Sriwanna *et al. Hum. Cent. Comput. Inf. Sci.* (2017) 7:21
DOI 10.1186/s13673-017-0103-8

Human-centric Computing and Information Sciences

RESEARCH Open Access

Graph clustering-based discretization of splitting and merging methods (GraphS and GraphM)

Kittakorn Sriwanna*, Tossapon Boongoen and Natthakan Iam-On

*Correspondence:
kittakorn.sri@gmail.com
School of Information Technology, Mae Fah Luang University, Phahon Yothin Road, Muang, Chiang Rai 57100, Thailand

Abstract

Discretization plays a major role as a data preprocessing technique used in machine learning and data mining. Recent studies have focused on multivariate discretization that considers relations among attributes. The general goal of this method is to obtain the discrete data, which preserves most of the semantics exhibited by original continuous data. However, many techniques generate the final discrete data that may be less useful with natural groups of data not being maintained. This paper presents a novel graph clustering-based discretization algorithm that encodes different similarity measures into a graph representation of the examined data. The intuition allows more refined data-wise relations to be obtained and used with the effective graph clustering technique based on *normalized association* to discover nature graphs accurately. The goodness of this approach is empirically demonstrated over 30 standard datasets and 20 imbalanced datasets, compared with 11 well-known discretization algorithms using 4 classifiers. The results suggest the new approach is able to preserve the natural groups and usually achieve the efficiency in terms of classifier performance, and the desired number of intervals than the comparative methods.

Keywords: Multivariate discretization, Graph clustering, Normalized cuts, Normalized association, Data mining

Background

Discretization is a data reduction preprocessing technique in data mining. It transforms a numeric or continuous attribute to a nominal or categorical attribute by replacing the raw values of a continuous attribute with non-overlapping interval labels (e.g., 0–5, 6–10, etc.). Different data mining algorithms are designed to handle different data types. Some are designed to handle only either numerical data or nominal data, while some can cope with both. Because real datasets are always a combination of numeric and nominal vales, for an algorithm that only takes nominal data, numerical attributes need to be discretized into nominal attributes before the learning algorithm. After discretization, the subsequence mining process may be more efficient as the data is reduced and simplified resulting in more noticeable patterns [1–3]. Moreover, discretization is also expected to improve the predictive accuracy for classification [4] and Label Ranking [5].

Two main goals of discretization are to find the best intervals or the best cut points[1] and to find the finite number of intervals or the number of cut points that are better adapted to the learning. Therefore, modelling a discretization requires the development of the following two subjects. First, *discretization criterion* is the criterion made for choosing the best cut points in order to split a set of distinct numeric value into intervals (Splitting, Top–Down methods) or merge a pair of adjacent intervals (Merging, Bottom–Up methods). Second, *stopping criterion* is a criterion for stopping the discretization process in order to yield the finite number of intervals.

The process of discretization typically consists of 3 main steps that are sorting attribute values, finding the cut-point model or discretization scheme of the attributes by iterative splitting or merging, and finally, assigning each value in the attribute with a discrete label corresponding to the interval it falls into. Hence, the number of intervals obtained for attributes under question may be different depending on the number of possible cut points, the relation with the target class, for instance.

Discretization techniques can be classified in several different ways, such as supervised versus unsupervised, univariate versus multivariate, splitting versus merging, direct versus incremental, and more [6–9]. Supervised methods consider the class information, whereas unsupervised ones do not consider the class information. Splitting algorithms start from one interval and recursively select the best cut point to split the instances into two intervals, while merging methods begin with the set of single value intervals and iteratively merge adjacent intervals. The univariate category discretizes each attribute independently without considering its relationship between other attributes. However, multivariate methods also consider other attributes to determine the best cut points. Direct techniques require inputting the number of intervals supplied by the user. Example of this type of methods are equal-width and equal-frequency discretization algorithms [10]. The number of intervals is equal for all attributes in these algorithms. In contrast, incremental methods do not require the number of intervals, but they require the stopping criterion to stop the discretization process in order to yield the best number of intervals of each attribute.

Most discretization algorithms proposed in the past were univariate methods [3]. ChiMerge [11], Chi2 [12], Modified Chi2 [13], and area-based [14] are examples of univariate, supervised, incremental and merging methods. They use statistic-based algorithms to determine the similarity of adjacent intervals. They divide the instances into intervals and then merge adjacent intervals based on χ^2 distribution. CAIM [15], ur-CAIM [16], and CADD [17] are univariate, supervised, incremental and splitting methods. These algorithms use class-attribute interdependence information as the discretization criterion for the optimal discretization with a greedy approach, by finding locally maximum values of the criterion. Ent-MDLP [18], D-2 [19], and FEBFP [20], which are univariate, supervised, incremental and splitting methods, recursively select the best cut points to partition the instances based on the class information entropy [10]. PKID and FFD [21] are univariate and unsupervised methods that maintain discretization bias and variance by tuning the interval frequency and the interval numbers, especially used with Naive-Bayes classifier.

[1] A cut point is the midpoints of the adjacent distinct values after sorting the attribute.

The fact that univariate methods discretize only a single attribute at a time and do not consider interactions among attribute, may lead to important information loss [3] and not getting a global optimal result [22]. ICA [22], Cluster-Ent-MDLP [23], HDD [3], and Cluster-RS-Disc [24] are examples of multivariate discretization methods. ICA, tries to get a global optimal result by transforming the original attributes to new attribute space that considers other attributes. Then, it discretizes the new attribute space using the univariate discretization method (Ent-MDLP). Cluster-Ent-MDLP, similar to ICA, finds the pseudo-class by clustering the original data via *k*-means [25] and SNN [26] clustering algorithms and then uses the target class and pseudo-class to discretize using the univariate discretization method (Ent-MDLP). It finds the best cut point by averaging both entropies that considers the target class and pseudo-class. HDD extends the univariate discretization method (CAIM) by improving the stopping criterion and taking into account information specific to other attributes. Cluster-RS-Disc attempts to obtain the natural intervals of attribute values by first partitioning using DBSCAN [27] clustering algorithm. After that, it discretizes an attribute based on the rough set theory.

Although several multivariate discretization algorithms overcome the drawback of univariate discretization methods by considering interactions among attributes, there are some weaknesses. Some of them have made use of cluster labels as a part of discretization criterion. As a matter of fact, different clustering algorithms or even the same algorithm with multiple trials may produce different cluster labels. Therefore, it is extremely difficult for a user to decide the proper algorithm and parameters [28–30]. Despite the reported improvement, some multivariate discretization algorithms such as EMD [31] do not concentrate the natural group of data. In fact, EMD is an evolutionary discretization algorithm that defines the fitness function based only on high predictive accuracy and lower number of intervals. Hence, the identified cut points may damage the natural group of data.

In order to solve the aforementioned problems, this study presents a novel graph clustering based algorithm that allows encoding of different similarity measures into a graph representation of the examined data. Instead of using cluster labels, the proposed method uses the similarity between data pair, which is the weighted combination between distance and target-class agreement. Each pair of data is formed into graph, which is then partitioned in order to find the appropriate set of cut-points. The insightful observations and the benefit of graph clustering are present as follows.

In the clustering process, the instances are partitioned into clusters based on their similarity. Instances in the same cluster are similar to one another and dissimilar to instances in other clusters [1, 32]. Clustering is a method for recognizing and discovering natural groups of similar elements in a dataset [33, 34]. Recently, clustering with graphs has been widely studied and become very popular. The method is to treat the entire clustering problem as a graph problem. The graph vertices are grouped into clusters based on the edge structure and property [35]. Graph clustering algorithms are suitable for data that does not comply with a Gaussian or a spherical distribution [36]. They can be used to detect clusters of any size and shape. Moreover, graph clustering is a very useful technique for detecting densely connected groups [37]. The goal of graph clustering is the separation of sparsely connected dense subgraphs from one another based on various criteria such as vertex connectivity or neighborhood similarity [38].

The main graph clustering formulations are based on graph cut and partitioning problems [39, 40]. According to the research of Foggia et al. [36], the performance of five graph clustering algorithms: fuzzy *c*-means MST (minimum spanning tree) clustering [41], Markov clustering [34, 42], iterative conductance cutting algorithm [43], geometric MST clustering [44], and normalized cut clustering [40] are evaluated and compared. Based on this empirical study, it is confirmed that the normalized cut clustering provides good performance and appears to be robust across application domains. The normalized cut criterion avoids any unnatural bias for partitioning out small sets of points [40]. It is used in many works such as image segmentation and spectral clustering [45, 46].

With these insightful observations, the paper introduces new discretization algorithms, which exploit graph clustering and the normalized cut criterion. Because the minimum normalized cut criterion is equivalent to the maximum normalized association criterion (see "Measures with clusters" for details), to design the *stopping criterion*, this normalize association criterion is chosen in this study. In particular, two new graph clustering-based techniques, which are splitting method (GraphS) and merging method (GraphM), are proposed. It is worthwhile to highlight several aspects of the proposed models here:

- The proposed discretization algorithms are incremental and multivariate methods. They find the number of cut points automatically and preserve the natural group of data by considering the correlation dependence among the attributes.
- The algorithm encodes information of data under examination as a graph, where the weight of each edge is evaluated by both natural distance and class similarity. This is a novel with the capability to address both data-wise relation and class-specific in the same decision making process.
- The normalize cut criterion prevents an unnatural bias that partitions out a small set of points. It helps to avoid acquiring too-small-size intervals with only a few members, thus demoting the over-fit problem.

The rest of this paper is organized as follows. "Graph clustering and partitioning problems" provides the basis of graph clustering as to set the scene for proposed work and following discussions. In "A novel graph clustering-based approach", the new algorithms are presented with related design issues being explained. The performance evaluation is included in "Performance evaluation", based on the experiments with a rich collection of benchmark data. The paper is concluded in "Conclusion", with possible future work.

Graph clustering and partitioning problems

In this paper, the undirected weighted graph with no self-loops and global graph clustering are focused. Global graph clustering assigns all of the vertices to a cluster, whereas local graph clustering only assigns a certain subset of vertices [35]. In brief, the reviews in this section are on these two types of undirected weighed and global graph clustering.

Throughout this paper, let $G = (V, E, \omega)$ be an undirected graph, $V = \{v_1, \ldots, v_n\}$ is a set of vertices and the number of vertices $n = |V|$, where each vertex v_i represents a data point x_i, and $E = \{(u, v) \mid u, v \in V\}$ is a set of edges and the number of edges $m = |E|$,

where $\omega(u, v)$ is the positive weighting of the edges. Let the set of partitioning or clustering of G be $\pi_k = \{C_1, \ldots, C_k\}$, where C_i is a cluster in k sub-clusters.

Clustering measures

Measure with vertices

This type of measures is based on similarities or distances of the object pairs. They are used to apply to many areas such as pattern recognition, information retrieval, and clustering [47]. The common distance measures that are used in clustering are Euclidean and Manhattan distance [48].

Euclidean distance It simply is a geometric distance in a multidimensional space. It is a multivariate extension of the *Pythagorean distance* between two points. The formula for this distance between data points $u(u_1, \ldots, u_d)$ and $v(v_1, \ldots, v_d.)$ with d dimensions is shown in the Eq. 1.

$$dist_E(u, v) = \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2} \tag{1}$$

Manhattan distance It is also called the City-block distance, Rectilinear distance, and Taxicab norm. The result of this distance is the distance similar to Euclidean distance. However, the effect of outliers is dampened because they are not taking the square root [49]. The distance is computed as the Eq. 2.

$$dist_M(u, v) = \sum_{i=1}^{d} |u_i - v_i| \tag{2}$$

Measures with clusters

Graph clustering is dividing a graph into groups (cluster, subgraph) that vertices highly connect in the same group [50]. To compare the quality of a given cluster C, the *cluster fitness measure* (*quality function*) or *indices for graph clustering* [33, 44, 51] are used [35, 52]. This study categorizes the clustering indices into two groups: density measures, and cut-based measures.

Density measures In the wrok of Görke et al. [53], many density measures such as intracluster density, intercluster density, intercluster conductance, and modularity are examined. These measures are used for unweighed graphs. They compute with number of vertices and edges in a cluster and another clusters. Brandes et al. [44] proposed the *coverage of the graph clustering* based on the number of *intra-cluster edges* and *inter-cluster edges* as shown in Eq. 3, where $m(C_i, C_i)$ and $m(C_i, \bar{C}_i)$ are the numbers of intra-edges and inter-edges in the cluster C_i respectively. Those edges are shown in Fig. 1. A large value of *coverage* indicates the better quality.

$$coverage(C_i) = \frac{m(C_i, C_i)}{m(C_i, C_i) + m(C_i, \bar{C}_i)} \tag{3}$$

Cut-based measures In undirected weighted graphs, graph-cut techniques such as ratio cut [54], min–max cut [55], and normalized cut [40] are mostly used to solve graph

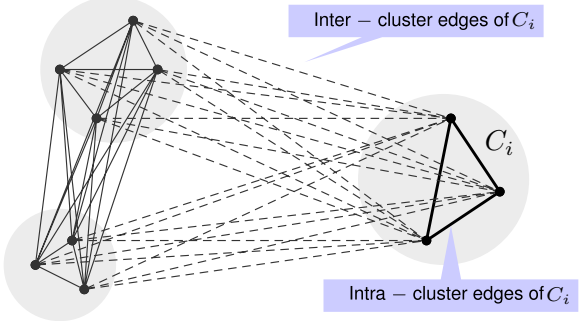

Fig. 1 Inter and intra cluster edges. The example of inter-cluster and intra-cluster edges of C_i, the *dashed lines* are the inter-cluster edges and the *solid lines* in cluster C_i are the intra-cluster edges

partitioning problem, which are ones of the main keys to graph clustering algorithms. Those measures are formulated in Eqs. 4, 5, and 6 respectively, where $\omega(C_i, C_i)$ and $\omega(C_i, \bar{C}_i)$ are the sum of weighted edges of inter-cluster edges and intra-cluster edges of C_i, respectively; $n(C_i)$ is the number of vertices in the cluster C_i.

$$Ratio\ Cut(\pi_k) = \sum_{i=1}^{k} \frac{\omega(C_i, \bar{C}_i)}{n(C_i)} \tag{4}$$

$$Min\ Max\ Cut(\pi_k) = \sum_{i=1}^{k} \frac{\omega(C_i, \bar{C}_i)}{\omega(C_i, C_i)} \tag{5}$$

$$NCut(\pi_k) = \sum_{i=1}^{k} \frac{\omega(C_i, \bar{C}_i)}{\omega(C_i, C_i) + \omega(C_i, \bar{C}_i)} \tag{6}$$

The smallest value of these criteria indicates a better quality of partitioning. Normalized cut avoids unnaturally partitioning out any too small set of vertices by taking the fraction out of the total weighted edges connected to all nodes in the graph. On the other hand, instead of considering the minimum cut among the clusters, partitioning can be done based on the maximum connection, called *normalized association*, as defined in Eq. 7. In addition, *normalized association* can define as Eq. 8.

$$NAsso(\pi_k) = \sum_{i=1}^{k} \frac{\omega(C_i, C_i)}{\omega(C_i, C_i) + \omega(C_i, \bar{C}_i)} \tag{7}$$

$$NAsso(\pi_k) = NAsso(C_1) + NAsso(C_2) + \cdots + NAsso(C_k) \tag{8}$$

provided that

$$NAsso(C_i) = \frac{\omega(C_i, C_i)}{\omega(C_i, C_i) + \omega(C_i, \bar{C}_i)} \tag{9}$$

The minimum *NCut* and the maximum *NAsso* are equivalent and naturally related as $NAsso(\pi_k) = k - NCut(\pi_k)$. In the authors' observation, the $coverage(C_i)$ is similar to $NAsso(C_i)$, which changes the number of edges to the sum of weighted edges as demonstrated in [33].

Global graph clustering

The problem of global graph clustering is a dividing or grouping each vertex in the graph into clusters of predefined size, such that vertices in the same cluster are highly related and less related with vertices in the other clusters. The main clustering problem is NP-hard; therefore, the algorithms selected are *approximation algorithm*, *heuristic algorithm*, or *greedy algorithm* [35] so that the computation time is reduced.

Spectral clustering

The main tool for spectral clustering is the *Laplacian matrices technique* [56]. The algorithm transforms the affinity matrix (similarity matrix) into Laplacian matrix of the graph and then finds the eigenvector and eigenvalue from the matrix. Each row of eigenvector represents a vertex (data point). The final stage is clustering or partitioning of those vertices by recursive *two-way* (bipartition) or direct *k*-way [39].

The unnormalized *k*-means algorithm [57] is an example of direct *k*-way clustering. It clusters the vertices using *k*-mean algorithm with the number of desired clusters. The two-way normalized spectral clustering (2NSC) and *k*-way normalized spectral clustering (KNSC) algorithms are proposed by [40] to solve the generalized eigenvalue system using *NCut* criterion. 2NSC is a recusive two-way clustering that the number of clusters is controlled directly by the maximum allowed *NCut* value, whereas KNSC is direct *k*-way clustering based on *k*-mean algorithm.

Markov chains and random walks

The Markov cluster algorithm (MCL) [34] finds the cluster structure in the form of graphs using the mathematical bootstrapping procedure. The main idea of MCL is to simulate the flow within a graph, promote the flow where the current is strong and demote the flow where the current is weak. If there is any natural group in the graph, the current across between groups will wither away. The algorithm also simulates random walks within a graph, starting from a vertex and randomly travelling to another vertex that connected with many steps. Travelling within the same cluster is more likely than across the other cluster.

A novel graph clustering-based approach

This section introduces a new graph clustering-based approach to discretization problems. Figure 2 summarizes the entire process of the proposed framework, which includes three main steps: (1) create the affinity matrix using the pairwise distances and class similarity, (2) create discretization schemes, and (3) discretize the datasets by applying the discretization schemes.

This approach is based on the intuition of representing data points as a graph, finding cut point models of numeric attributes based on graph clustering, and transforming numeric to nominal data. Given the Toy dataset as shown in Fig. 3a, the graphical

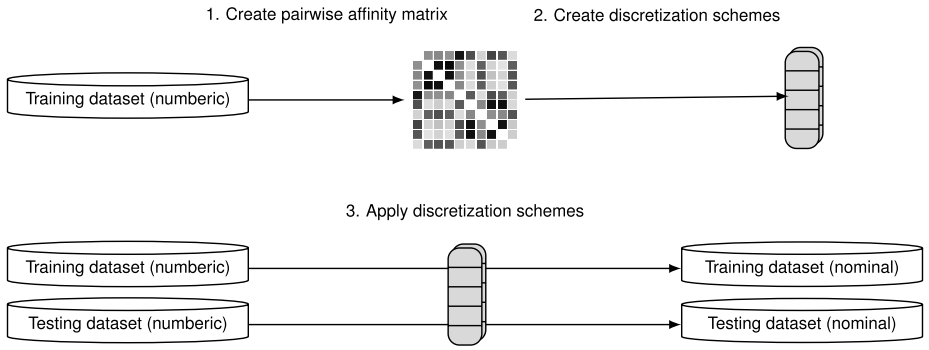

Fig. 2 Proposed framework. The framework of the graph clustering-based discretization. A graph clustering-based algorithm first creates the affinity matrix to represent a graph; then, generates a discretization scheme of each numeric attribute. Finally, the numeric attributes are transformed into the nominal attributes by applying the dicretization schemes

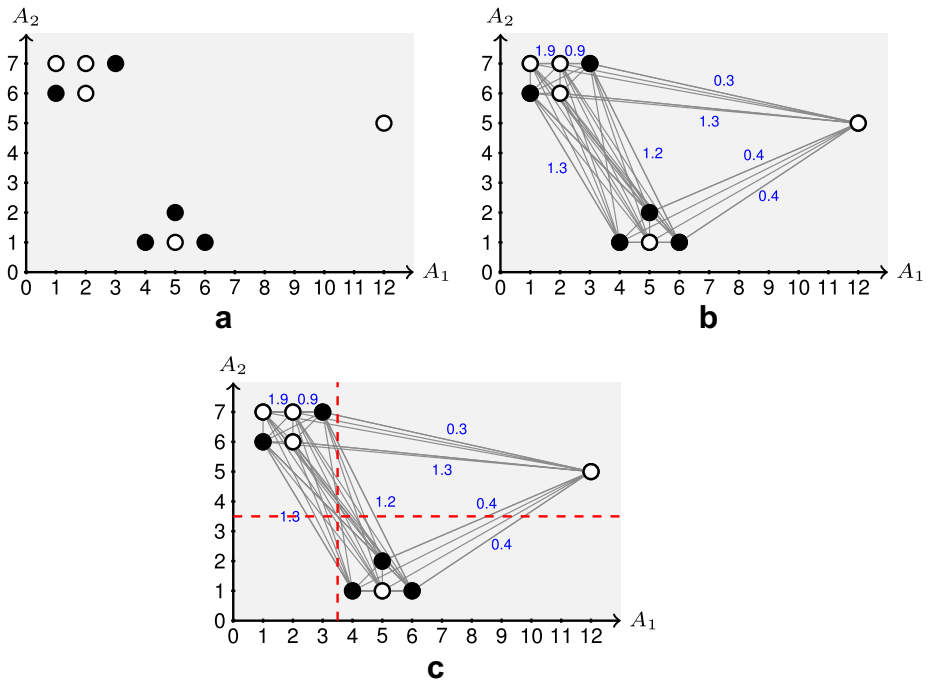

Fig. 3 Graph representation of data points. Example of **a** the scatter plot of the toy dataset, **b** the proposed graph representation of data points, and **c** the cut point selections that cut the minimum weighted edges. Note that, *white* and *black circlets* represent a different class

representation of this dataset is demonstrated in Fig. 3b. It assigns the weight of each edge between each pair of the data points based on similarity, which both target class and distance are considered. If the data points of the pair are close and in the same class, the edge will have a high score. This scoring is stored in the affinity matrix (see "Pairwise affinity matrix" for details). After that, graph clustering is used to find the best cut point that cuts the minimum weighted edge as shown in Fig. 3c, where vertical and horizontal dashed lines are the best cut points that cut minimum weighted edges of attributes A_1 and A_2, respectively (see "Graph clustering-based discretization algorithm" for details).

Pairwise affinity matrix

The pairwise affinity matrix (AF matrix) is an $n \times n$ matrix that contains the similarity scores of the pair of data points, where a high score means the pair has high similarity. In this study, the graph is represented by an AF matrix. The distance and class similarity values of the pair are considered for scoring in order to find the similarity. Before scoring, each value of the data points x_i in the numeric attribute A_j, val_{ij} is the rescaled (normalized) between 0 and 1 in order to give all attributes the same treatment as shown in Eq. 10, where val^*_{ij} is new rescale value of val_{ij}, min_{A_j} and max_{A_j} are the minimum and maximum values of the attribute A_j, respectively.

$$val^*_{ij} = \frac{val_{ij} - \min_{A_j}}{\max_{A_j} - \min_{A_j}} \tag{10}$$

In this study, the similarity measure, $sim(u, v)$, is proposed as Eq. 11. It contains 2 main measures. First, $simC(u, v)$ is a class-label measure of a pair of the data points u and v. Second, the distance measure, based on the Euclidean distance (see Eq. 1), adds a fraction of the number of attribute d in the square root in order to normalize the distance to be between 0 and 1 and subtracts the distances from 1 in order to change to the similarity (small distance values indicates high similarity). In addition, the distance of a pair in the nominal attribute is set to 1 if the pair has the same value and 0 otherwise.

$$sim(u, v) = simC(u, v) + \left(1 - \sqrt{\frac{\sum_{i=1}^{d} (u_i - v_i)^2}{d}}\right), \tag{11}$$

provided that

$$simC(u, v) = \begin{cases} 1 & \text{if } u \text{ and } v \text{ have the same class label,} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The proposed similarity measure considers both the distance and class labels for scoring. The measure weights those values equally by combining them altogether. Each element in the AF matrix contains the $sim(u, v)$ value of each data-point pair, and $AF \in [0, 2]^{n \times n}$.

Graph clustering-based discretization algorithm

Typically, the discretization algorithm discretizes attribute one by one. The process consists of three main processes. First, sorting an attribute A_i and finding all the possible cut points. Second, finding the cut point model of the attribute by iteratively finding the model one by one until discretization is complete on all of the numeric attribute. Finally, transforming the numeric attribute to the nominal attribute.

In order to show the difference of the data view of the proposed algorithm, an example of the Toy dataset shown in Fig. 3 is again considered. The dataset has 2 numeric attributes, A_1 and A_2. In order to discretize the attribute A_1, the data views of discretization algorithms are different depending on discretization methods. Figure 4 shows the four different data points views.

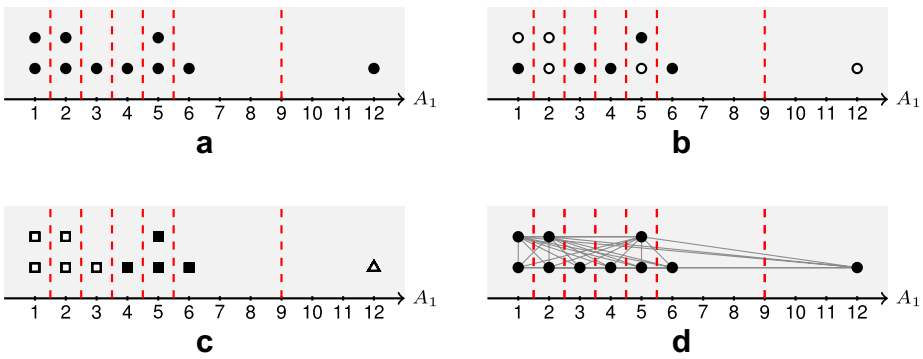

Fig. 4 Four different data point views. The data point views of attribute A_1 of Toy dataset, where the *vertical dashed lines* are the possible cut points, **a** the view under unsupervised method, **b** the view under the class labels, **c** the view under the cluster labels, and **d** the view under the graph representation method. Note that, *white* and *black circlets* represent a different class, *square* and *triangle* represent a different cluster, the *edge* between the *circlets* represents the similarity value

- After sorting the attribute A_1, discretization algorithm that is unsupervised and univariate method only considers the cut points as shown in Fig. 4a. The algorithm selects the cut point that all intervals have the similar number of data points or similar length.
- For discretization algorithm that is the supervised and univariate method, after sorting A_1, the algorithm only considers the class labels as shown in Fig. 4b. The algorithm discretized based on the purity class. The data points with the same class label are grouped. However, this method does not consider other attributes; therefore, it tends to lose the natural group.
- The Cluster-Ent-MDLP [23] is an example of supervised and multivariate method. This type of algorithms includes other attributes into consideration. The data are clustered and labelled. Then, the algorithm discretizes by considering class labels (see Fig. 4b for details) and cluster labels (see Fig. 4c for details) together.
- In the proposed algorithm, the data point view is based on the graph as shown in Fig. 4d. The vertices represent the data points and the edges's weights represent the scores of similarity of the vertex pairs (see "Pairwise affinity matrix"). As both class and cluster labels are considered, this data view preserves much more information than the other views.

As the benefit of the data view under graph representation, this paper proposed 2 discretization algorithms using this data view: spiting and merging methods. The discretization criterion and stopping criterion of these algorithms are based on the *NAsso* criterion (see "Measures with clusters" for details). The algorithms attempt to find the best set of cut points so that the data points in each cluster after partitioning are mostly similar judging from the weighed edges.

The proposed splitting discretization algorithm

In the basis of splitting methods (top–down methods), all data points are grouped into clusters/intervals. The algorithms add a new cut point to bi-partition the clusters iteratively. The process stops partition until the number of cut points is satisfied. Similarly, the proposed splitting discretization bi-partitions the clusters as a tree structure as shown in Fig. 5.

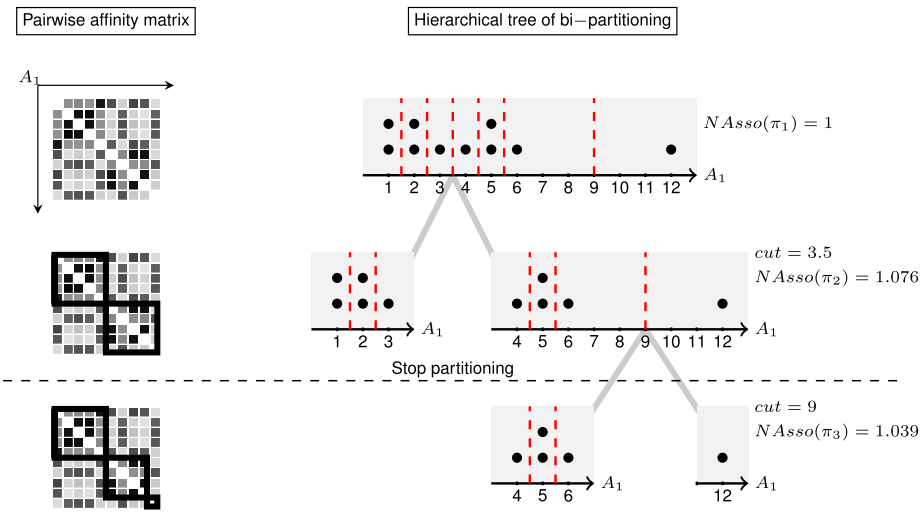

Fig. 5 GraphS. The affinity matrix and hierarchical tree views of the proposed splitting discretization algorithm

In this proposed top–down algorithm, the *discretization criterion* of *NAsso* is used to consider the best new cut point. This criterion based on the sum of weights of intra-cluster and inter-cluster. To easily find these values, the order of the pairwise affinity matrix is used to calculate. Figure 5 is an example of discretization of attribute A_1 of the Toy dataset. The procedures are as follows:

- Firstly, reorder the pairwise affinity matrix of A_1 in ascending as shown in the top matrix in Fig. 5. The matrix is 10×10 elements. Each element contains the pair-similarity value, which in this example is represented by color, where a dark color indicates high similarity and a light color indicates low similarity.
- Then, calculate the *NAsso* values of all possible cut points (vertical dashed-lines). The example of calculating *NAsso* value of the cut point 3.5 is shown in the middle pairwise affinity matrix. The two square boxes in the matrix are the intra-cluster weights of 2 partitions. The row outside the box is the inter-cluster weight of the boxed. The algorithm calculates *NAsso* values of all 6 cut points and select a cut point at 3.5 according to its highest *NAsso* values. Then, it bi-partitions a graph into 2 clusters.
- Finally, iteratively find the new best cut point until the *stopping criterion* is satisfied. In the figure, the second cut point is 9 with the highest *NAsso* value of 1.039. Because the value does not improve the *NAsso* value at the previous cut point ($cut = 3.5$, $NAsso = 1.076$), the stopping criterion is stratified and stops at the previous step (see next paragraph for the details). Hence, attribute A_1 has 2 intervals with the cut point at 3.5.

The proposed stopping criterion is based on the significant improvement of *NAsso* values, as the higher *NAsso* denotes the higher similarity or community of data points in each cluster. If *NAssso* value drops or does not significantly improve after partitioning, it should stop discretization. The proposed stopping criterion is shown in Eq. 13, where $NAsso(\pi_i)$ and $NAsso(\pi_{i-1})$ are *NAsso* values of the graph partitioning results with the

set of i and $i-1$ clusters, respectively, and β is the significant improvement percentage. If the stopping criterion is true, the discretizaton will stop and the result is given as $\pi_* = \pi_{i-1}$.

$$NAsso(\pi_i) < NAsso(\pi_{i-1}) \times \beta \tag{13}$$

The examples of the trend of *NAsso* value (average of all numeric attributes) of 4 real datasets obtained from the benchmark UCI repository [58] of blood, glass, iris, and yeast are illustrated in Fig. 6. It shows that, for all 4 datasets, when increasing the number of clusters (number of intervals), the *NAsso* values drastically increase at the beginning, then slowly increase and finally slowly decrease. To prevent over partitioning at the slowly increasing of *NAsso* value, the new *NAsso* value, $NAsso(\pi_i)$, should be significantly better than the previous *NAsso* value, $NAsso(\pi_{i-1})$. This study sets the new *NAsso* value should have 1% higher in significance than the previous value i.e., $\beta = 1.01$ (see "Parameter analysis" for details).

The details of the proposed splitting graph clustering-based discretization algorithm is shown in Fig. 7.

The proposed merging discretization algorithm

In contrast to splitting methods, the basic merging methods (bottom–up methods) start from many initial clusters. Each initial cluster contains data points that have the same attribute value. The algorithm iteratively merges the adjacent pair of clusters that the data points of the pair are most similar until the stopping criterion is met.

The proposed merging discretization algorithm includes 2 main steps. The first step is creating initial clusters, the data points in the clusters have the same attribute value. After that, the adjacent initial clusters that are given highest *NAsso* value after merging are selected to merge. The algorithm iterates merging the adjacent clusters, one by one until all of the clusters is grouped into a single cluster (1 interval), its forming is like a hierarchical tree as shown in Fig. 8. The next step is finding the best set of clusters (π_*) by examining top-down. This step starts from two intervals. It selects three intervals or higher intervals if significantly improve the *NAsso* value.

For demonstrating an example of this process, the Toy dataset is used again as shown in Fig. 8. In the first step, the initial clusters are created by grouping the data points of attribute A_1 that have the same attribute values and reordered the values in ascending. The initial clusters have 7 clusters, $\pi_7 = \{C_1, C_2, \ldots, C_7\}$. After that, two adjacent

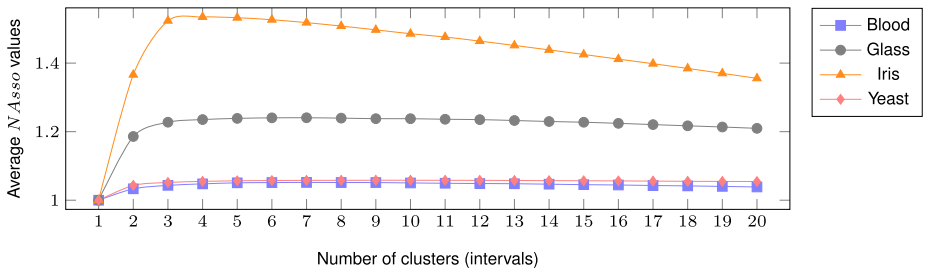

Fig. 6 Trend of NAsso. The trend of the average *NAsso* values of 4 datasets: blood, glass, iris, and yeast, as the number of clusters increases

input : training dataset $(train)$
output: the discretization scheme of each numeric attribute (A) of $train$
Create affinity matrix (AF) of $train$
for each A_a **do**
  Reorder AF by A_a values in ascending order and find all possible cut points (Cut)
  Group all data points into a cluster $\pi_1 = \{C_1\}$
  Calculate and save $NAsso(C_1)$ using AF
  repeat
    for each C_j in π_k **do**
      if $cut(C_j) = null$ **then** /* $cut(C_j)$ is the best cut point of cluster C_j */
        for each cut_i in C_j **do** /* $cut_i \in Cut$ that is the possible cut point in C_j */
          $Nasso(C_{ij}) = NAsso(C^L_{ij}) + NAsso(C^R_{ij})$ /* $Nasso(C_{ij})$ is the $NAsso$ value of C_j that bi-partition using cut_i, where C^L_{ij} and C^R_{ij} are left and right sub-clusters of C_j after partition, respectively */
          if $NAsso(C_{ij})$ is highest among previous the $NAsso$ value of other cut points **then**
            $cut_m = cut_i$
            $NAsso(C_{mj}) = NAsso(C_{ij})$
            $NAsso(C^L_{mj}) = NAsso(C^L_{ij})$
            $NAsso(C^R_{mj}) = NAsso(C^R_{ij})$
        $cut(C_j) = cut_m$
        $NAsso(C^L_j) = NAsso(C^L_{mj})$
        $NAsso(C^R_j) = NAsso(C^R_{mj})$
        $I(C_j) = NAsso(C_{mj}) - NAsso(C_j)$ /* $I(C_j)$ is the increase in $NAsso$ of C_j after partition using the best cut point cut_m */
    $I(C_b) = \max_{i \in 1,\ldots,k} \varphi(I(C_i))$ /* $I(C_b)$ is increasing $NAsso$ value of C_b that highest in π_k */
    Bi-partition C_b using $cut(C_b)$ and set the $NAsso$ value of two sub children as $NAsso(C^L_b)$ and $NAsso(C^R_b)$
    $NAsso(\pi_{k+1}) = \sum_{\forall i \in 1,\ldots,k+1} NAsso(C_i)$ /* note that now there is one cluster is added */
    if $k \geq 2$ and $NAsso(\pi_{k+1}) < NAsso(\pi_k) \times \beta$ **then**
      $\pi_* = \pi_k$
      break
    Save π_{k+1}
  until no more cut point to partition;
  Find the discretization scheme of π_*

Fig. 7 GraphS algorithm. Graph clustering-based discretization of Splitting method algorithm

clusters that have highest *NAsso* values are merged until all clusters are grouped into one cluster (π_1). In Fig. 8, the adjacent clusters of the cut point at 1.5, two initial clusters that contain attribute values of 1 and 2 possess the highest *NAsso* value, therefore, this pair is merged first. The adjacent clusters (intervals) are merged further until only one cluster whose *NAsso* value is 1 remained. In the second step, the best set of clusters (π_*) is searched by evaluating the *NAsso* values top–down. This bottom–up algorithm uses the *stopping criterion* as in the top–down algorithm (see Eq. 13). As one can see that, the set of 3 clusters (π_3) separated by the cut point of 9, the $NAsso(\pi_3)$ value is 1.039 that less than 1.076 of the $NAsso(\pi_2)$ value in the set of 2 clusters, hence $\pi_* = \pi_2$.

The detail of the proposed merging graph clustering-based discretization algorithm is shown in Fig. 9.

Numeric to nominal transformation

In Fig. 7 (GraphS algorithm) and Fig. 9 (GraphM algorithm), each cluster in π_* is equal to the interval, which contains the data point values. The intervals are transformed to discretization scheme as Eq. 14, where D_A is a discretization scheme of attribute A, and cut_i is the result of cut point selection of attribute A.

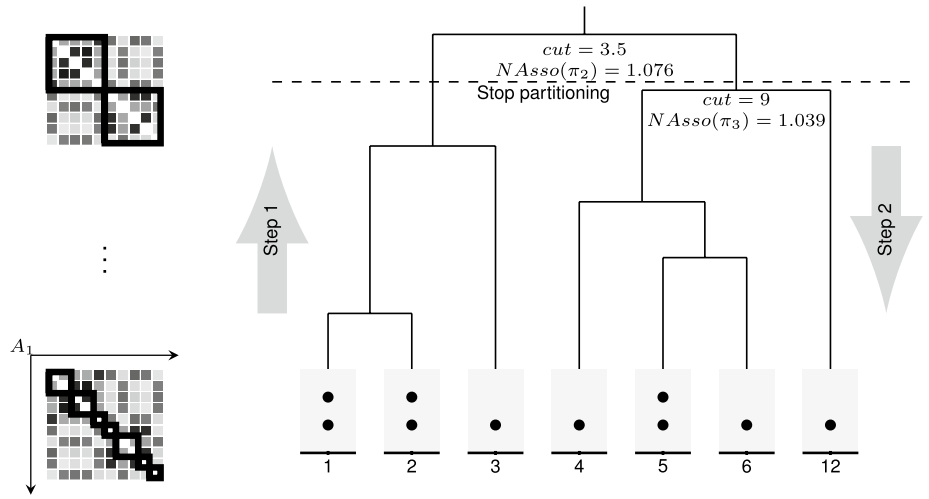

Fig. 8 GraphM. The affinity matrix and hierarchical tree views of the proposed merging discretization algorithm

input : training dataset $(train)$
output: the discretization scheme of each numeric attribute (A) of $train$

Create affinity matrix (AF) of $train$
for each A_a **do**
  Reorder AF by A_a values in ascending
  Create initial clusters $\pi_k = \{C_1, C_2, \ldots, C_k\}$ that group the same attribute value
  for $C_{j=1}$ to $C_{j=k}$ **do**
    Calculate and save $NAsso(C_j)$ using AF
  for $C_{j=1}$ to $C_{j=k-1}$ **do**
    Calculate and save $NAsso(C_j, C_{j+1})$ /* $NAsso(C_j, C_{j+1})$ is the $NAsso$ value of the group of C_j and C_{j+1} */
    $I(C_{j,j+1}) = NAsso(C_j, C_{j+1}) - (NAsso(C_j) + NAsso(C_{j+1}))$ /* $I(C_{j,j+1})$ is increasing $NAsso$ value after merge C_j and C_{j+1} together */
  repeat
    $I(C_{b,b+1}) = \max_{i \in 1,\ldots,k-1} \varphi(I(C_{i,i+1}))$ /* $I(C_{b,b+1})$ is the highest increasing $NAsso$ value after merge C_b and C_{b+1} together */
    $C_m = C_b$ merge C_{b+1} /* note that now the number of clusters decreases one, C_m is a cluster that merge C_b and C_{b+1} together */
    $NAsso(C_m) = NAsso((C_b, C_{b+1}))$
    Calculate and save $NAsso(C_m, C_{m+1})$ and $NAsso(C_{m-1}, C_m)$
    Calculate and save $I(C_{m,m+1})$ and $I(C_{m-1,m})$
    Save π_k /* each step of merging is saved for examining top-down in order to find the best partition */
  until all clusters are merged into one cluster;
  for $\pi_{j=3}$ to $\pi_{j=k}$ **do** /* note that, because the minimum interval is 2, $j = 3$ */
    if $NAsso(\pi_j) < NAsso(\pi_{j-1}) \times \beta$ **then**
      $\pi_* = \pi_{j-1}$
      break
  Find the discretization scheme of π_*

Fig. 9 GraphM algorithm. Graph clustering-based discretization of Merging method algorithm

$$D_A = \{(-\infty, cut_1], (cut_1, cut_2], \ldots, (cut_k, +\infty)\} \tag{14}$$

In the example of attribute A_1 of the Toy dataset, the cut point list has only 1 cut point at 3.5. The discretization scheme of the attribute is $\{(-\infty, 3.5], (3.5, +\infty]\}$. In this study, the numeric values of A_1 that are equal or less than 3.5 would be changed to "(inf_3.5]" and

the values that are higher than 3.5 are denoted as "(3.5_inf]". Each value of the numeric attribute is transformed to the interval name by checking the condition in the discretization scheme.

Performance evaluation

This section presents the performance evaluation of the graph clustering-based approaches that are splitting approach (GraphS) and merging approach (GraphM). In order to show the goodness of the proposed approaches that can handle with real world applications, this study investigated 30 real standard datasets and 20 imbalanced datasets. This study evaluates the proposed methods by comparing with 11 discretization algorithms using 4 classifiers with the objective to evaluate and compare the performance of the selected discretization algorithms.

Investigated datasets

The experimental evaluation is conducted on 30 standard datasets and 20 imbalanced datasets. The standard datasets obtained from the benchmark UCI repository [58] as summarized in Table 1. The imbalanced datasets achieved from Keel-dataset repository [59, 60] as described in Table 2. In these datasets, some datasets have only numeric attributes and some have both numeric and nominal attributes. Because a small number of distinct values of a numeric attribute could be ambiguous with the nominal attribute. In the table, a numeric attribute with no more than two distinct values is regarded as a nominal attribute. This study also evaluated the Toy dataset (see Fig. 3 for details) in order to compare the different results of the algorithms of interest.

Experiment design

An experiment is set up to investigate the performance of the proposed algorithms compared to 11 discretization algorithms, which have variously different techniques. For comparison, 4 classifiers of: C4.5 (j48) [61], K-Nearest Neighbors (KNN) [62], Naive Bayes (NB) [63], and Support Vector Machine (SVM) [64] classifiers which are in the top 10 algorithm in data mining [65, 66] are examined.

The standard datasets considered are partitioned using the tenfold cross-validation procedure [67]. Each discretization algorithm performed with pairs of 10 training and 10 testing of each dataset. For the imbalanced datasets, each dataset was separated in fivefold already, this study evaluates them using fivefold cross-validation procedure.

The performance measures

To evaluate the quality of the discretization algorithms, the following 3 measures are employed: number of intervals, running time, and predictive accuracy, respectively.

The number of intervals The result of the number of intervals is expected to be a small number as a large number of intervals may cause the learning to be slow and ineffective [7, 19], and the small number of intervals is easier to understand. However, too simple, too small discretization schemes may lead to lose classification performance [16].

Furthermore, this study compared the average number of intervals per attribute of the discreteization algorithms. Let $D^u = \{d_1^u, d_2^u, \ldots\}$ be a set of numeric attributes, $NC(d_i^u, r_j)$ be a number of cut points of attribute d_i^u of dataset r_j. The average number of

Table 1 Description of 30 standard datasets

Dataset	*n*	*d*	d^u	d^o	*c*
Australian	690	14	10	4	2
Autos	205	24	14	10	6
Banknote	1372	4	4	0	2
Biodeg	1055	41	38	3	2
Blood	748	4	4	0	2
Bupa	345	6	6	0	2
Cleve	295	13	6	7	2
Column2C	310	6	6	0	2
Column3C	310	6	6	0	3
Ecoli	336	8	5	3	8
Faults	1941	27	25	2	7
Glass	214	9	9	0	6
Haberman	306	3	3	0	2
Hayes	132	5	5	0	3
Heart	270	13	10	3	2
Hepatitis	155	19	6	13	2
ILPD	583	10	9	1	2
Ionosphere	351	34	32	2	2
Iris	150	4	4	0	3
Liver	345	6	6	0	2
Pima	768	8	8	0	2
Seeds	210	7	7	0	3
Segment	2310	19	18	1	7
Sonar	208	60	60	0	2
Tae	151	5	3	2	3
Transfusion	748	4	4	0	2
Vowel	990	13	11	2	11
Wine	178	13	13	0	3
Wisconsin	683	9	9	0	2
Yeast	1484	9	7	2	10

n, number of instances; *d*, number of attributes; d^u, number of numeric attributes; d^o, number of nominal attributes; *c*, number of classes

intervals per attribute of dataset r_x, $AI(r_x)$ is computed as Eq. 15, where $NC(d_i^u, r_x) + 1$ is the number of intervals of attribute d_i^u in the dataset r_x.

$$AI(r_x) = \frac{1}{|D^u|} \sum_{i=1}^{|D^u|} \left(NC(d_i^u, r_x) + 1 \right) \tag{15}$$

The running times To create a discretization scheme (cut points model), the process of discretization is done only once with the training dataset. It seems to be an important evaluation; however, it should not take too long in order to be able to discretize with the real world dataset, which normally high number of data points.

The predictive accuracy The successful discretization algorithm should perform discretization such that the predictive accuracies are increased or without significant reduction of predictive accuracy. This study evaluates the classification performance of 30

Table 2 Description of 20 imbalanced datasets

Dataset	n	$n_{\min}$	$k_{\min}$	d	d^u	d^o	c
Abalone19	4174	32	0.008	8	7	1	2
Abalone9-18	731	42	0.057	8	7	1	2
Ecoli-0_vs_1	220	77	0.350	7	5	2	2
Ecoli-0-1-3-7_vs_2-6	281	7	0.025	7	5	2	2
Ecoli1	336	77	0.229	7	5	2	2
Ecoli2	336	52	0.155	7	5	2	2
Ecoli3	336	35	0.104	7	5	2	2
Ecoli4	336	20	0.060	7	5	2	2
Glass0	214	70	0.327	9	9	0	2
Glass1	214	76	0.355	9	9	0	2
Glass2	214	17	0.079	9	9	0	2
Glass4	214	13	0.061	9	9	0	2
Glass5	214	9	0.042	9	9	0	2
Page-blocks0	5472	559	0.102	10	10	0	2
Pima	768	268	0.349	8	8	0	2
Segment0	2308	329	0.143	19	18	1	2
Vehicle0	846	199	0.235	18	18	0	2
Vowel0	988	90	0.091	13	11	2	2
Wisconsin	683	239	0.350	9	9	0	2
Yeast-0-5-6-7-9_vs_4	528	51	0.097	8	7	1	2

n, number of instances; n_{min}, number of minority classes; k_{min} minority class ratio; d number of attributes; d^u, number of numeric attributes; d^o, number of nominal attributes; c, number of classes

standard datasets using predictive accuracy. The predictive accuracy results of the dataset are summarized by the mean of its 10 folds.

AUC Because the predictive accuracy is not a suitable measure for imbalanced data, this study evaluates 20 imbalanced datasets using AUC (area under the ROC curve) [68, 69]. The AUC is wildly used for classification evaluation for imbalanced data [70, 71], which measure the diagnostic accuracy of a test. The AUC value lies between 0 and 1, the higher value is the better average accuracy test. This study summarized AUC results of 20 imbalanced dataset by the mean of its 5 folds.

Statistical analysis

In this study, the ranking of discretization algorithms is obtained from the Friedman test [72, 73]. The discretization algorithm that obtains the lowest rank value is the better performance. Friedman test is non-parametric statistical test for assessing the difference between several related samples. It gives the mean rank for each discretization algorithm based on median difference, such that the algorithm with high averaged accuracy may not be in the first rank. If the Friedman test can show that at least one algorithm is significantly different from at least another algorithm (using a level of significance $\alpha = 0.05$), two post-hoc tests of Nemenyi and Holm are used to find the significantly different.

1. The Nemenyi post-hoc test [74] is used to find the critical difference (CD). Two algorithms are significantly different if the corresponding average ranks differ by at least the CD (using 95% confident level).

2. The Holm post-hoc test [75, 76] is used to find the p-value of the post-hoc Holm (p_{Holm}) of each pair comparison. The discretization algorithm that obtains the lowest rank value is set as a control algorithm. The control algorithm is compared against the rest algorithms.

These tests performed with three comparisons of: a ranking of the number of intervals, ranking of running time, and ranking of classification performances (predictive accuracy and AUC).

Compared discretization algorithms

In order to properly examine the potential of the proposed discretization algorithms (GraphS, GraphM), they are evaluated against 11 well-known discretization algorithms. Those algorithms are Ameva, CAIM, ChiMerge, EMD, FFD, FUSINTER, HDD, MChi2, PKID, ur-CAIM, and Zeta, which various discretization types as shown in Fig. 10. EMD and ur-CAIM are recently published which show high classifier's performance. This study takes ur-CAIM program that is distributed from the authors ur-CAIM[2], EMD program is derived from the authors of EMD, and the rest discretization programs are taken from KEEL software [59, 60]. The programs of GraphS and GraphM are included as an Additional file 1. The details of comparison algorithms are described as:

Ameva [77] An autonomous discretization algorithm (Ameva) is univariate, supervised and splitting methods. The discretization criterion of the algorithm based on χ^2 values. There are two objectives of Ameva: maximize the dependency relationship between the target class and an attribute, and minimize the minimum number of intervals.

CAIM [15] Class-Attribute Interdependence Maximization discretization algorithm (CAIM) is a splitting method proposed by Kurgan and Cios. The goal of the algorithm is to find the minimum number of discrete intervals, while minimizing the loss of class-attribute interdependency for the optimal discretization with a greedy approach. It iteratively finds the best cut point in order to split into two intervals until the stopping criterion is satisfied. The algorithm mostly generates discretization schemes that the number of intervals equal to the number of classes [16, 78].

ChiMerge [11] ChiMerge is a merging method introduced by Kerber. It uses a χ^2 values as a discretization criterion. It divides the instances into intervals of distinct values and then iteratively merges the best adjacent intervals until the stopping criterion is fulfilled. The stopping criterion of ChiMerge is related to the $\chi^2-threshold$, and in order to compute the threshold the user must specify the significance level.

EMD [31] The Evolutionary Multivariate Discretizer (EMD) is a multivariate method based on CHC algorithm [79], the subclass of Genetic algorithm that is one of the powerful search methods. The algorithm defines a fitness function for two objectives of: the lower classification error (based on C4.5 and NB classifiers) and the lower number of cut points. A chromosome is encoded as a binary array of the cut-points selection, 1 is selected and 0 otherwise. The chromosome is encoded for all possible cut-points of continuous attributes. Therefore, the algorithm requires a lot of time in order to search for the optimal result, especially in high dimension of data and large number of instances.

[2] http://www.uco.es/grupos/kdis/wiki/ur-CAIM.

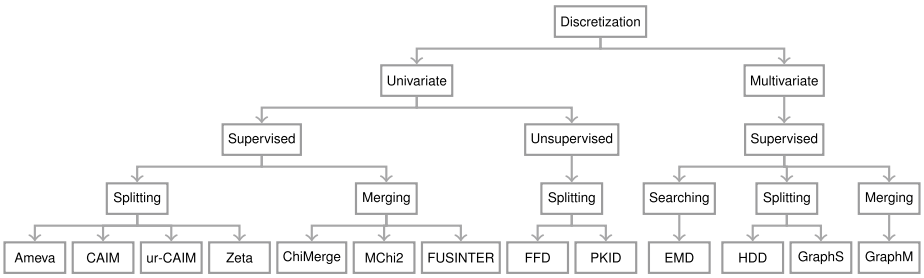

Fig. 10 Discretization categories. The categories of discretization methods of 13 discretization algorithms

FFD and PKID [21] These algorithms are unsupervised method proposed by Young and Webb. The key of the algorithms is to maintain discretization bias and variance by tuning the interval frequency and the interval numbers, especially used in Naive-Bayes classier. The *fixed frequency discretization* (FFD) sets a sufficient interval frequency m, then discretizes such that all intervals have approximately the same number m of training instances with adjacent values. The *proportional discretization* (PKID) sets the interval frequency and the interval number to be proportional to the amount of training data in order to achieve a low variance and a low bias.

FUSINTER [80] This algorithm is a greedy merging method. It uses the same strategy as ChiMerge. The main characteristic of the algorithm is that it is based on the sensitivity measure of the sample size to avoid very thin partitioning. The algorithm first merges the adjacent intervals that all instances of the intervals are the same target class, and continues until no improvement is possible or the number of intervals reaches 1. The user must specify 2 parameters, α and λ, which are significance level and variable tuning, in order to control the performances of the discretization procedure.

HDD [3] HDD extends CAIM by improving the stopping criterion and discretization by taking other attributes into account (multivariate method). The algorithm considers the distribution of both target class and continuous attributes. The algorithm divides the continuous attribute space into a finite number of hypercubes that the objects within each hypercube belongs to the same decision class. However, the algorithm mostly generates a large number of intervals and has slow discretization time [16].

MChi2 [13] Modified Chi2 (MChi2) is proposed by Tay and Shen. It is a merging method using statistic-based (χ^2) to determine the similarity of the adjacent intervals. The algorithm enhances Chi2 algorithm [12] by making the discretization process completely automatic. It replaces the inconsistency check in the Chi2 with a level of consistency that is approximated after each step of discretization. The algorithm also considers the factor of degree of freedom in order to improve the accuracy.

ur-CAIM [16] The algorithm proposed by Janssens et al. It improves CAIM which combines the CAIR [81], CAIU [82], and CAIM discretization criterion together in order to generate more flexible discretization schemes, require lower running time than CAIM, and improve predictive accuracy, especially in unbalanced data.

Zeta [83] This algorithm is introduced by Ho and Scott. It is a direct method that the user must specify the number of intervals, k, where each attribute is discretized into k intervals. The discretization criterion of the algorithm is based upon *lambda* [84] that is

widely used to measure strength association between nominal variables. This criterion is defined as the maximum accuracy achievable when each value of a feature predicts a different class value.

Parameter settings

Many discretizaion algorithms desire the input parameters from the user before discretization. Same parameters of the classifiers must also be set. Since the different parameters affect the performance, in order to evaluate the quality of discretization algorithms, those parameters of discretization algorithms and classifiers specified in Table 3 are set as recommended by [7] and [31] .

Experiment results and analysis of standard datasets

Number of intervals

The average numbers of intervals per attribute of 30 standard datasets for 13 discretization algorithms are shown in Fig. 11a (calculated with Eq. 15). The details of the result are included as an Additional file 2.

In the figure, the lowest average number of intervals per attribute of all datasets belongs to EMD (2.04), the second is Ameva (2.51), and the third is ChiMerge (2.99). Since EMD uses the predictive accuracy of C4.5 and NB as a part of the fitness function, some attributes excluded from the final classification model will have no cut-point, i.e., the whole attribute domain is considered as only 1 interval. For ChiMerge, the number of intervals depends on the user's specification on the significant level (α). If the user sets this value high (α close to 0), the algorithm will over merging, leading to a low number of intervals. EMD and ChiMerge generate one number of intervals of some attributes, the attributes are removed in classification learning. Unlike EMD and ChiMerge that are an implicit coupling of supervised feature selection and discretization, GrpahM and GraphS concentrate on the later, with the possibility to combine with many advanced feature selection methods. The average number of remove attributes of all discretization algorithms is summarized in Fig. 11c.

Many datasets of CAIM, and Zeta have the lowest number of intervals. CAIM discretizes with the number of intervals close to the number of target classes. If there are many target classes, the number of intervals of CAIM is high too (if there is enough number of

Table 3 Parameters of classifiers and discretizers

Method	Parameters
Classifier	
C4.5	Pruned tree = *true*, confidence = 0.25, minimum example per leaf = 2
KNN	$k = 3$, distanced function = *EuclideanDistance*
Discretizer	
ChiMerge	Confidence threshold = 0.05
EMD	Population size = 50, $M_e = 10,000$, $\alpha = 0.7$, $R_{rate} = 0.1$, $R_{perc} = 0.5$
FFD	Frequency size = 30
FUSINTER	$\alpha = 0.975$, $\lambda = 1$
HDD	Coefficient = 0.8
GraphS, GraphM	$\beta = 1.01$

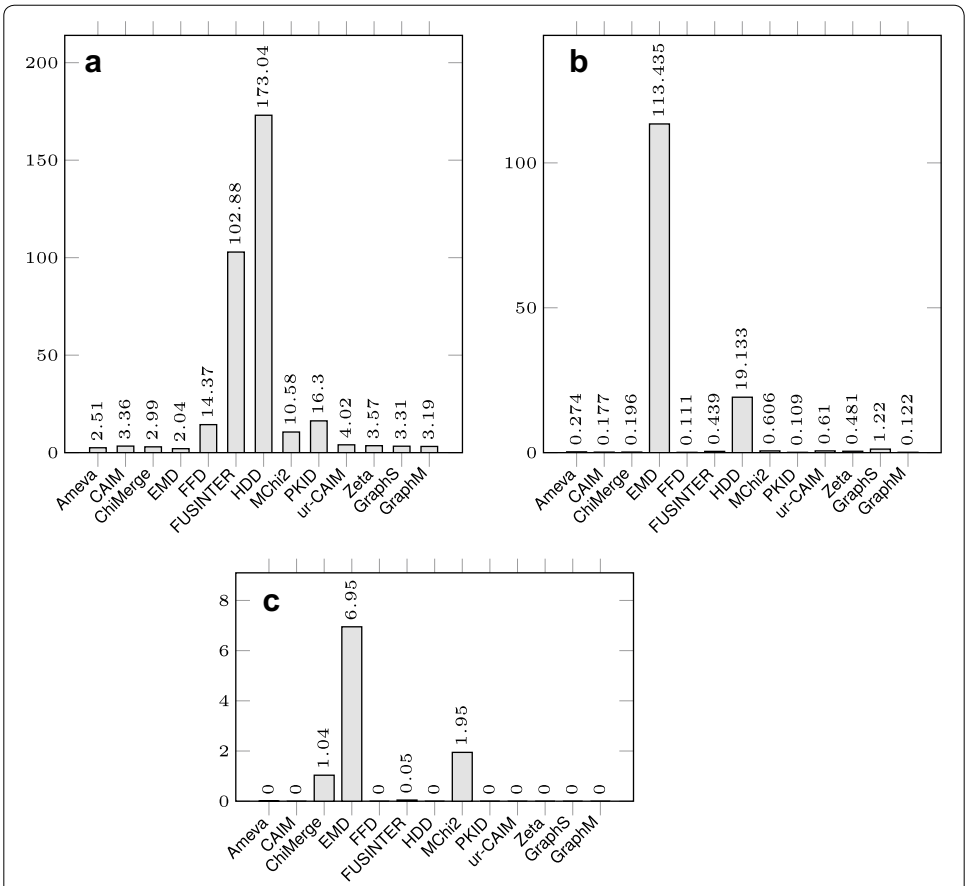

Fig. 11 Discretization results of: **a** average number of intervals, **b** average running times (seconds), and **c** average number of remove attributes for standard datasets

cut points to split). Zeta is a direct method that fixes the number of intervals equal the number of target classes. Resulting in all attributes having the equal number of intervals.

Running time

Figure 11b presents the actual computational time (in seconds) required for creating the discretization scheme and discretization from the experimented datasets. These algorithms are implemented in Java and all experiments are conducted on an Intel(R) Xeon(R) CPU@2.40 GHz and 4 GB RAM. The execution time is measured using the *System.currentTimeMillis*() Java method (*endTime* − *startTime*).

The execution time averages from tenfold discretization (the time of classifier learning not included). The fastest technique over 30 standard datasets is PKID (0.109 s), while the slowest is EMD (113.435 s). With these, EMD is more than a 1000 times slower than PKID. That is because EMD is designed as an evolutionary process, which uses a wrapper fitness function with the chromosome encoding all cut-point of examined attributes. Without applying any approximation heuristics, EMD naturally requires a long time to look for the optimal fitness value.

GraphS and GraphM are multivariate method similar to EMD and HDD, however, their running times are similar to the other univariate discretization algorithms. The

average running time of GraphM is more than ten times faster than GraphS. It is because GraphM iteratively merged considering two adjacent intervals. In contrast, GraphS considered all cut points in order to find the best cut point and hence lose some time in searching. More details are discussed in "Time complexity and parameter analysis".

Predictive accuracy

Based on the classification predictive accuracy, Fig. 12 summarizes the average predictive accuracies over C4.5, KNN, NB, SVM, and all classifiers. The details of these results are included as an Additional file 2.

According to these results, the three discretization algorithms with the highest average predictive accuracy across 30 standard datasets with C4.5 classifier are EMD (78.20%), GraphS (77.88%), and GraphM (77.79%), respectively. The similar observation with KNN classifier are GraphS (78.04%), ur-CAIM (77.95%), and GraphM (77.73%). The three highest average predictive accuracy with NB classifier are ur-CAIM (77.24%), GraphS (77.20%), and EMD (77.12%). The results indicate that GraphS, GraphM, ur-CAIM, and EMD generally performed better than the rest techniques included in this experiment. With respect to the overall measures across three classifiers, the first highest accuracy belongs to GraphS (77.42%), the second is ur-CAIM (77.2%), and the third is GraphM (77.16%). As such, it has been demonstrated that the graph clustering-based discretization algorithm is usually more accurate than many other well-known counterparts.

Friedman rankings with critical differences (CD)

The average ranking of 13 discretization algorithms with three comparisons of: number of intervals, running times, and predictive accuracy is shown in Figs. 13 and 14. Because all the p value of these comparisons based on Friedman test are 0.000, all the algorithms are not equivalent ranking. Therefore, the critical differences (CD) based on Nemenyi post-hoc test and p_{Holm} of Holm post-hoc test can be computed. Each figure, the left top horizontal line is the CD interval, the second line that rank from 1 to 13 is the axis that contains average ranks of 13 algorithms, and the best value ranking is given as a small value. Note that the tick horizontal lines are the marked intervals. If the distance between algorithm ranks does not over the CD interval, the marked interval will be shown, i.e., they are not significantly different.

In the ranking of average number of intervals, EMD and ChiMerge are the first and second lowest ranking, there is no significantly different of the pair. Otherwise, EMD shows significantly lower number of intervals than the rest algorithms. In the ranking of average running times, ur-CAIM obtains the first ranking, whereas EMD appears to be the last. In addition, ur-CAIM is not significantly different to GraphM, CAIM, ChiMerge, FFD, PKID, Zeta, and GraphS.

In order to finely compute the ranking of predictive accuracy of each discretization algorithm, 300 predictive accuracy results over 30 standard datasets and tenfolds evaluation are examined. In Fig. 14, the lowest average accuracy rankings for C4.5 classifier belongs to EMD. However, it is not significantly better to GraphM, GraphS, ur-CAIM, and Ameva. GraphM obtains the lowest ranking for KNN classifier, however, it is not significantly different to ur-CAIM, GraphS, and EMD. Three lowest rankings for NB classifier are GraphM, GraphS, and EMD, there are no significantly different. In the average

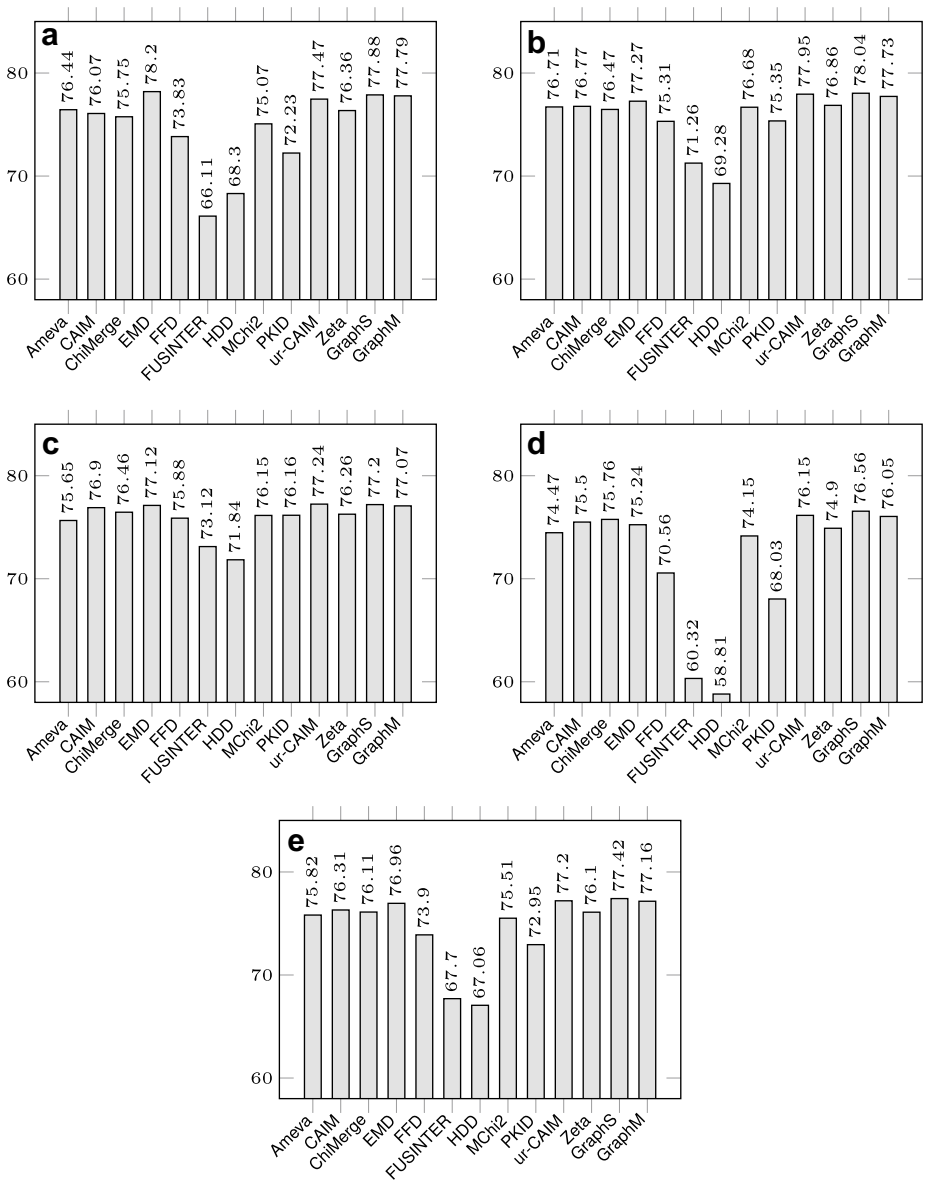

Fig. 12 Predictive accuracy results of standard datasets. Average predictive accuracy of: **a** C4.5 classifier, **b** K-Nearest Neighbors classifier, **c** Naive Bayes classifier, **d** Support Vector Machine classifier, and **e** all classifiers

ranking for SVM classifier, GraphS obtains the lowest ranking, but it is not significantly better than ChiMerge, ur-CAIM, GraphM, and CAIM. In summary, the average ranking over all classifiers, GraphM, GraphS, and ur-CAIM obtain the first, second, and third lowest rankings, respectively, there are no significantly different. However, GraphM and GraphS are significantly better than the rest algorithms.

Given these findings, GraphM and GraphS can be useful not only for data analysis with high accuracy, but also with a reasonable time requirement. Also, the resulting discretized dimensions can be coupled with many effective feature selection approaches found in the literature.

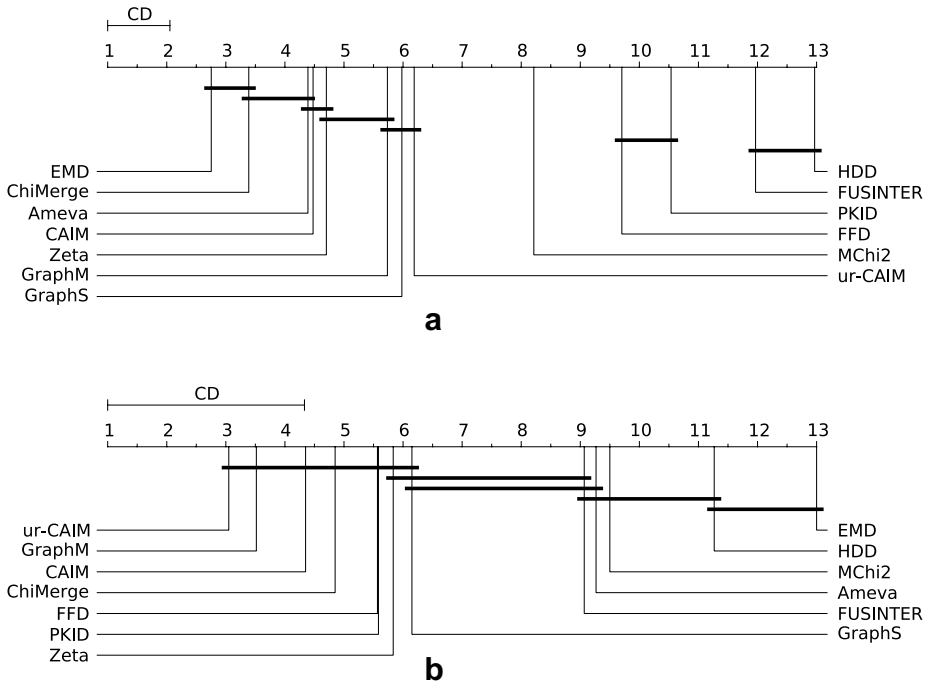

Fig. 13 The average rankings of: **a** average number of intervals and **b** average running times for standard datasets. These rankings obtain from the non-parametric Friedman test and the critical difference (CD) of the Nemenyi post hoc test. Note that, the ranking of number of intervals in **a** is tested with 300 samples (30 standard datasets and tenfolds), the ranking of running times in **b** is tested with 30 samples (30 standard datasets)

Friedman rankings with p_{Holm}

Because the Friedman test of all comparisons are not equivalent ranking, p_{Holm} of Holm post-hoc test can be computed. All results of the tests are provided in Table 4. In order to compare the p_{Holm}, This study used a level of significance is 0.01. If the p_{Holm} value is lower than 0.01, there is significantly different of the pair.

The results of p_{Holm} test for a number of intervals, running times, and predictive accuracy of all classifiers is similar to the Nemenyi post-hoc test. EMD is not significantly lower number of intervals than ChiMerge. ur-CAIM is not significantly faster than GraphM, CAIM, ChiMerge, FFD, PKID, and GraphS. For average ranking of predictive accuracy over all classifiers, GraphM, GraphS, and ur-CAIM obtain the first, second, and third lowest ranking. By using significant level $\alpha = 0.1$, GraphM is not significantly better accuracy than GraphS and ur-CAIM, in fact, GraphM is significantly better accuracy than the rest discretization algorithms. Besides, by using $\alpha = 0.05$, GraphM shows significantly better predictive accuracy than the other well-known discretization algorithms.

Experiment results and analysis of imbalanced datasets

Number of intervals

The average numbers of intervals per attribute of 20 imbalanced datasets for 13 discretizers are shown in Fig. 15a. The details of the result are included as an Additional file 3.

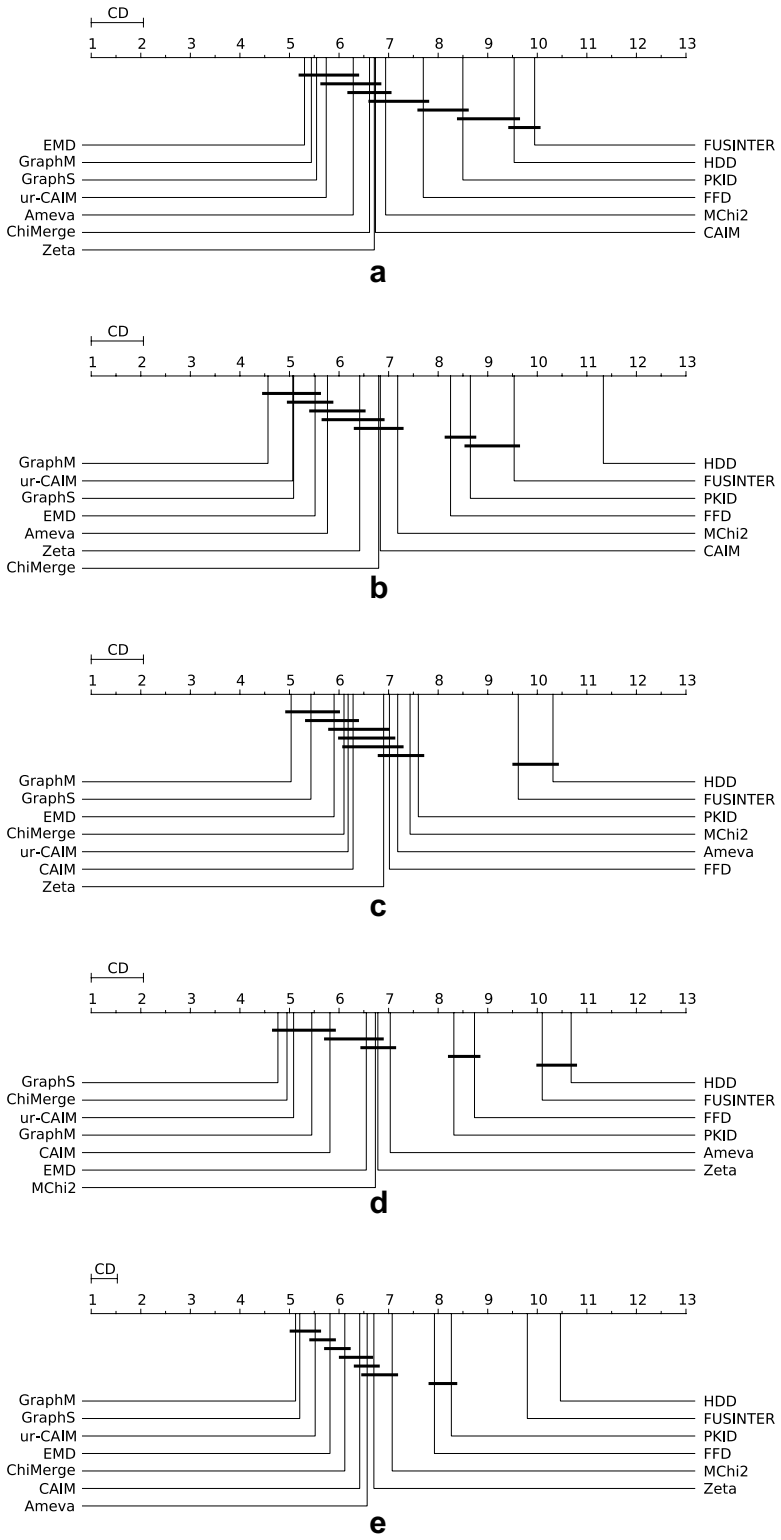

Fig. 14 Ranking of predictive accuracies for standard datasets. Average predictive accuracies ranking with different classifiers of: **a** C4.5, **b** KNN, **c** NB, **d** SVM, and **e** all classifiers, obtain from the non-parametric Friedman test and the critical difference (CD) of the Nemenyi post hoc test. Note that, the ranking in **a**,**b**, **c**, and **d** are tested with 300 samples (30 standard datasets and tenfolds), while the ranking in **e** is tested with 1200 samples (30 standard datasets of 4 classifiers and tenfolds)

Table 4 Average Friedman rankings and p_{Holm} of the number of intervals, running time, and predictive accuracy for standard datasets

Discretizer	Raking	p_{Holm}	Discretizer	Raking	p_{Holm}
Number of intervals			Running times		
EMD	2.7533	–	ur-CAIM	3.05	–
ChiMerge	3.3883	0.045827	GraphM	3.5167	0.642579
Ameva	4.39	0.000001	CAIM	4.35	0.392135
CAIM	4.4767	0	ChiMerge	4.85	0.220322
Zeta	4.7017	0	FFD	5.5667	0.058782
GraphM	5.735	0	PKID	5.5833	0.058782
GraphS	5.98	0	Zeta	5.8333	0.033841
ur-CAIM	6.1883	0	GraphS	6.15	0.014349
MChi2	8.2133	0	FUSINTER	9.0667	0
FFD	9.7033	0	Ameva	9.2667	0
PKID	10.5367	0	MChi2	9.5	0
FUSINTER	11.9667	0	HDD	11.2667	0
HDD	12.9667	0	EMD	13	0
Accuracy of C4.5			Accuracy of KNN		
EMD	5.3033	–	GraphM	4.5667	–
GraphM	5.4417	0.888248	ur-CAIM	5.0667	0.05
GraphS	5.5467	0.888248	GraphS	5.0833	0.025
ur-CAIM	5.7417	0.504152	EMD	5.5167	0.016667
Ameva	6.2867	0.007941	Ameva	5.7667	0.0125
ChiMerge	6.615	0.000185	Zeta	6.4167	0.01
Zeta	6.7117	0.000057	ChiMerge	6.8	0.008333
CAIM	6.735	0.000047	CAIM	6.8333	0.007143
MChi2	6.94	0.000002	MChi2	7.1833	0.00625
FFD	7.7	0	FFD	8.25	0.005556
PKID	8.4983	0	PKID	8.65	0.005
HDD	9.5333	0	FUSINTER	9.5333	0.004545
FUSINTER	9.9467	0	HDD	11.3333	0.004167
Accuracy of NB			Accuracy of SVM		
GraphM	5.0333	–	GraphS	4.7667	–
GraphS	5.4333	0.208413	ChiMerge	4.95	0.638626
EMD	5.9	0.012839	ur-CAIM	5.0833	0.638626
ChiMerge	6.1	0.002385	GraphM	5.45	0.094907
ur-CAIM	6.1833	0.001194	CAIM	5.8167	0.003839
CAIM	6.2833	0.000423	EMD	6.55	0
Zeta	6.9	0	MChi2	6.7333	0
FFD	7.0167	0	Zeta	6.7833	0
Ameva	7.1833	0	Ameva	7.0333	0
MChi2	7.4333	0	PKID	8.3167	0
PKID	7.6	0	FFD	8.7333	0
FUSINTER	9.6167	0	FUSINTER	10.1	0
HDD	10.3167	0	HDD	10.6833	0
Accuracy of all classifiers					
GraphM	5.1229	–			
GraphS	5.2075	0.594723			
ur-CAIM	5.5188	0.025572			
EMD	5.8175	0.000037			
ChiMerge	6.1163	0			
CAIM	6.4171	0			

Table 4 continued

Discretizer	Raking	p_{Holm}	Discretizer	Raking	p_{Holm}
Ameva	6.5675	0			
Zeta	6.7029	0			
MChi2	7.0725	0			
FFD	7.925	0			
PKID	8.2663	0			
FUSINTER	9.7992	0			
HDD	10.4667	0			

The average number of intervals per attribute result is similar to the standard datasets, which the lowest and the highest average number of intervals belong to EMD (1.48) and HDD (154.51), respectively. Some discretization algorithms are an implicit coupling with supervised feature selection, especially EMD, ChiMerge, and MChi2 as shown in Fig. 15c, average number of remove attributes. The proposed methods obtain the lower number of intervals, the average values of GraphS and GraphM is 2.4. They do not remove any attributes. The proposed methods can be combined with advanced feature selection methods on the latter, the classification performance may improve.

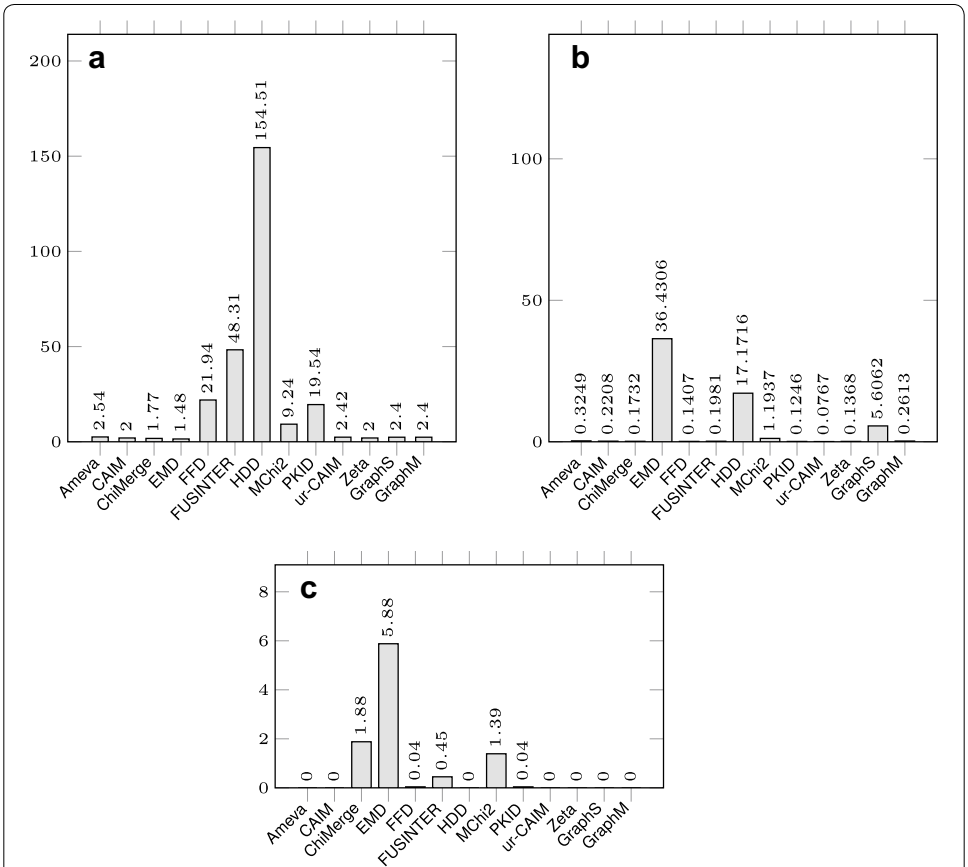

Fig. 15 Discretization results of: **a** average number of intervals, **b** average running times (seconds), and **c** average number of remove attributes for imbalanced datasets

Running time

The running time result is averaged from fivefold discretization, including two processes of the crating discretization scheme and transform numeric to nominal dataset. The fastest running time belongs to ur-CAIM (0.0757 s), while the slowest is EMD (36.4306 s) as shown in Fig. 15b, average running time in seconds. Generally a multivariate method requires higher running time than univariate method. GraphS and GraphM are multivariate methods same as EMD and HDD. However, their running time is faster than EMD and HDD, which the running time is similar to univariate method. The details of running time results are included as an Additional file 3.

AUC

Based on the classification performance of AUC, Fig. 16 summarizes the average AUC of 20 imbalanced datasets over C4.5, KNN, NB, SVM, and all classifiers. For more details of these results, this study included the results as an Additional file 3. The proposed methods usually achieve high AUC value. For the average of all classifiers, GraphS, GraphM, and ur-CAIM show the first, second, and third highest AUC values that are 0.8324, 0.8299, and 0.8188, respectively. The results suggest that the proposed methods can handle with imbalanced datasets, which usually achieve high AUC value than many well-known discretizers.

Friedman rankings with critical differences (CD)

Based on Friedman ranking test, 20 imbalanced datasets, Figs. 17 and 18 show the ranking of: number of intervals, running times, and AUC. Each ranking is not equivalent ranking (p value of the test is 0.000), the critical differences (CD) based on Nemenyi post-hoc test is included in the figures.

This ranking of average number of intervals is similar to the ranking of the standard datasets, EMD and ChiMerge show the first and second lowest ranking, while HDD shows the highest ranking. In the ranking of average running times, GraphM obtains the first ranking, however, it is not significantly different to ur-CAIM, Zeta, ChiMerge, FDD, PKID, and GraphS. The slowest running time still belongs to EMD.

The average ranking of AUC for C4.5, KNN, NB, SVM, and all classifiers shown in Fig. 18. In the ranking of AUC for C4.5 and SVM classifiers, the top three lowest rankings belong to EMD, GraphM, and GraphS. These tree discretizers are close in the rankings and are not significant differences. In the average ranking of AUC for KNN classifier, GraphS and GraphM obtain the lowest ranking. However, they are not significant differences to FFD, PKID, ur-CAIM, and MChi2. For NB classifier, the first and second lowest ranking belongs to FFD and PKID, where GraphS and GraphM lie in the fourth and seventh ranking, respectively. In summary, the average ranking over all classifiers, GraphS and GraphM obtain the first and second best ranking. However, they are not significantly different to ur-CAIM.

These ranking suggests that the proposed methods can be useful for imbalanced data, achieve the best ranking of AUC. Furthermore, they require lower running times and obtain a lower number of intervals.

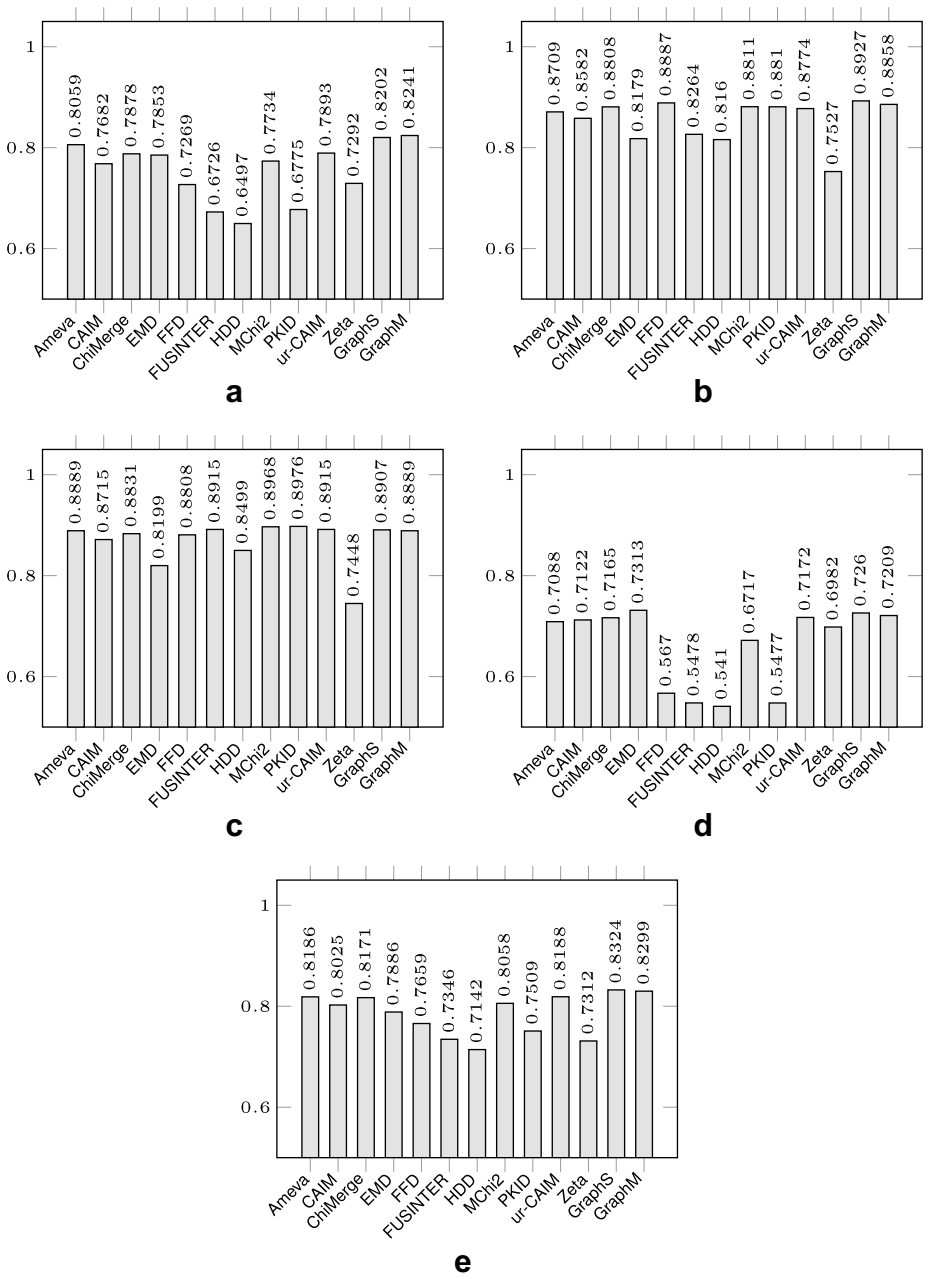

Fig. 16 AUC results of imbalanced datasets. Average AUC of: **a** C4.5 classifier, **b** K-Nearest Neighbors classifier, **c** Naive Bayes classifier, **d** Support Vector Machine classifier, and **e** all classifiers

Friedman rankings with p_{Holm}

Because the Friedman ranking test of the number of intervals, running times, and AUC of imbalanced datasets are not equivalent ranking test, these tree ranking can compute p_{Holm} of Holm post-hoc test. The p_{Holm} results of these rankings are provided in Table 5. In order to compare p_{Holm} of the rankings, the significant level (α) is set to 0.01. If the p_{Holm} value is lower than 0.01, there is significantly different of the pair.

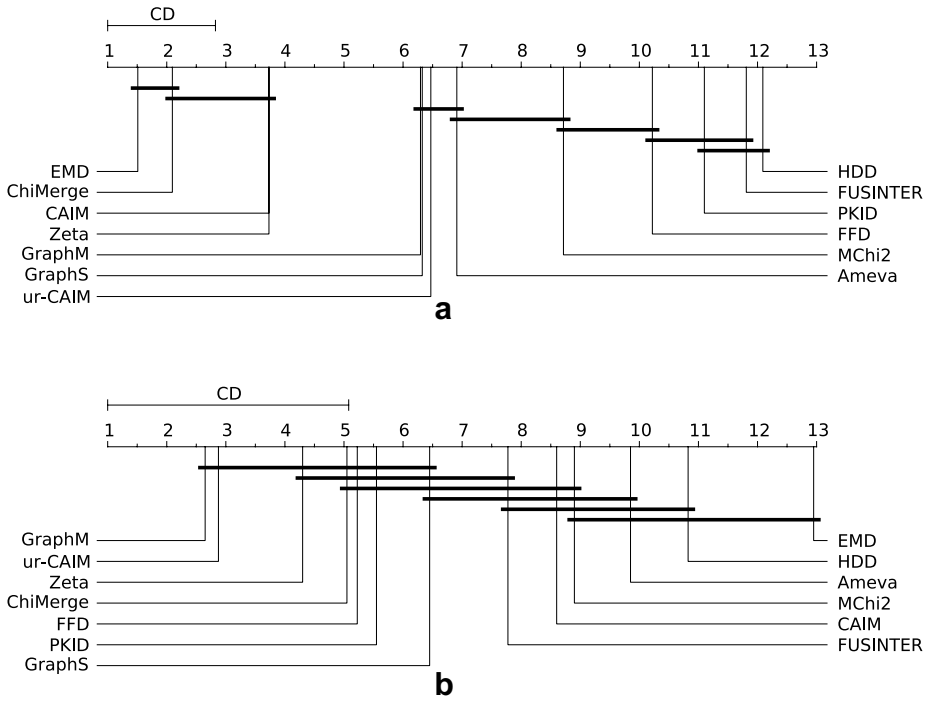

Fig. 17 The average rankings of: **a** average number of intervals and **b** average running times for imbalanced datasets. These rankings obtain from the non-parametric Friedman test and the critical difference (CD) of the Nemenyi post hoc test. Note that, the ranking of number of intervals in **a** is tested with 100 samples (20 imbalanced datasets and fivefolds), the ranking of running times in **b** is tested with 20 samples (20 imbalanced datasets)

The p_{Holm} results of the average ranking number of intervals shows that EMD is not a significant difference to ChiMerge. However, it is significant lower number intervals to the rest discretizers. In the average ranking running time, GraphM is not significantly faster than ur-CAIM, Zeta, ChiMerge, FFD, PKID, and GraphS. For an average ranking of AUC over all classifiers, GraphS is not significantly better AUC than GraphM and ur-CAIM. However, by using $\alpha = 0.05$, GraphS shows significantly better AUC than the other well-known discretization algorithms.

Experiment results of Toy dataset

This section aims at showing characteristics of different discretization algorithms on the Toy dataset. The discretization results of Toy dataset are represented as a scatter plot as shown in Fig. 19, where the vertical dashed lines represent the cut points of attribute A_1 and horizontal dashed lines represent the cut points of attribute A_2.

The results show that, ChiMerge and FFD created no cut points. The number of cut points selected by ChiMerge depends on the significant level (α), and in this study α is set to 0.05 (recommended by the authors of ChiMerge). Too high significant level (α is close to 0) will lead to over merging. In order to allow the algorithm to select the cut points, the significant level should be reduced. FFD algorithm is an unsupervised method that the user must specify the frequency size. In this study the frequency size is set 30 (recommended by the authors of FFD). Since the number of the data points is

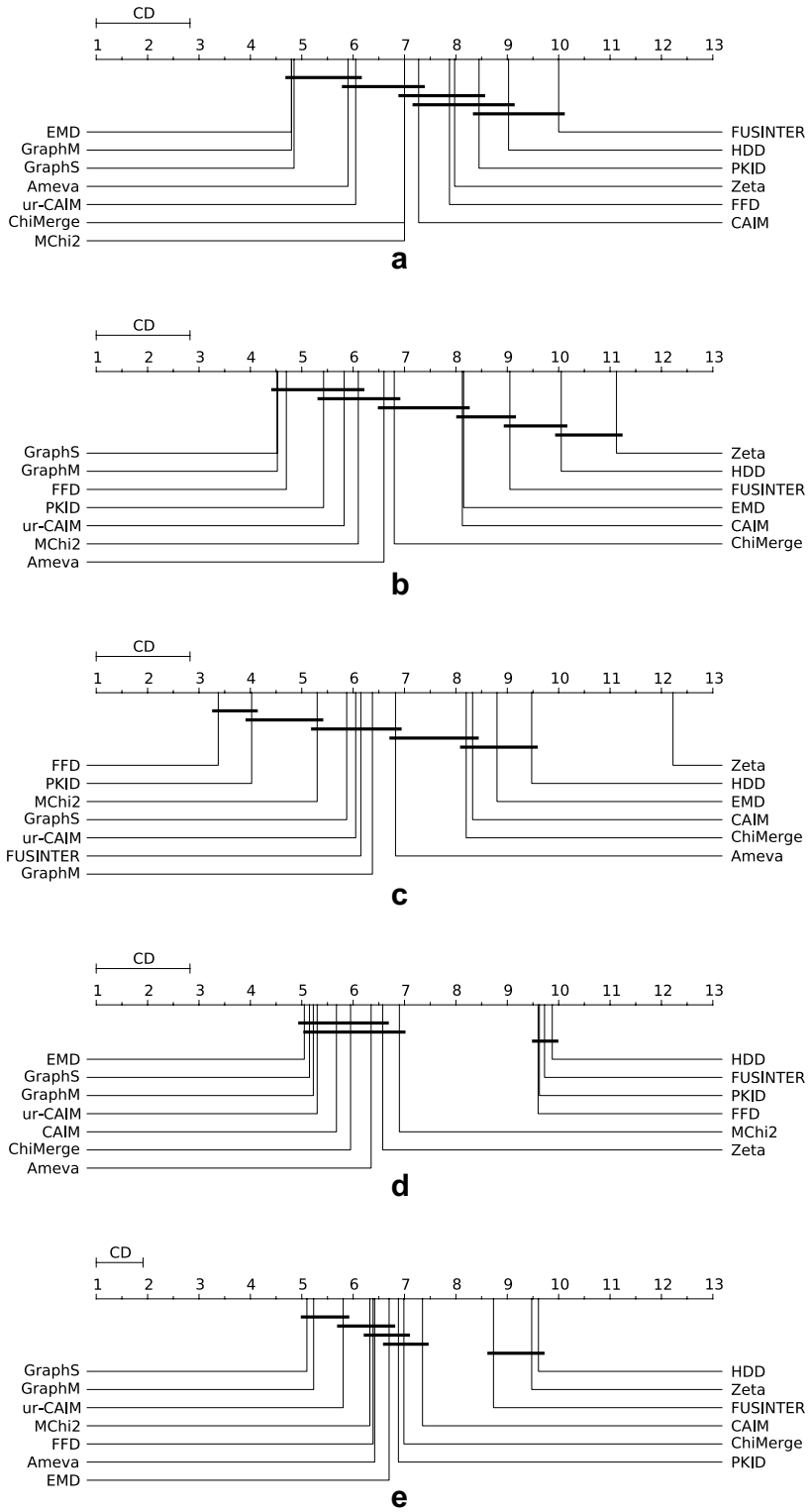

Fig. 18 Ranking of AUC for imbalanced datasets. Average AUC ranking with different classifiers of: **a** C4.5, **b** KNN, **c** NB, **d** SVM, and **e** all classifiers, obtain from the non-parametric Friedman test and the critical difference (CD) of the Nemenyi post hoc test. Note that, the ranking in **a**–**d** are tested with 100 samples (20 imbalanced datasets and fivefolds), while the ranking in **e** is tested with 400 samples (20 imbalanced datasets of 4 classifiers and fivefolds)

Table 5 Average Friedman rankings and p_{Holm} of the number of intervals, running time, and AUC for imbalanced datasets

Discretizer	Raking	p_{Holm}	Discretizer	Raking	p_{Holm}
Number of intervals			Running times		
EMD	1.51	–	GraphM	2.65	–
ChiMerge	2.095	0.288157	ur-CAIM	2.875	0.855034
CAIM	3.73	0.000167	Zeta	4.3	0.360623
Zeta	3.73	0.000167	ChiMerge	5.05	0.15396
GraphM	6.295	0	FFD	5.225	0.14615
GraphS	6.325	0	PKID	5.55	0.092665
ur-CAIM	6.47	0	GraphS	6.45	0.012189
Ameva	6.91	0	FUSINTER	7.775	0.000221
MChi2	8.715	0	CAIM	8.6	0.000011
FFD	10.22	0	MChi2	8.9	0.000003
PKID	11.1	0	Ameva	9.85	0
FUSINTER	11.81	0	HDD	10.825	0
HDD	12.09	0	EMD	12.95	0
AUC of C4.5			AUC of KNN		
EMD	4.8	–	GraphS	4.525	–
GraphM	4.8	1.855328	GraphM	4.525	1.501358
GraphS	4.85	1.855328	FFD	4.7	1.501358
Ameva	5.9	0.137394	PKID	5.425	0.306705
ur-CAIM	6.05	0.092927	ur-CAIM	5.825	0.073023
ChiMerge	7	0.000389	MChi2	6.1	0.021202
MChi2	7	0.000389	Ameva	6.6	0.000989
CAIM	7.275	0.000049	ChiMerge	6.8	0.000253
FFD	7.875	0	CAIM	8.125	0
Zeta	7.975	0	EMD	8.15	0
PKID	8.45	0	FUSINTER	9.05	0
HDD	9.025	0	HDD	10.05	0
FUSINTER	10	0	Zeta	11.125	0
AUC of NB			AUC of SVM		
FFD	3.375	–	EMD	5.05	–
PKID	4.025	0.237923	GraphS	5.15	1.949658
MChi2	5.3	0.000947	GraphM	5.225	1.949658
GraphS	5.875	0.000017	ur-CAIM	5.3	1.949658
ur-CAIM	6.05	0.000005	CAIM	5.675	1.025834
FUSINTER	6.15	0.000002	ChiMerge	5.95	0.511174
GraphM	6.375	0	Ameva	6.35	0.109535
Ameva	6.825	0	Zeta	6.575	0.03937
ChiMerge	8.2	0	MChi2	6.9	0.006258
CAIM	8.325	0	FFD	9.6	0
EMD	8.8	0	PKID	9.625	0
HDD	9.475	0	FUSINTER	9.725	0
Zeta	12.225	0	HDD	9.875	0
AUC of all classifiers					
GraphS	5.1	–			
GraphM	5.2312	0.633635			
ur-CAIM	5.8062	0.020656			
MChi2	6.325	0.000026			
FFD	6.3875	0.000012			
Ameva	6.4187	0.000008			

Table 5 continued

Discretizer	Raking	p_{Holm}	Discretizer	Raking	p_{Holm}
EMD	6.7	0			
PKID	6.8812	0			
ChiMerge	6.9875	0			
CAIM	7.35	0			
FUSINTER	8.7312	0			
Zeta	9.475	0			
HDD	9.6062	0			

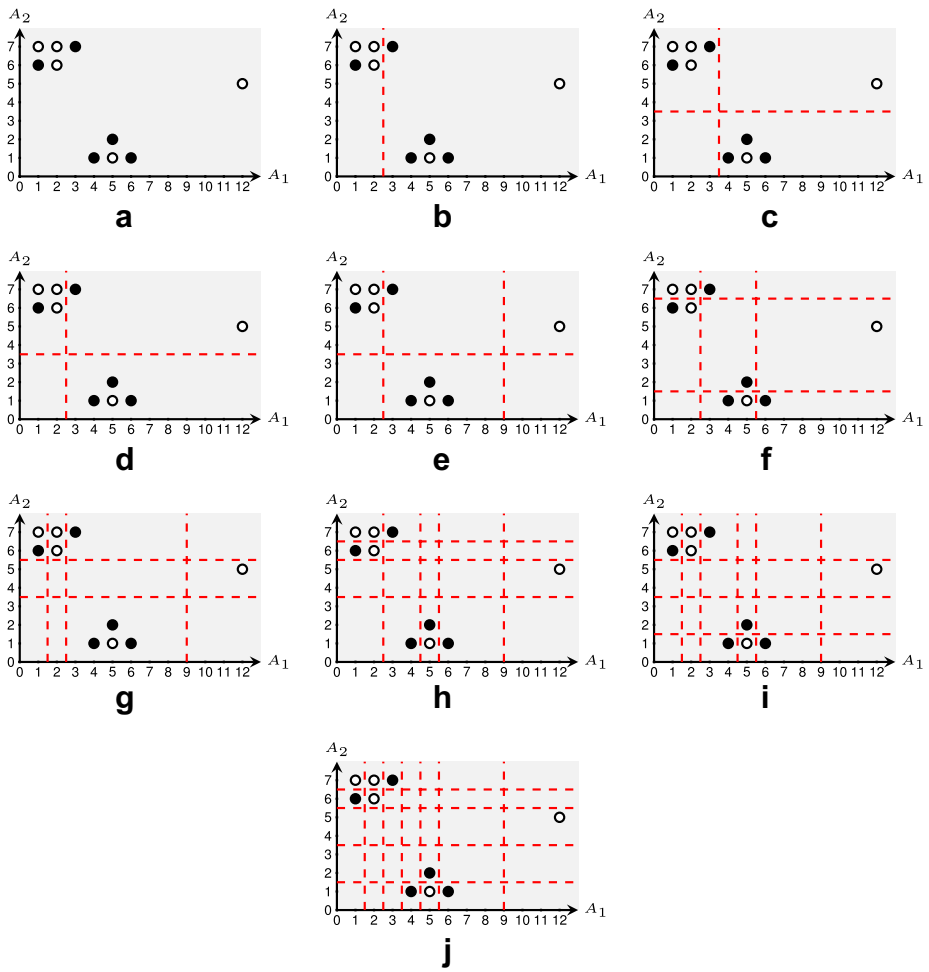

Fig. 19 The cut-point results with a Toy dataset for 13 discretization algorithms of: **a** ChiMerge and FFD, **b** EMD, **c** GraphS and GraphM, **d** CAIM and Zeta, **e** Ameva, **f** PKID, **g** ur-CAIM, **h** FUSINTER, **i** MChi2, and **j** HDD. Note that, the *vertical dashed-lines* represent the cut points of attribute A_1 and *horizontal dashed-line* represent the cut points of attribute A_2

no more than 30, all data points are grouped into one interval. In order to achieve the desired cut points, the frequency size should be smaller than the number of data points.

The result cut-point selection using EMD is only a single cut-point at 2.5 of attribute A_1. Because the fitness function of EMD is weighted from the lower predictive error and the lower number of intervals, the attributes that do not use to create the classifier model

mostly generate null cut-point. Specific to the aforementioned result, this selected cut-point also damages the natural groups of data, which is illustrated by the upper-left 5 data points.

The proposed algorithms of GraphS and GraphM show the same discretization results. The cut points of A_1 and A_2 are 3.5. Although the adjacent data points at the cut point 3.5 of A_1 (3:7, 4:1) are the same target class, they highly different in natural groups, which the upper left 5 data points are the same natural group and the lower 5 data points is the other. The graph-based algorithms could preserve the natural groups of data points by selected this cut point. In addition, the proposed algorithm do not partition a right data point (12:5) to be alone same as FUSINTER, MChi2, and HDD. It is clearly shown that the graph clustering-based discretization algorithms give a finite number of cut points, prevent an unnatural bias for undesiredly partitioning out some data points, and preserve the natural groups of data points. It is different from CAIM, Zeta, Ameva, and ur-CAIM that only considered the target class. Based on the purity class, CAIM, Zeta, Ameva, and ur-CAIM selected the cut point at 2.5 of A_1. The result damaged the upper left natural group.

PKID is an unsupervised method that does not consider the target class and the natural groups. The algorithm discretizes by giving all intervals similar numbers of data points. Considering A_1, there are 3 intervals and the number of data points in the intervals are 4, 4, and 2. For A_2, the number of data points in the divided intervals are 3, 4, and 3.

FUSINTER, MChi2, and HDD algorithms resulted in very large numbers of intervals, especially HDD algorithm that selected every single possible cut point.

Time complexity and parameter analysis

Time complexity

Besides previous quality assessments, the computational requirements of the graph clustering-based methods are discussed here. To estimate the time complexity, only the time taken to discretize a single attribute is considered. As the AF matrix is created once and share resources to discretize other attributes, the complexity time did not include the time of creating the matrix. In addition, for the ease of computing the denominator of *NAsso*, $\omega(C_i, C_i) + \omega(C_i, \bar{C}_i)$ (see Eq. 7), the sum of each row is calculated and stored in *sumRowsVector*.

For GraphS algorithm (see Fig. 7 for details), the primary time complexity of the algorithm is spent at line 9 to line 15, which is the step of finding the best cut point. The algorithm selected the best cut point from all possible cut points (c) by computing *NAsso*. As the denominator of *NAsso* could easily be calculated by summing all of the rows in the *sumRowsVector*, the primary time is only spent on calculating the numerator, $\omega(C_i, C_i)$ (inter-cluster). In Fig. 20 balance partition and balance the number of cut points are assumed; therefore at each parallel step of finding the best cut points, only half of the original or previous matrices are taken into account, the time complexity is diminished. Hence, the time complexity is formulated as Eq.16.

$$T(n) = 2T\left(\frac{cn^2}{8}\right) + \frac{n^2}{2} \times c \tag{16}$$

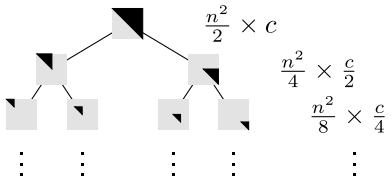

Fig. 20 The time complexity of GraphS. Note that, the *black triangles* denote time complicity of *NAsso*

where, $n^2/2$ is the time complexity of *NAsso* and c is the number of possible cut points. In this case, it is straightforward to sum across each row of the recursion tree to obtain the total time complexity. However, the depth of the tree does not really matter because the amount of time at each level drastically decrease as it expands deeper such that the total time is only a constant factor more than the time spent at the root. Therefore, the time complexity of GraphS is $\mathcal{O}(cn^2)$. However, if each data value of an attribute is the distinction, the number of possible cut points is $n-1$. Hence, the time complexity is $\mathcal{O}(n^3)$.

For GraphM algorithm (see Fig. 9 for details), the time complexity of *NAsso* is similar to GraphS but GraphM does not take any time to find the best cut point. From line 7 to line 9, GraphM computes *NAsso* of a small matrices many times; however, the process is very fast. After that, from line 10 to line 17 the algorithm iteratively merges until all intervals merge into one. Each merging computes *NAsso* twice as that of the adjacent interval; therefore, the time complexity of GraphM is $\mathcal{O}(2n^2) = \mathcal{O}(n^2)$.

Parameter analysis

The proposed algorithms include one parameter that is the β, the significant better rate, which is used in the stopping criterion. This study sets $\beta = 1.01$. A good β value will resulting finite number of intervals that leads to achieve high classification performance. The parameter analysis aims to provide a practical means that users can make the best use of the graph clustering-based framework. To this extent, Fig. 21 illustrates such a relationship, based on the predictive accuracy of standard datasets, the number of intervals of standard dataset, AUC of imbalanced datasets, and number of intervals of imbalanced datasets across all classifiers. In the figure, the value of β is varied from 1.00 through 1.05, in steps of 0.005.

An important observation of average accuracy of the standard dataset is that the proposed algorithms performed well with the β value are between 1.005 and 1.015. The accuracy and performance dropped if β is greater than 1.015 as the stopping criterion are quickly met and hence the process stops too early such that only two intervals are obtained. In contrast, if β is not considered ($\beta = 1$), the algorithm will be over partitioning and result in too many intervals. Therefore, a good β value that gives a decent number of intervals are in the rank 1.005–1.015. Similarly, the average AUC results of imbalanced datasets perform well with the β are in the rank 1.000–1.015, especially $\beta = 1.005$.

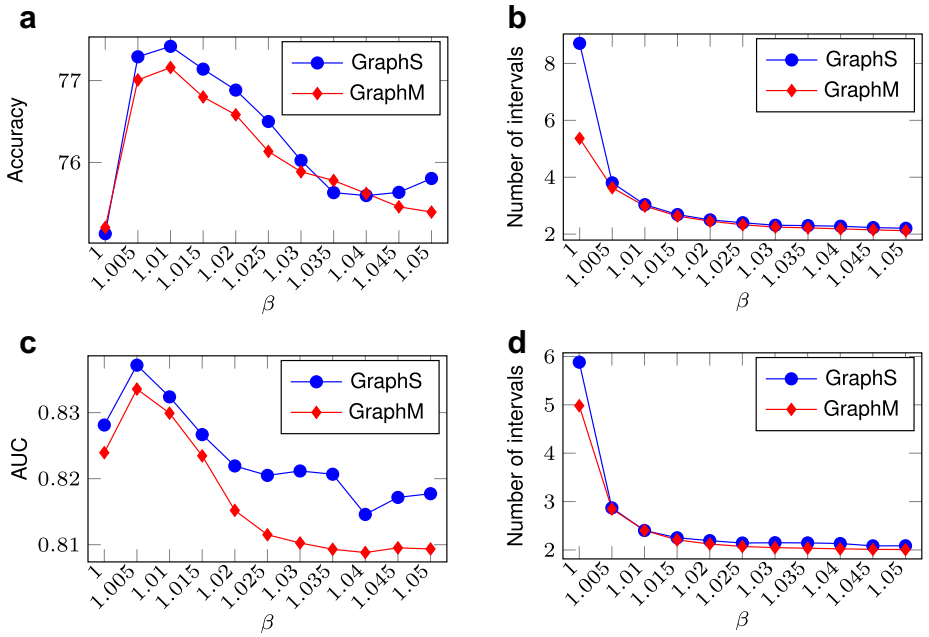

Fig. 21 β parameter. The effect of β values on the discretization results of: **a** average predictive accuracies of standard datasets, **b** average number of intervals of standard dataset, **c** average AUC of imbalanced datasets, and **d** average number of intervals of imbalanced dataset

Conclusion

This paper presents two novel, highly effective graph clustering-based discretization algorithms that are graph clustering-based discretization of splitting method (GraphS) and merging method (GraphM). They aim to discretize by considering the natural groups and prevent partitioning any interval that will possess a small number of data points. The algorithms view the data points as a graph, where the vertices are the data points (instances) and the weighted edges are the similarity between the data points. The *NAsso* measure is used as the discretization criterion of the algorithms. The empirical study, with different discretization algorithms, classifiers, two types of dataset of standard datasets and imbalanced datasets, suggests that the proposed graph clustering-based methods usually achieve superior discretization results compared to those of the previous well-known discretization algorithms. The prominent future work may include an extensive study regarding the scoring of the weighted similarity edges by considering other distance measures. In addition, this methodology will also be applied to specific domains such as biomedical datasets; where discretization is not only required for accurate prediction, but also an interpretable learning model.

Additional files

Additional file 1. GraphS and GraphM programs. The programs are implemented in Java, which are the Java class file.
Additional file 2. The performance results of 30 standard datasets. The results of predictive accuracy, number of intervals, number of remove attributes, and running times of 13 discretization algorithms of 30 standard datasets (tenfold) and 4 classifiers.
Additional file 3. The performance results of 20 imbalanced datasets. The results of AUC, number of intervals, number of remove attributes, and running times of 13 discretization algorithms of 20 imbalanced datasets (fivefold) and 4 classifiers.
Additional file 4. Datasets. All datasets of: 30 standard datasets and 20 imbalanced datasets.

Authors' contributions
KS drafts this manuscript, developed the algorithms and conducted experiments using the datasets and analysed the results. TB provided guidelines and helped draft the manuscript. NI being supervisor this research, suggested the methods and helped draft the manuscript. All authors corrected the manuscript. All authors read and approved the final manuscript.

Authors' information
KS is a lecturer at the School of Computer and Information Technology, Chiang Rai Rajabhat University (CRRU), Thailand, since 2009. He received the BEng degree in computer engineering (first class honors) from Naresuan University (NU) in 2008 and MEng degree in computer engineering from Kasetsart University (KU) in 2012. Currently, he is a Ph.D. candidate in computer engineering at Mae Fah Luang University (MFU). His primary research interests are in the area of machine learning, data mining, data preprocessing, and data reduction. TB is a Lecturer at the School of Information Technology, Mae Fah Luang University, Chiang Rai, Thailand. He obtained Ph.D. in Artificial Intelligence from Cranfield University in 2003, and worked as Post-Doctoral Research Associate (PDRA) at Aberystwyth University, during 2007-2010. His PDRA work focused on anti-terrorism using data analytical and decision support synthesizes. He has been the leader of research projects in exploiting biometrics technology for anti-terrorism in southern-provinces of Thailand, funded by Ministry of Defense. He also serves as a committee and reviewer of several venues, IEEE SMC, IEEE TKDE, Knowledge Based Systems, International Journal of Intelligent Systems Technologies and Applications, for instance. NI is an Assistant Professor at the School of Information Technology, Mae Fah Luang University. She received Ph.D. in Computer Science from Aberystwyth University in 2010, funded by Royal Thai Government. Her Ph.D. work won the Thesis Prize of 2012 by Thai National Research Council. Her present research of improving face classification for anti-terrorism and crime protection has been funded by Ministry of Science and Technology. She serves as an editor for International Journal of Data Analysis Techniques and Strategies; as a committee and reviewer of several venues, IEEE SMC, IEEE TKDE, Machine Learning, for instance.

Acknowledgements
The authors would like to thank KEEL software [59, 60] for distributing the source code of discretization algorithms, and the authors of EMD [31] for EMD program, and the authors of ur-CAIM [16] for distributing the ur-CAIM program.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
All datasets in this research, including 30 standard datasets and 20 imbalanced datasets can be found at website http://archive.ics.uci.edu/ml and http://sci2s.ugr.es/keel/category.php?cat=imb, respectively. In addition, these datasets are included in Additional file 4.

Publisher's Note

Received: 13 December 2016 Accepted: 30 May 2017
Published online: 03 August 2017

References
1. Han J, Kamber M, Pei J (2011) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco
2. Sriwanna K, Puntumapon K, Waiyamai K (2012) An enhanced class-attribute interdependence maximization discretization algorithm. Springer, Berlin
3. Yang P, Li J-S, Huang Y-X (2011) Hdd: a hypercube division-based algorithm for discretisation. Int J Syst Sci 42(4):557–566
4. Bay SD (2001) Multivariate discretization for set mining. Knowl Inf Syst 3(4):491–512
5. de Sá CR, Soares C, Knobbe A (2016) Entropy-based discretization methods for ranking data. Information Sciences 329:921–936 **(special issue on Discovery Science)**
6. Ramírez-Gallego S, García S, Mouriño-Talín H, Martínez-Rego D, Bolón-Canedo V, Alonso-Betanzos A, Benítez JM, Herrera F (2016) Data discretization: taxonomy and big data challenge. Wiley Interdiscip Rev 6(1):5–21
7. Garcia S, Luengo J, Sáez JA, López V, Herrera F (2013) A survey of discretization techniques: taxonomy and empirical analysis in supervised learning. IEEE Trans Knowl Data Eng 25(4):734–750
8. Sang Y, Li K (2012) Combining univariate and multivariate bottom-up discretization. Multiple-Valued Logic and Soft Computing 20(1–2):161–187
9. Liu H, Hussain F, Tan CL, Dash M (2002) Discretization: an enabling technique. Data Min Knowl Discov 6(4):393–423
10. Dougherty J, Kohavi R, Sahami M et al (1995) Supervised and unsupervised discretization of continuous features. In: Machine learning: proceedings of the Twelfth international conference, vol 12, pp 194–202
11. Kerber R (1992) Chimerge: discretization of numeric attributes. In: Proceedings of the tenth national conference on artificial intelligence. Aaai Press, San Jose, pp 123–128
12. Liu H, Setiono R (1997) Feature selection via discretization. IEEE Trans Knowl Data Eng 9(4):642–645
13. Tay FE, Shen L (2002) A modified chi2 algorithm for discretization. IEEE Trans Knowl Data Eng 14(3):666–670

14. Sang Y, Qi H, Li K, Jin Y, Yan D, Gao S (2014) An effective discretization method for disposing high-dimensional data. Inf Sci 270:73–91
15. Kurgan LA, Cios KJ (2004) Caim discretization algorithm. IEEE Trans Knowl Data Eng 16(2):145–153
16. Cano A, Nguyen DT, Ventura S, Cios KJ (2016) ur-caim: improved caim discretization for unbalanced and balanced data. Soft Computing 20(1):173–188
17. Ching JY, Wong AK, Chan KCC (1995) Class-dependent discretization for inductive learning from continuous and mixed-mode data. IEEE Trans Pattern Anal Mach Intell 17(7):641–651
18. Fayyad UM, Irani KB (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings of the 13th international joint conference on artificial intelligence. Chambéry, France, 28 Aug–3 Sept 1993, pp 1022–1029
19. Catlett J (1991) On changing continuous attributes into ordered discrete attributes. In: Kodratoff Y. (eds) Machine Learning — EWSL-91. EWSL 1991. Lecture notes in computer science (Lecture notes in artificial intelligence), vol 482. Springer, Berlin
20. Zeinalkhani M, Eftekhari M (2014) Fuzzy partitioning of continuous attributes through discretization methods to construct fuzzy decision tree classifiers. Inf Sci 278:715–735
21. Yang Y, Webb GI (2009) Discretization for naive-bayes learning: managing discretization bias and variance. Mach Learn 74(1):39–74
22. Kang Y, Wang S, Liu X, Lai H, Wang H, Miao B (2006) An ICA-based multivariate discretization algorithm. Springer, Berlin
23. Gupta A, Mehrotra KG, Mohan C (2010) A clustering-based discretization for supervised learning. Stat Probab Lett 80(9):816–824
24. Singh GK, Minz S (2007) Discretization using clustering and rough set theory. In: International conference on computing: theory and applications, 2007. ICCTA'07. IEEE, New York, pp 330–336
25. Hartigan JA, Wong MA (1979) Algorithm as 136: a k-means clustering algorithm. Appl Stat 28:100–108
26. Ertoz L, Steinbach M, Kumar V (2002) A new shared nearest neighbor clustering algorithm and its applications. In: Workshop on clustering high dimensional data and its applications at 2nd SIAM international conference on data mining, pp 105–115
27. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd 96:226–231
28. Sriwanna K, Boongoen T, Iam-On N (2016) In: Lavangnananda K, Phon-Amnuaisuk S, Engchuan W, Chan JH (eds) An enhanced univariate discretization based on cluster ensembles. Springer, Cham, pp 85–98
29. Iam-On N, Boongoen T, Garrett S, Price C (2011) A link-based approach to the cluster ensemble problem. IEEE Trans Pattern Anal Mach Intell 33(12):2396–2409
30. Huang X, Zheng X, Yuan W, Wang F, Zhu S (2011) Enhanced clustering of biomedical documents using ensemble non-negative matrix factorization. Inf Sci 181(11):2293–2302
31. Ramirez-Gallego S, Garcia S, Benitez JM, Herrera F (2016) Multivariate discretization based on evolutionary cut points selection for classification. IEEE Transactions on Cybernetics 46(3):595–608
32. Parashar A, Gulati Y (2012) Survey of di erent partition clustering algorithms and their comparative studies. International Journal of Advanced Research in Computer Science 3(3):675–680
33. Brandes U, Gaertler M, Wagner D (2007) Engineering graph clustering: models and experimental evaluation. ACM J Exp Algorithm 12(1.1):1–26
34. Van Dongen SM (2001) Graph clustering by ow simulation. PhD thesis, University of Utrecht
35. Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1(1):27–64
36. Foggia P, Percannella G, Sansone C, Vento M (2009) Benchmarking graph-based clustering algorithms. Image Vis Comput 27(7):979–988
37. Zhou Y, Cheng H, Yu JX (2009) Graph clustering based on structural/attribute similarities. Proc VLDB Endow 2(1):718–729
38. Cheng H, Zhou Y, Yu JX (2011) Clustering large attributed graphs: a balance between structural and attribute similarities. ACM Trans Knowl Discov Data 5(2):12
39. Nascimento MC, De Carvalho AC (2011) Spectral methods for graph clustering-a survey. Eur J Oper Res 211(2):221–231
40. Shi J, Malik J (2000) Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell 22(8):888–905
41. Foggia P, Percannella G, Sansone C, Vento M (2007) In: Escolano F, Vento M (eds) Assessing the performance of a graph-based clustering algorithm. Springer, Berlin, pp 215–227
42. Enright AJ, Van Dongen S, Ouzounis CA (2002) An efficient algorithm for large-scale detection of protein families. Nucleic Acids Res 30(7):1575–1584
43. Kannan R, Vempala S, Vetta A (2004) On clusterings: good, bad and spectral. J ACM 51(3):497–515
44. Brandes U, Gaertler M, Wagner D (2003) Experiments on graph clustering algorithms. Springer, Berlin, pp 568–579
45. Kong W, Hu S, Zhang J, Dai G (2013) Robust and smart spectral clustering from normalized cut. Neural Comput Appl 23(5):1503–1512
46. Sen D, Gupta N, Pal SK (2013) Incorporating local image structure in normalized cut based graph partitioning for grouping of pixels. Inf Sci 248:214–238
47. Cha S-H (2007) Comprehensive survey on distance/similarity measures between probability density functions. City 1(2):1
48. Everitt B, Landau S, Leese M (1993) Cluster analysis (Edward Arnold, London). ISBN 0-470-22043-0
49. Soman KP, Diwakar S, Ajay V (2006) Data mining: theory and practice [with CD]. PHI Learn
50. Chapanond A (2007) Application aspects of data mining analysis on evolving graphs. PhD thesis, Troy
51. Boutin F, Hascoet M (2004) Cluster validity indices for graph partitioning. In: Proceedings, eighth international conference on information visualisation, 2004. IV 2004. IEEE, New York, pp 376–381
52. Dua S, Chowriappa P (2012) Data mining for bioinformatics. CRC Press, Boca Raton
53. Görke R, Kappes A, Wagner D (2014) Experiments on density-constrained graph clustering. J Exp Algorithmics 19:6

54. Leighton T, Rao S (1988) An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: 29th annual symposium on foundations of computer science, 1988. IEEE, New York, pp 422–431
55. Ding CH, He X, Zha H, Gu M, Simon HD (2001) A min-max cut algorithm for graph partitioning and data clustering. In: Proceedings IEEE international conference on data mining, 2001, ICDM 2001. IEEE, New York, pp 107–114
56. Mohar B, Alavi Y (1991) The laplacian spectrum of graphs. Graph Theory Comb Appl 2:871–898
57. Von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416
58. Lichman M (2013) UCI machine learning repository
59. Alcalá J, Fernández A, Luengo J, Derrac J, García S, Sánchez L, Herrera F (2010) Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. J Mult Valued Log Soft Comput 17(255–287):11
60. Alcalá-Fdez J, Sánchez L, García S, del Jesus MJ, Ventura S, Garrell JM, Otero J, Romero C, Bacardit J, Rivas VM, Fernández JC, Herrera F (2009) Keel: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput 13(3):307–318
61. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco
62. Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6(1):37–66
63. John GH, Langley P (1995) Estimating continuous distributions in Bayesian classifiers. Proceedings of the eleventh conference on uncertainty in artificial intelligence. UAI'95. Morgan Kaufmann Publishers Inc., San Francisco, pp 338–345
64. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
65. Wu X, Kumar V (2009) The top ten algorithms in data mining, 1st edn. Chapman & Hall/CRC, Boca Raton
66. Wu X, Kumar V, Quinlan JR, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Philip SY et al (2008) Top 10 algorithms in data mining. Knowl Inf Syst 14(1):1–37
67. Kohavi R et al (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. Ijcai 14:1137–1145
68. Bradley AP (1997) The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recognit 30(7):1145–1159
69. Huang J, Ling CX (2005) Using auc and accuracy in evaluating learning algorithms. IEEE Trans Knowl Data Eng 17(3):299–310
70. Ruan J, Jahid MJ, Gu F, Lei C, Huang YW, Hsu YT, Mutch DG, Chen CL, Kirma NB, Huang THM (2016) A novel algorithm for network-based prediction of cancer recurrence. Genomics. doi:10.1016/j.ygeno.2016.07.005
71. Lv J, Peng Q, Chen X, Sun Z (2016) A multi-objective heuristic algorithm for gene expression microarray data classification. Expert Syst Appl 59:13–19
72. Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc 32(200):675–701
73. Friedman M (1940) A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat 11(1):86–92
74. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
75. García S, Fernández A, Luengo J, Herrera F (2010) 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
76. Holm S (1979) A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics 6:65–70
77. Gonzalez-Abril L, Cuberos FJ, Velasco F, Ortega JA (2009) Ameva: an autonomous discretization algorithm. Expert Syst Appl 36(3):5327–5332
78. Tsai C-J, Lee C-I, Yang W-P (2008) A discretization algorithm based on class-attribute contingency coefficient. Inf Sci 178(3):714–731
79. Eshelman LJ (1991) The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. Found Genet Algorithms 1:265–283
80. Zighed DA, Rabaséda S, Rakotomalala R (1998) Fusinter: a method for discretization of continuous attributes. Int J Uncertain Fuzziness Knowl Based Syst 6(03):307–326
81. Wong AKC, Liu TS (1975) Typicality, diversity, and feature pattern of an ensemble. IEEE Trans Comput 100(2):158–181
82. Huang W (1997) Discretization of continuous attributes for inductive machine learning. Toledo, Department Computer Science, University of Toledo
83. Ho KM, Scott PD (1997) Zeta: a global method for discretization of continuous variables. In: Proceedings of the third international conference knowledge discovery and data mining (KDD97), pp 191–194
84. Healey J (2014) Statistics: a tool for social research. Cengage Learn