef Steinberger, Mijail Kabadjov, and Massimo Poesio. Coreference applications to summarization. In Massimo Poesio, Roland Stuckardt, and Yannick Versley, editors, *Anaphora Resolution: Algorithms, Resources, and Applications*, pages 433–456, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[79] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[80] Marc Vilain, John Burger, John Aberdeen, Dennis Connolly, and Lynette Hirschman. A model-theoretic coreference scoring scheme. page 45. Association for Computational Linguistics, 1995.

[81] Ulrike Von Luxburg, Agnes Radl, and Matthias Hein. Getting lost in space: Large sample analysis of the commute distance. *Advances in Neural Information Processing Systems*, 23:2622–2630, 2010.

[82] Dorothea Wagner and Frank Wagner. Between min cut and graph bisection. In *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*, MFCS '93, pages 744–750, London, UK, UK, 1993. Springer-Verlag.

[83] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[84] J. C. Willis and G. U. Yule. Some statistics of evolution and geographical distribution in plants and animals, and their significance. 109:177–179, 1922.

[85] V. A. Yakubovich. S-procedure in nonlinear control theory. 1971.

[86] V.A. Yakubovich. Minimization of quadratic functionals under quadratic constraints and the necessity of a frequency condition in the quadratic criterion for absolute stability of nonlinear control systems. *Soviet Mathematics Doklady*, pages 593–596, 1973.

[87] Luh Yen, Francois Fouss, Christine Decaestecker, Pascal Francq, and Marco Saerens. Graph nodes clustering based on the commute-time kernel. In Zhi-Hua Zhou, Hang Li, and Qiang Yang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 4426, pages 1037–1045, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[88] Damián H Zanette and Susanna C Manrubia. Vertical transmission of culture and the distribution of family names. *Physica A: Statistical Mechanics and its Applications*, 295(1-2):1–8, 2001.

[89] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, pages 321–328. MIT Press, 2004.

[90] Xiangyang Zhou, Jiaxin Zhang, and Brian Kulis. Power-law graph cuts. *arXiv preprint arXiv:1411.1971*, 2014.

[91] Xiaojin Zhu. *Semi-supervised learning with graphs.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[92] George Kingsley Zipf. *Human behavior and the principle of least effort: an introduction to human ecology.* Martino Publishing [u.a.], Mansfield Centre, Conn, 2012.