Chapter 4
Attribute-Based Decision Graphs and Their Roles in Machine Learning Related Tasks

João Roberto Bertini Junior and Maria do Carmo Nicoletti

Abstract Recently, new supervised machine learning algorithm has been proposed which is heavily supported by the construction of an attribute-based decision graph (AbDG) structure, for representing, in a condensed way, the training set associated with a learning task. Such structure has been successfully used for the purposes of classification and imputation in both, stationary and non-stationary environments. This chapter provides a detailed presentation of the motivations and main technicalities involved in the process of constructing AbDGs, as well as stresses some of the strengths of this graph-based structure, such as robustness and low computational costs associated with both, training and memory use. Given a training set, a collection of algorithms for constructing a weighted graph (i.e., an AbDG) based on such data is presented. The chapter describes in details algorithms involved in creating the set of vertices, the set of edges and, also, assigning labels to vertices and weights to edges. Ad-hoc algorithms for using AbDGs for both, classification or imputation purposes, are also addressed.

Keywords Attribute-based decision graphs · Graph-based data representation · Imputation of missing data · Data classification

J. R. Bertini Junior (✉)
School of Technology, University of Campinas, R. Paschoal Marmo 1888,
Jd. Nova Itália, Limeira, SP 13484-332, Brazil
e-mail: bertini@ft.unicamp.br

M. do Carmo Nicoletti
Campo Limpo Paulista School, R. Guatemala 167, Campo Limpo Paulista,
SP 13231-230, Brazil
e-mail: carmo@cc.faccamp.br

M. do Carmo Nicoletti
Computer Science Department, Federal University of São Carlos,
Rodovia Washington Luís s/n, São Carlos, SP 13565-905, Brazil

U. Stańczyk et al. (eds.), *Advances in Feature Selection for Data and Pattern Recognition*, Intelligent Systems Reference Library 138,
https://doi.org/10.1007/978-3-319-67588-6_4

4.1 Introduction

In the Machine Learning (ML) area, two broad groups of algorithms can be considered, referred to as supervised and unsupervised algorithms. Supervised algorithms use a particular information associated to each training instance, referred to as *class*; such algorithms induce knowledge representations which are conventionally known as classifiers. Usually unsupervised algorithms do not require the *class* information; most of them can be characterized as clustering algorithms which, given as input a set of training instances induce, as output, a set of disjoint sets of such instances (i.e., a clustering). In an unsupervised context, the inductive process can be viewed as the provider of some sort of organization to the given data; the concept of similarity (or dissimilarity) is used to guide the grouping of similar instances [33].

In the context of supervised automatic learning as well as of the several supervised methods focused on this chapter, there are mainly two kinds of data which can be organized as graph-based structures. Data that naturally reflect a graph structure are the so-called relational data [27], and the commonly available data, usually described as vectors of attribute values, referred to as vector-based data. Lately, there has been an increasing number of machine learning tasks addressed by graph-based approaches (see e.g. [1, 11]). Graph-based approaches have been adopted in supervised classification tasks in works such as [10, 25].

Also, the capability of a graph-based structure to model data distribution has been explored in the context of unsupervised learning, involving clustering tasks [33]. There is an emphasis on graph-based representation in both, unsupervised and semi-supervised learning environments, as the basic structure to model knowledge, particularly in semi-supervised tasks, such as transduction and induction [12, 16]. Reference [38] describes a semi-supervised learning framework based on graph embedding. Within the complex network theory [31], for instance, large data sets can be clustered using a community detection algorithm, such as in [19, 28, 37]. In [23, 24] the graph-based relational learning (GBRL) is discussed as a subarea of graph-based data mining (GBDM), which conceptually differs from logic-based relational learning, implemented by, for example, inductive logic programming algorithms [29, 30]. As pointed out in [23], GBDM algorithms tend to focus on finding frequent subgraphs i.e., subgraphs in the data whose number of instances (they represent) are above some minimum support; this is distinct from a few GBRL developed systems which, typically, involve more than just considering the frequency of the pattern in the data, such as the Subdue [15] and the GBI [39].

So far in the literature, research work having focus on the process of graph construction, for representing a particular training set of vector-based data, has not yet attracted the deserved scientific community attention; this is particularly odd, taking into account the crucial role that data representation plays in any automatic learning process [40]. The way a training data is represented has a deep impact on its further use by any learning algorithm. Although one can find several works where graphs have been used as structures for representing training sets, the many ways of using graphs' representational potentialities have not been completely explored yet.

Invariably, most of the available graph-based learning algorithms are restricted, considering they employ only one out of a few algorithms for creating the graph that represents a given training set. Also, the graph construction methods surveyed so far always represent each training instance as a vertex and, then, define edges between vertices as a way of representing some sort of similarity between them. Graphs constructed in this way are referred to as *data graphs*. Among the most popular methods for constructing graphs are those that construct KNN graphs and ϵ-graphs [14, 17], as well as fully connected weighted graphs, where weights are defined by a given function, such as the Gaussian, as in [41]. Regardless of being a suitable solution for data mining related problems, these kind of graphs are still tied to their inherent advantages and disadvantages.

As already mentioned, generally methods for constructing graphs rely on some *ad-hoc* concept of neighborhood, which commonly gives rise to local structures within the data set; the global data structure is then left to be addressed by the learning algorithm [22]. Proposals contemplating new types of graph-based structures for representing data and, also, algorithms for dealing with them, will certainly contribute for further progress in areas such as data mining and automatic learning. As pointed out in [3], graph-based representations of vector-based data are capable of modelling arbitrary local data configurations, enabling the learning algorithm best capture the underlying data distribution.

A new data structure for storing relevant information about training data sets was proposed in [5] and is referred to as *Attribute-based Data Graph (AbDG)*; an AbDG models a given vector-based training data set as a weighted graph. The main motivation for proposing the AbDG was to devise a data structure compact and easily manageable, able to represent and condense all information present in a given training data set, which could also be used as the source of information for classification tasks. This chapter addresses and reviews Attribute-based Data Graph (AbDG) as well as the two task-oriented subjacent algorithms associated with AbDGs; the first that constructs the graph representing a giving training set of vector-based instances and, the second, that uses the graph for classification purposes.

Besides the Introduction section, this chapter is organized into six more sections. Section 4.2 introduces the main technicalities involved for the establishment of the concept of Attribute-based Decision Graph, focusing on the proposal of two possible graph structures namely a *p-partite*, in Sect. 4.2.1.1, and a *complete p-partite*, in Sect. 4.2.1.2, where p refers to the number of attributes that describe a vector-based training instance. Formal notation is introduced and the processes involved in the AbDG construction are described. The section first discusses the construction of the vertex set, given a vector-based data set and, then, the two possible strategies for inserting edges connecting vertices. Once the construction of the graph-structure is finished, the process that assigns labels to vertices and its counterpart, that assigns weights to edges, complete and finalize the graph construction process. Section 4.3 details the process of using the information embedded in an AbDG for classification purposes and, for that, defines the classification process as some sort of graph matching process between the AbDG and one of its subgraphs i.e., the one defined by the instance to be classified. Section 4.4 presents a numerical example of the

induction of an AbDG aiming at a classification task and the use of the induced AbDG in the processes of classifying a new unclassified instance. Section 4.5 reviews the convenience of using AbDGs when learning from data sets with absent attribute values. Section 4.6 approaches the process of constructing an AbDG as a search task conducted by a genetic algorithm (GA) and, finally, Sect. 4.7 resumes the main contributions of the AbDG approach, highlighting some of its main advantages and the technicalities involved in the construction/use of such structure. The section ends with some new insights for continuing the research work with focus on AbDGs.

4.2 The Attribute-Based Decision Graph Structure (AbDG) for Representing Training Data

The AbDG is a graph-based structure proposed in [5] and extended in [9], aiming at modelling data described as vectors of attribute values; such structure has been later employed in several machine learning related tasks, such as those presented in [5–8].

4.2.1 Constructing an AbDG

Consider a training data set X, where each instance $\mathbf{x} = (x_1, \ldots, x_p, c)$ in X is a p-dimensional data vector of features followed by a class label $c \in \{\omega_1, \ldots \omega_M\}$, representing one among M classes. The process that constructs the AbDG for representing X initially focuses on the construction of the set of vertices, then on the construction of the set of edges and finally, on assigning labels to vertices and weights to edges of the induced graph, turning it into a weighted labeled data graph representing X.

As mentioned before, given a data set X having N p-dimensional instances, most approaches for constructing the set of vertices of a graph that represents X, usually define each data instance in X as a vertex of the graph being constructed, resulting in a graph with N vertices. Approaches adopting such a strategy can be found in [2, 33, 36]. In an AbDG graph the vertices represent data intervals associated with values the attributes that describe training instances can have. Thus, once attribute A_a has been divided into n_a intervals, it can be viewed as a set of disjoint intervals $A = \{I_{a,1}, \ldots, I_{a,n_1}\}$ where each interval $I_{a,i}$ stands for vertex $v_{a,i}$ in the graph.

Due to vertices being defined by data intervals, the construction of an AbDG is heavily dependent on the type of the attributes used for describing the training instances. Basically three types of attributes are commonly used for describing a given training set X, referred to as numerical (continuous-valued), categorical (whose possible values are limited and usually fixed, having no inherent order) and ordinal (whose possible values follow a particular pre-established order).

The construction of the set of vertices of an AbDG graph, representing a given data set X, starts by discretizing the values of each one of the p attributes $\{A_1, A_2, A_3, \ldots A_p\}$ that describes X. A discretization process applied to each one of the p attributes associates, to each of them, a set of disjoint intervals of attribute values. As the set of intervals associated to each attribute A_i ($i = 1, \ldots p$) depends on the type of the attribute A_i (i.e., categorical, numerical (continuous-valued) or ordinal), as well as the range of values of A_i in X, a discretization method should deal with the different types of attributes.

If an attribute is categorical, the simplest way to create its associated set of vertices is by considering each of its possible values as a degenerate interval; this is the approach used for constructing AbDGs. As an example, if the values of attribute A_5 in X (taking into account all instances in X) are 0, 1, 2, 3, 4 and 5, during the construction of the vertex set, the set of degenerate intervals: {[0 0], [1 1], [2 2], [3 3], [4 4], [5 5]} is associated with A_5 and each interval is considered a vertex in the graph under construction.

When the attribute is numerical, the usual basic procedure a discretization method adopts is to divide the attribute range into disjoint intervals. Several discretization methods found in the literature can be used to create such set of disjoint intervals associated with a continuous-valued attribute [21].

As pointed out in [9] in relation to the problem of discretizing a continuous-valued attribute, the solution starts by finding a set of what is called *cut point candidates* in the range of the attribute, then to use a heuristic to evaluate the potentialities of the selected cut point candidates and, finally, choose the most promising subset as the actual cut points for defining the intervals. Cut point candidates are determined by sorting each attribute and then, searching for consecutive different attribute values whose corresponding instances belong to different classes, a process formalized as follows. Consider the values of attribute A and let sA represent an ordered version of the values of A. The process can be formally stated as if $sA[i] \neq sA[i+1]$ and $class\_instance(sA[i]) \neq class\_instance(sA[i+1])$, where $class\_instance()$ gives the class of the data instance having $A[i]$ as value for attribute A, then determine the middle point between values $sA[i]$ and $sA[i+1]$ and assume the obtained value as a cut point. Once the vertex set has been built, edges are then established by taking into account the corresponding attribute values of patterns in X, aiming at connecting intervals (i.e. vertices) to reflect the correlations between different attributes [13]. Taking into account a p-dimensional data set, two edge structures are considered, which give rise to two different graph structures, the p-partite (Sect. 4.2.1.1) and the complete p-partite (Sect. 4.2.1.2).

4.2.1.1 The AbDG as a p-Partite Graph

Given a data set X and considering that the sets of vertices associated with each attribute have already been created, the induction of a p-partite graph, for representing X, assumes a pre-defined order among the attributes, which can be randomly established or, then, by sorting the attributes according to some criteria. So, given

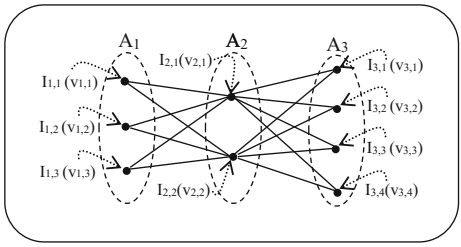

Fig. 4.1 A 3-partite AbDG structure created from a data set X with N vector-based instances, each described by 3 attributes, A_1, A_2 and A_3. The discretization process associates to each attribute A_i ($1 \leq i \leq 3$), n_i intervals (i.e., vertices), $n_1 = 3$, $n_2 = 2$ and $n_3 = 4$

the pre-defined attribute order and the sets of vertices associated to each attribute, edges can only be introduced between two consecutive (taking into account the given order) vertices. The whole graph structure is affected by the established order.

It has been empirically verified that sorting the attributes according to the descent order of their corresponding information gain values can be a convenient choice (see [9]) since it promotes and maintains connections between attributes with highest information gains. Figure 4.1 shows an example of a *3-partite* AbDG associated with a hypothetical training set X defined by three attributes A_1, A_2 and A_3 and an associated class, where the discretization process associated to each attribute A_i ($1 \leq i \leq 3$) produced, respectively, n_i intervals (i.e., vertices), namely $n_1 = 3$, $n_2 = 2$ and $n_3 = 4$. Figure 4.2 shows a high level pseudocode for creating a p-partite AbDG structure, given a data set with N instances described as p-dimensional vectors and an associated class, out of M possible classes.

4.2.1.2 The AbDG as a Complete p-Partite Graph

When constructing the complete p-partite structure, however, the attribute order is irrelevant, due to the intrinsic nature of complete p-partite graphs. In a complete p-partite graph all possible edges between intervals (i.e., vertices) associated with different attributes are inserted in the graph under construction. Figure 4.3 shows an example of a *3-partite* AbDG associated with a hypothetical training set X, defined by three attributes A_1, A_2 and A_3 and an associated class.

4.2.2 Assigning Weights to Vertices and Edges

This section gives the motivations for introducing labels and weights in an AbDG structure, and explains how labels associated with vertices and weights associated with edges of an AbDG are defined. Let X be a data set having N training instances from M different classes, $\{\omega_1, \omega_2, \ldots, \omega_M\}$, where each instance is described by p attributes and an associated class.

*Procedure construct_p_partite_AbDG(*X, λ)
Input:
X: data set with *N* instances, each described by *p* attributes {A_1, A_2, ..., A_p} and an associate class, from *M* possible classes.
λ: user defined attribute order.

% creating the set of vertices of the p-partite AbDG
Graph ← ∅
Vertices ← ∅
for *a* ← 1 to *p* **do**
begin
{$I_{a,1}$, $I_{a,2}$, ..., I_{a,n_a}} ←*discretize_range*(A_a)
Vertices ← Vertices ∪ {$I_{a,1}$, $I_{a,2}$, ..., I_{a,n_a}}
end

% creating the set of edges of the p-partite AbDG
{A_1, A_2, ..., A_p} ← *ordering_and_renaming_attrib*({A_1, A_2, ..., A_p},λ)
Edges ← ∅
for_all $x_b \in X$ **do**
for a ← 1 to *p* – 1 **do**
% *interval* $I_{a,k}$ *represents vertex* $v_{a,k}$ *and interval* $I_{a+1,q}$ *vertex* $v_{a+1,q}$
if $x_{b,a} \in I_{a,k}$ and $x_{b,a+1} \in I_{a+1,q}$ **then**
begin
($v_{a,k}$, $v_{a+1,q}$) ← *create_edge*($v_{a,k}$, $v_{a+1,q}$)
Edges ← Edges ∪ {($v_{a,k}$, $v_{a+1,q}$)}
end
Graph ← (Vertices,Edges)
Graph ← *assign_weights*(Graph)
Output: Graph {a *p*-partite AbDG}

Fig. 4.2 High-level pseudocode for constructing a *p*-partite AbDG structure

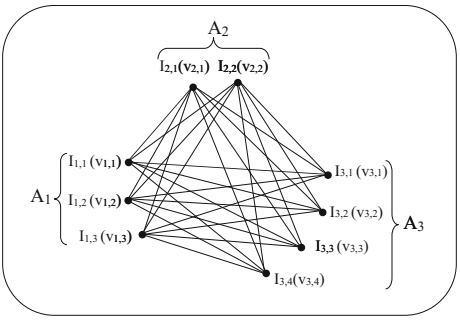

Fig. 4.3 A complete 3-partite AbDG structure created from a data set X with N vector-based instances, each described by 3 attributes, A_1, A_2 and A_3. The discretization process applied to each attribute associates to each attribute A_i $(1 \leq i \leq 3)$, n_i intervals (i.e., vertices), namely $n_1 = 3$, $n_2 = 2$ and $n_3 = 4$. Edges are created between all vertices except those related with the same attribute

After the AbDG structure has been constructed, as a p-partite or complete p-partite, M-dimensional vectors are created and associated to each vertex and to each edge of the AbDG structure, by the two processes described next. In real domain data it is not common that instances sharing the same class are the only ones that have values of an attribute A_i ($1 \leq a \leq p$) within one of the n_a subintervals (i.e., $I_{a,1}, \ldots, I_{a,n_a}$). Instances belonging to other classes may as well have their A_i values within that same subinterval. Aiming at evaluating the *representativeness* (i.e., the information for characterizing the class of an instance) of each particular subinterval created (i.e., each vertex of the AbDG), a M-dimensional weight vector is associated to each vertex.

4.2.2.1 Assigning Weights to Each Vertex of the AbDG

Let n_a represent the number of vertices associated with attribute A_a. The vertex $v_{a,k}$ ($1 \leq a \leq p$ and $1 \leq k \leq n_a$) has an associated weight vector given by $\Gamma_{a,k} = \langle \gamma_1, \ldots, \gamma_j, \ldots, \gamma_M \rangle$, where γ_j relates to class ω_j, noted as $\Gamma_{a,k}(j)$. Considering that $I_{a,k}$ is the interval that defines $v_{a,k}$, $\Gamma_{a,k}(j)$ is defined by Eq. (4.1).

$$\Gamma_{a,k}(j) = P(\omega_j | I_{a,k}) = \frac{P(I_{a,k}, \omega_j)}{P(I_{a,k})} \tag{4.1}$$

The joint probability in Eq. (4.1), $P(I_{a,k}, \omega_j)$, is the probability of a data instance having both i.e., class ω_j and its value of attribute A_a in the interval $I_{a,k}$. By rewriting the joint probability as $P(I_{a,k}, \omega_j) = P(\omega_j)P(I_{a,k}|\omega_j)$, the conditional probability $P(I_{a,k}|\omega_j)$ can be given by Eq. (4.2). $P(\omega_j)$ is the marginal probability of class ω_j, obtained by dividing the number of data instances belonging to class ω_j by the total number of data instances (i.e., N).

$$P(I_{a,k}|\omega_j) = \frac{|\{\mathbf{x}_i \in X \mid x_{i,a} \in I_{a,k} \wedge c_i = \omega_j\}|}{|\{\mathbf{x}_i \mid c_i = \omega_j\}|} \tag{4.2}$$

In Eq. (4.1) the marginal probability, $P(I_{a,k})$, is the normalizing term, defined as the sum of the probabilities $P(I_{a,k}|\omega_j)$, for all possible classes i.e., $P(I_{a,k}) = \sum_{i=1}^{M} P(I_{a,k}|\omega_i)$.

4.2.2.2 Assigning Weights to Each Edge of the AbDG

The procedure for assigning a weight to an AbDG's edge is similar to the one for assigning a label to a vertex. Let $(v_{a,k}, v_{b,q})$ be an edge between the vertices representing the kth interval of attribute A_a and the qth interval of attribute A_b, and let this edge be weighted by the weight vector $\Delta_{k,q}^{a,b} = \langle \delta_1, \ldots, \delta_M \rangle$, where δ_j ($1 \leq j \leq M$) is associated to class ω_j, noted as $\Delta_{k,q}^{a,b}(j)$. The edge weight δ_j ($1 \leq j \leq M$) repre-

sents the probability of a given data instance $\mathbf{x}_i$, with attribute value $x_{i,a} \in I_{a,k}$ and $x_{i,b} \in I_{b,q}$, belonging to class ω_j, as given by Eq. (4.3).

$$\Delta_{k,q}^{a,b}(j) = P(\omega_j | I_{a,k}, I_{b,q}) = \frac{P(I_{a,k}, I_{b,q}, \omega_j)}{P(I_{a,k}, I_{b,q})} \tag{4.3}$$

Considering that $P(I_{a,k}, I_{b,q}, \omega_j) = P(\omega_j)P(I_{a,k}, I_{b,q}|\omega_j)$, then define $P(I_{a,k}, I_{b,q}|\omega_j)$ as the ratio of the number of instances belonging to class ω_j, whose values of attribute A_a lay within the kth interval and those of the attribute A_b lay within the qth interval, as in Eq. (4.4).

$$P(I_{a,k}, I_{b,q}|\omega_j) = \frac{|\{\mathbf{x}_i \in X | c_i = \omega_j \wedge x_{i,a} \in I_{a,k} \wedge x_{i,b} \in I_{b,q}\}|}{|\{\mathbf{x}_i | c_i = \omega_j\}|} \tag{4.4}$$

The probability of a data instance to have attribute values belonging to interval $I_{a,k}$ and $I_{b,q}$, regardless its class label, is the normalizing term in Eq. (4.3) and is given by the sum of Eq. (4.4), over all classes, as states Eq. (4.5).

$$P(I_{a,k}, I_{b,q}) = \sum_{j=1}^{M} P(I_{a,k}, I_{b,q}, \omega_j) \tag{4.5}$$

4.2.3 Computational Complexity for Building an AbDG

The complexity order for constructing an AbDG has been completely derived in Refs. [8, 9]. In what follows, a brief overview on the computational complexity required to build the AbDG is presented. Consider the complexity order with respect to the size of the training set, N, and to the number of attributes, p; thus building the AbDG involves:

1. **Constructing the vertex set**, which depends on the employed discretization method. As sorting is usually required as a preprocessing step to various discretization methods, building the vertex set has order of $O(pNlogN)$. If $p << N$, which is true for most domains, than building the vertex set has order of $O(NlogN)$; otherwise, if $p \approx N$ it can scale up to the order of $O(N^2logN)$.
2. **Defining the weights of an AbDG** for vertices and edges, has complexity order of $O(N)$. The complexity order associated to the number of attributes for vertex weighting has order of $O(p)$. Edge weighting depends on the graph structure; for the p-partite structure the order is linear on the number of attributes, $O(p)$, while for the complete p-partite, the complexity order of edge weighting scales quadratically to the number of attributes, $O(p^2)$.

Therefore, building an AbDG has an order of $O(NlogN)$ with the size of the data set, when the discretization method requires sorting. What is costly about

building the graph is sorting each attribute prior to apply some discretization method, as the MDLPC [18] for instance, to obtain the vertices. However, if some heuristic is employed, as dividing each attribute into subintervals of equal length, and thus not requiring sorting, building the vertex set has complexity of $O(N)$. Results regarding both ways to build the graph are reported in [9]; indeed, the equi-sized version presented comparative, or even better results than those versions which employ a discretization method.

4.3 Using the AbDG Structure for Classification Tasks

Once the construction of an AbDG for representing a given data set X of instances has finished, it can be used as the source of information on X for various tasks; among them, it can support a classifier, provided a procedure for exploring the information stored in the AbDG is defined. This section focuses on the description of such procedure. Taking into account a given AbDG, the assignment of classes to new unclassified instances (which can be modelled as a p-partite sub-graph of the given AbDG), can be conducted by checking how the subgraph defined by an instance, conforms to the existing connection patterns in the AbDG, embedded in its structure. As a consequence, the graph structure has a vital importance on the classification accuracy of AbDG-based classifiers.

Among the various possible ways to combine the information given by an AbDG, the proposal described next has shown to be a sound alternative for implementing a classification process based on a AbDG, and is based on calculating the product of vertex weights and the sum of edge weights. Consider classifying a new data instance $\mathbf{y}$. Given the AbDG and $\mathbf{y}$, two conditional probabilities, $P(\mathbf{y}|\omega_i)$ and $Q(\mathbf{y}|\omega_i)$, can be calculated. $P(\mathbf{y}|\omega_i)$ relates $\mathbf{y}$ to the vertex set of the AbDG, and $Q(\mathbf{y}|\omega_i)$ relates $\mathbf{y}$ to the edge set of the AbDG. Equations (4.6) and (4.7) describe both probabilities, respectively, for a p-partite AbDG.

$$P(\mathbf{y}|\omega_i) = \frac{PW(\mathbf{y})_{\omega_i}}{\sum_{j=1}^{M} PW(\mathbf{y})_{\omega_j}} \; ; \qquad PW(\mathbf{y})_{\omega_i} = \prod_{a=1, y_a \in I_{a,k}}^{p} \gamma_i \in \Gamma_{a,k} \tag{4.6}$$

$$Q(\mathbf{y}|\omega_i) = \frac{SW(\mathbf{y})_{\omega_i}}{\sum_{j=1}^{M} SW(\mathbf{y})_{\omega_j}} \; ; \qquad SW(\mathbf{y})_{\omega_i} = \sum_{a=1, b=a+1, y_a \in I_{a,k} \wedge y_b \in I_{b,q}}^{p-1} \delta_i \in \Delta_{k,q}^{a,b} \tag{4.7}$$

After determining both probabilities, an estimate for the class label of the $\mathbf{y}$ instance is given by Eq. (4.8). The class inferred for the new data $\mathbf{y}$, noted $\varphi(\mathbf{y})$, is the one having the greatest value for the mean of the normalized probabilities.

$$\varphi(\mathbf{y}) = \arg \max_{\{\omega_j | j=1,\ldots,M\}} \Big(\eta P(\mathbf{y}|\omega_j) + (1-\eta) Q(\mathbf{y}|\omega_j) \Big) \tag{4.8}$$

In Eq. (4.8), η is a parameter of the classifier which allows to vary the emphasis between the conditional probabilities and enhances the flexibility of the model. The classification of $\mathbf{y}$ can be approached as a graph matching process. First the p attribute values of $\mathbf{y}$ help to detect the corresponding interval associated with each of the attributes i.e., the p vertices defining the subgraph representing $\mathbf{y}$. Then edges connecting each pair of vertices in sequence are added, resulting in a subgraph structure of the AbDG. Then, the weights associated to the graph help to define the class of $\mathbf{y}$, as the one that promotes the best matching.

An in-depth detailed analysis of the complexity orders of AbDG-related algorithms, considering both structures, the p-partite and the complete p-partite, is presented in [9]. Both structures share the same process for the vertex set definition, as well as the corresponding vertex labeling process, and differ in relation to the set of edges they have. In the referred work, both structures are approached separately when dealing with the complexity of the algorithm for constructing the edge set and the corresponding weighting process. Also, in [9], the authors present an empirical validation of the AbDG-based algorithms by conducting an experimental comparative analysis of classification results with other five well-known methods namely, the C4.5 [32], the multi-interval ID3 [18], the weighted KNN [20], the Probabilistic Neural Networks [34] and the Support Vector Machine [35]. Statistical analyses conducted by the authors show evidence that the AbDG approach has significantly better performance than four out of the five chosen algorithms.

4.4 Using an AbDG for Classification Purposes - A Case Study

This section presents a simple example illustrating the classification process based on an AbDG. Figure 4.4 shows a 3-partite AbDG structure, similar to the one depicted in Fig. 4.1, but now with the associated values for the intervals. In the figure, A_1 and A_2 are numerical attributes whose values are in the interval [0, 10] and [0, 1], respectively. A_3 is a categorical attribute having values in the set {0, 1, 2, 3}. Consider classifying the instance $\mathbf{y} = (5.5, 0.31, 2)$, whose match against the AbDG is shown

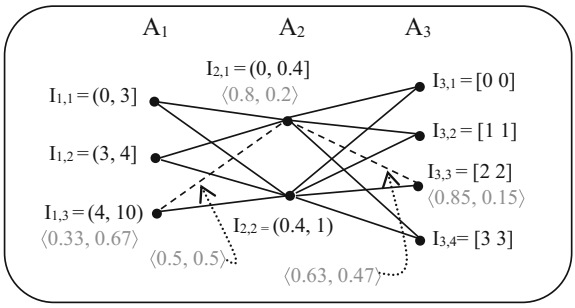

Fig. 4.4 Match of instance $\mathbf{y} = (5.5, 0.31, 2)$ to a particular 3-partite AbDG with the purpose of classification

in the figure (dashed line). For the sake of visualization, only the weights associated with matching the sub-graph representing $\mathbf{y}$ against the AbDG are displayed. The task is to classify $\mathbf{y}$ into one of two possible classes $\{\omega_1, \omega_2\}$.

The classification procedure is carried out by matching the unlabeled instance against the AbDG followed by estimating the probabilities, as stated in Eqs. (4.6) and (4.7), for all classes in the problem. In the case study, given $\mathbf{y} = (5.5, 0.31, 2)$, the following matches to the AbDG are obtained: $5.5 \in I_{1,3} = (4, 10)$; $0.31 \in I_{2,1} = (0, 0.4]$ and $2 \in I_{3,3} = [2\ 2]$. For classification purposes, when a new value falls off the attribute range it is considered to belong to the nearest one. As a consequence, the weight vectors to be used are: $\Gamma_{1,3} = \langle 0.33, 0.67 \rangle$; $\Gamma_{2,1} = \langle 0.8, 0.2 \rangle$ and $\Gamma_{3,3} = \langle 0.85, 0.15 \rangle$ and the corresponding edges weights are $\Delta_{3,1}^{1,2} = \langle 0.5, 0.5 \rangle$ and $\Delta_{1,3}^{2,3} = \langle 0.63, 0.47 \rangle$. Next, PW and SW are calculated according to Eqs. (4.6) and (4.7).

$$
\begin{aligned}
PW(\mathbf{y})_{\omega_1} &= 0.33 \times 0.8 \times 0.85 = 0.2244 \\
SW(\mathbf{y})_{\omega_1} &= 0.5 + 0.63 = 1.13 \\
PW(\mathbf{y})_{\omega_2} &= 0.67 \times 0.2 \times 0.15 = 0.0201 \\
SW(\mathbf{y})_{\omega_2} &= 0.5 + 0.47 = 0.97
\end{aligned}
$$

Therefore the probabilities for each class are given as follows,

$$
\begin{aligned}
P(\mathbf{y}|\omega_1) &= 0.2244/0.2445 = 0.918 \\
Q(\mathbf{y}|\omega_1) &= 1.13/2.1 = 0.538 \\
P(\mathbf{y}|\omega_2) &= 0.0201/0.2445 = 0.082 \\
Q(\mathbf{y}|\omega_2) &= 0.97/2.1 = 0.462
\end{aligned}
$$

Finally, considering $\eta = 0.5$, according to Eq. (4.8) $\mathbf{y}$ is classified in class ω_1, since $0.5 \times 0.918 + 0.5 \times 0.538 > 0.5 \times 0.082 + 0.5 \times 0.462$.

4.5 Using the AbDG Structure for Imputation Tasks

Missing attribute values is a common problem present in almost every kind of real world application. The most frequent solutions to handle such problem reported in the literature are: (1) remove the instances having missing attribute values; (2) employ an imputation algorithm as a preprocessing step to the learning method and (3) adopt a learning method having internal mechanisms that enable training and classifying in the presence of missing attribute values. Regarding the previous alternatives, the first is the most used and may work well for applications having a few missing values in the training set and, also, those where ignoring a test instance with a missing value is acceptable. Clearly, this method imposes too many restrictions and it can be applied to very specific tasks. Therefore, alternatives (2) and (3) are more appealing to the majority of real world applications.

*Procedure imputation_from_AbDG(G, **x**)*
Input:
G: An AbDG built from a data set X with N instances, each described by p attributes $\{A_1, A_2, ..., A_p\}$ and an associate class, from M possible classes.
x: A data instance with missing value.

for $a \leftarrow 1$ to p **do**
  begin
    if $x_a = \emptyset$ **then**
    begin
      $\boldsymbol{s}_a \leftarrow \{0\}_{n_a}$ % *each position in* $\boldsymbol{s}$ *represents an interval of* A_a *in* G
      for $b \leftarrow 1$ to p **do**
      begin
        if $x_b \neq \emptyset$ and $a \neq b$ **then**
        begin
          find the interval in G, into which x_b belongs to, referred to as q.
          knowing q, calculate $\boldsymbol{s}_{a,k}$, for $k = 1, ..., n_a$, as in Eq. (4.9).
        end
      end
    I_a = argmax $\boldsymbol{s}_{a,k}$, for $k = 1, ..., n_a$
    $x_a \leftarrow$ *infer_value*(I_a)
  end
end
Output: Completed **x**

Fig. 4.5 High-level pseudocode for conducting an imputation process through an AbDG

The AbDG approach can be employed in both situations, either as an imputation method used to infer plausible values for the missing ones, prior to some other learning algorithm [8], or as a classification method able to handle the missing values found at the training or the classification phase [9]. The core mechanism to handle missing values through an AbDG, for both tasks, is practically the same, and is outlined in Fig. 4.5.

Let G be an AbDG and **x** be a data instance with, at least, a missing attribute value, say x_a. The imputation procedure based on AbDGs aims to find the interval of A_a in G, where the value of x_a should belong to. Thus, for each one of the n_a intervals of A_a, an estimate $s_{a,k}$ $(1 \leq k \leq n_a)$ is calculated, taking into account the sub-graph resulted from the match between the existing values of **x** and the AbDG. Let $\Gamma_{a,k}(j)$ be the weight of the vertex associated to class ω_j which $s_{a,k}$ represents; and for each existing value in **x**, say x_b, laying in interval q of attribute A_b, let β_b, $\Gamma_{b,q}(j)$ and $\Delta^{a,b}_{k,q}(j)$ be the information gain (or some other attribute measure) of A_b, the weight of the vertex $v_{b,q}$ and the weight of the edge $(v_{a,k}, v_{b,q})$ associated to class ω_j, respectively; $s_{a,k}$, for a complete p-partite graph, is given by Eq. (4.9).

$$s_{a,k} = \Gamma_{a,k}(j) \sum_{b=1, b\neq a, \exists x_{i,b} \wedge x_{i,b} \in I_{b,q}}^{p} \beta_b \Gamma_{b,q}(j) \Delta^{a,b}_{k,q}(j) \tag{4.9}$$

Once all the $s_{a,k}$, $1 \leq k \leq n_a$, have been calculated, the one with the highest value indicates the most plausible interval into which the missing attribute value should belong to. If the AbDG has been used as a classifier, knowing the interval is enough to proceed. However, if it has been used as an imputation method, a step to infer an actual value is necessary. In the pseudocode given in Fig. 4.5, the procedure *infer_value()* infers a single value, given an interval as argument; possible methods which could implemented are the mean, the mode, a random number within the interval, and so on. Notice that in Eq. (4.9) the used weights are all from a single class, ω_j, which is the same class as the data instance being imputed. When imputing from unlabeled data instances, the process is carried out for all possible classes and, then, the highest estimate, considering all classes, is chosen as the one which the missing value should belong to.

As commented earlier, the AbDG has been used for imputing missing data as a preprocessing step to a learning method. It has been compared against the following imputation methods: CART, MEAN, Bayesian Linear Regression, Fast Imputation, Linear Regression and Random Forests (see [8] for details). Several imputation methods were employed as a preprocessing step for the learning algorithms: Support Vector Machines, Multinomial Logistic Regression, Naive Bayes, Multilayer Perceptron, Parzen classifier, K-nearest neighbor, Classification And Regression Tree and Probabilistic Neural Networks (details can also be found in [8]).

The AbDG has showed the overall best results and the most stable performance along a varying rate of missing values. Not only has the AbDG showed its effectiveness to deal with missing value as an imputation method but, in [9], it has been tested as a classification algorithm that automatically handles missing values. When compared to the C4.5 and CART, which are two algorithms that support missing values, the AbDG had showed superior performance and has confirmed itself as an efficient alternative to cope with missing data.

4.6 Searching for Refined AbDG Structures via Genetic Algorithms

The AbDG structures reviewed in this chapter were the p-partite structure, which has subsets of vertices consecutively connected, based on a pre-defined order of attributes, and the complete p-partite. However, several other graph-based structures can be devised for the same purpose. As pointed out before, during the creation of an AbDG, the only restriction to take into account, when creating its corresponding set of edges, is not to create edges between vertices associated with the same attribute.

As far as the construction of the AbDG edge set is concerned both structures, the p-partite and the complete p-partite, have minor drawbacks, mostly related to their fixed (although dependent on the discretization process applied to all attributes) number of edges, as well as the patterns of connections they establish. Also, add to that the fact that the algorithm for inducing a p-partite AbDG expects to be

given, as input, the order in which attributes should be considered (see Fig. 4.2). Depending on the number of attributes that describe a given data set X, the task of defining a convenient order is an extra task to be conducted previously to the induction of the AbDG.

Both structures have all possible connections between vertices, but still subject to the restriction above mentioned and, if a p-partite, to the given order of attributes. So, considering their vast number of edges, both structures become capable enough to represent all sorts of data. It has been empirically verified that, most times, although depending of the data set, such massive creation of edges is not necessary. A smaller subset of them would suffice for representing a given data set X.

As an strategy for searching for AbDGs having their set of edges customized to the given data set X, the work described in [4] explores the use of Genetic Algorithms (GAs) for inducing parts of an AbDG structure, specifically, a more customized set of edges, to the given set X. In the work a GA-based algorithm named *GA-AbDG* was proposed, for searching for a suitable edge set for a partially constructed AbDG, which only has its vertex set defined, aiming at finding a more refined set of edges, which could represent X better than both, *p-partite* and *complete p-partite*.

The GA starts with a population of N_P randomly generated individuals, where each individual is an AbDG classifier; individuals differ from each other only in relation to their edge set. The algorithm aims at identifying the best possible AbDG among all individuals in the population, using as criteria the value of their accuracy, by evolving their associated edge sets.

Let P represent a population of individuals such that $|P| = N_P$. At each iteration, a number of $N_{best} < N_P$ individuals from P are selected (based on their accuracy values in a given validation set) for composing the next population. The selected individuals are then used to create new individuals, which will, eventually, replace those considered not suitable enough (i.e., those with low accuracy values), when defining the new population. The new individuals are created by crossover up to restoring the population to its original size (i.e., N_P). At each generation, any individual, except for the overall best (elitism), is suitable to undergo the mutation operator. The evolutionary process is controlled by an user-defined number of iterations ($itMax$). At the end of the process, the AbDG with the highest accuracy is selected.

Before presenting the operators, a formal notation is introduced. An AbDG graph, $G = (V, E)$, can be viewed as a set of vertices (V) and a set of edges (E). If G is a p-partite graph, its set of vertices can be described as a set of disjoint vertex subsets, $V = \{V_1, \ldots, V_p\}$, where set V_a stands for the set of vertices obtained from discretizing the values associated with attribute A_a, $a = 1, \ldots, p$. Similarly, the edge set E can be written as the set of all edge sets between every pair of distinct attributes V_a and V_b, for $a = 1, \ldots, p-1$, $b = 2, \ldots, p$ and $b > a$, as $E = \{E_{1,2}, \ldots, E_{p-1,p}\}$. Hence, resulting in $\binom{p}{2}$ subsets, where $E_{a,b}$ comprises the set of all possible edges between vertices in V_a and V_b. The description of an AbDG as a chromosome is given by the description of its edge set. Each edge set $E_{a,b}$ can be represented by a $|V_a| \times |V_b|$ matrix. In this way, an individual is represented by a set of $\binom{p}{2}$ matrices.

```
Procedure GA-AbDG(X, Y, itMax, N, N_best, ρ, μ)
Input:
X: training labeled data set
Y: validation labeled data set
itMax: number of iterations
N: population size
N_best: number of individuals to select at each iteration
ρ: crossover rate
μ: mutation rate

begin
    P_best ← ∅
    build N AbDGs, each having a  random edge set and compose P
    it ← 1
    while it ≤ itMax do
        begin
          P ← fitness(Y)
          P_best ← selection(P,N_best)
          P ← reproduction(P_best)
          P ← crossover(P, ρ)
          P ← mutation(P,μ)
          it ← it + 1
        end
end
return: AbDG ∈ P with the highest fitness value.
```

Fig. 4.6 High-level pseudocode of the GA-AbDG procedure

As each individual in the population is an AbDG, let $G^{(i)} = (V^{(i)}, E^{(i)})$ be individual i, and straightforwardly $E^{(i)}_{a,b}$ be the set of edges between vertices from $V^{(i)}_a$ and $V^{(i)}_b$. The high level pseudocode of procedure GA-AbDG is given in Fig. 4.6. Following this notation, consider henceforward, i and j as indexes for parenting individuals, and o and m indexes for offspring individuals. In the following, each operator is described in details.

Reproduction - Reproduction is accomplished by selecting consecutive pairs of individuals ordered according to their fitness values. Each parenting pair $G^{(i)}$ and $G^{(j)}$ gives rise to two new offsprings $G^{(o)}$ and $G^{(m)}$. When obtaining each of them, each edge $(v_{a,k}, v_{b,q})$ that is common to both, $G^{(i)}$ and $G^{(j)}$, edge sets, is maintained in both offsprings i.e., $G^{(o)}$ and $G^{(m)}$. For those vertices that only belong to one of the parents, each offspring follows the configuration of one of the parents, with an associated probability of θ (whose value is a parameter to the reproduction procedure). The reproduction process implements a procedure that generates the offspring $G^{(o)}$ resembling to $G^{(i)}$; so, $G^{(o)}$ repeats the configuration of $G^{(i)}$ with probability θ and of $G^{(j)}$ with probability $1 - \theta$. Offspring $G^{(m)}$ that resembles $G^{(j)}$ is straightforward. Remember that each reproduction always generates two offspring.

Crossover - Also performed at every iteration, the crossover operator requires two parenting individuals $G^{(i)}$ and $G^{(j)}$, randomly selected from the N_{best} individuals, and also generates two offspring $G^{(o)}$ and $G^{(m)}$. Crossover is performed by randomly selecting a crossing point in the edge sets of both parents and exchanging their configuration. Crossover is considered at a rate of ρ, usually set to high values.

Mutation - At each iteration an individual has the probability μ of undergoing mutation. Mutation is implemented by randomly selecting a set of edges between two attributes e.g., $E_{a,b}$, for V_a and V_b. Then, for each possible pair in the set, with probability μ, the mutation operator is applied by adding an edge if such edge does not exist or by removing it otherwise.

The work described in [4] presents the classification results obtained with the C4.5 [32], the original AbDG, with a p-partite structure and the GA-AbDG, obtained as the result of the GA-based search process previously described, in 20 data sets from the UCI-Repository [26]. The results obtained with the GA-AbDG outperformed those obtained by both, the C4.5 and the original AbDG in 15 out of the 20 data sets used. The authors concluded that the improvements in classification performance achieved by the GA-AbDG over the original AbDG, makes the GA-based search aiming at finding a more suitable edge set worth the extra computational effort.

4.7 Conclusions

This chapter reviews a new data structure proposed in the literature as a suitable way of condensing and representing the information contained in a training set. The structure was devised to be used mainly by classification and imputation algorithms, in supervised automatic learning environments. It is named *Attribute-based Data Graph (AbDG)* and it can be described as a labeled p-partite weighted graph.

Taking into account a few other graph-based approaches for representing data, found in the literature, the main novelty introduced by AbDGs relates to the role played by the AbDG vertices. While in traditional methods vertices represent training instances, in the AbDG they represent intervals of values related to attributes that describe the training instances. The AbDG approach is a new way to build a graph from data which provides a different and more compact way of data representation for data mining tasks.

This chapter presents and discusses in detail various formal concepts and procedures related to the design, construction, and use of AbDGs, namely:

- The creation of the set of vertices of the graph, which involves the choice and use of discretization methods;
- Two different ways edges can be inserted, either constructing a *p-partite* or, then, a *complete p-partite* graph-based structure;
- Several technicalities and formal concepts involved in vertex labeling and edge weighting procedures, which play a fundamental role in adjusting the AbDG structure for representing X;

- A procedure for using a given AbDG for classification purposes, which can be approached as a method for determining how a subgraph of the AbDG, representing the new unclassified instance conforms to the AbDG structure;
- A GA-based procedure which aims at searching for a suitable set of edges of an AbDG, so to better represent a given input data set.

The chapter also briefly introduces a few other issues related to an AbDG structure, particularly its contribution for supporting imputation processes, as described in [7, 8]. Although this chapter has no focus on experiments and analyses of their results, many such results and analyses can be found in a number of works cited in this chapter. It is a fact though that most of the experimental results published can be considered evidence of the suitability of the AbDG structure for summarizing and representing data, as well as the great potential of the proposed algorithms involved in the AbDG construction and use, mainly due to their robustness, low computational costs associated with training and memory occupation.

References

1. Agarwal, S., Branson, K., Belongie, S.: Hihger order learning with graphs. In: Proceedings of the 23rd International Conference on Machine Learning (ICML 2006), pp. 17–24. ACM, New York (2006)
2. Aupetit, M.: Learning topology with the generative Gaussian graph and the em algorithm (2006). In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems 18, pp. 83–90. MIT Press, Cambridge (2006)
3. Belkin, M., Niyogi, P., Sindhwani, V.: Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J. Mach. Learn. Res. **1**, 1–48 (2006)
4. Bertini Jr., J.R., Nicoletti, M.C.: A genetic algorithm for improving the induction of attribute-based decision graph classifiers. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 4104–4110. IEEE Press, New York (2016)
5. Bertini Jr., J.R., Nicoletti, M.C., Zhao, L.: Attribute-based decision graphs for multiclass data classification. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 1779–1785 (2013)
6. Bertini Jr., J.R., Nicoletti, M.C., Zhao, L.: Ensemble of complete p-partite graph classifiers for non-stationary environments. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 1802–1809 (2013)
7. Bertini Jr., J.R., Nicoletti, M.C., Zhao, L.: Imputation of missing data supported by complete p-partite attribute-based decision graph. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN), pp. 1100–1106 (2014)
8. Bertini Jr., J.R., Nicoletti, M.C., Zhao, L.: An embedded imputation method via attribute-based decision graphs. Expert Syst. Appl. **57**, 159–177 (2016)
9. Bertini Jr., J.R., Nicoletti, M.C., Zhao, L.: Attribute-based decision graphs: a framework for multiclass data classification. Neural Netw. **85**, 69–84 (2017)
10. Bertini Jr., J.R., Zhao, L., Motta, R., Lopes, A.A.: A nonparametric classification method based on k-associated graphs. Inf. Sci. **181**, 5435–5456 (2011)
11. Bi, W., Kwok, J.: Multi-label classification on tree and dag-structured hierarchies. In: Proceedings of the 28th International Conference on Machine Learning (ICML 2011), pp. 17–24. ACM, New York (2011)
12. Chapelle, O., Zien, A., Schölkopf, B. (eds.): Semi-supervised Learning, 1st edn. MIT Press, Cambridge (2006)

13. Cheh, Z., Zhao, H.: Pattern recognition with weighted complex networks. Phys. Rev. E **78**(056107), 1–6 (2008)
14. Chen, J., Fang, H.R., Saad, Y.: Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. J. Logic Program. **10**, 1989–2012 (2009)
15. Cook, D., Holder, L.: Graph-based data mining. IEEE Intell. Syst. **15**(2), 32–41 (2000)
16. Culp, M., Michailidis, G.: Graph-based semisupervised learning. IEEE Trans. Pattern Anal. Mach. Intell. **30**(1), 174–179 (2008)
17. Eppstein, D., Paterson, M.S., Yao, F.: On nearest-neighbor graphs. Discret. Comput. Geom. **17**(3), 263–282 (1997)
18. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous valued attributes for classification learning. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence, vol. 2, pp. 1022–1027. Morgan Kaufmann Publishers, San Francisco (1993)
19. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**, 75–174 (2010)
20. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd edn. Springer, Canada (2009)
21. Hein, M., Audibert, J.Y., von Luxburg, U.: Discretization: an enabling technique. Data Min. Knowl. Disc. **6**, 393–423 (2002)
22. Hein, M., Audibert, J.Y., von Luxburg, U.: Graph Laplacians and their convergence on random neighborhood graphs. J. Mach. Learn. Res. **8**, 1325–1368 (2007)
23. Holder, L., Cook, D.: Graph-based relational learning: current and future directions. ACM SIGKDD Explor. **5**(1), 90–93 (2003)
24. Holder, L., Cook, D., Coble, J., Mukherjee, M.: Graph-based relational learning with application to security. Fundamenta Informaticae **66**, 1–19 (2005)
25. Jensen, D., Neville, J., Gallagher, B.: Why collective inference improves relational classification? In: Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD'04), pp. 593–598. ACM (2004)
26. Lichman, M.: UCI machine learning repository (2013). http://archive.ics.uci.edu/ml
27. Macskassy, S., Provost, F.: A simple relational classifier. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD), Workshop on Multi-Relational Data Mining, pp. 64–76 (2003)
28. Malliaros, F., Vazirgiannis, M.: Clustering and community detection in directed networks: a survey. Phys. Rep. **533**, 95–142 (2013)
29. Muggleton, S.: Inductive logic programming: issues, results and the challenge of learning language in logic. Artif. Intell. **114**, 283–296 (1999)
30. Muggleton, S., Raedt, L.D.: Inductive logic programming: theory and methods. J. Logic Progr. **19–20**, 629–679 (1994)
31. Newman, M.: The structure and function of complex networks. SIAM Rev. **45**(2), 167–256 (2003)
32. Quinlan, J.R.: C4.5 Programs for Machine Learning, 1st edn. Morgan Kaufmann Publishers, San Francisco (1993)
33. Schaeffer, S.: Graph clustering. Comput. Sci. Rev. **1**, 27–34 (2007)
34. Specht, D.F.: Probabilistic neural networks. Neural Netw. **3**, 109–118 (1990)
35. Vapnik, V.: The nature of statistical learning theory. Springer, Berlin (1999)
36. Wenga, L., Dornaikab, F., Jina, Z.: Graph construction based on data self-representativeness and Laplacian smoothness. Neurocomputing **207**, 476–487 (2016)
37. Xie, J., Kelley, S., Boleslaw, K.S.: Overlapping community detection in networks: the state-of-the-art and comparative study. ACM Comput. Surv. **45**(43), 1–35 (2013)
38. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33rd International Conference on Machine Learning (2016)
39. Yoshida, K., Motoda, H., Indurkhya, N.: Graph-based induction as a unified learning framework. J. Appl. Intell. **4**, 297–328 (1994)
40. Zhu, X.: Semi-supervised learning literature survey. Technical report 1530, Computer-Science, University of Wisconsin-Madison (2008)
41. Zhu, X., Lafferty, J., Ghahramani, Z.: Semi-supervised learning: from Gaussian fields to Gaussian processes. Technical report CMU-CS-03-175, Carnegie Mellon University (2003)