Chapter 7
Network-Based Semi-Supervised Learning

Abstract In this chapter, we present network-based algorithms that run in the semi-supervised learning scheme. The semi-supervised learning paradigm lies somewhere in-between the unsupervised learning paradigm, which does not employ any external information to infer knowledge, and the supervised learning paradigm, which in contrast makes use of a fully labeled set to train models. Semi-supervised learning aims, among other features, to reduce the work of human experts in the labeling process. This feature is quite interesting especially when the labeling process is expensive and time consuming as in video indexing, classification of audio signals, text categorization, medical diagnostics, genome data, among many other applications. In network-based methods, the graph structure is the main driver in propagating labels from labeled vertices to unlabeled vertices. We show that different techniques apply different criteria in their label diffusion processes, generating, as a result, distinct outcomes. In addition, we discuss some of the shortcomings and benefits of the within-graph semi-supervised learning process, also called transductive learning.

7.1 Introduction

Semi-supervised learning is a learning paradigm concerned with the study of how computers and natural systems, such as humans, learn in the presence of both labeled and unlabeled data [39]. Traditionally, learning has been studied either in the unsupervised paradigm (e.g., clustering, outlier detection), in which all of the data are unlabeled, or in the supervised paradigm (e.g., classification, regression), in which all of the data are labeled. The goal of semi-supervised learning is to understand how combining labeled and unlabeled data may change the learning behavior and to design algorithms that take advantage of such a combination. Semi-supervised learning is of great interest in machine learning and data mining because it can use readily available unlabeled data to improve supervised learning tasks when the labeled data is scarce or expensive. Semi-supervised learning also displays potential when conceived as a quantitative tool to understanding categorical human learning, in which most of the input or received information is self-evidently unlabeled [39].

T.C. Silva, L. Zhao, *Machine Learning in Complex Networks*,
DOI 10.1007/978-3-319-17290-3_7

During the last years, the most active area of research in the field of semi-supervised learning has been related to methods based on graphs or networks. The common point of these techniques is in the representation of data items as vertices of the network, while the existence of links between data items depends both on the network formation strategy and on the labels of the labeled vertices [8]. A noticeable advantage of using networks for data analysis is the ability of revealing the topological structure of the data relationships. Thus, network-based methods allow for the detection of classes and groups with arbitrary shapes that are in turn difficult to identify for techniques that do not use structured data representations to perform the learning process [14].

Network-based semi-supervised learning begins by constructing a network with the input vector-based data using network formation strategies.[1] Once the network is built, the learning process consists in assigning a label for every unlabeled vertex in the test set. The inference is done by diffusing labels through the edges that interconnect vertices of the network [8]. The learning process employs both the labeled and unlabeled sets in the label diffusion process. In contrast to the traditional techniques that make use of attribute-value tables to conduct their analyzes on the data, network-based techniques directly use the direct and/or indirect neighborhood structures of the graph constructed from the input data to analyze and predict labels for unlabeled instances. As explained in several researches in the literature [20, 21, 23–25, 36], this feature may generate classifiers that are more robust and efficient.

In a semi-supervised learning process, algorithms can be either inductive or transductive. While inductive techniques construct general rules from the given training set, transductive learning limits the prediction to specific test instances.[2] Most network-based methods are transductive techniques, meaning that they aim at inferring a class for each unlabeled vertex in the test set only; thus they are not required to design a global generalizing function to other new vertices that are not in the test set.

Among the main advantages of network-based semi-supervised learning algorithms, we may highlight [8, 37]:

- The network structure can effectively detect clusters of various forms.
- The learning process does not make decisions based explicitly on distance functions.
- The representation of data sets with multiple classes is facilitated.
- Some problems are naturally represented by networks, for example: protein-protein interaction networks, blood mainstream, Internet, among others. In this case, reversing from a network-based data representation to a vector-based

[1]See Chap. 4 for a discussion on different network formation methods.

[2]Intuitively, if the learning problem is an exam, then the labeled data correspond to the few example problems that the teacher solved in class. The teacher also provides a set of unsolved problems. In the transductive setting, these unsolved problems are a take-home exam and you want to do well on them in particular. In the inductive setting, these are practice problems of the sort you will encounter on the in-class exam.

representation would be a lossy transformation. To see this, it would be difficult to model cycles in graphs using a vector-based data. Cycles permit recursiveness in the data relationships and are naturally modeled by networks.

Many semi-supervised learning techniques, such as Transductive SVM, can identify data classes of well-defined forms, but usually fail to identify classes of irregular forms. Thus, assumptions on the class distributions have to be made [8]. Unfortunately, such information is usually unknown *a priori*. In order to overcome this problem, several graph-based methods have been developed in the last years. Among them, we may highlight: mincut [40], local and global consistency [35], local learning regularization [34], local and global regularization [33], manifold regularization [4], semi-supervised modularity [22], D-Walks [7], random walk techniques [12, 29], and label propagation techniques [32, 38]. However, most of the graph-based methods share the same regularization framework, differing basically in the particular choice of the loss function and the regularization function [2, 3, 5, 13, 35, 40], and most of them have cubic order of computational complexity ($\mathscr{O}(V^3)$). This factor makes their applicability limited to small- or middle-sized data sets [36]. As data sets get larger and larger, the development of efficient semi-supervised learning methods is still necessary.

7.2 Network-Based Semi-Supervised Learning Assumptions

Recall that the main difference between supervised and semi-supervised learning is that the latter uses unlabeled data to improve the generalization performance of the classifier. In order to effectively use the unlabeled data in the learning process, we must assume some underlying data structure. Bad matching of the problem structure with the model assumption can lead to degradation of the classifier's performance. For example, quite a few semi-supervised learning methods assume that the decision boundary should avoid regions with high densities of data items. Nonetheless, if data are generated from two heavily overlapping Gaussian distributions, the decision boundary would go right through the densest region, and the majority of the existing methods that rely on such assumption would perform badly. Detecting bad matching in advance, however, is hard and remains an open question [39].

Semi-supervised learning algorithms make use of at least one of the following assumptions [8]:

- *Smoothness assumption*: data points in the attribute space that are close to each other are more likely to share the same label. This is also generally assumed in supervised learning and yields a preference for geometrically simple decision boundaries. In the case of semi-supervised learning, the smoothness assumption additionally yields a preference for decision boundaries in low-density regions, so that it is expected that few points of different classes will reside near the regions where these decision boundaries cross. The smoothness assumption can also be related to the belief that "similar examples ought to have similar labels."

- *Cluster assumption*: data points that can be connected via (many) paths through high-density regions are likely to have the same label. This is a special case of the smoothness assumption and gives rise to feature learning with clustering algorithms.
- *Manifold assumption*: each class lies on a separate manifold of much lower dimension than the input space. In this case, we can attempt to learn the manifold using both the labeled and unlabeled data to avoid the curse of dimensionality. Then learning can proceed using distances and densities defined on the manifold. The manifold assumption is practical when high-dimensional data are being generated by some process that may be hard to model directly, but has only a few degrees of freedom. For instance, human voice is controlled by a few vocal folds [28], and images of various facial expressions are controlled by a few muscles. We would like in these cases to use distances and smoothness in the natural space of the generating problem, rather than in the space of all possible acoustic waves or images respectively.

The typical scenario in a semi-supervised learning task is the following. Denote $\mathscr{T} = \{(x_1, y_1), \ldots, (x_L, y_L)\}$ as the set containing tuples of the form: vertex $x_i \in \mathscr{L}$ and its corresponding label $y_i \in \mathscr{Y}$, where $\mathscr{L}$ symbolizes the set of labeled vertices and $\mathscr{Y}$ stands for the set of class labels. $\mathscr{U} = \{x_{L+1}, \ldots, x_{L+U}\}$ is the unlabeled set, such that $\mathscr{V} = \mathscr{L} \bigcup \mathscr{U}$ is the vertex set. There are $L = |\mathscr{T}| = |\mathscr{L}|$ labeled instances and $U = |\mathscr{U}|$ unlabeled instances. Thus, there is a total of $V = |\mathscr{V}|$ data items in the semi-supervised learning process and $Y = |\mathscr{Y}|$ classes. When $Y = 2$, we have a binary classification problem. When $Y > 2$, we have a multi-class problem. y_i denotes the true label of the i-th data item x_i, while $\hat{y}_i$ represents the approximated label output by the semi-supervised learning algorithm. Frequently, we have few labeled instances and several unlabeled instances, such that the condition $U \gg L$ holds. The goal is to label the unlabeled instances in accordance with some convenient label propagation process using both labeled and unlabeled data in the learning process. Network-based techniques use a graph to approximate the low-dimensional manifold.

The premise that unlabeled data also helps in the learning process is discussed in [26]. Therein, it is shown that, using a finite sample analysis, if the complexity of the distributions under consideration is too high to be learnt using L labeled data points, but is small enough to be learnt using $U \gg L$ unlabeled data points, then semi-supervised learning can improve the performance of a supervised learning task.

One final note is of the existence of multiple manifolds. For instance, in handwritten digit recognition, each digit forms its own manifold in the feature space; in computer vision motion segmentation, moving objects trace different trajectories that are low-dimensional manifolds [31]. These manifolds may intersect or partially overlap, while having different dimensionality, orientation, and density. In graph-based algorithms, if we create a graph that connects points on different manifolds near a manifold intersection, then labels will propagate to other manifolds in an incorrect manner. In this case, we must be aware to construct isolated graph components that do not interconnect, thus avoiding wrong label propagation.

7.3 Representative Network-Based Semi-Supervised Learning Techniques

Semi-supervised methods that rely on networks define a graph in which labeled and unlabeled examples in the data set are represented by vertices, and edges reflect the similarity of those examples. These methods usually assume label smoothness over the graph. In general, graph methods are nonparametric, discriminative, and transductive.

They are generally nonparametric because they make no assumptions about the probability distributions of the data items that are being analyzed (distribution-free). Recall that the difference between parametric and nonparametric models is that the former has a fixed number of parameters, while the number of parameters in the latter grows with the amount of training data [10, 19]. Moreover, we stress that nonparametric models are not the same as none-parametric models: parameters are determined by the training data, not the model. The nonparametric nature of network-based semi-supervised learning algorithms is a positive characteristic, as it prevents the insertion of wrong or misleading biases during the learning process.

Network-based semi-supervised methods are generally discriminative models, because they model the dependence of an unobserved variable y, the class or label in our context, on one or more observed variables x (data items and their similarities). In discriminative models, unlike generative models, there is no room to allow for generating samples from the joint distribution of x and y. For tasks of classification in which that joint distribution is not needed, discriminative models can yield superior performance [15, 17, 27]. In contrast, generative models are generally more flexible than discriminative models in expressing dependencies in complex learning tasks.

Network-based semi-supervised methods usually employ transductive inference because they estimate labels or classes from observed, specific data items (labeled and unlabeled set) to only specific items (unlabeled set). In contrast, inductive inference is reasoning from observed training cases to general rules, which are then applied not only to the unlabeled set but also to other new test cases.

An extensive review on network-based semi-supervised learning techniques can be found in [8, 36, 39].

Many graph-based methods can be expressed in terms of a regularization framework, in which the goal is to minimize a cost or energy function C that is composed of two complementary terms:

$$C = f_{\text{loss}} + f_{\text{reg}}. \tag{7.1}$$

Each term in (7.1) serves different purposes, which are:

1. *Loss function* (f_{loss}): it leads the algorithm to penalize decisions that flip labels of pre-labeled vertices. Practically, to minimize this term, it is enough to prevent the change of pre-labeled vertices.

2. *Regularization function* (f_{reg}): it is responsible for modeling the cost of propagating labels to unlabeled vertices. Given that many algorithms rely on the smoothness assumption, this function must be smooth in dense regions of the network.

One implicit assumption here is that labeled data items are totally reliable. In imperfect learning, where there are noisy or wrongly labeled data items, the loss function would force algorithms that rely on the regularization framework to propagate wrong labels to the unlabeled data. Depending on the rate of imperfect data, the diffusion of bad labels could easily overwhelm the one of correct labels.

In the next sections, we explore several representative network-based semi-supervised learning techniques.

7.3.1 Maximum Flow and Minimum Cut

This method is presented in [5]. The original method classifies in a binary way, i.e., there are only two classes and the labels are confined in the set $y_i \in \{0, 1\}$, $\forall i \in \mathscr{V}$. The semi-supervised learning classification is posed as a graph mincut problem. In the binary case, positive labels act as sources, and negative labels act as sinks. The objective is to find a minimum set of edges whose removal (cut) would block all flow from the sources to the sinks. After the cut, the vertices connected to the sources are then labeled positive, and those to the sinks are labeled negative. We can view the term f_{loss} as a quadratic loss function with infinity weight:

$$f_{\text{loss}} = \lim_{w \to \infty} w \sum_{i \in \mathscr{L}} (\hat{y}_i - y_i)^2, \tag{7.2}$$

so that the influence of the regularization term in labeled data is effectively disabled. Consequently, the values of labeled data are in fact fixed at their true labels.[3] We are left to discuss the label diffusion process applied to the unlabeled data, which is governed by the following regularization function:

$$f_{\text{reg}} = \frac{1}{2} \sum_{i,j \in \mathscr{V}} \mathbf{A}_{ij} |\hat{y}_i - \hat{y}_j| = \frac{1}{2} \sum_{i,j \in \mathscr{V}} \mathbf{A}_{ij} \left(\hat{y}_i - \hat{y}_j\right)^2, \tag{7.3}$$

in which $\mathbf{A}_{ij}$ is the edge weight linking i to j and $\hat{y}_i$ is the estimated label of vertex $i \in \mathscr{V}$. Note that the second equality only holds due to the binary nature of $\hat{y}_i$, $\forall i \in \mathscr{V}$. Substituting (7.2) and (7.3) into (7.1), we get our objective function:

[3]Otherwise, the objective function that is composed of the loss and regularization terms is infinity. To note that, observe that if $\hat{y}_i \neq y_i$, $i \in \mathscr{L}$, then $f_{\text{loss}} \to \infty$.

$$C = \lim_{w \to \infty} w \sum_{i \in \mathscr{L}} (\hat{y}_i - y_i)^2 + \frac{1}{2} \sum_{i,j \in \mathscr{V}} \mathbf{A}_{ij} \left(\hat{y}_i - \hat{y}_j\right)^2, \tag{7.4}$$

subject to the constraint $\hat{y}_i \in \{0, 1\}$, $\forall i \in \mathscr{V}$ (crisp labeling). Effectively, the estimation of labeled instances $i \in \mathscr{L}$ must coincide with their initial label beliefs, in a way that we are only allowed for deciding the labels of unlabeled instances in $\mathscr{U}$.

The classical graph mincut approach has a number of attractive properties [6]. First, it can be computed in polynomial time using network flow tools. Second, the learning process can be viewed as providing the most probable configuration of labels in the associated Markov random field. Lastly, it can also be motivated from sample-complexity considerations.

The mincut algorithm, however, also suffers from several drawbacks. One noticeable drawback of mincut is that it only outputs hard or crisp classification without confidence intervals. In statistical terms, it only computes the mode, rather than marginal probabilities. For instance, in the research in [6], the graph is perturbed by adding random noise to the edge weights. Mincut is applied to multiple perturbed graphs, and labels are determined by a majority vote criterion. The procedure is similar to bagging, and effectively creates a "soft" mincut. The research in [13] gives a method based on spectral partitioning that produces an approximate minimum ratio cut in the graph. Another shortcoming comes from a practical perspective. A graph may have many minimum cuts, and the mincut algorithm produces just one, typically the "leftmost" one using standard network flow algorithms. For instance, a line of V vertices between two labeled points i and j has $V - 1$ cuts of size 1, and the leftmost cut will be especially unbalanced.

7.3.2 Gaussian Field and Harmonic Function

One of the main limitations of the mincut is that the algorithm only classifies using binary crisp labels. Data that are located in bordering or overlapping regions, however, may be labeled with less confidence than those data items that are in the core of their respective classes. Gaussian random fields and harmonic function methods [38, 40] try to address these problems. These techniques can be viewed as a form of nearest neighbors approach, in which the nearest labeled examples are computed in terms of a random walk on the graph. These learning methods have intimate connections with random walks, electric networks, and spectral graph theory, in particular with heat kernels and normalized cuts.

In the context of networks, harmonic functions estimate the label of an unlabeled vertex according to a weighted average of the labels of vertices in the neighborhood. In this way, the classification becomes smooth. Gaussian random fields and harmonic function methods [38, 40] are a continuous relaxation to the complex discrete Markov random fields. Likewise mincut, they employ a quadratic loss function with infinity weight, so that labeled data are clamped to their pre-defined labels and the regularization function is based on a quadratic form of the graph Laplacian $\mathbf{L}$. The cost function, therefore, is expressed as:

$$\begin{aligned} C &= \lim_{w\to\infty} w \sum_{i\in\mathscr{L}} (\hat{y}_i - y_i)^2 + \frac{1}{2}\sum_{i,j\in\mathscr{V}} \mathbf{A}_{ij}\left(\hat{y}_i - \hat{y}_j\right)^2 \\ &= \lim_{w\to\infty} w \sum_{i\in\mathscr{L}} (\hat{y}_i - y_i)^2 + \frac{1}{2}\hat{Y}^T\mathbf{L}\hat{Y}, \end{aligned} \tag{7.5}$$

in which $\hat{y}_i \in [0, 1]$, $\hat{Y} = [y_1, y_2, \ldots, y_V]^T$ is a vector that stores the estimated labels of all of the vertices, and $\mathbf{L}$ is the graph Laplacian. The fuzziness of $\hat{Y}$ is a key relaxation towards the mincut technique that only allows for crisp classification, i.e., $\hat{y}_i \in \{0, 1\}$, $i \in \mathscr{V}$. Recall that the (i,j)-th entry of the graph Laplacian $\mathbf{L}$ with adjacency matrix $\mathbf{A}$ is:

$$\mathbf{L}_{ij} = \mathbf{D}_{ij} - \mathbf{A}_{ij} = \begin{cases} k_i, & \text{if } i = j \\ -\mathbf{A}_{ij}, & \text{otherwise,} \end{cases} \tag{7.6}$$

in which k_i is the degree of vertex i computed using the adjacency matrix $\mathbf{A}$. $\mathbf{D}$ is the degree matrix that is computed as:

$$\mathbf{D}_{ij} = \begin{cases} k_i, & \text{if } i = j. \\ 0, & \text{otherwise.} \end{cases} \tag{7.7}$$

While it is clear that the solution form of the loss function in (7.5) is the one that does not flip labels of pre-labeled instances, it may not be clear the solution of the regularization term $\hat{Y}^T\mathbf{L}\hat{Y}$ that assures smoothness in the labeling process. We elaborate on that in the following. From (7.5), we see that:

$$\hat{Y}^T\mathbf{L}\hat{Y} = \sum_{i,j\in\mathscr{V}} \mathbf{A}_{ij}\left(\hat{y}_i - \hat{y}_j\right)^2. \tag{7.8}$$

The regularization term in (7.8) is a measure of non-smoothness of the estimated labels according to the network topology. For estimated labels that are not similar in the neighborhood, the term $\hat{Y}^T\mathbf{L}\hat{Y}$ yields large values. For estimated labels that are similar in the neighborhood, the term $\hat{Y}^T\mathbf{L}\hat{Y}$ produces small values.

Consider the eigenequation:

$$\mathbf{L}v = \lambda v, \tag{7.9}$$

in which v is one of the eigenvectors of $\mathbf{L}$ and λ is the associated eigenvalue of that eigenvector. If we right-multiply (7.10) by the transpose of the eigenvector v, we get:

$$\begin{aligned} v^T\mathbf{L}v &= v^T\lambda v \\ v^T\mathbf{L}v &= \lambda v^T v \\ v^T\mathbf{L}v &= \lambda, \end{aligned} \tag{7.10}$$

in which the second equality comes from the factor that λ is a scalar and therefore we can rearrange vector v^T, while the third equality holds from the orthonormality of v, i.e., $v^T v = 1$.

If we consider that the eigenvector v is one of the solutions of regularization function as in (7.8), i.e., $v = y$, we get:

$$y^T\mathbf{L}y = \lambda, \tag{7.11}$$

that is, the eigenvalue λ associated to the solution y of the Laplacian $\mathbf{L}$ gives us an idea of the non-smoothness of the estimated labels (y). As we select eigenvector solutions that have larger and larger associated λ values, the less smooth are the estimated labels. Considering that the loss function in (7.5) cannot be changed by the learning process as our goal is to minimize C, we are effectively minimizing the regularization term constrained to the given labeled instances. Therefore, the estimated labels must be near or equal eigenvectors that have associated eigenvalues with small magnitude.

7.3.3 Tikhonov Regularization Framework

The Tikhonov regularization algorithm in [2] uses a general form of loss function:

$$f_{\text{loss}} = \frac{1}{L}\sum_{i\in\mathscr{L}} V(\hat{y}_i, y_i), \tag{7.12}$$

in which $V(\hat{y}_i, y_i)$ is some loss function. For instance, $V(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$, then we have a regularized least squares technique, while $V(\hat{y}_i, y_i) = \max(0, 1 - \hat{y}_i y_i)$ leads to the SVM algorithm. Note that now the loss function allows for changes in the prior labeled set.

The regularization function is:

$$f_{\text{reg}} = \hat{Y}^T\mathbf{S}\hat{Y}, \tag{7.13}$$

in which $\mathbf{S}$ is a smoothness matrix, such as the Laplacian matrix $\mathbf{L}$.

In this way, the cost function of the regularization framework becomes:

$$F = \frac{1}{L}\sum_{i\in\mathscr{L}} (\hat{y}_i - y_i)^2 + \mu\hat{Y}^T\mathbf{S}\hat{Y}, \tag{7.14}$$

in which μ is a regularization parameter that modulates the influences played by the loss and regularization functions. For stability purposes, the prior belief of labeled instances is subtracted from its mean.

The Tikhonov regularization framework has some interesting advantages:

- It eliminates the need of computing multiple eigenvectors or complicated graph invariants (mincut, max flow etc.). There is also a simple closed form solution for the optimal regressor. The problem is reduced to a single, usually sparse, linear system of equations whose solution can be computed efficiently. One of the algorithms proposed (interpolated regularization) is extremely simple with no free parameters.
- The generalization error can be bounded and related to properties of the underlying graph using arguments from algorithmic stability.
- If the graph arises from the local connectivity of data obtained from sampling an underlying manifold, then the approach has natural connections to regularization on that manifold.

7.3.4 Local and Global Consistency

This method has been proposed by Zhou et al. [35] and is one of the first studies in network-based semi-supervised learning techniques. This method considers the general problem of learning from labeled and unlabeled data by means of constructing a classification function that is sufficiently smooth with respect to the intrinsic labeled and unlabeled data structures.

The technique considers the evolution of a set of matrices $\mathscr{M}$ with dimensions $V \times Y$, all of which with nonnegative entries. The matrix $\hat{\mathbf{Y}} = [\hat{Y}_1^T, \ldots, \hat{Y}_V^T]^T \in \mathscr{M}$ corresponds to the fuzzy classification of the data items $\mathscr{V}$, such that, for each labeled or unlabeled vertex $x_i \in \mathscr{V}$, we designate a label in accordance with the expression $\hat{y}_i = \arg\max_{y \in \mathscr{Y}} \mathbf{Y}_{iy}$. One can think of $\hat{\mathbf{Y}}$ as a vectorial function that attributes, for each unlabeled data x_i, the maximum value of $\hat{\mathbf{Y}}_{iy}, y \in \mathscr{Y}$. Define also the matrix $\mathbf{Y}$ with dimensions $V \times Y$, such that $\mathbf{Y}_{iy} = 1$ if x_i is labeled as $y \in \mathscr{Y}$, and $\mathbf{Y}_{iy} = 0$, otherwise. The algorithm evolves as follows:

1. Generate the adjacency matrix $\mathbf{A}$ according to the Gaussian kernel, which is given by $\mathbf{A}_{ij} = \exp\left(\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$ if $i \neq j$, and $\mathbf{A}_{ii} = 0$, otherwise;
2. Construct the matrix $\mathbf{S} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, in which $\mathbf{D}$ is a diagonal matrix with each entry (i, i) equivalent to the sum of the i-th row of $\mathbf{A}$;
3. Iterate $\hat{\mathbf{Y}}(t + 1) = \alpha\mathbf{S}\hat{\mathbf{Y}}(t) + (1 - \alpha)\mathbf{Y}$ until it converges, where $\alpha \in (0, 1)$;
4. Consider that $\hat{\mathbf{Y}}^*$ denotes the limit of the sequence $\{\hat{\mathbf{Y}}(t) : t \in \mathbb{N}\}$. Then, label each unlabeled vertex x_i following the formula: $\hat{y}_i = \arg\max_{j \in \mathscr{Y}} \hat{\mathbf{Y}}^*_{ij}$.

Moreover, it can be shown that the sequence $\Im = \{\hat{\mathbf{Y}}(t) : t \in \mathbb{N}\}$ converges and assumes the following closed formula:

$$\hat{\mathbf{Y}}^* = \lim_{t\to\infty} \hat{\mathbf{Y}}(t) = (\mathbf{I} - \alpha\mathbf{S})^{-1}\mathbf{Y}. \tag{7.15}$$

Still in [35], a regularization framework is molded with the aforementioned dynamics. In this framework, one aims at minimizing a cost or energy expression. The encountered expression, here written as $C(\hat{\mathbf{Y}})$, is given as:

$$C(\hat{\mathbf{Y}}) = \frac{1}{2}\left(\sum_{i,j\in\mathscr{V}} \mathbf{A}_{ij} \left\| \frac{1}{\sqrt{\mathbf{D}_{ii}}}\hat{\mathbf{Y}}_i - \frac{1}{\sqrt{\mathbf{D}_{jj}}}\hat{\mathbf{Y}}_j \right\|^2 + \mu \sum_{i\in\mathscr{V}} \|\hat{\mathbf{Y}}_i - \mathbf{Y}_i\|^2\right), \tag{7.16}$$

in which $\mu > 0$ is a regularization parameter. In this case, the optimal values for the classification function become:

$$\hat{\mathbf{Y}}^* = \arg\min_{F\in\mathscr{M}} C(\hat{\mathbf{Y}}). \tag{7.17}$$

The first term in (7.16) enforces smoothness decisions by the classifier, meaning that a good classification function must not have large derivatives in high-density areas. This is exactly the definition of a regularization function. The second term symbolizes the adjustment restriction, revealing that a good classification function also must not exchange the labels from already labeled data. In this case, this definition perfectly fits into the description of a loss function. The counterweight between these two conflicting quantities is given by the positive constant μ.

The advantage of this technique is its simplicity. As one can see, the propagation is done by utilizing a linear update rule and convergence issues have been fully described, enabling one to understand the dynamics of such model in the long run. However, the algorithm suffers from some drawbacks: since the propagation is done utilizing a linear function, nonlinear characteristics of the data may pass unseen by the algorithm. Moreover, since a matrix inversion is involved to find the optimal solution, the algorithm requires $\mathscr{O}(V^3)$ to run, which is unfeasible for large-scale networks.

7.3.5 Adsorption

The adsorption technique was first introduced in [1]. It was then further extended and given a theoretical analysis in [30]. Adsorption has many desirable properties, among which we can highlight:

- Possibility to perform multiclass classification ($Y > 2$).
- Definition can be stated in terms of a parallelized implementation, thus enabling its application on large-scale data sets.
- Mechanism to deal with imperfect training data.[4]

Likewise other label propagation algorithms that work in a networked environment, adsorption propagates label information from the labeled examples to the entire set of vertices via the edges (network topology). The labeling is represented using a non-negative score for each label, in which high scores are attributed to those labels that indicate the highest associations or similarities to unlabeled vertices. If these scores are additively normalized, they can be thought of as a conditional distribution over the labels given the unlabeled data.

Let $\hat{\mathbf{Y}} = [\hat{Y}_1^T, \ldots, \hat{Y}_V^T]^T \in V \times Y$ be a matrix in which the v-th row $\hat{\mathbf{Y}}_v$ corresponds to the fuzzy classification of $v \in \mathcal{V}$ towards the Y possible classes. That is, $\hat{\mathbf{Y}}_{vy}$ is the fuzzy classification of the data item v with respect to class y. Similarly $\mathbf{Y}$ encodes the initial belief of all of the vertices in the network towards the existent classes. At the initial phase of the algorithm, we must supply the belief for all of the vertices $v \in \mathcal{V}$. If v is an unlabeled vertex, we can simply set the belief of v as the zero-valued vector. Adsorption outputs an estimated belief or fuzzy classification label for each class in the vector $\hat{\mathbf{Y}}_v$, $v \in \mathcal{V}$.

We can view the learning mechanism performed by the adsorption algorithm as controlled random walks that are conducted in accordance with the network topology. The control over the random walk is realized via three possible actions: *inject, continue, abandon*, each of which with occurrence probabilities on vertex $v \in \mathcal{V}$ of $p_v^{(\text{inject})}$, $p_v^{(\text{continue})}$, and $p_v^{(\text{abandon})}$, respectively. For a valid transition probability, one must have:

$$p_v^{(\text{inject})} + p_v^{(\text{continue})} + p_v^{(\text{abandon})} = 1. \tag{7.18}$$

To label each unlabeled or even already labeled vertex $v \in \mathcal{V}$, we first initiate a random walk starting at v. At each time step, the random walk is allowed to choose over three actions:

1. With probability $p_v^{(\text{inject})}$, the random walker stops and returns the pre-defined initial belief Y_v, i.e., $\hat{\mathbf{Y}}_v = \mathbf{Y}_v$. A further constraint is also imposed to force label diffusion to unlabeled instances in the graph. Whenever v is unlabeled, we fix $p_v^{(\text{inject})} = 0$, so that the random walk cannot output the initial belief of an unlabeled vertex as its classification decision.
2. With probability $p_v^{(\text{abandon})}$, the random walker abandons the labeling diffusion process and returns the zero-valued vector as the classification decision, that is, $\hat{\mathbf{Y}}_v$.

[4]Imperfect training data arises when the labeled instances are not totally reliable. We explore in detail another semi-supervised learning algorithm that deal with the detection and prevention of labels that are diffused by possibly wrong labeled data in Chap. 10.

3. With probability $p_v^{(\text{continue})}$, the random walker continues to navigate in the graph, specifically to one of the neighbors of v with a probability proportional to the edge weight. The transition probability follows the transition matrix of a random walk process as explored in Sect. 2.4.1. For convenience, we rewrite the transition matrix as follows:

$$\mathbf{P}[u \mid v] = \mathbf{P}_{vu} = \frac{\mathbf{A}_{vu}}{\sum_{i \in \mathscr{V}} \mathbf{A}_{vi}}. \tag{7.19}$$

Considering this three-way dynamics of the walker, the expected score $\hat{\mathbf{Y}}_v$, $v \in \mathscr{V}$, is given by:

$$\hat{\mathbf{Y}}_v = p_v^{(\text{inject})} Y_v + p_v^{(\text{continue})} \sum_{u \in \mathscr{N}(v)} \mathbf{P}[u \mid v] \hat{\mathbf{Y}}_u + p_v^{(\text{abandon})} 0_Y, \tag{7.20}$$

in which 0_Y is the zero-valued vector with Y entries and $\mathscr{N}(v)$ returns the set of neighbors of v.

Alternatively, in order to guarantee the positiveness of $\hat{\mathbf{Y}}_v$, a slight modification can be introduced whenever the random walker abandons the walk. Instead of returning a zero-valued vector, we can create a dummy label $y_d \notin \mathscr{Y}$ and designate that dummy label as the estimated label of v. We can conceive this additional dummy class as encoding ignorance or uncertainty about the correct label of v. With this modification, at least one of the three terms in (7.20) always assumes a positive value. Thus, $\hat{\mathbf{Y}}_v$ is positive.

The smoothness assumption of the adsorption algorithm is modeled by the second term in the RHS of (7.20). Note that the estimated label of v, $\hat{\mathbf{Y}}_v$, is composed of a weighted linear combination of the estimated labels in the neighborhood of v. This averaging view then defines a set of fixed-point equations to update the predicted labels. Since the past trajectories or states of the walkers do not need to be maintained, the adsorption algorithm is memoryless. As such, it scales to large-scale data sets with possibly dense configurations and also can be easily parallelized [1].

Some heuristics have been proposed to estimate $p_v^{(\text{inject})}$, $p_v^{(\text{continue})}$, and $p_v^{(\text{abandon})}$ [1, 30]. Effectively, these heuristics suggest that:

$$\begin{aligned} p_v^{(\text{continue})} &\propto c_v, \\ p_v^{(\text{inject})} &\propto d_v. \end{aligned} \tag{7.21}$$

The first quantity $c_v \in [0, 1]$ is a value that monotonically decreases with the number of neighbors of vertex v. That is, the more neighbors v has in the network topology, the smaller is c_v. Intuitively, if v connects with several other vertices, it is probably a difficult vertex to be classified. Hence, the idea is to prevent further label propagation that comes from it. This mechanism assures preferential trajectories that pass through vertices with small degrees.

The other quantity $d_v \geq 0$ is a value that monotonically increases with the entropy (for labeled vertices), and in this case we prefer to use the prior belief rather than the computed quantities from the neighbors. The entropy of vertex v is evaluated using the transition matrix, as follows:

$$H(v) = -\sum_{u \in \mathcal{N}(v)} \mathbf{P}[u \mid v] \log \mathbf{P}[u \mid v]. \tag{7.22}$$

Once computed, we pass the entropy through the following monotonically decreasing function:

$$f(x) = \frac{\log(\beta)}{\log(\beta + e^x)}, \tag{7.23}$$

and the term c_v is defined as:

$$c_v = f(H(v)), \tag{7.24}$$

and the term d_v:

$$d_v = \begin{cases} (1 - c_v)\sqrt{H(v)}, & \text{if } v \text{ is labeled.} \\ 0, & \text{otherwise.} \end{cases} \tag{7.25}$$

Finally, to ensure that (7.18) holds, we set:

$$p_v^{(\text{continue})} = \frac{c_v}{z_v}, \tag{7.26}$$

$$p_v^{(\text{inject})} = \frac{d_v}{z_v}, \tag{7.27}$$

$$p_v^{(\text{abandon})} = 1 - \frac{c_v}{z_v} - \frac{d_v}{z_v}. \tag{7.28}$$

in which z_v is a normalization constant given by:

$$z_v = \max(c_v + d_v, 1). \tag{7.29}$$

7.3.6 Semi-Supervised Modularity Method

This algorithm has been proposed by Silva and Zhao [22] and is inspired by the modularity greedy algorithm, which we have introduced in Sect. 6.3.2.1. In the original modularity greedy algorithm, at each time step, two communities, say i

and j, are merged, in such a way that the largest increment (or least decrement) of the modularity occurs at a particular step. No restrictions on the communities to be merged are specified by the original model.

In order to adapt the modularity greedy algorithm for the context of semi-supervised learning, we make the following modifications:

1. Initially, we have L labeled vertices in the network. The task consists in propagating their labels to the unlabeled vertices. Once an unlabeled vertex receives a label, it cannot be changed.
2. At each step, we merge the communities (at the beginning, each community encompasses only one vertex) in such a way that the modularity increment is maximal. However, such merge is subjected to some constraints: in light of mimicking the propagation of labels in the network, a merge only occurs if at least one of the candidate communities has been labeled before. Suppose that communities c_i and c_j have been selected to be merged, each of which carrying the labels y_i and y_j. Let $\emptyset$ denote the unlabeled class. Then, one of the following four cases occurs:

   Case 1. The merge does not occur if $y_i \neq y_j$, provided that $y_i \neq \emptyset$ and $y_j \neq \emptyset$. This case represents a clash between two different classes that have been previously labeled.

   Case 2. The merge occurs if $y_i \neq \emptyset$ and $y_j = \emptyset$, or $y_i = \emptyset$ and $y_j \neq \emptyset$. This case represents the traditional label propagation from a labeled community to an unlabeled one. c_j receives the label from c_i in the first case and c_i receives the label from c_j in the second case.

   Case 3. The merge occurs if $y_i = y_j$, provided that $y_i \neq \emptyset$ and $y_j \neq \emptyset$. In this case, the merge process just puts two communities of the same class together, maximizing the modularity.

   Case 4. The merge does not occur if $y_i = y_j = \emptyset$, since no label is being propagated.

If the merge does not occur, then we select other two communities that have the second largest entry in the modularity increment matrix $\Delta\mathbf{Q}$ to be potentially merged, i.e., the Step 2 is repeated, and so on, until a valid merge takes place.

Keeping in mind that the modularity algorithm tries to maximize the number of edges among vertices of the same community, while also attempting to minimize the same quantity among distinct communities, the dynamic of the procedure will propagate labels so as to maintain the cluster and smoothness assumptions. In this way, the modified modularity greedy algorithm performs the work of propagating the labels in an optimized manner, provided that the network is strongly connected among vertices of the same class and weakly connected among vertices of distinct classes.

For the stop criterion of this algorithm, it simply needs to be run until no unlabeled vertex remains, regardless of the value of the modularity, since we are not looking for a good network division, but for an *ordered way* of labeling vertices. The mechanism of maximizing the modularity does this job for us. The convergence is guaranteed to happen and a proof has been provided in [22].

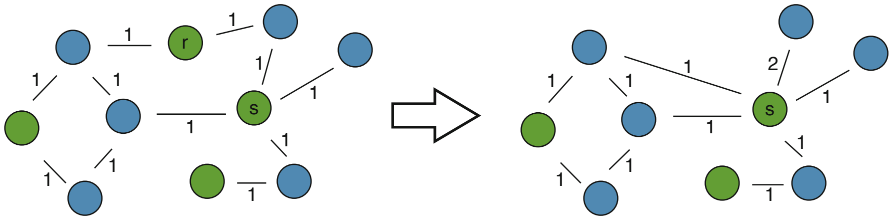

Fig. 7.1 Process of coalescence of vertices s and r. After the merge, s consumes r and becomes a super-vertex. All neighbors of r are connected to s during this procedure. Reproduced from [22] with permission from Elsevier

Additionally, in an attempt to make feasible the application of this semi-supervised algorithm on large-scale networks, a network reduction technique is explored. Let $\varphi(y) \in [0, 1]$ denote the proportion of reduction to be performed over the class pertaining to label $y \in \mathscr{Y}$. Denote also $\Psi_y(t)$ as set of data items that belongs to class $y \in \mathscr{Y}$. Then, the reduction is done by the following procedure, step-by-step:

1. Randomly choose pairs of pre-labeled vertices $r \in \Psi_y(t)$ and $s \in \Psi_y(t)$ to coalesce. In this process, r is removed from the network and s is entitled as a super-vertex, by virtue of the fact that it now represents more than one vertex in the network. In this process, all of the links that are connected to r are redirected to s. Suppose a connection between w and s already exists and w is also a neighbor of r, then we strengthen the connection between w and s by adding the weight from the former edge (w, r) to (w, s). Figure 7.1 illustrates this idea. This is done until $|\Psi_y(t + \Delta_t)| = (1 - \varphi(y))|\Psi_y(t)|$, where $\Delta_t > 0$ is a parameter bounded by the upper limit provided by the convergence proof in [22]. Essentially, $|\Psi_y(t+\Delta_t)|$ denotes the size of Ψ_y in a future time that can be reached in a finite number of steps. If $\varphi(y) = 1$, then we deliberately continue to merge process until $|\Psi_y(t + \Delta_t)| = 1$, i.e., until only one element remains of that class.
2. All of the self-loops, brought into the network by the reduction process, are removed. This prevents the modified modularity greedy algorithm from trying to merge a certain community with itself.
3. This process of reduction is performed for every class $y \in \mathscr{Y}$ that exists in the network.

In general, at the end of the reduction process, it is expected that the network will shrink, because $(V-L)+\sum_{y\in\mathscr{Y}} |\Psi_y(t + \Delta_t)| = (V-L)+\sum_{y\in\mathscr{Y}} [1 - \varphi(y)]|\Psi_y(t)| \leq V$, since $0 \leq \varphi(y) \leq 1$ and thus $\sum_{y\in\mathscr{Y}} [1 - \varphi(y)]|\Psi_y(t)| \leq L$. If the proportion of labeled vertices is large, then this process greatly reduces the network size, provided that $\varphi(y), \forall y \in \mathscr{Y}$, is large. Usually, the quantity of labeled vertices is small, because the task of labeling is generally expensive and cumbersome.

The main advantage of the aforementioned technique is that it does not require any kinds of parameters in order to work. Considering that the task of parameter tuning often takes a considerable time to complete and that lots of traps are involved in this investigation, such as the presence of local maxima, an expert may be needed such as to set feasible initial kickoff values for the parameters tuning procedure.

Therefore, the feature of having no free parameters makes the applicability of the semi-supervised modularity technique in real-world applications easy. A major drawback of the aforementioned technique is that it suffers from the inherent resolution problem that the original modularity greedy algorithm presents.[5] This often leads to bad results when there are classes with very distinct sizes.

7.3.7 Interaction Forces

The interaction forces technique was presented in [9]. This method is nature-inspired and relies on attraction forces. It models data instances as points in a P-dimensional space and performs their motion accordingly to the resultant force applied upon them. The labeled instances act as attraction points, while unlabeled instances receive forces and move towards these attraction points. Under some circumstances, unlabeled instances receive labels from labeled points and thereafter become new attraction points.

The use of attraction forces between labeled and unlabeled instances can provide a model for semi-supervised learning that fits well the smoothness and cluster assumptions. Labeled instances are fixed attraction points that apply attraction forces on unlabeled instances. As a result, unlabeled instances move towards the direction of the resultant force. Eventually, they converge to an attraction point. Once an unlabeled instance gets close enough to a labeled instance, say inside a circle with radius δ, the label from that attraction point propagates to the unlabeled instance located at its surroundings. As a consequence of the labeling process, it then becomes a new fixed attraction point. At the end of the process, it is expected that all of the instances will converge to some attraction point. By means of the attraction forces, instances are kept together in their dense groups (clusters), while different labeled points are responsible for dividing the space under the smoothness assumption.

We need two considerations in order to accomplish the above mentioned behavior and classify the unlabeled instances correctly. One of them is to guarantee that the process is stable, and the other is to certify that the labels propagate adequately through the unlabeled instances, in the sense that the algorithm converges and achieves good classification accuracy. The stability issue can be treated using similar approaches from swarm aggregation methods [11, 16], while the label propagation dynamic can be analyzed in terms of the parameters that underpin the attraction forces.

The motion or the stepwise differential movement of an unlabeled point v_i at step t, denoted here as $\dot{v}_i$, is governed by the following system:

$$\dot{v}_i(t) = \sum_{j \in \mathscr{L}} f[v_j(t) - v_i(t)], \tag{7.30}$$

[5]See Sect. 2.3 for details.

$\forall i \in \mathscr{U}$, where function f is the attraction force among instances. As described by (7.30), each unlabeled instance v_i receives attractive forces from all of the labeled instances and the resultant force is the sum of all individual forces. Thus, the direction and magnitude of $v_i(t)$'s motion are determined by the forces applied by the labeled instances.

The attraction function between an unlabeled item $i \in \mathscr{U}$ and a labeled item $j \in \mathscr{L}$ (attractor) is defined as a Gaussian field with parameters α and β:

$$f[v_j(t) - v_i(t)] = [v_j(t) - v_i(t)]\frac{\alpha}{e^{\beta\|(v_j(t)-v_i(t))\|^2}}. \tag{7.31}$$

The attraction function guarantees that the closer an unlabeled point is to an attractor labeled point, the stronger is the force. Moreover, the parameters of the Gaussian field provide an easy way to adjust the function amplitude and range.

Algorithm 1 summarizes the interaction forces method. The method is performed iteratively in four steps (from 2 to 5), until all of the instances are properly labeled.

7.3.8 Discriminative Walks (D-Walks)

This technique is introduced in [7]. D-walks rely on random walks performed on the input graph seen as a Markov chain. For a review on Markov chain theory, see Sect. 2.4.1. More precisely, D-Walks are essentially computed as a betweenness measure that is based on passage times during constrained random walks of bounded lengths. Unlabeled vertices are assigned to the category for which the betweenness is the highest. The D-walks approach has the following properties:

Algorithm 1 : Interaction forces technique

```
Input:
$\mathscr{L}$ : labeled data set
$\mathscr{U}$ : unlabeled data set
Output:
$l_i$ : estimated class for each $\mathbf{x}_i \in \mathscr{U}$
Initialization:
    1.$(\alpha, \beta, \delta)$ = Initialize parameters
Classification:
DO
    2. Calculate distances among points
    3. Calculate attraction forces
    4. Update points' positions
    5. Update labels
WHILE (there are unlabeled instances)
```

- It has a linear time complexity with respect to the number of edges, the maximum walk length and the number of classes; such a low complexity allows the technique to deal with very large sparse graphs.
- It can handle directed or undirected graphs.
- It can deal with multi-class problems.
- It has a unique hyper-parameter, the walk length, that can be tuned efficiently.

We first transform the (weighted) adjacency matrix into a transition matrix, which is a row-stochastic matrix, using the following operation:

$$P[X_t = q' \mid X_{t-1} = q] = \mathbf{P}_{qq'} \triangleq \frac{\mathbf{A}_{qq'}}{\sum_{k \in \mathscr{V}} \mathbf{A}_{qk}}, \tag{7.32}$$

in which $\mathbf{A}_{qq'}$ stands for the edge weight linking q to q'. Each vertex in the network corresponds to a state in the Markov chain system. The graph can be directed or undirected, weighted or non-weighted.

We now introduce the discriminative random walks (D-Walks). Essentially, a D-walk is a random walk starting in a labeled vertex and ending when any vertex having the same label (possibly the starting vertex itself) is reached for the first time.

Definition 7.1. D-Walk Given a Markov chain defined on the state set $\mathscr{V}$ and a class label $y \in \mathscr{Y}$, a D-Walk is a sequence of states $q_0, \ldots, q_\lambda$, $\lambda > 0$, such that $y_{q_0} = y_{q_\lambda} = y$ and $y_{q_t} \neq y$, $0 < t < \lambda$.

The notation $\mathscr{D}^y$ refers to the set of all D-Walks starting and ending in vertices of class y.

The betweenness function $B(q, y)$ measures the extent an unlabeled vertex $q \in \mathscr{U}$ is located "in-between" vertices of class $y \in \mathscr{Y}$. The betweenness $B(q, y)$ is formally defined as the expected number of times vertex q is reached during D-Walks on $\mathscr{D}^y$.

Definition 7.2. D-Walks Betweenness Given an unlabeled vertex $q \in \mathscr{U}$ and a class $y \in \mathscr{Y}$, we define the D-Walk betweenness function $\mathscr{U} \times \mathscr{Y} \to \mathbb{R}^+$ as:

$$\mathbf{B}(q, y) \triangleq \mathbb{E}\left[\text{pt}(q) \mid \mathscr{D}^y\right], \tag{7.33}$$

in which pt(q) is the passage time of vertex $q \in \mathscr{V}$ whose formal definition has been reviewed in Sect. 2.4.1.

Vertices belonging to class y are first duplicated such that the original vertices are used as absorbing states and the duplicated ones as starting states. The transition matrix $\mathbf{P}$ is augmented as follows:

1. We duplicate the rows of $\mathbf{P}$ corresponding to labeled vertices of class $y \in \mathscr{Y}$ at the bottom of the matrix.
2. We add columns full of zeroes at the right of $\mathbf{P}$. The number of added columns is equal to the number of added rows in the previous step.
3. We define $p_{qq'} = 1 \iff q = q'$ and 0 otherwise, for all of the vertices belonging to y. The augmented matrix is denoted here by $\mathbf{P}^y$. The initial distribution vector is adapted accordingly, resulting in the vector $p(0)^y$.

The betweenness is finally computed as follows:

$$\mathbf{B}(q,y) = \left[(p(0)_T^y)'(\mathbf{I} - \mathbf{P}_T^y)^{-1}\right]_q, \tag{7.34}$$

in which $\mathbf{P}_T^y$ and $p(0)_T^y$ denote, respectively, the transition matrix and the initial distribution vector restricted to transient states. Matrix inversion, however, is computed in $\mathscr{O}(V^3)$, limiting the use of the technique for large-scale graphs.

The authors in [7] instead propose to use bounded walks. Bounding the walk length systematically provides a better classification rate with the additional benefit that the betweenness can be computed very efficiently using forward and backward recurrences. Let $\mathscr{D}_\lambda^y$ denote to the set of all D-Walks of length exactly equal to λ. Moreover, consider that $\mathscr{D}_{\leq\lambda}^y$ refer to the set of all bounded D-Walks up to a given length λ. We define the bounded betweenness measure $\mathbf{B}_\lambda(q,y)$ as follows.

Definition 7.3. Bounded D-Walks Betweenness Given an unlabeled vertex $q \in \mathscr{U}$ and a class $y \in \mathscr{Y}$, we define the bounded D-Walk betweenness function $\mathscr{U} \times \mathscr{Y} \to \mathbb{R}^+$ as:

$$\mathbf{B}_\lambda(q,y) \triangleq \mathbb{E}\left[\text{pt}(q) \mid \mathscr{D}_{\leq\lambda}^y\right]. \tag{7.35}$$

Following the authors in [7], limiting the random walk length brings to major advantages in a classification process:

- The algorithm presents better accuracy rates in relation to unbounded D-Walks.
- We can compute the bounded betweenness measure very efficiently.

An efficient way to evaluate the bounded betweenness measure is to use forward-backward variables, similar to those employed in the Baum-Welch algorithm for hidden Markov models [18]. Given a state $q \in \mathscr{V}$ and a time $t \in \mathbb{N}$, the forward variable $\alpha^y(q,t)$ computes the probability of reaching state q after t steps without visiting vertices of class $y \in \mathscr{Y}$, while starting from any state in class y. We can evaluate the forward variables using the following recurrence:

$$\begin{aligned} &(\text{case } t = 1)\ \ \alpha^y(q,1) = \frac{1}{V_y}\sum_{q' \in \mathscr{L}_y} p_{q'q} \\ &(\text{case } t \geq 2)\ \ \alpha^y(q,t) = \sum_{q' \in \mathscr{U}} \alpha^y(q', t-1) p_{q'q} \end{aligned}, \tag{7.36}$$

in which $\mathscr{L}_y$ is the set of labeled vertices of class y and $V_y = |\mathscr{L}_y|$. The initial recurrence (case $t = 1$) assumes that the walker can start at any vertex that is member of class y with uniform probability $\frac{1}{V_y}$. Thus, the equation provides the probability that q is visited in the next iteration. We also observe that, while in the first recurrence we loop through members of class y, we forbid visits in members of class y when $t \geq 2$. In fact, a D-Walk finishes whenever the walk visits a vertex with label coincident with the label of the starting vertex.

In an opposite perspective, the backward variable $\beta^y(q,t)$ computes the probability that state q is attained by the process t steps before reaching any vertex labeled y for the first time. We evaluate the backward variables using the following recurrence:

$$\begin{aligned}
&(\text{case } t = 1)\ \beta^y(q,1) = \sum_{q' \in \mathscr{L}_y} p_{qq'} \\
&(\text{case } t \geq 2)\ \beta^y(q,t) = \sum_{q' \in \mathscr{U}} \beta^y(q', t-1) p_{qq'}
\end{aligned}. \tag{7.37}$$

To compute $B_\lambda(q,y)$, we first calculate the mean passage time in a vertex $q \in \mathscr{U}$ during $\mathscr{D}_\lambda^y$. The length-conditioned passage time function $\text{pt}(q)$, $\mathbb{E}\left[\text{pt}(q) \mid \mathscr{D}_\lambda^y\right]$, can be decomposed as a sum of indicator variables: $\text{pt}(q) = \sum_{t=1}^{\lambda-1} \mathbb{1}_{[X_t = q]}$. Consequently,

$$\begin{aligned}
\mathbb{E}\left[\text{pt}(q) \middle| \mathscr{D}_\lambda^y\right] &= \mathbb{E}\left[\sum_{t=1}^{\lambda-1} \mathbb{1}_{[X_t=q]} \middle| \mathscr{D}_\lambda^y\right] \\
&= \sum_{t=1}^{\lambda-1} \mathbb{E}\left[\mathbb{1}_{[X_t=q]} \mid \mathscr{D}_\lambda^y\right] \\
&= \sum_{t=1}^{\lambda-1} P\left(X_t = q \mid \mathscr{D}_\lambda^y\right) \\
&= \sum_{t=1}^{\lambda-1} \frac{P\left(X_t = q \wedge \mathscr{D}_\lambda^y\right)}{P\left(\mathscr{D}_\lambda^y\right)},
\end{aligned} \tag{7.38}$$

in which the second equality comes from the linearity of the expectation operator, the third equality holds because $\mathbb{E}[\mathbb{1}_{[A]}] = P(A)$, and the fourth equality is true due to Bayes theorem.

We can compute the joint probability in the numerator of (7.38) as:

$$P\left(X_t = q \wedge \mathscr{D}_\lambda^y\right) = \alpha^y(q,t)\beta^y(q, \lambda - t), \tag{7.39}$$

which is the probability to start in any vertex of class y, to reach the unlabeled vertex q at time t and then to complete the walk $\lambda - t$ steps later.

The denominator of (7.38) accounts for the probability to perform D-Walks with length λ and can be computed as:

$$P\left(\mathscr{D}_\lambda^y\right) = \sum_{q' \in \mathscr{L}_y} \alpha^y(q', \lambda). \tag{7.40}$$

Plugging (7.39) and (7.40) into (7.38), we get:

$$\mathbb{E}\left[\mathrm{pt}(q)\middle|\mathscr{D}_\lambda^y\right] = \frac{\sum_{t=1}^{\lambda-1} \alpha^y(q,t)\beta^y(q,\lambda-t)}{\sum_{q'\in\mathscr{L}_y} \alpha^y(q',\lambda)}. \tag{7.41}$$

The bounded betweenness measure based on walks up to length λ is obtained as an expectation of the betweennesses for all length $1 \le l \le \lambda$:

$$\begin{aligned}\mathbf{B}_\lambda(q,y) &= \sum_{l=1}^{\lambda} \frac{P\left(\mathscr{D}_l^y\right)}{Z}\mathbb{E}\left[\mathrm{pt}(q)\middle|\mathscr{D}_l^y\right] \\ &= \frac{\sum_{l=1}^{\lambda}\sum_{t=1}^{l-1} \alpha^y(q,t)\beta^y(q,l-t)}{\sum_{l=1}^{\lambda}\sum_{q'\in\mathscr{L}_y} \alpha^y(q',l)}.\end{aligned} \tag{7.42}$$

Finally, the decision process consists in classifying unlabeled vertices using a maximum a posteriori (MAP) decision rule from the betweenness computed for each class $y \in \mathscr{Y}$.

7.4 Chapter Remarks

Semi-supervised learning concerns with how computers and natural systems such as humans learn in the presence of both labeled and unlabeled data. Semi-supervised learning is of great interest in machine learning and data mining because it can use readily available unlabeled data to improve supervised learning tasks when the labeled data is scarce or expensive. Semi-supervised learning also displays potential when conceived as a quantitative tool to understanding categorical human learning, in which most of the input or received information is self-evidently unlabeled. In order to effectively use the unlabeled data in the learning process, we have seen that some assumptions on the unlabeled data must be satisfied, such as the cluster, smoothness, and manifold assumptions.

The most active area of research in the field of semi-supervised learning has been related to methods based on graphs or networks. A noticeable advantage of using networks for data analysis is the ability to reveal the topological structure of the data set. Once a graph is constructed, the goal is to propagate labels from labeled to unlabeled instances in accordance with a diffusive process. Many graph-based methods can be expressed in terms of a regularization framework, in which the goal is to minimize a cost or energy function that is composed of two terms: the loss and regularization functions. The loss function tends to penalize decisions that flip the labels of pre-labeled vertices. In contrast, the regularization function is responsible for modeling the cost of propagating labels to unlabeled vertices. Given that many algorithms rely on the smoothness assumption, this function must be smooth in dense regions of the network. A main drawback of these techniques is that a matrix inversion operation if often necessary in the propagation process. Hence, the application of these techniques in large-scale graphs is reduced.

In the case study of semi-supervised learning in Chap. 10, we present an alternative semi-supervised learning technique that relies on a competitive-cooperative process among several particles. These particles navigate in the network forming teams. The goal of each team is to conquer new vertices, while also defending their previously conquered vertices. The visiting process of the particles has an analogy with the label propagation process in these regularization frameworks. However, the particle navigation does not need matrix inversion, which enables us to use it in large-scale problems. In addition, we show that the particle competition algorithm can be used to detect and prevent error propagation. In this regard, the algorithm considers that the initial beliefs or pre-labeled instances are not totally reliable. In the learning process, the algorithm then rearranges these labels whenever it spots non-smoothness in the learning problem.

References

1. Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., Aly, M.: Video suggestion and discovery for youtube: taking random walks through the view graph. In: Proceedings of the 17th International Conference on World Wide Web, WWW '08, pp. 895–904. Association for Computing Machinery, New York, NY (2008)
2. Belkin, M., Matveeva, I., Niyogi, P.: Regularization and semi-supervised learning on large graphs. In: Shawe-Taylor, J., Singer, Y. (eds.) Learning Theory, Lecture Notes in Computer Science, vol. 3120, pp. 624–638. Springer, Berlin, Heidelberg (2004)
3. Belkin, M., Niyogi, P., Sindhwani, V.: On manifold regularization. In: Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTAT 2005), pp. 17–24. Society for Artificial Intelligence and Statistics, Cliffs, NJ (2005)
4. Belkin, M., Niyogi, P., Sindhwani, V.: Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J. Mach. Learn. Res. **7**, 2399–2434 (2006)
5. Blum, A., Chawla, S.: Learning from labeled and unlabeled data using graph mincuts. In: Proceedings of the Eighteenth International Conference on Machine Learning, pp. 19–26. Morgan Kaufmann, San Francisco (2001)
6. Blum, A., Lafferty, J., Rwebangira, M.R., Reddy, R.: Semi-supervised learning using randomized mincuts. In: Proceedings of the Twenty-first International Conference on Machine Learning, p. 13. Association for Computing Machinery, New York, NY (2004)
7. Callut, J., Françoise, K., Saerens, M., Duppont, P.: Semi-supervised classification from discriminative random walks. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Lecture Notes in Artificial Intelligence, vol. 5211, pp. 162–177 (2008)
8. Chapelle, O., Schölkopf, B., Zien, A. (eds.): Semi-supervised Learning. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA (2006)
9. Cupertino, T.H., Gueleri, R., Zhao, L.: A semi-supervised classification technique based on interacting forces. Neurocomputing **127**, 43–51 (2014)
10. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. Inf. Sci. **180**(10), 2044–2064 (2010)
11. Gazi, V., Passino, K.M.: Stability analysis of swarms. IEEE Trans. Autom. Control **48**, 692–697 (2003)

12. Grady, L.: Random walks for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **28**(11), 1768–1783 (2006)
13. Joachims, T.: Transductive learning via spectral graph partitioning. In: Proceedings of International Conference on Machine Learning, pp. 290–297. Association for the Advancement of Artificial Intelligence Press, Palo Alto, CA (2003)
14. Karypis, G., Han, E.H., Kumar, V.: Chameleon: Hierarchical clustering using dynamic modeling. Computer **32**(8), 68–75 (1999)
15. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, pp. 282–289. Morgan Kaufmann Publishers Inc., San Francisco, CA (2001)
16. Liu, Y., Passino, K.M., Polycarpou, M.: Stability analysis of one-dimensional asynchronous swarms. IEEE Trans. Autom. Control **48**, 1848–1854 (2003)
17. Ng, A.Y., Jordan, M.I.: On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes. In: Dietterich, T., Becker, S., Ghahramani, Z. (eds.) Advances in Neural Information Processing Systems, vol. 14, pp. 841–848. MIT Press, Cambridge, MA (2002)
18. Rabiner, L., Juang, B.H.: Fundamentals of Speech Recognition. Prentice-Hall, Englewood Cliffs (1993)
19. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. Chapman & Hall/CRC, Boca Raton (2007)
20. Silva, T.C., Zhao, L.: Network-based high level data classification. IEEE Trans. Neural Netw. Learn. Syst. **23**(6), 954–970 (2012)
21. Silva, T.C., Zhao, L.: Network-based stochastic semisupervised learning. IEEE Trans. Neural Netw. Learn. Syst. **23**(3), 451–466 (2012)
22. Silva, T.C., Zhao, L.: Semi-supervised learning guided by the modularity measure in complex networks. Neurocomputing **78**(1), 30–37 (2012)
23. Silva, T.C., Zhao, L.: Stochastic competitive learning in complex networks. IEEE Trans. Neural Netw. Learn. Syst. **23**(3), 385–398 (2012)
24. Silva, T.C., Zhao, L.: Uncovering overlapping cluster structures via stochastic competitive learning. Inf. Sci. **247**, 40–61 (2013)
25. Silva, T.C., Zhao, L.: High-level pattern-based classification via tourist walks in networks. Inf. Sci. **294**(0), 109–126 (2015). Innovative Applications of Artificial Neural Networks in Engineering
26. Singh, A., Nowak, R.D., Zhu, X.: Unlabeled data: now it helps, now it doesn't. In: The Conference on Neural Information Processing Systems NIPS, pp. 1513–1520 (2008)
27. Singla, P., Domingos, P.: Discriminative training of Markov logic networks. In: Proceedings of the 20th National Conference on Artificial Intelligence, AAAI'05, vol. 2, pp. 868–873. Association for the Advancement of Artificial Intelligence Press, Menlo Park, CA (2005)
28. Stevens, K.: Acoustic Phonetics. MIT Press, Cambridge, MA (2000)
29. Szummer, M., Jaakkola, T.: Partially labeled classification with Markov random walks. In: Advances in Neural Information Processing Systems, vol. 14, pp. 945–952 (2001)
30. Talukdar, P.P., Crammer, K.: New regularized algorithms for transductive learning. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II, ECML PKDD '09, pp. 442–457. Springer, Berlin, Heidelberg (2009)
31. Vidal, R., Tron, R., Hartley, R.: Multiframe motion segmentation with missing data using powerfactorization and GPCA. Int. J. Comput. Vis. **79**(1), 85–105 (2008)
32. Wang, F., Zhang, C.: Label propagation through linear neighborhoods. IEEE Trans. Knowl. Data Eng. **20**(1), 55–67 (2008)
33. Wang, F., Li, T., Wang, G., Zhang, C.: Semi-supervised classification using local and global regularization. In: AAAI'08: Proceedings of the 23rd National Conference on Artificial Intelligence, pp. 726–731. Association for the Advancement of Artificial Intelligence Press, Palo Alto, CA (2008)

34. Wu, M., Schölkopf, B.: Transductive classification via local learning regularization. In: 11th International Conference on Artificial Intelligence and Statistics, pp. 628–635. Microtome, Brookline, MA (2007)
35. Zhou, D., Bousquet, O., Lal, T.N., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: Advances in Neural Information Processing Systems, vol. 16, pp. 321–328. MIT Press, Cambridge, MA (2004)
36. Zhu, X.: Semi-supervised learning literature survey. Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison (2005)
37. Zhu, X.: Semi-supervised learning with graphs. Doctoral thesis, Carnegie Mellon University CMU-LTI-05-192 (2005)
38. Zhu, X., Ghahramani, Z.: Learning from labeled and unlabeled data with label propagation. Tech. Rep. CMU-CALD-02-107, Carnegie Mellon University, Pittsburgh (2002)
39. Zhu, X., Goldberg, A.B.: Introduction to Semi-Supervised Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, San Rafael, CA (2009)
40. Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised learning using gaussian fields and harmonic functions. In: International Conference on Machine Learning, pp. 912–919 (2003)